

REDES DE COMPUTADORAS

REDES DE COMPUTADORAS

Tecnicatura en informática



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright ©year by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department with the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data:

Title, etc
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

PART I

1. INTRODUCCIÓN Y TECNOLOGÍAS SUBYACENTES

- **Capítulo 1 Introducción**
- **Capítulo 2 El Modelo OSI y el Protocolo TCP/IP**
- **Capítulo 3 Tecnologías Subyacentes**

CHAPTER 1

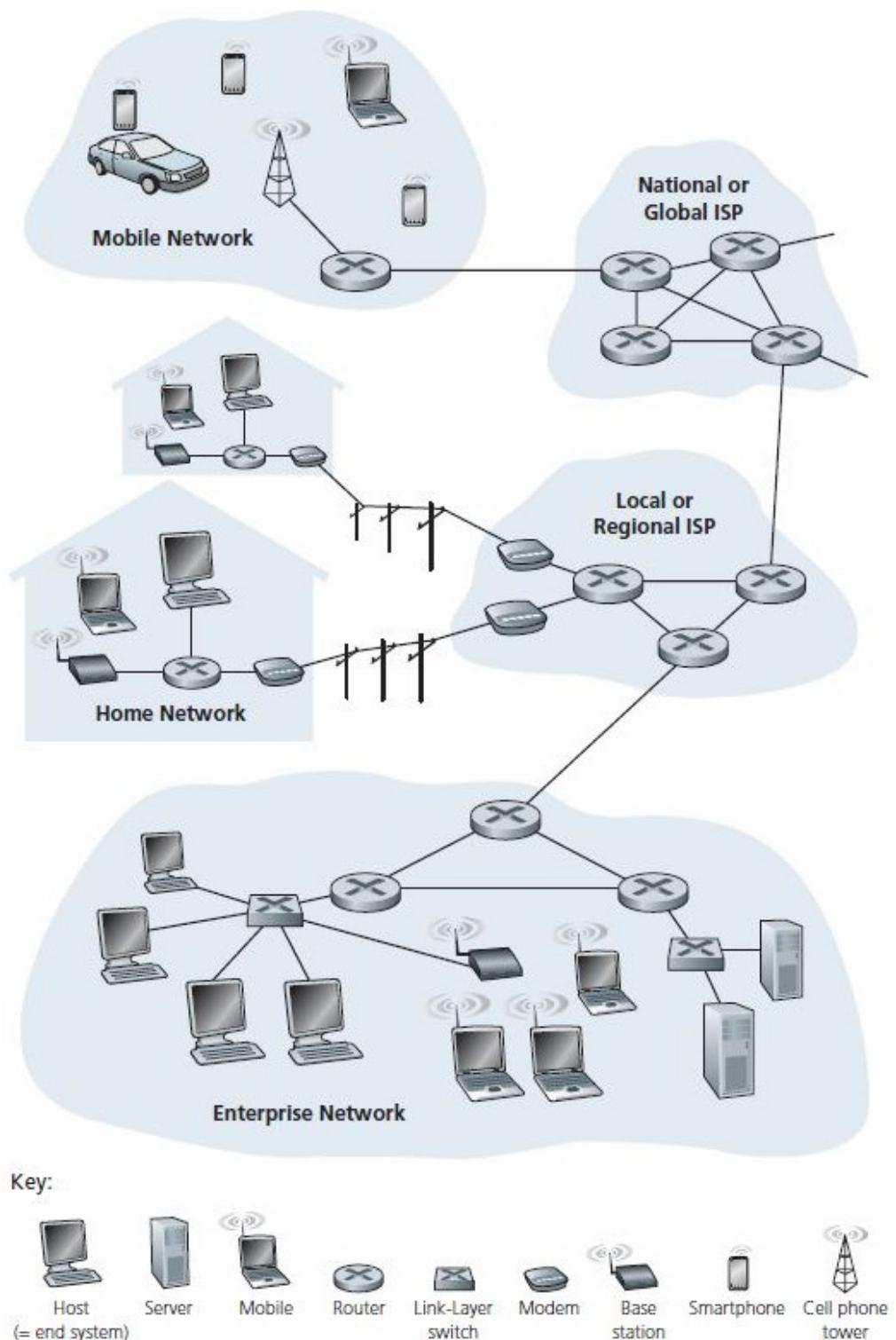
INTRODUCCION

La Internet de hoy es sin duda el más grande sistema de ingeniería jamás creado por el hombre, con cientos de millones de computadoras conectadas, enlaces de comunicación e interruptores; con miles de millones de usuarios que se conectan a través de computadoras portátiles, tablets y teléfonos inteligentes; y con una serie de nuevos dispositivos conectados a Internet, como sensores, cámaras Web, consolas de juegos, marcos de cuadros e incluso máquinas lavadoras. Dado que Internet es tan grande y tiene tantos componentes y usos diversos, ¿hay alguna esperanza de entender cómo funciona? ¿Existen principios y estructuras de orientación que pueden proporcionar una base para la comprensión de un sistema tan increíblemente grande y complejo? Y si es así, es posible que en realidad pueda ser interesante y divertido aprender sobre las redes de computadoras? Afortunadamente, las respuestas a todas estas preguntas es un rotundo sí! De hecho, es nuestro objetivo en este libro ofrecerle una introducción moderna al campo dinámico de las redes informáticas, que le da los principios y conocimientos prácticos que necesita para comprender no sólo las redes de hoy en día, sino a las del mañana también.

Este primer capítulo presenta una visión general de las redes de computadoras y de Internet. Nuestro objetivo aquí es dibujar un cuadro amplio y establecer el contexto para el resto del curso. Vamos a cubrir mucho terreno en este capítulo introductorio y discutimos muchas de las piezas de una red de computadoras, sin perder de vista el panorama general.

1.1 ¿Qué es la Internet?

En este libro, vamos a utilizar la Internet pública, una red informática específica, como nuestro principal vehículo para la discusión de las redes de computadoras y sus protocolos. Pero qué *es* Internet? Hay un par de maneras de responder a esta pregunta. En primer lugar, podemos describir los aspectos básicos de Internet, es decir, los componentes básicos de hardware y software que componen Internet. En segundo lugar, podemos describir el Internet en términos de una infraestructura de red que proporciona servicios a las aplicaciones distribuidas. Vamos a comenzar con una descripción de *tuercas-y-pernos*, usando la Figura 1.1 para ilustrar nuestra discusión.

**Figura 1.1 Algunas partes de Internet**

1.1.1 Una Descripción de tuercas y pernos

La Internet es una red informática que interconecta cientos de millones de dispositivos informáticos de todo el mundo. No hace mucho tiempo, estos dispositivos de computación eran principalmente las PC tradicionales de escritorio, estaciones de trabajo Linux, y los llamados servidores que almacenan y transmiten información como páginas web y mensajes de correo electrónico. Sin embargo, se están conectando cada vez más sistemas finales no

tradicionales, tales como computadoras portátiles, teléfonos inteligentes, tablets, televisores, consolas de juegos, cámaras web, automóviles, dispositivos sensores ambientales, marcos de cuadros, y sistemas de seguridad para Internet. De hecho, el término *red de computadoras* empieza a sonar un poco anticuado, dados los muchos dispositivos no tradicionales que están siendo conectados a Internet. En la jerga de Internet, todos estos dispositivos se denominan **hosts** o **sistemas finales**. A partir de julio de 2011, había cerca de 850 millones de sistemas finales conectados a Internet [ISC 2012], sin contar teléfonos inteligentes, computadoras portátiles y otros dispositivos que están conectados de forma intermitente a Internet. En total, más hay un estimado de 2 millones de usuarios de Internet [UIT-2011].

Los sistemas finales están conectados entre sí por una red de **enlaces de comunicación** y **comutadores de paquetes**. Ya veremos en la sección 1.2 que hay muchos tipos de enlaces de comunicación, los cuales se componen de diferentes tipos de medios físicos, incluyendo cable coaxial, cable de cobre, fibra óptica, y el espectro de radio. Los diferentes enlaces pueden transmitir datos a velocidades diferentes, con la **velocidad de transmisión** de un enlace medido en bits/segundo. Cuando un sistema final tiene datos para enviar a otro sistema final, el sistema final emisor segmenta los datos a enviar y agrega bytes de cabecera para cada segmento. Los paquetes resultantes de información, conocidos como **paquetes** en la jerga de las redes de computadoras, se envían a través de la red al sistema final de destino, en el que se vuelven a montar en los datos originales.

Un comutador de paquetes toma un paquete que llega en uno de sus enlaces de comunicaciones entrantes y reenvía ese paquete a uno de sus enlaces de comunicación saliente. Los comutadores de paquetes vienen en muchas formas y sabores, pero los dos tipos más prominentes de Internet de hoy son los **routers** y **switches de capa de enlace**. Ambos tipos de comutadores reenvian paquetes hacia sus destinos finales. Los switches de capa de enlace se utilizan normalmente en las redes de acceso, mientras que los routers se utilizan normalmente en el núcleo de la red. La secuencia de los enlaces de comunicación y comutadores de paquetes atravesado por un paquete desde el sistema de extremo emisor al sistema de extremo receptor se conoce como una **ruta (route)** o **camino (path)** a través de la red. La cantidad exacta de tráfico que ha experimentado Internet es difícil de estimar, pero Cisco [Cisco VNI 2011] estima que el tráfico global de Internet era de casi 40 exabytes por mes en 2012.

Las redes de conmutación de paquetes (que transportan paquetes) son en muchos aspectos similares a las redes de transporte de autopistas, rutas e intersecciones (que transportan vehículos). Consideremos, por ejemplo, una fábrica que tiene que mover una gran cantidad de carga a algunos depósitos de destino, situados a miles de kilómetros de distancia. En la fábrica, la carga está segmentada y se carga en una flota de camiones. Cada uno de los camiones luego viaja de forma independiente a través de la red de autopistas, rutas, intersecciones al almacén de destino. En el almacén de destino, la carga se descarga y se agrupa con el resto de la carga que llega con el mismo envío. Por lo tanto, en muchos aspectos, los paquetes son análogos a los camiones, los enlaces de comunicación son análogos a caminos y rutas, los comutadores de paquetes son análogas a las intersecciones, y los sistemas finales son análogos a los edificios. Del mismo modo que un camión toma un camino a través de la red transporte, un paquete toma un camino a través de una red informática.

Los sistemas finales acceden a Internet a través de **proveedores de servicios de Internet (ISP)**, incluyendo ISP residenciales, tales como compañías de cable o de telefonía local, ISP corporativos, ISP universitarios, y proveedores de Internet que proporcionan acceso WiFi en aeropuertos, hoteles, cafeterías y otros lugares públicos. Cada ISP es en sí mismo una red de comutadores de paquetes y canales de comunicación. ISP ofrecen una variedad de tipos de acceso a la red a los sistemas finales, incluyendo el acceso de banda ancha residencial como un módem de cable o DSL, de alta velocidad de acceso a la red de área local, conexión inalámbrica, y acceso telefónico por módem de 56 kbps. Los ISP también ofrecen acceso a Internet a los proveedores de contenidos, conectando sitios Web directamente a Internet. Internet es todo acerca de la conexión de los sistemas de extremo entre ellos, por lo que los proveedores de Internet que proporcionan acceso a los sistemas de extremo también debe estar interconectados.

Estos ISP de nivel inferior están interconectados a través de los ISP nacionales e internacionales de nivel superior, tales como Level 3 Communications, AT&T, Sprint y NTT. Un ISP de nivel superior se compone de los routers de alta velocidad interconectados con enlaces de fibra óptica de alta velocidad. Cada red ISP, ya sea de nivel superior o de nivel inferior, se gestiona de forma independiente, se ejecuta el protocolo IP (véase más adelante), y se ajusta a ciertas convenciones de nombres y direcciones. Vamos a examinar los ISP y su interconexión más estrechamente en la Sección 1.3.

Los sistemas finales, los conmutadores de paquetes, y otras piezas de Internet ejecutan **protocolos** que controlan el envío y recepción de información en Internet. El **Protocolo de Control de Transmisión (TCP)** y el **Protocolo de Internet (IP)** son dos de los protocolos más importantes de Internet. El protocolo IP especifica el formato de los paquetes que se envían y se reciben entre los routers y sistemas finales. Los protocolos principales de Internet se conocen colectivamente como **TCP/IP**.

Dada la importancia de los protocolos de Internet, es importante que todo el mundo esté de acuerdo en lo que hacen todos y cada uno de los protocolos, para que las personas puedan crear sistemas y productos que interactúen. Aquí es donde entran en juego las normas. Los **estándares de Internet** son desarrollados por el Grupo de Trabajo de Ingeniería de Internet (IETF) [IETF 2012]. Los documentos de normas IETF se llaman las **solicitudes de comentarios (RFC, Requests for comments)**. RFC comenzaron como solicitudes generales de comentarios (de ahí el nombre) para resolver problemas de diseño de protocolo de red que enfrentó el precursor de Internet [Allman 2011]. Las RFC tienden a ser bastante técnicas y detalladas. Definen protocolos tales como TCP, IP, HTTP (para la web) y SMTP (por correo electrónico). Actualmente hay más de 6.000 RFC. Otros órganos también especifican normas para componentes de red, sobre todo para los enlaces de red. El Comité IEEE de Estándares 802 LAN / MAN [IEEE 802 de 2012], por ejemplo, especifica el Ethernet y WiFi estándares inalámbricos.

1.1.2 Una descripción de servicios

Nuestra discusión anterior ha identificado muchas de las piezas que componen Internet. Pero también podemos describir la Internet desde un ángulo completamente diferente, a saber, como *una infraestructura que proporciona servicios a las aplicaciones*. Estas aplicaciones incluyen correo electrónico, navegación por Internet, redes sociales, mensajería instantánea, VoiceoverIP (VoIP), la transmisión de vídeo, juegos distribuidos, peer-to-peer (P2P), la televisión a través de Internet, acceso remoto, y mucho, mucho más . Las aplicaciones se dice que son **aplicaciones distribuidas**, ya que implican múltiples sistemas finales que intercambian datos entre sí. Es importante destacar que las aplicaciones de Internet se ejecutan en los sistemas finales -no se ejecutan en los conmutadores de paquetes en red central. A pesar de que los conmutadores de paquetes facilitan el intercambio de datos entre sistemas finales, no tienen que ver con la aplicación que es la fuente o sumidero de datos.

Vamos a explorar un poco más de lo que entendemos por una infraestructura que proporciona servicios a las aplicaciones. Con este fin, supongamos que tienes una nueva idea interesante para una aplicación distribuida de Internet, una de la que puede beneficiarse en gran medida la humanidad o una que simplemente puede hacerte rico y famoso. ¿Cómo podés transformar esta idea en una aplicación real de Internet? Dado que las aplicaciones se ejecutan en sistemas finales, vas a tener que escribir programas que se ejecutan en los sistemas finales. Es posible, por ejemplo, escribir tus programas en Java, C o Python. Ahora, debido a que está desarrollando una aplicación distribuida de Internet, los programas que se ejecutan en los diferentes sistemas finales tendrán que enviar los datos entre sí. Y aquí llegamos a un tema central que conduce a la forma alternativa de describir Internet como una plataforma para aplicaciones. ¿Cómo hace un programa que se ejecuta en un sistema final instruir a Internet para entregar datos a otro programa que se ejecuta en otro sistema final?

Los sistemas finales conectados a Internet proporcionan una **interfaz de programación de aplicaciones (API)** que especifica cómo un programa que se ejecuta en un sistema final solicita la infraestructura de Internet para entregar datos a un programa de destino específico que se ejecuta en otro sistema final. Esta API de Internet es un conjunto de reglas que el programa emisor deberá seguir para que Internet pueda entregar los datos al programa de destino. Discutiremos la API de Internet en detalle en el capítulo 2. Por ahora, vamos a recurrir a una simple analogía, una que vamos a utilizar con frecuencia en estos apuntes. Supongamos que Alice quiere enviar una carta a Bob utilizando el servicio postal. Alice, por supuesto, no puede limitarse a escribir la carta (los datos) y dejar caer la carta por la ventana. En su lugar, el servicio postal requiere que Alice ponga la carta en un sobre; escriba el nombre completo, dirección de Bob, y código postal en el centro del sobre; sellar el sobre; poner un sello en la esquina superior derecha del mismo; y, finalmente, dejar caer el sobre en un buzón oficial del servicio postal. Por lo tanto, el servicio postal tiene su propia "API del servicio postal," o conjunto de reglas, que Alice debe seguir para que el servicio postal entregue su carta a Bob. De una manera similar, la Internet tiene una API que el programa emisor de datos debe seguir para que Internet entregue los datos al programa que va a recibir los datos.

El servicio postal, por supuesto, proporciona más de un servicio a sus clientes. Proporciona entrega urgente, confirmación de la recepción, el uso ordinario, y muchos más servicios. De una manera similar, el Internet ofrece múltiples servicios a sus aplicaciones. Cuando se desarrolla una aplicación de Internet, también deberás elegir uno

de los servicios de Internet para tu aplicación. Vamos a describir los servicios de Internet en el Capítulo 2.

Acabamos de dar dos descripciones de Internet; una en términos de sus componentes de hardware y software, y el otro en términos de una infraestructura para la prestación de servicios de aplicaciones distribuidas. Pero tal vez todavía estés confundido en cuanto a lo que es Internet. ¿Qué son la conmutación de paquetes y TCP/IP? ¿Qué son los routers? ¿Qué tipos de enlaces de comunicación están presentes en Internet? ¿Qué es una aplicación distribuida? ¿Cómo puede una tostadora o un sensor meteorológico conectarse a Internet? Si te sientes un poco abrumado por todo esto ahora, no te preocunes, el propósito de curso es dar a conocer que tanto los aspectos básicos de Internet y de los principios que rigen el cómo y el por qué funciona. Vamos a explicar estos términos y preguntas importantes en las siguientes secciones y capítulos.

1.1.3 ¿Qué es un protocolo?

Ahora que tenemos una pequeña idea de lo que es Internet, vamos a considerar otra palabra de moda importante en las redes de computadoras: *protocolo*. ¿Qué es un protocolo? ¿Qué hace un protocolo?

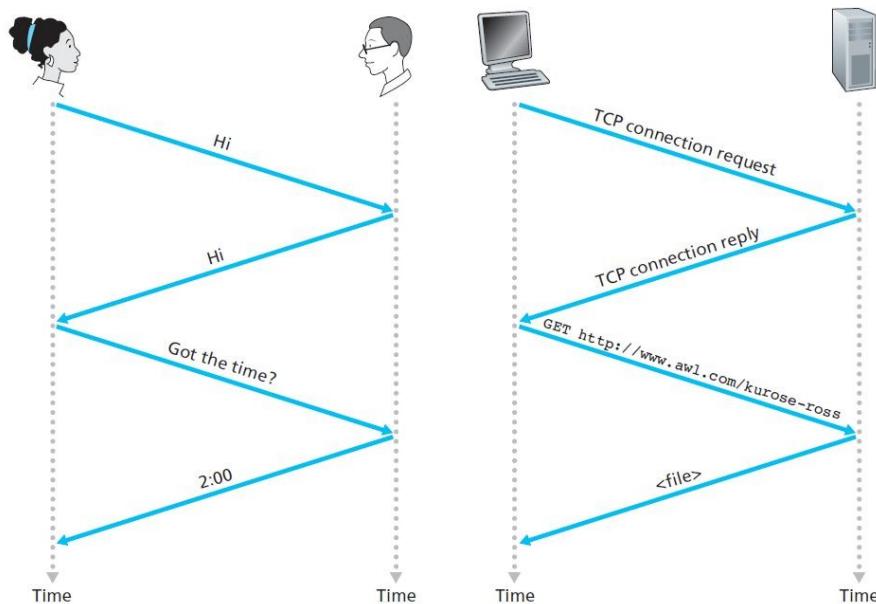


Figura 1.2 Un Protocolo Humano y un Protocolo de Redes de Computadoras

Una analogía humana

Probablemente es más fácil de entender la noción de un protocolo de red informática considerando en primer lugar algunas analogías humanas, ya que los humanos ejecutan protocolos todo el tiempo. Considere lo que se hace cuando se quiere pedir a alguien la hora del día. Un intercambio típico se muestra en la Figura 1.2. El protocolo humano (o las buenas costumbres, al menos) dicta que uno primero ofrece un saludo (el primer "Hola" en la Figura 1.2) para iniciar la comunicación con otra persona. La respuesta típica a un "Hola" es un mensaje devuelto "Hola". Implícitamente, entonces uno toma una respuesta cordial "Hola" como una indicación de que se puede proceder y pedir la hora del día. Una respuesta diferente al inicial "Hola" (tales como "No me molestes!" O "yo no hablo Inglés", o alguna respuesta no imprimible) podría indicar una falta de voluntad o incapacidad de comunicarse. En este caso, el protocolo humano sería no pedir la hora del día. A veces uno no obtiene respuesta a una pregunta, en cuyo caso uno se da por vencido pidiendo a esa persona la hora. Notemos que en nuestro protocolo humano, *hay mensajes específicos que enviamos, y acciones específicas que tomamos en respuesta a los mensajes de respuesta recibidos u otros eventos* (como por ejemplo, ninguna respuesta dentro de una cierta cantidad de tiempo determinado). Es evidente que los mensajes transmitidos y recibidos, y las acciones tomadas cuando estos mensajes son enviados o recibidos u otros hechos, juegan un papel central en un protocolo humano. Si la gente ejecuta protocolos diferentes (por ejemplo, si una persona tiene modales, pero el otro no, o si uno entiende el concepto de tiempo y el otro no) los protocolos no interoperan y ningún trabajo útil se puede lograr. Lo mismo

es cierto en redes -se necesitan dos (o más) entidades ejecutando el mismo protocolo con el fin de realizar una tarea.

Consideremos una segunda analogía humana. Supongamos que estás en una clase de la universidad (una clase de redes informáticas, por ejemplo!). El maestro está explicando acerca de los protocolos y estás confundido. El maestro se detiene a preguntar: "¿Hay alguna pregunta?" (Un mensaje que se transmite a, y recibido por, todos los estudiantes que no están durmiendo). Levantar la mano (transmitiendo un mensaje implícito al maestro). Su maestro le reconoce con una sonrisa, diciendo "Sí..." (Un mensaje transmitido que le anima a hacer su pregunta -a los maestros les *encanta* ser interrogados), y luego realiza su pregunta (es decir, transmite su mensaje a su maestro). Su maestro escucha su pregunta (recibe su pregunta) y responde (le transmite una respuesta). Una vez más, vemos que la transmisión y recepción de mensajes, y un conjunto de acciones convencionales tomadas cuando estos mensajes se envían y reciben, están en el corazón de este protocolo de preguntas y respuestas.

Protocolos de red

Un protocolo de red es similar a un protocolo humano, excepto que las entidades que intercambian mensajes y realizan acciones son componentes de hardware o software de algún dispositivo (por ejemplo, ordenador, teléfono inteligente, tablet, un router u otro dispositivo compatible con la red). Toda la actividad en Internet que consiste en dos o más entidades que se comunican a distancia se rige por un protocolo. Por ejemplo, los protocolos implementados en hardware en dos computadoras conectadas físicamente controlan el flujo de bits en el "cable" ("wire") entre las dos tarjetas de interfaz de red; los protocolos de control de congestión en sistemas finales controlan la velocidad a la que los paquetes se transmiten entre el emisor y el receptor; los protocolos en los routers determinan la ruta de un paquete desde el origen hasta el destino. Los protocolos se ejecutan todas partes en Internet, y en consecuencia, gran parte de este curso es sobre los protocolos de redes de computadoras.

Como un ejemplo de un protocolo de red informática con la que estés probablemente familiarizado, considera lo que sucede cuando se hace una petición a un servidor Web, es decir, cuando se escribe la URL de una página Web en el navegador Web. El escenario se ilustra en la mitad derecha de la figura 1.2. En primer lugar, el equipo enviará un mensaje de petición de conexión con el servidor Web y esperará una respuesta. El servidor Web finalmente recibirá su mensaje de solicitud de conexión y devolverá un mensaje de respuesta de conexión. Sabiendo que ahora está bien solicitar el documento Web, el ordenador envía entonces el nombre de la página Web que quiere obtener desde el servidor Web en un mensaje GET. Por último, el servidor Web devuelve la página Web (archivo) a la computadora.

Teniendo en cuenta los ejemplos humanos y de redes anteriores, el intercambio de mensajes y las acciones tomadas cuando estos mensajes se envían y reciben son los elementos claves que definen un protocolo:

Un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más entidades comunicantes, así como las medidas adoptadas en la transmisión y/o recepción de un mensaje u otro evento.

Internet y las redes informáticas en general, hacen un amplio uso de los protocolos. Diferentes protocolos se utilizan para llevar a cabo diferentes tareas de comunicación. A medida que leas este curso, aprenderás 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 la comprensión del qué, por qué y cómo de los protocolos de red.

1.2 El extremo de la red

En la sección anterior hemos presentado una descripción de alto nivel de los protocolos de Internet y de redes. Ahora vamos a profundizar un poco más en los componentes de una red informática (y en Internet, en particular). Comenzamos en esta sección en los márgenes de una red y echamos un vistazo a los componentes con los que estamos más familiarizados, a saber, las computadoras, los teléfonos inteligentes y otros dispositivos que utilizamos a diario. En la siguiente sección vamos a pasar desde el borde de la red hasta el núcleo de la misma y examinaremos la comutación y encaminamiento en redes de computadoras.

Recordemos de la sección anterior que en la jerga de las redes de computadoras, las computadoras y otros dispositivos conectados a Internet son llamados a menudo como sistemas finales. Se los conoce como sistemas finales, ya que se ubican en los límites de Internet, como se muestra en la Figura 1.3. Los sistemas finales de Internet incluyen

las computadoras de escritorio (por ejemplo, computadoras de sobremesa, Mac y boxes de Linux), servidores (por ejemplo, Web y servidores de correo electrónico) y equipos móviles (por ejemplo, computadoras portátiles, teléfonos inteligentes y tablets). Además, un número cada vez mayor de dispositivos no tradicionales están siendo conectadas a Internet, como sistemas finales (véase el recuadro).

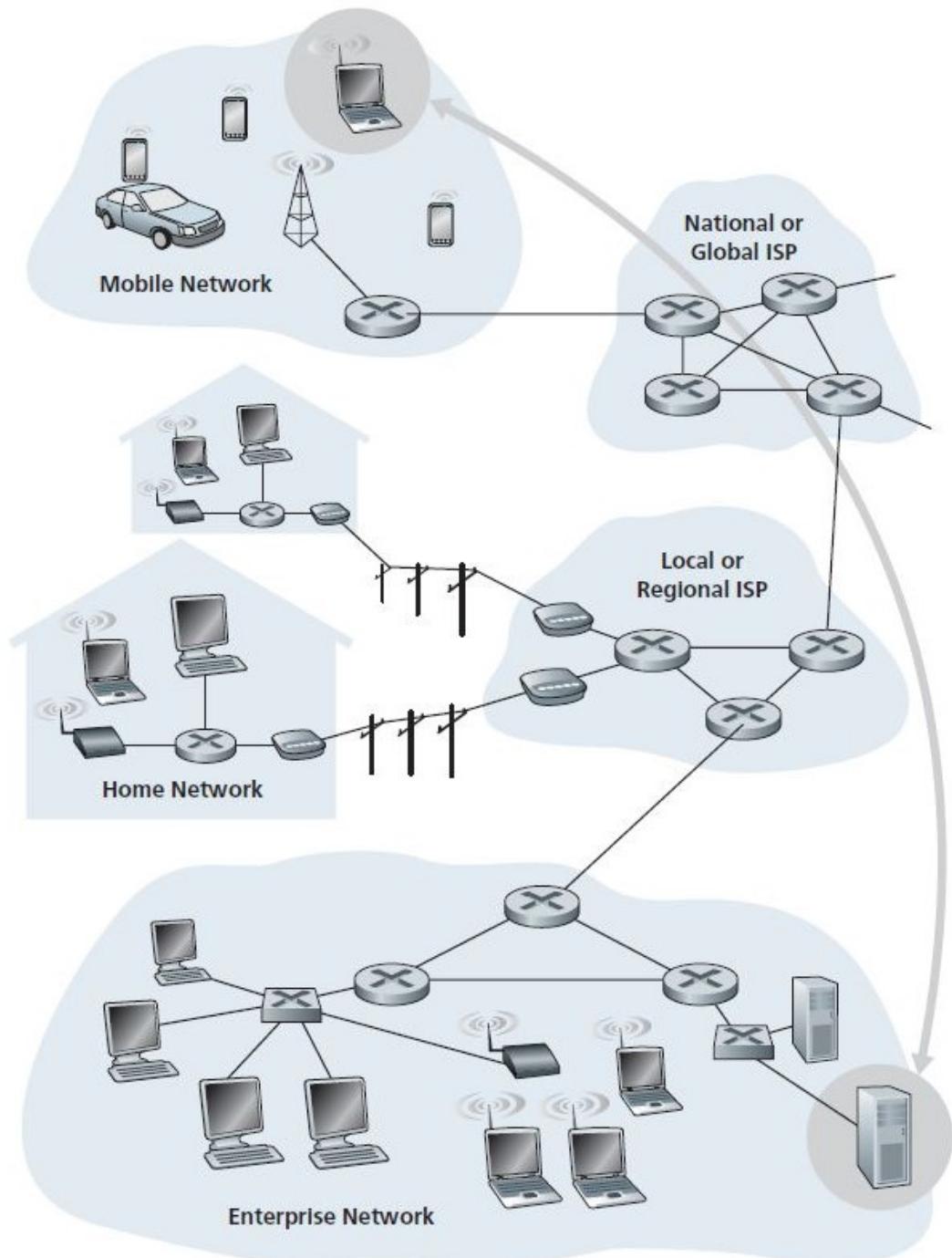


Figura 1.3 Interacción de Sistemas Finales

Los sistemas finales también se les conoce como *hosts* porque alojan (es decir, ejecutan) programas de aplicación tal como un programa navegador Web, un programa de servidor Web, un programa cliente de correo electrónico, o un programa de servidor de correo electrónico. A lo largo de este curso usaremos los términos *hosts* y *sistemas finales* de manera intercambiable; es decir, *host = sistema final*. Los hosts a veces se dividen en dos categorías: **clientes** y

servidores. De manera informal, los clientes tienden a ser PCs de escritorio y móviles, teléfonos inteligentes, etc., mientras que los servidores tienden a ser máquinas más potentes que almacenan y distribuyen páginas web, flujo de vídeo, retransmiten correo electrónico, y así sucesivamente. Hoy en día, la mayoría de los servidores de los que recibimos los resultados de búsqueda, correos electrónicos, páginas web y vídeos, residen en grandes centros de datos. Por ejemplo, Google tiene 30-50 centros de datos, y muchos de ellos más de cien mil servidores.

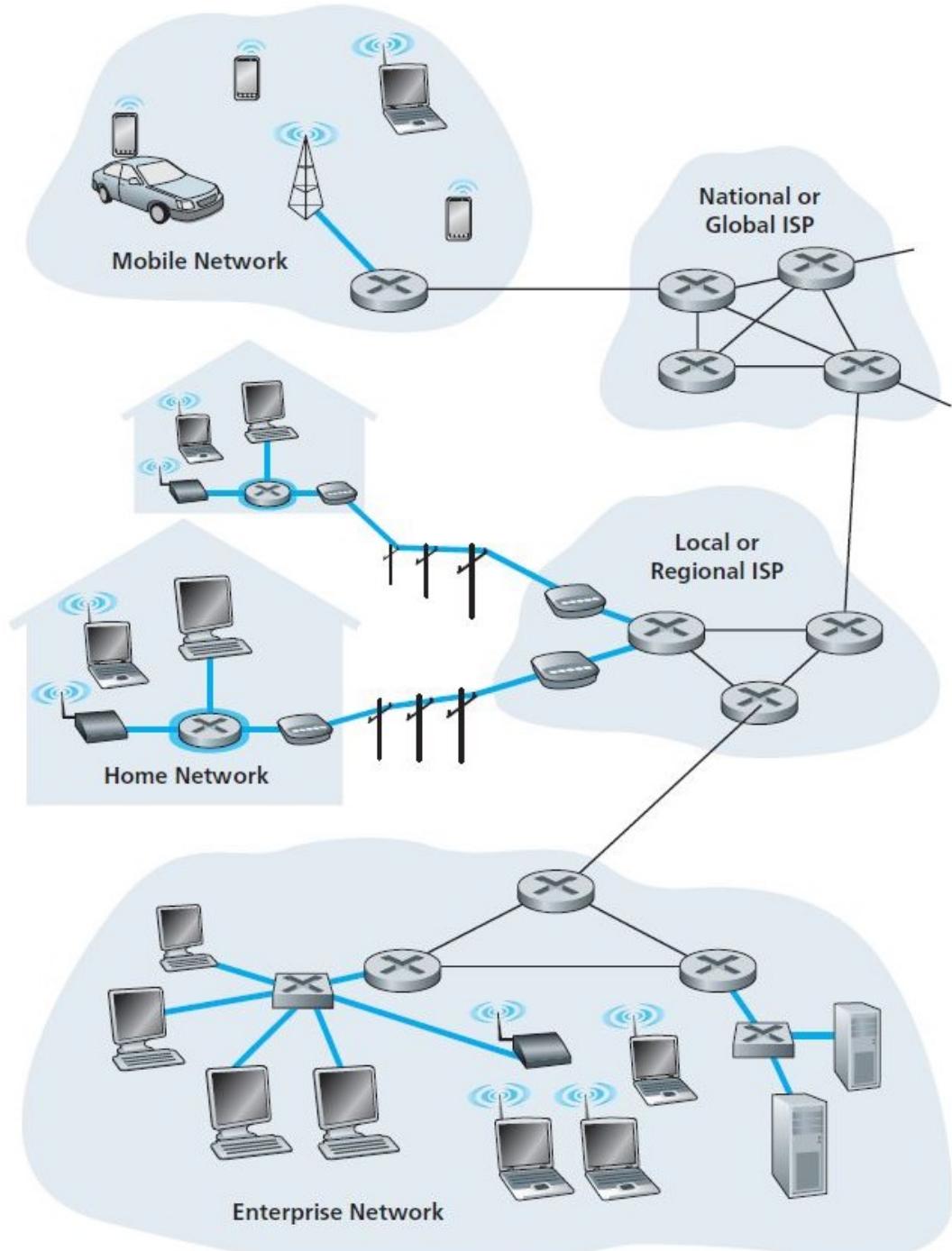


Figura 1.4 Redes de Acceso

1.2.1 Redes de Acceso

Habiendo examinado las aplicaciones y sistemas finales en el "extremo de la red", vamos a considerar a continuación la red de acceso -la red que conecta físicamente a un sistema final con el primer router (también conocido

como el "router de extremo") en un camino desde el sistema final a cualquier otro sistema final lejano. La Figura 1.4 muestra varios tipos de redes de acceso con líneas gruesas, líneas sombreadas, y la configuración (el hogar, la empresa, y el área amplia inalámbrica móvil) en el que se utilizan.

Acceso en Casa: DSL, cable, FTTH, Dial-Up, y Satélite

En los países desarrollados hoy en día, más del 65 por ciento de los hogares tiene acceso a Internet, con Corea, Países Bajos, Finlandia y Suecia a la cabeza con más del 80 por ciento de los hogares teniendo acceso a Internet, casi todos a través de una conexión de banda ancha de alta velocidad [UIT 2011]. Finlandia y España han declarado recientemente el acceso a Internet de alta velocidad como un "derecho legal". Dado este interés intenso en el acceso en casa, vamos a comenzar nuestra visión general de las redes de acceso al considerar cómo los hogares se conectan a Internet.

Hoy en día, los dos tipos más frecuentes de acceso residencial de banda ancha son la **línea de abonado digital (DSL)** y el cable. Una típica residencia obtiene acceso a Internet DSL de la misma compañía telefónica local (telecomunicaciones) que proporciona acceso a su cable de teléfono local. Por lo tanto, cuando se utiliza DSL, la compañía de telecomunicaciones de un cliente es también su proveedor de Internet. Como se muestra en la Figura 1.5, cada módem DSL del cliente utiliza la línea telefónica existente (par trenzado de alambre de cobre, del que hablaremos en la Sección 1.2.2) para intercambiar datos con un multiplexor de acceso de línea de abonado digital (DSLAM), ubicado en las oficinas centrales (OC) locales de la empresa de telecomunicaciones. El módem DSL de la casa toma datos digitales y los traduce a los tonos de alta frecuencia para la transmisión a través de cables de teléfono a la empresa de telecomunicaciones; las señales analógicas procedentes de muchas de estas casas se convierten de nuevo en formato digital en el DSLAM.

La línea telefónica residencial lleva los datos y señales telefónicas tradicionales simultáneamente, los que se codifican a diferentes frecuencias:

- Un canal descendente de alta velocidad, en la banda de 50 kHz a 1 MHz

- Un canal de subida de velocidad media, en la banda de 4 kHz a 50 kHz

- Un canal ordinario de teléfono de dos vías, en la banda de 0 a 4 kHz

Este enfoque hace que el enlace simple DSL parezca como si fuera tres enlaces separados, de forma que una llamada telefónica y una conexión a Internet pueden compartir el enlace DSL al mismo tiempo. (Vamos a describir esta técnica de multiplexación por división de frecuencia en la Sección 1.3.1). En el lado del cliente, un divisor separa las señales de datos y telefónicas que llegan a la casa y envía la señal de datos al módem DSL. En el lado de telecomunicaciones, en el OC, el DSLAM separa las señales de telefonía y de datos y envía los datos a Internet. Cientos o incluso miles de hogares se conectan a un único DSLAM [Dischinger de 2007].

Las normas DSL definen las tasas de transmisión de 12 Mbps de bajada y 1,8 Mbps de subida [ITU 1999], y 24 Mbps de bajada y 2,5 Mbps de subida de la [ITU 2003]. Debido a que las tasas de bajada y de subida son diferentes, se dice que el acceso es asimétrico. Las velocidades de transmisión descendentes y ascendentes reales obtenidos pueden ser inferiores a los niveles observados anteriormente, a medida que el proveedor de DSL pueda limitar a propósito una tasa residencial cuando se ofrecen servicios por niveles (diferentes velocidades, disponible a precios diferentes), o porque la velocidad máxima pueda limitarse por la distancia entre el hogar y la OC, el indicador de la línea de par trenzado y el grado de interferencia eléctrica. Los ingenieros han diseñado expresamente DSL para distancias cortas entre el hogar y la OC; Generalmente, si la residencia no se encuentra dentro de 5 a 10 millas de la OC, la residencia debe recurrir a una forma alternativa de acceso a Internet.

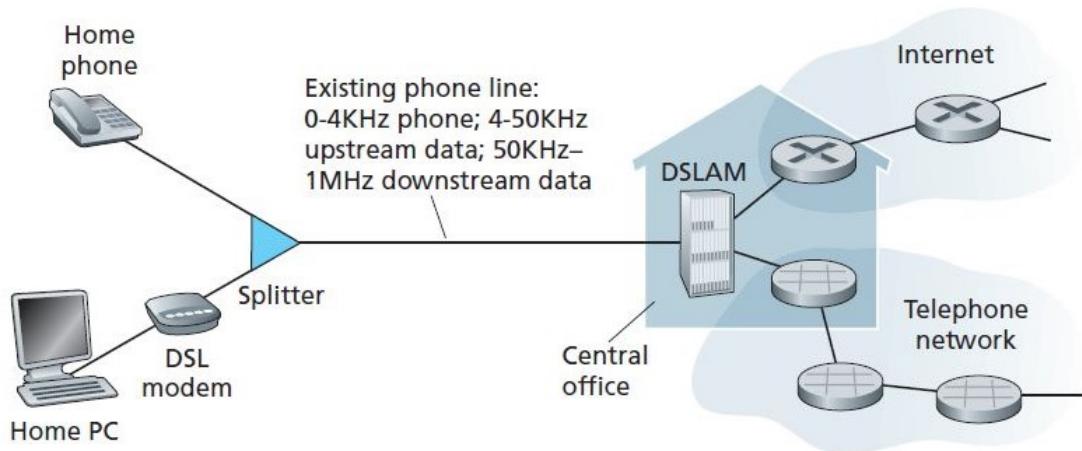


Figura 1.5 Internet de acceso DSL

Mientras DSL hace uso de la infraestructura de telefonía local existente de la compañía de telecomunicaciones, el **acceso a Internet por cable** hace uso de la infraestructura de televisión por cable existente de la compañía de televisión por cable. Una residencia obtiene el acceso a Internet por cable de la misma compañía que ofrece su televisión por cable. Como se ilustra en la Figura 1.6, las fibras ópticas conectan el extremo del cable a las uniones a nivel del barrio, de la que luego se utiliza el cable coaxial tradicional para llegar a las casas y apartamentos individuales. Cada unión de barrio soporta típicamente de 500 a 5.000 hogares. Debido a que tanto la fibra y cable coaxial se emplean en este sistema, a menudo se lo refiere como fibra coaxial híbrida (HFC).

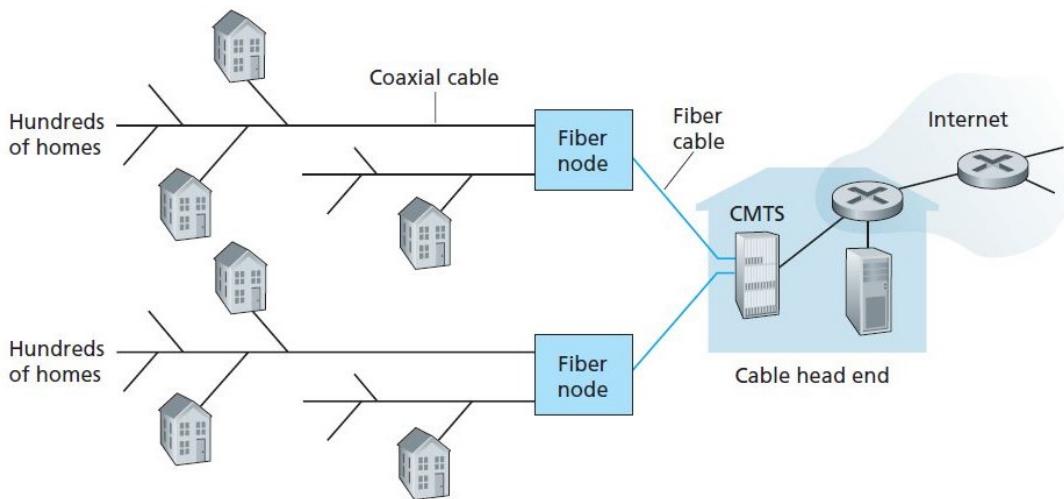


Figura 1.6 Red de Acceso Híbrida Fibra-Coaxial

El acceso a Internet por cable requiere módems especiales, llamados módems de cable. Al igual que con un módem DSL, el módem de cable es típicamente un dispositivo externo y se conecta a la PC de la casa a través de un puerto Ethernet. En el extremo del cable, el sistema de terminación del módem de cable (CMTS) cumple una función similar a la del DSLAM de la red DSL -convertir la señal analógica enviada desde los módems de cable en muchos hogares descendentes de nuevo en formato digital. Los módems de cable dividen la red HFC en dos canales, un canal descendente y uno ascendente. Al igual que con DSL, el acceso es típicamente asimétrico, con el canal descendente normalmente provisto de una velocidad de transmisión más alta que el canal ascendente. El estándar DOCSIS 2.0 define velocidades de bajada de hasta 42,8 Mbps y velocidades de subida de hasta 30,7 Mbps. Como en el caso de las redes DSL, la tasa máxima alcanzable no se puede realizar debido a velocidades de datos contratados inferiores o impedimentos de medios.

Una característica importante del acceso a Internet por cable es que es un medio de difusión compartida. En particular, cada paquete enviado por la parte superior del extremo viaja por el canal de bajada en cada enlace para cada hogar y cada paquete enviado por un hogar viaja por el canal de subida hacia la parte superior. Por esta razón, si varios usuarios están descargando al mismo tiempo un archivo de vídeo en el canal descendente, la tasa real en el que cada usuario recibe su archivo de vídeo será significativamente más baja que la tasa total del cable de bajada. Por otro lado, si hay sólo unos pocos usuarios activos y todos ellos sólo navegar en la Web, entonces cada uno de los usuarios puede recibir de hecho páginas Web a la velocidad total de bajada del cable, debido a que los usuarios rara vez van a solicitar una página Web exactamente a la misma hora. Debido a que el canal ascendente también es compartido, se necesita un protocolo de acceso múltiple distribuido para coordinar las transmisiones y evitar colisiones.

Aunque las redes de cable y DSL actualmente representan más del 90 por ciento de acceso de banda ancha residencial en los Estados Unidos, una tecnología prometedora y capaz, promete incluso velocidades más altas es el despliegue de **fibra hasta el hogar (FTTH)** [2011a FTTH Council]. Como su nombre lo indica, el concepto de FTTH es simple - proporcionar una ruta de fibra óptica desde la OC directamente a la casa.

Existen varias tecnologías que compiten por la distribución óptica desde la OC a los hogares. La red de distribución óptica más simple se llama fibra directa, con una fibra desde la CO hasta cada casa. Más comúnmente, cada fibra proveniente desde la oficina central es en realidad compartida por muchos hogares; no es hasta que la fibra se pone relativamente cerca de las casas que se divide en fibras individuales específicas del cliente. Hay dos arquitecturas de distribución óptica de red competidoras que realizan esta división: redes ópticas activas (AONs) y las redes ópticas pasivas (PON). AON es esencialmente Ethernet permutado.

