FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

**FACULTAD DE INGENIERIA U.N.A.M.**
**DIVISION DE EDUCACION CONTINUA**

## CENTRO DE INFORMACION Y DOCUMENTACION "ING. BRUNO MASCANZONI"

**El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.**

**Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:**

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

**Los materiales a disposición son:**

- **Libros.**
- **Tesis de posgrado.**
- **Noticias técnicas.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1980 a la fecha.**

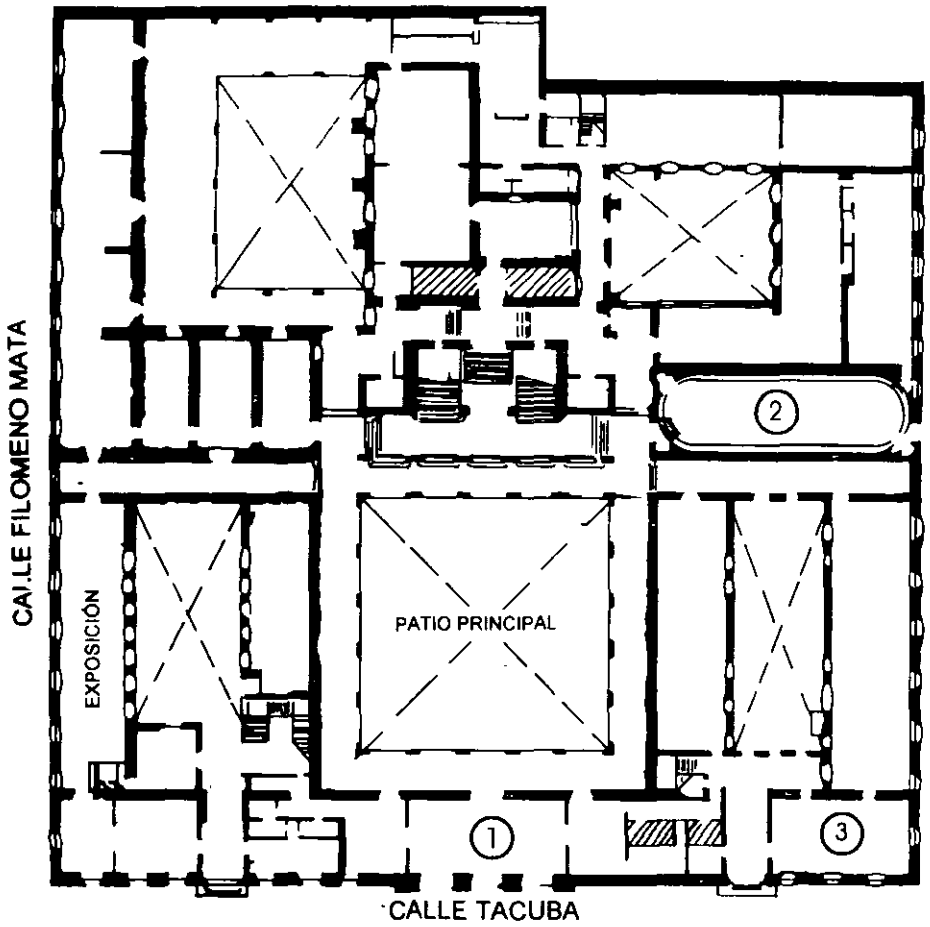
**En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.**

**El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.**

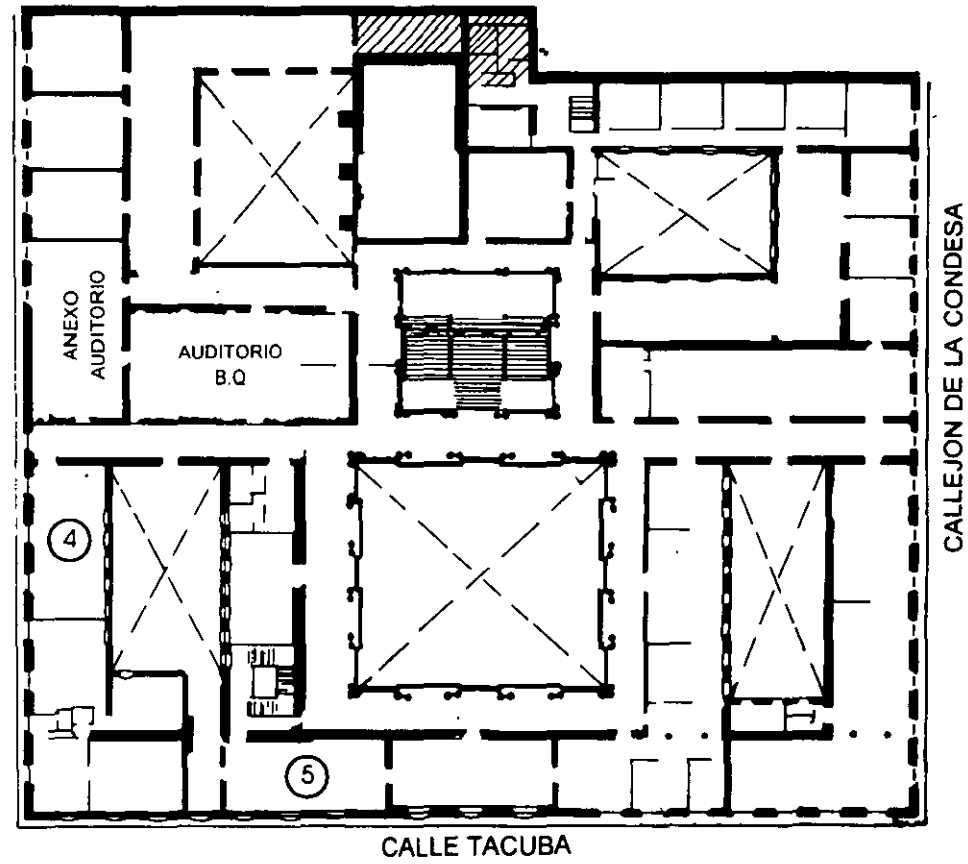
**El horario de servicio es de 10:00 a 19:30 horas de lunes a viernes.**

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

# PALACIO DE MINERIA

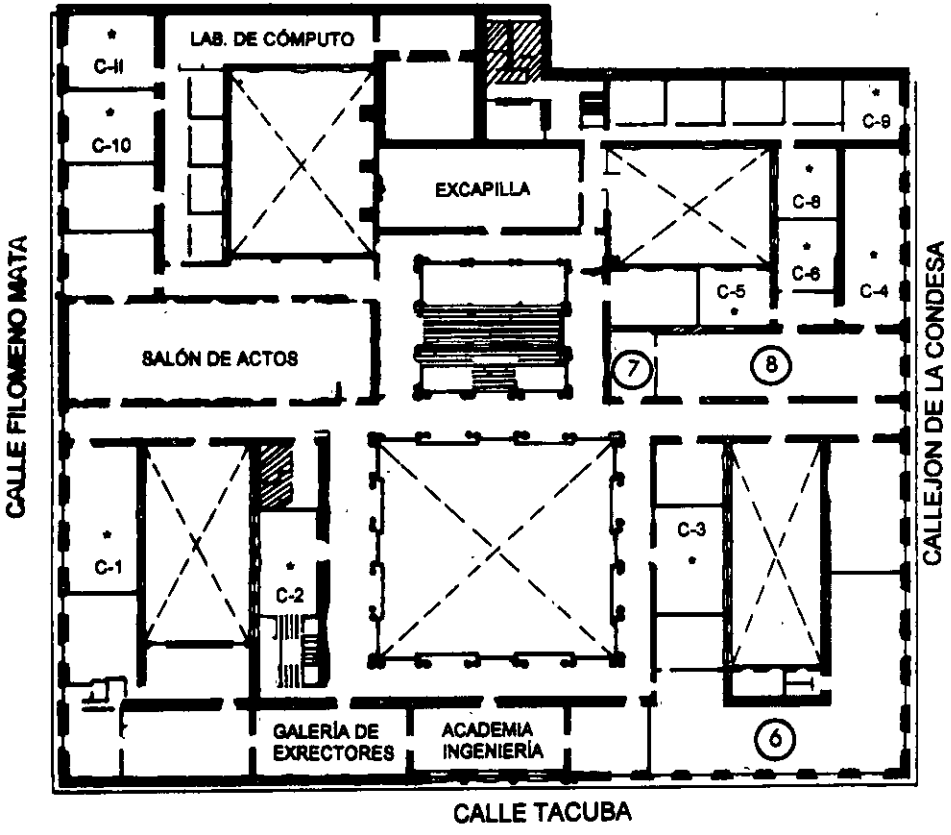



## PLANTA BAJA




## MEZZANINNE

# PALACIO DE MINERIA




**Ier. PISO**

## GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS


DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

Cursos Abiertos

# Diplomado en Redes WAN

## Módulo IV

## TCP /IP, Protocolos de Internet

(CA 127)

***Notas del Curso***

Octubre de 1998


Expositor:

**Ing. Maricarmen Herández**



Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Telefonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

# 5

# Transformación de direcciones Internet en direcciones físicas (ARP)

## 5.1 Introducción

Hemos descrito el esquema de direcciones TCP/IP, en el que cada anfitrión tiene asignada una dirección de 32 bits; asimismo, hemos dicho que una red de redes se comporta como una red virtual que utiliza sólo direcciones asignadas cuando envía y recibe paquetes. También hemos revisado muchas tecnologías de redes físicas y hemos notado que dos máquinas, en una red física, se pueden comunicar *solamente si conocen sus direcciones físicas de red.* Lo que no hemos mencionado es cómo un anfitrión o un ruteador transforman una dirección IP en la dirección física correcta cuando necesitan enviar un paquete a través de una red física. En este capítulo, se considera dicha transformación y se muestra de qué manera se implementa para los dos esquemas más comunes de direccionamiento de red física.

## 5.2 El problema de la asociación de direcciones

Considere que dos máquinas, *A* y *B*, comparten una red física. Cada una tiene asignada una dirección IP, $I_A$ e $I_B$, así como una dirección física, $P_A$ y $P_B$. El objetivo es diseñar un software de bajo nivel que oculte las direcciones físicas y permita que programas de un nivel más alto trabajen sólo

con direcciones de la red de redes. Sin embargo, la comunicación debe llevarse a cabo por medio de redes físicas, utilizando cualquier esquema de direcciones físicas proporcionado por el hardware. Suponga que la máquina *A* quiere enviar un paquete a la máquina *B* a través de una red física a la que ambas se conectan, pero *A* sólo tiene la dirección de red de redes $I_B$ de *B*. Surge, pues, la siguiente pregunta: ¿cómo transforma *A* dicha dirección en la dirección física $P_B$ de *B*?

La transformación de direcciones se tiene que realizar en cada fase a lo largo del camino, desde la fuente original hasta el destino final. En particular, surgen dos casos. Primero, en la última fase de entrega de un paquete, éste se debe enviar a través de una red física hacia su destino final. La computadora que envía el paquete tiene que transformar la dirección Internet de destino final en su dirección física. Segundo, en cualquier punto del camino, de la fuente al destino, que no sea la fase final, el paquete se debe enviar hacia un ruteador intermedio. Por lo tanto, el transmisor tiene que transformar la dirección Internet del ruteador en una dirección física.

El problema de transformar direcciones de alto nivel en direcciones físicas se conoce como *problema de asociación de direcciones* y se ha resuelto de muchas maneras. Algunos grupos de protocolos cuentan con tablas en cada máquina que contienen pares de direcciones, de alto nivel y físicas. Otros solucionan el problema al codificar direcciones de hardware en direcciones de alto nivel. Basarse en cualquiera de estos enfoques sólo hace que el direccionamiento de alto nivel sea muy delicado. En este capítulo, se tratan dos técnicas para la definición de direcciones utilizadas por los protocolos TCP/IP y se muestra cuándo es apropiada cada una de ellas.

## 5.3 Dos tipos de direcciones físicas

Existen dos tipos básicos de direcciones físicas, ejemplificados por Ethernet que tiene direcciones físicas grandes y fijas, así como por proNET que tiene direcciones físicas cortas y de fácil configuración. La asociación de direcciones es difícil para las redes de tipo Ethernet, pero resulta sencilla para redes como proNET. Consideraremos, primero, el caso más fácil.

## 5.4 Asociación mediante transformación directa

Considere una red token ring tipo proNET. Recuerde que, en el capítulo 2, vimos que proNET utiliza números enteros pequeños para sus direcciones físicas y permite que el usuario elija una dirección de hardware cuando instala una tarjeta de interfaz en una computadora. La clave para facilitar la definición de direcciones con dicho hardware de red radica en observar que, mientras se tenga la libertad de escoger tanto la dirección IP como la física, se puede hacer que ambas posean las mismas partes. Normalmente, una persona asigna direcciones IP con el campo hostid igual a 1, 2, 3, etcétera, y luego, cuando instala hardware de interfaz de red, selecciona una dirección física que corresponda a la dirección IP. Por ejemplo, el administrador de sistema podría seleccionar la dirección física 3 para una computadora que tenga la dirección IP 192.5.48.3, debido a que la dirección anterior es tipo *C* y tiene el campo de anfitrión igual a 3.

Para las redes como proNET, computar una dirección física basándose en una dirección IP es trivial. El cómputo consiste en extraer el campo de anfitrión de la dirección IP. La extracción es

computacionalmente eficiente pues sólo necesita unas cuantas instrucciones de máquina. La transformación es fácil de mantener porque se puede realizar sin consultar datos externos. Por último, es posible agregar nuevas máquinas a la red sin cambiar las asignaciones ya existentes ni recopilar los códigos.

Conceptualmente, escoger un esquema de numeración que facilite la asociación de direcciones significa seleccionar una función $f$ que transforme direcciones IP en direcciones físicas. El diseñador también puede ser capaz de seleccionar un esquema de numeración para direcciones físicas, dependiendo del hardware. Definir la dirección IP, $I_A$, implica computar:

$$P_A = f(I_A)$$

Queremos que el cómputo de $f$ sea eficiente. Si se constriñe el juego de direcciones físicas, puede ser posible realizar transformaciones eficientes, diferentes a la que se ejemplifica arriba. Por ejemplo, cuando se utiliza el IP en una red orientada a la conexión como ATM, no se pueden escoger las direcciones físicas. En redes como esa, una o más computadoras almacenan pares de direcciones, en donde cada par contiene una dirección Internet y su dirección física correspondiente. Por ejemplo, los valores se pueden almacenar dentro de una tabla en memoria, que se tiene que buscar. Para lograr que, en esos casos, la definición de direcciones sea eficaz, el software podría valerse de una función convencional de comprobación aleatoria para buscar dentro de la tabla. En el ejercicio 5.1, se sugiere una alternativa relacionada.

## 5.5 Definición mediante enlace dinámico

Para entender por qué la definición de direcciones es difícil para algunas redes, consideremos la tecnología Ethernet. Recuerde que, en el capítulo 2, vimos que cada interfaz Ethernet tiene asignada una dirección física de 48 bits desde la fabricación del producto. En consecuencia, cuando el hardware falla y se necesita reemplazar una interfaz Ethernet, la dirección física de la máquina cambia. Además, como la dirección Ethernet es de 48 bits, no hay posibilidad de codificarla en una dirección IP de 32 bits.[1]

Los diseñadores de los protocolos TCP/IP encontraron una solución creativa para el problema de la asociación de direcciones en redes como Ethernet, que tienen capacidad de difusión. La solución permite agregar nuevas máquinas a la red, sin tener que recopilar el código y no requiere tener una base de datos centralizada. Para evitar la definición de una tabla de conversiones, los diseñadores utilizan un protocolo de bajo nivel para asignar direcciones en forma dinámica. Conocido como *Protocolo de Asociación de Direcciones (ARP)*, éste proporciona un mecanismo razonablemente eficaz y fácil de mantener.