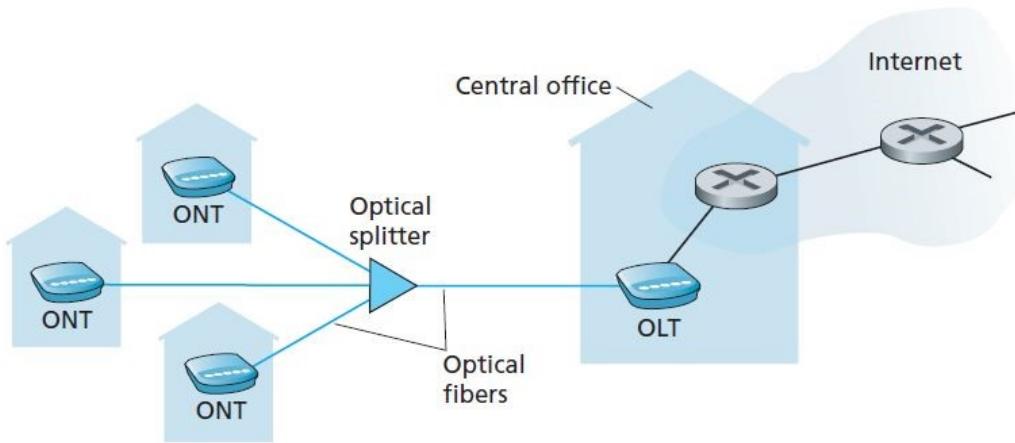


Figura 1.7 Internet de Acceso FTTH

Aquí, se discute brevemente PON, que se utiliza en el servicio FiOS de Verizon. La figura 1.7 muestra FTTH utilizando la arquitectura de distribución PON. Cada vivienda tiene un terminador de red óptica (ONT), que está conectado, por fibra óptica dedicada, a un divisor de barrio. El divisor combina un número de hogares (por lo general menos de 100) en una sola fibra óptica compartida, que se conecta a un terminador de línea óptica (OLT) en OC de la compañía de telecomunicaciones. El OLT, que proporciona la conversión entre las señales ópticas y eléctricas, se conecta a Internet a través de un router de telecomunicaciones. En el hogar, los usuarios conectan un router (normalmente un enrutador inalámbrico) a la ONT y acceden a Internet a través de este router hogareño. En la arquitectura de PON, todos los paquetes enviados desde OLT al divisor se replican en el divisor (similar a un cable en el extremo superior).

FTTH potencialmente puede proporcionar velocidades de acceso a Internet en un rango de gigabits por segundo. Sin embargo, la mayoría de los ISPs FTTH ofrecen diferentes ofertas de tasas, con las tasas más altas, naturalmente, constando más dinero. La velocidad de descarga promedio de los clientes de EE.UU de FTTH fue de aproximadamente 20 Mbps en 2011 (en comparación con 13 Mbps para redes de acceso por cable y menos de 5 Mbps para DSL) [2011b FTTH Council].

Otras dos tecnologías de acceso a la red también se utilizan para proporcionar acceso a Internet al hogar. En los lugares donde DSL, cable y FTTH no están disponibles (por ejemplo, en algunos entornos rurales), un enlace por satélite se puede utilizar para conectar una residencia a Internet a velocidades de más de 1 Mbps; StarBand y HughesNet son dos de esos proveedores de acceso por satélite. El acceso a través de líneas telefónicas tradicionales se basa en el mismo modelo que DSL -un módem hogareño se conecta a través de una línea telefónica con un módem en el ISP. En comparación con DSL y otras redes de acceso de banda ancha, el acceso dial-up es muy lento que 56 kbps.

El acceso en la empresa (y el hogar): Ethernet y Wi-Fi

En los campus universitarios y corporativos, y cada vez más en la configuración del hogar, una red de área local (LAN) se utiliza para conectar un sistema final al router del extremo. Aunque hay muchos tipos de tecnologías LAN, Ethernet es, con mucho, la tecnología de acceso más frecuente en las empresas, universidades y redes domésticas. Como se muestra en la Figura 1.8, los usuarios utilizan Ethernet de cable de par trenzado de cobre para conectarse a un comutador Ethernet. El switch de Ethernet, o una red de este tipo de switches interconectados es, a su vez, conectada a la Internet mayor. Con acceso a Ethernet, los usuarios suelen tener acceso 100 Mbps al comutador Ethernet, mientras que los servidores pueden tener 1 Gbps o incluso 10 Gbps de acceso.

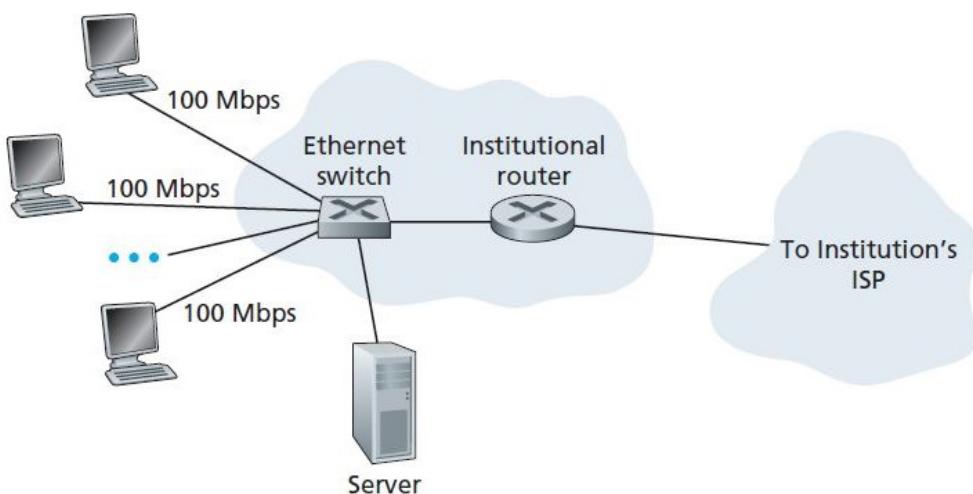


Figura 1.8 Internet de Acceso Ethernet

Sin embargo, cada vez más las personas están accediendo a Internet de forma inalámbrica desde computadoras portátiles, teléfonos inteligentes, tablets y otros dispositivos. En un entorno de LAN inalámbrica, los usuarios inalámbricos transmiten/reciben paquetes hacia/desde un punto de acceso que está conectado en la red de la empresa (lo más probable incluyendo Ethernet por cable), que a su vez está conectado a la Internet por cable. Un usuario de LAN inalámbrica típicamente debe estar dentro de unas pocas decenas de metros del punto de acceso. El acceso a una LAN inalámbrica basada en la tecnología IEEE 802.11, más coloquialmente conocido como Wi-Fi, ahora está casi en todas partes -universidades, oficinas, cafés, aeropuertos, viviendas, e incluso en los aviones. En muchas ciudades, uno puede estar parado en una esquina de la calle y estar dentro del rango de diez o veinte estaciones base. Como se analiza en detalle en el capítulo 6, 802.11 hoy proporciona una velocidad de transferencia conjunta de hasta 54 Mbps.

A pesar de que las redes de acceso a Ethernet y WiFi fueron desarrollados inicialmente en una configuración empresarial (empresa, universidad), se han convertido recientemente en componentes relativamente comunes de las redes domésticas. Muchos hogares combinan el acceso residencial de banda ancha (es decir, los módems de cable o DSL) con estas tecnologías LAN inalámbricas de bajo costo para crear redes domésticas de gran alcance [Edwards 2011]. La figura 1.9 muestra una red doméstica típica. Esta red doméstica consiste en un ordenador portátil de itinerancia, así como un PC por cable; una estación de base (el punto de acceso inalámbrico), que se comunica con el PC inalámbrico; un módem de cable, proporcionando acceso de banda ancha a Internet; y un router, que interconecta la estación base y el PC estacionario con el módem por cable. Esta red permite a los

miembros del hogar tener acceso de banda ancha a Internet con uno de los miembros de yendo desde la cocina hasta el patio trasero y a los dormitorios.

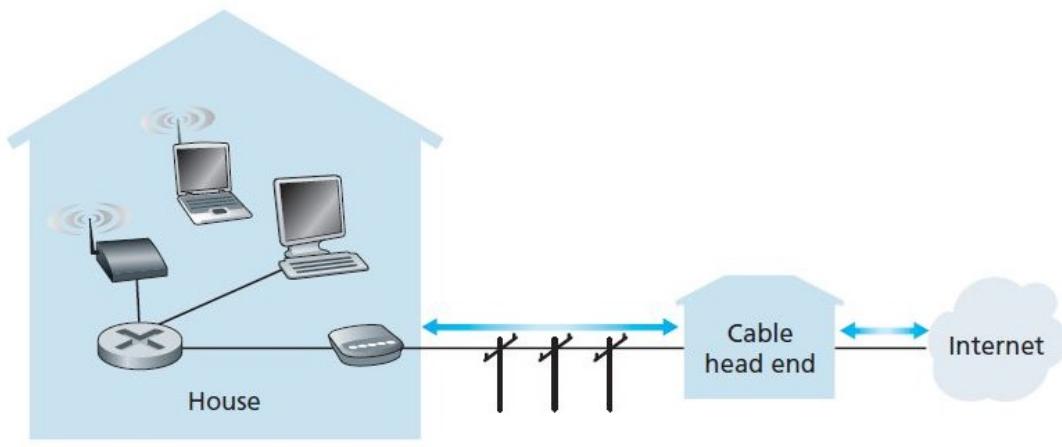


Figura 1.9 Red Hogareña Típica

Zona amplia de acceso inalámbrico: 3G y LTE

Cada vez más, los dispositivos como iPhones, BlackBerrys, y los dispositivos Android están siendo utilizados para enviar correo electrónico, navegar por Internet, Tweet, y descargar música, mientras que están en la ruta. Estos dispositivos emplean la misma infraestructura inalámbrica que se utiliza para la telefonía celular para enviar/recibir paquetes a través de una estación base que es operado por el proveedor de la red celular. A diferencia de WiFi, un usuario sólo tiene que estar dentro de unas pocas decenas de kilómetros (a diferencia de unas pocas decenas de metros) de la estación base.

Las empresas de telecomunicaciones han hecho enormes inversiones en la llamada red inalámbrica de tercera generación (3G), que proporciona una amplia zona de acceso a Internet inalámbrico de conmutación de paquetes a velocidades superiores a 1 Mbps. Pero incluso una tecnología de acceso de mayor velocidad de zona área -una cuarta generación (4G) de redes inalámbricas de área amplia- están siendo desarrolladas. LTE (para "Evolución a Largo Plazo", un candidato para el peor Acrónimo del Año) tiene sus raíces en la tecnología 3G, y potencialmente puede alcanzar tasas superiores a 10 Mbps. Las velocidades de bajada de LTE de muchas decenas de Mbps se han reportado en implementaciones comerciales.

1.2.2 El Medio Físico

En la subsección anterior, tuvimos una visión general de algunas de las tecnologías más importantes de acceso a la red en Internet. Mientras describíamos estas tecnologías, también indicamos los medios físicos utilizados. Por ejemplo, hemos dicho que el HFC utiliza una combinación de cable de fibra y cable coaxial. Hemos dicho que DSL y Ethernet utilizan alambre de cobre. Y dijimos que las redes de acceso móviles utilizan el espectro radioeléctrico. En este apartado se proporciona una breve descripción de estos y otros medios de transmisión que se utilizan comúnmente en Internet.

Con el fin de definir lo que se entiende por un medio físico, reflexionemos sobre la breve vida de un bit. Considera un bit viajando desde un sistema final, a través de una serie de enlaces y routers, a otro sistema final. Este pobre bit es expulsado alrededor y transmitido muchas, muchas veces! El sistema final de origen primero transmite el bit, y poco después el primer router de la serie recibe el bit; el primer router a continuación, transmite el bit, y poco después el segundo router recibe el bit; y así. Por lo tanto nuestro bit, cuando viaja desde el origen al destino, pasa a través de una serie de pares emisores-receptores. Para cada par emisor-receptor, el bit se envía mediante la propagación de ondas electromagnéticas o impulsos ópticos a través de un **medio físico**. El medio físico puede tomar muchas formas y no tiene que ser del mismo tipo para cada par emisor-receptor a lo largo del camino. Los ejemplos de medios físicos incluyen cable de cobre de par trenzado, cable coaxial, cable de fibra óptica multimodo, el espectro radioeléctrico terrestre, y el espectro de radio por satélite. Los medios físicos se dividen en dos cate-

gorías: **medios guiados** y **medios no guiados**. Con medios guiados, las ondas son guiadas a lo largo de un medio sólido, tal como un cable de fibra óptica, un cable de cobre de par trenzado o un cable coaxial. Con los medios de comunicación no guiados, las ondas se propagan en la atmósfera y en el espacio exterior, tal como en una LAN inalámbrica o un canal de satélite digital.

Pero antes de entrar en las características de los diferentes tipos de medios, digamos unas palabras sobre sus costos. El costo real del enlace físico (cable de cobre, cable de fibra óptica, etc.) a menudo es relativamente menor en comparación con otros costos de red. En particular, el coste de mano de obra asociada con la instalación de la conexión física puede ser de varias órdenes de magnitud más altas que el costo del material. Por esta razón, muchos constructores instalan par trenzado, fibra óptica, cable coaxial en todas las habitaciones del edificio. Incluso si sólo un medio se utiliza inicialmente, hay una buena probabilidad de que otro medio podría ser utilizado en un futuro próximo, y así se ahorra dinero al no tener que colocar cables adicionales en el futuro.

Par Trenzado-alambre de cobre

El medio de transmisión guiada más barato y más comúnmente utilizado es el cable de cobre de par trenzado. Durante más de cien años ha sido utilizado por las redes telefónicas. De hecho, más de 99 por ciento de las conexiones cableadas desde el auricular del teléfono hasta switch de telefónico local usan cable de cobre de par trenzado. La mayoría de nosotros hemos visto par trenzado en nuestros hogares y ambientes de trabajo. El par trenzado se compone de dos cables de cobre aislados, cada uno de aproximadamente 1 mm de espesor, dispuestos en un patrón en espiral regular. Los cables están trenzados juntos para reducir la interferencia eléctrica de pares similares cerca. Típicamente, un número de pares están empaquetados juntos en un cable, envolviendo los pares en un escudo protector. Un par de cables constituye un único enlace de comunicación. **Par trenzado sin blindaje (UTP, Unshielded Twister pair)** se utiliza comúnmente para las redes de computadoras dentro de un edificio, es decir, para redes de área local. Las tasas de datos para redes de área local que utilizan par trenzado hoy oscilan de 10 Mbps a 10 Gbps. Las velocidades de datos que se pueden obtener dependen del grosor del cable y la distancia entre el transmisor y el receptor.

Cuando la tecnología de fibra óptica surgió en la década de 1980, muchas personas menospreciaron el par trenzado debido a sus velocidades de bits relativamente bajas. Algunas personas incluso sintieron que la tecnología de fibra óptica podría sustituir por completo al par trenzado. Pero el par trenzado no se dio por vencido tan fácilmente. La tecnología moderna de par trenzado, como el cable de categoría 6A, puede alcanzar velocidades de datos de 10 Gbps para distancias de hasta cien metros. Al final, de par trenzado se ha convertido en la solución dominante para la creación de redes LAN de alta velocidad.

Como se señaló anteriormente, el par trenzado también se utiliza comúnmente para el acceso a Internet residencial. Vimos que la tecnología de módem de acceso telefónico (dial-up) permite el acceso a velocidades de hasta 56 kbps sobre par trenzado. También hemos visto que la tecnología DSL (línea de abonado digital) ha permitido a los usuarios residenciales acceder a Internet en decenas de Mbps a través del par trenzado (cuando los usuarios viven cerca de módem del ISP).

Cable coaxial

Como el par trenzado, el cable coaxial consta de dos conductores de cobre, pero los dos conductores son concéntricos en vez de en paralelo. Con esta construcción y con aislamiento y blindaje especiales, el cable coaxial puede alcanzar altas velocidades de transmisión de datos. El cable coaxial es bastante común en los sistemas de televisión por cable. Como hemos visto anteriormente, los sistemas de televisión por cable han sido recientemente acoplados junto con cable de módems para 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 cambia la señal digital a una banda de frecuencia específica, y la señal analógica resultante es enviada desde el transmisor a uno o más receptores. El cable coaxial se puede utilizar como un **medio compartido** guiado. Específicamente, una serie de sistemas finales se pueden conectar directamente al cable, con cada uno de los sistemas finales recibiendo la señal enviada por los otros sistemas finales.

Fibra óptica

Una fibra óptica es un medio delgado y flexible que conduce pulsos de luz, con cada pulso representando un bit. Una sola fibra óptica puede soportar velocidades de bits enormes, de hasta decenas o incluso cientos de gigabits por segundo. Ellos son inmunes a las interferencias electromagnéticas, tienen muy baja atenuación de señal de hasta 100 kilómetros, y son muy difíciles de interceptar. Estas características han hecho de la fibra óptica el medio guiado de transmisión de gran recorrido preferido, en particular para los enlaces de ultramar. Muchas de las redes telefónicas de larga distancia en los Estados Unidos y en otros lugares utilizan actualmente la fibra óptica exclusivamente. La fibra óptica es también frecuente en la columna vertebral de Internet. Sin embargo, el alto costo de los dispositivos ópticos -tales como emisores, receptores, y los interruptores- han dificultado su desarrollo para el transporte de corta distancia, como en una LAN o en el hogar en una red de acceso residencial. Las velocidades del enlace estándar de portador óptico (OC) oscilan entre 51,8 Mbps a 39,8 Gbps; estas especificaciones se refieren a menudo como OCn, donde la velocidad de enlace es igual a $n \times 51,8$ Mbps. Los estándares en uso hoy en día incluyen OC-1, OC-3, OC-12, OC-24, OC-48, OC-96, OC-192, OC-768. [Mukherjee 2006, Ramaswamy 2010] proporcionan una cobertura de diversos aspectos de las redes ópticas.

Los canales de radio terrestre

Los canales de radio llevan señales en el espectro electromagnético. Ellos son un medio atractivo ya que no requieren cable físico para ser instalado, puede penetrar paredes, proporcionar conectividad a un usuario móvil, y potencialmente puede llevar una señal largas distancias. Las características de un canal de radio dependen significativamente del entorno de propagación y de la distancia sobre la cual una señal se va a transportar. Las consideraciones del entorno determinan la pérdida de camino y el descavecimiento de la sombra (que disminuyen la intensidad de la señal a medida que la señal viaja a través de cierta distancia y alrededor/a través de los objetos obstruyentes), desvanecimiento múltiple (debido a la reflexión de la señal en los objetos interfirientes), y la interferencia (debido a otras transmisiones y señales electromagnéticas).

Los canales de radio terrestres se pueden clasificar en tres grupos: los que operan sobre distancias muy corta (por ejemplo, con uno o dos metros); los que operan en áreas locales, por lo general abarca de diez a unos pocos cientos de metros; y las que operan en la zona amplias, que se extiende decenas de kilómetros. Los dispositivos personales tales como auriculares inalámbricos, teclados y dispositivos médicos funcionan a corta distancia; las tecnologías LAN inalámbricas que se describen en la sección 2.1 usan los canales de radio de área local; las tecnologías de acceso de radio celulares utilizan canales de área amplia.

Los canales de radio por satélite

Un satélite de comunicación une dos o más emisores/receptores de microondas basado en la Tierra, conocidos como las estaciones de tierra. El satélite recibe las transmisiones en una banda de frecuencia, se regenera la señal utilizando un repetidor (discutido más adelante), y transmite la señal en otra frecuencia. Hay dos tipos de satélites que se utilizan en las comunicaciones: los **satélites geoestacionarios** y **satélites terrestres de órbita baja (LEO)**.

Los satélites geoestacionarios se mantienen de forma permanente sobre el mismo punto en la Tierra. Esta presencia estacionaria se consigue colocando el satélite en órbita a 36.000 kilómetros sobre la superficie de la Tierra. Esta enorme distancia de la estación de tierra a través del satélite, de regreso a la estación de tierra introduce un retardo de propagación de la señal sustancial de 280 milisegundos. Sin embargo, los enlaces por satélite, que pueden operar a velocidades de cientos de Mbps, se utilizan a menudo en zonas sin acceso a DSL o acceso a Internet por cable.

Los satélites LEO se colocan mucho más cercanos a la Tierra y no se quedan permanentemente por encima de un punto de la Tierra. Giran alrededor de la Tierra (tal como lo hace la Luna) y pueden comunicarse entre sí, así como con las estaciones de tierra. Para proporcionar una cobertura continua a una zona, muchos satélites necesitan ser colocados en órbita. En este momento hay muchos sistemas de comunicación de baja altitud en desarrollo. La página Web de constelaciones de satélites de Lloyd [Wood 2012] ofrece y recoge información sobre los sistemas de constelación de satélites de comunicaciones. La tecnología de los satélites LEO puede ser utilizada para acceder a Internet en algún momento en el futuro.

1.3 El núcleo de la Red

Habiendo examinado los márgenes de Internet, ahora vamos a ahondar más profundamente en el interior del núcleo de la red -la malla de conmutadores de paquetes y enlaces que interconecta los sistemas finales de Internet. La figura 1.10 remarcá el núcleo de la red con líneas gruesas y sombreadas.

1.3.1 La conmutación de paquetes

En una aplicación de red, los sistemas finales intercambian **mensajes** entre sí. Los mensajes pueden contener cualquier cosa que el diseñador de la aplicación quiera. Los mensajes pueden realizar una función de control (por ejemplo, los mensajes "Hola" en nuestro ejemplo el apretón de manos en la Figura 1.2) o pueden contener datos, como un mensaje de correo electrónico, una imagen JPEG o un archivo de audio MP3. Para enviar un mensaje de un sistema final de origen a un sistema final de destino, el emisor parte mensajes largos en fragmentos de datos más pequeños conocidos como **paquetes**. Entre el origen y el destino, cada paquete viaja a través de enlaces de comunicación y **comutadores de paquetes** (de los cuales hay dos tipos predominantes, **routers** y **switches de capa de enlace**). Los paquetes se transmiten a través de cada enlace de comunicación a una tasa igual a la tasa de transmisión *total* del enlace. Por lo tanto, si un sistema final emisor o un comutador de paquetes está enviando un paquete de L bits sobre un enlace con una tasa de transmisión de R bits/seg, entonces el tiempo para transmitir el paquete es de L/R segundos.

Transmisión de Almacenamiento y Envío

La mayoría de los comutadores de paquetes utilizan **transmisión de almacenamiento y envío (store-and-forward transmission)** en las entradas de los enlaces. La transmisión de almacenamiento y envío significa que el comutador de paquetes debe recibir todo el paquete antes de que pueda comenzar a transmitir el primer bit del paquete en el enlace de salida. Para explorar la transmisión de almacenamiento y envío con más detalle, considere una red simple que consiste en dos sistemas finales conectados por un solo router, como se muestra en la Figura 1.11. Un router normalmente tendrá muchos enlaces de incidentes, ya que su trabajo consiste en conmutar un paquete entrante a un enlace de salida; En este sencillo ejemplo, el router tiene la simple tarea de transferir un paquete desde un enlace (de entrada) hacia el único otro enlace adjunto. En este ejemplo, el emisor tiene tres paquetes, cada uno compuesto de L bits, para enviar al destino. En la instantánea de tiempo mostrada en la figura 1.11, la fuente ha transmitido algunas partes del paquete 1, y la parte frontal del paquete 1 ya ha llegado al router. Debido a que el router emplea almacenamiento y envío, en este instante de tiempo, el router no puede transmitir los bits que ha recibido; en su lugar, primero debe almacenar los bits del paquete (es decir, "almacenar"). Sólo después de que el router ha recibido *todos* los bits del paquete puede empezar a transmitir (es decir, "enviar") el paquete en el enlace de salida. Para hacerse una idea de la transmisión store-and-forward, ahora vamos a calcular la cantidad de tiempo que transcurre desde que la fuente comienza a enviar el paquete hasta que el destino ha recibido el paquete entero. (Aquí vamos a ignorar el retraso de la propagación de tiempo que toma para que los bits viajen a través del cable cerca de la velocidad de la luz). La fuente comienza a transmitir en el tiempo 0; en el tiempo de segundos L/R , la fuente ha transmitido el paquete completo, y todo el paquete ha sido recibido y almacenado en el router (ya que no hay retraso de propagación). En el tiempo L/R segundos, ya que el router acaba de recibir todo el paquete, se pueden empezar a transmitir el paquete en el enlace de salida hacia el destino; en el tiempo $2L/R$, el router ha transmitido el paquete completo, y todo el paquete ha sido recibido por el destino. Por lo tanto, el retraso total es $2L/R$. Si el interruptor en lugar reenvía los bits tan pronto como llegan (sin antes recibir el paquete entero), entonces el retraso total sería de L/R puesto que los bits no son detenidos en el router. Pero, como veremos en la sección 1.4, los routers necesitan recibir, almacenar y procesar todo el paquete antes de reenviarlo.

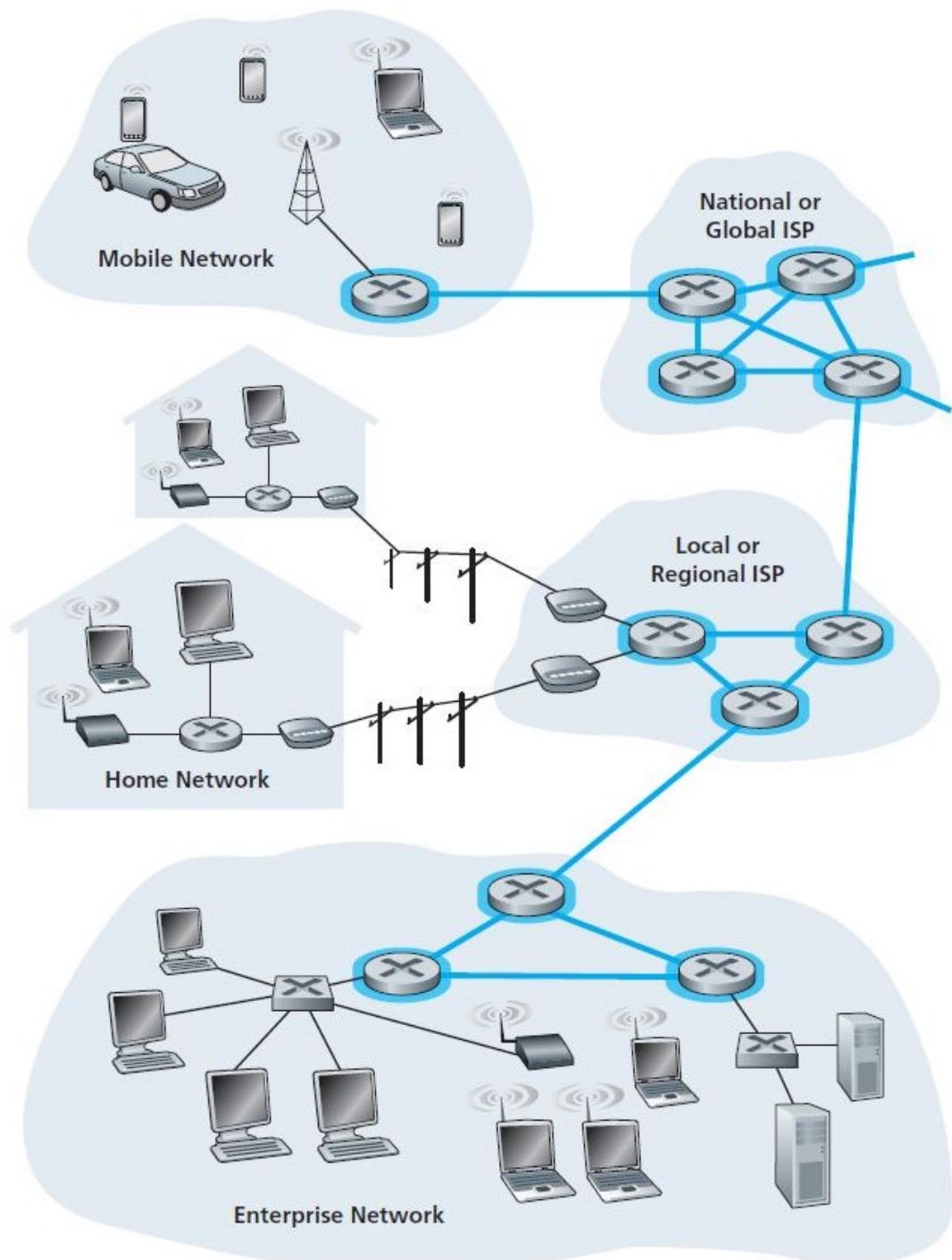


Figura 1.10 El Corazón de la Red

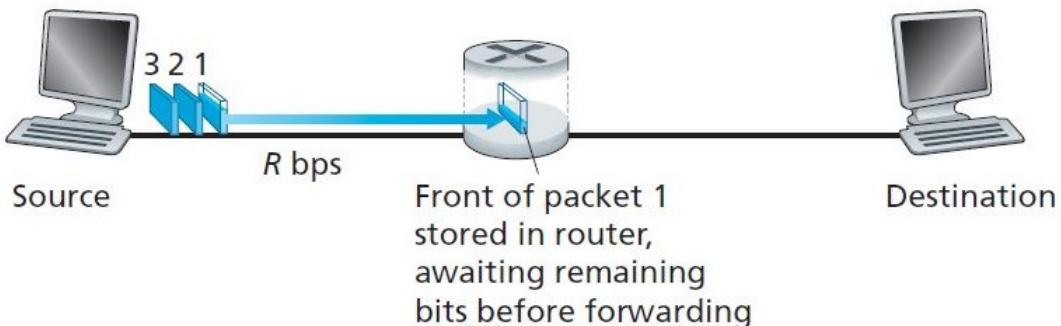


Figura 1.11 Intercambio de Paquetes con Almacenamiento y Envío

Ahora vamos a calcular la cantidad de tiempo que transcurre desde que el emisor comienza a enviar el primer paquete hasta que el destino ha recibido las tres paquetes. Como antes, en el momento de L/R , el router comienza a reenviar el primer paquete. Pero también en el tiempo de L/R , el emisor comenzará a enviar el segundo paquete, ya que acaba de terminar el envío de todo el primera paquete. Por lo tanto, en el tiempo $2L/R$, el destino ha recibido el primer paquete y el router ha recibido el segundo paquete. Del mismo modo, en el tiempo $3L/R$, el destino ha recibido los dos primeros paquetes y el router ha recibido el tercer paquete. Por último, en el momento $4L/R$ el destino ha recibido las tres paquetes!

Consideremos ahora el caso general de enviar un paquete desde el emisor hacia el destino sobre una trayectoria que consiste en N enlaces cada uno con una tasa R (por lo tanto, hay $N-1$ routers entre la fuente y de destino). Aplicando la misma lógica que el anterior, vemos que el retardo de extremo a extremo es:

$$d_{end-to-end} = N \frac{L}{R}$$

Es posible que ahora quieras tratar de determinar cuál sería el retardo de P paquetes enviados a través de una serie de N enlaces.

Retrasos de encolado y pérdida de paquetes

Cada conmutador de paquetes tiene varios enlaces adjuntos a la misma. Para cada enlace adjunto, el conmutador de paquetes tiene un **búfer de salida** (también llamado una **cola de salida**), que almacena los paquetes que el router está a punto de enviar a ese enlace. Los buffers de salida juegan un papel clave en la conmutación de paquetes. Si un paquete que llega debe ser transmitido en un enlace, pero encuentra al enlace ocupado con la transmisión de otro paquete, el paquete que llega debe esperar en la buffer de salida. Por lo tanto, además de los retrasos de almacenamiento y envío, los paquetes sufren **demoras de encolamiento** en los buffer de salida. Estos retrasos son variables y dependen del nivel de congestión en la red. Puesto que la cantidad de espacio de memoria de los buffers es finito, un paquete que llega puede encontrar que el buffer está completamente lleno con otros paquetes en espera de transmisión. En este caso, se producirá la **pérdida de paquetes** -ya sea que el paquete que llega o uno de los paquetes ya encolados sea dado de baja.

La Figura 1.12 ilustra una sencilla red de conmutación de paquetes. Como en la figura 1.11, los paquetes están representados por losas tridimensionales. La anchura de una losa representa el número de bits en el paquete. En esta figura, todos los paquetes tienen el mismo ancho y por lo tanto la misma longitud. Supongamos que los hosts A y B están enviando paquetes al host E. Los hosts A y B primero envían sus paquetes a lo largo de enlaces Ethernet de 10 Mbps, al primer router. El router entonces dirige estos paquetes al enlace de 1,5 Mbps. Si, durante un corto intervalo de tiempo, la tasa de llegada de paquetes al enrutador (cuando se convierte en bits por segundo) es superior a 1,5 Mbps, una congestión se producirá en el router a medida que los buffers de paquetes en

el enlace de salida, antes de ser transmitidos por el enlace. Por ejemplo, si los hosts A y B, cada uno envían una ráfaga de cinco paquetes uno detrás de otro, al mismo tiempo, entonces la mayoría de estos paquetes pasarán un tiempo esperando en la cola. La situación es, de hecho, completamente análoga a muchas situaciones comunes diarias, por ejemplo, cuando esperamos en la cola por un cajero de banco o esperamos delante de una casilla de peaje.

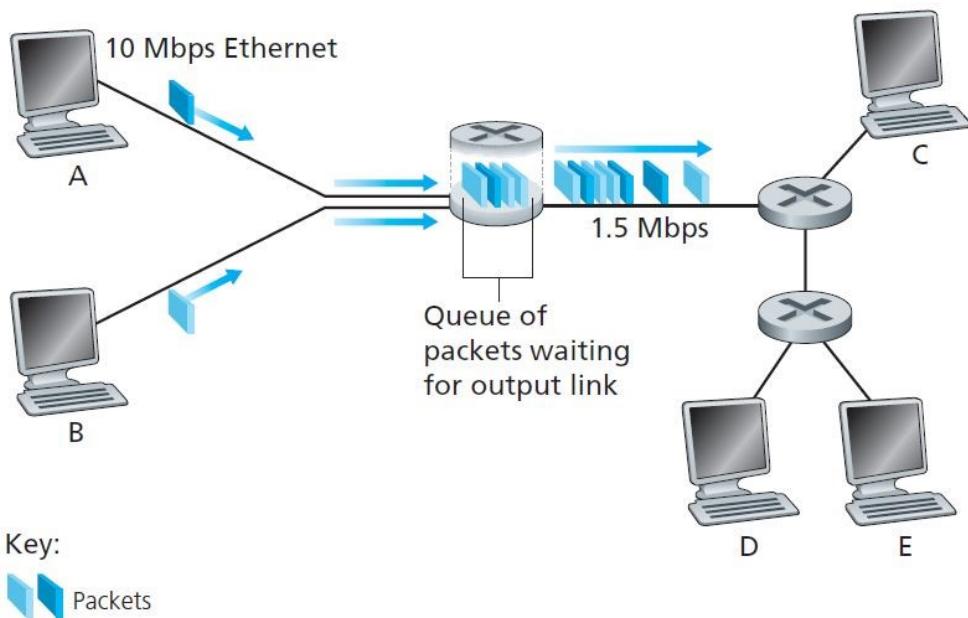


Figura 1.12 Intercambio de Paquetes

Tablas de reenvío y protocolos de enrutamiento

Anteriormente, dijimos que un router toma un paquete que llega en uno de sus enlaces de comunicación conectados y envía ese paquete a otro de sus enlaces de comunicación conectados. Pero ¿cómo determina el router a cuál enlace debe remitir el paquete? El reenvío de paquetes se hace realmente de manera diferente en distintos tipos de redes de computadoras. A continuación, describimos brevemente cómo se hace en Internet.

En Internet, cada sistema final tiene una dirección de llamada dirección IP. Cuando un sistema final origen desea enviar un paquete a un sistema final de destino, la fuente incluye la dirección IP de destino en el encabezado del paquete. Al igual que con las direcciones postales, esta dirección tiene una estructura jerárquica. Cuando un paquete llega a un router en la red, el router examina la porción de la dirección de destino del paquete y envía el paquete a un router adyacente. Más específicamente, cada router tiene una tabla de reenvío que asigna direcciones de destino (o partes de las direcciones de destino) a los enlaces salientes de ese router. Cuando un paquete llega a un router, el router examina la dirección y busca en su **tabla de reenvío**, utilizando esta dirección de destino, para encontrar el enlace de salida correspondiente. El router entonces dirige el paquete a este enlace saliente.

El proceso de enrutamiento de extremo a extremo es análogo a un conductor de coche que no utiliza mapas sino que prefiere pedir direcciones. Por ejemplo, supongamos que Juan está manejando desde Buenos Aires a Bv San Juan 156, en Córdoba Capital. Juan primero conduce a su estación de servicio en el barrio y pregunta cómo llegar a Bv San Juan 156, en Córdoba Capital. El empleado de la estación extrae la porción de la dirección de Córdoba Capital y le dice a Juan que necesita tomar la ruta 9 sur, que tiene una entrada justo al lado de la estación. También le dice a Juan que una vez que entre en Córdoba Capital, deberá preguntarle a alguien más allá. Juan entonces toma la ruta 9 sur hasta que llega a Córdoba, momento en el que pide a otro empleado de otra estación de servicios nuevas indicaciones. El empleado extrae la porción de la dirección de Córdoba Capital y le dice a Juan que debe continuar por la ruta 9 sur hasta cierto punto y luego pedir otra indicación a otra persona. En dicho punto, otro empleado de una estación de servicios también extrae la parte de la dirección de Córdoba Capital y le dice a Juan que debe tomar la ruta 56 directamente a Córdoba Capital. Juan toma la ruta 56 y se baja en la salida de Córdoba Capital. Juan va a otra estación, y esta vez el encargado extrae la porción de Bv. San Juan 156 de la

dirección y le dice a Juan el camino que debe seguir para llegar a Bv. San Juan. Una vez que Juan llega a Bv. San Juan, le pregunta a un niño en una bicicleta como llegar a su destino. El chico extrae la porción 156 de la dirección y señala la casa. Juan finalmente llega a su destino final. En la analogía anterior, los empleados de las estaciones y el niño en bicicleta son análogos a los routers.

Acabamos de aprender de que un router utiliza la dirección de destino de un paquete para indexar una tabla de reenvío y determinar el enlace de salida correspondiente. Pero esta declaración pide otra pregunta: ¿Cómo se establecen las tablas de reenvío? Están configuradas a mano en todos y cada enrutador, o usa Internet un procedimiento más automatizado? Este problema será estudiado en profundidad en el capítulo 4. Sin embargo, para dar una pequeña introducción, tendremos en cuenta que Internet tiene una serie de **protocolos de enrutamiento** especiales que se utilizan para ajustar automáticamente las tablas de reenvío. Un protocolo de enrutamiento puede ser, por ejemplo, determinar la ruta más corta desde cada router a cada destino y utilizar los resultados de camino más corto para configurar las tablas de reenvío en los routers.

¿Cómo te gustaría ver realmente la ruta de extremo a extremo que los paquetes toman en el Internet? Te invitamos a ensuciarse las manos al interactuar con el programa traceroute. Sólo tiene que visitar el sitio www.traceroute.org, elija una fuente en un determinado país, y trazar la ruta de esa fuente al ordenador. (Para una discusión de Traceroute, véase la sección 1.4).

1.3.2 Comutación de circuitos

Existen dos aproximaciones fundamentales para enviar datos a través de una red de enlaces y comitadores: **comutación de circuitos** y **comutación de paquetes**. Habiendo cubierto las redes de comitación de paquetes en la sección previa, volvamos nuestra atención a las redes de comutación de circuitos.

En las redes de comutación de circuitos, los recursos que se necesitan a lo largo de un recorrido (buffers, ancho de banda del enlace) para proporcionar la comunicación entre los sistemas terminales están *reservados* durante la duración de la sesión de comunicación entre los sistemas finales. En las redes de comutación de paquetes, estos recursos *no* son reservados; los mensajes de la sesión utilizan los recursos bajo demanda, y en consecuencia pueden tener que esperar (es decir, encolarse) para acceder a los recursos del enlace de comunicación. Como una analogía sencilla, consideremos dos restaurantes: uno que precisa reserva, y otro que ni precisa ni acepta reserva. Para el restaurante que precisa reserva, tenemos que pasar por el lío de llamar antes de salir de casa, pero cuando llegamos al restaurante podemos, en principio, ser ubicados y pedir nuestra comida. Para el restaurante que no requiere reserva no necesitamos preocuparnos de reservar una mesa, pero cuando lleguemos al restaurante, puede que tengamos que esperar mesa antes de ser ubicados.

Las redes de telefonía tradicionales son ejemplos de redes de comutación de circuitos. Consideremos qué ocurre cuando una persona quiere enviar información (voz o fax) a otra a través de la red. Antes de que el emisor pueda enviar información, la red debe establecer primero una conexión entre el emisor y el receptor. Esta es una conexión *bona fide, de buena fe* para la cual los comutadores a lo largo del camino entre el emisor y el receptor mantienen un estado de conexión para la misma. En la jerga de la telefonía, esta conexión es llamada **circuito**. Cuando la red establece un circuito, también reserva una velocidad de transmisión constante en los enlaces de la red (representando una fracción de la capacidad de transmisión de cada enlace) durante la duración de la conexión. Debido a que una velocidad de transmisión ha sido reservada para esta conexión emisor-receptor, el emisor puede transferir los datos al receptor a la velocidad constante *garantizada*.

La Figura 1.13 muestra una red de comutación de circuitos. En esta red, los cuatro switches de circuitos están interconectados por cuatro enlaces. Cada uno de estos enlaces tiene cuatro circuitos, por lo que cada enlace puede soportar cuatro conexiones simultáneas. Los hosts (por ejemplo, PC y estaciones de trabajo) están conectados directamente a uno de los switches. Cuando dos hosts quieren comunicarse, la red establece una **conexión terminal-a-terminal** dedicada entre las dos nodos. Por tanto, para que el Host A envíe mensajes al Host B, la red debe reservar primero 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 enlace. Como cada enlace tiene cuatro circuitos, por cada enlace utilizado en la conexión extremo-a-extremo, la conexión conseguirá un cuarto del ancho de banda total del enlace para la duración de la conexión. Así, por ejemplo, si cada enlace entre switches adyacentes tienen una velocidad de transmisión de 1 Mbps, entonces cada conexión de comutación

de circuitos extremo a extremo, recibe 250 kbps de velocidad de transmisión dedicada.

Por el contrario, considere lo que ocurre cuando un host quiere enviar un paquete a otro host a través de una red de conmutación de paquetes, tal como Internet. Al igual que con la conmutación de circuitos, el paquete se transmite a través de una serie de enlaces de comunicación. Pero a diferencia de la conmutación de circuitos, el paquete se envía a la red sin reservar ningún recursos del enlace que sea. Si uno de los enlaces está congestionado debido a que otros paquetes necesitan ser transmitidos a través del enlace, al mismo tiempo, entonces el paquete tendrá que esperar en un buffer intermedio en el lado emisor del enlace de transmisión y sufrir un retraso. Internet hace su mejor esfuerzo para entregar los paquetes en el momento oportuno, pero no hace ninguna garantía.

Multiplexado en redes de conmutación de circuitos

Un circuito es un enlace implementado ya sea mediante **multiplexado por división de frecuencia (FDM)**, o bien mediante **multiplexado por división de tiempo (TDM)**. Con FDM, el espectro de frecuencia de un enlace es compartido entre las conexiones establecidas a lo largo del enlace. Específicamente, el enlace dedica una banda de frecuencia para conexión durante la duración de la misma. En redes telefónicas, esta banda de frecuencia tiene típicamente un ancho de banda de 4 kHz (es decir, 4.000 hertz. o 4.000 ciclos por segundo). El ancho de esta banda se llama, lógicamente, **ancho de banda**. Las estaciones de radio FM utilizan también FDM para compartir el espectro de frecuencias entre 88 MHz y 108 MHz, con cada estación siendo alojada en una banda de frecuencia específica.

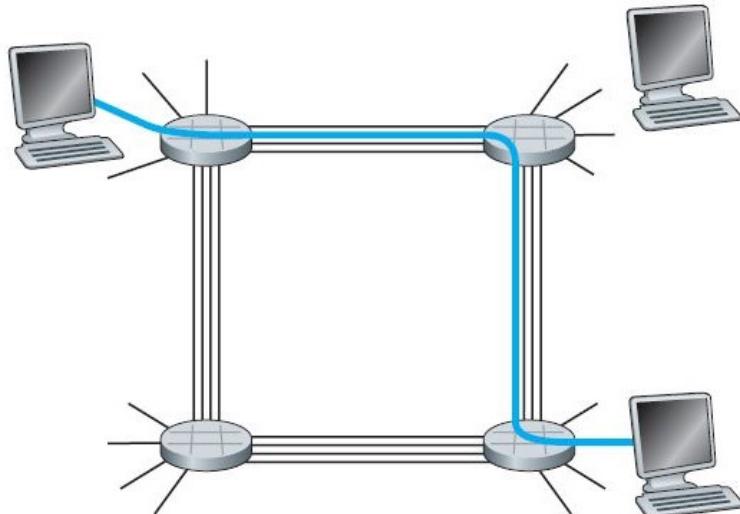


Figura 1.13 Red Simple de Circuito de Intercambio consistente de 4 switches y 4 links

Para un enlace TDM, el tiempo está dividido en marcos de duración fija, y cada marco está dividido en un número fijo de particiones (*slots*). Cuando la red establece una conexión sobre el enlace, dedica una partición de tiempo en cada marco de la conexión. Estas particiones están dedicadas para uso exclusivo de dicha conexión, con una partición de tiempo disponible para utilizar (en cada marco) para transmitir los datos de la conexión.

La Figura 1.14 muestra FDM y TDM para un enlace de red específico que soporta hasta 4 circuitos. Para FDM, el dominio de frecuencia está segmentado en cuatro bandas, cada una con un ancho de banda de 4 kHz. Para TDM, el dominio del tiempo está segmentado en marcos, con cuatro particiones de tiempo en cada marco; cada circuito es asignado a la misma franja dedicada en la suelta de los marcos TDM. Para TDM, la tasa de transmisión de un circuito es igual a la tasa de marcos multiplicada por el número de bits en una partición. Por ejemplo, si el enlace transmite 8.000 marcos por segundo y cada partición consta de 8 bits, entonces la tasa de transmisión de un circuito es de 64 Kbps.

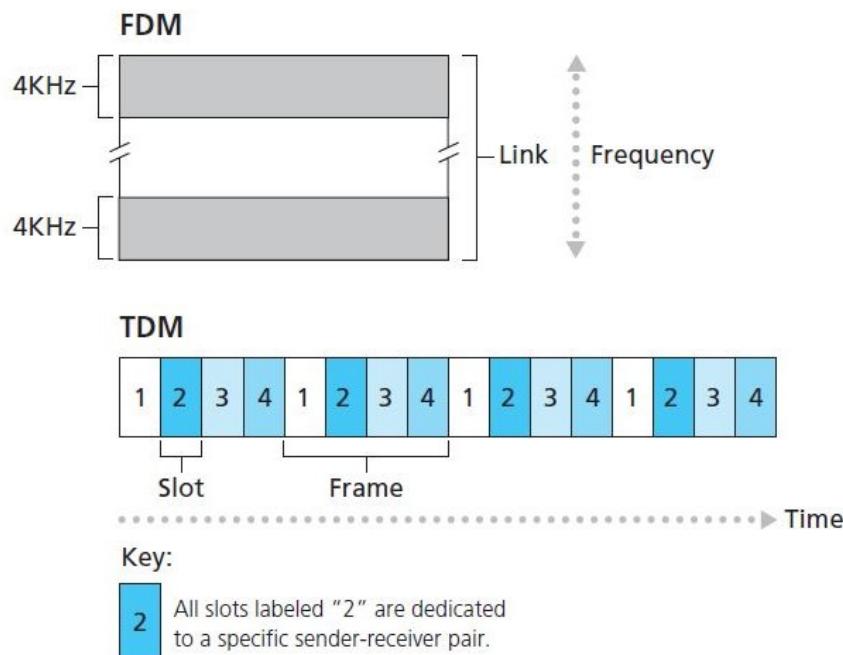


Figura 1.14 Con FDM, cada circuito continuamente recibe una fracción del ancho de banda. Con TDM, cada circuito recibe toda el ancho de banda periódicamente durante breves intervalos de tiempo (es decir, durante slots)

Los defensores de la conmutación de paquetes siempre han argumentado que la conmutación de circuitos es derrochadora, porque los circuitos dedicados están vacíos durante los **periodos de silencio**. Por ejemplo, cuando una persona en una llamada telefónica deja de hablar, los recursos de red vacíos (bandas de frecuencia o particiones en los enlaces de la ruta de conexión) no pueden ser utilizados por otras conexiones en curso. Para otro ejemplo de cómo se pueden infrautilizar estos recursos, consideremos un radiólogo que utiliza una red de conmutación de circuitos para acceder de forma remota a una serie de radiografías. El radiólogo prepara la conexión, solicita una imagen, contempla la imagen, y entonces solicita una nueva imagen. Los recursos de red son desperdiciados durante los períodos de contemplación del radiólogo. Los defensores de la conmutación de paquetes disfrutan también señalando que el establecimiento de circuitos y la reserva de ancho de banda terminal-a-terminal es complicado y requiere un software de señal complejo para coordinar la cooperación a lo largo de todo el recorrido terminal-a-terminal.

Antes de acabar nuestra discusión sobre conmutación de circuitos, vamos a trabajar sobre un ejemplo numérico que nos aclare cualquier duda sobre el tema. Vamos a considerar cuánto tiempo lleva enviar una archivo de 640,000 bits desde el Host A hasta el Host B a través de una red de conmutación de circuitos. Supongamos que todos los enlaces de la red son TDM con 24 particiones, y tienen una tasa de 1,536 Mbps. Supongamos también que se precisan 500 msec para establecer un circuito terminal-a-terminal antes de que el Host A empiece a transmitir el archivo. ¿Cuánto tiempo lleva enviar el archivo? Cada circuito tiene una velocidad de transmisión de $(1,536 \text{ Mbps})/24 = 64 \text{ kbps}$, por lo que se precisan $(640000 \text{ bits}/64 \text{ kbps}) = 10 \text{ segundos}$ para transmitir el archivo. A estos 10 segundos debemos añadir el tiempo de establecimiento del circuito, lo que da 10,5 segundos para enviar el archivo. Hay que tener en cuenta que el tiempo de transmisión es independiente del número de circuitos: el tiempo de transmisión sería de 10 segundos independientemente de si el circuito terminal-a-terminal pasara a través de un enlace o de 100 enlaces. (El retraso terminal-a-terminal incluye también el de propagación: véase la Sección 1.4)

Comutación de paquetes frente a conmutación de circuitos

Una vez descritas la conmutación de circuitos y la de paquetes, vamos a compararlas. Los detractores de la conmutación de paquetes han argumentado, a menudo, que la conmutación de paquetes no es adecuada para servicios de tiempo real (por ejemplo, llamadas de teléfono y de videoconferencia), por sus retardos variables e impredecibles terminal-a-terminal (debidos principalmente a los retardos de encolamiento variables e impredecibles). Los defensores de la conmutación de paquetes argumentan que (1) ofrece un reparto del ancho de banda mejor que la

comutación de circuitos y (2) es más sencilla, más eficiente y menos costosa de implementar que la comutación de circuitos. Hablando de forma general, la gente a la que no le gusta ocuparse de las reservas de hoteles prefiere comutación de paquetes en lugar de comutación de circuitos.

¿Por qué es más eficiente la comutación de paquetes? Consideremos un ejemplo sencillo. Supongamos que varios usuarios comparten un enlace de 1 Mbps. Supongamos también que cada usuario alterna entre períodos de actividad (en los que genera datos a una tasa constante de 100 kbps), y períodos de inactividad (en los que no genera datos). Supongamos además que el usuario está activo solo el 10 por ciento del tiempo (y está desocupado tomando un café el 90 por ciento restante). Con comutación de circuitos, se deben *reservar* 100 Kbps para *cada* usuario todo el tiempo. Por ejemplo, con TDM de comutación de circuitos, si una trama de un segundo se divide en 10 intervalos de tiempo de 100 msec cada uno, a continuación, cada usuario se asigna una ranura de tiempo por trama.

Por tanto, el enlace puede soportar solo 10 (= 1Mbps/100 kbps) usuarios simultáneamente. Con la comutación de paquetes, la probabilidad de que un usuario específico esté activo es de 0,1 (es decir, el 10 por ciento). Si hay 35 usuarios, la probabilidad de que haya 11 o más usuarios activos simultáneamente es de 0,0004 aproximadamente. Cuando hay 10 o menos usuarios activos simultáneamente (lo que ocurre con probabilidad 0,9996), la tasa de llegada de datos agregada es menor o igual que 1 Mbps, la tasa de salida del enlace. Por tanto, cuando hay 10 o menos usuarios activos, los paquetes de los usuarios fluyen a través del enlace esencialmente sin retraso, como en el caso de la comutación de circuitos. Cuando hay más de 10 usuarios activos simultáneamente, la tasa de llegada agregada de paquetes excede la capacidad de salida del enlace, y la cola de salida comenzará a crecer. (Continuará creciendo hasta que la tasa de entrada agregada caiga por debajo de 1 Mbps, en cuyo caso la cola comenzará a disminuir en longitud). Dado que la probabilidad de tener más de 10 usuarios activos simultáneamente es minúscula en este ejemplo, la comutación de paquetes casi siempre tiene las mismas prestaciones de retraso que la comutación de circuitos, *pero lo consigue permitiendo un número de 3 usuarios tres veces mayor*.

Aunque tanto la comutación de paquetes como la de circuitos son prevalentes en las redes de telecomunicaciones de hoy, la tendencia va sin duda en la dirección de la comutación de paquetes. Incluso muchas de las redes telefónicas de comutación de circuitos están migrando lentamente hacia la comutación de paquetes. En particular, las redes de telefonía utilizan con frecuencia la comutación de paquetes para la costosa porción de una llamada telefónica al extranjero.

1.4 Demoras, Pérdidas y Rendimiento en la Red

En la sección 1.1 dijimos que Internet puede ser visto como una infraestructura que ofrece servicios a las aplicaciones distribuidas que se ejecutan en los sistemas finales. Idealmente, nos gustaría que los servicios de Internet fueran capaces de mover todos los datos que queremos entre dos sistemas finales, de forma instantánea, sin ninguna pérdida de datos. Por desgracia, esto es un noble objetivo, que es inalcanzable en la realidad. En cambio, las redes de computadoras limitan necesariamente el rendimiento (la cantidad de datos por segundo que pueden ser transferidos) entre sistemas finales, introducen retardos entre sistemas finales, y de hecho pueden perder paquetes. Por un lado, es lamentable que las leyes físicas de la realidad introduzcan retrasos y pérdidas, así como restricciones en el rendimiento. Por otro lado, dado que las redes de computadoras tienen estos problemas, hay muchos temas fascinantes al respecto de cómo hacer frente a los problemas (temas más que suficientes para llenar un curso sobre redes de computadoras y para motivar a miles de tesis de doctorado!) En esta sección, vamos a empezar a analizar y cuantificar el retraso, la pérdida, y el rendimiento en redes de computadoras.

1.4.1 Visión General de la Demoras en las Redes de Comutación de Paquetes

Recordemos que un paquete comienza en un host (el origen), pasa a través de una serie de routers, y finaliza su viaje en otro host (el destino). Mientras un paquete va desde un nodo (host o router) hasta el nodo subsiguiente (host o router) a lo largo de su recorrido, el paquete sufre diferentes tipos de retardos en *cada* nodo a lo largo del recorrido. Los más

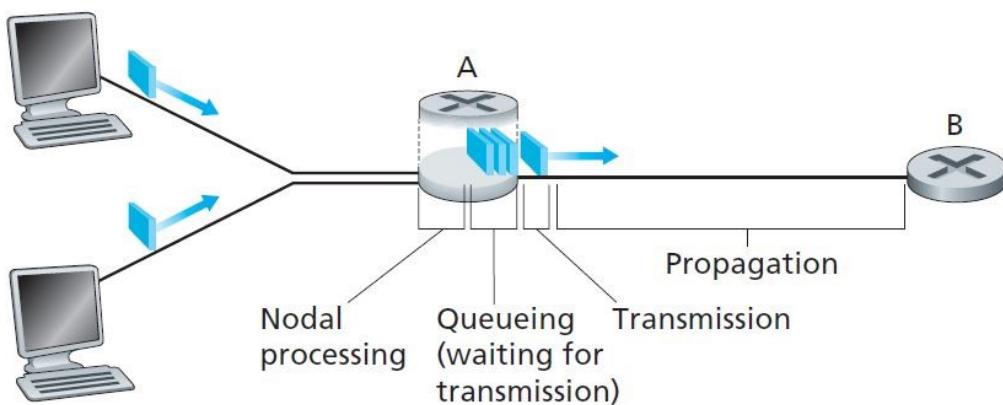


Figura 1.16 El Retraso Nodal en el Router A

importantes de estos retardos son: el **retardo de proceso nodal**, el **retardo de cola**, el **retardo de transmisión**, y el **retardo de propagación**; estos retardos se acumulan conjuntamente para dar un **retardo nodal total**. El rendimiento de muchas aplicaciones, (tales como la búsqueda de Internet, navegación web, correo electrónico, mapas, mensajería instantánea y voz sobre IP) se ven afectados en gran medida por los retrasos en la red. Con el fin de adquirir una comprensión profunda de la conmutación de paquetes y redes informáticas, tenemos que comprender la naturaleza y la importancia de estos retardos.

Tipos de Retardo

Vamos a explorar estos retardos en el contexto de la Figura 1.18. Como una parte de su ruta terminal-a-terminal entre la fuente y el destino, un paquete se envía desde el nodo anterior a través del router A al router B. Nuestro objetivo es caracterizar el retardo nodal en el router A. Hay que tener en cuenta que el router A tiene un enlace externo hacia el router B. Este enlace está precedido por una cola (conocida como búfer). Cuando el paquete llega al router A desde el nodo anterior, el router A examina la cabecera del paquete para determinar el enlace exterior para el paquete, y dirige entonces el paquete hacia el enlace. En este ejemplo, el enlace exterior para el paquete es uno que lo dirige hacia el router B. Un paquete se puede transmitir en un enlace si no se está transmitiendo en ese momento otro paquete en el enlace y si no hay otros paquetes precedentes en la cola; si el enlace está ocupado o hay otros paquetes encolados por el enlace, el paquete nuevo que llega se añadirá a la cola.

Retardo de procesamiento

El tiempo requerido para examinar la cabecera del paquete y determinar hacia donde debe dirigirse es parte del **retardo de procesamiento**. El retardo de procesamiento puede incluir también otros factores, como el tiempo necesario para comprobar los errores a nivel de bit que han ocurrido durante la transmisión de los bits del paquete desde el nodo anterior al router A. Los retardos de procesamiento en los routers de elevada velocidad están típicamente en el orden de los microsegundos o menos. Después del procesamiento nodal, el router dirige el paquete a la cola que precede al enlace hacia el router B. (En el Capítulo 4 estudiaremos los detalles de cómo funciona un water).

Retardo de cola

En la cola, el paquete experimenta un **retardo de cola** mientras espera para ser transmitido en el enlace. El retardo de cola de un paquete específico dependerá del número de paquetes que hayan llegado anteriormente y que están encolados y esperando para la transmisión sobre el enlace. El retardo de un paquete dado puede variar significativamente de un paquete a otro. Si la cola está vacía y ningún otro paquete se está transmitiendo actualmente, entonces el retardo de cola de nuestro paquete es cero. Por otro lado, si el tráfico es pesado y otros muchos paquetes están esperando para ser transmitidos, el retardo de cola será largo. Inmediatamente veremos que el número de paquetes que un paquete que llega podría esperar encontrar a su llegada es función de la intensidad y naturaleza del tráfico que llega a la cola. Los retardos de cola pueden ser en la práctica del orden de microsegundos a milisegundos.

Retardo de transmisión

Suponiendo que los paquetes se transmiten de la forma primero en llegar primero en ser servido, como es normal en las redes de commutación de paquetes, nuestro paquete sólo puede ser transmitido después de que todos los paquetes que han llegado antes hayan sido transmitidos. Indicamos la longitud del paquete por L bits, y a la tasa de transmisión del enlace desde el router A al router B por R bits/seg. La tasa R se determina por la tasa de transmisión del enlace al router B. Por ejemplo, para un enlace Ethernet de 10 Mbps, la tasa es $R = 100$ Mbps; para un enlace Ethernet de 100 Mbps, la tasa es $R = 100$ Mbps. El **retardo de transmisión** (también llamado retardo de almacenar y reenviar, como se discutió en la Sección 1.3) es L/R . Ésta es la cantidad de tiempo que se requiere para empujar (es decir, transmitir) todos los bits del paquete en el enlace. Los retardos de transmisión típicos son, en la práctica, del orden de los milisegundos.

Retardo de propagación

Una vez que un bit es empujado al enlace, necesita propagarse al router B. El tiempo necesario para propagarse desde el enlace hasta el router B es el **retardo de propagación**. El bit se propaga a la velocidad de propagación del enlace. La velocidad de propagación depende del medio del enlace (es decir, fibra óptica, cable de cobre de par trenzado, etc.) y está en el rango de:

$$2 \times 10^3 \text{ metros/seg a } 3 \times 10^8 \text{ metros/seg}$$

que es igual o un poco menos que la velocidad de la luz. El retardo de propagación es la distancia entre dos routers dividida por la velocidad de propagación. Es decir, el retardo de propagación es 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 propaga hasta el nodo B, este bit y todos los bits precedentes del paquete son almacenados en el router B. El proceso total continúa entonces, realizando ahora el envío de paquetes el router B. En redes de área amplia, los retardos de propagación son del orden de milisegundos.

Comparación de los retardos de transmisión y propagación

Los principiantes en el campo de las redes de computadoras tienen, a veces, dificultad para comprender la diferencia entre retardo de transmisión y de propagación. La diferencia es sutil pero importante. El retardo de transmisión es la cantidad de tiempo que precisa el router para enviar el paquete; es una función de la longitud del paquete y de la tasa de transmisión del enlace, pero no tiene nada que ver con la distancia entre los dos routers. El retardo de propagación, por el contrario, es el tiempo que precisa un bit para propagarse desde un router hasta el siguiente; es una función de la distancia entre los dos routers, pero no tiene nada que ver con la longitud del paquete o la tasa de transmisión del enlace.

Una analogía podría clarificar las nociones de retardo de transmisión y propagación. Consideremos una autopista que tiene un peaje cada 100 kilómetros. Se puede considerar que los segmentos de autopista entre peajes son enlaces y los peajes routers. Supongamos que un coche viaja (es decir, se propaga) por la autopista a una velocidad de 100 km/hora (es decir, cuando un coche deja un peaje, acelera instantáneamente hasta 100 km/hora) y mantiene la velocidad entre peajes). Supongamos que hay una caravana de 10 coches viajando juntos y que los 10 coches van uno detrás del otro en un orden determinado. Puede considerar cada coche como un bit, y la caravana como un paquete. Supongamos también que cada uno de los peajes sirve (es decir, transmite) un coche a una tasa de un coche cada 12 segundos, y que es ya de noche, de forma que los coches de la caravana son los únicos coches en la autopista. Finalmente, supongamos que cuando el primer coche de la caravana llega a un peaje, espera a la entrada hasta que los otros 9 coches han llegado y se alinean detrás de él. (Por tanto, la caravana entera debe almacenarse en el peaje antes de que pueda empezar a ser difundida). El tiempo preciso para que el peaje empuje la caravana entera en la autopista es $(10 \text{ coches})/(5 \text{ coches/minuto}) = 2 \text{ minutos}$. Este tiempo es análogo al retardo de transmisión en un router. El tiempo requerido por un coche para viajar desde la salida desde un peaje hasta el siguiente es de $100 \text{ km}/100 \text{ km/hora} = 1 \text{ hora}$. Este tiempo es análogo al retardo de propagación. Por tanto, el tiempo desde que se almacena la caravana delante de un peaje hasta que la caravana es almacenada frente al siguiente es la suma del retardo de transmisión y el retardo de propagación (en este ejemplo, 62 minutos).

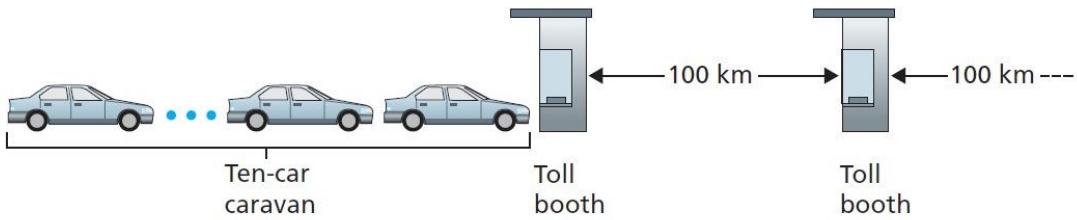


Figura 1.17 La Analogía de Caravana

Vamos a explorar esta analogía un poco más. ¿Qué ocurriría si el tiempo de servicio del peaje para una caravana fuera mayor que el tiempo que necesita un coche entre dos peajes? Por ejemplo, supongamos ahora que los coches viajan a una velocidad de 1.000 km/hora y el peaje sirve a los coches a la velocidad de un coche por minuto. Entonces, el retardo de viajar entre dos peajes es 6 minutos, y el tiempo para servir una caravana es 10 minutos. En este caso, los primeros coches de la caravana llegarán al segundo peaje antes de que los últimos coches de la caravana abandonen el 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 muchos de los restantes bits están todavía esperando a ser transmitidos por el router precedente.

Si denotamos como d_{proc} , d_{cola} , d_{trans} y d_{prop} a los retardos de procesamiento, de cola, de transmisión y de propagación, entonces el retardo nodal está dado por:

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

La contribución de esos componentes de retardo puede variar significativamente. Por ejemplo, d_{prop} puede ser despreciable (por ejemplo, un par de microsegundos) para un enlace que conecta dos routers en el mismo campus universitario; sin embargo, d_{prop} es de cientos de milisegundos para dos routers conectados por un enlace de satélite geoestacionario, y puede ser el término dominante en d_{nodal} . De forma similar, d_{trans} puede ser despreciable o significativo. Su contribución es normalmente despreciable para velocidades de transmisión de 10 Mbps y más elevadas (por ejemplo, para LAN); sin embargo, puede ser de cientos de milisegundos para grandes paquetes de Internet enviados sobre enlaces de modem telefónicos de baja velocidad. El retardo de proceso, d_{proc} , es a menudo despreciable; sin embargo, incide fuertemente en el máximo *throughput* de un router, que es la velocidad máxima a la que un router puede difundir paquetes.

1.4.2 Retardo de cola y pérdida de paquetes

El componente más complicado e interesante del retardo nodal es el retardo de cola. De hecho, el retardo de cola es tan importante e interesante en las redes de computadoras que se han escrito miles de trabajos y numerosos libros sobre él [Bertsekas 1991; Daigle 1991; Kleinrock 1975, 1976; Ross 1995]. Aquí ofrecemos una discusión de alto nivel, intuitiva, del retardo de cola; el lector más curioso puede querer ojear algunos libros (o incluso eventualmente escribir una tesis doctoral sobre el tema). Distinto de los otros tres retardos (es decir, d_{proc} , d_{trans} y d_{prop}), el retardo de cola puede variar de un paquete a otro. Por ejemplo, si 10 paquetes llegan a una cola vacía al mismo tiempo, el primer paquete transmitido no sufrirá retardo de cola, mientras que el último sufrirá un retardo de cola relativamente grande (mientras espera a que se transmitan los otros nueve paquetes). Por tanto, cuando se caracteriza el retardo de cola, normalmente se utilizan medidas estadísticas, como el retardo promedio, la varianza del mismo, y la probabilidad de que el retardo de cola supere un valor determinado.

¿Cuándo es grande y cuándo es insignificante el retardo de cola? La respuesta a esta pregunta depende en gran medida de la tasa a la que llegue el tráfico a la cola, de la velocidad de transmisión del enlace, y de la naturaleza del tráfico que llega (es decir, de si el tráfico llega periódicamente o llega a rachas). Para profundizar un poco más, sea a la velocidad media a la que llegan los paquetes a la cola (a está en unidades de paquete/seg). Recordemos que R es la velocidad de transmisión, es decir, la velocidad (en bits/seg) a la que los bits son sacados de la cola. Suponemos también, para simplificar, que todos los paquetes constan de L bits. Entonces, la velocidad media a la que llegan los bits a la cola es La bits/seg. Por último, supongamos que la cola es muy grande, por lo que puede mantener un número infinito de bits. La relación La/R , llamada **intensidad de tráfico**, juega, a menudo, un papel

importante en la estimación de la extensión del retardo de cola. Si $La/R > 1$, entonces la velocidad media a la que los bits llegan a la cola supera la velocidad a la que 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 ingeniería de tráfico es: *Diseña tu sistema de forma que la intensidad del tráfico no sea mayor que 1.*

Consideremos ahora el caso $La/R \leq 1$. Aquí, la naturaleza del tráfico de llegada impacta en el retardo de cola. Por ejemplo, si los paquetes llegan periódicamente (es decir, si un paquete llega cada L/R segundos), entonces cada paquete llegará a una cola vacía, y no habrá retardo de cola. Por otro lado, si los paquetes llegan a rachas pero periódicamente, puede haber un retardo de cola significativo. Por ejemplo, supongamos que N paquetes llegan simultáneamente cada $(L/R)N$ segundos. Entonces, el primer paquete transmitido tiene un retardo de cola de L/R segundos; y de forma más general, el n -ésimo paquete transmitido tiene un retardo de cola de $(n - 1)L/R$ segundos. Dejamos como un ejercicio para el lector calcular el retardo de cola promedio en este ejemplo.

Los dos ejemplos de llegadas periódicas descritos anteriormente son un poco académicos. Por lo general, el proceso de llegada a una cola es *aleatorio*; es decir no sigue ningún patrón, y los paquetes están espaciados entre sí por cantidades aleatorias de tiempo. En este caso más realista, la cantidad La/R no es suficiente normalmente para caracterizar totalmente los estadísticos del retardo. En particular, si la intensidad de tráfico está próxima a cero, entonces las llegadas de paquetes son escasas y lejanas entre ellas, y es improbable que un paquete que llega se encuentre a otro en la cola. Aquí, el retardo de cola promedio estará próximo a cero. Por otro lado, cuando la intensidad de tráfico esté próxima a 1, habrá intervalos de tiempo en los que la tasa de llegada excederá la capacidad de transmisión (debido a la acumulación de llegadas), y se formará una cola. A medida que la intensidad de tráfico se aproxime a 1, la longitud media de la cola se hará cada vez más grande. La dependencia cualitativa del retardo medio de cola respecto a la intensidad de tráfico se muestra en la Figura 1.18.

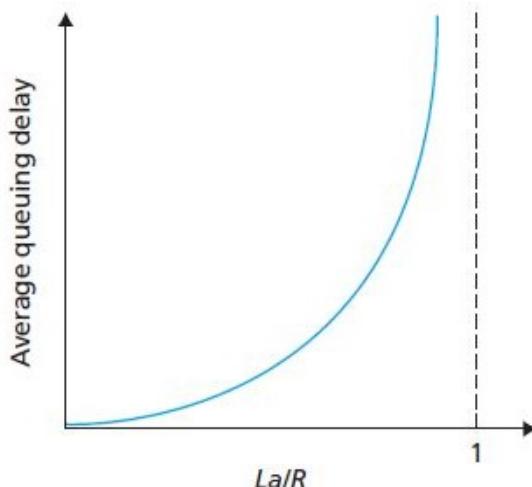


Figura 5.41 Traducción

Un aspecto importante de la Figura 1.18 es el hecho de que cuando la intensidad de tráfico se approxima a 1, el retardo medio de cola aumenta rápidamente. Un pequeño porcentaje en la intensidad producirá un incremento porcentual mucho más grande en el retardo. Quizás hayas experimentado este fenómeno en la autopista. Si conduces regularmente en una ruta que está congestionada normalmente, ese hecho significa que la intensidad de tráfico está muy próxima a 1. Si algún evento hace que se produzca una cantidad de tráfico ligeramente mayor que la habitual, los retardos que experimentará serán muy grandes.

Pérdida de paquetes

En nuestras discusiones anteriores, hemos supuesto que la cola es capaz de mantener un número infinito de paquetes. En realidad, una cola que precede a un enlace tiene una capacidad limitada, aunque la capacidad de la cola depende fundamentalmente del diseño y coste del switch. Como la capacidad de la cola es finita, los retardos de paquetes no se aproximan realmente a infinito cuando la intensidad de tráfico se approxima a 1. En lugar de esto,

un paquete puede encontrar una cola llena cuando llega. Si no hay lugar para almacenar el paquete, un router lo **abandonará**, es decir, el paquete se **perderá**.

Desde un punto de vista de sistema terminal, esto parecerá como un paquete introducido en el núcleo de la red que no ha emergido nunca de la misma en su destino. La fracción de los paquetes perdidos aumenta a medida que aumenta la intensidad de tráfico. Por tanto, las prestaciones de un nodo se miden con frecuencia no solo en función del retardo, sino también de la probabilidad de pérdida de paquetes. Como se discutirá en capítulos posteriores, un paquete perdido puede ser retransmitido en un contexto terminal-a-terminal, bien por la aplicación, bien por el protocolo de la capa de transporte.

1.4.3 Retardo terminal-a-terminal

Hasta este momento, nuestra discusión se ha enfocado en el retardo de nodo, es decir, el retardo en un único router. Vamos a finalizar nuestra discusión considerando brevemente el retardo desde la fuente hasta el destino. Para conseguir una base sobre este concepto, supongamos que hay $N - 1$ routers entre la máquina origen y la máquina destino. Supongamos también que la red no está congestionada (es decir, que los retardos de cola son despreciables), que el retardo de proceso tu cada router y en la máquina origen es d_{proc} , que la velocidad de transmisión hacia fuera de cada router y de la máquina origen es de R bits/seg, y que la velocidad de propagación de cada enlace es d_{prop} . Los retardos de nodo se acumulan, y dan un retardo terminal a terminal de

$$d_{end-end} = N(d_{proc} + d_{trans} + d_{prop})$$

donde, una vez más $d_{trans} = L/R$, donde L es el tamaño del paquete. Dejarnos al lector generalizar esta fórmula para el caso de retardos heterogéneos en los nodos y en presencia de un retardo promedio de cola en cada nodo.

Traceroute

Para tener un primer conocimiento sobre el retardo en una red de computadoras, podemos utilizar el programa de diagnóstico Traceroute. Traceroute es un programa sencillo que puede correr en cualquier máquina Internet. Cuando el usuario especifica un nombre de máquina destino, el programa en la maquina origen envía múltiples paquetes especiales hacia el destino. A medida que esos paquetes siguen su camino hacia el destino, pasan a través de una serie de routers. Cuando un router recibe uno de estos paquetes especiales, envía un mensaje corto al origen. Este mensaje contiene el nombre y la dirección del router.

De forma más concreta, supongamos que hay $N - 1$ routers entre el origen y el destino. Entonces, el origen enviará N paquetes especiales a la red, cada uno direccionado al destino final. Hay N paquetes especiales, marcados de 1 a N , con el primero marcado con 1 y el último con N . Cuando el router n -ésimo recibe el n -ésimo paquete, señalado como n , el router destruye el paquete y envia un mensaje hacia el origen. Y cuando la máquina de destino recibe el N -ésimo paquete, el destino lo destruye también y devuele, de nuevo, un mensaje de vuelta al origen.

El origen registra el tiempo que transcurre desde que envía un paquete hasta que recibe el correspondiente mensaje de vuelta; también registra el nombre y dirección del router (o máquina 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, el origen puede determinar los retardos de ida y vuelta para todos los routers que intervienen. Traceroute en realidad repite el experimento descrito tres veces, de modo que el origen en realidad envía $3 \times N$ paquetes hacia el destino. RFC 1393 describe Traceroute en detalle.