Como se muestra en la figura 5.1, la idea detrás de la asociación dinámica con ARP es muy sencilla: cuando el anfitrión $A$ quiere definir la dirección IP, $I_B$, transmite por difusión un paquete especial que pide al anfitrión que posee la dirección IP $I_B$, que responda con su dirección física, $P_B$. Todos los anfitriones, incluyendo a $B$, reciben la solicitud, pero sólo el anfitrión $B$ reconoce su propia dirección IP y envía una respuesta que contiene su dirección física. Cuando $A$ recibe la respues-

[1] Debido a que la transformación directa es más conveniente y eficiente que la asignación dinámica, la próxima generación de IP se está diseñando para permitir que las direcciones de 48 bits se puedan codificar en direcciones IP.

ta, utiliza la dirección física para enviar el paquete de red de redes directamente a *B*. Se puede resumir que:

*El Protocolo de Asociación de Direcciones ARP permite que un anfitrión encuentre la dirección física de otro anfitrión dentro de la misma red física con sólo proporcionar la dirección IP de su objetivo.*

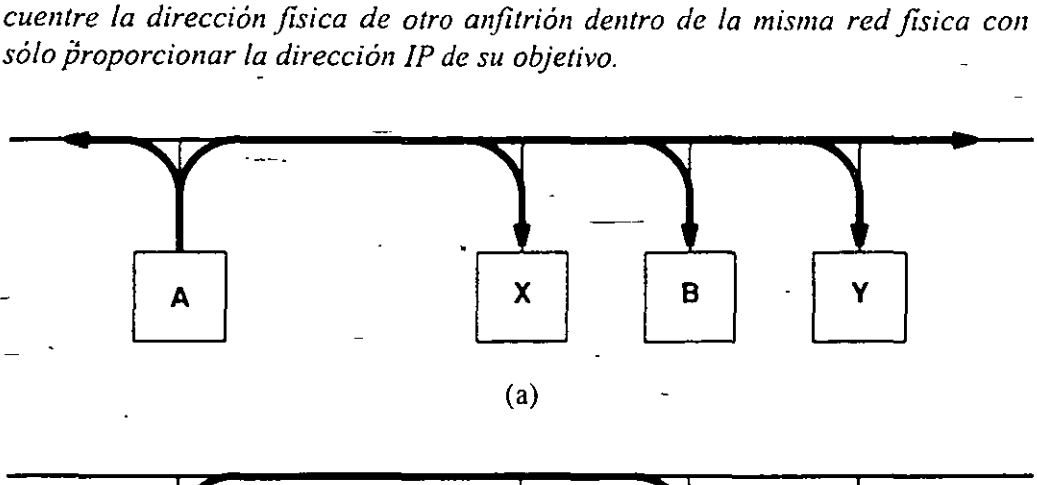



**Figura 5.1** Protocolo ARP. Para determinar la dirección física $P_B$ de *B*, desde su dirección IP, $I_B$, (a) el anfitrión *A* transmite por difusión una solicitud ARP que contiene $I_B$ a todas las máquinas en la red, y (b) el anfitrión *B* envía una respuesta ARP que contiene el par $(I_B, P_B)$.

## 5.6 Memoria intermedia para asociación de direcciones

Puede parecer extraño que para que *A* envíe un paquete a *B*, primero, tenga que transmitir una difusión que llegue a *B*. Podría parecer aún más extraño que *A* transmita por difusión la pregunta ¿cómo puedo llegar hasta ti?, en lugar de sólo transmitir por difusión el paquete que quiere entregar. Pero existe una razón importante para este intercambio. La difusión es demasiado cara para utilizarse cada vez que una máquina necesita transmitir un paquete a otra, debido a que requiere que cada máquina en la red procese dicho paquete. Para reducir los costos de comunicación, las computadoras, que utilizan ARP, mantienen una memoria intermedia de las asignaciones de dirección IP a dirección física recientemente adquiridas, para que no tengan que utilizar ARP varias veces. Siempre que una computadora recibe una respuesta ARP, ésta guarda la dirección IP del transmisor, así como la dirección de hardware correspondiente, en su memoria intermedia, para utilizarla en búsquedas posteriores. Cuando transmite un paquete, una computadora siempre busca, en su memoria intermedia, una asignación antes de enviar una solicitud ARP. Si una computadora en-

cuentra la asignación deseada en su memoria intermedia ARP, no necesitan transmitir una difusión la red. La experiencia nos indica que, como la mayor parte de la comunicación en red comprende más que la sola transferencia de un paquete, hasta una memoria intermedia pequeña es muy valiosa.

## 5.7 Refinamientos ARP

Se pueden lograr muchos refinamientos de ARP. Primero, observe que si el anfitrión *A* va a utilizar ARP porque necesita enviar algo a *B*, existe una alta posibilidad de que *B* necesite enviar algo a *A* en un futuro cercano. Para anticipar la necesidad de *B* y evitar tráfico de red adicional, *A* incluye su asignación de dirección IP como dirección física cuando envía una solicitud a *B*. *B* extrae la asignación de *A* de la solicitud, la graba en su memoria intermedia ARP y envía la respuesta hacia *A*. Segundo, nótese que, debido a que *A* transmite por difusión su solicitud inicial, todas las máquinas en la red la reciben y pueden extraer, así como grabar, en su memoria intermedia, la asignación de dirección IP como dirección física de *A*. Tercero, cuando a una máquina se le reemplaza la interfaz de anfitrión (por ejemplo, a causa de una falla en el hardware), su dirección física cambia. Las otras computadoras en la red, que tienen almacenada una asignación en su memoria intermedia ARP, necesitan ser informadas para que puedan cambiar el registro. Un sistema puede notificar a otros sobre una nueva dirección al enviar una difusión ARP cuando se inicia.

La siguiente regla resume los refinamientos:

> *El transmisor incluye, en cada difusión ARP, su asignación de dirección IP como dirección física; los receptores actualizan su información en memoria intermedia antes de procesar un paquete ARP.*

## 5.8 Relación de ARP con otros protocolos

ARP proporciona un mecanismo para transformar direcciones IP en direcciones físicas; ya hemos visto que algunas tecnologías de red no lo necesitan. El punto es que ARP sería totalmente innecesario si pudiéramos hacer que todo el hardware de red reconociera direcciones IP. Por lo tanto, ARP sólo impone un nuevo esquema de direccionamiento sobre cualquier mecanismo de direccionamiento de bajo nivel que el hardware utilice. La idea se puede resumir de la siguiente manera:

> *ARP es un protocolo de bajo nivel que oculta el direccionamiento físico subyacente de red, al permitir que se asigne una dirección IP arbitraria a cada máquina. Pensamos en ARP como parte del sistema físico de red, no como parte de los protocolos de red de redes.*

## 5.9 Implantación de ARP

De manera funcional, ARP está dividido en dos partes. La primera parte transforma una dirección IP en una dirección física cuando se envía un paquete y la segunda responde solicitudes de otras

máquinas. La definición de direcciones para los paquetes salientes parece muy clara, pero los pequeños detalles complican la implantación. Al tener una dirección IP de destino, el software consulta su memoria intermedia ARP para encontrar la transformación de la dirección IP a la dirección física. Si la conoce, el software extrae la dirección física, pone los datos en una trama utilizando esa dirección y envía la trama. Si no conoce la transformación, el software debe transmitir una difusión que contenga la solicitud ARP y esperar una respuesta.

La difusión de una solicitud ARP para encontrar una transformación de direcciones se puede volver compleja. La máquina de destino puede estar apagada o tan sólo muy ocupada para aceptar la solicitud. Si es así, el transmisor quizá no reciba la respuesta o la reciba con retraso. Debido a que Ethernet es un sistema de entrega con el mejor esfuerzo, también se puede perder la solicitud de difusión inicial ARP (en cuyo caso, el que la envía debe retransmitirla por lo menos una vez). Mientras tanto, el anfitrión tiene que almacenar el paquete original para que se pueda enviar ya que se haya asociado la dirección IP a la dirección de red.[2] De hecho, el anfitrión debe decidir si permite que otros programas de aplicación funcionen mientras realiza una solicitud ARP (la mayor parte de ellos lo permite). Si así es, el software debe manejar el hecho de que una aplicación genere solicitudes ARP adicionales para la misma dirección sin transmitir por difusión muchas solicitudes para un mismo objetivo.

Por último, considere el caso en el que la máquina *A* ya obtuvo una asignación para la máquina *B*, pero el hardware de *B* falla y es reemplazado. Aunque la dirección de *B* ha cambiado, las asignaciones en memoria temporal de *A* no lo han hecho, así que *A* utiliza una dirección de hardware que no existe, por lo que la recepción exitosa se vuelve imposible. En este caso se muestra por qué es importante tener software ARP que maneje de manera temporal la tabla de asignaciones y que remueva los registros después de un periodo establecido de tiempo. Claro está, el controlador de tiempo para un registro en la memoria temporal se debe reiniciar cada vez que llegue una difusión ARP que contenga la asignación (pero no se reinicia cuando el registro se utiliza para enviar un paquete).

La segunda parte del código ARP maneja paquetes que llegan por medio de la red. Cuando llega un paquete ARP, el software extrae la dirección IP del transmisor y la dirección del hardware, luego, examina la memoria temporal local para verificar si ya existe un registro para el transmisor. Si es así, el controlador actualiza el registro al sobreescribir la dirección física con la dirección obtenida del paquete. Después, el receptor procesa el resto del paquete ARP.

El receptor debe manejar dos tipos de paquetes ARP entrantes. Si llega una solicitud ARP, la máquina receptora debe verificar si es el objetivo de la solicitud (por ejemplo, si alguna otra máquina transmitió por difusión una solicitud de la dirección física del receptor). Si es así, el software ARP formula una respuesta al proporcionar su dirección física de hardware y la envía directamente al solicitante. El receptor también agrega el par de direcciones del transmisor a su memoria temporal si éstas no están presentes. Si la dirección IP mencionada en la solicitud ARP no corresponde a la dirección IP local, el paquete solicitará la transformación de alguna otra máquina en la red aunque podría ser ignorado.

El otro caso interesante sucede cuando llega una respuesta ARP. Dependiendo de la implantación, el controlador quizá necesite crear un registro en su memoria temporal o el registro se pueda crear cuando se genere la solicitud. En cualquiera de estos casos, una vez que se actualiza la memoria temporal, el receptor intenta encontrar una correspondencia entre la respuesta y una solicitud expedida con anterioridad. Por lo general, las respuestas llegan obedeciendo a una solicitud

[2] Si el retraso es significativo, el anfitrión puede descartar los paquetes salientes.

que se generó porque la máquina tiene que entregar un paquete. Entre el tiempo en que una máquina transmite por difusión su solicitud ARP y recibe la respuesta, los programas de aplicación o los protocolos de un nivel más alto pueden generar solicitudes adicionales para la misma dirección; el software debe recordar que ya envió una solicitud para no enviar más. Por lo común, el software ARP coloca los paquetes adicionales en una cola de espera. Una vez que llega la respuesta y se conoce la asignación de dirección, el software ARP remueve los paquetes de la cola de espera, pone cada paquete en una trama y utiliza la asignación de dirección para llenar la dirección física del destino. Si, con anterioridad, no expidió una solicitud de la dirección IP en la respuesta, la máquina actualizará el registro del transmisor en su memoria temporal y tan sólo dejará de procesar el paquete.

## 5.10 Encapsulación e identificación de ARP

Cuando los mensajes ARP viajan de una máquina a otra, se deben transportar en tramas físicas. En la figura 5.2, se muestra cómo se transporta el mensaje ARP en la porción de datos de una trama.

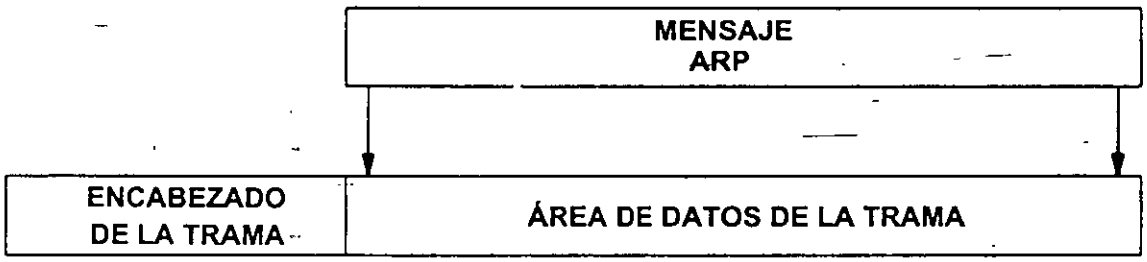



**Figura 5.2** Mensaje ARP encapsulado en una trama de red física.

Para identificar que la trama transporta una mensaje ARP, el transmisor asigna un valor especial al campo de tipo en el encabezado de la trama y coloca el mensaje ARP en el campo de datos de la misma. Cuando llega una trama a una computadora, el software de red utiliza el campo de tipo de trama para determinar su contenido. En la mayor parte de las tecnologías, se utiliza un solo valor para el tipo de todas las tramas que transportan un mensaje ARP —el software de red en el receptor debe examinar el mensaje ARP para distinguir entre solicitudes y respuestas. Por ejemplo, en una Ethernet, las tramas que transportan mensajes ARP tienen un campo de tipo de $0806_{16}$. Este es un valor estándar asignado por la autoridad para Ethernet; otras tecnologías de hardware de red emplean otros valores.

## 5.11 Formato del protocolo ARP

A diferencia de la mayor parte de los protocolos, los datos en los paquetes ARP no tienen un encabezado con formato fijo. Por el contrario, para hacer que ARP sea útil para varias tecnologías de

red, la longitud de los campos que contienen direcciones depende del tipo de red. Sin embargo, para hacer posible la interpretación de un mensaje ARP arbitrario, el encabezado incluye campos fijos cerca del comienzo, que especifican la longitud de las direcciones que se encuentran en los campos siguientes. De hecho, el formato de un mensaje ARP es lo suficientemente general como para permitir que sea utilizado con direcciones físicas arbitrarias y direcciones arbitrarias de protocolos. En el ejemplo de la figura 5.3 se muestra el formato de 28 octetos de un mensaje ARP que se utiliza en el hardware Ethernet (en el que las direcciones físicas tienen una longitud de 48 bits o de 6 octetos), cuando se asocian direcciones de protocolo IP (que tienen una longitud de 4 octetos).

En la figura 5.3, se muestra un mensaje ARP con 4 octetos por línea, formato estándar a través de todo este texto. Por desgracia, a diferencia de la mayor parte de los otros protocolos, los campos de longitud variable en los paquetes ARP no se alinean firmemente en fronteras de 32 bits, lo cual causa que el diagrama sea difícil de leer. Por ejemplo, la dirección de hardware del transmisor, etiquetada como *SENDER HA*, ocupa 6 octetos contiguos, por lo que abarca dos líneas en el diagrama.