Aquí hay un ejemplo de la salida del programa Traceroute, donde la ruta fue siendo trazada desde la máquina origen *eniac.seas.upenn.edu* (en la Universidad de Pennsylvania) para *diane.ibp.fr* (en la Universidad de Paris VI). La salida tiene seis columnas: la primera columna es el valor n descrito anteriormente, es decir, el número del router a lo largo de la ruta; la segunda columna es el nombre del router; la tercera columna es la dirección del router (de la forma xxx.xxx.xxx.xxx); las tres columnas ultimas son los retardos de ida y vuelta para los tres experimentos. Si el origen recibe menos de tres mensajes desde cualquier router dado (debido a la pérdida de paquete en la red), Traceroute coloca un asterisco justo después del número de router, e informa de menos de tres veces ida y vuelta para el 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 po0-0.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

```

En la traza anterior hay 20 routers entre el origen y el destino. La mayoría 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`. Mirando de los datos proporcionados por el mismo muten vemos que en la primera de las tres pruebas del retardo de ida y vuelta entre el origen y el router fue de 1.03 mseg. Los retardos de ida y vuelta para las dos pruebas consecuentes fueron de 0.48 y 0.45 mseg. Estos retornos de ida y vuelta incluyen todos los retardos discutidos anteriormente, como los retardos de transmisión, de propagación, de proceso del router, y de cola. Dado que el retardo de cola varía con el tiempo, el retardo de ida y vuelta del paquete n enviado al router n puede ser, en realidad, más largo que el retardo de ida y vuelta del paquete $n + 1$ enviado al router $n + 1$. De hecho, observamos este fenómeno en el ejemplo de arriba: las demoras al Router 6 son más grandes que las demoras al Router 7

¿Queres probar vos mismo Traceroute? Es *muy* recomendado que visite <http://www.traceroute.org>, que proporciona una interfaz web a una lista intensiva de orígenes para trazar rutas. Vos elegís un destino y proporciona el nombre de máquina para cualquier destino. El programa Traceroute realiza todo el trabajo.

Sistemas Finales, Aplicaciones y otros retrasos

Además de los retardos de procesamiento, transmisión y de propagación, pueden haber retrasos significativos adicionales en los sistemas finales. Por ejemplo, un sistema de extremo queriendo transmitir un paquete en un medio compartido (por ejemplo, como en un escenario WiFi o módem de cable) puede retrasar *a propósito* su transmisión como parte de su protocolo para compartir el medio con otros sistemas finales; vamos a considerar este tipo de protocolos en detalle en el capítulo 5. Otro retraso importante es el retardo de empaquetamiento de medios de comunicación, que está presente en Voz-IP (VoIP). En VoIP, el lado emisor debe llenar primero un paquete con voz digitalizada codificada antes de pasar el paquete a Internet. Esta vez para llenar un paquete - llamado el retardo de empaquetamiento- puede ser significativo y puede afectar al usuario la calidad percibida de una llamada VoIP. Esta cuestión se analizará más en un problema de tarea al final de este capítulo.

1.4.4 El rendimiento en redes de computadoras

Además del retraso y la pérdida de paquetes, otra medida crítica de la performance en las ciencias de computación es el rendimiento de extremo a extremo. Para definir el rendimiento, consideremos la transferencia de un archivo grande del host A al host B a través de una red informática. Esta transferencia puede ser, por ejemplo, un gran videoclip de un punto a otro en un sistema de intercambio de archivos P2P. El **rendimiento instantáneo** en cualquier instante de tiempo es la tasa (en bits / segundo) a la que el host B está recibiendo el archivo. (Muchas aplicaciones, incluyendo muchos sistemas de intercambio de archivos P2P, muestran el caudal instantáneo durante las descargas en la interfaz de usuario, tal vez hayas observado esto antes!) Si el archivo se compone de bits F y la transferencia lleva T segundos para que el Host B reciba los F bits, entonces el **rendimiento promedio** de la transferencia de archivos es F/T bits/seg. Para algunas aplicaciones, tales como telefonía por Internet, es deseable tener un bajo retardo y un rendimiento instantáneo constantemente por encima de un umbral (por ejemplo, más de 24 kbps para algunas aplicaciones de telefonía por Internet y más de 256 kbps para algunas aplicaciones de vídeo en tiempo real). Para otras aplicaciones, incluyendo los que implican la transferencia de archivos, el retardo no es crítico, pero es deseable tener el mayor rendimiento posible.

Para obtener una mayor comprensión del concepto importante de rendimiento, vamos a considerar algunos ejemplos. La figura 1.19 (a) muestra dos sistemas finales, un servidor y un cliente, conectados por dos enlaces de comunicación y un router. Considere el rendimiento para una transferencia de archivos desde el servidor al cliente. Sea R_s denotando la tasa de la conexión entre el servidor y el router; y R_c denota la tasa de la conexión entre

el router y el cliente. Supongamos que los únicos bits enviados en toda la red son aquellos desde el servidor al cliente. Ahora nos preguntamos, en este escenario ideal, ¿cuál es el rendimiento servidor-cliente? Para responder a esta pregunta, podemos pensar los bits como un *flujo* y los enlaces de comunicación como *tuberías*. Claramente, el servidor no puede bombear los bits a través de su enlace a un ritmo más rápido que R_s bps; y el router no puede enviar bits a una velocidad mayor que R_c bps. Si $R_s < R_c$, entonces los bits bombeados por el servidor "fluirán" derecho a través del router y llegarán al cliente a una velocidad de R_s bps, dando un rendimiento de R_s bps. Si, por el contrario, $R_c < R_s$, a continuación, el router no será capaz de enviar bits con la misma rapidez con que las recibe. En este caso, los bits sólo dejarán el router a una velocidad R_c , dando un rendimiento de extremo a extremo de R_c . (Tenga en cuenta también que si los bits continúan llegando al router a una tasa de R_s , y seguir dejando el router a R_c , el remanente de bits en el router esperando ser transmitidos al cliente va a crecer y crecer, -una situación indeseable!) Por lo tanto, para esta simple red de dos enlace, el rendimiento es $\min(R_c, R_s)$, es decir, es la velocidad de transmisión del **cuello de botella del enlace**. Una vez determinado el rendimiento, ahora podemos aproximar el tiempo que se tarda en transferir un archivo grande de F bits del servidor al cliente como $F/\min(R_c, R_s)$. Para un ejemplo específico, supongamos que está descargando un archivo MP3 de $F = 32$ millones de bits, el servidor tiene una velocidad de transmisión de $R_s = 2$ Mbps, y tiene un enlace de acceso de $R_c = 1$ Mbps. El tiempo necesario para transferir el archivo es entonces 32 segundos. Por supuesto, estas expresiones para el rendimiento y el tiempo de transferencia son sólo aproximaciones, ya que no tienen en cuenta los retrasos store-and-forward y procesamiento, así como las cuestiones de protocolo.

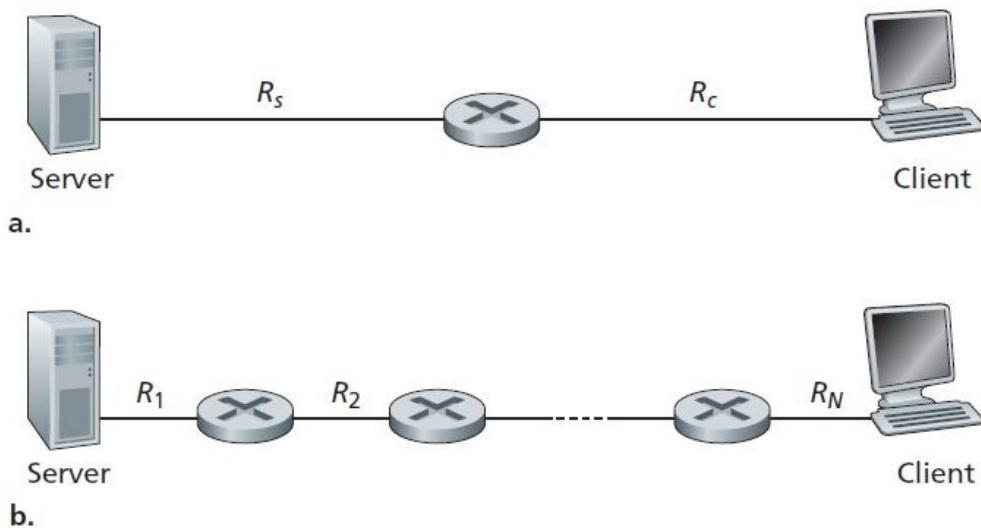


Figura 1.19 Rendimiento para una transferencia de archivos del servidor al cliente

La figura 1.19 (b) muestra ahora una red con N vínculos entre el servidor y el cliente, con las velocidades de transmisión de los enlaces N siendo R_1, R_2, \dots, R_N . Aplicando el mismo análisis que para la red de dos articulaciones, nos encontramos con que el rendimiento de transferencia de archivos desde el servidor al cliente es $\min(R_1, R_2, \dots, R_N)$, que es una vez más la velocidad de transmisión del enlace de cuello de botella a lo largo de la trayectoria entre el servidor y el cliente.

Consideremos ahora otro ejemplo motivado por la Internet de hoy. La figura 1.20(a) muestra dos sistemas finales, un servidor y un cliente, conectados a una red informática. Considere el rendimiento para una transferencia de archivos desde el servidor al cliente. El servidor está conectado a la red con un enlace de acceso de de tasa R_s y el cliente está conectado a la red con un enlace de acceso de la tasa de R_c . Supongamos ahora que todos los enlaces en el núcleo de la red de comunicación tienen velocidades de transmisión muy altas, muy superior a R_s y R_c . De hecho, hoy en día, el núcleo de Internet sobre provisionado con enlaces de alta velocidad que experimentan poco congestión. Supóngase también que los únicos bits enviados en toda la red son aquellos desde el servidor al cliente. Debido a que el núcleo de la red es como un tubo ancho en este ejemplo, la velocidad a la que los bits pueden fluir

desde el origen al destino es de nuevo el mínimo entre R_s y R_c , es decir, rendimiento = $\min R_s, R_c$. Por lo tanto, el factor limitante para el rendimiento en la Internet de hoy es típicamente la red de acceso.

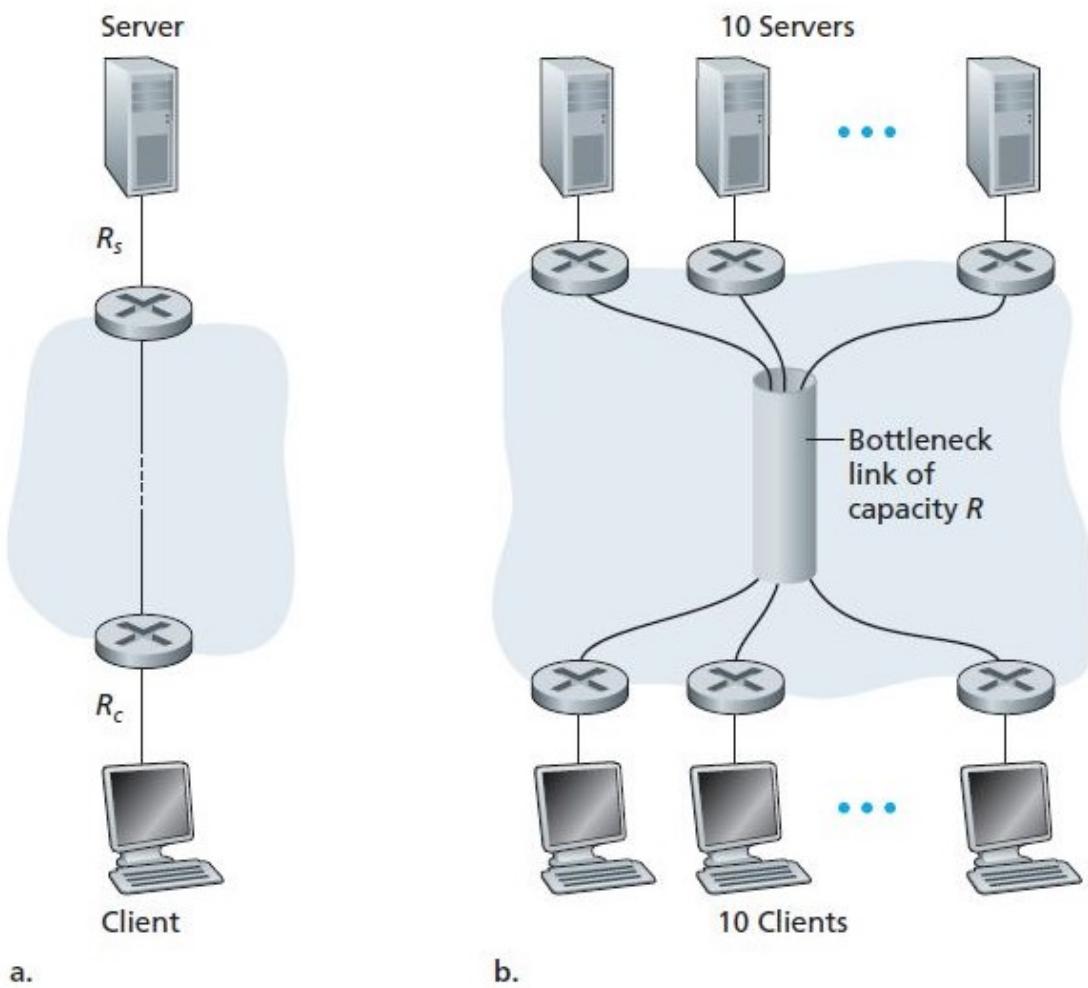


Figura 1.20 Rendimiento Extremo a Extremo: (a) El Cliente descarga un archivo desde el Servidor; (b) 10 clientes descargando con 10 servidores

Para un ejemplo final, considere la figura 1.20(b) en el que hay 10 servidores y 10 clientes conectados al núcleo de la red de computadoras. En este ejemplo, hay 10 descargas simultáneas teniendo lugar, con la participación de 10 parejas cliente-servidor. Supongamos que estos 10 descargas son el único tráfico en la red en el momento actual. Como se muestra en la figura, hay un enlace en el núcleo que está atravesado por los 10 descargas. Denotemos R a la velocidad de transmisión de este enlace R . Vamos a suponer que todos los enlaces de acceso al servidor tienen los mismos tasa R_s , todos los enlaces de acceso del cliente tienen la misma velocidad R_c , y las velocidades de transmisión de todos los enlaces en el núcleo -salvo el enlace común de tasa R - son mucho mayores que R_s , R_c y R . Ahora nos preguntamos, ¿cuáles son los rendimientos de las descargas? Está claro que si la tasa del enlace común, R , es grande, digamos cien veces más grande que R_s y R_c -entonces el rendimiento por cada descarga será una vez más $\min R_s, R_c$. Pero ¿y si la tasa del vínculo común es del mismo orden que R_s y R_c ? ¿Cómo será el rendimiento en este caso? Vamos a echar un vistazo a un ejemplo específico. Supongamos que $R_s = 2$ Mbps, $R_c = 1$ Mbps, $R = 5$ Mbps, y el enlace común divide su velocidad de transmisión en partes iguales entre los 10 descargas. Entonces, el cuello de botella para cada descarga ya no está en la red de acceso, en cambio, ahora es el enlace compartido en el núcleo, que sólo provee a cada descarga 500 kbps de rendimiento. Así, el rendimiento de extremo a extremo para cada descarga se ha reducido a 500 kbps.

Los ejemplos en la Figura 1.19 y la Figura 1.20(a) muestran que el rendimiento depende de la velocidad de transmisión de los enlaces sobre los que fluyen los datos. Vimos que cuando no hay ningún otro tráfico interveniendo, el

rendimiento puede simplemente ser aproximado como la velocidad de transmisión mínima a lo largo de la trayectoria entre el origen y el destino. El ejemplo en la Figura 1.20(b) muestra que de manera más general el rendimiento no sólo depende de las velocidades de transmisión de los enlaces a lo largo de la ruta, sino también en el tráfico interveniente. En particular, un enlace con una alta tasa de transmisión puede no obstante ser el enlace cuello de botella para una transferencia de archivos si muchos otros flujos de datos también están pasando a través de ese enlace. Vamos a examinar el rendimiento en redes de computadoras más estrechamente en los problemas de tarea y en los capítulos subsiguientes.

1.5 Capas de protocolo y sus modelos de servicio

De toda nuestra discusión, está claro que Internet es un sistema *extremadamente* complicado. Hemos visto que hay muchas partes en Internet: numerosas aplicaciones y protocolos, varios tipos de sistemas terminales y conexiones entre ellos, y varios tipos de medios de enlace. Dada su enorme complejidad, ¿hay una esperanza de organizar una arquitectura de red o al menos nuestra discusión sobre la misma? Afortunadamente la respuesta es sí.

1.5.1 Arquitectura en capas

Antes de intentar organizar nuestras ideas sobre la arquitectura de Internet, busquemos una analogía humana. Actualmente, tratamos con sistemas complejos todo el tiempo en nuestra vida diaria. Imagine que alguien le pidiera describir, por ejemplo, el sistema de una compañía aérea. ¿Cómo encontraría la estructura para definir este sistema complejo que tiene agentes de venta de boletos, facturadores de equipaje, personal de puertas, pilotos, aviones, control de tráfico aéreo y un sistema mundial de rutas aéreas? Una forma de describir este sistema podría ser describir la serie de acciones que realiza (o que otros realizan por usted) cuando vuela en una compañía aérea. Compra su boleto, factura su equipaje, va a la puerta de embarque, y finalmente toma el avión. El avión despegue y es dirigido hacia su destino. Después el avión aterriza, usted desembarca por la puerta y reclama su equipaje. Si el viaje fue malo, reclama al vendedor en relación sobre el vuelo (no supone ningún esfuerzo). Este escenario se muestra en la Figura 1.21.

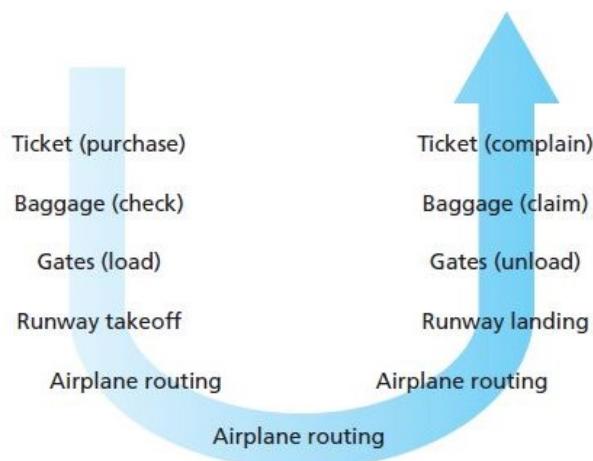


Figura 1.21 Tomando un vuelo: acciones

Ya se pueden apreciar aquí algunas analogías con las redes de computadoras. Usted es llevado desde el origen al destino por una línea aérea, del mismo modo que un paquete se envía desde la máquina origen hasta la máquina destino a través de Internet. Pero ésta no es la analogía que pretendemos resaltar. Estamos buscando alguna *estructura* en la Figura 1.21. Mirando a la Figura 1.21, nos damos cuenta de que existe una función de venta de boletos en cada terminal; también existe una función de equipaje para los viajeros con boleto, y una función puerta para los pasajeros con billete y que ya han facturado su equipaje. Para los pasajeros que han atravesado la puerta (es decir, los que tienen billete, han facturado su equipaje y han pasado por la pueria), hay una función de despegue y aterrizaje, y mientras se está en vuelo, hay una función de rutado. Esto sugiere que podemos mirar la funcionalidad de la Figura 1.20 de una manera *horizontal*, como se ve en la Figura 1.22.

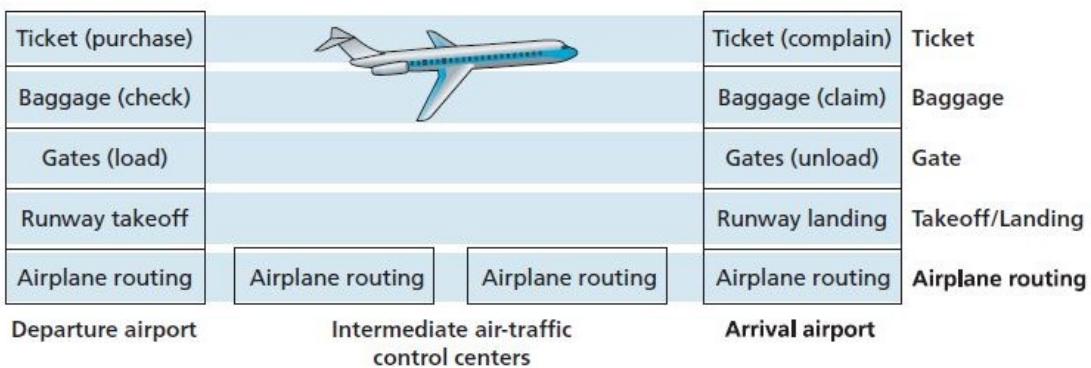


Figura 1.22 Estratificación horizontal de la funcionalidad de la línea aérea.

La Figura 1.22 ha dividido la funcionalidad de la línea aérea en capas, proporcionando un marco en el que podemos discutir el viaje en una línea aérea. Por ejemplo, cuando discutimos la funcionalidad de la puerta, sabemos que estamos discutiendo la funcionalidad que se encuentra debajo de la organización de equipaje y encima del despegue y aterrizaje. Notemos que cada capa, combinada con las capas inferiores a la misma, implementan alguna funcionalidad, algún *servicio*. En la capa de venta de boletos y en las que están por debajo, se realiza la transferencia de una persona de mostrador a mostrador. En la capa de equipaje y en las inferiores se ejecuta la transferencia de una persona y su equipaje desde su facturación hasta su recogida. Observe que la capa de equipaje proporciona este servicio sólo para una persona con billete. En la capa de la puerta, se realiza la transferencia de una persona y su equipaje desde la puerta de salida hasta la de llegada. En la capa de despegue/aterrizaje, se realiza la transferencia de una persona y su equipaje desde la pista de despegue a la de aterrizaje. Cada capa realiza su servicio (1) realizando ciertas acciones en esa capa (por ejemplo, en la capa de puerta, recogiendo y dejando gente de un avión) y (2) utilizando servicios de la capa que está inmediatamente debajo de ella (por ejemplo, en la capa de puerta, utilizando el servicio de transferencia de pasajeros de pista a pista de la capa de despegue/aterrizaje).

Una arquitectura de capas nos permite discutir una parte específica bien definida de un sistema grande y complejo. Esta simplificación posee, en sí misma, un valor considerable proveyendo modularidad. Cuando un sistema tiene una estructura estratificada es mucho más fácil cambiar la implementación del servicio proporcionado por la capa. Mientras que la capa proporciona el mismo servicio a la capa anterior y utiliza los mismos servicios de la capa de abajo, el resto del sistema no cambia cuando se modifica la implementación de una capa. (¡tenga en cuenta que cambiar la implementación de un servicio es diferente de cambiar el propio servicio!). Por ejemplo, si se cambiaron las funciones de la puerta (por ejemplo, tener gente a bordo y desembarcar por altura), el resto del sistema de la línea aérea permanecería sin cambios, porque la capa de puertas proporciona la misma función (cargar y descargar gente); simplemente se implementa la función de una manera diferente después del cambio. Para sistemas grandes y complejos que se están actualizando constantemente, la capacidad de cambiar la implementación de un servicio sin afectar a sus componentes es otra ventaja de la estratificación.

Protocolo en Capas

Pero basta de líneas aéreas. Volvamos nuestra atención a los protocolos de red. Para reducir la complejidad del diseño, los diseñadores de redes organizan protocolos (y el hardware y software de red que implementa los protocolos) en **capas**. Con una arquitectura de protocolo en capas, cada protocolo pertenece a una de las capas, de la misma forma que cada función en la arquitectura de la aerolínea en la Figura 1.22 pertenecía a una capa. Estamos nuevamente interesados en los **servicios** que una capa ofrece a la capa de arriba -el tan llamado **modelos de servicio** de una capa. Así como en el caso de nuestro ejemplo de la aerolínea, cada capa provee sus servicios (1) realizando ciertas acciones dentro de la capa y (2) usando los servicios de la capa directamente por denajo. Por ejemplo, los servicios provistos por la capa n podrían incluir entrega confiable de mensajes desde un extremo de la red a otro. Esto podría estar implementado usando una entrega de extremo a extremo no confiable de la capa $n - 1$, y añadiendo la capa n funcionalidad para detectar y retransmitir mensajes perdidos.

Es importante darse cuenta de que un protocolo en la capa n es *distribuida* entre las entidades de la red (incluyendo sistemas terminales y switches de paquetes) que implementan el protocolo, igual que las funciones en la arquitectura.

tura por capas de nuestra línea aérea se distribuían entre los aeropuertos de salida y llegada. Dicho de otro modo, hay una parte de la capa n en cada una de las entidades de la red. Estas partes se comunican entre si intercambiando mensajes de la capa n . Estos mensajes se llaman unidades de datos del protocolo de la capa n , o más comúnmente **n -PDU**. El contenido y formato de un n -PDU, así como la forma en que se intercambian los n -PDU entre los elementos de red, se define por un protocolo de la capa n . Cuando se consideran todos juntos, los protocolos de las diferentes capas se llaman pila de protocolos.

Un protocolo de capa puede estar implementado en software, en hardware, o ser una combinación de ambos. Cuando la capa n de la máquina A envía un n -PDU a la capa n de la máquina B, la capa n de la máquina A *pasa* el n -PDU a la capa $n - 1$ y entonces permite a la capa $n - 1$ liberar el n -PDU a la capa n de B; por tanto, se dice que la capa n confía en la capa $n - 1$ para entregar su n -PDU en el destino. Un concepto fundamental es el de **modelo de servicio** de una capa. Se dice que la capa $n - 1$ ofrece **servicios** a la capa n . Por ejemplo, la capa $n - 1$ puede garantizar que el n -PDU llegará sin error a la capa si en el destino en un segundo, o puede sólo garantizar que el n -PDU llegará eventualmente al destino sin ninguna garantía sobre el error.

El concepto de estratificación de protocolos es bastante abstracto y, a veces, difícil de entender. El concepto quedará claro cuando estudiemos las capas de Internet y sus protocolos constituyentes con más detalle. Pero perdámonos un poco dentro de la estratificación de protocolos y pilas de protocolos con un ejemplo. Consideraremos una red que organiza sus protocolos de comunicación en cuatro niveles (lo que es diferente de Internet, que los organiza en cinco niveles), como muestra la Figura 1.22. Como hay cuatro niveles, hay

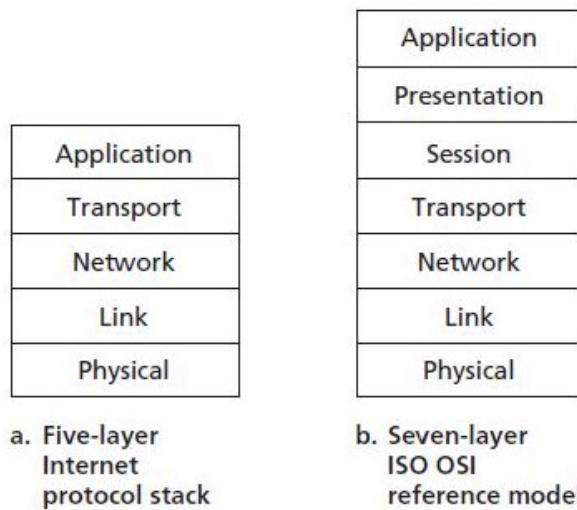


Figura 1.23 La Pila del Protocolo de Internet (a) y el modelo de referencia OSI (b)

cuatro tipos de PDU: 1-PDC, 2-PDU, 3-PDU, y 4-PDU. Como se ve en la Figura 1.22 la aplicación, que trabaja en la capa más alta, crea un mensaje, M . Cualquier mensaje creado en este nivel más alto es un 4-PDU. El mensaje M mismo puede contar de muchos campos diferentes (de la misma forma que una estructura o un registro en un lenguaje de programación puede constar de diferentes campos); es importante para la aplicación definir e interpretar los campos del mensaje. Los campos podrán contener el nombre del remitente, un código indicando el tipo de mensaje, y algunos datos adicionales.

Dentro de la máquina origen, el contenido del mensaje completo M es *pasado* hacia abajo en la pila de protocolos a la capa 3. En el ejemplo de la Figura 1.22, la capa 3 en la máquina origen divide un 4-PDU, M , en dos partes: M_1 y M_2 . La capa tres en la máquina origen añade entonces a M_1 y M_2 las llamadas **cabeceras** para crear dos PDU de la capa 3. Las cabeceras contienen la información adicional necesaria para que las partes emisora y receptora de la capa 3 implementen el servicio que la capa 3 proporciona a la capa 4. El procedimiento continua en el origen, añadiendo más cabecera con cada capa, hasta que se crean los 1-PDU. Los 1-PDU son enviados desde la máquina origen al enlace físico. En el otro extremo, la máquina destino recibe los 1-PDU y los dirige hacia arriba en la pila de protocolos. En cada capa se elimina la cabecera correspondiente. Finalmente, M es recomposto a partir de M_1

y M_2 y pasado a la aplicación.

Tenga en cuenta que en la Figura 1.22, la capa n utiliza los servicios de la capa $n - 1$. Por ejemplo, una vez que la capa 4 crea el mensaje M , lo pasa hacia la capa 3, y confía en dicha capa 3 para entregar el mensaje en la capa 4 del destino.

Por raro que parezca, esta noción de confiar en los servicios de la capa inferior es predominante en otras muchas formas de comunicación. Por ejemplo, consideremos el correo postal ordinario. Cuando usted escribe una carta, incluye información en el sobre, como la dirección de destino y la dirección de retorno, junto con la carta. La carta, junto con la información de la dirección, se puede considerar un PDU del nivel más alto de la pila de protocolos. Puede depositar el PDU en un buzón. A partir de este momento, la carta está fuera de sus manos. El servicio postal añadirá alguna información propia a su carta, añadiendo fundamentalmente una cabecera a la misma. Por ejemplo, en Estados Unidos se imprime a menudo un código de barras en la carta.

Una vez que haya usted depositado su sobre en un buzón, confía en los servicios de Correos para entregar la carta en el destino correcto en un plazo razonable. Por ejemplo, no se tiene que preocupar de si un camión postal se estropea mientras estaba transportando la carta, sino que el servicio de Correos se preocupa de ello, presumiblemente con un plan bien definido de recuperación de dichos fallos. Además, en el propio servicio de Correos hay capas, y los protocolos de una capa confían y utilizan los servicios de la capa de abajo.

Con el fin que una capa pueda interoperar con la capa inferior, las interfaces entre las dos capas deben estar definidas de forma precisa. Organizaciones estándares definen las interfaces entre capas adyacentes (por ejemplo, el formato de los PDU pasados entre capas) y permiten a los desarrolladores de hardware y software de redes implementar el interior de las capas como les guste. Por tanto, si se entrega una implementación nueva y mejorada de una capa, la nueva implementación puede reemplazar a la antigua, y, en teoría, las capas continuarán interoperando.

Funciones de capa

En una red de computadoras, cada capa puede realizar una o más del siguiente conjunto genérico de tareas:

- **Control de error**, que hace más confiable el canal lógico entre las capas en dos elementos colegas de la red.
- **Control de flujo**, que impide que un colega más lento sea saturado con PDU.
- **Segmentación y reensamblado**, que en el lado de transmisión divide trozos grandes de datos en partes más pequeñas, y en el lado del receptor reúne las partes pequeñas para reconstruir el trozo grande original.
- **Multiplexado**, que permite que varias sesiones de alto nivel comparten una única conexión de bajo nivel.
- **Establecimiento de conexión**, que proporciona acuerdo con un colega.

La estratificación de protocolos tiene ventajas estructurales y conceptuales. Mencionarémos, sin embargo, que algunos investigadores e ingenieros de redes se oponen vehementemente a la estratificación [Wakeman 1992]. Una desventaja potencial de la estratificación es que una capa puede duplicar la funcionalidad de una capa inferior. Por ejemplo, muchas pilas de protocolos proporcionan recuperación de errores tanto para el enlace como para la consideración terminal-a-terminal. Una segunda desventaja funcional es que la funcionalidad de una capa puede necesitar información (por ejemplo, un valor de marca de tiempo) que está presente sólo en otra capa, lo cual viola el objetivo de separación de capas.

1.5.2 La pila de protocolos de Internet

La pila de Internet consta de cinco capas: la física, la de enlace de datos, la de red, la de transporte y la de aplicación. En lugar de utilizar la terminología enrevesada n -PDU para cada una de las cinco capas, damos nombres especiales a los PDU en cuatro de las cinco capas: frame, datagrama, segmento, y mensaje. No damos nombre a la unidad de datos de la capa física (normalmente esta capa se utiliza sin nombre). La pila Internet y los nombres correspondientes de PDU se muestran en la Figura 1.23.

Una capa de protocolo puede implementarse en software, en hardware, o en una combinación de los dos. Los protocolos de la capa de aplicación, tales como HTTP y SMTP -casi siempre se implementan en software en los

sistemas finales; también lo están los protocolos de la capa de transporte. Debido a que las capas física y de enlace de datos son responsables de manejar la comunicación a través de un enlace específico, se implementan típicamente en una tarjeta de interfaz de red (por ejemplo, tarjetas de interfaz de WiFi o Ethernet) asociado a un determinado enlace. La capa de red es a menudo una implementación mixta de hardware y software. También notemos que al igual que las funciones de la arquitectura de línea aérea en capas se distribuyeron entre los distintos aeropuertos y centros de control de vuelo que componen el sistema, también lo es un protocolo de capa n *distribuidos* entre los sistemas finales, los commutadores de paquetes, y otros componentes que conforman la red. Es decir, hay a menudo una pieza de un protocolo de capa n en cada uno de estos componentes de la red.

La estratificación de protocolos tiene ventajas conceptuales y estructurales [RFC 3439]. Como hemos visto, la estratificación proporciona una forma estructurada para discutir los componentes del sistema. La modularidad hace que sea más fácil actualizar los componentes del sistema. Mencionamos, sin embargo, que algunos investigadores e ingenieros de redes se oponen vehementemente a la estratificación [Wakeman 1992]. Una desventaja potencial de la estratificación es que una capa puede duplicar la funcionalidad de una capa inferior. Por ejemplo, muchas pilas de protocolos proporcionan la recuperación de errores tanto en función de cada enlace y una base de extremo a extremo. Una segunda desventaja potencial es que la funcionalidad en una capa puede necesitar información (por ejemplo, un valor de marca de tiempo) que está presente sólo en otra capa; esto viola el objetivo de la separación de las capas.

Cuando se toman juntos, los protocolos de las diferentes capas se denominan la pila de protocolos. La pila de protocolo de Internet consiste en cinco capas: la, capas físicas enlace, red, transporte y aplicación, como se muestra en la Figura 1.23 (a). Si se examina la tabla de contenido, verá que hemos organizado más o menos este libro utilizando las capas de la pila de protocolos de Internet. Tomamos un enfoque de arriba hacia abajo, primero que cubre la capa de aplicación y luego proceder a la baja.

Una capa de protocolo se puede implementar en software, en hardware, o en una combinación de los dos. Los protocolos de la capa de aplicación (como HTTP y SMTP) están casi siempre implementados en software en los sistemas terminales; son, por tanto, protocolos de la capa de transporte. Como las capas física de enlace de datos son responsables del mantenimiento de la comunicación sobre un enlace específico, están implementadas típicamente en una tarjeta de interfaz de red (por ejemplo, las tarjetas de interfaz Ethernet o ATM) asociada con un enlace dado. La capa de red es, a menudo, una implementación mezcla de hardware y software. Resumimos ahora las capas de Internet y los servicios que proporcionan.

Capa de aplicación

La capa de aplicación es responsable de soportar las aplicaciones de red. La capa de aplicación incluye muchos protocolos, entre ellos HTTP para soportar Web, SMTP para soportar correo electrónico, y FTP para soportar transferencia de archivos. En el Capítulo 2 veremos que es muy fácil crear nuestros propios protocolos nuestra de la capa de aplicación.

Capa de transporte

La capa de transporte proporciona el servicio de transporte de mensajes de la capa de aplicación entre los lados del cliente y el servidor de una aplicación. En Internet hay dos protocolos de transporte: TCP y UDP: cualquiera de ellos puede transportar mensajes de la capa de aplicación. TCP proporciona a sus aplicaciones un servicio orientado a conexión. Este servicio incluye la entrega garantizada de los mensajes de la capa de aplicación a su destino, y el control del flujo (es decir, la concordancia de la velocidad del emisor y el receptor). TCP segmenta también los mensajes largos y proporciona un mecanismo de control de cogestión, de modo que un emisor disminuye su velocidad de transmisión cuando la red está congestionada. El protocolo UDP proporciona a sus aplicaciones un servicio sin conexión, que (como vimos en la Sección 1.2) es un servicio sin extras.

Capa de red

La capa de red es responsable de rutar los datagramas de una máquina a otra. La capa de red de Internet tiene dos componentes principales. Tiene un protocolo que define los campos del datagrama IP y cómo actúan los sistemas terminales y routers sobre esos campos. Este protocolo es el celebrado IP. Sólo existe un protocolo IP, y todos los componentes Internet que tengan una capa de red deben ejecutar el protocolo IP. La capa de protocolo de red de

Internet debe contener también los protocolos de rutado que determinan las rutas que toman los datagramas entre las fuentes y los destinos. Internet tiene muchos protocolos de rutado. Como se vio en la Sección 1.3, Internet es una red de redes, y en una red, el administrador de la misma puede ejecutar cualquier protocolo de rutado deseado. Aunque la capa de red contiene tanto el protocolo IP como numerosos protocolos de rutado, normalmente se la designa como capa IP, lo que refleja el hecho de que IP es el pegamento que junta Internet.

El protocolo de la capa de transporte (TCP o UDP) en una máquina origen pasa un segmento de la capa de transporte y un destino a la capa IP, del mismo modo que usted proporciona al servicio postal una dirección de destino. La capa IP proporciona entonces el servicio de entrega del segmento a la máquina de destino. Cuando el paquete llega a la máquina destino, IP pasa el segmento a la capa de transporte de la máquina.

Capa de enlace

La capa de red ruta un paquete a través de una serie de switches de paquetes (llamados routers en Internet) entre la fuente y el destino. Para mover un paquete de un nodo (máquina o switch de paquetes) al siguiente nodo en la ruta, la capa de red debe confiar en los servicios de la capa de enlace. En particular, en cada nodo la capa de red baja el datagrama hacia la capa de enlace, que entrega el datagrama al siguiente nodo de la ruta. En el siguiente nodo, la capa de enlace sube el datagrama a la capa de red.

Los servicios proporcionados por la capa de enlace dependen del protocolo específico de dicha capa que se emplea sobre el enlace. Por ejemplo, algunos protocolos proporcionan una entrega confiable sobre la base del enlace (es decir, del nodo transmisor), sobre el enlace, al nodo receptor. Ten en cuenta que este servicio de entrega es diferente del servicio de entrega confiable de TCP, que proporciona entrega confiable de un sistema terminal al otro. Ejemplos de capas de enlace son Ethernet y PPP; en algunos contextos, ATM y Frame Relay se pueden considerar capas de enlace. Como los datagramas necesitan normalmente atravesar varios enlaces para ir desde el origen al destino, un datagrama puede ser soportado por diferentes protocolos de la capa de enlace en diferentes enlaces a lo largo de la ruta. Por ejemplo, un datagrama puede ser soportado por Ethernet en un enlace y por PPP en el siguiente. La capa de red recibirá un servicio diferente en cada uno de los diferentes protocolos de la capa de enlace.

Capa física

Mientras el trabajo de la capa de enlace es mover frames completos de un elemento de red al adyacente, el trabajo de la capa física es mover los bits *individuales* del marco de un nodo hasta el siguiente. Los protocolos de esta capa son, de nuevo, dependientes del enlace, y dependen además del medio de transmisión actual del enlace (por ejemplo, el par trenzado de cobre, la fibra óptica de modo simple). Por ejemplo, Ethernet tiene muchos protocolos de capa física: uno para cable de par trenzado de cobre, otro para cable coaxial, otro para fibra, y así sucesivamente. En cada caso, un bit se mueve sobre el enlace de forma distinta.

Hemos organizado aproximadamente este curso utilizando las capas de la pila de protocolos de Internet. Consideraremos una **aproximación de arriba abajo**, cubriendo primero la capa de aplicación y yendo después hacia abajo.

1.5. 3Entidades y capas de red

Las entidades más importantes de la red son los sistemas terminales y los switches de paquetes. Como se discutirá más adelante en este curso, hay dos tipos de switches de paquetes: los routers y los bridges. En las secciones anteriores hemos presentado un resumen de los routers. Los bridges se discutirán con detalle en el Capítulo 5, mientras que los routers se verán con detalle en el Capítulo 4. De forma semejante a los sistemas terminales, los routers y los bridges organizan el hardware y el software de red en capas. Pero los routers y los bridges no implementan *todas* las capas de la pila de protocolos; normalmente sólo implementan las capas inferiores. Como se ve en la Figura 1.24, los bridges implementan las capas 1 y 2; los routers implementan las capas 1 a 3. Esto significa, por ejemplo, que los routers Internet son capaces de implementar el protocolo IP (un protocolo de capa 3), mientras que los bridges no. Veremos más adelante que mientras que los bridges no reconocen las direcciones de la capa 2 (como las direcciones IP), son capaces de reconocer direcciones de la capa 2 (como las direcciones Ethernet). Dese cuenta de que los computadoras implementan las cinco capas: esto es consistente con la visión de que la arquitectura Internet pone mucha de su funcionalidad en los *bordes* de la red.

1.6 Problemas y preguntas de repaso

1.6.1 Preguntas de repaso

Secciones 1.1-1.5

1. ¿Cuál es la diferencia entre un *host* y un *sistema terminal*? Enumere diferentes tipos de sistemas terminales.
¿La Web es un sistema terminal?
2. La palabra "protocolo" se utiliza, a menudo, en los medios referida a las relaciones diplomáticas. Dé un ejemplo de un protocolo diplomático.
3. ¿Qué es un programa cliente? ¿Qué es un programa servidor? ¿Recibe un programa servidor servicios de solicitud y recepción desde un programa cliente?
4. ¿Cuál son los dos tipos de servicios que Internet proporciona a sus aplicaciones? ¿Cuáles son algunas características de cada uno de estos servicios?
5. Se ha dicho que el control de flujo y el de cogestión son equivalentes. ¿Es eso verdad para el servicio orientado a conexión de Internet? ¿Los objetivos del control de flujo y del de cogestión son los mismos?
6. Proporcione una descripción breve de alto nivel de como el servicio orientado a conexión de Internet proporciona transporte confiable.
7. ¿Qué ventaja tiene una red de conmutación de circuitos frente a una de conmutación de paquetes? ¿Qué ventajas tiene TDM frente a FDM en una red de conmutación de circuitos?
8. ¿Por qué se dice que la conmutación de paquetes emplea el multiplexado estadístico? Contraste el multiplexado estadístico con el que tiene lugar en TDM.
9. Supongamos que hay exactamente un switch de paquetes entre un host que envía y un host que recibe. Las tasas de transmisión entre el host que envía y el que recibe son R_1 , y R_2 , respectivamente. Suponiendo que el switch utiliza conmutación de paquetes almacenar-y-enviar. ¿cuál es el retardo total terminal-a-terminal para enviar un paquete de longitud L? (Ignorar el retardo de cola, de propagación y de proceso.)
10. ¿Qué se entiende por información sobre el estado de la conexión en una red de circuito virtual? Si en un switch en una red de VC, las conexiones se establecen y se rompen a un ritmo de una conexión por milisegundos (de promedio), ¿a qué tasa necesita ser modificada la tabla de reenvío en el switch?
11. Supongamos que está desarrollando un estándar para un nuevo tipo de red. Necesita decidir dónde utilizará su red rutado de VC o datagrama. ¿Cuáles son las ventajas y los inconvenientes de utilizar los VC?
12. ¿Cuál son las ventajas de la segmentación de mensajes en las redes de conmutación de paquetes? ¿Cuáles son las desventajas?
13. Cite seis tecnologías diferentes de acceso. Clasifique cada uno como de acceso residencial, acceso de empresa o acceso móvil.
14. ¿Cuál es la diferencia fundamental que distingue un proveedor de servicios Internet de nivel 1 y otro de nivel 2?
15. ¿Cuál es la diferencia entre un POP y un NAP?
16. ¿Es el ancho de banda HFC dedicado, o compartido entre usuarios? ¿Son posibles las colisiones en un canal HFC hacia atrás? ¿Por qué si o por qué no?
17. ¿Cuál es la tasa de transmisión de las redes de área local de Ethernet? Para una tasa de transmisión dada, ¿puede cada usuario de la LAN transmitir continuamente a esa tasa?
18. ¿Cuáles son algunos medios físicos sobre los que puede funcionar Ethernet?
19. Los modems de teléfono, HFC, y ADSL, se utilizan todos para acceso residencial. Por cada una de estas tecnologías, proporcione un rango de transmisión, y comente en cada caso si el ancho de banda es compartido o dedicado.

Secciones 1.6-1.8

20. Considere el envío de una serie de paquetes desde un host emisor hasta otro receptor sobre una ruta fijada. Señale los componentes de retardo en el retardo terminal-a-terminal para un único paquete. ¿Cuáles de dichos retardos son constantes y cuáles variables?
21. Cite cinco tareas que pueda realizar una capa, ¿Es posible que una o más de dichas tareas puedan ser realizadas por dos (o más) capas?
22. ¿Cuáles son las cinco capas de la pila de protocolos de Internet? ¿Cuáles son las principales responsabilidades de esas capas?
23. ¿Cuál de las capas de la pila de protocolos de Internet realiza el proceso de rutado?

1.6.2 Problemas

1. Diseñe y describa un protocolo del nivel de aplicación para ser usado entre un cajero automático y el computador central del banco. Su protocolo debe permitir que se verifique la tarjeta y la clave del usuario, que se consulte el balance de la cuenta (que se mantiene en el computador central), y que se realicen reintegros de una cuenta (es decir, dinero reintegrado al usuario). Las entidades de su protocolo deben ser capaces de resolver el problema tan común de que no haya suficiente dinero en la cuenta para cubrir el reintegro. Especifique su protocolo listando los mensajes intercambiados y la acción tomada por el cajero automático o el computador central del banco ante la transmisión o recepción de mensajes. Esboce la operación de su protocolo para el caso de un reintegro sencillo sin errores, utilizando un diagrama semejante al de la Figura 1.2. Establezca explícitamente las suposiciones realizadas por su protocolo sobre el servicio de transporte terminal-a-terminal subyacente.
2. Considere una aplicación que transmite datos a una velocidad estacionaria (por ejemplo, el emisor genera una unidad de N -bits de datos cada k unidades de tiempo, donde k es pequeño y fijo). Cuando comienza dicha aplicación, continua ejecutándose durante un periodo de tiempo relativamente largo. Responda a las siguientes cuestiones, justificando brevemente su respuesta:
 - (a) ¿Sería más apropiada para esta aplicación una red de conmutación de paquetes o una de conmutación de circuitos? ¿Por qué?
 - (b) Supongamos que se utiliza una red de conmutación de paquetes y que el único tráfico de esta red llega de las aplicaciones descritas anteriormente. Además, supongamos que el resultado de la suma de las tasas de datos de la aplicación es menor que las capacidades de cada uno de los enlaces. ¿Se precisa alguna forma de control de congestión? ¿Por qué?
3. Considere la red de conmutación de circuitos en la Figura 1.5. Recuerde que hay n circuitos en cada enlace.
 - (a) ¿Cuál es el número máximo de conexiones simultáneas que pueden estar funcionando al mismo tiempo en esta red?
 - (b) Supongamos que todas las conexiones están entre el switch en la esquina superior izquierda y el de la esquina inferior derecha. ¿Cuál es el número máximo de conexiones simultáneas que pueden estar funcionando al mismo tiempo en esta red?
4. Revise la analogía de la caravana de coches en la Sección 1.6. De nuevo, suponer una velocidad de propagación de 100 km/hora.
 - (a) Supongamos que la caravana viaja 200 km, comenzando delante de un peaje, pasando a través de un segundo peaje, y finalizando justo delante de un tercer peaje. ¿Cuál es el retardo terminal-a-terminal?
 - (b) Repita (a), suponiendo ahora que hay siete coches en la caravana en lugar de 10.
5. Considere el envío de un archivo de $F = M \cdot L$ bits sobre un recorrido de Q enlaces. Cada enlace transmite a R bps. La red está ligeramente cargada, de manera que no hay retardos de cola. Cuando se utiliza una forma de conmutación de paquetes, los $M \cdot L$ bits se dividen en M paquetes, cada uno de L bits. El retardo de propagación es despreciable.

- (a) Supongamos que la red es una de circuito virtual de conmutación de paquetes. Se denota el tiempo de activación de VC por t , segundos. Supongamos que las capas de envío añaden un total de h bits de cabecera de cada paquete. ¿Cuánto tiempo se precisa para enviar el archivo desde la fuente hasta el destino?
- (b) Supongamos que la red es una red de datagramas de conmutación de paquetes y que se utiliza un servicio sin conexión. Ahora supongamos que cada paquete tiene $2h$ bits de cabecera. ¿Cuánto tiempo se precisa para enviar el archivo?
- (c) Repetir (b), pero suponer que se utiliza la conmutación de mensajes (es decir, se añaden $2h$ bits al mensaje y el mensaje no es segmentado).
- (d) Finalmente, supongamos que la red es una de conmutación de circuitos. Además, supongamos que la tasa de transmisión del circuito entre fuente y destino es de R bps. Suponiendo que t , es el tiempo de activación y que se añaden h bits de cabecera al archivo entero, ¿cuánto tiempo se precisa para el envío del archivo?
6. Experimente con el applet de lava de segmentación de mensajes del sitio •eb del libro, ¿Los retardos en el applet corresponden a los retardos en la cuestión anterior? ¿Cómo afectan los retardos de propagación de enlace al retardo total terminal-a-terminal para la conmutación de paquetes (con segmentación de mensajes) y para la conmutación de mensajes? 7. Considere enviar un archivo grande de F bits desde el host A al host B . Hay dos enlaces (y un switch) entre A y B , y los enlaces están descongestionados (es decir, no hay colas de retardo). El host A segmenta el archivo en segmentos de S bits cada uno, y añade 40 bits de cabecera para cada segmento, formando paquetes de $L = 40 + S$ bits. Cada enlace tiene una tasa de transmisión de R bps. Encuentre el valor de S que minimiza el retardo de mover el archivo desde el host A al host B . No considere el retardo de propagación.
7. Este problema elemental comienza explorando e retardo de propagación y de transmisión, dos conceptos fundamentales en redes de datos. Considere dos host, A y B , conectados por un único enlace a la tasa de R bps. Supongamos que los dos host están separados m metros, y supongamos que la velocidad de propagación entre el enlace es s metros/seg. El host A envía un paquete de L bits de tamaño al host B .
- Expresar el retardo de propagación, d_{prop} , en términos de m y s .
 - Determinar el tiempo de transmisión del paquete, d_{trans} en función de L y R .
 - Ignorando los retardos de proceso y de cola, obtener una expresión del retardo terminal-a-terminal.
 - Supongamos que el host A comienza a transmitir el paquete en el instante $t = 0$. En el tiempo $t = d_{trans}$ ¿dónde está el último bit del paquete?
 - Supóngase que d_{prop} es más grande que d_{trans} . En el instante $t = d_{trans}$ ¿dónde está el primer bit del paquete?
 - Supóngase que d_{prop} es menor que d_{trans} . En el instante $t = d_{trans}$ ¿dónde está el primer bit del paquete?
 - Supóngase $s = 2,5 \cdot 10^8$, $L = 100$ bits, y $R = 28$ Kbps. Encontrar la distancia m tal que d_{prop} sea igual a d_{trans} .
8. En este problema consideremos enviar voz desde el host A al host B sobre una red de conmutación de paquetes (por ejemplo, teléfono Internet). El host A convierte voz analógica en una secuencia de bit digital de 64 Kbps sobre la marcha. Después, el host A agrupa los bits en paquetes de 48 bytes. Hay un enlace entre A y B ; su tasa de transmisión es de 1 Mbps, y su retardo de propagación de 2 mseg. Tan pronto como el host A reúne un paquete, lo envía al host B . Tan pronto como el host B recibe un paquete completo, convierte los bits del paquete en señal analógica. ¿Cuánto tiempo transcurre desde el instante en que se crea un bit (de la señal original en el host A) hasta que ese bit es decodificado (como parte de una señal analógica en el host B)?
9. Supóngase que varios usuarios comparten un enlace de 1 Mbps. Supóngase también que cada usuario precisa 100 Kbps cuando transmite, pero transmite sólo durante el 10 por ciento del tiempo (Véase la discusión sobre conmutación de paquetes frente a conmutación de circuitos en la Sección 1.3).
- Cuando se utiliza conmutación de circuitos, ¿cuántos usuarios se pueden soportar?
 - Para el resto de este problema, suponer que se utiliza conmutación de paquetes. Encontrar la probabilidad de que un usuario dado esté transmitiendo.
 - Supongamos que hay 40 usuarios. Encontrar la probabilidad de que en un instante dado estén transmitiendo exactamente n usuarios simultáneamente. (Pista: Utilizar la distribución binomial.)

- (d) Encontrar la probabilidad de que haya 11 u más usuarios transmitiendo simultáneamente.
10. Considérese la discusión de la Sección 1.3, commutación de paquetes frente a commutación de circuitos, en la que se proporciona un ejemplo con un enlace de 1 Mbps. Los usuarios están generando datos a una tasa de 100 Kbps cuando están ocupados, pero están ocupados generando datos sólo con una probabilidad $p = 0,1$. Supongamos que el enlace de 1 Mps es reemplazado por otro de 1 Gbps.
- ¿Cuál será N , el máximo número de usuarios que puede ser soportado simultáneamente bajo commutación de circuitos?
 - Consideremos ahora commutación de paquetes y una población de M usuarios. Proporcione una fórmula (en términos de p , M , N) para la probabilidad de que más de N usuarios estén enviando datos.
11. Considérese el retardo de cola en un búfer de router (que precede a un enlace de salida). Supongamos que todos los paquetes son de L bits, la tasa de transmisión es de R bps, y N paquetes llegan simultáneamente al búfer cada LN/R segundos. Encontrar el promedio de retardo de cola de un paquete (Pista: El retardo de cola para el primer paquete es cero; para el segundo es L/R ; para el tercero $2L/R$. El N -ésimo paquete ya ha sido transmitido cuando llega el segundo.)
12. Considérese el retardo de cola en un búfer de router. Sea I la intensidad de tráfico, es decir, $I = La/R$. Supongamos que el retardo de cola tiene la forma $IL/R(1 - I)$ para $1 \leq I \leq 1$
- Proporcione una fórmula para el retardo total, es decir, el retardo de cola más el retardo de transmisión.
 - Dibuje el retardo total como una función de L/R .
13. (a) Aplique la fórmula del retardo terminal-a-terminal de la Sección 1.6, para tasas de proceso, tasas de transmisión y retardos de propagación heterogéneos.
- (b) Repita (a), pero ahora suponga que hay un retardo de cola de d_{cola} en cada nodo.
14. Supongase que dos host, A y B , están separados por 10.000 kilómetros y conectados por un enlace directo de $R = 1$ Mbps. Supongase que la velocidad de propagación sobre el enlace es de $2.5 * 10^8$ metros/seg.
- Calcúlese el *producto retardo de ancho de banda* $R * t_{prop}$. Considere el envío de un archivo de 400.000 bits desde el host A hasta el host B . Supóngase que el archivo se envía de forma continua como un gran mensaje. ¿Cuál es el número máximo de bits que estarán en el enlace en un instante dado?
 - Proporcione una interpretación de producto retardo-ancho de banda.
 - ¿Cuál es la anchura (en metros) de un bit en el enlace? ¿Es más grande que un campo de fútbol?
 - Obténgase una expresión general para el ancho de un bit en función de la velocidad de propagación (s) el ancho de banda (R) y la longitud del enlace (m).
15. Volviendo al problema 15, suponga que queremos modificar R . ¿Para qué valor de R el ancho de un bit es tan grande como la longitud del enlace?
16. Considere el Problema 15, pero ahora con un enlace de $R = 1$ Gbps.
- Calcule el ancho de banda del producto retardo-ancho de banda, $R * t_{prop}$.
 - Considere el envío de un archivo de 400.000 bits desde el host A al host B . Suponga que el archivo se envía de forma continua como un gran mensaje. ¿Cuál es el número máximo de bits que estarán en el enlace en un instante dado?
 - ¿Cuál es el ancho en metros de un bit en el enlace?
17. Nos referimos de nuevo al Problema 15.
- ¿Cuánto tiempo se precisa para enviar el archivo, suponiendo que se envie de forma continua?
 - Suponga ahora que el archivo se divide en diez paquetes, conteniendo cada uno 40.000 bits. Suponga que cada paquete es reconocido por el receptor, y que el tiempo de transmisión de un paquete de reconocimiento es despreciable. Por último, suponga que el emisor no puede enviar un paquete hasta que se ha reconocido el precedente. ¿Cuánto tiempo se precisa para enviar el archivo?
 - Compare Los resultados de (a) y (b).

18. Supongase que hay un enlace de microondas de 10 Mbps entre un satélite y su estación base en la Tierra. Cada minuto, el satélite toma una foto digital y la envía a la estación base. Supóngase una velocidad de propagación de $2,4 * 10^8$ metros/seg.
 - (a) ¿Cuál es el retardo de propagación del enlace?
 - (b) ¿Cuál es el producto retardo-ancho de banda, $R * t_{prop}$
 - (c) Considere x como el tamaño de la foto. ¿Cuál es el valor mínimo de x para que el enlace de microondas esté transmitiendo continuamente?
19. Considérese la analogía del viaje en un avión de aire en nuestra discusión sobre estratificación de la Sección 1.7, y la inclusión de cabeceras para las unidades de datos del protocolo como fluyen hacia debajo de la pila de protocolos. ¿Hay una noción equivalente a la información de cabecera que es añadida a los pasajeros y su equipaje cuando se mueven hacia debajo de la pila de protocolos de la línea aérea?

PART II

2. LA CAPA DE APLICACION

CHAPTER 2

INTRODUCCION

Las aplicaciones de red son la *razón de ser* de las redes de computadoras. Si no pudiésemos concebir ninguna aplicación de utilidad, no sería necesario diseñar protocolos de red que les dieran soporte. Desde la concepción de Internet, numerosas aplicaciones útiles y de entretenimiento han sido creadas. Estas aplicaciones han sido la fuerza conductora detrás del éxito de Internet, motivando a gente en sus hogares, escuelas, gobiernos, y comerciantes, a hacer de Internet una parte integral de sus actividades diarias.

Entre estas aplicaciones se incluyen las clásicas basadas en texto que se hicieron populares en los '70s y '80s: el correo electrónico de texto, acceso remoto a computadoras, la transferencia de archivos, los grupos de noticias y los *chat* textuales. También la aplicación *definitiva* de mediados de 1990, la World Wide Web (Web), abarcando la navegación Web, búsqueda y comercio electrónico. Éstos incluyen mensajeo instantáneo y compartición de archivos P2P, las dos aplicaciones definitivas introducidas al final del milenio. Desde 2000, hemos visto una explosión de aplicaciones populares de voz y video, incluyendo: voz-sobre-IP (VoIP) y video conferencias sobre IP como Skype; distribución de videos generados por usuarios, como Youtube; y películas bajo demanda como Netflix. Durante el mismo período, también hemos visto la emergencia de juegos online multijugadores altamente atractivos, incluyendo Second Life y World of Warcraft. Y más recientemente, hemos visto la emergencia de una nueva generación de aplicaciones de redes sociales, tales como Facebook y Twitter, las cuales han creado redes humanas atractivas en la cima de las redes de Internet de routers y enlaces de comunicación. Claramente, no ha habido una ralentización de nuevas y emocionantes aplicaciones de Internet. Quizás algunos de los lectores de este libro van a crear la próxima generación de aplicaciones definitivas de Internet!

En este capítulo se estudian los aspectos conceptuales y de implementación de las aplicaciones de red. Se comienza definiendo los conceptos clave de la capa de aplicación, incluyendo los servicios de red requeridos por las aplicaciones, clientes y servidores, procesos, y las interfaces de la capa de transporte. Después se examinan en detalle distintos protocolos de la capa de aplicación, incluyendo la Web, e-mail, DNS, y distribución de archivos *peer-to-peer* (P2P) (El capítulo 8 se enfoca en aplicaciones multimedia, incluyendo streaming video y VoIP). Posteriormente, se trata el desarrollo de aplicaciones de red, tanto sobre TCP como sobre UDP. En particular, se estudia la API sockets y una aplicación sencilla cliente-servidor en Python. También proveemos varias asignaciones de programación divertidas e interesantes al final del capítulo.

La capa de aplicación es un lugar particularmente bueno para comenzar nuestro estudio de los protocolos. Es un campo conocido en el que estamos familiarizados con muchas aplicaciones que se basan en los protocolos que estudiaremos. Esto nos dará una buena idea de lo que son los protocolos, y nos presentará muchos de los aspectos que veremos de nuevo cuando estudiemos los protocolos de las capas de transporte, red y enlace de datos.