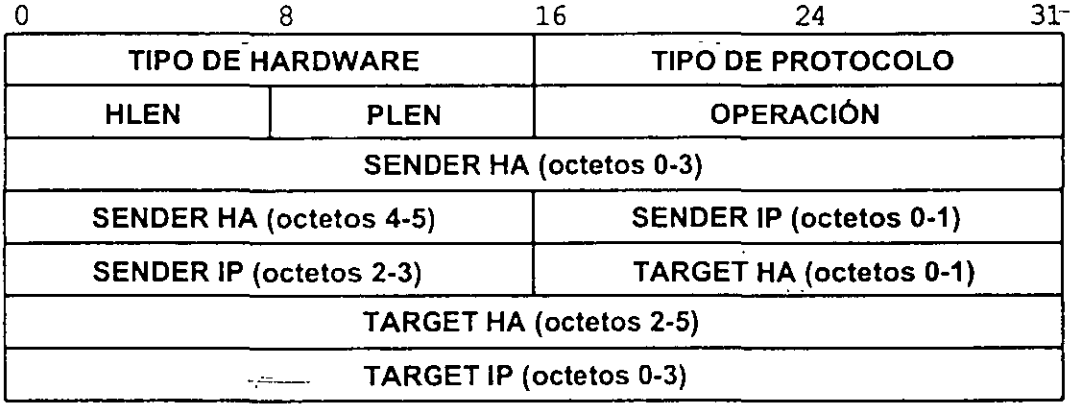



**Figura 5.3** Ejemplo del formato de mensaje ARP/RARP cuando se utiliza para la transformación de una dirección IP en una dirección Ethernet. La longitud de los campos depende del hardware y de la longitud de las direcciones de protocolos, que son de 6 octetos para una dirección Ethernet y de 4 octetos para una dirección IP.

El campo *HARDWARE TYPE* especifica un tipo de interfaz de hardware para el que el transmisor busca una respuesta; contiene el valor *1* para Ethernet. De forma similar, el campo *PROTOCOL TYPE* especifica el tipo de dirección de protocolo de alto nivel que proporcionó el transmisor: contiene $0800_{16}$ para la dirección IP. El campo *OPERATION* especifica una solicitud ARP (*1*), una respuesta ARP (*2*), una solicitud RARP[3] (*3*) o una respuesta RARP (*4*). Los campos *HLEN* y *PLEN* permiten que ARP se utilice con redes arbitrarias ya que éstas especifican la longitud de la dirección de hardware y la longitud de la dirección del protocolo de alto nivel. El transmisor proporciona sus direcciones IP y de hardware, si las conoce, en los campos *SENDER HA* y *SENDER IP*.

[3] En el siguiente capítulo se describe RARP, otro protocolo que utiliza el mismo formato de mensajes.

Cuando realiza una solicitud, el transmisor también proporciona la dirección IP del objetivo (ARP) o la dirección de hardware del objetivo (RARP), utilizando los campos *TARGET HA* y *TARGET IP*. Antes de que la máquina objetivo responda, completa las direcciones faltantes, voltea los pares de objetivo y transmisor, y cambia la operación a respuesta. Por lo tanto, una respuesta transporta las direcciones tanto de hardware como de IP del solicitante original, lo mismo que las direcciones de hardware e IP de la máquina para la que se realizó asignación.

## 5.12 Resumen

Las direcciones IP se asignan independientemente de la dirección física de hardware de una máquina. Para enviar un paquete de red de redes a través de una red física desde una máquina hacia otra, el software de red debe transformar la dirección IP en una dirección física de hardware y utilizar esta última para transmitir la trama. Si las direcciones de hardware son más pequeñas que las direcciones IP, se puede establecer una transformación directa al codificar la dirección física de una máquina dentro de su dirección IP. De otra forma, la transformación debe realizarse de manera dinámica. El Protocolo de Definición de Direcciones (ARP) realiza la definición dinámica de direcciones, utilizando sólo el sistema de comunicación de red de bajo nivel. ARP permite que las máquinas asocien direcciones sin tener un registro permanente de asignaciones.

Una máquina utiliza ARP para encontrar la dirección de hardware de otra máquina al transmitir por difusión una solicitud ARP. La solicitud contiene la dirección IP de la máquina de la que se necesita la dirección de hardware. Todas las máquinas en una red reciben la solicitud ARP. Si la solicitud corresponde a la dirección IP de una máquina, ésta responde al enviar una respuesta que contiene la dirección de hardware requerida. Las respuestas se dirigen a una sola máquina; no se transmiten por difusión.

Para lograr que ARP sea eficiente, cada máquina guarda en su memoria temporal las asignaciones de dirección IP a dirección física. Como el tráfico de una red de redes tiende a ser una secuencia de interacciones entre pares de máquinas, la memoria temporal elimina la mayor parte de las solicitudes ARP transmitidas por difusión.

## PARA CONOCER MÁS

El protocolo de definición de direcciones que aquí se utiliza está proporcionado por Plummer (RFC 826) y se ha convertido en un estándar de protocolos TCP/IP para red de redes. Dalal y Printis (1981) describen la relación entre las direcciones IP y las direcciones Ethernet; asimismo, Clark (RFC 814) trata en general las direcciones y las asignaciones. Parr (RFC 1029) analiza la definición de direcciones tolerante de fallas. Kirkpatrick y Recker (RFC 1166) especifican valores utilizados para identificar tramas de red en el documento Internet Numbers. En el volumen II de esta obra, se presenta el ejemplo de una ejecución ARP y se analiza el procedimiento respecto a la memoria temporal.

## EJERCICIOS

**5.1** Teniendo un pequeño grupo de direcciones físicas (números enteros positivos), ¿puede encontrar una función *f* y una asignación de direcciones IP, de tal forma que *f* transforme las direcciones IP, una por una, en direcciones físicas y que el cómputo de *f* sea eficiente? Pista: consulte la documentación sobre dispersión perfecta, (*perfect hasshing*).

**5.2** ¿En qué casos especiales un anfitrión conectado a una Ethernet no necesita utilizar ARP o una memoria temporal ARP antes de transmitir un datagrama IP?

**5.3** Un algoritmo común para manejar la memoria temporal ARP reemplaza el registro menos utilizado cuando agrega uno nuevo. ¿Bajo qué circunstancias este algoritmo podría generar un tráfico de red innecesario?

**5.4** Lea cuidadosamente el estándar. ¿ARP debe actualizar la memoria intermedia si ya existe un registro antiguo para cierta dirección IP? ¿Por qué?

**5.5** ¿El software ARP debe modificar la memoria intermedia inclusive cuando recibe información sin solicitarla de manera específica? ¿Por qué?

**5.6** Cualquier implantación ARP que utilice una memoria temporal de tamaño fijo puede fallar cuando se utiliza en una red que tiene muchos anfitriones y mucho tráfico ARP. Explique cómo.

**5.7** A veces, se refieren a ARP como una debilidad de seguridad. Explique por qué.

**5.8** Explique qué puede pasar si el campo de dirección de hardware en una respuesta ARP se corrompe durante la transmisión. Pista: algunas implantaciones ARP no remueven los registros en memoria temporal si se utilizan con frecuencia.

**5.9** Suponga que la máquina *C* recibe una solicitud ARP de *A* buscando al objetivo *B*, y suponga que *C* tiene la asignación de $I_B$ a $P_B$ en su memoria temporal. ¿*C* debe contestar la solicitud? Explíquelo.

**5.10** ¿Cómo puede utilizar ARP una estación de trabajo cuando se inicia para descubrir si alguna otra máquina en la red la está personificando? ¿Cuáles son las desventajas del esquema?

**5.11** Explique de qué manera el envío de paquetes hacia direcciones no existentes en una Ethernet remota puede generar tráfico de difusión excesivo en esa red.

# 6

# Determinación en el arranque de una dirección Internet (RARP)

## 6.1 Introducción

Hasta ahora sabemos que las direcciones físicas de red son de bajo nivel y dependientes del hardware. Asimismo, entendemos que cada máquina que utiliza el TCP/IP tiene asignada una o más direcciones IP de 32 bits, independientemente de su dirección de hardware. Los programas de aplicación siempre utilizan la dirección IP cuando especifican un destino. Los anfitriones y los ruteadores deben utilizar direcciones físicas para transmitir datagramas a través de las redes subyacentes; confían en los esquemas de asociación de direcciones como ARP para realizar los enlaces.

Por lo general, la dirección IP de una máquina se mantiene en el área secundaria de almacenaje, en donde el sistema operativo la encuentra en el momento del arranque. Ahora bien, surge la siguiente pregunta, ¿cómo puede una máquina que no cuenta con disco permanente determinar su dirección IP? El problema es crítico para las estaciones de trabajo que almacenan sus archivos en un servidor remoto, ya que dichas máquinas necesitan una dirección IP antes de poder utilizar protocolos TCP/IP estándar para transferencia de archivos a fin de obtener su imagen inicial de arranque. En este capítulo, se examina la cuestión de cómo obtener una dirección IP y se describe el protocolo que muchas máquinas utilizan antes del arranque desde un servidor remoto de archivos.

Debido a que una imagen de sistema operativo que tiene una dirección IP específica, limitada dentro del código, no se puede utilizar en muchas computadoras, los diseñadores por lo general tratan de evitar la compilación de una dirección IP en el código del sistema operativo o dentro del software de apoyo. En particular, el código de iniciación que se encuentra, a menudo, en la Memoria de Sólo Lectura (ROM), generalmente se construye para que la misma imagen pueda correr en

muchas máquinas. Cuando un código así inicia su ejecución, en una máquina sin disco, utiliza el hardware para contactar un servidor y, con ello, obtener su dirección IP.

El proceso de iniciación parece paradójico: una máquina se comunica con un servidor remoto a fin de obtener la dirección que necesita para la comunicación. Sin embargo, esta paradoja es sólo aparente, ya que la máquina *sabe* cómo comunicarse. Puede utilizar su dirección física para comunicarse a través de una sola red. Por tanto, la máquina debe recurrir de manera temporal al direccionamiento físico de red, de la misma forma en que el sistema operativo utiliza el direccionamiento físico de memoria para establecer tablas de página para el direccionamiento virtual. Una vez que la máquina conoce su dirección IP, se puede comunicar a través de una red de redes.

La idea detrás de encontrar una dirección IP es sencilla: una máquina que necesita conocer su dirección envía una solicitud a un *servidor*[1] en otra máquina y espera a que el servidor, a su vez, mande la respuesta. Asumimos que el servidor accesa un disco en el que guarda una base de datos de las direcciones internas. En la solicitud, sólo la máquina que necesita saber su dirección de red de redes se tiene que identificar, para que el servidor pueda buscar la dirección correcta y responder. Tanto la máquina que genera la solicitud como el servidor que responde utilizan direcciones físicas de red durante su breve comunicación. ¿Cómo sabe el solicitante la dirección física de un servidor? Por lo general, no lo sabe tan sólo transmite por difusión la solicitud a todas las máquinas de la red local. Y entonces, uno o más servidores responden.

Una máquina que transmite por difusión una solicitud de dirección sólo tiene que identificarse. ¿Qué información se puede incluir en esa solicitud que únicamente identificará a la máquina? Cualquier sufijo único para la identificación del hardware (por ejemplo, el número serial de la CPU). Sin embargo, la identificación tiene que basarse en algo que un programa en ejecución pueda obtener con facilidad. El objetivo es crear una sola imagen de software que pueda ejecutarse en un procesador cualquiera. Además, la longitud o el formato de la información específica de la CPU puede variar entre los diferentes modelos de procesadores, y nos gustaría imaginar un servidor que acepte solicitudes realizadas por todas las máquinas en la red por medio del uso de un solo formato.

## 6.2 Protocolo de asociación de direcciones por réplica (RARP)

Los diseñadores de los protocolos TCP/IP se dieron cuenta de que ya existe otra pieza disponible para la identificación exclusiva, a saber, la dirección física de red de la máquina. Utilizar la dirección física como identificación única tiene dos ventajas. Debido a que un anfitrión obtiene sus direcciones físicas del hardware de interfaz de red, dichas direcciones siempre están disponibles y no tienen que limitarse al código de iniciación. Como la información de identificación depende de la red y no del modelo o la marca de la CPU, todas las máquinas en una red proporcionarán identificadores únicos y uniformes. Por lo tanto, el problema se convierte en el inverso de la asociación de direcciones; una vez dada una dirección física de red, invente un esquema que permita que un servidor la transforme en una dirección de red de redes.

Una máquina sin disco utiliza un protocolo TCP/IP para red de redes llamado *RARP (Protocolo inverso de asociación de direcciones)* a fin de obtener su dirección IP desde un servidor. RARP es una adaptación al protocolo ARP que vimos en el capítulo anterior y utiliza el mismo formato de mensajes mostrado en la figura 5.3. En la práctica, el mensaje RARP enviado para solici-

[1] En el capítulo 19 se analizan los servidores más detalladamente.

tar una dirección de red de redes es un poco más general de lo que hemos subrayado arriba: permite que una máquina solicite la dirección IP de una tercera máquina tan fácilmente como si solicitara la suya. También lo permite cuando se trata de muchos tipos de redes físicas.

Al igual que un mensaje ARP, un mensaje RARP se envía de una máquina a otra, encapsulado en la porción de datos de una trama de red. Por ejemplo, una trama Ethernet que transporta una solicitud RARP tiene el preámbulo usual, las direcciones Ethernet tanto fuente como destino y campos de tipo de paquete al comienzo de la trama. El tipo de trama contiene el valor $8035_{16}$ para identificar que el contenido de la trama contiene un mensaje RARP. La porción de datos de la trama contiene el mensaje RARP de 28 octetos.

En la figura 6.1, se ilustra la manera en que un anfitrión utiliza RARP. El que envía transmite por difusión una solicitud RARP especificada como máquina transmisora y receptora, y proporciona su dirección física de red en el campo de dirección de hardware objetivo. Todas las máquinas en la red reciben la solicitud, pero sólo las autorizadas para proporcionar el servicio RARP la procesan y envían la respuesta; dichas máquinas se conocen de manera informal como *servidores RARP*. Para que RARP funcione correctamente, la red debe contener por lo menos un servidor RARP.

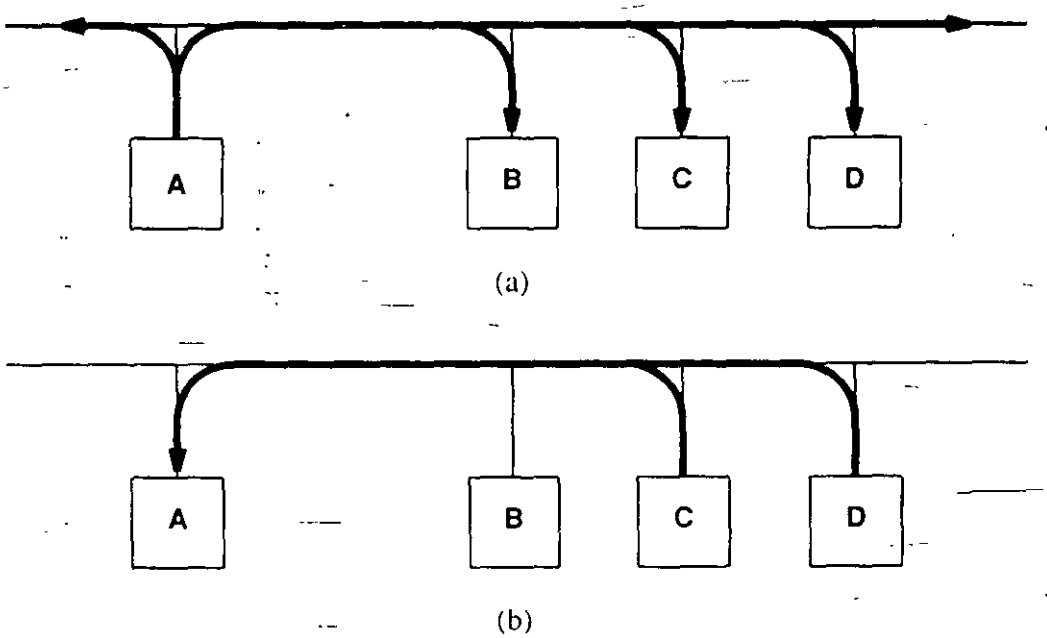



**Figura 6.1** Ejemplo de un intercambio en el que se utiliza el protocolo RARP. (a) la máquina *A* transmite por difusión una solicitud RARP especificándose como destino y (b) las máquinas autorizadas para proporcionar el servicio RARP (*C* y *D)* responden directamente a *A*.