2.1 Principios de las Aplicaciones de Red

Supón que tienes una idea para una nueva aplicación de red. Tal vez esta aplicación será un gran servicio a la humanidad, o va a complacer a tu profesor, o te traerá una gran riqueza, o será simplemente divertida de desarrollar. Cualquiera que sea la motivación, por ahora vamos a examinar cómo transformar la idea en una aplicación de red en el mundo real.

En el núcleo de desarrollo de aplicaciones de red está la escritura de programas que se ejecutan en diferentes sistemas finales y se comunican entre sí a través de la red. Por ejemplo, en la aplicación web hay dos programas diferentes que se comunican entre sí: el programa navegador que se ejecuta en el host del usuario (de escritorio, portátiles, tablets, teléfonos inteligentes, etc.); y el programa de servidor Web que se ejecuta en la máquina del servidor Web. Como otro ejemplo, en un sistema de intercambio de archivos P2P hay un programa en cada host que participa en la comunidad de intercambio de archivos. En este caso, los programas en los diversos hosts pueden ser similares o idénticos.

Por lo tanto, mientras desarrollas tu nueva aplicación, necesitas escribir software que va a ejecutarse en múltiples sistemas finales. Este software se puede escribir, por ejemplo, en C, Java o Python. Es importante destacar que no es necesario escribir el software que se ejecuta en dispositivos del núcleo de la red, como routers o switches de capa de enlace. Incluso si quisieras escribir software de aplicación para estos dispositivos del núcleo de la red, no serías capaz de hacerlo. Como aprendimos en el Capítulo 1, y como se muestra anteriormente en la Figura 1.24, los dispositivos del núcleo de la red no funcionan en la capa de aplicación, en lugar de ello, funcionan en las capas inferiores (en concreto, a nivel de red y por debajo). Este diseño básico (a saber, confinando software de aplicación para los sistemas finales) como se muestra en la Figura 2.1, ha facilitado el rápido desarrollo y despliegue de una amplia gama de aplicaciones de red.

2.1.1 Arquitectura de las Aplicaciones de Red

Antes de entrar en la codificación de software, debemos tener un amplio plan arquitectónico para nuestra aplicación. Ten en cuenta que la arquitectura de una aplicación es claramente diferente de la arquitectura de red (por ejemplo, la arquitectura de Internet de cinco capas discutido en el capítulo 1). Desde la perspectiva del desarrollador de aplicaciones, la arquitectura de red es fija y proporciona un conjunto específico de servicios de aplicaciones. La **arquitectura de la aplicación**, por otra parte, es diseñada por el desarrollador de la aplicación y dicta cómo la aplicación se estructura en los diversos sistemas finales. En la elección de la arquitectura de la aplicación, un desarrollador de aplicaciones probablemente recurra a uno de los dos paradigmas arquitectónicos predominantes utilizados en aplicaciones de redes modernas: la arquitectura cliente-servidor o la arquitectura peer-to-peer (P2P).

En una **arquitectura cliente-servidor**, siempre hay un host encendido, llamado *servidor*, que da servicio a las peticiones de muchos otros hosts, llamados *clientes*. Un ejemplo clásico es la aplicación Web para el que un servidor Web siempre encendido atendiendo peticiones de los navegadores que se ejecutan en un host cliente. Cuando un servidor Web recibe una solicitud de un objeto desde un host cliente, responde enviando el objeto solicitado al host cliente. Observemos 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 directamente. Otra característica de la arquitectura cliente-servidor es que el servidor tiene una dirección fija, bien conocida, llamada dirección IP (de la que hablaremos en breve). Debido a que el servidor tiene una dirección fija, bien conocida, y a que el servidor está siempre encendido, un cliente siempre puede ponerse en contacto con el servidor mediante el envío de un paquete a la dirección IP del servidor. Algunas de las aplicaciones más conocidas con una arquitectura cliente-servidor incluyen la Web, FTP, Telnet, y el correo electrónico. La arquitectura cliente-servidor se muestra en la Figura 2.2 (a).

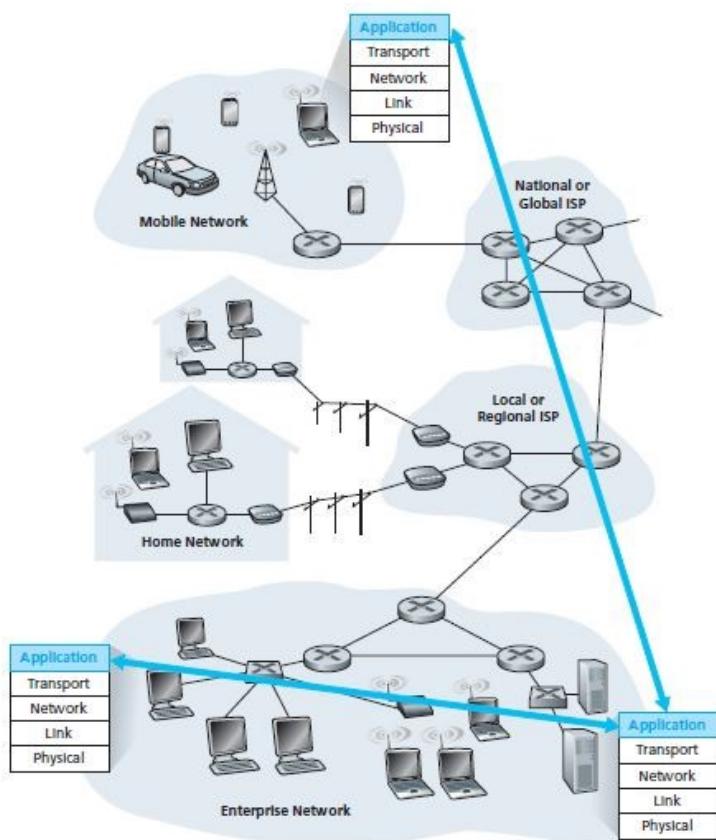


Figura 2.1

A menudo, en una aplicación cliente-servidor, un servidor único es incapaz de manejar todas las peticiones de los clientes. Por ejemplo, un sitio de redes sociales populares puede verse rápidamente abrumado si sólo tiene un manejo de todas las solicitudes en el servidor. Por esta razón, un **centro de datos**, que alberga un gran número de hosts, se utiliza a menudo para crear un servidor virtual de gran alcance. Los servicios de Internet más populares -tales como los motores de búsqueda (por ejemplo, Google y Bing), el comercio por Internet (por ejemplo, Amazon y e-Bay), el correo electrónico basado en la Web (por ejemplo, Gmail y Yahoo Mail), redes sociales (por ejemplo, Facebook y Twitter) -emplean uno o más centros de datos. Como se discutió en la Sección 1.3.3, Google tiene entre 30 y 50 centros de datos distribuidos en todo el mundo, que manejan colectivamente búsquedas, YouTube, Gmail y otros servicios. Un centro de datos puede tener cientos de miles de servidores, los cuales deben ser alimentados eléctricamente y mantenidos. Además, los proveedores de servicios deben pagar los costos recurrentes de interconexión y de ancho de banda para enviar datos desde sus centros de datos.

En una **arquitectura P2P**, hay una mínima dependencia (o no) en los servidores dedicados en los centros de datos. En cambio, la aplicación aprovecha la comunicación directa entre pares de hosts conectados de forma intermitente, llamados *pares (peers)*. Los pares no son propiedad del proveedor de servicios, sino que son computadoras de escritorio y laptops controladas por los usuarios, con la mayoría de los pares residiendo en hogares, universidades y oficinas. Debido a que los pares se comunican sin pasar por un servidor dedicado, la arquitectura se llama peer-to-peer. Muchas de las aplicaciones más populares y de gran intensidad de tráfico actuales se basan en arquitecturas P2P. Estas aplicaciones incluyen el intercambio de archivos (por ejemplo, BitTorrent), aceleración de descarga asistida por pares (por ejemplo, Xunlei), telefonía por Internet (por ejemplo, Skype), e IPTV (por ejemplo, Kankan y ppstream). La arquitectura P2P se ilustra en la Figura 2.2(b). Hemos de mencionar que algunas aplicaciones tienen arquitecturas híbridas, combinando elementos de ambos, cliente-servidor y P2P. Por ejemplo, para muchas aplicaciones de mensajería instantánea, los servidores se utilizan para realizar un seguimiento de las direcciones IP de los usuarios, pero los mensajes de usuario a usuario se envían directamente entre los hosts usuarios (sin pasar por servidores intermedios).

Una de las características más atractivas de las arquitecturas P2P es su **auto-escalabilidad**. Por ejemplo, en una aplicación de intercambio de archivos P2P, aunque cada uno de los pares genera carga de trabajo solicitando archivos, cada par también añade la capacidad de servicio al sistema mediante la distribución de archivos a otros pares. Las arquitecturas P2P también son rentables, ya que normalmente no requieren infraestructura de servidores y ancho de banda de servidores significativos (en contraste con los diseños de los clientes-servidor con centros de datos). Sin embargo, las futuras aplicaciones P2P se enfrentan a tres retos principales:

- *ISP amigable.* La mayoría de los ISP residenciales (incluyendo DSL e ISP por cable) han sido dimensionados para el uso del ancho de banda "asimétricas", es decir, para mayor tráfico de bajada que de subida. Pero video streaming P2P y las aplicaciones de archivos distribuidos cambiaron el tráfico de subida desde los servidores a los ISP residenciales, poniendo de este modo una presión significativa sobre los proveedores de Internet. Las futuras aplicaciones P2P necesitan ser diseñadas de manera que sean amigables con los ISP [Xie 2008].
- *Seguridad.* Debido a su naturaleza altamente distribuida y abierta, las aplicaciones P2P puede ser un desafío para la seguridad [Doucer 2002; Yu 2006; Liang 2006; Naoumov 2006; Dhungel 2008; Leblond 2011].
- *Incentivos.* El éxito del futuro de las aplicaciones P2P también depende del convencimiento de los usuarios para ser voluntarios de ancho de banda, almacenamiento y recursos de computación a las aplicaciones, lo cual es el desafío del incentivo del diseño [Feldman 2005; Piatek 2008; Aperjis 2008; Liu 2010].

2.1.2 Comunicación de Procesos

Antes de construir tu aplicación de red, también es necesario un conocimiento básico de cómo los programas, que se ejecutan en múltiples sistemas finales, se comunican entre sí. En la jerga de los sistemas operativos, no son de hecho los programas de software los que se comunican, sino los **procesos**. Un proceso puede ser visto como un programa ejecutándose en un sistema final. Cuando los procesos se ejecutan en el mismo sistema final, se comunican entre si utilizando comunicación interproceso, usando reglas de la comunicación interproceso que son gobernadas por el sistema operativo del sistema final. Pero en este libro no nos interesa cómo se comunican los procesos en una mismo host, sino cómo se comunican los procesos que se ejecutan en sistemas *diferentes* (con, potencialmente, distintos sistemas operativos).

Procesos en dos sistemas finales distintos se comunican entre si intercambiando **mensajes** a través de una red de computadoras. El proceso emisor crea y envía mensajes sobre la red; el proceso receptor recibe estos mensajes y, posiblemente, responde enviando mensajes de vuelta. Las aplicaciones de red constan de protocolos de la capa de aplicación que definen el formato y orden en que los procesos intercambian mensajes, así como las acciones que se toman en la transmisión o recepción de un mensaje. En la Figura 2.1 se muestra procesos que se comunican entre sí utilizando la capa de aplicación de la pila de protocolos de cinco capas.

Procesos Cliente y Servidor

Una aplicación de red consta de pares de procesos que envían mensajes entre sí a través de una red. Por ejemplo, en la aplicación Web, un proceso navegador cliente intercambia mensajes con un proceso servidor Web. En un sistema de intercambio de archivos P2P, un archivo se transfiere de un proceso en un par a un proceso en otro par. Para cada par de procesos de comunicación, normalmente denominamos uno de los dos procesos como el **cliente** y el otro proceso como el **servidor**. Con la Web, un navegador es un proceso cliente y un servidor Web es un proceso de servidor. Con el uso compartido de archivos P2P, el par que está descargando el archivo se etiqueta como el cliente, y el par que está cargando el archivo se etiqueta como el servidor.

Es posible que hayas observado que en algunas aplicaciones, tales como en el intercambio de archivos P2P, un proceso puede ser a la vez un cliente y un servidor. De hecho, un proceso en un sistema de intercambio de archivos P2P puede tanto subir como bajar archivos. Sin embargo, en el contexto de cualquier sesión de comunicación dada entre un par de procesos, todavía podemos etiquetar un proceso como el cliente y el otro proceso como el servidor. Definimos los procesos cliente y servidor de la siguiente manera:

*En el contexto de una sesión de comunicación entre un par de procesos, el proceso que inicia la comunicación (es decir, inicialmente se contacta con el otro proceso al principio de la sesión) se etiqueta como **cliente**. El proceso que espera ser contactado para iniciar la sesión es el **servidor**.*

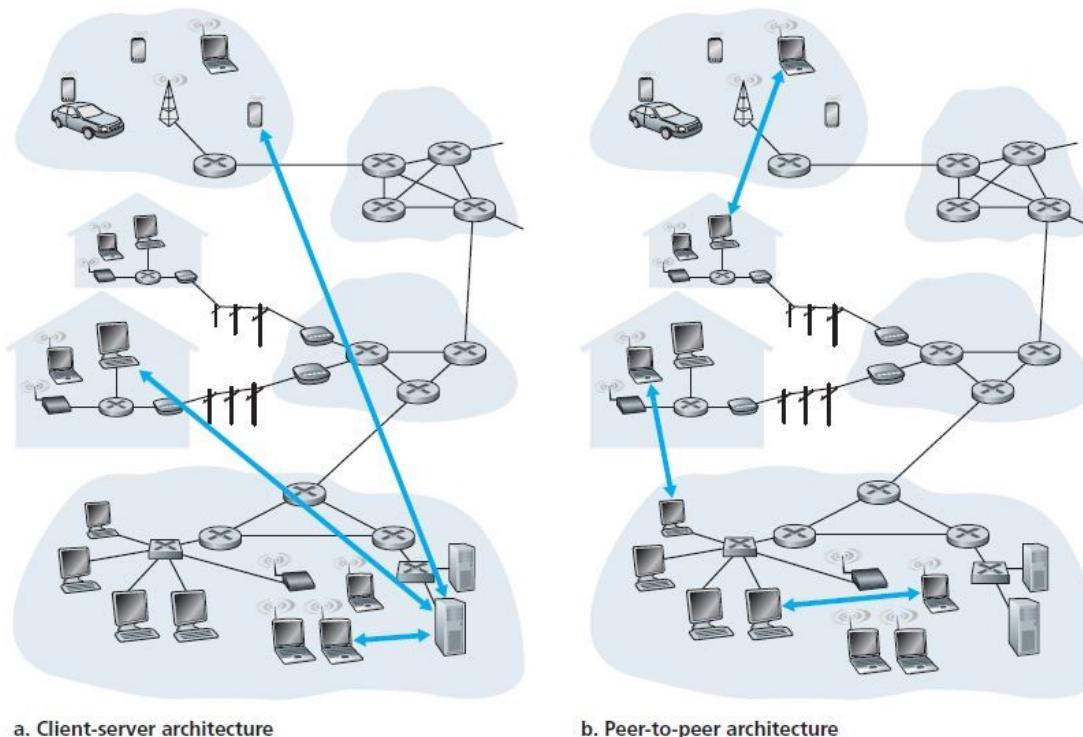


Figura 2.2

En la Web, un proceso navegador inicializa contacto con un proceso servidor Web; por lo tanto, el proceso del navegador es el cliente y el proceso servidor Web es el servidor. En el intercambio de archivos P2P, cuando el Par A solicita al Par B que envíe un archivo específico, el Par A es el cliente y el Par B es el servidor en el contexto de esta sesión de comunicación específica. Cuando no haya confusión, algunas veces también se utiliza la terminología de "el lado del cliente y el lado del servidor de una aplicación". Al final de este capítulo, vamos a desplazarnos a través de un código simple para ambos los lados de cliente y servidor de aplicaciones de red.

La interfaz entre los Procesos y las Redes de Computadoras

Como se observó anteriormente, muchas aplicaciones consisten en un par de procesos comunicándose, con ambos procesos en cada par enviándose mensajes entre ellos. Cualquier mensaje enviado desde un proceso hacia el otro debe de viajar a través de la red subyacente. Un proceso envía mensajes a la red, y recibe mensajes desde la red, a través de una interfaz de software llamada **socket**. Consideremos una analogía para ayudarnos a entender los procesos y los sockets. Un proceso es análogo a una casa y su socket es análogo a su puerta. Cuando un proceso quiere enviar un mensaje a otro proceso en otro host, *empuja* al mensaje por su puerta (socket). Este proceso asume que existe una infraestructura de transporte al otro lado de la puerta que transportará el mensaje a través de Internet hasta la puerta del proceso de destino. Una vez que el mensaje llega al host de destino, el mensaje pasa a través de la puerta (socket) del proceso receptor, que actúa sobre el mensaje.

La Figura 2.3 muestra la comunicación socket entre dos procesos que se comunican a través de Internet. (En la Figura 2.3 se asume que el protocolo de transporte subyacente usado por el proceso es el protocolo de Internet TCP). Como se muestra en la figura, un socket es la interfaz entre la capa de aplicación y la capa de transporte en un host. También es conocida como **API (Interfaz de programación de aplicaciones)** entre la aplicación y la red, dado que el socket es la interfaz de programación con la que se construyen en Internet las aplicaciones de red. El desarrollador de aplicaciones tiene control completo sobre la parte de la capa de aplicación del socket, aunque el control es pequeño sobre la parte de la capa de transporte del mismo. El único control que tiene el desarrollador sobre la capa de transporte es (1) la elección del protocolo de transporte, y (2) quizás la posibilidad de fijar algunos parámetros de la capa de transporte, como el buffer y los tamaños de segmento máximos (que serán vistos en el capítulo 3). Una vez que el desarrollador de aplicaciones elige el protocolo de transporte (si la elección está disponible), la aplicación se construye utilizando los servicios de la capa de transporte proporcionados por dicho protocolo. Se verán los detalles de los sockets en las Secciones 2.7

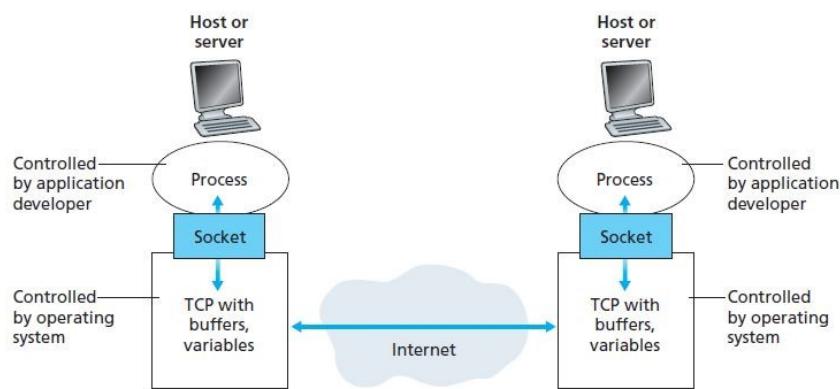


Figura 2.3

Direccionamiento de procesos

Para que un proceso de un host pueda enviar un mensaje a otro proceso de otro host, es necesario que el proceso emisor identifique al proceso receptor. Para identificar al proceso receptor, típicamente se deben especificar dos tipos de información: (1) el nombre o dirección del host, y (2) un identificador que especifique al proceso receptor en el host de destino.

Consideremos en primer lugar las direcciones de host. En Internet, el host de destino se identifica por su **dirección IP**. En el Capítulo 4 se describirán las direcciones IP en gran detalle. De momento, es suficiente saber que la dirección IP es una cantidad de 32 bits que identifica *únivamente* al host (más precisamente, identifica únicamente la interfaz de red que conecta el host a Internet). Además de conocer la dirección del host al que se dirige el mensaje, el proceso emisor debe también identificar el proceso receptor (más específicamente, el socket receptor) ejecutándose en el host. Esta información es necesaria debido a que en general un host puede estar ejecutando muchas aplicaciones de red. En Internet, esta tarea la realiza un **número de puerto** de destino. A los protocolos más conocidos de la capa de aplicación se les han asignado números de puerto específicos. Por ejemplo, un proceso servidor web se identifica con el número de puerto 80. El proceso servidor de correo (que utiliza el protocolo SMTP) es identificado con el número de puerto 25. Se puede encontrar una lista de números de puertos 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 Aplicaciones

Recuerde que un socket es la interfaz entre el proceso de aplicación y el protocolo de transporte. La aplicación en el lado emisor envía el mensaje a través del socket. Al otro lado del socket, el protocolo de transporte tiene la responsabilidad de transmitir el mensaje a través de la red hasta la puerta del proceso receptor.

Muchas redes, incluida Internet, proporcionan más de un protocolo de transporte. Cuando se desarrolla una aplicación se debe elegir uno de los protocolos de transporte disponibles. ¿Cómo se puede hacer esta elección? Probablemente, se deben estudiar los servicios proporcionados por los protocolos de transporte disponibles, y seleccionar aquel protocolo cuyos servicios mejor se adapten a las necesidades de la aplicación. La situación es similar a elegir un transporte de tren o avión entre dos ciudades (por ejemplo, Córdoba y Buenos Aires). Se debe elegir uno u otro, y cada modo de transporte ofrece distintos servicios. (Por ejemplo, el tren ofrece paradas en las ciudades, mientras que el avión ofrece tiempos de transporte más cortos.)

¿Qué servicios podría necesitar una aplicación de red de un protocolo de transporte? En general, se pueden clasificar los requisitos de servicio de las aplicaciones según tres dimensiones: pérdida de datos, ancho de banda, y temporización.

Transferencia Confiable de datos

Como se discutió en el Capítulo 1, los paquetes pueden perderse dentro de las redes de computadoras. Por ejemplo, un paquete puede saturar un buffer en un router, o puede ser descartado por un host o router después de tener algunos bits corruptos. Para muchas aplicaciones, como el correo electrónico, la mensajería instantánea, la transferencia de archivos, el acceso a estaciones remotas, la transferencia de documentos web y las aplicaciones financieras, la pérdida de datos puede tener consecuencias (en el último caso, ¡para el banco o para el cliente!). Entonces, para soportar estas aplicaciones, algo debe de hacerse para garantizar que los datos enviados por un extremo de la aplicación sean entregado correcta y completamente al otro extremo de la aplicación. Si un protocolo provee tal servicio de entrega garantizada de datos, se dice que provee **transferencia confiable de datos**. Un importante servicio que un protocolo de la capa de transporte puede potencialmente proveer a una aplicación es transferencia de datos confiable proceso a proceso. Cuando un protocolo de transporte proveer este servicio, el proceso emisor puede simplemente pasar sus datos hacia el socket y saber con completa confianza que los datos llegarán sin errores al proceso receptor.

Cuando un protocolo de capa de transporte no proporciona transferencia de datos confiable, algunos de los datos enviados por el proceso de envío pueden no llegar al proceso receptor. Esto puede ser aceptable para **aplicaciones tolerantes a pérdidas**, como aplicaciones multimedia de audio/vídeo en tiempo real o almacenamiento de audio/video, que pueden tolerar cierta cantidad de datos perdidos. En estas aplicaciones multimedia, la pérdida de datos puede provocar un pequeño defecto en la reproducción de audio/video, que no supone un perjuicio crucial. Los efectos de tal pérdida sobre la calidad de la aplicación, y la cantidad real de pérdida tolerable de paquetes, dependerán fuertemente de la aplicación y del esquema de codificación utilizado.

Ancho de banda

En el capítulo 1 hemos introducido el concepto de rendimiento disponible, que, en el contexto de una sesión de comunicación entre dos procesos a lo largo de una ruta de red, es la velocidad a la que el proceso emisor puede entregar bits para al proceso receptor. Debido a que otras sesiones van a compartir el ancho de banda a lo largo de la ruta de red, y porque estas otras sesiones estarán entrando y saliendo, el caudal disponible puede variar con el tiempo. Estas observaciones conducen a otro servicio natural que un protocolo de capa de transporte podría proporcionar, a saber, el rendimiento disponible garantizado a una tasa especificada. Con este servicio, la aplicación podría solicitar un rendimiento garantizado de r bits/seg, y el protocolo de transporte entonces garantizaría que el caudal disponible es siempre al menos r bits/seg. Tal rendimiento garantizado de servicio sería de interés para muchas aplicaciones. Por ejemplo, si una aplicación telefónica de internet codifica la voz a 32Kbps, entonces debe ser capaz de enviar datos por la red en esa proporción a la aplicación receptora. Si ese ancho de banda no está disponible, la aplicación debería codificar a una tasa diferente (y recibir el ancho de banda suficiente para soportar esta tasa de codificación) o abandonar, ya que recibir la mitad del ancho de banda requerido no es de ninguna utilidad para esta **aplicación sensible al ancho de banda**. Muchas aplicaciones multimedia actuales son sensibles al ancho de banda, si bien aplicaciones futuras podrían utilizar técnicas adaptativas que codificaran a una tasa que se correspondiera con el ancho de banda disponible en cada momento.

Mientras que las aplicaciones sensibles al ancho de banda necesitan de una cantidad determinada de ancho de banda, las **aplicaciones flexibles** pueden hacer uso de tanto ancho de banda como haya disponible. El correo electrónico, la transferencia de archivos y la transferencias web son todas aplicaciones flexibles. Por supuesto, cuanto mayor sea el ancho de banda, mejor. (Existe un chiste que dice que no se puede ser demasiado rico, demasiado delgado, o tener demasiado ancho de banda.)

Temporización

Un protocolo de capa de transporte también puede proporcionar garantías de temporización. Al igual que con las garantías de rendimiento, las garantías de temporización pueden venir en muchas formas y maneras. Una ejemplo de garantía podría ser que todos los bits que envía hacia el socket lleguen al socket del receptor no más de 100 msec después. Tal servicio podría ser atractivo para aplicaciones interactivas en tiempo real, como la telefonía por Internet, entornos virtuales, teleconferencias, y juegos multijugador, todos los cuales requieren restricciones de tiempo ajustados en la entrega de datos con el fin de ser eficaces. (Véase el Capítulo 7, [Gauthier 1999; Ramjee 1994].) Las largas demoras en la telefonía por Internet, por ejemplo, tienden a provocar pausas en la conversación no naturales; en un juego de varios jugadores o el entorno virtual interactivo, una larga demora entre la toma de una acción y ver la respuesta del entorno (por ejemplo, de otro jugador en la otra punta de una conexión de extremo a extremo) hace que la aplicación se sienta menos realista. Para aplicaciones que no son en tiempo real, un menor

retardo es siempre preferible a un mayor retraso, pero ninguna limitación de ajuste se coloca en los retardos de extremo a extremo.

Seguridad

Por último, un protocolo de transporte puede proveer a una aplicación con 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 de envío, y en el host receptor, el protocolo de capa de transporte puede descifrar los datos antes de la entrega de los datos para el proceso receptor. Tal servicio podría proporcionar confidencialidad entre los dos procesos, incluso si los datos se observan alguna manera entre los procesos emisor y receptor. Un protocolo de transporte también puede proporcionar otros servicios de seguridad, además de la confidencialidad, incluyendo la integridad de datos y autenticación de sistema final, temas que vamos a cubrir en detalle en el capítulo 8.

2.1.4 Los servicios de transporte prestados por Internet

Hasta este punto, hemos considerado qué servicios de transporte que una red informática *podría* proporcionar en general. Ahora vamos a ser más específicos y examinar el tipo de servicios de transporte prestados por Internet. Internet (y, más en general, las redes TCP/IP) proveen dos protocolos de transporte disponibles para las aplicaciones, UDP y TCP. Cuando tu (como un desarrollador de aplicaciones) creas una nueva aplicación de red para Internet, una de las primeras decisiones que debes tomar es si vas a 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 muestra los requisitos de servicio para algunas aplicaciones seleccionadas.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Figura 2.4

Servicios TCP

El modelo de servicio TCP incluye un Servicio orientado a la conexión y un servicio confiable de transferencia de datos. Cuando una aplicación invoca a TCP como su protocolo de transporte, la aplicación recibe los siguientes dos servicios.

- **Servicio orientado a la conexión:** TCP hace que el cliente y servidor intercambien entre si información de control de la capa de transporte *antes* de que comience el flujo de los mensajes del nivel de aplicación. Este procedimiento denominado acuerdo (*handshaking*), alerta al cliente y al servidor, permitiendo que se preparen para una acometida de paquetes. Después de esta fase de acuerdo, se dice que existe una **conexión TCP** entre los sockets de los dos procesos. La conexión es *full-duplex*, en el sentido de que los dos procesos pueden enviarse mensajes entre si sobre la conexión, y al mismo tiempo. Cuando la aplicación deja de enviar mensajes debe romper la conexión. El servicio se denomina *orientado a la conexión*, en vez de servicio de *conexión* (o servicio de circuito virtual) porque los dos procesos están conectados de una forma muy vaga. En el Capítulo 3 se discutirá en detalle el servicio orientado a la conexión y se examinará su implementación.
- **Servicio confiable de transporte:** Los procesos de comunicación se basan en TCP para enviar todos los datos sin errores y en el orden apropiado. Cuando un lado de la aplicación pasa un flujo de bytes a un socket, puede contar con que TCP entregará el mismo flujo de datos al socket receptor, sin pérdida ni duplicación de bytes.

TCP también incluye un mecanismo de control de congestión, un servicio para el bienestar general de Internet más que para el beneficio directo de los procesos que se comunican. El mecanismo TCP de control de congestión regula el proceso emisor (cliente o servidor) cuando la red está congestionada entre el emisor y el receptor. Como se verá en el Capítulo 3, el control de congestión de TCP intenta limitar cada conexión TCP a su cuota justa de ancho de banda de la red.

Servicios UDP

UDP es un protocolo de transporte ligero, sin adornos, con un modelo de servicio minimalista. UDP funciona sin conexión, en el sentido de que no hay acuerdo antes de que los dos procesos comiencen a comunicarse. UDP proporciona un servicio de transferencia de datos no confiable; esto es, cuando un proceso envía un mensaje a un socket UDP, éste *no* garantiza que el mensaje llegue al proceso receptor. Más aún, los mensajes que llegan al proceso receptor, pueden hacerlo desordenados.

UDP no incluye un mecanismo de control de congestión, por lo que el lado emisor de un socket UDP puede dirigir datos a la capa por debajo (la capa de red) a cualquier tasa que desee. (Tenga en cuenta, sin embargo, que el rendimiento real de extremo a extremo puede ser menor que esta tasa debido a la capacidad de transmisión limitada de los enlaces intermedios o debido a la congestión).

Servicios no prestados por protocolos de transporte de Internet

Hemos organizado los servicios del protocolo de transporte a lo largo de cuatro dimensiones: la transferencia de datos confiable, el rendimiento, el tiempo y la seguridad. ¿Cuál de estos servicios son proporcionados por TCP y UDP? Ya hemos señalado que el TCP proporciona una transferencia de datos confiable de extremo a extremo. Y también sabemos que TCP puede mejorar fácilmente en la capa de aplicación con SSL para proporcionar servicios de seguridad. Sin embargo, en nuestra breve descripción de TCP y UDP, estuvo notoriamente ausente cualquier mención de garantías sobre rendimiento o de temporización, servicios *no* proporcionados por los protocolos de transporte de Internet de hoy en día. ¿Esto significa que las aplicaciones sensibles al tiempo tales como la telefonía por Internet no se pueden ejecutar en la Internet de hoy? La respuesta es claramente no (Internet ha sido el anfitrión de aplicaciones sensibles al tiempo durante muchos años). Estas aplicaciones a menudo funcionan bastante bien, ya que han sido diseñadas para hacer frente, en la mayor medida posible, a esta falta de garantía. Investigaremos varios de estos trucos de diseño en el Capítulo 7. Sin embargo, el diseño inteligente tiene sus limitaciones cuando el retraso es excesivo, o el rendimiento de extremo a extremo es limitado. En resumen, la Internet de hoy a menudo puede proporcionar un servicio satisfactorio para aplicaciones sensibles al tiempo, pero no puede ofrecer ninguna garantía de tiempo o en el rendimiento.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figura 2.5

La Figura 2.5 indica los protocolos de transporte empleados por algunas aplicaciones Internet conocidas. Se ve que el correo electrónico, los accesos a terminales remotos, la Web y la transferencia de archivos utilizan TCP. Estas aplicaciones han elegido TCP porque proporciona un servicio de transferencia confiable de datos, garantizando que todos los datos alcanzan finalmente su destino. Dado que las aplicaciones de telefonía por Internet (como Skype) a menudo pueden tolerar alguna pérdida pero requieren una velocidad mínima para que sea efectiva, los desarrolladores de aplicaciones de telefonía por Internet por lo general prefieren ejecutar sus aplicaciones a través de UDP, evitando así el mecanismo de control de congestión de paquetes y los gastos generales del TCP. Pero debido a que muchos firewalls están configurados para bloquear (la mayoría de los tipos de) el tráfico UDP,

las aplicaciones de telefonía por Internet a menudo están diseñados para utilizar TCP como respaldo si falla la comunicación UDP.

2.1.5 Protocolos de nivel de aplicación

Acabamos de aprender que los procesos de red se comunican entre sí mediante el envío de mensajes a sockets. Pero, ¿cómo se estructuran estos mensajes? ¿Cuáles son los significados de los distintos campos en los mensajes? ¿Cuándo los procesos envían los mensajes? Estas preguntas nos llevan al reino de los protocolos de la capa de aplicación. Un **protocolo de capa de aplicación** define cómo los procesos de una aplicación, que se ejecutan en diferentes sistemas finales, pasan los mensajes entre ellos. En particular, un protocolo de capa de aplicación define:

- Los tipos de mensajes que se intercambian, por ejemplo, mensajes de solicitud y mensajes de respuesta
- La sintaxis de los diversos tipos de mensajes, tales como los campos en el mensaje y cómo se delimitan los campos
- La semántica de los campos, es decir, el significado de la información en los campos
- Reglas para determinar cuándo y cómo un proceso envía mensajes y responde a las mensajes

Algunos protocolos de la capa de aplicación se especifican en RFC y por lo tanto son de dominio público. Por ejemplo, el protocolo Web de la capa de aplicación, HTTP (Protocolo de Transferencia de Hipertexto [RFC 2616]), está disponible como un RFC. Si un desarrollador del navegador sigue las reglas del RFC de HTTP, el navegador será capaz de recuperar páginas web de cualquier servidor Web que también ha seguido las reglas del RFC de HTTP. Muchos otros protocolos de capa de aplicación son propietarios e intencionalmente no están disponibles en el dominio público. Por ejemplo, Skype utiliza protocolos propietarios de la capa de aplicaciones.

Es importante distinguir entre aplicaciones de red y protocolos de la capa de aplicación. Un protocolo de capa de aplicación es sólo una pieza de una aplicación de red (no obstante, una pieza muy importante de la aplicación desde nuestro punto de vista!). Veamos un par de ejemplos. La Web es una aplicación cliente-servidor que permite a los usuarios obtener documentos de servidores Web en la demanda. La aplicación web se compone de muchos componentes, incluyendo un estándar para formatos de documento (es decir, HTML), navegadores web (por ejemplo, Firefox y Microsoft Internet Explorer), servidores web (por ejemplo, servidores Apache y Microsoft), y un protocolo de capa de aplicación. El protocolo de capa de aplicación de la web, HTTP, define el formato y la secuencia de mensajes intercambiados entre el navegador y el servidor Web. Por lo tanto, HTTP es sólo una pieza (no obstante, una pieza importante) de la aplicación web. Como otro ejemplo, una aplicación de correo electrónico de Internet también tiene muchos componentes, incluidos los servidores de correo que albergan los buzones de usuario; los clientes de correo (como Microsoft Outlook) que permiten a los usuarios leer y crear mensajes; un estándar para definir la estructura de un mensaje de correo electrónico; y los protocolos de capa de aplicación que definen cómo se transmiten los mensajes entre servidores, cómo se transmiten los mensajes entre servidores y clientes de correo, y cómo el contenido de los encabezados de los mensajes se han de interpretar. El protocolo principal de la capa de aplicación para el correo electrónico es SMTP (Simple Mail Transfer Protocol) [RFC 5321]. Por lo tanto, el principal protocolo de capa de aplicación de correo electrónico, SMTP, es sólo una pieza (no obstante, una pieza importante) de la aplicación de correo electrónico.

2.1.6 Aplicaciones de red tratadas en este libro

Todos los días se están desarrollando nuevas aplicaciones Internet de dominio público y propietarias. En vez de abarcar un gran número de aplicaciones Internet de forma enciclopédica, hemos decidido centrarnos en un número reducido de aplicaciones importantes y conocidas. En este capítulo se describen en detalle cinco aplicaciones populares: la Web, la transferencia de archivos, el correo electrónico, el servicio de directorio, y las aplicaciones P2P. En primer lugar se describe la Web, y no sólo porque sea una aplicación enormemente popular, sino también porque su protocolo de la capa de aplicación, HTTP, es relativamente sencillo, e ilustra muchos de los principios clave de los protocolos de red. Despues se describe la transferencia de archivos, que proporciona un buen contraste con HTTP, y que permite resaltar otros principios adicionales. También se describe el correo electrónico, la primera aplicación Internet abrumadora. Veremos que el correo electrónico moderno hace uso no de uno, sino de diversos protocolos de la capa de aplicación. La cuarta aplicación que trata este capítulo, el sistema de nombres de dominio

(DNS; *Domain Name System*), proporciona un servicio de directorio para Internet. La mayoría de los usuarios no interactúan directamente con el DNS; en lugar de ello, los usuarios invocan al DNS indirectamente por medio de otras aplicaciones (incluyendo la Web, la transferencia de archivos y el correo electrónico). El DNS muestra claramente cómo se puede implementar en Internet una base de datos distribuida. Finalmente, se discuten en este capítulo varias aplicaciones P2P, centrándose en aplicaciones para compartir archivos y servicios de búsqueda distribuidos. En el capítulo 7, vamos a cubrir las aplicaciones multimedia, incluyendo la transmisión de vídeo y voz sobre IP.

2.2 La Web y HTTP

Hasta la década de 1990, Internet era utilizada fundamentalmente por investigadores, académicos y estudiantes universitarios para acceder a host remotos, para transferir archivos desde host locales a remotos y viceversa, para recibir y enviar noticias, y para recibir y enviar correo electrónico. A pesar de que estas aplicaciones eran (y siguen siendo) extremadamente útiles, Internet era básicamente desconocida fuera de las comunidades académicas e investigadoras. Fue entonces, a principios de la década de 1990, cuando una nueva e importante aplicación entró en escena: la World Wide Web [Berners-Lee 1994]. La Web es una aplicación Internet que captó la atención del público general. Cambió drásticamente, y continúa cambiando, la forma de interacción de la gente dentro y fuera de los entornos de trabajo. Hizo que Internet pasara de ser simplemente una de las muchas redes de datos a ser esencialmente la única red de datos.

Tal vez lo que atrae a la mayor parte de los usuarios es que la web funciona *bajo demanda*. Los usuarios reciben lo que quieren, cuando lo quieren. Esto es a diferencia de la radio y la televisión de difusión tradicionales, la cual obliga a los usuarios a sintonizar cuando el proveedor de contenido hace que el contenido esté disponible. Además de estar disponible bajo demanda, la Web tiene muchas otras características maravillosas que las personas aman y aprecian. Es sumamente fácil para cualquier persona hacer que la información esté disponible a través de Internet (todo el mundo puede convertirse en un editor a un costo extremadamente bajo). Los hipervínculos y motores de búsqueda nos ayudan a navegar a través de un océano de sitios Web. Los gráficos estimulan nuestros sentidos. Los formularios, JavaScript, applets de Java, y muchos otros dispositivos nos permiten interactuar con las páginas y sitios. Y la web sirve como plataforma para muchas aplicaciones abrumadoras emergentes después de 2003, como YouTube, Gmail y Facebook.

2.2.1 Introducción a HTTP

El **Protocolo de transferencia de hipertexto (HTTP)**, el protocolo de la capa de aplicación de la Web, está en el corazón de la Web. Está definida en [RFC 1954] y [RFC 2616]. HTTP está implementado en dos programas: un programa cliente y otro servidor. Los programas cliente y servidor, que se ejecutan en sistemas finales diferentes, conversan entre si intercambiando mensajes HTTP. HTTP define la estructura de esos mensajes y cómo se realiza el intercambio entre el cliente y el servidor. Antes de explicar en detalle HTTP, es interesante revisar algo de terminología.

Una **página web** (también denominada documento) consta de objetos. Un **objeto** es simplemente un archivo (como por ejemplo, un archivo HTML, una imagen JPEG, una imagen GIF, un applet Java, una parte de audio, etc.) que es direccionable por un único URL. La mayoría de las páginas web están formadas por un **archivo HTML base** y diversos objetos referenciados. Por ejemplo, si una página web contiene texto HTML y cinco imágenes JPEG, entonces la página web tiene seis objetos: el archivo HTML base y las cinco imágenes. El archivo HTML base referencia a los otros objetos de la página con los URL de los objetos. Cada URL tiene dos componentes: el nombre de host del servidor que alberga el objeto, y el nombre de la ruta del objeto. Por ejemplo, el URL

`http://www.algunaEscuela.edu/algunDepartamento/imagen.gif`

tiene como nombre de host `www.algunaEscuela.edu`, y como nombre de ruta `/algunDepartamento/imagen.gif`. Debido a que los **navegadores web** (como Internet Explorer y Firefox) implementan el lado cliente de HTTP, en el contexto de la Web, intercambiaremos el uso de las palabras *navegador* y *cliente*. Los **servidores web** implementan el lado servidor de HTTP, albergan objetos Web, cada uno direccionable por una URL. Entre los servidores web más populares se encuentran Apache y Microsoft Internet Information Server.

HTTP define cómo los clientes web (por ejemplo, navegadores) demandan páginas web, y cómo los servidores transfieren estas páginas web a los clientes. Posteriormente se describe en detalle la interacción entre el cliente y el servidor, si bien la idea general está descrita en la Figura 2.6. Cuando un usuario pide una página web (por ejemplo, pulsa sobre un hipervínculo), el navegador envía al servidor un mensaje HTTP de petición de los objetos de la página.

HTTP utiliza TCP como protocolo de transporte subyacente (en vez de correr sobre UDP). En primer lugar, el cliente HTTP inicia una conexión TCP con el servidor. Una vez establecida la conexión, los procesos del navegador y del servidor acceden a TCP por medio de sus interfaces de socket. Como se describió 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 HTTP de petición a su interfaz de socket y recibe mensajes HTTP de respuesta de su interfaz de socket. De forma similar, el servidor HTTP recibe mensajes de petición de su interfaz de socket y envía mensajes de respuesta a su interfaz de socket. Una vez que el cliente envía un mensaje a su interfaz de socket, éste está *frente a las manos del cliente*, y está "en las manos" de TCP. Recuerde de la Sección 2.1 que TCP proporciona a HTTP un servicio confiable de transferencia de datos. Esto implica que cada mensaje HTTP de petición emitido por un proceso cliente, eventualmente llega sin modificaciones al servidor; igualmente, cada mensaje HTTP de respuesta emitido por el proceso servidor, eventualmente llega intacto al cliente. Aquí vemos una de las mayores ventajas de la arquitectura de capas: HTTP no necesita preocuparse de la pérdida de datos o de los detalles de cómo TCP recupera la pérdida o reordena los datos en la red. Este es el trabajo de TCP y de los protocolos de las capas inferiores de la pila de los protocolos.

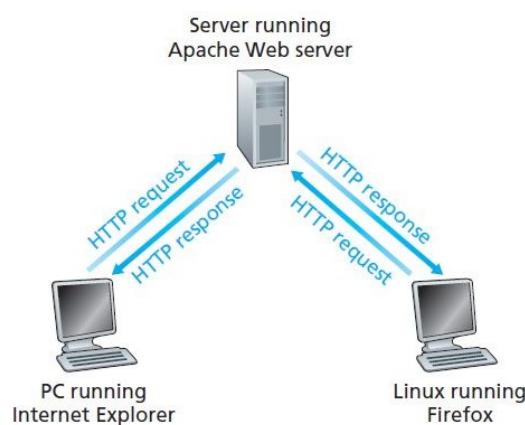


Figura 2.6

Es importante observar que el servidor envía los archivos pedidos sin almacenar ninguna información de estado sobre el cliente. Si un cierto cliente pide dos veces el mismo objeto en un periodo de tiempo de unos pocos segundos, el servidor no responde que acaba de servir el objeto al cliente, sino que reenvía el objeto, ya que ha olvidado por completo lo que hizo anteriormente. Se dice que HTTP es un **protocolo sin estado**, porque el servidor HTTP no guarda información sobre los clientes. También destacamos que la Web utiliza la arquitectura de aplicaciones cliente-servidor, tal como se describe en la Sección 2.1. Un servidor web está siempre encendido, con una dirección IP fija, y atiende peticiones de los potencialmente millones de diferentes navegadores.

2.2.2 Conexiones no persistentes y persistentes

En muchas aplicaciones de Internet, el cliente y el servidor se comunican por un período prolongado de tiempo, con el cliente realizando una serie de peticiones y el servidor respondiendo a cada una de las solicitudes. Dependiendo de la aplicación y de cómo se está utilizando la aplicación, la serie de solicitudes podrá hacerse de forma back-to-back, periódicamente, a intervalos regulares, o de forma intermitente. Cuando esta interacción cliente-servidor se lleva a cabo a través de TCP, el desarrollador de aplicaciones tiene que tomar una decisión importante: cada par solicitud/respuestas debe ser enviado a través de una conexión TCP independiente, o deberían todas las solicitudes y sus respuestas correspondientes ser enviadas a través de la misma conexión TCP? En el primer enfoque, se dice que la aplicación utiliza **conexiones no persistentes**; y en el último enfoque, **conexiones persistentes**. Para tener una profunda comprensión de este problema de diseño, vamos a examinar las ventajas y desventajas de las conexiones persistentes en el contexto de una aplicación específica, es decir, HTTP, que puede utilizar ambas, conexiones

no persistentes y conexiones persistentes. A pesar de que HTTP utiliza conexiones persistentes en el modo predefinido, los clientes HTTP y servidores pueden ser configurados para usar conexiones no persistentes en su lugar.

HTTP con conexiones no persistentes

Revisemos los pasos de transferir una página web desde el servidor al cliente para el caso de las conexiones no persistentes. Supongamos que la página contiene un archivo base HTML y 10 imágenes JPEG, y que todos de estos 11 objetos residen en el mismo servidor. Supongamos, además, la dirección del archivo HTML base es

`http://www.algunEscuela.edu/algúnDepartamento/home.index`

Esto es lo que ocurre:

1. El cliente HTTP inicia la conexión TCP con el servidor `www.algunaEscuela.edu` sobre el número de puerto 80, que es el puerto por defecto para HTTP.
2. El cliente HTTP envía al servidor un mensaje HTTP de petición a través del socket asociado con la conexión TCP establecida en el paso 1. El mensaje de petición incluye el nombre de ruta `/algúnDepartamento/home.index`. (Posteriormente se discutirán en detalle los mensajes HTTP.)
3. El servidor HTTP recibe el mensaje de petición a través del socket asociado con la conexión, recupera el objeto `/algúnDepartamento/home.index` de su almacenamiento (RAM o disco), encapsula el objeto en el mensaje HTTP de respuesta, y envía el mensaje de respuesta al cliente a través del socket.
4. El servidor HTTP dice a TCP que cierre la conexión TCP. (Pero TCP no finaliza en realidad la conexión hasta que se sepá con seguridad que el cliente ha recibido intacto el mensaje de respuesta.)
5. El cliente HTTP recibe el mensaje de respuesta. Finaliza la conexión TCP. 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 de los diez objetos JPEG.
6. Se repiten los cuatro primeros pasos para cada uno de los objetos JPEG referenciados.

Cuando el navegador recibe la página web, la muestra al usuario. Dos navegadores distintos podrían interpretar (esto es, mostrar al usuario) la página web de forma algo distinta. HTTP no tiene nada que ver en cómo el cliente interpreta la página web. Las especificaciones HTTP ([RFC 1945] y [RFC 2616]) sólo definen el protocolo de comunicación entre los programas HTTP cliente y servidor.

Los pasos anteriores utilizan conexiones no persistentes, porque cada una de las conexiones TCP es cerrada después de que el servidor envíe el objeto (la conexión no persiste para los otros objetos). Observe que cada conexión TCP transporta exactamente un mensaje de petición y uno de respuesta. Por tanto, en este ejemplo, cuando un usuario solicita la página web, se generan 11 conexiones TCP.

En los pasos descritos anteriormente, se ha dejado intencionadamente sin definir si el cliente obtiene los diez JPEG en diez conexiones TCP en serie, o si algunos de los JPEG se obtienen en conexiones TCP paralelas. De hecho, los usuarios pueden configurar los navegadores modernos para controlar el grado de paralelismo. En el modo por defecto, muchos navegadores abren de cinco a diez conexiones TCP paralelas, y cada una de estas conexiones maneja una transacción de solicitud-respuesta. Si el usuario lo prefiere, se puede establecer el número máximo de conexiones en uno, con lo que las diez conexiones se establecerían en serie. Como se verá en el siguiente capítulo, la utilización de conexiones paralelas reduce el tiempo de respuesta.

Antes de continuar, hagamos un cálculo estimado del tiempo que transcurre desde que el cliente pide el archivo HTML base hasta que éste es recibido completamente por el cliente. Para este propósito, se define el **tiempo de ida y vuelta (RTT; round-trip time)**, que es el tiempo que tarda un pequeño paquete en ir desde el cliente hasta el servidor y después de vuelta al cliente. El RTT incluye los retardos de propagación de paquetes, los de las colas en los routers y switches intermedios, y los de procesamiento de paquetes. (Estos retardos son discutidos en la Sección 1.4.) A continuación se considera lo que ocurre cuando un usuario pulsa sobre 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 *acuerdo en tres vías*: el cliente envía un pequeño segmento TCP al servidor, el servidor reconoce

y responde con un pequeño segmento TCP, y, finalmente, el cliente hace un reconocimiento de vuelta al servidor. Transcurre un RTT tras las dos primeras partes del acuerdo en tres fases. Después de completar las dos primeras partes del acuerdo, el cliente envía el mensaje HTTP de petición en combinación con la tercera parte del acuerdo (el reconocimiento) sobre la conexión TCP. Una vez que el mensaje de petición llega al servidor, éste envía el archivo HTML sobre la conexión TCP. Esta petición/respuesta HTTP consume otro RTT. Por tanto, aproximadamente, el tiempo total de respuesta es de dos RTT más el tiempo de transmisión del archivo HTML en el servidor.

HTTP con Conexiones persistentes

Las conexiones no persistentes muestran algunos defectos. En primer lugar, se debe establecer y mantener una nueva conexión para *cada uno de los objetos pedidos*. Para cada una de estas conexiones, se deben reservar búferes y variables TCP que deben ser almacenados tanto en el cliente como en el servidor. Esto supone una carga importante sobre el servidor web, que puede estar sirviendo simultáneamente peticiones de cientos de clientes distintos. En segundo lugar, como se acaba de describir, cada objeto sufre un retardo de entrega de dos RTT: una para establecer la conexión TCP, y otro para la solicitud y recepción del objeto.

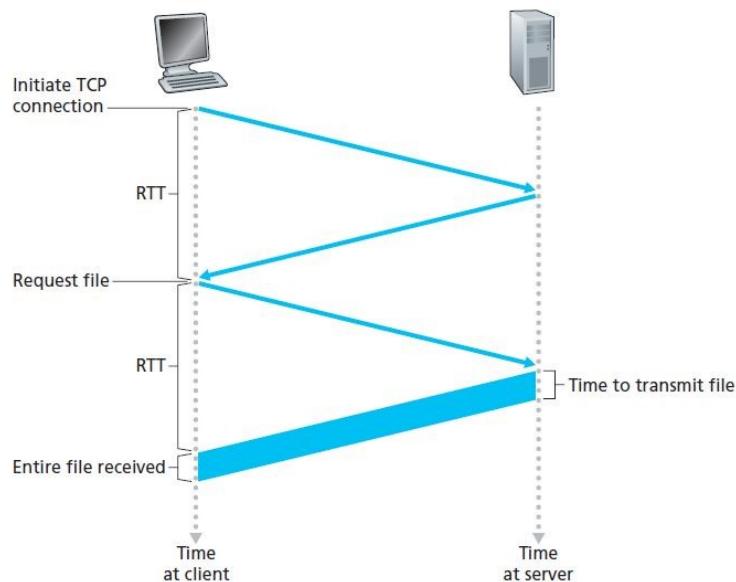


Figura 2.7

Con las conexiones persistentes, el servidor deja abierta la conexión TCP después de enviar una respuesta. Las siguientes peticiones y respuestas entre el mismo cliente y servidor pueden ser enviadas sobre la misma conexión. En particular, una página web completa (en el ejemplo anterior, el archivo HTML base y las diez imágenes) puede ser enviada sobre una única conexión TCP persistente. Más aun, múltiples páginas web que residen en el mismo servidor pueden ser enviadas desde el servidor hasta un mismo cliente sobre una única conexión TCP persistente. Estas solicitudes de objetos se pueden hacer back-to-back, sin esperar respuestas a las solicitudes pendientes (pipelining). Típicamente, el servidor HTTP cierra una conexión cuando no se utiliza durante un cierto tiempo (un intervalo de tiempo de espera configurable). Cuando el servidor recibe las peticiones back-to-back, envía los objetos back-to-back. El modo por defecto de HTTP utiliza conexiones persistentes con pipeline. Vamos a comparar cuantitativamente el rendimiento de las conexiones no persistentes y persistentes en los problemas de la tarea de los capítulos 2 y 3. También se les anima a ver [Heidemann 1997; Nielsen 1997].

2.2.3 Formato del mensaje HTTP

La especificación de HTTP ([RFC 1945] y [RFC 2616]) incluyen las definiciones de los formatos de los mensajes HTTP. Existen dos tipos de mensajes HTTP: los mensajes de solicitud, y los mensajes de respuesta: ambos se describen a continuación.

Mensaje HTTP de solicitud

A continuación aparece un típico mensaje HTTP de petición:

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

Podemos aprender mucho echando un vistazo de cerca a este simple mensaje de solicitud. En primer lugar, vemos que el mensaje está escrito en texto ASCII ordinario, por lo que puede ser leído por persona con conocimientos informáticos. En segundo lugar, vemos que el mensaje se compone de cinco líneas, cada una seguida por un retorno de carro y un avance de línea. La última línea es seguida por un retorno de carro y salto de línea adicional. Aunque este mensaje de petición particular tiene cinco líneas, un mensaje de solicitud puede tener muchas más líneas o tan sólo una línea. La primera línea de un mensaje de solicitud HTTP se llama **línea de solicitud**; las líneas posteriores se llaman **líneas de cabecera**. La línea de solicitud tiene tres campos: el campo método, el campo URL, y el campo de versión HTTP. El campo método puede asumir varios valores diferentes, incluyendo GET, POST, HEAD, PUT y DELETE. La gran mayoría de los mensajes de solicitud HTTP utiliza el método GET. El método GET se utiliza cuando el navegador solicita un objeto, con el objeto solicitado identificado en el campo URL. En este ejemplo, el navegador solicita el objeto /somedir/page.html. La versión se explica por sí sola; en este ejemplo, el navegador implementa la versión HTTP/1.1.

Veamos ahora las líneas de cabecera en el ejemplo. La línea de cabecera Host : www.algunaescuela.edu especifica el host en el que reside el objeto. Se podría pensar que esta línea de cabecera es innecesaria, pues ya existe una conexión TCP con el host. Pero, como veremos en la Sección 2.2.5, la información proporcionada por la línea de encabezado de host es requerido por las memorias caché del proxy web. Con la inclusión de líneas de cabecera Connection:close, el navegador está diciendo al servidor que no quiere molestar con conexiones persistentes; quiere 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 realizando la petición al servidor. Aquí el agente de usuario es Mozilla/5.0, un navegador Firefox. Esta línea de cabecera es útil porque el servidor en realidad puede enviar diferentes versiones de un mismo objeto para diferentes tipos de aplicaciones de usuario. (Cada una de las versiones es direccionada por la misma URL.) Por último, el encabezado Accept-language : indica que el usuario prefiere recibir una versión francesa del objeto, si existe tal objeto en el servidor; de lo contrario, el servidor debe enviar su versión por defecto. La cabecera Accept-Language : es sólo una de muchas cabeceras de negociación de contenido disponibles en HTTP.

Habiendo examinado un ejemplo, veamos ahora el formato general de un mensaje de solicitud, como se muestra en la Figura 2.8. Vemos que el formato general sigue de cerca el ejemplo anterior. Puedes haber notado, sin embargo, que después de las líneas de cabecera (y el retorno de carro y salto de línea adicional) hay un "cuerpo de la entidad." El cuerpo de entidad está vacío con el método GET, sin embargo se utiliza con el método POST. Un cliente HTTP a menudo utiliza el método POST cuando el usuario rellena un formulario, por ejemplo, cuando un usuario proporciona las palabras de búsqueda de un motor de búsqueda. Con un mensaje POST, el usuario sigue solicitando una página web desde el servidor, pero el contenido específico de la página Web dependerá de lo que el usuario ingresó en los campos del formulario. Si el valor del campo método es POST, entonces el cuerpo de la entidad contiene lo que el usuario ingresó en los campos del formulario.

Seríamos negligentes si no mencionamos que una solicitud generada con un formulario no necesariamente utiliza el método POST. En su lugar, los formularios HTML a menudo usan el método GET e incluyen los datos introducidos (en los campos de formulario) en la dirección URL solicitada. Por ejemplo, si un formulario utiliza el método GET, tiene dos campos, y las entradas a los dos campos son monos y bananas, la URL tendrá la estructura www.algunositio.com/animalsearch?monos&bananas. En su navegar del día a día en la Web, es probable que haya notado URL prolongados de este tipo.

El método HEAD es similar al método GET. Cuando un servidor recibe una petición con el método HEAD, responde con un mensaje de HTTP pero deja fuera el objeto solicitado. Los desarrolladores de aplicaciones a menudo utilizan el método HEAD para la depuración. El método PUT se utiliza a menudo en combinación con herramientas

de publicación Web. Permite al usuario añadir un objeto a una ruta específica (directorio) en un servidor web específico. El método PUT también es utilizado por las aplicaciones que necesitan subir objetos a los servidores Web. El método DELETE permite a un usuario o una aplicación, eliminar un objeto en un servidor Web.

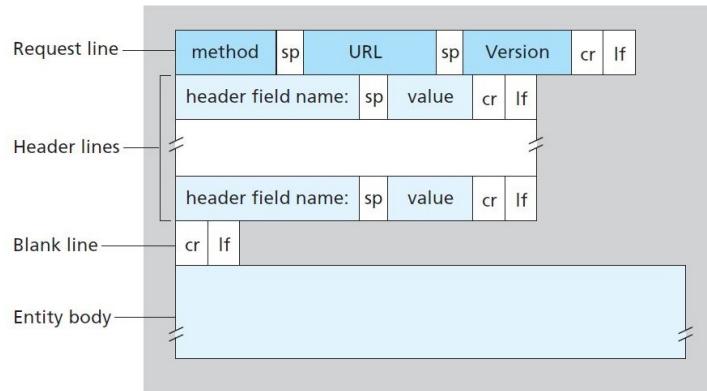


Figura 2.8

Mensaje de respuesta HTTP

A continuación ofrecemos un típico mensaje de respuesta HTTP. Este mensaje de respuesta podría ser la respuesta al mensaje de solicitud del ejemplo que acabamos de discutir.