Una vez llenado el campo de dirección de protocolo objetivo, los servidores contestan las solicitudes, cambian el tipo de mensaje de *solicitud* a *respuesta* y envían ésta de vuelta directamente a la máquina que la solicitó. La máquina original recibe respuesta de todos los servidores RARP, aunque sólo se necesite una contestación.

Tenga en mente que toda la comunicación entre la máquina que busca su dirección IP y el servidor que la proporciona se debe llevar a cabo utilizando sólo una red física. Además, el protocolo permite que un anfitrión pregunte sobre un objetivo arbitrario. Por lo tanto, el transmisor proporciona su dirección de hardware separada de la dirección de hardware del objetivo y el servidor tiene cuidado de enviar la respuesta a la dirección de hardware del transmisor. En una Ethernet, tener un campo para la dirección de hardware del transmisor podría parecer redundante ya que la información también está contenida en el encabezado de la trama Ethernet. Sin embargo, no todo el hardware Ethernet proporciona al sistema operativo acceso al encabezado de la trama física.

## 6.3 Temporización de las transacciones RARP

Como cualquier comunicación en una red de entrega con el mejor esfuerzo, las solicitudes RARP son susceptibles de pérdida o corrupción. Ya que RARP utiliza directamente la red física, ningún otro software de protocolos cronometrará la respuesta ni retransmitirá la solicitud; es el software RARP el que debe manejar estas tareas. En general, RARP se utiliza sólo en redes de área local, como Ethernet, en las que la probabilidad de falla es muy baja. Sin embargo, si una red tiene sólo un servidor RARP, dicha máquina quizá no sea capaz de manejar la carga y, por tanto, los paquetes se pierdan.

Algunas estaciones de trabajo que dependen de RARP para realizar su proceso de iniciación reintentan éste una y otra vez hasta que reciben una respuesta. Otras implementaciones, al cabo de un par de intentos, lo suspenden indicando que hay fallas y evitan con ello inundar la red con tráfico innecesario de difusión (por ejemplo, en el caso de que el servidor no esté disponible). En una Ethernet, la falla de red no sucede solamente por la sobrecarga del servidor. Hacer que el software RARP retransmita rápidamente puede causar un efecto indeseable: inundar con más tráfico un servidor congestionado. Valerse de un retraso largo garantiza que los servidores tengan tiempo suficiente para satisfacer la solicitud y generar una respuesta.

## 6.4 Servidores RARP primarios y de respaldo

La principal ventaja de tener varias máquinas funcionando como servidores RARP es que se obtiene un sistema más confiable. Si un servidor falla, o está demasiado congestionado como para responder, otro servidor contestará la solicitud. Por tanto, es mucho más probable que el servicio siempre se encuentre disponible. La principal desventaja de utilizar varios servidores es que, cuando una máquina transmite por difusión una solicitud RARP, la red se sobrecarga en el momento en que todos los servidores intentan responder. Por ejemplo, en una Ethernet, emplear muchos servidores RARP ocasiona que la probabilidad de colisión sea muy alta.

¿Cómo se puede distribuir el servicio RARP para mantenerlo a disposición y confiable sin sufrir el costo por solicitudes excesivas y simultáneas? Existen, por lo menos, dos posibilidades y ambas implican el retraso de las respuestas. En la primera, a cada máquina que realiza solicitudes RARP se le asigna un *servidor primario*. Bajo circunstancias normales, sólo el servidor primario de la máquina responde a su solicitud RARP. Todos los servidores que no son primarios reciben la

solicitud, pero únicamente registran su tiempo de llegada. Si el servidor primario no está disponible, la máquina original cronometrará el tiempo de respuesta y, si ésta no aparece, transmitirá de nuevo por difusión la solicitud. Cuando un servidor no primario recibe una segunda copia de una solicitud RARP, poco tiempo después de la primera, éste responde.

En la segunda posibilidad, se emplea un esquema similar pero se intenta evitar que todos los servidores no primarios transmitan de manera simultánea las respuestas. Cada máquina no primaria que recibe una solicitud computa un retraso en forma aleatoria y, luego, envía la respuesta. Bajo circunstancias normales, el servidor primario responde de inmediato y las respuestas sucesivas se retrasan, así que existe una probabilidad muy baja de que lleguen al mismo tiempo. Cuando el servidor primario no está disponible, la máquina solicitante pasa por un corto retraso antes de recibir una respuesta. Al escoger con cuidado los tiempos de retraso, el diseñador puede asegurar que las máquinas solicitantes no hagan transmisiones por difusión antes de recibir una respuesta.

## 6.5 Resumen

En el arranque del sistema, una computadora que no tenga un disco permanente debe contactar a un servidor para encontrar su dirección IP antes de que se pueda comunicar por medio del TCP/IP. Examinamos el protocolo RARP, el cual utiliza el direccionamiento físico de red para obtener la dirección de red de redes de la máquina. El mecanismo RARP proporciona la dirección de hardware físico de la máquina de destino para identificar de manera única el procesador y transmite por difusión la solicitud RARP. Los servidores en la red reciben el mensaje, buscan la transformación en una tabla (de manera presumible en su almacenamiento secundario) y responden al transmisor. Una vez que una máquina obtiene su dirección IP, la guarda en la memoria y no vuelve a utilizar RARP hasta que se inicia de nuevo.

## PARA CONOCER MÁS

Los detalles de RARP se encuentran en Finlayson, *et. al.* (RFC 903). Finlayson (RFC 906) describe el proceso de iniciación de una estación de trabajo utilizando el protocolo TFTP. Bradley y Brown (RFC 1293) especifican un protocolo relacionado, *ARP inverso*. El protocolo ARP inverso permite que una computadora busque en la máquina ubicada en el extremo opuesto de una conexión de hardware para determinar su dirección IP. ARP fue diseñado para las computadoras conectadas por medio de Frame Relay. En el volumen II de esta obra se describe un ejemplo de una implementación de RARP.

En el capítulo 21, se consideran alternativas a RARP, conocidas como BOOTP y DHCP, una extensión más reciente. A diferencia del esquema de determinación de direcciones de bajo nivel que proporciona RARP, tanto BOOTP como DHCP están construidos con protocolos de más alto nivel, como IP y UDP. En el capítulo 21, se comparan los dos enfoques y se analizan las ventajas y debilidades de cada uno.

## EJERCICIOS

**6.1** Un servidor RARP puede transmitir por difusión respuestas RARP a todas las máquinas o transmitir cada respuesta de manera directa a la máquina que lo solicite. Caracterice una tecnología de red en la que sea benéfica la transmisión de respuestas por difusión a todas las máquinas.

**6.2** RARP es un protocolo enfocado de manera específica, en el sentido de que sus respuestas sólo contienen una pieza de información (por ejemplo, la dirección IP solicitada). Cuando una computadora se inicia, por lo general necesita conocer su nombre además de su dirección Internet. Amplíe el concepto de RARP para proporcionar la información adicional.

**6.3** ¿Qué tanto más grandes serán las tramas Ethernet cuando se añada información a RARP como se describe en el ejercicio anterior?

**6.4** Agregando un segundo servidor RARP a una red aumenta su confiabilidad. ¿Tiene sentido agregar un tercero? ¿Por qué sí o por qué no?

**6.5** Las estaciones de trabajo sin disco utilizan RARP para obtener sus direcciones IP, pero siempre asumen que la respuesta proviene del servidor de archivos de la estación de trabajo. La máquina sin disco trata de obtener una imagen de iniciación de ese servidor. Si no recibe una respuesta, la estación de trabajo entra en un bucle infinito de transmisión de solicitudes. Explique cómo agregar un servidor RARP de respaldo a una configuración de este tipo puede causar que la red se congestione con difusiones. Pista: piense en las fallas de alimentación de corriente.

**6.6** Monitoree una red local mientras reinicia varias computadoras. ¿Cuál utiliza RARP?

**6.7** Los servidores RARP de respaldo mencionados en el texto se valen de la llegada de una segunda solicitud dentro de un corto periodo de tiempo para realizar una respuesta. Considere el esquema de servidor RARP, que hace que todos los servidores contesten la primera solicitud, pero evita el congestionamiento al hacer que cada servidor retrase por un periodo aleatorio la respuesta. ¿Bajo qué circunstancias dicho diseño daría mejores resultados que el diseño descrito en el texto?

# 7

# Protocolo Internet: entrega de datagramas sin conexión

## 7.1 Introducción

En capítulos anteriores, revisamos algunas partes del hardware y del software de red que hacen posible la comunicación entre redes, explicamos la tecnología de red subyacente y la asignación de direcciones. En este capítulo, se explica el principio fundamental de la entrega sin conexión y analiza cómo se proporciona ésta por medio del *Protocolo Internet (IP)*, uno de los dos protocolos más importantes utilizados en el enlace de redes. Estudiaremos el formato de los datagramas IP y veremos como éstos forman la base para toda la comunicación en una red de redes. En los siguientes dos capítulos, continuaremos nuestro examen del Protocolo Internet analizando el ruteo de datagramas y el manejo de errores.

## 7.2 Una red virtual

En el capítulo 3, se analizó una arquitectura de red de redes en la que los ruteadores conectan múltiples redes físicas. Considerar esto exclusivamente como una arquitectura puede ser engañoso, debido a que el enfoque se daría hacia la interfaz que proporciona la red de redes al usuario, sin considerar la tecnología de interconexión.

*Un usuario concibe una red de redes como una sola red virtual que interconecta a todos los anfitriones, y a través de la cual es posible la comunicación; la arquitectura subyacente permanece oculta y es irrelevante.*

En cierto sentido, una red de redes es una abstracción de una red física porque, en los niveles inferiores, proporciona la misma funcionalidad: acepta paquetes y los entrega. En niveles superiores el software de la red de redes aporta la mayor parte de las funciones más elaboradas que percibe el usuario.

## 7.3 Arquitectura y filosofía de Internet

Conceptualmente, como se muestra en la figura 7.1, una red de redes TCP/IP proporciona tres conjuntos de servicios; su distribución en la figura sugiere una dependencia entre ellos. En el nivel inferior, un servicio de entrega sin conexión proporciona el fundamento sobre el cual se apoya el resto. En el siguiente nivel, un servicio de transporte confiable proporciona una plataforma de alto nivel de la cual dependen las aplicaciones. Exploraremos cada uno de estos servicios, entendiendo qué es lo que proporciona cada uno y considerando los protocolos asociados a ellos.

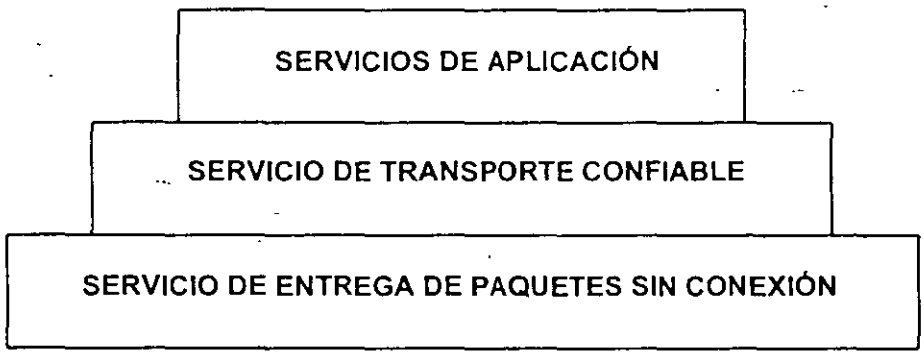



**Figura 7.1** Las tres capas conceptuales de los servicios de Internet.

## 7.4 El concepto de entrega no confiable

Aun cuando podemos asociar un software de protocolo con cada uno de los servicios en la figura 7.1, la razón para identificarlos como partes conceptuales de la red de redes es que éstos constituyen claramente un aspecto esencial respecto a la filosofía del diseño. El punto a considerar es el siguiente:

*El software de Internet está diseñado en torno a 3 conceptos de servicios de red arreglados jerárquicamente; muchos de los éxitos alcanzados se deben a esta arquitectura sorprendentemente robusta y adaptable.*

Una de las ventajas más significativas de esta separación de conceptos es que es posible reemplazar un servicio sin afectar a los otros. Así, los investigadores y desarrolladores pueden proceder concurrentemente en los tres.

## 7.5 Sistema de entrega sin conexión

El servicio más importante de la red de redes consiste en un sistema de entrega de paquetes. Técnicamente, el servicio se define como un sistema de entrega de paquetes sin conexión y con el mejor esfuerzo, análogo al servicio proporcionado por el hardware de red que opera con un paradigma de entrega con el mejor esfuerzo. El servicio se conoce como *no confiable* porque la entrega no está garantizada. Los paquetes se pueden perder, duplicar, retrasar o entregar sin orden, pero el servicio no detectará estas condiciones ni informará al emisor o al receptor. El servicio es llamado *sin conexión* dado que cada paquete es tratado de manera independiente de todos los demás. Una secuencia de paquetes que se envían de una computadora a otra puede viajar por diferentes rutas, algunos de ellos pueden perderse mientras otros se entregan. Por último, se dice que el servicio trabaja con base en una *entrega con el mejor esfuerzo* porque el software de red de redes hace un serio intento por entregar los paquetes. Esto es, la red de redes no descarta paquetes caprichosamente; la no confiabilidad aparece sólo cuando los recursos están agotados o la red subyacente falla.

## 7.6 Propósito del Protocolo Internet

El protocolo que define el mecanismo de entrega sin conexión y no confiable es conocido como *Protocolo Internet* y, por lo general se le identifica por sus iniciales, *IP*.[1] El protocolo IP proporciona tres definiciones importantes. Primero, define la unidad básica para la transferencia de datos utilizada a través de una red de redes TCP/IP. Es decir, especifica el formato exacto de todos los datos que pasarán a través de una red de redes TCP/IP. Segundo, el software IP realiza la función de *ruteo*, seleccionando la ruta por la que los datos serán enviados. Tercero, además de aportar especificaciones formales para el formato de los datos y el ruteo, el IP incluye un conjunto de reglas que le dan forma a la idea de entrega de paquetes no confiable. Las reglas caracterizan la forma en que los anfitriones y ruteadores deben procesar los paquetes, cómo y cuándo se deben generar los mensajes de error y las condiciones bajo las cuales los paquetes pueden ser descartados. El IP es una parte fundamental del diseño de la red de redes TCP/IP, que a veces se conoce como *tecnología basada en el IP*.

Iniciaremos nuestra consideración acerca del IP en este capítulo revisando el formato de los paquetes y sus especificaciones. Dejaremos para capítulos posteriores los temas de ruteo y manejo de errores.

[1] De la abreviatura IP se genera la expresión "dirección IP"

## 7.7 El datagrama de Internet

La analogía entre una red física y una red de redes TCP/IP es muy fuerte. En una red física, la unidad de transferencia es una trama que contiene un encabezado y datos, donde el encabezado contiene información sobre la dirección de la fuente (física) y la del destino. La red de redes llama a esta unidad de transferencia básica *datagrama Internet*, a veces *datagrama IP* o simplemente *datagrama*. Como una trama común de red física, un datagrama se divide en áreas de encabezado y datos. También, como una trama, el encabezado del datagrama contiene la dirección de la fuente y del destino, contiene también un campo de tipo que identifica el contenido del datagrama. La diferencia, por supuesto, es que el encabezado del datagrama contiene direcciones IP en tanto que el encabezado de la trama contiene direcciones físicas. La figura 7.2 muestra la forma general de un datagrama:


**Figura 7.2** Forma general de un datagrama IP, la estructura análoga de TCP/IP con respecto a la trama de una red. El IP especifica el formato de un encabezado incluyendo las direcciones de fuente y destino. El IP no especifica el formato del área de datos; el datagrama se puede utilizar para transportar datos arbitrarios.

### 7.7.1 Formato de datagrama

Ahora que hemos descrito la disposición general de un datagrama IP, podemos observar su contenido con mayor detalle. La figura 7.3 muestra el arreglo de campos en un datagrama.

Debido a que el proceso de los datagramas se da en el software, el contenido y el formato no está condicionado por ningún tipo de hardware. Por ejemplo, el primer campo de 4 bits en un datagrama (*VERS*) contiene la versión del protocolo IP que se utilizó para crear el datagrama. Esto se utiliza para verificar que el emisor, el receptor y cualquier ruteador entre ellos proceda de acuerdo con el formato del datagrama. Todo software IP debe verificar el campo de versión antes de procesar un datagrama para asegurarse de que el formato corresponde al tipo de formato que espera el software. Si hay un cambio en el estándar, las máquinas rechazarán los datagramas con versiones de protocolo que difieren del estándar, evitando con ello que el contenido de los datagramas sea mal interpretado debido a un formato obsoleto. El protocolo IP actual trabaja con la versión número 4.

El campo de longitud encabezado *(HLEN)*, también de 4 bits, proporciona el encabezado del datagrama con una longitud medida en palabras de 32 bits. Como podremos ver, todos los campos del encabezado tienen longitudes fijas excepto para el campo *OPTIONS* de *IP* y su correspondiente campo *PADDING*. El encabezado más común, que no contiene opciones ni rellenos, mide 20 octetos y tiene un campo de longitud de encabezado igual a 5.

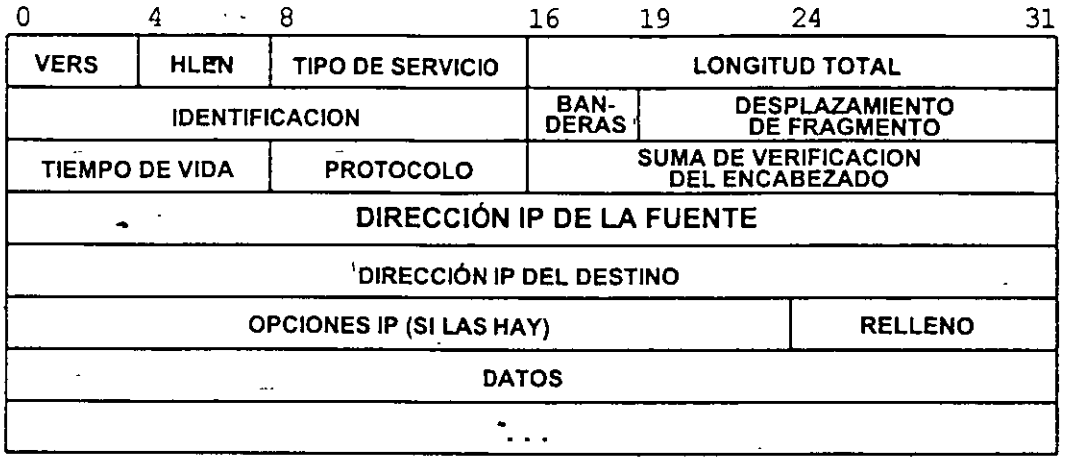



**Figura 7.3** Formato de un datagrama Internet, la unidad básica de transferencia en una red de redes TCP/IP.

El campo *TOTAL LENGTH* proporciona la longitud del datagrama IP medido en octetos, incluyendo los octetos del encabezado y los datos. El tamaño del área de datos se puede calcular restando la longitud del encabezado (*HLEN*) de *TOTAL LENGTH*. Dado que el campo *TOTAL LENGTH* tiene una longitud de 16 bits, el tamaño máximo posible de un datagrama IP es de $2^{16}$ o 65,535 octetos. En la mayor parte de las aplicaciones, ésta no es una limitación severa, pero puede volverse una consideración importante en el futuro, si las redes de alta velocidad llegan a transportar paquetes de datos superiores a los 65,535 octetos.

### 7.7.2 Tipo de datagramas de servicios y prioridad de datagramas

Conocido informalmente como *Type Of Service (TOS)*, el campo de 8 bits *SERVICE TYPE* especifica cómo debe manejarse el datagrama; el campo está subdividido en 5 subcampos, como se muestra en la figura 7.4:

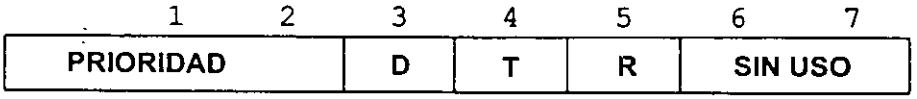



**Figura 7.4** Los cinco subcampos que componen el campo *SERVICE-TYPE* de 8 bits.

Tres bits *PRECEDENCE* especifican la prioridad del datagrama, con valores que abarcan de 0 (prioridad normal) a 7 (control, de red), permitiendo con ello indicar al emisor la importancia de cada datagrama. Aun cuando la mayor parte del software de los anfitriones y los ruteadores ignora el tipo de servicio, éste es un concepto importante dado que proporciona un mecanismo que permi-

te controlar la información que tendrá prioridad en los datos. Por ejemplo, si todos los anfitriones y ruteadores responden a la prioridad, es posible implementar algoritmos de control de congestionamiento que no se vean afectados por el mismo congestionamiento que desean controlar.

Los bits *D, T* y *R* especifican el tipo de transporte deseado para el datagrama. Cuando está activado, el bit *D* solicita procesamiento con retardos cortos, el bit *T* solicita un alto desempeño y el bit *R* solicita alta confiabilidad. Por supuesto, no es posible para una red de redes garantizar siempre el tipo de transporte solicitado (por ejemplo, éste sería el caso si no se encuentra una ruta adecuada). De esta manera, debemos pensar en una solicitud de transporte como en una simple indicación para los algoritmos de ruteo, no como en un requerimiento obligatorio. Si un ruteador no conoce más que una posible ruta para alcanzar un destino determinado, puede utilizar el campo de tipo de transporte para seleccionar una con las características más cercanas a la petición deseada. Por ejemplo, supongamos que un ruteador puede seleccionar entre una línea arrendada de baja capacidad y una conexión vía satélite con un gran ancho de banda (pero con un retardo alto). Los datagramas que acarrean la información tecleada por un usuario hacia una computadora remota pueden tener el bit *D* activado, solicitando que la entrega sea lo más rápida posible, mientras que el transporte de datagramas en la transferencia de un archivo de datos grande podría tener activado el bit *T*, solicitando que el recorrido se haga a través de una ruta que incluya un satélite de alta capacidad.

También es importante para la realización del proceso que los algoritmos de ruteo seleccionen de entre las tecnologías de red física subyacente, las características de retardo, desempeño y confiabilidad. Con frecuencia, una tecnología dada intercambiará una característica por otra (por ejemplo, un alto desempeño implicará un mayor retardo). Así, la idea es proporcionar un algoritmo de ruteo como si se tratara de una indicación de qué es lo más importante; rara vez es necesario especificar los tres tipos de servicio juntos. En resumen:

> *Hemos visto la especificación del tipo de transporte como una indicación para el algoritmo de ruteo que ayuda en la selección de una ruta entre varias hacia un destino, con base en el conocimiento de las tecnologías de hardware disponibles en esas rutas. Una red de redes no garantiza la realización del tipo de transporte solicitado.*

### 7.7.3 Encapsulación de datagramas

Antes de que podamos entender los siguientes campos de un datagrama es importante considerar cómo los datagramas se relacionan con las tramas de las redes físicas. Comenzaremos con una pregunta: "¿qué tan grande puede ser un datagrama?" A diferencia de las tramas de las redes físicas que pueden ser reconocidas por el hardware, los datagramas son manejados por el software. Estos pueden tener cualquier longitud seleccionada por el diseño de protocolo. Hemos visto que el formato de los datagramas actuales asignan solamente 16 bits al campo de longitud total, limitando el datagrama a un máximo de 65,535 octetos. Sin embargo, este límite puede modificarse en versiones de protocolos recientes.

Las limitaciones más importantes en el tamaño de un datagrama se dan en la práctica misma. Sabemos que, como los datagramas se mueven de una máquina a otra, éstos deben transportarse siempre a través de una red física subyacente. Para hacer eficiente el transporte en la red de redes,

queremos garantizar que cada datagrama puede viajar en una trama física distinta. Esto es, queremos que nuestra abstracción de un paquete de red física se transforme directamente en un paquete real si es posible.

La idea de transportar un datagrama dentro de una trama de red es conocida como *encapsulación*. Para la red subyacente un datagrama es como cualquier otro mensaje que se envía de una máquina a otra. El hardware no reconoce el formato del datagrama ni entiende las direcciones de destino IP. Así, como se muestra en la figura 7.5, cuando una máquina envía un datagrama IP hacia otra, el datagrama completo viaja en la porción de datos de la trama de red.

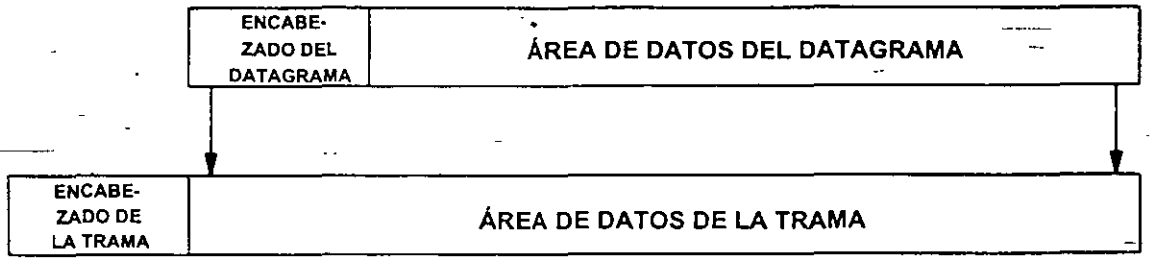



**Figura 7.5** Encapsulación de un datagrama IP en una trama. La red física trata al datagrama entero, incluyendo el encabezado, como si se tratara de datos.

### 7.7.4 Tamaño de datagrama, MTU de red y fragmentación

En un caso ideal, el datagrama IP completo se ajusta dentro de una trama física haciendo que la transmisión a través de la red física sea eficiente.[2] Para alcanzar esta eficiencia, los diseñadores de IP tendrían que seleccionar un tamaño máximo de datagrama, de manera que el datagrama siempre se ajuste dentro de una trama. Pero ¿qué tamaño de trama deberían seleccionar? Después de todo, un datagrama debe viajar a través de muchos tipos de redes físicas conforme se mueve a través de una red de redes hacia su destino final.

Para entender el problema, necesitamos considerar un hecho a propósito del hardware de red: cada tecnología de conmutación de paquetes establece un límite superior fijo para la cantidad de datos que pueden transferirse en una trama física. Por ejemplo, Ethernet limita la transferencia de datos a 1,500[3] octetos, mientras que FDDI permite aproximadamente 4,470 octetos por trama. Nos referiremos a estos límites como la *unidad de transferencia máxima* de una red (*maximum transfer unit*, o *MTU* por sus siglas en inglés). El tamaño de MTU puede ser muy pequeño: algunas tecnologías de hardware limitan la transferencia a 128 octetos o menos. La limitación de los datagramas para que se ajusten a la MTU más pequeña posible en una red de redes hace que la transferencia sea ineficiente cuando estos datagramas pasan a través de una red que puede transportar tramas de tamaño mayor. Sin embargo, permitir que los datagramas sean más grandes que la MTU

[2] Un campo en el encabezado de trama por lo general identifica el tipo de datos que se están transportando; Ethernet utiliza el valor tipo $0800_{16}$ para especificar que el área de datos contiene un datagrama IP encapsulado.

[3] El límite de 1500 proviene de las especificaciones de Ethernet; cuando se utilizaba con un encabezado SNAP, el estándar 802.3 de IEEE limitaba los datos a 1492 octetos. Las plantaciones de algunos vendedores permiten transferencias ligeramente mayores.

mínima de una red, en una red de redes, puede significar que un datagrama no siempre se ajuste dentro de una sola trama de red.

La selección debería ser obvia: el punto a considerar en el diseño de una red de redes es ocultar la tecnología de red subyacente y hacer la comunicación conveniente para el usuario. Así, en lugar de diseñar datagramas que se ajusten a las restricciones de la red física, el software TCP/IP selecciona un tamaño de datagrama más conveniente desde el principio y establece una forma para dividir datagramas en pequeños fragmentos cuando el datagrama necesita viajar a través de una red que tiene una MTU pequeña. Las pequeñas piezas dentro de un datagrama dividido se conocen con el nombre de *fragmentos* y el proceso de división de un datagrama se conoce como *fragmentación*.

Como se ilustra en la figura 7.6, la fragmentación por lo general se da en un ruteador a lo largo del trayecto entre la fuente del datagrama y su destino final. El ruteador recibe un datagrama de una red con una MTU grande y debe enviarlo a una red en la que la MTU es más pequeña que el tamaño del datagrama.

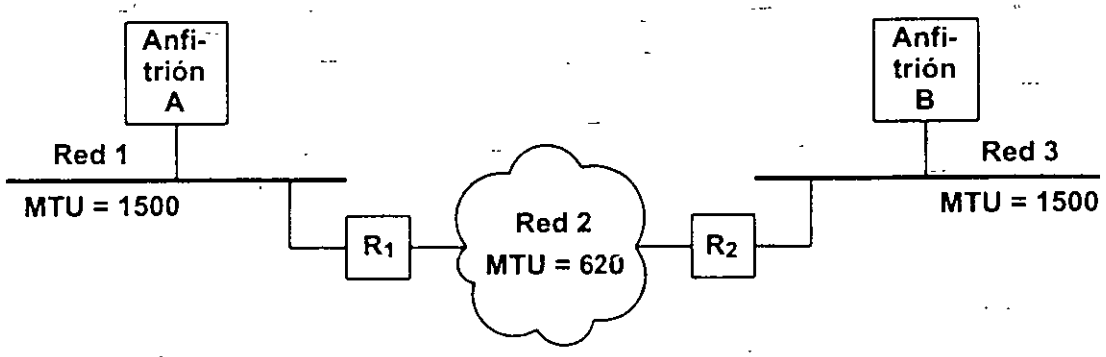



**Figura 7.6** Una ilustración de los casos en que se presenta la fragmentación. El ruteador $R_1$ fragmenta un datagrama extenso para enviarlo desde $A$ hacia $B$; $R_2$ fragmenta un datagrama extenso para enviarlo desde $B$ hacia $A$.