```

HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
  
```

Echamos una mirada cuidadosa a este mensaje de respuesta. Tiene tres secciones: una **línea de estado** inicial, seis **líneas de cabecera**, y luego el **cuerpo de entidad**. El cuerpo de entidad es la sustancia del mensaje -contiene el objeto solicitado en sí mismo (representado por los data data data data data ...). La línea de estado tiene tres campos: el campo de versión del protocolo, un código de estado y un mensaje de estado correspondiente. En este ejemplo, la línea de estado indica que el servidor está utilizando HTTP/1.1 y que todo está bien (es decir, el servidor ha sido encontrado, y está enviando, el objeto solicitado).

Ahora vamos a ver las líneas de cabecera. El servidor utiliza la línea de cabecera `Connection: close` para decirle al cliente que se va a cerrar la conexión TCP después de enviar el mensaje. La línea de cabecera `Date:` indica la hora y fecha en que la respuesta HTTP fue creada y enviada por el servidor. Tenga en cuenta que este no es el momento en que se creó el objeto o modificó por última vez; es el tiempo cuando el servidor recupera el objeto de su sistema de archivos, se inserta el objeto en el mensaje de respuesta, y se envía el mensaje de respuesta. 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:` en el mensaje de petición HTTP. La cabecera `Last-Modified:` indica la hora y fecha en que se creó el objeto o modificó por última vez. La cabecera `Last-Modified:`, que pronto vamos a cubrir con más detalle, es crítica para el almacenamiento en caché de objetos, tanto en el cliente local y en red de servidores de caché (también conocidos como servidores proxy). La línea de cabecera `Content-Length:` indica el número de bytes en el objeto que están siendo enviados. El encabezado `Content-Type:` indica que el objeto en el cuerpo de la entidad es texto HTML. (El tipo de objeto se indica oficialmente por el encabezado `Content-Type:` y no por la extensión de archivo.)

Habiendo examinado un ejemplo, vamos a examinar ahora el formato general de un mensaje de respuesta, que se muestra en la Figura 2.9. Este formato general del mensaje de respuesta coincide con el ejemplo anterior de un mensaje de respuesta. Digamos algunas palabras adicionales acerca de los códigos de estado y sus frases. El código de estado y su frase asociada indican el resultado de la solicitud. Algunos códigos de estado más comunes y frases asociadas incluyen:

- 200 OK: solicitud tuvo éxito y la información se devuelve en la respuesta.
- 301 Moved Permanently: El objeto solicitado ha sido trasladado de manera permanente; la nueva URL se especifica en la cabecera `Location:` del mensaje de respuesta. El software de cliente recuperará automáticamente la nueva URL.
- 400 Bad Request: Este es un código de error genérico que indica que la solicitud no puede ser entendida por el servidor.
- 404 Not Found: El documento solicitado no existe en este servidor.
- 505 HTTP Version Not Supported: La versión del protocolo HTTP solicitada no está soportada por el servidor.

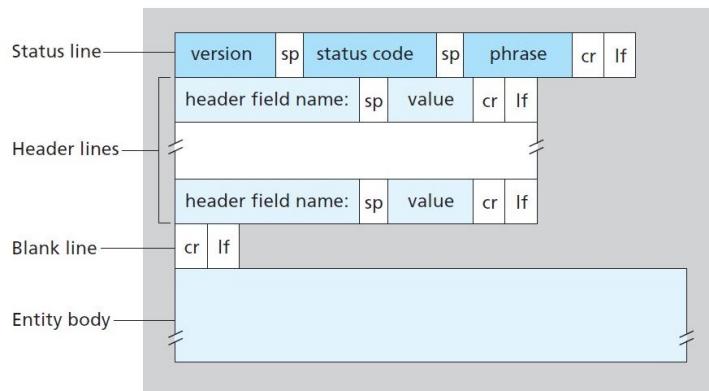


Figura 2.9

¿Cuánto te gustaría ver un mensaje de respuesta HTTP real? Esto es muy recomendable y muy fácil de hacer! En primer lugar envía un mensaje Telnet a tu servidor Web favorito. Luego escribe un mensaje de solicitud de una línea para un objeto que esté alojado en el servidor. Por ejemplo, si tienes acceso a un símbolo del sistema, escribe:

```
telnet cis.poly.edu 80
GET / Ross/HTTP/1.1
HOST: cis.poly.edu
```

(Pulse el retorno de carro dos veces después de escribir la última línea.) Esto abre una conexión TCP con el puerto 80 del host `cis.poly.edu` y luego envía el mensaje de petición HTTP. Deberías ver un mensaje de respuesta que incluye el archivo HTML base de la página principal del profesor Ross. Si lo que queremos es ver las líneas de mensajes HTTP y no recibir el objeto en sí, reemplace `GET` con `HEAD`. Por último, sustituya `/ ross/` con `/ banana/` y observa qué tipo de mensaje de respuesta se obtiene.

En esta sección discutimos una serie de líneas de cabecera que se pueden utilizar dentro de una petición HTTP y sus mensajes de respuesta. La especificación HTTP define muchas, muchas más líneas de cabeceras que pueden ser insertados por los navegadores, servidores Web, servidores de caché y la red. Hemos cubierto sólo un pequeño número de la totalidad de líneas de cabecera. Vamos a cubrir unas pocas más abajo y otro número pequeño cuando discutamos el almacenamiento en caché Web de la red en la Sección 2.2.5. Una discusión altamente legible y completa del protocolo HTTP, incluyendo sus cabeceras y códigos de estado, se da en [Krishnamurthy 2001].

¿Cuándo un navegador decide qué líneas de cabecera incluir en un mensaje de petición? ¿Cómo decide un servidor Web qué líneas de cabecera incluir en un mensaje de respuesta? Un navegador va a generar líneas de cabecera en función del tipo y versión del navegador (por ejemplo, un navegador HTTP/1.0 no generará ninguna línea de cabecera 1.1), la configuración de usuario del navegador (por ejemplo, el idioma preferido), y si actualmente el navegador tiene una versión del objeto almacenada, pero posiblemente fuera de fecha. Los servidores Web se comportan de manera similar: Existen diferentes productos, versiones y configuraciones, todas las cuales influencia que líneas de cabecera se incluyen en los mensajes de respuesta.

2.2.4 Interacción Usuario-Servidor: Cookies

Ya hemos mencionado que un servidor HTTP no tiene estado. Esto simplifica el diseño del servidor y ha permitido a los ingenieros desarrollar servidores Web de alto rendimiento que pueden manejar miles de conexiones TCP simultáneas. Sin embargo, a menudo es deseable para un sitio Web identificar a los usuarios, ya sea porque el servidor desea restringir el acceso de usuarios o porque quiere servir contenido como función de la identidad del usuario. A estos efectos, HTTP utiliza cookies. Las cookies, que se definen en [RFC 6265], permiten a los sitios realizar un seguimiento de los usuarios. La mayoría de los principales sitios web comerciales utilizan cookies en la actualidad.

Como se muestra en la Figura 2.10, la tecnología de cookies tiene 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 petición HTTP; (3) un archivo de cookies mantenido en el sistema del usuario final y gestionado por el navegador del usuario; y (4) una base de datos back-end en el sitio Web. Utilizando la figura 2.10, vamos a examinar a través de un ejemplo cómo funcionan las cookies. Supongamos que Susana, que siempre tiene acceso a la Web utilizando Internet Explorer de su ordenador de casa, contacta Amazon.com por primera vez. Supongamos que en el pasado ya ha visitado el sitio de eBay. Cuando la solicitud entra en el servidor Web de Amazon, el servidor crea un número de identificación único y crea una entrada en su base de datos back-end que está indexado por el número de identificación. El servidor Web de Amazon a continuación, responde al navegador de Susana, incluyendo en la respuesta HTTP un encabezado `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 al archivo cookie especial que gestiona. Esta línea incluye el nombre de host del servidor y el número de identificación en la cabecera `Set-Cookie:`. Tenga en cuenta que el archivo de cookies ya tiene una entrada para eBay, ya que Susana ha visitado ese sitio en el pasado. Como Susana sigue navegando por el sitio de Amazon, cada vez que se solicita una página Web, su navegador consulta a su archivo de cookies, extrae su número de identificación de este sitio, y pone una línea de cabecera cookie que incluye el número de identificación de la petición HTTP. En concreto, cada una de sus peticiones HTTP al servidor de Amazon incluye la línea de cabecera:

`Cookie: 1678`

De esta manera, el servidor de Amazon es capaz de realizar un seguimiento de la actividad de Susana en el sitio de Amazon. Aunque el sitio web de Amazon no tiene por qué conocer el nombre de Susana, sabe exactamente qué páginas el usuario 1678 visitó, en qué orden y en qué momento! Amazon utiliza cookies para facilitar su servicio de carrito de compra -Amazon puede mantener una lista de todas las reservas de compra de Susana, para que pueda pagar por ellos colectivamente al final de la sesión.

Si Susan vuelve al sitio de Amazon, por ejemplo, una semana más tarde, su navegador continuará poniendo la línea de cabecera `Cookie: 1678` en los mensajes de solicitud. Amazon también recomienda productos a Susans sobre la base de las páginas Web que ha visitado en Amazon en el pasado. Si Susans también se registra a sí misma con Amazon -proporcionando el nombre completo, dirección de correo electrónico, dirección postal, y tarjetas de crédito -Amazon puede entonces incluir esta información en su base de datos, asociando el nombre de Susan con su número de identificación (y todas las páginas que ha visitado en el lugar en el pasado!). Así es como Amazon y otros sitios de comercio electrónico proporcionan "compra con un clic", cuando Susan decide comprar un artículo durante una visita posterior, no necesita volver a introducir su nombre, número de tarjeta de crédito, o dirección.

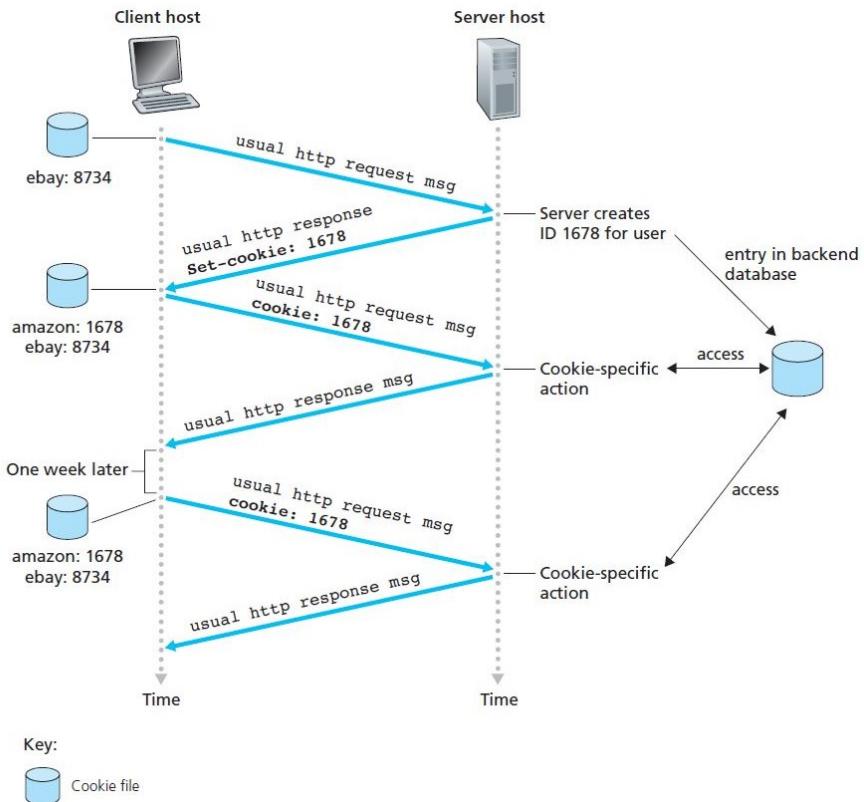


Figura 2.10

De esta discusión vemos que las cookies se pueden utilizar para identificar a un usuario. La primera vez que un usuario visita un sitio, el usuario puede proporcionar una identificación de usuario (posiblemente su nombre). Durante las sesiones subsiguientes, el navegador pasa una cabecera de cookie al servidor, identificando de este modo el usuario al servidor. Las cookies se pueden utilizar de este modo para crear una capa de sesión de usuario en la parte superior de HTTP sin estado. Por ejemplo, cuando un usuario inicia sesión en una aplicación de correo electrónico basado en web (como Hotmail), el navegador envía información de cookie al servidor, lo que permite el servidor para identificar al usuario a lo largo de la sesión del usuario con la aplicación.

Aunque las cookies a menudo simplifican la experiencia de compra en Internet para el usuario, son controvertidas, ya que también pueden ser consideradas como una invasión de la privacidad. Como acabamos de ver, usando una combinación de cookies e información de cuenta proporcionada por el usuario, una página web puede aprender mucho acerca de un usuario y potencialmente vender esta información a un tercero. Cookie Central [Cookie Central 2012] incluye una amplia información sobre la controversia de las cookie.

2.2.5 Web Caching (Almacenamiento Web)

Una **caché Web** -también llamado un **servidor de proxy**- es una entidad de red que satisface las peticiones HTTP en el nombre de un servidor Web de origen. El caché web tiene su propio almacenamiento en disco y guarda copias de los objetos recientemente solicitados en este almacenamiento. Como se muestra en la Figura 2.11, el navegador de un usuario se puede configurar de modo que todas las solicitudes HTTP de los usuarios se dirigen primero a la memoria caché Web. Una vez que se ha configurado un navegador, cada solicitud del navegador de un objeto se dirige en primer lugar a la caché web. A modo de ejemplo, supongamos que un navegador solicita el objeto <http://www.someschool.edu/campus.gif>. Esto es lo que sucede:

1. El navegador establece una conexión TCP con la Web caché y envía una solicitud HTTP para el objeto de la Web caché.
2. La Web caché comprueba para ver si tiene una copia del objeto almacenado localmente. Si lo hace, devuelve el objeto dentro de un mensaje de respuesta HTTP al navegador del cliente.

3. Si la Web caché no tiene el objeto, la memoria de la Web caché abre una conexión TCP al servidor de origen, es decir, a `www.someschool.edu`. La Web caché envía una solicitud HTTP para el objeto en la conexión TCP caché-con-servidor. Después de recibir esta solicitud, el servidor de origen envía el objeto dentro de una respuesta HTTP a la Web caché.
4. Cuando la Web caché recibe el objeto, se almacena una copia en su almacenamiento local y envía una copia, dentro de un mensaje de respuesta HTTP, al navegador del cliente (por la conexión TCP existente entre el navegador del cliente y la Web caché).

Tenga en cuenta que una caché es tanto un servidor como un cliente al mismo tiempo. Cuando se recibe peticiones de respuestas y envía a un navegador, es un servidor. Cuando se envía peticiones y recibe respuestas de un servidor de origen, es un cliente.

Normalmente, un caché web es comprado e instalado por un ISP. Por ejemplo, una universidad puede instalar una memoria caché en la red del campus y configurar todos los navegadores del campus para que apunten a la caché. O uno de los principales ISP residencial (como AOL) podría instalar uno o más cachés en su red y configurar previamente sus navegadores embarcados para apuntar a los cachés instalados.

El almacenamiento en caché web ha visto el despliegue en Internet por dos razones. En primer lugar, una memoria caché de Web pueden reducir sustancialmente el tiempo de respuesta para una petición de cliente, particularmente si el cuello de botella del ancho de banda entre el cliente y el servidor de origen es mucho menor que el cuello de botella del ancho de banda entre el cliente y la memoria caché. Si hay una conexión de alta velocidad entre el cliente y la memoria caché, como lo es a menudo, y si la caché tiene el objeto solicitado, entonces el caché será capaz de entregar el objeto rápidamente al cliente. En segundo lugar, como pronto ilustraremos con un ejemplo, cachés web pueden reducir sustancialmente el tráfico en el enlace de acceso de una institución a Internet. Al reducir el tráfico, la institución (por ejemplo, una empresa o una universidad) no tiene que actualizar el ancho de banda rápidamente, reduciendo así los costos. Por otra parte, cachés web pueden reducir sustancialmente el tráfico Web en Internet en su conjunto, mejorando así el rendimiento para todas las aplicaciones.

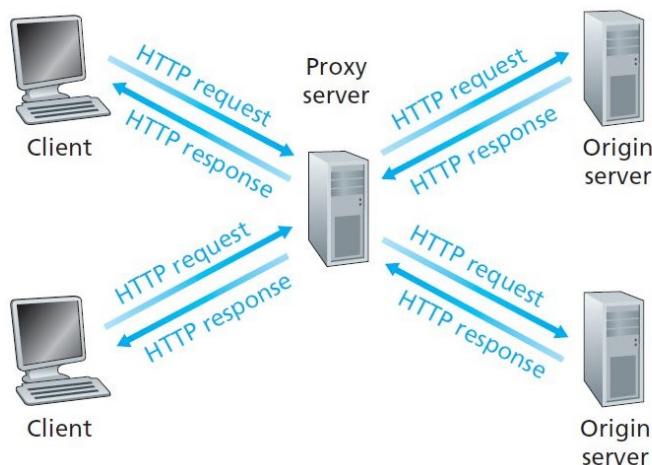


Figura 2.11

Para obtener una comprensión más profunda de los beneficios de las cachés, vamos a considerar un ejemplo en el contexto de la Figura 2.12. Esta figura muestra dos redes -la red institucional y el resto de Internet pública. La red institucional es una LAN de alta velocidad. Un router en la red institucional y un router en Internet están conectados por un enlace de 15 Mbps. Los servidores de origen se adjuntan a Internet, pero se encuentran en todo el mundo. Supongamos que el tamaño medio del objeto es de 1 Mbits y que la tasa promedio de peticiones de los navegadores de la institución para los servidores de origen es de 15 solicitudes por segundo. Supongamos que los mensajes de solicitud HTTP son despreciables y por lo tanto no crean ningún tráfico en las redes o en el enlace de acceso (desde el router a router de Internet institucional). También suponga que la cantidad de tiempo que se tarda desde que el enrutador en el lado de Internet del enlace de acceso en la Figura 2.12 envía una petición HTTP (dentro de un datagrama IP) hasta que recibe la respuesta (normalmente dentro de muchos datagramas IP) es de dos segundos en promedio. De manera informal, nos referimos a este último retraso, como el "retardo de Internet."

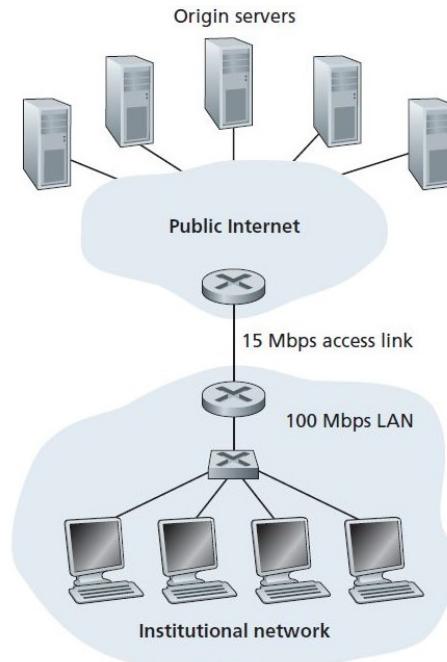


Figura 2.12

El tiempo total de respuesta - esto es, el tiempo desde que el navegador solicita un objeto hasta que lo recibe- es la suma del retardo de LAN, el retardo de acceso (es decir, el retardo entre los dos routers), y el retardo de Internet. Ahora vamos a hacer un cálculo muy crudo para estimar este retraso. La intensidad del tráfico en la LAN (ver Sección 1.4.2) es

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

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

$$(15 \text{ solicitudes/seg}) * (1 \text{ Mbits/solicitud}) / (15 \text{ Mbps}) = 1$$

Una intensidad de tráfico de 0,15 en una LAN típicamente resulta en, como máximo, decenas de milisegundos de retardo; por lo tanto, se puede prescindir del retraso LAN. Sin embargo, como se discute en la Sección 1.4.2, ya que la intensidad de tráfico se aproxima a 1 (como es el caso del enlace de acceso en la Figura 2.12), el retardo en un enlace se hace muy grande y crece sin límite. Por lo tanto, el tiempo medio de respuesta para satisfacer las solicitudes va a ser del orden de minutos, si no más, que es inaceptable para los usuarios de la institución. Es evidente que hay que hacer algo.

Una posible solución es aumentar la tasa de acceso de 15 Mbps a, por ejemplo, 100 Mbps. Esto reducirá la intensidad de tráfico en el enlace de acceso a 0,15, lo que se traduce en retrasos insignificantes entre los dos routers. En este caso, el tiempo de respuesta total será de aproximadamente dos segundos, es decir, el retardo de Internet. Sin embargo, esta solución también significa que la institución debe actualizar su enlace de acceso de 15 Mbps a 100 Mbps, una propuesta costosa.

Consideremos ahora la solución alternativa de no actualizar el enlace de acceso pero en su lugar, instalar un caché web en la red institucional. Esta solución se ilustra en la Figura 2.13. -las tasas de aciertos -la fracción de solicitudes que se satisfacen con una memoria caché- típicamente están entre 0,2 y 0,7 en la práctica. Con fines ilustrativos, supongamos que la memoria caché proporciona una tasa de éxito de 0,4 para esta institución. Debido a que los clientes y la memoria caché están conectados a la misma LAN de alta velocidad, el 40 por ciento de las solicitudes será satisfecha casi de inmediato, por ejemplo, dentro de los 10 milisegundos, por la memoria caché. Sin embargo, todavía tienen que ser satisfechos por los servidores de origen el 60 por ciento restante de las solicitudes. Sin embargo, con sólo el 60 por ciento de los objetos solicitados que pasan a través del enlace de acceso, la intensidad de tráfico en el enlace de acceso se reduce desde 1,0 hasta 0,6. Típicamente, una intensidad de tráfico de menos

de 0,8 corresponde a un pequeño retraso, por ejemplo, decenas de milisegundos, en un enlace de 15 Mbps. Este retraso es insignificante en comparación con el retardo de dos segundos de Internet. Dadas estas consideraciones, el retraso medio es por lo tanto

$$0,4 * (0.01 \text{ segundos}) + 0,6 * (2.01 \text{ segundos})$$

que es sólo ligeramente mayor que 1,2 segundos. Por lo tanto, esta segunda solución proporciona un tiempo de respuesta aún más bajo que la primera solución, y que no requiere que la institución mejore su enlace a Internet. La institución, por supuesto, tiene que comprar e instalar un caché web. Sin embargo, este costo es bajo -muchas caches utilizan el software de dominio público que se ejecuta en los computadoras personales de bajo costo.

A través del uso de las **Redes de Distribución de Contenidos (CDN)**, las cachés web están jugando cada vez más un papel importante en Internet. Una empresa CDN instala muchos cachés distribuidos geográficamente a través de Internet, localizando así gran parte del tráfico. Hay CDN compartidos (tales como Akamai y Limelight) y CDN dedicados (como Google y Microsoft). Vamos a discutir CDN con más detalle en el capítulo 7.

2.2.6 El GET condicional

Aunque el almacenamiento en caché puede reducir los tiempos de respuesta percibida por el usuario, introduce un nuevo problema -la copia de un objeto que reside en la memoria caché puede ser obsoleta. En otras palabras, el objeto alojado en el servidor Web puede haber sido modificado desde que la copia se almacenó en la caché del cliente. Afortunadamente, HTTP tiene un mecanismo que permite a una caché verificar que sus objetos estén al día. Este mecanismo se denomina **GET condicional**. Un mensaje de petición HTTP es llamado un mensaje GET condicional si (1) el mensaje de petición utiliza el método GET y (2) el mensaje de solicitud incluye una cabecera de línea `If-Modified-Since:`.

Para ilustrar cómo funciona el GET condicional, vamos a caminar a través de un ejemplo. En primer lugar, en el nombre de un navegador de internet, una memoria caché de proxy envía un mensaje de solicitud a un servidor Web:

```
GET /fruta/kiwi.gif HTTP / 1.1
Anfitrión: www.exotiquecuisine.com
```

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

```
HTTP / 1.1 200 OK
Fecha: 8 Oct 2011 15:39:29
Servidor: Apache / 1.3.0 (Unix)
Última modificación: Miér 7 Sep 2011 09:23:24
Content-Type: image / gif
```

(Datos de datos de datos de datos ...)

El caché envía el objeto al navegador solicitante, sino también almacena en caché el objeto localmente. Es importante destacar que la caché también almacena la fecha de última modificación junto con el objeto. En tercer lugar, una semana más tarde, otro navegador solicita el mismo objeto a través de la memoria caché, y el objeto se encuentra todavía en la memoria caché. Puesto que este objeto puede haber sido modificado en el servidor Web en la última semana, la caché realiza un chequeo hasta la fecha mediante la emisión de un GET condicional. En concreto, la caché envía:

```
GET /fruit/kiwi.gif HTTP / 1.1
Anfitrión: www.exotiquecuisine.com
Si-modified-since: Miér 7 Sep 2011 09:23:24
```

Tenga en cuenta 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 enviado por el servidor hace una semana. Este GET condicional está diciendo al servidor que envíe el objeto sólo si el objeto se ha modificado desde la fecha especificada. Supongamos que el objeto no se ha modificado desde el 7 Sep 2011 09:23:24. A continuación, en cuarto lugar, el servidor Web envía un mensaje de respuesta a la caché:

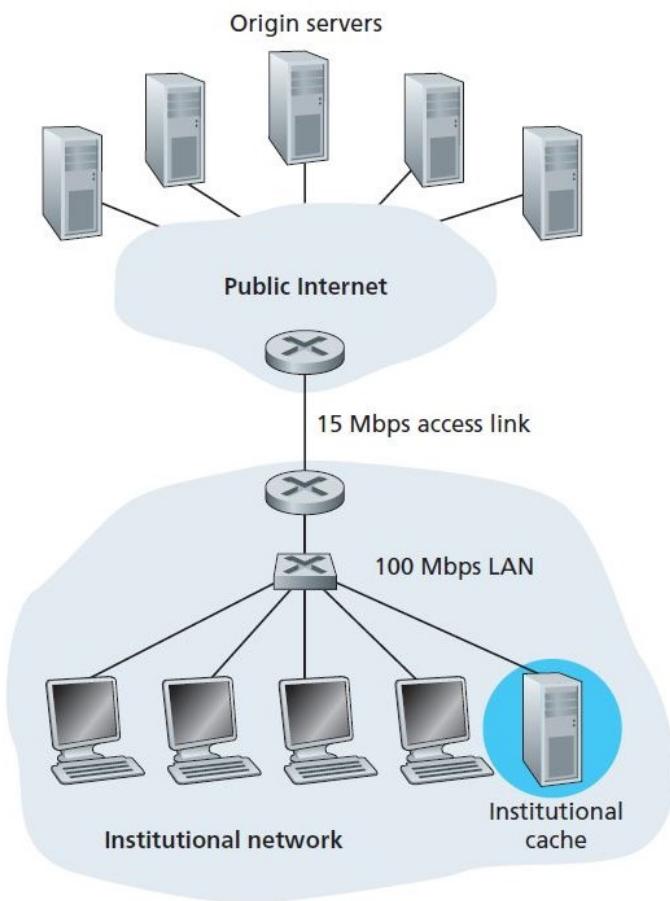


Figura 2.13

```

HTTP / 1.1 304 Not Modified
Fecha: 15 Oct 2011 15:39:29
Servidor: Apache / 1.3.0 (Unix)
(empty entity body)

```

Vemos que, en respuesta al GET condicional, el servidor Web envía un mensaje de respuesta, pero no incluye el objeto solicitado en el mensaje de respuesta. Incluir el objeto solicitado sólo desperdiciaría ancho de banda y aumentaría el tiempo de respuesta percibida por el usuario, particularmente si el objeto es grande. Tenga en cuenta que este mensaje de respuesta última tiene 304 Not modified en la línea de estado, que indica al caché que puede seguir adelante y remitir su copia en caché (de la caché de proxy) del objeto al navegador solicitante.

Esto pone fin a nuestra discusión de HTTP, el primer protocolo de Internet (un protocolo de capa de aplicación) que hemos estudiado en detalle. Hemos visto el formato de los mensajes HTTP y las acciones tomadas por el cliente y el servidor Web mientras que estos mensajes son enviados y recibidos. También hemos estudiado un poco de la infraestructura de las aplicaciones de la Web, incluyendo los cachés, cookies, y back-end de bases de datos, todos los cuales están vinculados de alguna manera con el protocolo HTTP.

2.3 Transferencia de archivos: FTP

En una típica sesión FTP, el usuario está sentado frente a un host (el host local) y quiere transferir archivos hacia o desde un host remoto. Para que el usuario acceda a la cuenta remota, el usuario debe proporcionar una identificación de usuario y una contraseña. Después de proporcionar esta información de autorización, el usuario puede transferir archivos desde el sistema de archivos local al sistema de archivos remoto y viceversa. Como se muestra en la figura 2.14, el usuario interactúa con FTP a través de un agente de usuario FTP. El usuario primero propor-

ciona el nombre de host del host remoto, haciendo que el proceso de cliente FTP en el ordenador local establezca una conexión TCP con el proceso del servidor FTP en el host remoto. Entonces, el usuario proporciona la identificación de usuario y una contraseña, que se envía a través de la conexión TCP como parte de los comandos FTP. Una vez que el servidor ha autorizado el usuario, el usuario copia uno o más archivos almacenados en el sistema de archivos local en el sistema de archivos remoto (o viceversa).

HTTP y FTP son dos protocolos de transferencia de archivos y tienen muchas características en común; por ejemplo, ambos se ejecutan en la parte superior de TCP. Sin embargo, los dos protocolos de capa de aplicación tienen algunas diferencias importantes. La diferencia más notable es que FTP utiliza dos conexiones TCP paralelas para transferir un archivo, una **conexión de control** y una **conexión de datos**. La conexión de control se utiliza para enviar información de

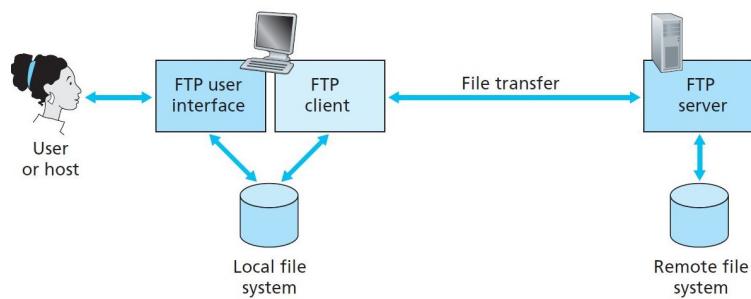


Figura 2.14

control entre los dos hosts -información tal como la identificación de usuario, contraseña, los comandos para cambiar directorio remoto y los comandos de "poner" y "obtener" los archivos. La conexión de datos se utiliza para enviar realmente un archivo. Debido a que FTP utiliza una conexión de control separada, se dice FTP envía su información de control **fuerza de banda**. HTTP, como se recordará, envía líneas de cabecera de solicitud y respuesta en la misma conexión TCP que lleva el archivo transferido en sí. Por esta razón, se dice HTTP envía su información de control en banda. En la siguiente sección, veremos que SMTP, el principal protocolo para el correo electrónico, que también envía información de control en banda. Las conexiones de control y de datos FTP se ilustran en la Figura 2.15.

Cuando un usuario inicia una sesión FTP con un host remoto, el lado cliente de FTP (usuario) inicia por primera vez una conexión TCP de control con del lado servidor (host remoto) en el puerto de servidor número 21. El lado cliente de FTP envía la identificación del usuario y contraseña a través de esta conexión de control. El lado del cliente de FTP también envía, a través de la conexión de control, los comandos para cambiar el directorio remoto. Cuando el servidor recibe una orden para una transferencia de archivos a través de la conexión de control (ya sea a, o desde el host remoto), el servidor inicia una conexión de datos TCP en el lado cliente. FTP envía exactamente un archivo a través de la conexión de datos y luego se cierra la conexión de datos. Si, durante la misma sesión, el usuario desea transferir otro archivo, FTP abre otra conexión de datos. De este modo, a través de FTP, la conexión de control permanece abierta durante toda la duración de la sesión del usuario, pero se crea una nueva conexión de datos para cada archivo transferido dentro de una sesión (es decir, las conexiones de datos son no persistentes).



Figura 2.15

A lo largo de una sesión, el servidor FTP debe mantener el estado sobre el usuario. En particular, el servidor debe asociar la conexión de control con una cuenta de usuario específica, y el servidor debe llevar un registro de directorio actual del usuario mientras el usuario deambula por el árbol de directorios remoto. Hacer un seguimiento de esta información de estado para cada sesión de usuario en curso restringe significativamente el número total de sesiones que FTP puede mantener de forma simultánea. Recordemos que HTTP, por el contrario, es sin estado, que no tiene que hacer un seguimiento de cualquier estado del usuario.

2.3.1 Comandos y Respuestas FTP

Terminamos esta sección con una breve discusión de algunos de los comandos FTP más comunes y respuestas. Los comandos, desde el cliente al servidor, y las respuestas, desde el servidor al cliente, se envían a través de la conexión de control en formato ASCII de 7 bits. Por lo tanto, al igual que los comandos HTTP, los comandos FTP pueden ser leídos por personas. Para delimitar los comandos sucesivos, un retorno de carro y la línea final de cada comando. Cada comando se compone de cuatro caracteres ASCII en mayúsculas, algunas de ellas con argumentos opcionales. Algunos de los comandos más comunes son los siguientes:

- **USER username**: Se utiliza para enviar la identificación del usuario al servidor.
- **PASS password**: Se utiliza para enviar la contraseña de usuario al servidor.
- **LIST**: Se utiliza para pedir al servidor que envie de vuelta una lista de todos los archivos en el directorio remoto actual. La lista de archivos se envía a través de una conexión de datos (nueva y no persistente) en lugar de la conexión de control TCP.
- **RETR filename**: Se utiliza para recuperar (es decir, obtener) un archivo desde el directorio actual de la máquina remota. Este comando hace que el host remoto inicie una conexión de datos y enviar el archivo solicitado por la conexión de datos.
- **STOR filename**: Se utiliza para memorizar (es decir, poner) un archivo en el directorio actual de la máquina remota.

Normalmente hay una correspondencia uno a uno entre el comando que el usuario maneja y el comando FTP enviado a través de la conexión de control. Cada comando es seguido por una respuesta, enviada desde el servidor al cliente. Las respuestas son números de tres dígitos, con un mensaje opcional después del número. Esto es similar en estructura al código de estado y la frase en la línea de estado del mensaje de respuesta HTTP. Algunas respuestas típicas, junto con sus posibles mensajes, son los siguientes:

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

Los lectores que estén interesados en aprender acerca de los otros comandos FTP y las respuestas se les anima a leer el RFC 959.

2.4 Correo Electrónico en Internet

El correo electrónico ha estado presente desde el comienzo de Internet. Era la aplicación más popular cuando la Internet estaba en su infancia [Segaller 1998], y se ha vuelto más y más elaborado y de gran alcance en los últimos años. Sigue siendo una de las más importantes y utilizadas de las aplicaciones de Internet.

Al igual que con el correo postal ordinario, el correo electrónico es un medio de comunicación asíncrona -las personas envían y leen mensajes cuando es conveniente para ellos, sin tener que coordinar con los horarios de otras personas. En contraste con el correo postal, el correo electrónico es rápido, fácil de distribuir y de bajo costo. Los correos electrónicos modernos tienen muchas características de gran alcance, incluyendo los mensajes con archivos

adjuntos, hipervínculos, texto con formato HTML, y las fotos incrustadas.

En esta sección, se examinan los protocolos de la capa de aplicaciones que se encuentran en el corazón del correo electrónico de Internet. Pero antes de saltar a una discusión más profunda sobre estos protocolos, vamos a dar una visión de alto nivel al sistema de correo de Internet y sus componentes clave.

La figura 2.16 presenta una visión de alto nivel del sistema de correo de Internet. Vemos en este diagrama que tiene tres componentes principales: los **agentes de usuario**, **servidores de correo**, y el **Protocolo Simple de Transferencia de Correo (SMTP)**. A continuación se describe cada uno de estos componentes en el contexto de un emisor, Alice, enviando un mensaje de correo electrónico a un destinatario, Bob. Los agentes de usuario permiten a los usuarios leer, responder, reenviar, guardar y redactar mensajes. Microsoft Outlook y Apple Mail son ejemplos de agentes de usuario de correo electrónico. Cuando Alice termina de elaborar su mensaje, su agente de usuario envía el mensaje a su servidor de correo, donde el mensaje se coloca en la cola de mensajes salientes del servidor de correo. Cuando Bob quiere leer un mensaje, su agente de usuario recupera el mensaje de su buzón de correo en su servidor de correo.

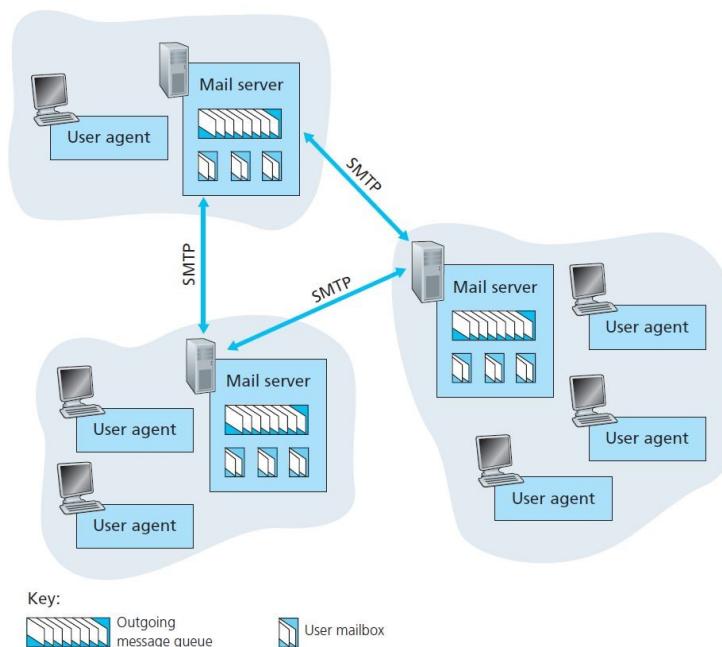


Figura 2.16

Los servidores de correo forman el núcleo de la infraestructura de correo electrónico. Cada receptor, como Bob, tiene un **buzón de correo** situado en uno de los servidores de correo. El buzón de correo de Bob gestiona y mantiene los mensajes que han sido enviados a él. Un mensaje típico comienza su travesía en el agente de usuario del remitente, viaja al servidor de correo del remitente, y viaja al servidor de correo del destinatario, donde se deposita en el buzón del destinatario. Cuando Bob desea acceder a los mensajes en su buzón, el servidor de correo que contiene el buzón autentica a Bob (con nombre de usuario y contraseña). El servidor de correo de Alice también tiene que hacer frente a los fallos en el servidor de correo de Bob. Si el servidor de Alice no puede entregar el correo al servidor de Bob, el servidor de Alice sostiene el mensaje en una cola de mensajes e intenta transferir el mensaje más tarde. Los reintentos a menudo se realizan cada 30 minutos más o menos; si no hay éxito después de varios días, el servidor elimina el mensaje y notifica al remitente (Alice) con un mensaje de correo electrónico.

SMTP es el principal protocolo principal de la capa de aplicación para el correo electrónico de Internet. Utiliza el servicio de transferencia de datos confiable de TCP para transferir el correo desde un servidor de correo del remitente al servidor de correo del destinatario. Al igual que con la mayoría de los protocolos de capa de aplicación, SMTP tiene dos partes: una del lado del cliente, que se ejecuta en el servidor de correo del remitente y un lado servidor, que ejecuta en el servidor de correo del destinatario. Tanto la parte cliente como servidora de SMTP se ejecutan en cada servidor de correo. Cuando un servidor de correo electrónico envía un mail a otros servidores de

correo, actúa como un cliente SMTP. Cuando un servidor de correo recibe un correo de otros servidores de correo, actúa como un servidor SMTP.

2.4.1 SMTP

SMTP, que se define en RFC 5321, se encuentra en el corazón de correo electrónico de Internet. Como se mencionó anteriormente, SMTP transfiere los mensajes de servidores de correos remitentes a los servidores de correo de los destinatarios. SMTP es mucho más antiguo que HTTP. (El RFC de SMTP original se remonta a 1982, y SMTP fue mucho antes de eso.) Aunque SMTP tiene numerosas cualidades maravillosas, como lo demuestra su ubicuidad en Internet, es una tecnología de legado que posee ciertas características arcaicas. Por ejemplo, restringe el cuerpo (no sólo los encabezados) de todos los mensajes de correo electrónico a simples ASCII de 7 bits. Esta restricción tenía sentido a principios de 1980, cuando la capacidad de transmisión era escasa y no había nadie enviaba por correo electrónico archivos adjuntos grandes o grandes archivos de imagen, audio o vídeo. Pero hoy en día, en la era multimedia, la restricción ASCII de 7 bits es un poco dolorosa -se requiere que los datos de multimedia binarios sean codificado en ASCII antes de ser enviados a través de SMTP; y requiere que el mensaje ASCII correspondiente sea decodificado de nuevo a binario después del transporte SMTP. Recuerde de la sección 2.2 que HTTP no requiere que los datos multimedia sean codificado a ASCII antes de la transferencia.

Para ilustrar el funcionamiento básico de SMTP, vamos a examinar un escenario común. Supongamos que Alice quiere enviar a Bob un mensaje ASCII simple.

1. Alice invoca su agente de usuario de correo electrónico, le provee el e-mail de Bob (por ejemplo, bob@someschool.edu), compone un mensaje, e instruye al agente de usuario para que envíe el mensaje.
2. El agente de usuario de Alice envía el mensaje a su servidor de correo, donde se coloca en una cola de mensajes.
3. El lado cliente de SMTP, que se ejecuta en el servidor de correo de Alice, ve el mensaje en la cola de mensajes. Se abre una conexión TCP a un servidor SMTP, que se ejecuta en el servidor de correo de Bob.
4. Despues de un apretón de manos inicial SMTP, el cliente SMTP envía el mensaje de Alice en la conexión TCP.
5. En el servidor de correo de Bob, el lado del servidor de SMTP recibe el mensaje. Entonces, el servidor de correo de Bob coloca el mensaje en el buzón de Bob.
6. Bob invoca su agente de usuario para leer el mensaje a su conveniencia.

El escenario se resume en la figura 2.17.

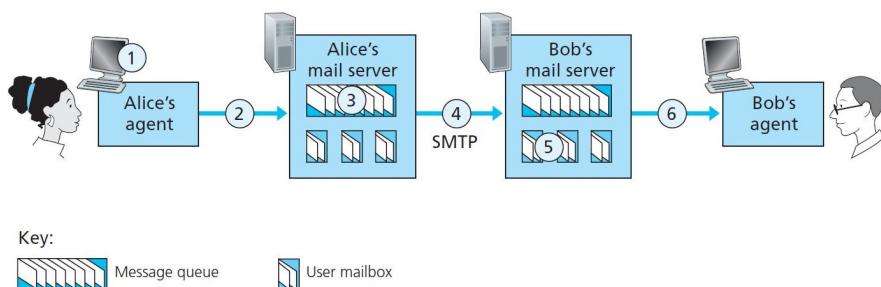


Figura 2.17

Es importante observar que SMTP normalmente no utiliza servidores de correo intermedios para el envío de correo, incluso cuando los dos servidores de correo se encuentran en extremos opuestos del mundo. Si el servidor de Alice se encuentra en Hong Kong y el servidor de Bob se encuentra en Córdoba, la conexión TCP es una conexión directa entre los servidores de Hong Kong y de Córdoba. En particular, si el servidor de correo de Bob se cae, el mensaje permanece en el servidor de correo de Alice y espera por un nuevo intento -el mensaje no es colocado en algún

servidor de correo intermedio.

Ahora vamos a echar un vistazo más de cerca sobre cómo SMTP transfiere un mensaje de un servidor de correo emisor a un servidor de correo receptor. Veremos que el protocolo SMTP tiene muchas similitudes con los protocolos que se utilizan para la interacción humana cara a cara. En primer lugar, el cliente SMTP (que se ejecuta en el host del servidor del correo remitente) tiene establecida una conexión TCP con el puerto 25 en el servidor SMTP (que se ejecuta en el servidor de correo del host receptor). Si el servidor no funciona, el cliente intenta de nuevo más tarde. Una vez establecida esta conexión, el servidor y el cliente realizan algunos acuerdos de capa de aplicación -al igual que los seres humanos a menudo se presentan antes de transferirse la información uno a otro, los clientes y los servidores SMTP se presentan antes de la transferencia de información. Durante esta fase de establecimiento de comunicación SMTP, el cliente SMTP indica la dirección de correo electrónico del remitente (la persona que ha generado el mensaje) y la dirección de correo electrónico del destinatario. Una vez que el cliente y el servidor SMTP se han introducido el uno al otro, el cliente envía el mensaje. SMTP puede confiar en el servicio de transferencia de datos confiable de TCP para hacer llegar el mensaje al servidor sin errores. Luego, el cliente repite este proceso sobre la misma conexión TCP si tiene otros mensajes para enviar al servidor; de lo contrario, indica a TCP que cierre la conexión.

Vamos echar un vistazo más de cerca a un ejemplo de transcripción 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 precedidos con C: son exactamente las líneas que el cliente envía en su socket TCP, y las líneas de texto ASCII precedidos por S: son exactamente 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 Hola crepes.fr, un gusto en conocerte
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Introduzca su correo, finalice con \." en una linea aparte
C: Le gusta la salsa de tomates?
C: Qué tal las salmueras?
C: .
S: 250 Mensaje aceptado para entrega
C: QUIT
S: 221 hamburger.edu cerrando conexión
```

En el ejemplo anterior, el cliente envía un mensaje ("¿Le gusta la salsa de tomate? ¿Qué hay de las salmueras?") desde el servidor de correo `crepes.fr` al servidor de correo `hamburger.edu`. Como parte del diálogo, el cliente emite cinco comandos: HELO (una abreviatura de HELLO), MAIL FROM, RCPT TO, DATA y QUIT. Estos comandos son fáciles de entender. El cliente también envía una línea que consta de un solo punto, que indica el final del mensaje en el servidor. (En la jerga ASCII, cada mensaje termina con CRLF. CRLF, donde CR y LF representan retorno de carro y salto de línea, respectivamente.) El servidor maneja respuesta a cada comando, con cada respuesta con un código de respuesta y algunos explicación (opcional) en inglés. Mencionamos aquí que SMTP utiliza conexiones persistentes: Si el servidor de correo remitente tiene varios mensajes para enviar al mismo servidor de correo entrante, puede enviar todos los mensajes a través de la misma conexión TCP. Para cada mensaje, el cliente comienza el proceso con una nueva MAIL FROM: `crepes.fr`, se designa el fin de mensaje con un punto aislado, y envía QUIT sólo después de que se han enviado todos los mensajes.

Es muy recomendable que utilices Telnet para llevar a cabo un diálogo directo con un servidor SMTP. Para ello, emita

```
serverName telnet 25
```

donde `ServerName` es el nombre de un servidor de correo local. Al hacer esto, simplemente estás estableciendo una conexión TCP entre el host local y el servidor de correo. Después de escribir esta línea, debes recibir de inmediato la respuesta 220 del servidor. A continuación, emita los comandos SMTP `HELO`, `MAIL FROM`, `RCPT TO`, `DATA`, `CRLF . CRLF`, y `QUIT` en el momento apropiado.

2.4.2 Comparación con HTTP

Ahora vamos a comparar brevemente SMTP con HTTP. Ambos protocolos se utilizan para transferir archivos desde un host a otro: HTTP transfiere archivos (también llamados objetos) desde un servidor Web a un cliente Web (típicamente un navegador); SMTP transfiere archivos (es decir, mensajes de correo electrónico) de un servidor de correo a otro servidor de correo. Al transferir los archivos, HTTP y SMTP utilizan conexiones persistentes. Por lo tanto, los dos protocolos tienen características comunes. Sin embargo, hay diferencias importantes. En primer lugar, HTTP es principalmente un **protocolo de extracción** -alguien carga información en un servidor web y los usuarios utilizan HTTP para extraer la información desde el servidor a su conveniencia. En particular, la conexión TCP es iniciada por la máquina que quiere recibir el archivo. Por otro lado, SMTP es principalmente un **protocolo de empuje** -el servidor de correo remitente empuja el archivo hacia el servidor de correo receptor. En particular, la conexión TCP se inicia por la máquina que quiere enviar el archivo.

Una segunda diferencia, que se ha citado anteriormente, es que SMTP requiere que cada mensaje, incluyendo el cuerpo de cada mensaje, esté en formato ASCII de 7 bits. Si el mensaje contiene caracteres que no son ASCII de 7 bits (por ejemplo, caracteres con acentos franceses) o contiene datos binarios (como 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 se refiere a cómo se maneja un documento que consta de texto e imágenes (posiblemente junto con otros tipos de medios). Como aprendimos en la sección 2.2, HTTP encapsula cada objeto en su propio mensaje de respuesta HTTP. El correo de Internet coloca todos los objetos del mensaje en un solo mensaje.

2.4.3 Formatos de los mensajes de correo

Cuando Alice escribe una correo a Bob, puede incluir todo tipo de información de cabecera periférica en la parte superior de la carta, como la dirección de Bob, su propia dirección de retorno, y la fecha. Del mismo modo, cuando un mensaje de correo electrónico se envía desde una persona a otra, una cabecera que contiene información periférica precede el cuerpo del mensaje en sí mismo. Esta información periférica está contenida en una serie de líneas de cabecera, que se definen en el RFC 5322. Las líneas de cabecera y el cuerpo del mensaje están separadas por una línea en blanco (es decir, por `CRLF`). RFC 5322 especifica el formato exacto para las líneas de encabezado de correo, así como sus interpretaciones semánticas. Al igual que con HTTP, cada línea de cabecera contiene texto legible, que consta de una palabra clave seguido de dos puntos, seguido de un valor. Algunas de las palabras clave son obligatorias y otras son opcionales. Cada cabecera debe tener una línea de cabecera `From:` y una línea de cabecera `To:`; una cabecera puede incluir una línea de cabecera `Subject:`, así como otras líneas de cabeceraopcionales. Es importante tener en cuenta que estas líneas de cabecera son *diferentes* de los comandos SMTP que estudiamos en la Sección 2.4.1 (a pesar de que contienen algunas palabras comunes como `"from"` y `"to"`). Los comandos de esa sección fueron parte del protocolo de establecimiento de conexión SMTP; las líneas de cabecera examinadas en esta sección son parte del mensaje de correo electrónico en sí.

Un encabezado de mensaje típico es el siguiente:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Buscando el objetivo de la vida.
```

Después de la cabecera del mensaje, sigue una línea en blanco; a continuación, el cuerpo del mensaje (en ASCII) sigue. Debe utilizar Telnet para enviar un mensaje a un servidor de correo que contiene un poco de líneas de cabecera, incluyendo el tema: línea de cabecera. Para ello, emita telnet `ServerName 25`, como se discute en la Sección 2.4.1.

2.4.4 Protocolos de Accesos a Correo

Una vez que SMTP entrega el mensaje del servidor de correo de Alice al servidor de correo de Bob, el mensaje se coloca en el buzón de Bob. A lo largo de esta discusión hemos asumido tácitamente que Bob lee su correo electrónico ingresando a la máquina del servidor y luego ejecutando un lector electrónico que se ejecuta en ese host. Hasta la década de 1990 esta era la forma habitual de hacer las cosas. Pero hoy en día, el acceso electrónico utiliza una arquitectura cliente-servidor -el típico usuario lee el correo electrónico con un cliente que se ejecuta en el sistema del usuario final, por ejemplo, en un PC de oficina, un ordenador portátil o un teléfono inteligente. Mediante la ejecución de un cliente de correo en un PC local, los usuarios disfrutan de un rico conjunto de características, incluyendo la posibilidad de ver los mensajes multimedia y archivos adjuntos.

Teniendo en cuenta que Bob (el receptor) ejecuta su agente de usuario en su PC local, es natural considerar colocar el servidor de correo en su PC local. Con este enfoque, el servidor de correo de Alice haría dialogar directamente con el PC de Bob. Sin embargo, hay un problema con este enfoque. Recordemos que un servidor de correo gestiona los buzones y se ejecuta las partes de cliente y servidor de SMTP. Si el servidor de correo de Bob fueran a residir en su PC local, a continuación, la PC de Bob tendría que permanecer siempre encendida, y conectada a Internet, con el fin de recibir correo nuevo, que puede llegar en cualquier momento. Esto es poco práctico 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 siempre activo compartido. Este servidor de correo se comparte con otros usuarios y por lo general es mantenido por el ISP del usuario (por ejemplo, una universidad o empresa).

Ahora vamos a considerar la trayectoria que un mensaje de correo electrónico toma cuando se envía desde Alice a Bob. Acabamos de enterarnos de que en algún momento a lo largo del camino que el mensaje de correo electrónico debe ser depositado en el servidor de correo de Bob. Esto podría hacerse simplemente teniendo al agente de usuario de Alice enviando el mensaje directamente al servidor de correo de Bob. Y esto se podría hacer con SMTP -de hecho, SMTP ha sido diseñado para empujar correo electrónico desde un host a otro. Sin embargo, normalmente el agente de usuario del remitente no dialoga directamente con el servidor de correo del destinatario. En su lugar, como se muestra en la Figura 2.18, el agente de usuario de Alice utiliza SMTP para empujar el mensaje de correo electrónico en su servidor de correo, a continuación, el servidor de correo de Alice utiliza SMTP (como cliente SMTP) para transmitir el mensaje de correo electrónico al servidor de correo de Bob. ¿Por qué el procedimiento de dos pasos? Principalmente porque sin retransmitir a través del servidor de correo de Alice, el agente de usuario de Alice no tiene ningún recurso a un servidor de correo de destino inalcanzable. Al tener Alice que depósito primero el correo electrónico en su propio servidor de correo, el servidor de correo de Alice puede intentar varias veces enviar el mensaje al servidor de correo de Bob, por ejemplo, cada 30 minutos, hasta que el servidor de correo de Bob esté en funcionamiento. (Y si el servidor de correo de Alice se cae, entonces ella tiene el recurso de quejarse con su administrador de sistema!) El RFC SMTP define cómo se pueden utilizar los comandos SMTP para transmitir un mensaje a través de múltiples servidores SMTP.

Pero todavía hay una pieza que falta en el rompecabezas! ¿Cómo hace un receptor como Bob, ejecutando un agente de usuario en su PC local, para obtener sus mensajes, que están ubicados en un servidor de correo dentro del ISP de Bob? Tenga en cuenta que el agente de usuario de Bob no puede usar SMTP para obtener los mensajes porque la obtención de los mensajes es una operación de extracción, mientras que SMTP es un protocolo de empuje. El rompecabezas se completa mediante la introducción de un protocolo de acceso electrónico especial que transfiere mensajes desde el servidor de correo de Bob a su PC local. En este momento hay una serie de protocolos de acceso electrónico populares, incluyendo **Post Office Protocol -Versión 3 (POP3)**, el **Protocolo de acceso al correo de Internet (IMAP)**, y HTTP.

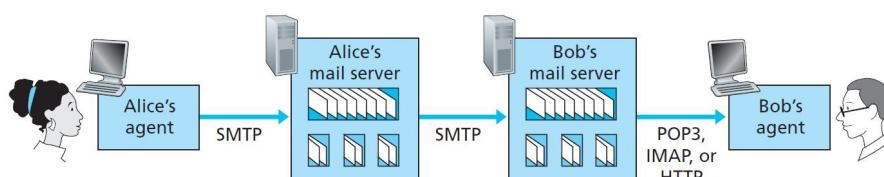


Figura 2.18

La figura 2.18 presenta un resumen de los protocolos que se utilizan para el correo de Internet: SMTP se utiliza para transferir mensajes del servidor de correo del remitente al servidor de correo del destinatario; SMTP también se utiliza para transferir el correo de agente de usuario del remitente al servidor de correo del remitente. Un protocolo de acceso al correo, como POP3, se utiliza para transferir mensajes del servidor de correo del destinatario al agente de usuario del destinatario.

POP3

POP3 es un protocolo de acceso electrónico extremadamente simple. Se define en [RFC 1939], que es corto y bastante legible. Debido a que el protocolo es tan simple, su funcionalidad es bastante limitada. POP3 comienza cuando el agente de usuario (cliente) abre una conexión TCP con el servidor de correo (el servidor) en el puerto 110. Con la conexión TCP establecida, POP3 progresará a través de tres fases: la autorización, la transacción y actualización. Durante la primera fase, la autorización, el agente de usuario envía un nombre de usuario y una contraseña para autenticar al usuario. Durante la segunda fase, la transacción, el agente de usuario recupera los mensajes; También durante esta fase, el agente de usuario puede marcar los mensajes para su eliminación, eliminar las marcas de borrado, y obtener estadísticas de correo. La tercera fase, la actualización, se produce después de que el cliente ha emitido el comando `quit`, terminando la sesión POP3; en este momento, el servidor de correo elimina los mensajes que se han marcado para su eliminación.

En una transacción POP3, el agente de usuario emite comandos, y el servidor responde a cada comando con una respuesta. Hay dos respuestas posibles: `+OK` (a veces seguido de datos de servidor a cliente), utilizado el servidor para indicar que el comando anterior estaba bien; y `-ERR`, utilizado el servidor para indicar que algo estaba mal con el comando anterior.

La fase de autorización tiene dos comandos principales: `user<username>` y `pass<password>`. Para ilustrar estos dos comandos, le sugerimos que telnetee directamente a un servidor POP3, utilizando el puerto 110, y emita estos comandos. Supongamos que el nombre del servidor de correo es `mailServer`. Verás algo como:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

Si se escribe incorrectamente un comando, el servidor POP3 responderá con un mensaje `-ERR`.

Ahora vamos a echar un vistazo a la fase de operación. Un agente de usuario, mediante POP3, a menudo se puede configurar (por el usuario) para "descargar y borrar" o "descargar y guardar". La secuencia de comandos emitidos por un agente de usuario POP3 depende de cuál de estos dos modos el agente de usuario esté funcionando. En el modo de descarga-y-borrado, el agente de usuario debe emitir los comandos `list`, `retr`, y `dele`. Como ejemplo, supongamos que el usuario tiene dos mensajes en su buzón de correo. En el siguiente diálogo, `C:` (representando al cliente) es el agente de usuario y `S:` (representando al servidor) es el servidor de correo. La transacción será algo como:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
```

```
S: .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

El agente de usuario primero solicita al servidor de correo que liste el tamaño de cada uno de los mensajes almacenados. El agente de usuario a continuación, recupera y elimina cada mensaje del servidor. Tenga en cuenta que después de la fase de autorización, el agente de usuario emplea sólo cuatro comandos: `list`, `retr`, `dele`, y `quit`. La sintaxis de estos comandos se define en el RFC 1939. Después de procesar el comando `quit`, el servidor POP3 entra en la fase de actualización y elimina los mensajes 1 y 2 del buzón de correo.

Un problema con este modo de descarga y eliminación es que el receptor, Bob, puede ser nómada y podría desear acceder a sus mensajes de correo desde varias máquinas, por ejemplo, su PC de la oficina, su PC casera, y su ordenador portátil. El modo de descarga-y-borrado partitiona los mensajes de correo de Bob entre estas tres máquinas; en particular, si Bob primero lee un mensaje en su PC de la oficina, no va a ser capaz de volver a leer el mensaje de su portátil en casa tarde en la noche. En el modo de descarga-y-almacenado, el agente de usuario deja los mensajes en el servidor de correo después de descargarlos. En este caso, Bob puede volver a leer los mensajes desde diferentes máquinas; se puede acceder a un mensaje de trabajo y acceder a él de nuevo más tarde en la semana desde casa.

Durante una sesión POP3 entre un agente de usuario y el servidor de correo, el servidor POP3 mantiene cierta información de estado; en particular, se realiza un seguimiento de cuales mensajes de usuario se han marcado como eliminados. Sin embargo, el servidor POP3 no lleva información de estado a través de sesiones POP3. Esta falta de información de estado a través de sesiones simplifica enormemente la implementación de un servidor POP3.

IMAP

Con acceso POP3, una vez que Bob ha descargado sus mensajes a la máquina local, puede crear carpetas de correo y mover los mensajes descargados en las carpetas. A continuación, Bob puede borrar mensajes, mover mensajes a través de carpetas, y buscar mensajes (por nombre de remitente o el asunto). Pero este paradigma -a saber, carpetas y mensajes en la máquina local- plantea un problema para el usuario nómada, quien preferiría mantener una jerarquía de carpetas en un servidor remoto que se puede acceder desde cualquier ordenador. Esto no es posible con POP3 -el protocolo POP3 no proporciona ningún medio a un usuario para crear carpetas remotas y asignar mensajes a carpetas.

Para resolver este y otros problemas, el protocolo IMAP, definido en [RFC 3501], fue inventado. Al igual que POP3, IMAP es un protocolo de acceso electrónico. Tiene muchas más funciones que POP3, pero también es significativamente más complejo. (Y por tanto las implementaciones laterales de cliente y servidor son significativamente más complejas.)

Un servidor IMAP asociará cada mensaje con una carpeta; cuando un mensaje llega por primera vez al servidor, es asociado a la carpeta *Bandeja de entrada* del destinatario. El destinatario puede mover el mensaje a una carpeta nueva, creada por el usuario, leer el mensaje, eliminar el mensaje, y así sucesivamente. 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 buscar en las carpetas remotas los mensajes que coinciden con criterios específicos. Tenga en cuenta que, a diferencia de POP3, un servidor IMAP mantiene información de estado del usuario a través de sesiones IMAP, por ejemplo, los nombres de las carpetas y los mensajes que están asociados con las carpetas.

Otra característica importante de IMAP es que tiene comandos que permiten que un agente de usuario obtenga componentes de mensajes. Por ejemplo, un agente de usuario puede obtener sólo el encabezado de un mensaje o sólo una parte de un mensaje MIME multiparte. Esta característica es útil cuando hay una conexión de bajo ancho de banda (por ejemplo, un enlace de módem de baja velocidad) entre el agente de usuario y su servidor de correo. Con una conexión de bajo ancho de banda, el usuario no puede querer descargar todos los mensajes en su buzón de correo, evitando en particular los mensajes largos que pueden contener, por ejemplo, un clip de audio o vídeo.

E-Mail basado en la Web

Cada vez son más los usuarios de hoy en día están enviando y accediendo a su correo electrónico a través de sus navegadores web. Hotmail introdujo el acceso basado en Web a mediados de la década de 1990. Ahora el correo electrónico basado en la Web también es proporcionado por Google, Yahoo!, así como casi todas las universidades y corporaciones. Con este servicio, el agente de usuario es un navegador Web convencional, y el usuario se comunica con su buzón de correo remoto a través de HTTP. Cuando un receptor, como Bob, quiere acceder a un mensaje en su buzón de correo, el mensaje de correo electrónico se envía desde el servidor de correo de Bob hacia el navegador de Bob utilizando el protocolo HTTP en lugar del protocolo POP3 o IMAP. Cuando un remitente, como Alice, quiere enviar un mensaje de correo electrónico, el mensaje de correo electrónico se envía de su navegador hacia su servidor de correo a través de HTTP en lugar de a través de SMTP. El servidor de correo de Alice, sin embargo, sigue enviando mensajes a, y recibe mensajes desde otros servidores de correo, utilizando SMTP.

2.5 DNS-El servicio de directorio de Internet

Los seres humanos podemos identificarnos de muchas maneras. Por ejemplo, podemos ser identificados por los nombres que aparecen en nuestros certificados de nacimiento. Podemos ser identificados a través de nuestros números de documento. Podemos ser identificados por los números de nuestra licencia de conducir. Aunque cada uno de estos identificadores se pueden utilizar para identificar a las personas, dentro de un contexto dado un identificador puede ser más apropiado que otro. Por ejemplo, las computadoras en el CIDI (la famosa agencia de Ciudadano Digital en Córdoba) prefieren utilizar los números de cuil longitud fija en lugar de nombres de certificado de nacimiento. Por otro lado, la gente común prefiere los nombres de los certificados de nacimiento más mnemotécnicos en lugar de números de documento. (De hecho, se puede imaginar diciendo: "Hola. Mi nombre es 36.604.892. Te presento a mi marido, 38.935.402").

Al igual que los seres humanos pueden identificarse de muchas maneras, también pueden hacerlo los hosts en Internet. Un identificador de un host es su **nombre de host**. Los nombres de host -tales como `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu`, y `cis.poly.edu`-son mnemotécnicos y por lo tanto, apreciados por los seres humanos. Sin embargo, los nombres de host proporcionan poca, o ninguna, información acerca de la ubicación dentro de Internet del host. (Un nombre de host como `www.eurecom.fr`, que termina con el código de país `.fr`, nos dice que el host está, probablemente, en Francia, pero no dice mucho más.) Por otra parte, debido a que los nombres de host pueden contener caracteres alfanuméricos de longitud variable, sería difícil de procesar por los routers. Por estas razones, los anfitriones también se identifican con las llamadas **direcciones IP**.

Se discuten las direcciones IP con cierto detalle en el capítulo 4, pero es útil decir unas breves palabras acerca de ellas ahora. Una dirección IP se compone de cuatro bytes y tiene una estructura jerárquica rígida. Una dirección IP es algo como 121.7.106.83, donde cada punto separa uno de los bytes expresados en notación decimal de 0 a 255. Una dirección IP es jerárquica porque a medida que se recorre la dirección de izquierda a derecha, obtenemos más y más información específica acerca donde se encuentra el host en Internet (es decir, dentro de qué red, en la red de redes). Del mismo modo, cuando escaneamos una dirección postal de abajo hacia arriba, obtenemos más y más información específica acerca de dónde se encuentra el destinatario.

2.5.1 Servicios proporcionados por el DNS

Acabamos de ver que hay dos maneras de identificar un host -por un nombre de host y por una dirección IP. La gente prefiere el identificador de nombre de host más mnemotécnica, mientras que los routers prefieren direcciones IP de longitud fija, estructuradas jerárquicamente. Con el fin de reconciliar estas preferencias, necesitamos un servicio de directorio que traduzca nombres de host a direcciones IP. Esta es la tarea principal del **sistema de nombres de dominio (DNS)** de Internet. El DNS es (1) una base de datos distribuida implementada en una jerarquía de servidores DNS, y (2) un protocolo de nivel de aplicación que permite a los computadoras consultar la base de datos distribuida. Los servidores DNS son a menudo máquinas UNIX que ejecuta el software Berkeley Internet Name Domain (BIND) [BIND 2012]. El protocolo DNS se ejecuta a través de UDP y utiliza el puerto 53.

DNS es empleado comúnmente por otros protocolos, incluyendo la capa de aplicaciones HTTP, SMTP y FTP- para traducir los nombres de host suministrados por el usuario a direcciones IP. A modo de ejemplo, consideremos lo

que sucede cuando un navegador (es decir, un cliente HTTP), que se ejecuta en el host de algún usuario, pide la URL `www.someschool.edu/index.html`. Con el fin de que el host del usuario pueda enviar un mensaje de petición HTTP al servidor Web `www.someschool.edu`, el host del usuario debe primero obtener la dirección IP de `www.someschool.edu`. Esto se realiza de la siguiente manera.

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

Vemos en este ejemplo que DNS agrega un retardo adicional -a veces sustancial- para las aplicaciones de Internet que lo utilizan. Afortunadamente, como veremos a continuación, la dirección IP deseada a menudo se almacena en un servidor DNS "cercano", lo que ayuda a reducir el tráfico de red DNS, así como el retraso promedio de DNS.

DNS proporciona algunos otros servicios importantes, además de la traducción de nombres de host a direcciones IP:

- **Host aliasing.** Un host con un nombre de host complicado puede tener uno o más alias. Por ejemplo, un nombre de host como `relay1.west-coast.enterprise.com` podría tener, por ejemplo, dos alias como `enterprise.com` y `www.enterprise.com`. En este caso, se dice que el nombre de host `relay1.west-coast.enterprise.com` es un **nombre de host canónico**. Los alias de nombres de host, cuando están presentes, son típicamente más mnemotécnica que los nombres de host canónicos. DNS puede ser invocado por una aplicación para obtener el nombre canónico de un nombre de host alias suministrado, así como la dirección IP de la máquina.
- **Aliasing para los servidores de correo.** Por razones obvias, es muy conveniente que las direcciones de correo electrónico sean mnemotécnico. Por ejemplo, si Bob tiene una cuenta en Hotmail, la dirección de correo electrónico de Bob podría ser tan simple como `bob@hotmail.com`. Sin embargo, el nombre de host del servidor de correo de Hotmail es más complicado y mucho menos mnemotécnico que simplemente `hotmail.com` (por ejemplo, el nombre canónico podría ser algo así como `relay1.west-coast.hotmail.com`). DNS puede ser invocado por una aplicación de correo para obtener el nombre canónico de un nombre de host alias suministrado, así como la dirección IP de la máquina. De hecho, el registro MX (véase más adelante) permite a al servidor de correo electrónico de una compañía y a su servidor web tener nombres de host idénticos (alias); por ejemplo, el servidor Web y el servidor de correo de una empresa puede llamarse `enterprise.com`.
- **Distribución de la carga.** DNS también se utiliza para realizar la distribución de carga entre los servidores replicados, tales como servidores web replicados. Sitios ocupados, tales como `cnn.com`, se replican a través de múltiples servidores, con cada servidor ejecutándose en un sistema final diferente y cada uno con una dirección IP diferente. Para los servidores Web replicados, un conjunto de direcciones IP se asocia con un solo nombre canónico. La base de datos DNS contiene este conjunto de direcciones IP. Cuando los clientes hacen una consulta DNS para un nombre asignado a un conjunto de direcciones, el servidor responde con todo el conjunto de direcciones IP, pero gira el orden de las direcciones dentro de cada respuesta. Debido a que un cliente normalmente envía su mensaje de solicitud HTTP a la dirección IP que se muestra por primera vez en el set, la rotación de DNS distribuye el tráfico entre los servidores replicados. La rotación de DNS también se utiliza para el correo electrónico de manera que múltiples servidores de correo pueden tener el mismo nombre de alias. Además, las empresas de distribución de contenidos como Akamai han utilizado DNS de manera más sofisticada [Dilley 2002] para proporcionar una distribución de contenidos web (véase el capítulo 7).

El DNS se especifica en el RFC 1034 y RFC 1035, y actualiza en varios RFC adicionales. Se trata de un sistema complejo, y sólo pretendemos aspectos clave de su funcionamiento aquí. El lector interesado puede consultar estos RFC y el libro de Albitz y Liu [Albitz 1993]; véase también el documento de retrospectiva [Mockapetris 1988], que proporciona una buena descripción del qué y por qué de DNS y [Mockapetris 2005].

2.5.2 Descripción general de cómo funciona DNS

Ahora presentamos una descripción de alto nivel de cómo funciona DNS. Nuestra discusión se centrará en el servicio de traducción de nombre de host a dirección IP.

Supongamos que alguna aplicación (como un navegador web o un lector electrónico) que se ejecuta en el host de un usuario tiene que traducir un nombre de host a una dirección IP. La aplicación invocará el lado del cliente de DNS, que especifica el nombre de host que necesita ser traducido. (En muchas máquinas basadas en UNIX, `gethostbyname()` es la llamada a la función que una aplicación llama a fin de realizar la traducción.) A continuación, el DNS en el host del usuario se hace cargo, enviando un mensaje de consulta en la red. Todos los mensajes consulta y respuesta DNS se envían dentro de datagramas UDP al puerto 53. Después de un retraso, que van desde milisegundos a segundos, el DNS de host del usuario recibe un mensaje de respuesta DNS que proporciona la asignación deseada. Esta asignación se pasa a continuación a la aplicación de invocación. Por lo tanto, desde la perspectiva de la aplicación de invocación en el host del usuario, DNS es una caja negra que proporciona un servicio de traducción simple y directa. Pero, de hecho, la caja negra que implementa el servicio es compleja, consistiendo en un gran número de servidores DNS distribuidos en todo el mundo, así como un protocolo de capa de aplicación que especifica cómo los servidores DNS y los hosts consultantes se comunican.

Un diseño simple para DNS tendría un servidor DNS conteniendo todas las asignaciones. En este diseño centralizado, los clientes simplemente dirigen todas las consultas al simple servidor DNS, y el servidor DNS responde directamente a los clientes que consultan. A pesar de que la simplicidad de este diseño es atractiva, no es apropiada para la Internet de hoy, con su gran (y creciente) número de hosts. Los problemas con un diseño centralizado incluyen:

- **Un punto único de fallo.** Si el servidor DNS se cae, también lo hace la totalidad de Internet!
- **Volumen de tráfico.** Un servidor DNS solo, tendría que manejar todas las consultas DNS (para todas las peticiones HTTP y mensajes de correo electrónico generados a partir de cientos de millones de hosts).
- **Base de datos centralizada distante.** Un único servidor DNS no puede estar "cerca" de todos los clientes que consultan. Si ponemos el único servidor DNS en la ciudad de Nueva York, a continuación, todas las consultas de Australia deben viajar al otro lado del globo, tal vez a través de vínculos lentos y congestionados. Esto puede dar lugar a retrasos significativos.
- **Mantenimiento.** El único servidor DNS tendría que llevar un registro de todos los hosts conectados a Internet. Esto no sólo haría que la base de datos centralizada sea enorme, sino que tendría que ser actualizada con frecuencia para llevar la cuenta de cada nuevo huésped.

En resumen, una base de datos centralizada en un único servidor DNS simplemente *no escala*. En consecuencia, el DNS está distribuido por diseño. De hecho, el DNS es un maravilloso ejemplo de cómo una base de datos distribuida puede ser implementada en Internet.

Una base de datos jerárquica distribuida

Con el fin de hacer frente a la cuestión de la escala, el DNS utiliza un gran número de servidores, organizados de forma jerárquica y distribuida en todo el mundo. No hay ningún servidor DNS que tenga todas las asignaciones para todos los hosts en Internet. En su lugar, las asignaciones se distribuyen a través de los servidores DNS. En una primera aproximación, hay tres clases de servidores DNS -los servidores DNS raíz, servidores DNS de dominio de nivel superior (TLD) y servidores DNS autorizados organizados en una jerarquía, como se muestra en la Figura 2.19. Para entender cómo funcionan estas tres clases de servidores, supongamos que un cliente DNS quiere determinar la dirección IP para el nombre de host `www.amazon.com`. En una primera aproximación, los siguientes eventos se llevarán a cabo. El cliente primero contacta uno de los servidores raíz, que devuelve las direcciones IP de los servidores de dominio de primer nivel para el dominio de nivel superior `.com`. El cliente entonces se contacta con uno de estos servidores TLD, que le devuelve la dirección IP de un servidor autorizado para `amazon.com`. Por último, el cliente contacta uno de los servidores autorizados para `amazon.com`, que devuelve la dirección IP para el nombre de host `www.amazon.com`. Pronto vamos a examinar este proceso de búsqueda de DNS en más detalle. Pero primero vamos a echar un vistazo más de cerca a estas tres clases de servidores DNS:

- **Servidores DNS raíz.** En Internet hay 13 servidores DNS raíz (etiquetados desde A a H), la mayoría de los cuales se encuentran en América del Norte. Un mapa de octubre de 2006 sobre los servidores DNS raíz se

muestra en la Figura 2.20; una lista de los servidores DNS raíz actuales está disponible a través de [Root-servers 2012]. Aunque nos hemos referido a cada uno de los 13 servidores DNS raíz como si fueran un solo servidor, cada uno de estos "servidores" es en realidad una red de servidores replicados, tanto para fines de seguridad como para fiabilidad. En total, hay 247 servidores raíz a partir del otoño de 2011.

- **Servidores de dominio superior (TLD).** Estos servidores son responsables de dominios de nivel superior, como com, org, net, edu, y gov, y todos los dominios de primer nivel nacional, tales como uk, fr, ca, y jp. La compañía Verisign Global Registry Services mantiene los servidores TLD para el dominio de nivel superior com, y la compañía Educause, mantiene los servidores TLD para el dominio de nivel superior edu. Ver [IANA TLD 2012] para obtener una lista de todos los dominios de nivel superior.
- **servidores DNS autorizados.** Cada organización con hosts de acceso público (tales como servidores web y servidores de correo) en Internet debe proporcionar registros DNS de acceso público que se asignan los nombres de los hosts a direcciones IP. Los servidores DNS autorizados de una organización albergan estos registros DNS. Una organización puede optar por aplicar su propio servidor DNS autorizado para mantener estos registros; Como alternativa, la organización puede pagar para que estos registros sean almacenados en un servidor DNS autorizado de algún proveedor de servicios. La mayoría de las universidades y grandes empresas implementan y mantienen su propio servidor DNS autorizado primario y secundario (copia de seguridad).

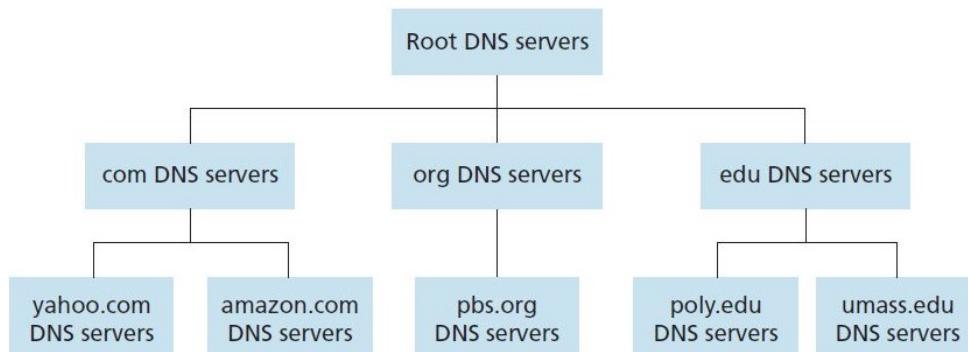


Figura 2.19

Todos los servidores DNS raíz, TLD, y autorizados pertenecen a la jerarquía de los servidores DNS, como se muestra en la Figura 2.19. Hay otro tipo importante de servidor DNS llamado **servidor DNS local**. Un servidor DNS local no pertenece estrictamente a la jerarquía de los servidores, pero es sin embargo el centro de la arquitectura DNS. Cada ISP -tales como una universidad, un departamento académico, la compañía de un empleado, o un ISP residencial- tiene un servidor DNS local (también llamado un servidor de nombres por defecto). Cuando un host se conecta a un ISP, el ISP proporciona al host con las direcciones IP de uno o más de sus servidores DNS locales (normalmente a través de DHCP, que se discute en el capítulo 4). Se puede determinar fácilmente la dirección IP de su servidor DNS local mediante el acceso a la ventana de estado de red en Windows o UNIX. Un servidor DNS local de un host está típicamente "cerca" del host. Para un ISP institucional, el servidor DNS local puede estar en la misma LAN que el host; para un ISP residencial, típicamente se separa del host por no más de unos pocos routers. Cuando un host hace una consulta DNS, la consulta se envía al servidor DNS local, que actúa como un proxy, enviando la consulta a la jerarquía del servidor DNS, como veremos en detalle más adelante.

Vamos a echar un vistazo a un ejemplo sencillo. Supongamos que el host `cis.poly.edu` desea la dirección IP de `gaia.cs.umass.edu`. También suponga que el servidor DNS local de Politécnico se llama `dns.poly.edu` y que un servidor DNS autorizado para `gaia.cs.umass.edu` se llama `dns.umass.edu`. Como se muestra en la Figura 2.21, el host `cis.poly.edu` primero envía un mensaje de consulta DNS a su servidor DNS local, `dns.poly.edu`. El mensaje de petición contiene el nombre de host a ser traducido, a saber, `gaia.cs.umass.edu`. El servidor DNS local envía el mensaje de consulta a un servidor DNS raíz. El servidor DNS raíz toma nota del sufijo `edu` y regresa al servidor DNS local una lista de direcciones IP para los servidores TLD responsables de `edu`. El servidor DNS local, entonces vuelve a enviar 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 autorizado para la Universidad de Massachusetts, a saber, `dns.umass.edu`. Por último, el servidor DNS local vuelve a enviar el mensaje

de consulta directamente a `dns.umass.edu`, que responde con la dirección IP de `gaia.cs.umass.edu`. Tenga en cuenta que en este ejemplo, con el fin de obtener la asignación para un nombre de host, se enviaron ocho mensajes DNS: cuatro mensajes de consulta y cuatro mensajes de respuesta! Pronto veremos cómo el almacenamiento en caché de DNS reduce este tráfico de consultas.

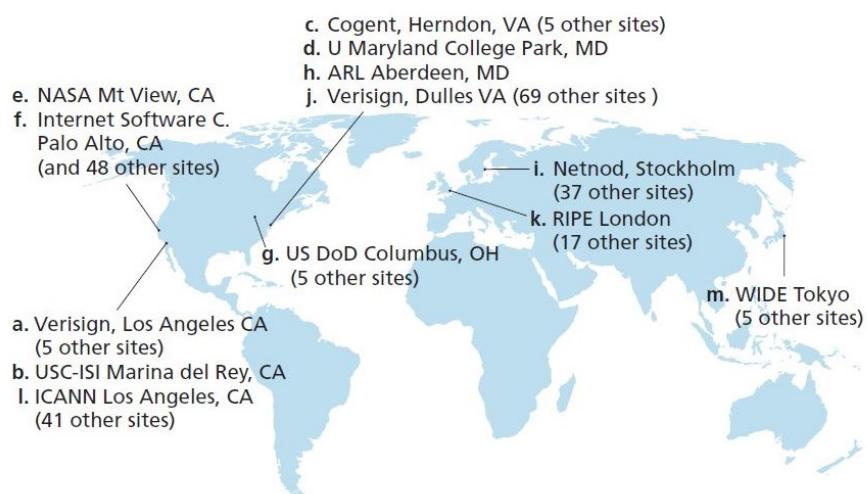


Figura 2.20

Nuestro ejemplo anterior supone que el servidor TLD conoce el servidor DNS autorizado para el nombre de host. En general, esto no siempre es cierto. En su lugar, el servidor TLD puede conocer solamente un servidor DNS intermedio, que a su vez conoce el servidor DNS autorizado para el nombre de host. Por ejemplo, supongamos que una vez más que la Universidad de Massachusetts

tiene un servidor DNS para la universidad, llamado `dns.umass.edu`. Supongamos también que cada uno de los departamentos de la Universidad de Massachusetts tiene su propio servidor DNS, y que cada servidor DNS departamental es autorizado para todos los hosts en el departamento. En este caso, cuando el servidor DNS intermedio, `dns.umass.edu`, recibe una consulta para un host con un nombre de host que termina con `cs.umass.edu`, devuelve a `dns.poly.edu` la dirección IP de `dns.cs.umass.edu`, que es la autoridad de todos los nombres que acaben con `cs.umass.edu`. El servidor DNS local `dns.poly.edu` envía la consulta al servidor DNS autorizado, que devuelve el mapeo deseado para el servidor DNS local, que a su vez devuelve el mapeo a la máquina solicitante. En este caso, un total de 10 mensajes DNS se envían!

El ejemplo que se muestra en la Figura 2.21 hace uso de **consultas recursivas** y de **consultas iterativas**. La consulta enviada desde `cis.poly.edu` a `dns.poly.edu` es una consulta recursiva, ya que la consulta pide a `dns.poly.edu` que obtenga el mapeo en su nombre. Pero los posteriores tres consultas son iterativos, ya que todas las respuestas son devueltas directamente a `dns.poly.edu`. En teoría, cualquier consulta DNS puede ser iterativa o recursiva. Por ejemplo, la Figura 2.22 muestra una cadena de consulta DNS para el que todas las consultas son recursivas. En la práctica, las consultas suelen seguir el patrón de la Figura 2.21: La consulta de la máquina solicitante al servidor DNS local es recursiva, y las consultas restantes son iterativas.

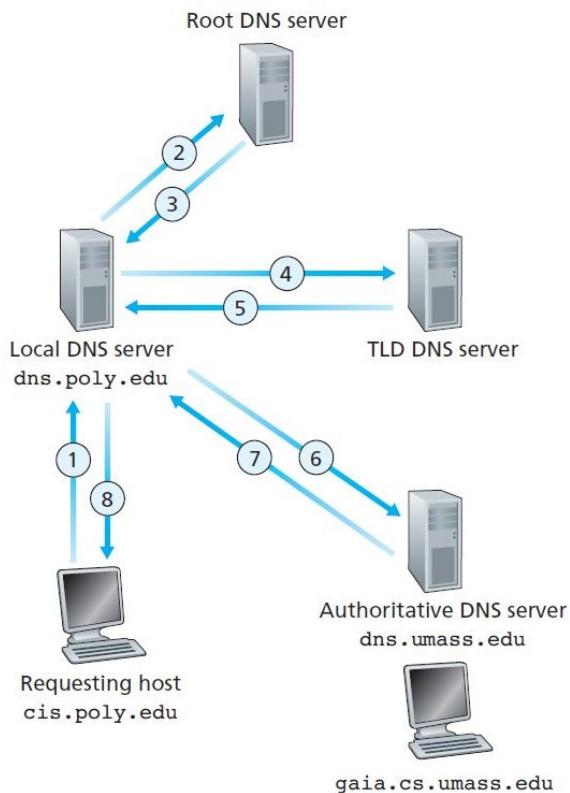


Figura 2.21

El almacenamiento en caché de DNS

Nuestra discusión hasta ahora ha ignorado el **almacenamiento en caché de DNS**, una característica muy importante del sistema DNS. En verdad, DNS explota extensivamente el almacenamiento en caché de DNS con el fin de mejorar el rendimiento de retardo y para reducir el número de mensajes DNS que rebotan en torno a Internet. La idea detrás del almacenamiento en caché de DNS es muy simple. En una cadena de consulta, cuando un servidor DNS recibe una respuesta DNS (que contiene, por ejemplo, una asignación de un nombre de host a una dirección IP), puede almacenar el mapeo en su memoria local. Por ejemplo, en la figura 2.21, cada vez que el servidor DNS local dns.poly.edu recibe una respuesta de algún servidor DNS, se puede almacenar en caché la información contenida en la respuesta. Si un par nombre de host/IP se almacena en un servidor DNS y otra consulta llega al servidor DNS para el mismo nombre de host, el servidor DNS puede proporcionar la dirección IP deseada, incluso si no está autorizado para el nombre de host. Debido a que los hosts y las asignaciones entre los nombres de host y direcciones IP no son permanentes, los servidores DNS descartan la información almacenada en caché después de un período de tiempo (a menudo se establece en dos días).

A modo de ejemplo, supongamos que un host apricot.poly.edu consulta a dns.poly.edu por la dirección IP para el nombre de host cnn.com. Más aún, supongamos que un par de horas más tarde, otro host de la Universidad Politécnica, por ejemplo, kiwi.poly.fr, también consulta a dns.poly.edu con el mismo nombre de host. Debido a la memoria caché, el servidor DNS local será capaz de devolver inmediatamente la dirección IP de cnn.com a este segundo host solicitante sin tener que consultar cualquier otro servidor DNS. Un servidor DNS local también puede almacenar en caché las direcciones IP de los servidores de dominio de primer nivel, lo que permite al servidor DNS local omitir los servidores DNS raíz en una cadena de consulta (esto ocurre a menudo).

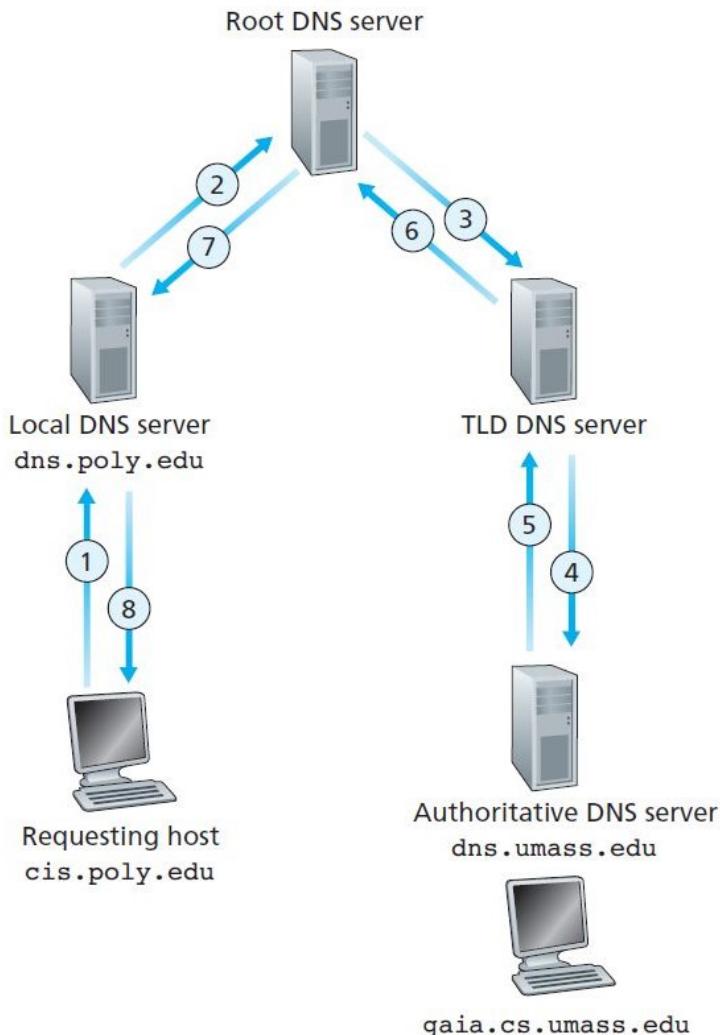


Figura 2.22

2.5.3 Los Registros DNS y los Mensajes

Los servidores DNS que juntos implementan la base de datos DNS distribuida almacenan **registros de recursos (RR)**, incluidos los RR que proporcionan las asignaciones nombre de host a direcciones IP. Cada mensaje de respuesta de DNS lleva uno o más registros de recursos. En esta y en la siguiente subsección, proporcionamos una breve descripción de los registros DNS y los mensajes de recursos; más detalles se pueden encontrar en [Abitz 1993] o en los RFC de DNS [RFC 1034; RFC 1035].

Un registro de recursos es una cuatro tupla que contiene los siguientes campos:

(Nombre, Valor, Tipo, TTL)

TTL es el tiempo de vida del registro de recursos; determina cuándo un recurso se debe quitar de la memoria caché. En el ejemplo de los registros que figuran a continuación, ignoramos el campo TTL. El significado del Name y Value dependen de Type:

- Si Type=A, entonces Name es un nombre de host y Value es la dirección IP para el nombre de host. Por lo tanto, un registro de tipo A proporciona la asignación estándar nombre de host a dirección IP. A modo de ejemplo, (`relay1.bar.foo.com`, `145.37.93.126`, A) es un registro Type A.
- Si Type=NS, entonces Name es un dominio (tal como `foo.com`) y Value es el nombre de host de un servidor DNS autorizado que sabe cómo obtener las direcciones IP de los hosts en el dominio. Este registro se

utiliza para enrutar las consultas DNS a lo largo de la cadena de consulta. A modo de ejemplo, (`foo.com`, `dns.foo.com`, `NS`) es un registro de tipo NS.

- Si `Type=CNAME`, entonces `Value` es un nombre canónico para el alias del nombre de host `Name`. Este registro puede proporcionar a hosts la consulta por el nombre canónico de un nombre de host. A modo de ejemplo, (`foo.com`, `relay1.bar.foo.com`, `CNAME`) es un registro CNAME.
- Si `Type=MX`, entonces `Value` es el nombre canónico de un servidor de correo que tiene por alias al nombre de host `Name`. A modo de ejemplo, (`foo.com`, `mail.bar.foo.com`, `MX`) es un registro MX. Los registros MX permiten que los nombres de host de los servidores de correo tengan alias simples. Tenga en cuenta que al utilizar el registro MX, una empresa puede tener el mismo nombre de alias para su servidor de correo y para uno de sus otros servidores (como su servidor Web). Para obtener el nombre canónico del servidor de correo, un cliente DNS debería pedir el registro MX; para obtener el nombre canónico para el otro servidor, el cliente DNS podría preguntar para el registro CNAME.

Si un servidor DNS está autorizado para un nombre de host en particular, entonces el servidor DNS contiene un tipo de registro A para el nombre de host. (Incluso si el servidor DNS no está autorizado, puede contener un registro de tipo A en su caché.) Si un servidor no está autorizado para un nombre de host, el servidor va a contener un tipo de registro NS para el dominio que incluye el nombre de host; También contendrá un tipo de un registro A que proporciona la dirección IP del servidor DNS en el campo `Value` del registro NS. A modo de ejemplo, supongamos que un servidor TLD `edu` que no está autorizado para el host `gaia.cs.umass.edu`. Entonces, este servidor contendrá un registro del dominio que incluye al host `gaia.cs.umass.edu`, por ejemplo, (`umass.edu`, `dns.umass.edu`, `NS`). El servidor TLD `edu` también contendrá un tipo de registro A, que mapea el servidor DNS `dns.umass.edu` a una dirección IP, por ejemplo, (`dns.umass.edu`, `128.119.40.111`, `A`).

Los mensajes DNS

Anteriormente en esta sección, nos referimos a los mensajes DNS de consulta y de respuesta. Estos son los únicos dos tipos de mensajes DNS. Además, ambos mensajes de consulta y de respuesta tienen el mismo formato, como se muestra en la Figura 2.23. La semántica de los diversos campos de un mensaje de DNS son los siguientes:

- Los primeros 12 bytes son la *sección de cabecera*, que tiene una serie de campos. El primer campo es un número de 16 bits que identifica la consulta. Este identificador se copia en el mensaje de respuesta a una consulta, permitiendo al cliente coincidir las respuestas recibidas con las consultas enviadas. Hay una serie de flags en el campo `flga`. Un flag de 1 bit consulta/respuesta indica si el mensaje es una consulta (0) o una respuesta (1). Una bandera autorizada de 1 bit se setea en un mensaje de respuesta cuando un servidor DNS es un servidor autorizado para un nombre consultado. Una flag de recursión deseada de 1 bit se establece cuando un cliente (host o servidor DNS) desea que el servidor DNS realice recursividad cuando no posea el registro. Un campo de recursividad disponible de 1 bit se setea en la respuesta si el servidor DNS es compatible con la recursividad. En la cabecera, hay también campos de cuatro números. Estos campos indican el número de ocurrencias de los cuatro tipos de secciones de datos que siguen a la cabecera.
- La *sección de preguntas* contiene información acerca de la consulta que se está realizando. En esta sección se incluye (1) un campo de nombre que contiene el nombre que se está consultando, y (2) un campo de tipo que indica el tipo de pregunta que se hace sobre el nombre -por ejemplo, una dirección de host asociado a un nombre (tipo A) o el servidor de correo para un nombre (tipo MX).
- En una respuesta de un servidor DNS, la *sección de respuesta* contiene los registros de recursos para el nombre que se le preguntó en un principio. Recordemos que en cada registro de recursos está el `Type` (por ejemplo, A, NS, CNAME, y MX), el `Value` y el `TTL`. Una respuesta puede devolver varios RR en la respuesta, ya que un nombre de host puede tener varias direcciones IP (por ejemplo, para los servidores web replicados, como se ha discutido anteriormente en esta sección).
- La *sección de autoridad* contiene los registros de otros servidores autorizados.
- La *sección adicional* contiene otros registros útiles. Por ejemplo, el campo de respuesta en una respuesta a una consulta MX contiene un registro de recursos que proporciona el nombre canónico de un servidor de correo. La sección adicional contiene un registro de tipo A que proporciona la dirección IP para el nombre canónico del servidor de correo.

Identification	Flags	
Number of questions	Number of answer RRs	12 bytes
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used

Figura 2.23

Inserción de registros en la base de datos DNS

La discusión anterior se centró en cómo los registros se recuperan de la base de datos de DNS. Tal vez te preguntes cómo se meten en los registros de la base de datos en el primer lugar. Veamos cómo se hace esto en el contexto de un ejemplo específico. Supongamos que acabas de crear una excitante nueva empresa startup llamada Red Utopía. La primera cosa que seguramente querrás hacer es registrar el nombre de dominio `redutopia.com` en un registrador. Un **registrador** es una entidad comercial que verifica la exclusividad del nombre de dominio, ingresa en el nombre de dominio en la base de datos DNS (como veremos más adelante), y recoge una pequeña tasa para usted por sus servicios. Antes de 1999, un único registro, Network Solutions, tenía el monopolio de registro de nombres de dominio para `.com`, `.net` y `.org`. Pero ahora hay muchas agencias de registro que compiten por los clientes, y la Corporación de Internet para Nombres y Números Asignados (ICANN) acredita los diversos registradores. Una lista completa de los registradores acreditados está disponible en <http://www.internic.net>.

Al registrar el nombre de dominio `redutopia.com` con cierto registro, también es necesario proporcionar el registro con los nombres y direcciones IP de los servidores DNS autorizados primario y secundario. Supongamos que los nombres y las direcciones IP son `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1` y `212.212.212.2`. Para cada uno de estos dos servidores DNS autorizados, el registrador entonces se asegura de que un registro tipo NS y un registro de tipo A se introduzcan en los servidores de dominio de nivel superior `.com`. En concreto, para el servidor autorizado principal para `redutopia.com`, el registrador insertaría 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 tenes que asegurarte de que el tipo A de registro de recursos para el servidor Web `www.networkutopia.com` y el de registro de recursos Tipo MX de su servidor de correo `mail.redutopia.com` sean introducen en sus servidores DNS autorizados. (Hasta hace poco, los contenidos de cada servidor de DNS se configuraban de forma estática, por ejemplo, de un archivo de configuración creado por un administrador de sistema. Más recientemente, una opción actualización se ha añadido al protocolo DNS para permitir que los datos que se añadan dinámicamente o eliminan de la base de datos a través de mensajes de DNS. [RFC 2136] y [RFC 3007] especifican las actualizaciones dinámicas.)

Una vez que todos estos pasos se han completado, la gente será capaz de visitar tu sitio web y enviar correo electrónico a los empleados en su empresa. Vamos a concluir nuestra discusión de DNS mediante la verificación de que esta afirmación es cierta. Esta verificación también ayuda a solidificar lo que hemos aprendido acerca de DNS. Supongamos que Alicia en Australia quiere ver la página Web `www.redutopia.com`. Como se señaló anteriormente, su host primero enviará una consulta DNS a su servidor DNS local. El servidor DNS local se

pondrá en contacto con un servidor de dominio de nivel superior com. (El servidor DNS local también tendrá que ponerse en contacto con un servidor DNS raíz si la dirección de un servidor de dominio de nivel superior com no se almacena en caché.) Este servidor de dominio de nivel superior contiene los registros de recursos de tipo A y tipo NS enumerados anteriormente, debido a que el registrador había insertado estos registros de recursos en todos los servidores de dominio de nivel superior com. El servidor TLD com envía una respuesta al servidor DNS local de Alice, con la respuesta que contiene los dos registros de recursos. El servidor DNS local envía una consulta DNS para 212.212.212.1, pidiendo el registro tipo A correspondiente a www.redutopia.com. Este registro proporciona la dirección IP del servidor Web deseado, por ejemplo, 212.212.71.4, que el servidor DNS local pasa de nuevo al host de Alice. El navegador de Alice ahora puede iniciar una conexión TCP con el host 212.212.71.4 y enviar una solicitud HTTP sobre la conexión. ¡Uf! Hay mucho más en juego de lo que parece a simple vista cuando se navega por la web!

2.7 Programación Socket: Creación de aplicaciones de red

Ahora que hemos analizado varias aplicaciones de red importantes, exploremos cómo se crean realmente los programas de aplicaciones de red. Recuerde en la Sección 2.1 que una aplicación de red típica consiste en un par de programas, un programa cliente y un programa servidor, que residen en dos sistemas finales diferentes. Cuando se ejecutan estos dos programas, se crean un proceso de cliente y un proceso de servidor, y estos procesos se comunican entre sí leyendo y escribiendo en sockets. Al crear una aplicación de red, la tarea principal del desarrollador es escribir el código para los programas del cliente y del servidor.

Hay dos tipos de aplicaciones de red. Un tipo es una implementación cuyo funcionamiento se especifica en un protocolo estándar, como un RFC o algún otro documento de estándares; Esta aplicación a veces se denomina "abierta", ya que todos conocen las reglas que especifican su funcionamiento. Para tal implementación, los programas cliente y servidor deben cumplir con las reglas dictadas por el RFC. Por ejemplo, el programa cliente podría ser una implementación del lado del cliente del protocolo FTP, que se describe en la Sección 2.3 y se define explícitamente en RFC 959; de manera similar, el programa del servidor podría ser una implementación del protocolo del servidor FTP, también explícitamente definido en RFC 959. Si un desarrollador escribe código para el programa cliente y otro desarrollador escribe código para el programa del servidor, y ambos desarrolladores siguen cuidadosamente las reglas de RFC, entonces los dos programas podrán interoperar. De hecho, muchas de las aplicaciones de red de hoy en día involucran la comunicación entre el cliente y los programas de servidor que han sido creados por desarrolladores independientes, por ejemplo, un navegador Firefox que se comunica con un servidor web Apache o un cliente BitTorrent que se comunica con el rastreador BitTorrent.

El otro tipo de aplicación de red es una aplicación de red propietaria. En este caso, los programas cliente y servidor emplean un protocolo de capa de aplicación que no se ha publicado abiertamente en un RFC o en otro lugar. Un solo desarrollador (o equipo de desarrollo) crea los programas del cliente y del servidor, y el desarrollador tiene control completo sobre lo que se incluye en el código. Pero como el código no implementa un protocolo abierto, otros desarrolladores independientes no podrán desarrollar un código que interactúe con la aplicación.

En esta sección, examinaremos los problemas clave en el desarrollo de una aplicación cliente-servidor y "nos ensuciaremos las manos" al observar el código que implementa una aplicación cliente-servidor muy simple. Durante la fase de desarrollo, una de las primeras decisiones que debe tomar el desarrollador es si la aplicación se ejecutará a través de TCP o UDP. Recuerde que TCP está orientado a la conexión y proporciona un canal de flujo de datos confiable a través del cual los datos fluyen entre dos sistemas finales. UDP no tiene conexión y envía paquetes de datos independientes de un sistema de extremo a otro, sin ninguna garantía de entrega. Recuerde también que cuando un programa cliente o servidor implementa un protocolo definido por un RFC, debe usar el número de puerto conocido asociado con el protocolo; a la inversa, al desarrollar una aplicación propietaria, el desarrollador debe tener cuidado de evitar el uso de números de puerto tan conocidos. (Los números de puerto se discutieron brevemente en la Sección 2.1. Se tratan con más detalle en el Capítulo 3).

Introducimos la programación de sockets UDP y TCP mediante una aplicación UDP simple y una aplicación TCP simple. Presentamos las aplicaciones UDP y TCP simples en Python. Podríamos haber escrito el código en Java, C o C++, pero elegimos Python principalmente porque Python expone claramente los conceptos clave de socket. Con Python hay menos líneas de código, y cada línea puede explicarse al programador novato sin dificultad. Pero no hay por qué asustarse si no está familiarizado con Python. Debería poder 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 recomendamos que visite el sitio web complementario de este libro de texto; de hecho, puede encontrar todos los ejemplos en esta sección (y laboratorios asociados) en Java. Para los lectores que estén interesados en la programación cliente-servidor en C, hay varias buenas referencias disponibles [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996]; nuestros ejemplos de Python a continuación tienen una apariencia similar a C.

2.7.1 Programación de socket con UDP

En esta subsección, escribiremos programas cliente-servidor simples que utilizan UDP; En la siguiente sección, escribiremos programas similares que usan TCP.

Recuerde de la Sección 2.1 que los procesos que se ejecutan en diferentes máquinas se comunican entre sí mediante el envío de mensajes a sockets. Dijimos que cada proceso es análogo a una casa y que el socket del proceso es análogo a una puerta. La aplicación reside en un lado de la puerta de la casa; El protocolo de la capa de transporte reside en el otro lado de la puerta en el mundo exterior. El desarrollador de la aplicación tiene el control de todo en el lado de la capa de la aplicación del socket; sin embargo, tiene poco control del lado de la capa de transporte.

Ahora echemos un vistazo más de cerca a la interacción entre dos procesos de comunicación que usan sockets UDP. Antes de que el proceso de envío pueda empujar un paquete de datos por la puerta del socket, cuando se usa UDP, primero debe adjuntar una dirección de destino al paquete. Después de que el paquete pase a través del socket del remitente, Internet utilizará esta dirección de destino para enrutar el paquete a través del Internet en el proceso de recepción. Cuando el paquete llega al socket receptor, el proceso de recepción lo recuperará a través del socket y luego inspeccionará el contenido del paquete y tomará las medidas apropiadas.

Así que ahora se estará preguntando, ¿qué pasa en la dirección de destino que se adjunta al paquete? Como es de esperar, la dirección IP del host de destino es parte de la dirección de destino. Al incluir la dirección IP de destino en el paquete, los enruteadores en Internet podrán enrutar el paquete a través de Internet al host de destino. Pero como un host puede ejecutar muchos procesos de aplicaciones de red, cada uno con uno o más sockets, también es necesario identificar el socket en particular en el host de destino. Cuando se crea un socket, se le asigna un identificador, denominado número de puerto. Por lo tanto, como es de esperar, la dirección de destino del paquete también incluye el número de puerto del socket. En resumen, el proceso de envío adjunta al paquete una dirección de destino que consiste en la dirección IP del host de destino y el número de puerto del conector de destino. Además, como veremos pronto, la dirección de origen del remitente, que consta de la dirección IP del host de origen y el número de puerto del socket de origen, también se adjunta al paquete. Sin embargo, la conexión de la dirección de origen al paquete generalmente no se realiza mediante el código de la aplicación UDP; en su lugar, se realiza automáticamente por el sistema operativo subyacente.

Usaremos la siguiente aplicación simple cliente-servidor para demostrar la programación de socket para UDP y 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 en 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 2.28 destaca la actividad principal relacionada con el socket del cliente y el servidor que se comunican a través del servicio de transporte UDP.

Ahora ensuciémonos y echemos un vistazo al par de programas cliente-servidor para una implementación UDP de esta sencilla aplicación. También proporcionamos un análisis detallado línea por línea después de cada programa. Comenzaremos con el cliente UDP, que enviará un simple mensaje de nivel de aplicación al servidor. Para que el servidor pueda recibir y responder al mensaje del cliente, debe estar listo y en ejecución; es decir, debe ejecutarse como un proceso antes de que el cliente envíe su mensaje.

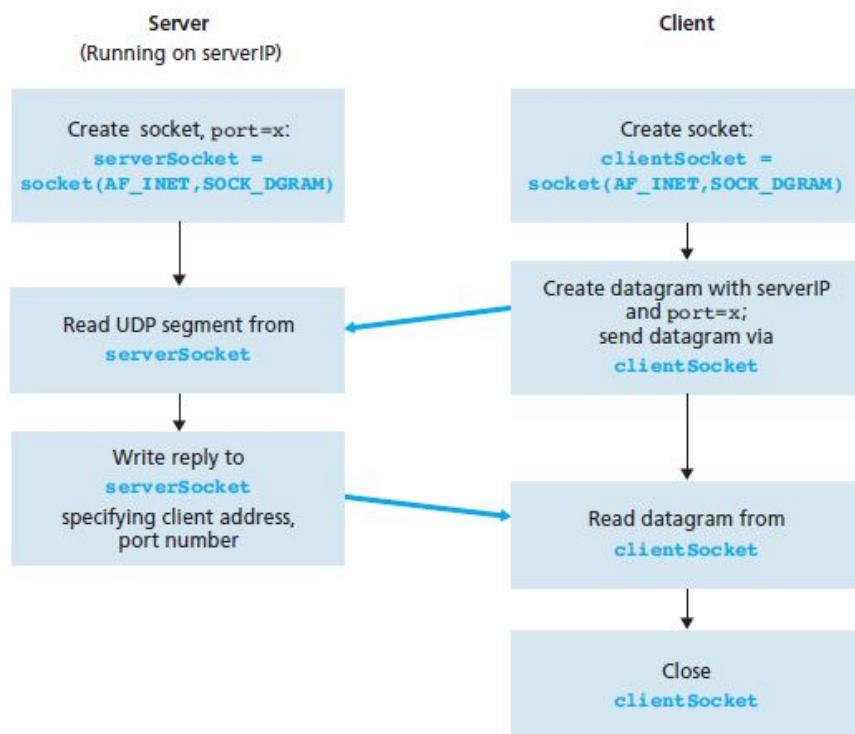


Figura 2.28

El programa cliente se llama `UDPClient.py`, y el programa servidor se llama `UDPServer.py`. Para enfatizar los problemas clave, intencionalmente proporcionamos un código que es mínimo. El "buen código" sin duda tendría algunas líneas auxiliares más, en particular para manejar los casos de error. Para esta aplicación, hemos elegido arbitrariamente 12000 para el número de puerto del servidor.

UDPClient.py

Aquí está el código para el lado del cliente de la aplicación:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message, (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Ahora echemos un vistazo a las diversas líneas de código en `UDPClient.py`.