En la figura, ambos anfitriones están conectados directamente a una red Ethernet, la cual tiene una MTU de 1,500 octetos. Así, ambos anfitriones pueden generar y enviar datagramas con un máximo de 1,500 octetos de largo. El trayecto entre ambos, sin embargo, incluye una red con una MTU de 620. Si el anfitrión $A$ envía al anfitrión $B$ un datagrama mayor a 620 octetos, el ruteador $R_1$ fragmentará el datagrama. De la misma forma, si $B$ envía un datagrama grande hacia $A$, el ruteador $R_2$ fragmentará el datagrama.

El tamaño de cada fragmento se selecciona de manera que cada uno de éstos pueda transportarse a través de la red subyacente en una sola trama. Además, dado que el IP representa el desplazamiento de datos en múltiplos de 8 octetos, el tamaño del fragmento debe seleccionarse de modo que sea un múltiplo de 8. Por supuesto, al seleccionar el múltiplo de 8 octetos más cercano a la MTU de la red no es usual dividir el datagrama en fragmentos de tamaños iguales; los últimos fragmentos por lo general son más cortos que los otros. Los fragmentos se deben *reensamblar* para producir una copia completa del datagrama original, antes de que pueda procesarse en su lugar de destino.

El protocolo IP no limita los datagramas a un tamaño pequeño, ni garantiza que los datagramas grandes serán entregados sin fragmentación. La fuente puede seleccionar cualquier tamaño de datagrama que considere apropiado; la fragmentación y el reensamblado se dan automáticamente sin que la fuente deba realizar ninguna acción especial. Las especificaciones IP establecen que los ruteadores pueden aceptar datagramas con una longitud equivalente al valor máximo de la MTU de las redes a las que están conectados. Además, un ruteador siempre maneja datagramas de hasta 576 octetos. (Los anfitriones también son configurados para aceptar y reensamblar, si es necesario, datagramas de por lo menos 576 octetos.)

Fragmentar un datagrama significa dividirlo en varios segmentos. Podría ser sorprendente aprender que cada fragmento tiene el mismo formato que el datagrama original. La figura 7.7 muestra el resultado de la fragmentación.

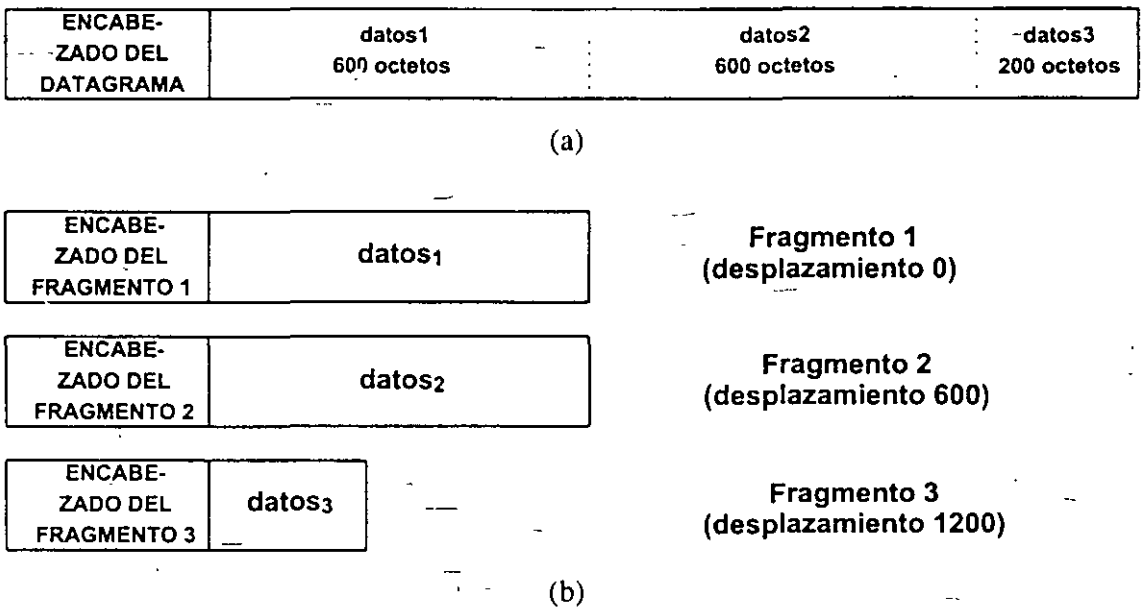



**Figura 7.7** (a) Un datagrama original que transporta 1400 octetos de datos y (b) los tres fragmentos para la red con una MTU de 620. Los encabezados 1 y 2 tienen activado el bit *más fragmentos*. El desplazamiento se muestra como un número decimal de octetos; estos valores se deben dividir entre 8 para obtener el valor en el que se localiza el encabezado de los fragmentos.

Cada fragmento contiene un encabezado de datagrama que duplica la mayor parte del encabezado del datagrama original (excepto por un bit en el campo *FLAGS* que muestra que éste es un fragmento), seguido por tantos datos como puedan ser acarreados en el fragmento siempre y cuando la longitud total se mantenga en un valor menor a la MTU de la red en la que debe viajar.

### 7.7.5 Reensamblado de fragmentos

¿Un datagrama debe ser reensamblado luego de pasar a través de una red o los fragmentos deben transportarse hasta el anfitrión final antes de ser reensamblados? En una red de redes TCP/IP, una vez que un datagrama se ha fragmentado, los fragmentos viajan como datagramas separados hacia su destino final donde serán reensamblados. Preservar los fragmentos en todo el trayecto hasta su destino final tiene dos desventajas. Primero, dado que los datagramas no son reemsamblados inmediatamente después de pasar a través de una red con una MTU pequeña, los fragmentos pequeños deben transportarse en esa forma desde el punto de fragmentación hasta el destino final. Reensamblar los datagramas en el destino final puede implicar que el proceso se realice con cierta ineficiencia: aun cuando se encuentre en una red física con una capacidad de MTU grande después del punto de fragmentación, ésta será atravesada por fragmentos pequeños. Segundo, si se pierde cualquier fragmento, el datagrama no podrá reensamblarse. La máquina de recepción hace que arranque un *temporizador de reensamblado* cuando recibe un fragmento inicial. Si el temporizador termina antes de que todos los fragmentos lleguen, la máquina de recepción descartará los fragmentos sin procesar el datagrama. Así, la probabilidad de perder un datagrama se incrementa con la fragmentación ya que la pérdida de un solo fragmento provoca la pérdida del datagrama completo.

Aún considerando desventajas menores, la realización del reensamblado en el destino final trabaja bien. Esto permite que cada fragmento se pueda rutear de manera independiente sin necesidad de que ruteadores intermedios almacenen o reensamblen fragmentos.

### 7.7.6 Control de fragmentación

Tres campos en el encabezado del datagrama, *IDENTIFICATION, FLAGS* y *FRAGMENT OFFSET*, controlan la fragmentación y el reensamblado de los datagramas. El campo *IDENTIFICATION* contiene un entero único que identifica al datagrama. Recordemos que cuando un ruteador fragmenta un datagrama, éste copia la mayor parte de los campos del encabezado del datagrama dentro de cada fragmento. El campo *IDENTIFICATION* debe copiarse. Su propósito principal es permitir que el destino tenga información acerca de qué fragmentos pertenecen a qué datagramas. Conforme llega cada fragmento, el destino utiliza el campo *IDENTIFICATION* junto con la dirección de la fuente del datagrama para identificar el datagrama. Las computadoras que envían datagramas IP deben generar un valor único para el campo *IDENTIFICATION* por cada datagrama.[4] Hay una técnica utilizada por el software IP que establece un contador global en memoria, lo incrementa cada vez que se crea un datagrama nuevo y asigna el resultado al campo del datagrama *IDENTIFICATION*.

Recordemos que cada fragmento tiene exactamente el mismo formato que un datagrama completo. Para un fragmento, el campo *FRAGMENT OFFSET* especifica el desplazamiento en el datagrama original de los datos que se están acarreando en el fragmento, medido en unidades de 8 octetos,[5] comenzando con un desplazamiento igual a cero. Para reensamblar el datagrama, el destino debe obtener todos los fragmentos comenzando con el fragmento que tiene asignado un despla-

[4] En teoría, en las retransmisiones de un datagrama se puede acarrear el mismo campo *IDENTIFICATION* que en el original; en la práctica, los protocolos de alto nivel por lo general realizan la retransmisión dando como resultado un datagrama nuevo con su propio campo *IDENTIFICATION*.

[5] Para ahorrar espacio en el encabezado, los desplazamientos se especifican en múltiplos de 8 octetos.

zamiento igual a *0* hasta el fragmento con el desplazamiento de mayor valor. Los fragmentos no necesariamente llegarán en orden, además no hay comunicación entre el ruteador que fragmentó el datagrama y el destino que trata de reensamblarlo.

Los 2 bits de orden menor del campo de 3 bits *FLAGS* controlan la fragmentación. Por lo general, el software de aplicación que utiliza TCP/IP no se ocupa de la fragmentación debido a que tanto la fragmentación como el reensamblado son procedimientos automáticos que se dan a bajo nivel en el sistema operativo, invisible para el usuario final. Sin embargo, para probar el software de red de redes o depurar problemas operacionales, podría ser importante probar el tamaño de los datagramas en los que se presenta la fragmentación. El primer bit de control ayuda en esta prueba especificando en qué momento se debe fragmentar un datagrama. Se le conoce como bit de *no fragmentación* porque cuando está puesto a *1* especifica que el datagrama no debe fragmentarse. Una aplicación podría seleccionar no permitir la fragmentación cuando sólo el datagrama completo es útil. Por ejemplo, consideremos la secuencia de iniciación de una computadora, en la que una máquina comienza a ejecutar un pequeño programa en ROM y utiliza la red de redes para solicitar una primera iniciación, y otra máquina envía de regreso una imagen de memoria. Si el software ha sido diseñado así, necesitará la imagen completa, pues de otra forma no le será útil; por ello, el datagrama debe tener activado el bit de *no fragmentación*. Cada vez que un ruteador necesita fragmentar un datagrama que tiene activado el bit de *no fragmentación*, el ruteador descartará el datagrama y devolverá un mensaje de error a la fuente.

El bit de orden inferior en el campo *FLAGS* especifica si el fragmento contiene datos intermedios del datagrama original o de la parte final. Este bit es conocido como *more fragments* (*más fragmentos*). Para entender por qué este bit es necesario, consideremos el software IP en el destino final cuando trata de reensamblar un datagrama. Este recibirá los fragmentos (es posible que en desorden) y necesitará saber cuándo ha recibido todos los fragmentos del datagrama. Cuando un fragmento llega, el campo *TOTAL LENGTH* en el encabezado consulta el tamaño del fragmento y no el tamaño del datagrama original; de esta manera el destino no puede utilizar el campo *TOTAL LENGHT* para determinar si ha reunido todos los fragmentos. El bit *más fragmentos* resuelve este problema con facilidad: cada vez que, en el destino, se recibe un fragmento con el bit *más fragmentos* desactivado, se sabe que este fragmento acarrea datos del extremo final del datagrama original. De los campos *FRAGMENT OFFSET* y *TOTAL LENGHT* se puede calcular la longitud del datagrama original. Examinando *FRAGMENT OFFSET* y *TOTAL LENGHT* en el caso de todos los fragmentos entrantes, un receptor puede establecer en qué momento los fragmentos que ha reunido contienen toda la información necesaria para reensamblar el datagrama original completo.

### 7.7.7 Tiempo de vida (time to live o TTL)

El campo *TIME TO LIVE* especifica la duración, en segundos, del tiempo que el datagrama tiene permitido permanecer en el sistema de red de redes. La idea es sencilla e importante: cada vez que una máquina introduce un datagrama dentro de la red de redes, se establece un tiempo máximo durante el cual el datagrama puede permanecer ahí. Los ruteadores y los anfitriones que procesan los datagramas deben decrementar el campo *TIME TO LIVE* cada vez que pasa un datagrama y eliminarlo de la red de redes cuando su tiempo ha concluido.

Una estimación exacta de este tiempo es difícil dado que los ruteadores por lo general no conocen el tiempo de tránsito por las redes físicas. Unas pocas reglas simplifican el procedimiento y

hacen fácil el manejo de datagramas sin relojes sincronizados. En primer lugar, cada ruteador, a lo largo de un trayecto, desde una fuente hasta un destino, es configurado para decrementar por *1* el campo *TIME TO LIVE* cuando se procesa el encabezado del datagrama. Sin embargo, para manejar casos de ruteadores sobrecargados que introducen largos retardos, cada ruteador registra el tiempo local cuando llega un datagrama, y decrementa el *TIME TO LIVE* por el número de segundos que el datagrama permanece dentro del ruteador esperando que se le despache.

Cada vez que un campo *TIME TO LIVE* llega a cero, el ruteador descarta el datagrama y envía un mensaje de error a la fuente. La idea de establecer un temporizador para los datagramas es interesante ya que garantiza que los datagramas no viajarán a través de la red de redes indefinidamente, aun cuando si una tabla de ruteo se corrompa y los ruteadores direccionen datagramas en un ciclo.

### 7.7.8 Otros campos de encabezado de datagrama

El campo *PROTOCOL* es análogo al campo tipo en una trama de red. El valor en el campo *PROTOCOL* especifica qué protocolo de alto nivel se utilizó para crear el mensaje que se está transportando en el área *DATA* de un datagrama. En esencia, el valor de *PROTOCOL* especifica el formato del área *DATA*. La transformación entre un protocolo de alto nivel y el valor entero utilizado en el campo *PROTOCOL* debe administrarlo por una autoridad central para garantizar el acuerdo entre los enteros utilizados en Internet.

El campo *HEADER CHECKSUM* asegura la integridad de los valores del encabezado. La suma de verificación IP se forma considerando al encabezado como una secuencia de enteros de 16-bits (en el orden de los octetos de la red), sumándolos juntos mediante el complemento aritmético a uno, y después tomando el complemento a uno del resultado. Para propósitos de cálculo de la suma de verificación, el campo *HEADER CHECKSUM* se asume como igual a cero.

Es importante notar que la suma de verificación sólo se aplica para valores del encabezado IP y no para los datos. Separar la suma de verificación para el encabezado y los datos tiene ventajas y desventajas. Debido a que el encabezado por lo general ocupa menos octetos que los datos, tener una suma de verificación separada disminuye el tiempo de procesamiento y ruteo, los cuales sólo necesitan calcular la suma de verificación del encabezado. La separación también permite a los protocolos de alto nivel seleccionar su propio esquema de suma de verificación para los datos. La mayor desventaja es que los protocolos de alto nivel se ven forzados a añadir su propia suma de verificación o corren el riesgo de que las alteraciones de datos no sean detectadas.