```
from socket import *
```

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

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

La primera línea establece la cadena `serverName` a `hostname`. Aquí, proporcionamos una cadena que contiene la dirección IP del servidor (por ejemplo, "128.138.32.126") o el nombre de host del servidor (por ejemplo,

"cis.poly.edu"). Si usamos el nombre de host, entonces se realizará una búsqueda de DNS para obtener la dirección IP.) La segunda línea establece la variable entera serverPort en 12000.

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

Esta línea crea el socket del cliente, llamado clientSocket. El primer parámetro indica la familia de direcciones; en particular, AF_INET indica que la red subyacente está utilizando IPv4. (No se preocupe por esto ahora; analizaremos IPv4 en el Capítulo 4). El segundo parámetro indica que el socket es del tipo SOCK_DGRAM, lo que significa que es un socket UDP (en lugar de un socket TCP). Tenga en cuenta que no estamos especificando el número de puerto del socket del cliente cuando lo creamos; En cambio, estamos dejando que el sistema operativo haga esto por nosotros. Ahora que se ha creado la puerta del proceso del cliente, queremos crear un mensaje para enviar a través de la puerta.

```
message = raw_input ('oración en minúscula de entrada:')
```

raw_input () es una función incorporada en Python. Cuando se ejecuta este comando, se le pregunta al usuario en el cliente con las palabras "Datos de entrada:" El usuario luego usa su teclado para ingresar una línea, que se coloca en el mensaje variable. Ahora que tenemos un socket y un mensaje, desearemos enviar el mensaje a través del socket al host de destino.

```
clientSocket.sendto (mensaje, (serverName, serverPort))
```

En la línea anterior, el método sendto () adjunta la dirección de destino (serverName, serverPort) al mensaje y envía el paquete resultante al socket del proceso, clientSocket. (Como se mencionó anteriormente, la dirección de origen también se adjunta al paquete, aunque esto se hace de forma automática en lugar de explícitamente mediante el código). ¡Enviar un mensaje de cliente a servidor a través de un socket UDP es así de simple! Después de enviar el paquete, el cliente espera recibir datos del servidor.

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

Con la línea anterior, cuando un paquete llega desde Internet al socket del cliente, los datos del paquete se colocan en la variable modifiedMessage y la dirección de origen del paquete se coloca en la variable serverAddress. La variable serverAddress contiene la dirección IP del servidor y el número de puerto del servidor. El programa UDPClient no necesita realmente esta información de dirección del servidor, ya que ya conoce la dirección del servidor desde el principio; pero esta línea de Python proporciona la dirección del servidor sin embargo. El método recvfrom también toma el tamaño de búfer 2048 como entrada. (Este tamaño de búfer funciona para la mayoría de los propósitos.)

```
print modifiedMessage
```

Esta línea imprime el mensaje modificado en la pantalla del usuario. Debe ser la línea original que el usuario escribió, pero ahora en mayúscula.

```
clientSocket.close()
```

Esta línea cierra el socket. El proceso luego termina.

UDPServer.py

Veamos ahora el lado del servidor de la aplicación:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
```

```
modifiedMessage = message.upper()
serverSocket.sendto(modifiedMessage, clientAddress)
```

Tenga en cuenta que el principio de UDPServer es similar a UDPClient. También importa el módulo de socket, también establece la variable entera serverPort en 12000 y también crea un socket de tipo SOCK_DGRAM (un socket UDP). La primera línea de código que es significativamente diferente de UDPClient es:

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

La línea anterior vincula (es decir, asigna) el número de puerto 12000 al socket del servidor. Así, en UDPServer, el código (escrito por el desarrollador de la aplicación) está asignando explícitamente un número de puerto al socket. De esta manera, cuando alguien envíe un paquete al puerto 12000 en la dirección IP del servidor, ese paquete se dirigirá a este socket. UDPServer luego entra en un bucle while; el bucle while permitirá a UDPServer recibir y procesar paquetes de clientes de forma indefinida. En el bucle while, UDPServer espera a que llegue un paquete.

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

Esta línea de código es similar a lo que vimos en UDPClient. Cuando un paquete llega al socket del servidor, los datos del paquete se colocan en el mensaje variable y la dirección de origen del paquete se coloca en la dirección de cliente variable. La variable clientAddress contiene tanto la dirección IP del cliente como el número de puerto del cliente. Aquí, UDPServer hará uso de esta información de dirección, ya que proporciona una dirección de retorno, similar a la dirección de retorno con correo postal ordinario. Con esta información de dirección de origen, el servidor ahora sabe hacia dónde debe dirigir su respuesta.

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

Esta línea es el corazón de nuestra sencilla aplicación. Toma la línea enviada por el cliente y usa el método upper () para capitalizarla.

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

Esta última línea adjunta la dirección del cliente (dirección IP y número de puerto) al mensaje en mayúscula, y envía el paquete resultante al socket del servidor. (Como se mencionó anteriormente, la dirección del servidor también se adjunta al paquete, aunque esto se hace de forma automática en lugar de hacerlo explícitamente mediante el código). Internet luego entregará el paquete a esta dirección del cliente. Después de que el servidor envíe el paquete, permanece en el bucle while, esperando que llegue otro paquete UDP (de cualquier cliente que se ejecute en cualquier host).

Para probar el par de programas, instale y compile UDPClient.py en un host y UDPServer.py en otro host. Asegúrese de incluir el nombre de host o la dirección IP adecuados del servidor en UDPClient.py. A continuación, ejecuta UDPServer.py, el programa de servidor compilado, en el servidor host. Esto crea un proceso en el servidor que permanece inactivo hasta que es contactado por algún cliente. A continuación, ejecuta UDPClient.py, el programa cliente compilado, en el cliente. Esto crea un proceso en el cliente. Finalmente, para usar la aplicación en el cliente, escribe una oración seguida de un retorno de carro.

Para desarrollar su propia aplicación UDP cliente-servidor, puede comenzar modificando ligeramente los programas cliente o servidor. Por ejemplo, en lugar de convertir todas las letras a mayúsculas, el servidor podría contar el número de veces que aparece la letra s y devolver este número. O puede modificar el cliente para que después de recibir una oración en mayúscula, el usuario pueda continuar enviando más oraciones al servidor.

2.7.2 Programación de socket 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 comenzar a enviar datos entre ellos, primero necesitan un protocolo de enlace y establecer una conexión TCP. Un extremo de la conexión TCP está conectado al socket del cliente y el otro extremo está conectado a un socket del servidor. Al crear la conexión TCP, asociamos con ella la dirección de socket del cliente

(dirección IP y número de puerto) y la dirección de socket del servidor (dirección IP y número de puerto).

Con la conexión TCP establecida, cuando un lado desea enviar datos al otro lado, simplemente coloca los datos en la conexión TCP a través de su socket. Esto es diferente de UDP, para el cual el servidor debe adjuntar una dirección de destino al paquete antes de colocarlo en el socket.

Ahora echemos un vistazo más de cerca a la interacción de los programas cliente y servidor en TCP. El cliente tiene el trabajo de iniciar contacto con el servidor. Para que el servidor pueda reaccionar ante el contacto inicial del cliente, el servidor debe estar listo. Esto implica dos cosas. Primero, como en el caso de UDP, **el servidor TCP debe ejecutarse como un proceso antes de que el cliente intente iniciar el contacto**. En segundo lugar, el programa del servidor debe tener una puerta especial, más precisamente, un zócalo especial, **que da la bienvenida a algún contacto inicial de un proceso cliente que se ejecuta en un host arbitrario**. Usando la analogía de nuestra casa/puerta para un proceso/socket, a veces nos referiremos al contacto inicial del cliente como "llamar a la puerta de bienvenida".

Con el proceso del servidor en ejecución, el proceso del 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 bienvenida en el servidor, es decir, la dirección IP del servidor y el número de puerto del socket. Después de crear su socket, el cliente inicia un protocolo de enlace de tres vías y establece una conexión TCP con el servidor. El protocolo de enlace de tres vías, que tiene lugar dentro de la capa de transporte, es completamente invisible para los programas del cliente y del servidor.

Durante el protocolo de enlace de tres vías, el proceso del cliente llama a la puerta de bienvenida del proceso del servidor. Cuando el servidor "escucha" las detonaciones, crea una nueva puerta; más precisamente, un nuevo conector dedicado a ese cliente en particular. En nuestro ejemplo a continuación, la puerta de bienvenida es un objeto de socket TCP al que llamamos serverSocket; El socket recién creado dedicado al cliente que realiza la conexión se llama connectionSocket. Los estudiantes que se encuentran con sockets TCP por primera vez a veces confunden el socket de bienvenida (que es el punto de contacto inicial para todos los clientes que desean comunicarse con el servidor), y cada socket de conexión del lado del servidor que se crea posteriormente para crear comunicación con cada cliente.

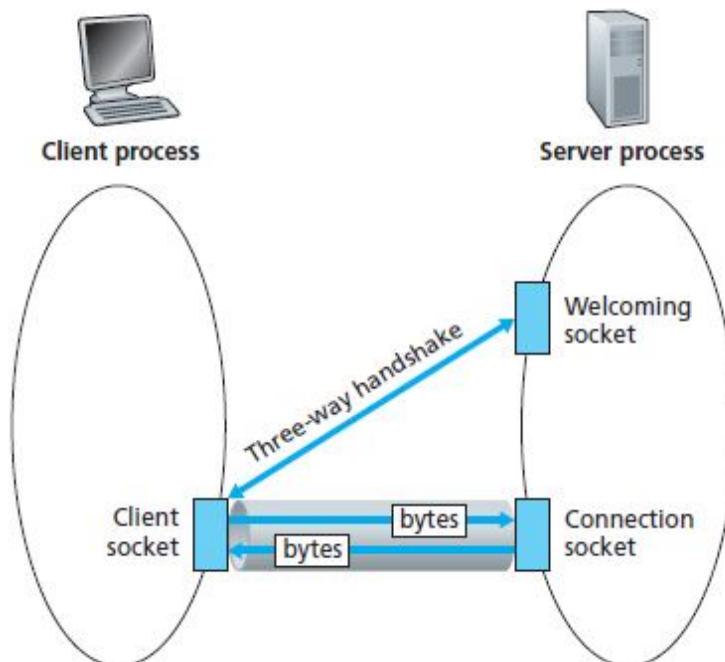


Figura 2.29

Desde la perspectiva de la aplicación, el socket del cliente y el socket de conexión del servidor están conectados directamente por una tubería. Como se muestra en la Figura 2.29, el proceso del cliente puede enviar bytes arbitrarios a su socket, y TCP garantiza que el proceso del servidor recibirá (a través del socket de conexión) cada byte en el orden enviado. TCP proporciona un servicio confiable entre el cliente y los procesos del servidor. Además, al igual que las personas pueden entrar y salir por la misma puerta, el proceso del cliente no solo envía bytes sino que también recibe bytes de su socket; De manera similar, el proceso del servidor no solo recibe bytes sino que también envía bytes a su socket de conexión.

Usamos la misma aplicación simple cliente-servidor para demostrar la programación de socket con TCP: el cliente envía una línea de datos al servidor, el servidor capitaliza la línea y la envía de vuelta al cliente. La Figura 2.30 destaca la actividad principal relacionada con el socket del cliente y el servidor que se comunican a través del servicio de transporte TCP.

TCPClient.py

Aquí está 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))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Veamos ahora las diversas líneas en el código que difieren significativamente de la implementación UDP. La primera de esas líneas es la creación del socket del cliente.

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

Esta línea crea el socket del cliente, llamado clientSocket. El primer parámetro nuevamente indica que la red subyacente está utilizando IPv4. El segundo parámetro indica que el socket es de tipo SOCK_STREAM, lo que significa que es un socket TCP (en lugar de un socket UDP). Tenga en cuenta que, una vez más, no estamos especificando el número de puerto del socket del cliente cuando lo creamos; En cambio, estamos dejando que el sistema operativo haga esto por nosotros. Ahora la siguiente línea de código es muy diferente de lo que vimos en UDPClient:

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

Recuerde que antes de que el cliente pueda enviar datos al servidor (o viceversa) utilizando un socket TCP, primero debe establecerse 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 del servidor de la conexión. Una vez que se ejecuta esta línea de código, se realiza el protocolo de enlace de tres vías y se establece una conexión TCP entre el cliente y el servidor.

```
frase = raw_input('frase en minúscula de entrada:')
```

Al igual que con UDPClient, lo anterior obtiene una oración del usuario. La oración de cadena continúa reuniendo caracteres hasta que el usuario termina la línea escribiendo un retorno de carro. La siguiente línea de código también es muy diferente de UDPClient:

```
clientSocket.send(oración)
```

La línea anterior envía la oración de cadena a través del socket del cliente y en la conexión TCP. Tenga en cuenta que el programa no crea explícitamente un paquete y adjunta la dirección de destino al paquete, como fue el caso con sockets UDP. En su lugar, el programa cliente simplemente coloca los bytes en la oración de cadena en la

conexión TCP. El cliente espera recibir bytes del servidor.

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

Cuando los caracteres llegan del servidor, se colocan en la cadena modificada-oración. Los caracteres continúan acumulándose en Sentence modificada hasta que la línea termina con un carácter de retorno de carro. Después de imprimir la oración en mayúscula, cerramos el socket del cliente:

```
clientSocket.close()
```

Esta última línea cierra el socket y, por lo tanto, cierra la conexión TCP entre el cliente y el servidor. Hace que TCP en el cliente envíe un mensaje TCP a TCP en el servidor (consulte la Sección 3.5).

TCPServer.py

Ahora echemos un vistazo al programa del servidor.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

Veamos ahora las líneas que difieren significativamente de UDPServer y TCPClient. Al igual que con TCPClient, el servidor crea un socket TCP con:

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

Similar a UDPServer, asociamos el número de puerto del servidor, serverPort, con este socket:

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

Pero con TCP, serverSocket será nuestro socket de bienvenida. Después de establecer esta puerta de bienvenida, esperaremos y escucharemos a algún cliente que toque la puerta:

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

Esta línea hace que el servidor escuche las 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 serverSocket, que crea un nuevo socket en el servidor, llamado connectionSocket, dedicado a este cliente en particular. Luego, el cliente y el servidor completan el protocolo de enlace, creando una conexión TCP entre el ClientSocket del cliente y el ConnectionSocket del servidor. Con la conexión TCP establecida, el cliente y el servidor ahora pueden enviarse bytes a través de la conexión. Con TCP, todos los bytes enviados desde un lado no solo tienen la garantía de llegar al otro lado, sino también la llegada en orden.

```
connectionSocket.close()
```

En este programa, después de enviar la frase modificada al cliente, cerramos el socket de conexión. Pero como serverSocket permanece abierto, otro cliente ahora puede llamar a la puerta y enviar al servidor una frase para modificar.

Esto completa nuestra discusión de la programación de socket en TCP. Se recomienda que ejecute los dos programas en dos hosts separados y también que los modifique para lograr objetivos ligeramente diferentes. Debe comparar el par de programas UDP con el par de programas TCP y ver en qué se diferencian. También debe realizar muchas de las asignaciones de programación de sockets descritas al final de los Capítulos 2, 4 y 7. Finalmente, esperamos que algún día, después de dominar estos y otros programas de sockets más avanzados, escriba su propia aplicación de red popular, y hágase rico, y famoso, y recuerda 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 aprendido acerca de la ubicua arquitectura cliente-servidor adoptada por muchas aplicaciones de Internet y hemos visto su uso en los protocolos HTTP, FTP, SMTP, POP3 y DNS. Hemos estudiado estos importantes protocolos de nivel de aplicación y sus correspondientes aplicaciones asociadas (la Web, la transferencia de archivos, el correo electrónico y el DNS) con cierto detalle. También hemos aprendido sobre la arquitectura P2P cada vez más frecuente y cómo se utiliza en muchas aplicaciones. Hemos examinado cómo se puede usar la API de socket para construir aplicaciones de red. Hemos recorrido el uso de sockets para servicios de transporte de extremo a extremo orientados a la conexión (TCP) y sin conexión (UDP). ¡El primer paso en nuestro viaje por la arquitectura de red en capas ahora está completo!

Al principio de este libro, en la Sección 1.1, proporcionamos una definición bastante vaga e independiente de un protocolo: “el formato y el orden de los mensajes intercambiados entre dos o más entidades comunicantes, así como las acciones tomadas en la transmisión y / o la recepción de un mensaje u otro evento”. El material de este capítulo, y en particular nuestro estudio detallado de los protocolos HTTP, FTP, SMTP, POP3 y DNS, ha agregado un contenido considerable a esta definición. Los protocolos son un concepto clave en la creación de redes; Nuestro estudio de los protocolos de aplicación nos ha brindado la oportunidad de desarrollar una sensación más intuitiva de lo que son los protocolos.

En la Sección 2.1, describimos los modelos de servicio que TCP y UDP ofrecen a las aplicaciones que los invocan. Analizamos estos modelos de servicio cuando desarrollamos aplicaciones simples que se ejecutan sobre TCP y UDP en la Sección 2.7. Sin embargo, hemos dicho poco sobre cómo TCP y UDP proporcionan estos modelos de servicio. Por ejemplo, sabemos que TCP proporciona un servicio de datos confiable, pero aún no hemos dicho cómo lo hace. En el siguiente capítulo, analizaremos con atención no solo el qué, sino también cómo y por qué de los protocolos de transporte.

Equipados con conocimiento sobre la estructura de la aplicación de Internet y los protocolos de nivel de aplicación, ahora estamos listos para avanzar más en la pila de protocolos y examinar la capa de transporte en el Capítulo 3.

2.9 Problemas y Preguntas de Repaso

2.9.1 Preguntas de repaso

2.9.2 Problemas