Los campos *SOURCE IP ADDRESS* y *DESTINATION IP ADDRESS* contienen direcciones IP de 32 bits de los datagramas del emisor y del receptor involucrado. Aun cuando los datagramas sean dirigidos a través de muchos ruteadores inmediatos, los campos de fuente y destino nunca cambian; éstos especifican la dirección IP de la fuente original y del destino final.[6]

El campo marcado con el nombre *DATA* en la figura 7.3 muestra el comienzo del área de datos de un datagrama. Su longitud depende, por supuesto, de qué es lo que se está enviando en el datagrama. El campo *OPTIONS* de *IP* que se analiza a continuación tiene una longitud variable. El campo señalado como *PADDING* depende de las opciones seleccionadas. Este representa un grupo de bits puestos en cero que podrían ser necesarios para asegurar que la extensión del encabezado

[6] Se hace una excepción cuando el datagrama incluye una lista de opciones de la fuente.

sea un múltiplo exacto de 32 bits (recordemos que el campo de longitud del encabezado se especifica en unidades formadas por palabras de 32 bits).

## 7.8 Opciones para los datagramas Internet

El campo *OPTIONS* del *IP* aparece a continuación de la dirección de destino y no se requiere en todos los datagramas; las opciones se incluyen en principio para pruebas de red o depuración. Sin embargo, el procesamiento de las opciones es parte integral del protocolo IP, por lo tanto, todos los estándares de implementaciones se deben incluir.

La longitud del campo *OPTIONS* de *IP* varía dependiendo de qué opción sea seleccionada. Algunas opciones tienen una longitud de un octeto; éstas consisten en un solo octeto de *código de opción*. Otras tienen longitudes variables. Cuando las opciones están presentes en un datagrama, aparecen contiguas, sin separadores especiales entre ellas. Cada opción consiste en un solo octeto de código de opción que debe llevar a continuación un solo octeto y un conjunto de octetos de datos para cada opción. El octeto de código de opción se divide en tres campos como se muestra en la figura 7.8.

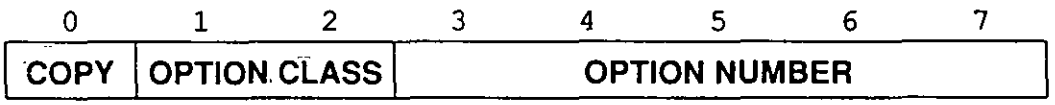



**Figura 7.8** División del octeto de código de opción en tres campos de 1, 2 y 5 bits.

El campo consiste en una bandera de 1 bit, llamada *COPY*, un segmento de 2 bits, *OPTION CLASS*, y un segmento de 5 bits, *OPTION NUMBER*. La bandera *COPY* controla la forma en que los ruteadores tratan las opciones durante la fragmentación. Cuando el bit *COPY* está puesto a *1*, especifica que la opción se debe copiar en todos los fragmentos. Cuando está puesto a cero el bit

| Option Class | Significado |
|---|---|
| 0 | Control de red o datagrama |
| 1 | Reservado para uso futuro |
| 2 | Depuración y medición |
| 3 | Reservado para uso futuro |

**Figura 7.9** Clases de opciones IP, como se codifican en los bits de *OPTION CLASS*, en un octeto de código de opción.

*COPY* significa que la opción sólo se debe copiar dentro del primer fragmento y no en todos los fragmentos.

Los bits *OPTION CLASS* y *OPTION NUMBER* especifican la clase general de opción y establecen una opción específica en esta clase. La tabla en la figura 7.9 muestra como se asignan las clases.

La tabla en la figura 7.10 lista las opciones posibles que pueden acompañar a un datagrama IP y muestra los valores para *OPTION CLASS* y *OPTION NUMBER*. Como se muestra en la lista, la mayor parte de las opciones se utiliza con propósito de control.

| Option Class | Option Number | Longitud | Descripción |
|---|---|---|---|
| 0 | 0 | - | Fin de la lista de opciones. Se utiliza si las opciones no terminan al final del encabezado (ver también campo de relleno de encabezado). |
| 0 | 1 | - | No operación (se utiliza para alinear octetos en una lista de opciones). |
| 0 | 2 | 11 | Seguridad y restricciones de manejo (para aplicaciones militares). |
| 0 | 3 | var | Ruteo no estricto de fuente. Se utiliza para rutear un datagrama a través de una trayectoria específica. |
| 0 | 7 | var | Registro de ruta. Se utiliza para registrar el trayecto de una ruta. |
| 0 | 8 | 4 | Identificador de flujo. Se utiliza para transportar un identificador de flujo SATNET (Obsoleto). |
| 0 | 9 | var | Ruteo estricto de fuente. Se utiliza para establecer la ruta de un datagrama en un trayecto específico. |
| 2 | 4 | var | Sello de tiempo Internet. Se usa para registrar sellos de hora a lo largo de una ruta. |

**Figura 7.10** Las ocho opciones posibles IP con su clase en forma numérica y los códigos de número. El valor *var* en la columna de longitud significa *variable*.

### 7.8.1 Opción de registro de ruta

Las opciones de ruteo y sello de hora (*timestamp*) son las más interesantes porque proporcionan una manera de monitorear o controlar la forma en que la red de redes maneja las rutas de los datagramas. La opción *registro de ruta* permite a la fuente crear una lista de direcciones IP y arreglarla para que cada ruteador que maneje el datagrama añada su propia dirección IP a la lista. La figura 7.11 muestra el formato de la opción de registro de ruta.

Como se describe arriba, el campo *CODE* contiene la clase de opción y el número de opción (*0* y *7* para el registro de rutas). El campo *LENGTH* especifica la longitud total de la opción como aparece en el datagrama IP, incluyendo los 3 primeros octetos. El campo comienza con un *FIRST IP ADDRESS* que comprende el área reservada para registrar las direcciones de la red de redes. El

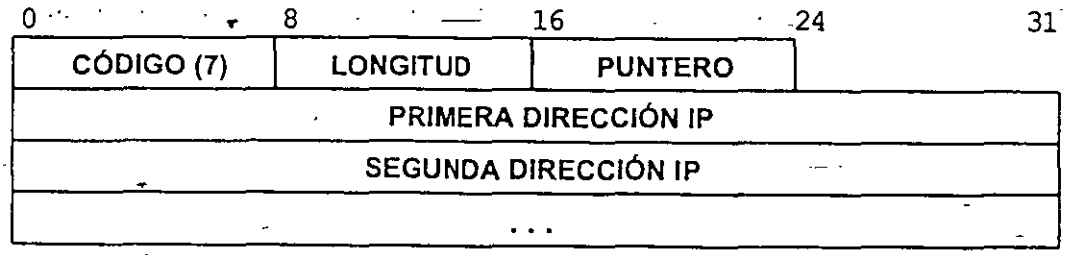



**Figura 7.11** Formato de una opción de registro de ruta en un datagrama IP. La opción comienza con tres octetos seguidos inmediatamente por una lista de direcciones. Aun cuando el diagrama muestra direcciones en unidades de 32 bits, éstas no están alineadas con ningún octeto en la frontera de un datagrama.

campo *POINTER* especifica el desplazamiento dentro de la opción de la siguiente ranura disponible.

Cada vez que una máquina maneja un datagrama que tiene activada la opción de registro de ruta, la máquina añade su dirección a la lista del registro de ruta (se debe colocar suficiente espacio en la opción desde la fuente original para manejar todas las entradas que pudieran ser necesarias). Para añadirse a sí misma en la lista, una máquina primero compara el puntero y el campo de longitud. Si el puntero es mayor que la longitud, la lista estará llena y la máquina continuará con el envío del datagrama sin incluirse. Si la lista no está llena, la máquina insertará su dirección IP de 4 octetos en la posición especificada por el *POINTER* e incrementará en 4 el valor de *POINTER*.

Cuando un datagrama llega a su destino, la máquina puede extraer y procesar la lista de direcciones IP. Por lo general, una computadora que recibe un datagrama ignora la ruta registrada. Para usar la opción de registro de ruta se requiere de dos máquinas que estén de acuerdo para cooperar; una computadora no recibirá rutas registradas de los datagramas entrantes ni activará la opción de registro de ruta en los datagramas de salida de manera automática. La fuente debe aceptar la habilitación de la opción de registro de ruta y el destino debe aceptar el procesamiento de la lista resultante.

### 7.8.2 Opción de ruta fuente

Otra idea que los creadores de redes encontraron interesante es la opción de la *source route* (*ruta de fuente*). La idea de fondo del ruteo de fuente es que proporciona para el emisor una forma en la que éste puede determinar una ruta a través de la red de redes. Por ejemplo, para probar el desempeño de una red física en particular *N*, el administrador de sistemas puede utilizar la ruta de fuente para forzar a los datagramas IP a viajar a través de la red *N*, incluso si los ruteadores normalmente seleccionan una ruta que no está incluida en esa trayectoria. La capacidad para realizar esta prueba es especialmente importante en un ambiente de producción, debido a que permite a los administradores de red tener la libertad de enviar los datagramas a través de la red en una forma que ellos conocen y saben que opera correctamente mientras prueban al mismo tiempo otras redes. Por supues-

to, este tipo de ruteo es útil sólo para personas que entienden la topología de red; el usuario promedio no necesita conocerlo o utilizarlo.

El IP soporta dos formas de ruteo de fuente. Una forma, conocida como *ruteo estricto de fuente*, especifica una vía de ruteo incluyendo una secuencia de direcciones IP en la opción como se muestra en la figura 7.12.

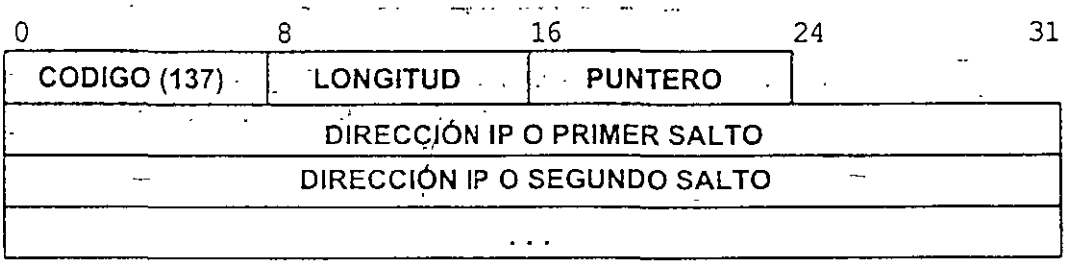



**Figura 7.12** La opción de ruta estricta de fuente especifica una ruta precisa estableciendo una lista de direcciones IP que el datagrama debe seguir.

El ruteo estricto de fuente significa que las direcciones especifican la ruta exacta que los datagramas deben seguir para llegar a su destino. La ruta entre dos direcciones sucesivas de la lista debe consistir en una sola red física; se producirá un error si el ruteador no puede seguir una ruta estricta de fuente. La otra forma, conocida como *loose source routing (ruteo no estricto de fuente)*, también incluye una secuencia de direcciones IP. Ésta especifica que el datagrama debe seguir la secuencia de direcciones IP, pero permite múltiples saltos de redes entre direcciones sucesivas de la lista.

Ambas opciones de ruta de fuente requieren que los ruteadores, a lo largo de la trayectoria, anoten su propia dirección de red local en la lista de direcciones. Así, cuando un datagrama llega a su destino, contiene una lista con todas las direcciones recorridas, igual que la lista producida por la opción de registro de ruta.

El formato de una opción de ruta de fuente recuerda al de la opción de registro de ruta, mostrada arriba. Cada ruteador examina los campos *POINTER* y *LENGTH* para ver si la lista está completa. Si es así, el campo puntero es mayor que la longitud y el ruteador establece la ruta del datagrama hacia su destino como lo hace normalmente. Si la lista no está completa, el ruteador sigue al puntero, toma la dirección IP, la reemplaza con la dirección del ruteador[7] y establece la ruta para el datagrama utilizando la dirección que obtuvo de la lista.

### 7.8.3 Opción de sello de hora

La *opción de sello de hora* trabaja como la opción de registro de ruta. La opción de sello de hora contiene una lista inicial vacía y cada ruteador, a lo largo de la ruta, desde la fuente hasta el destino, escribe sus datos en la lista. Cada entrada a la lista contiene 2 datos de 32 bits: la dirección IP

[7] Un ruteador tiene una dirección por cada interfaz; éste registra la dirección que corresponde a la red en la que se está definiendo una ruta para el datagrama.

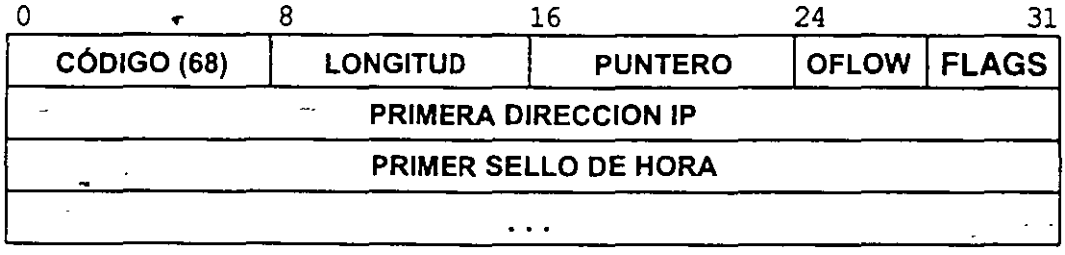



**Figura 7.13** Formato de una opción de sello de hora. Los bits en el campo FLAGS (BANDERAS) controlan el formato exacto y las reglas de ruteo que se utilizan para procesar esta opción.

del ruteador que proporciona la entrada y un entero de sello de hora de 32 bits. La figura 7.13 muestra el formato de la opción de sello de hora.

En la figura, los campos *LENGTH* y *POINTER* se utilizan para especificar la longitud del espacio reservado para la opción y la localización de la siguiente ranura no utilizada (exactamente como en la opción de registro de ruta). El campo de 4 bits *OFLOW* contiene un contador entero de ruteadores que podría no proporcionar un sello de hora si la opción fue demasiado pequeña.

El valor en el campo *FLAGS* de 4 bits controla el formato exacto de la opción y establece cómo los ruteadores deben suministrar el sello de hora. Los valores son:

| Valor de la bandera | Significado |
|---|---|
| 0 | Registro de sello de hora solamente; omite direcciones IP. |
| 1 | Anteponer a cada sello de hora una dirección IP (este es el formato que se muestra en la figura 7.13) |
| 3 | Las direcciones IP se especifican por el emisor; un ruteador sólo registra un sello de hora si la próxima dirección IP en la lista concuerda con la dirección IP del ruteador. |

**Figura 7.14** Interpretación de los valores en el campo FLAGS (BANDERAS) de la opción sello de hora

El sello de hora define la hora y la fecha en la que un ruteador manejó el datagrama, expresado en milisegundos desde la media noche Tiempo Universal.[8] Si la representación estándar para la hora no está disponible, el ruteador puede utilizar cualquier representación de tiempo local disponible activando el bit de orden superior en el campo de sello de hora. Por supuesto, el sello de hora para computadoras independientes no siempre será consistente si representan un tiempo universal. Cada máquina reportará una hora de acuerdo a su reloj local y los relojes pueden diferir. Así, el sello de hora deberá considerarse como una estimación, independientemente de la representación.

[8] El Tiempo Universal fue formalmente conocido como Hora del Meridiano de Greenwich; es la hora del día del primer meridiano.

Podría parecer extraño que la opción de sello de hora incluya un mecanismo para hacer que los ruteadores registren sus direcciones IP con sellos de hora dado que la opción de registro de ruta ya proporcionaba esta capacidad. Sin embargo, grabar las direcciones IP con sellos de hora elimina la ambigüedad. Tener un registro de ruta con sellos de hora es también útil pues permite que el receptor sepa con exactitud cuál fue la ruta seguida por el datagrama.

### 7.8.4 Fragmentación durante el procesamiento de las opciones

La idea en la que se apoya la implementación de bit *COPY* en el campo de opción *CODE* debe estar clara ahora. Cuando se fragmenta un datagrama, un ruteador reproduce algunas opciones IP en todos los fragmentos y, a la vez, coloca algunas otras sólo en parte de esos fragmentos. Por ejemplo, consideremos la opción utilizada para registrar la ruta del datagrama. Habíamos dicho que cada fragmento sería manejado como un datagrama independiente, la cual no garantiza que todos los fragmentos sigan la misma ruta hacia su destino. Si todos los fragmentos contienen la opción de registro de ruta, el destino podría recibir una lista diferente de rutas de cada fragmento. De esta manera, no podría producir una sola lista completa de las rutas al reensamblar los datagramas. Sin embargo, el estándar IP especifica que la opción de registro de ruta sólo se debe copiar dentro de uno de los fragmentos.

No todas las opciones IP se pueden restringir a un fragmento. Consideremos la opción de ruta de fuente, por ejemplo, que especifica de qué manera debe viajar un datagrama a través de la red de redes. La información de ruteo de fuente debe reproducirse en todos los encabezados de los fragmentos pues, de lo contrario, los fragmentos no seguirán la ruta especificada., Así pues, el campo de código para la ruta de fuente especifica que la opción se debe copiar en todos los fragmentos.

## 7.9 Resumen

El servicio fundamental proporcionado por el software TCP/IP de red de redes es un sistema de entrega de paquetes sin conexión, no confiable, y con el mejor esfuerzo. El Protocolo Internet (IP) especifica formalmente el formato de los paquetes en la red de redes, llamados *datagramas*, e informalmente le da cuerpo a la idea de entrega sin conexión. En este capítulo, nos concentramos en el formato de datagramas; en capítulos posteriores, analizaremos el ruteo IP y el manejo de errores.

De la misma forma que la trama física, el datagrama IP se divide en áreas de encabezado y áreas de datos. Además de información de otro tipo, el encabezado de datagrama contiene las direcciones de fuente y destino, control de fragmentación, prioridad y suma de verificación utilizada para identificar errores de transmisión. Además de los campos de longitud fija, cada encabezado de datagrama puede contener un campo de opciones. El campo de opciones tiene una longitud variable, dependiendo del número y tipo de opciones utilizadas así como del tamaño del área de datos para cada opción. Empleadas para ayudar a monitorear y controlar la red de redes, las opciones permiten especificar o registrar rutas o permiten reunir sellos de hora conforme viajan los datagramas por la red de redes.

## PARA CONOCER MÁS

Postel (1980) trata las formas posibles de acercamiento hacia los protocolos, el direccionamiento y el ruteo en las redes de redes. En publicaciones posteriores, Postel (RFC 791) plantea el estándar del Protocolo Internet. Braden (RFC 1122) refina aún más el estándar. Horning (RFC 894) especifica el estándar para la transmisión de datagramas IP a través de una red Ethernet. Clark (RFC 815) describe el reensamblado eficiente de fragmentos. Además del formato para los paquetes, las autoridades de Internet también especifican muchas constantes necesarias en los protocolos de red. Estos valores se pueden encontrar en Reynolds y Postel (RFC 1700). Kent y Mogul (1987) tratan las desventajas de la fragmentación.

Un conjunto alternativo de protocolos de red de redes, conocido como *XNS*, se plantea en Xerox (1981). Boggs *et. al.* (1980) describe el protocolo Paquete Universal PARC (PUP), una abstracción de *XNS* relacionada estrechamente con los datagramas IP.

## EJERCICIOS

**7.1** ¿Cuál es la mayor ventaja que hay en el hecho de que la suma de verificación de IP cubra sólo el encabezado del datagrama y no los datos? ¿Cuál es la desventaja?

**7.2** ¿Siempre es necesario utilizar una suma de verificación IP cuando se envían paquetes en una red Ethernet? ¿Por qué sí o por qué no?

**7.3** ¿Cuál es el tamaño MTU para ANSNET? ¿Para Hyperchannel? ¿Para un conmutador ATM?

**7.4** ¿Esperaría que una red de área local de alta velocidad tuviera un tamaño de MTU mayor o menor que una red de área amplia?

**7.5** Analice qué fragmentos deberían tener un encabezado pequeño no estándar.

**7.6** Determine cuándo se dio el último cambio de versión del protocolo IP. ¿Definir un número de versión de protocolo es realmente útil?

**7.7** ¿Puede usted imaginar por qué la suma de verificación por complemento a uno fue seleccionada para IP, en lugar de la verificación por redundancia cíclica?

**7.8** ¿Cuáles son las ventajas de reensamblar en el destino final, en lugar de hacerlo luego de que el datagrama ha atravesado una red?

**7.9** ¿Cuál es la MTU mínima requerida para enviar un datagrama IP que contenga cuando menos un octeto de datos?

**7.10** Supongamos que usted está interesado en implantación un procesamiento de datagramas IP en hardware. ¿Hay algún arreglo de campos del encabezado que pudiera hacer al hardware más eficiente? ¿Más fácil de construir?

**7.11** Si tiene acceso a una implantación de IP, revísela y pruebe sus implantaciones locales disponibles de IP para comprobar si rechazan datagramas IP con un número de versión obsoleta.

**7.12** Cuando un datagrama IP de tamaño mínimo viaja a través de una red Ethernet, ¿qué tan grande es la trama?

# 8

# Protocolo Internet: ruteo de datagramas IP

## 8.1 Introducción

Hemos visto que todos los servicios de red de redes utilizan un sistema sin conexión de entrega de paquetes y también que la unidad básica de transferencia en una red de redes TCP/IP es el datagrama IP. En este capítulo, se proporciona mayor información sobre el servicio sin conexión, pues se describe cómo los ruteadores direccionan datagramas IP y cómo los entregan en su destino final. Pensamos en el formato de datagramas descrito en el capítulo 7 como los aspectos estáticos del Protocolo Internet. La descripción del ruteo en este capítulo presenta los aspectos operacionales. En el siguiente capítulo, concluirá nuestra presentación del IP con una descripción de cómo se manejan los errores: en los capítulos subsecuentes se mostrará cómo otros protocolos utilizan el IP para proporcionar servicios de un nivel más alto.

## 8.2 Ruteo en una red de redes

En un sistema de conmutación de paquetes, el *ruteo* es el proceso de selección de un camino sobre el que se mandarán paquetes y el *ruteador* es la computadora que hace la selección. El ruteo ocurre a muchos niveles. Por ejemplo, dentro de una red de área amplia que tiene muchas conexiones físicas entre conmutadores de datos, la red por sí misma es responsable de rutear paquetes desde que llegan hasta que salen. Dicho ruteo interno está completamente contenido dentro de la red de área

amplia. Las máquinas en el exterior no pueden participar en las decisiones; sólo ven la red como una entidad que entrega paquetes.

Recuerde que el objetivo del IP es proporcionar una red virtual que comprenda muchas redes físicas, así como ofrecer un servicio sin conexión de entrega de paquetes. Por lo tanto, nos enfocaremos en el *ruteo en red de redes* o *ruteo IP*.[1] De forma análoga al ruteo dentro de una red física, el ruteo IP selecciona un camino por el que se debe enviar un datagrama. El algoritmo de ruteo IP debe escoger cómo enviar un datagrama pasando por muchas redes físicas.

El ruteo en una red de redes puede ser difícil, en especial entre computadoras que tienen muchas conexiones físicas de red. De forma ideal, el software de ruteo examinaría aspectos como la carga de la red, la longitud del datagrama o el tipo de servicio que se especifica en el encabezado del datagrama, para seleccionar el mejor camino. Sin embargo, la mayor parte del software de ruteo en red de redes es mucho menos sofisticado y selecciona rutas basándose en suposiciones sobre los caminos más cortos.

Para entender completamente el ruteo IP, debemos regresar y recordar la arquitectura de una red de redes TCP/IP. Primero, recuerde que una red de redes se compone de muchas redes físicas, interconectadas por computadoras conocidas como *ruteadores*. Cada ruteador tiene conexiones directas hacia dos o más redes. En contraste, por lo general un anfitrión se conecta directamente a una red física. Sin embargo, sabemos que es posible tener un anfitrión multi-homed conectado directamente a muchas redes.

Tanto los anfitriones como los ruteadores participan en el ruteo de datagramas IP que viajan a su destino. Cuando un programa de aplicación en un anfitrión intenta comunicarse, los protocolos TCP/IP eventualmente generan uno o más datagramas IP. El anfitrión debe tomar una decisión de ruteo cuando elige a dónde enviar los datagramas. Como se muestra en la figura 8.1, los anfitriones deben tomar decisiones de ruteo, inclusive si sólo tienen una conexión de red.

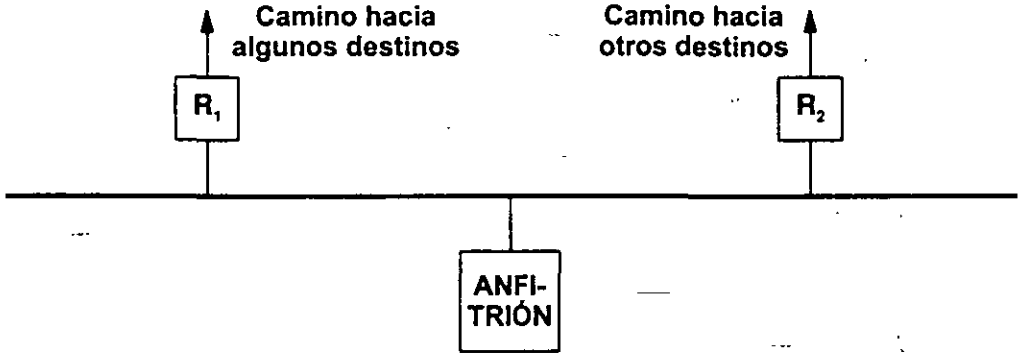



**Figura 8.1** Ejemplo de un anfitrión singly-homed que debe rutear datagramas. El anfitrión debe enviar un datagrama al ruteador $R_1$ o al ruteador $R_2$, ya que cada uno proporciona el mejor camino hacia algunos destinos.

[1] Los fabricantes también utilizan los términos *direccionamiento IP* y *conmutación IP para describir el ruteo IP*. Asimismo, es interesante que la mayoría todavía se refiere a la información requerida como *información de ruteo IP*.

Por supuesto, los ruteadores también toman decisiones de ruteo IP (ese es su principal propósito y la razón de llamarlos *ruteadores*). ¿Qué hay sobre los anfitriones multi-homed? Cualquier computadora con muchas conexiones de red puede actuar como ruteador y, como veremos, los anfitriones multi-homed que ejecutan el TCP/IP tienen todo el software necesario para el ruteo. Además, los sitios que no pueden adquirir ruteadores por separado a veces utilizan máquinas de tiempo compartido y propósito general como anfitriones y ruteadores (esta práctica por lo general se ve limitada a los sitios en universidades). Sin embargo, los estándares TCP/IP hacen una gran diferenciación entre las funciones de un anfitrión y las de un ruteador, además los sitios que intentan mezclar funciones de anfitrión con funciones de ruteador en una sola máquina, a veces, encuentran que sus anfitriones multi-homed llevan a cabo interacciones inesperadas. Por ahora, distinguiremos los anfitriones de los ruteadores y asumiremos que los primeros no realizan la función, exclusiva de los ruteadores, de transferir paquetes de una red a otra.

## 8.3 Entrega directa e indirecta

Hablando sin formalismos, podemos dividir el ruteo en dos partes: *entrega directa* y *entrega indirecta*. La entrega directa, que es la transmisión de un datagrama desde una máquina a través de una sola red física hasta otra, es la base de toda la comunicación en una red de redes. Dos máquinas solamente pueden llevar a cabo la entrega directa si ambas se conectan directamente al mismo sistema subyacente de transmisión física (por ejemplo, una sola Ethernet). La *entrega indirecta* ocurre cuando el destino no es una red conectada directamente, lo que obliga al transmisor a pasar el datagrama a un ruteador para su entrega.

### 8.3.1 Entrega de datagramas sobre una sola red

Sabemos que una máquina en una red física puede enviar una trama física directamente a otra máquina en la misma red. Para transferir un datagrama IP, el transmisor encapsula el datagrama dentro de una trama física, transforma la dirección IP en una dirección física y utiliza la red para entregar el datagrama. En el capítulo 5 se describieron dos mecanismos posibles para la definición de direcciones, incluyendo la utilización del protocolo ARP para la asignación dinámica de direcciones en redes de tipo Ethernet. En el capítulo 7 se analizó la encapsulación de datagramas. Por lo tanto, ya hemos visto todas las piezas necesarias para entender la entrega directa. En resumen:

> *La transmisión de un datagrama IP entre dos máquinas dentro de una sola red física no involucra ruteadores. El transmisor encapsula el datagrama dentro de una trama física, transforma la dirección IP de destino en una dirección física de hardware y envía la trama resultante directamente a su destino.*

¿Cómo sabe el transmisor si el destino reside en una red directamente conectada? La respuesta es la siguiente: sabemos que las direcciones IP se dividen en un prefijo específico de red y un sufijo específico de anfitrión. Para averiguar si un destino reside en una de las redes directamente conectadas, el transmisor extrae la porción de red de la dirección IP de destino y la compara con

la porción de red de su propia dirección IP. Si corresponden, significa que el datagrama se puede enviar de manera directa. Aquí vemos una de las ventajas del esquema de direccionamiento de Internet, a saber:

> *Debido a que las direcciones de red de redes de todas las máquinas dentro de una sola red incluyen un prefijo en común y como la extracción de dicho prefijo se puede realizar mediante unas cuantas instrucciones de máquina, la comprobación de que una máquina se puede alcanzar directamente es muy eficiente.*

Desde la perspectiva de una red de redes, la forma más fácil de pensar en la entrega directa es como el paso final de cualquier transmisión de datagramas, aun si el datagrama atraviesa muchas redes y ruteadores intermedios. El último ruteador del camino entre la fuente del datagrama y su destino siempre se conectará directamente a la misma red física que la máquina de destino. Por lo tanto, el último ruteador entregará el datagrama utilizando la entrega directa. Podemos pensar en la entrega directa entre la fuente y el destino como un caso especial de ruteo de propósito general — en una ruta directa, el datagrama nunca pasa a través de ningún ruteador intermedio.

### 8.3.2 Entrega indirecta

La entrega indirecta es más difícil que la directa porque el transmisor debe identificar un ruteador para enviar el datagrama. Luego, el ruteador debe encaminar el datagrama hacia la red de destino.

Para visualizar cómo trabaja el ruteo indirecto, imagínese una gran red con muchas redes interconectadas por medio de ruteadores, pero sólo con dos anfitriones en sus extremos más distantes. Cuando un anfitrión quiere enviar un datagrama a otro, lo encapsula y lo envía hacia el ruteador más cercano. Sabemos que se puede alcanzar un ruteador debido a que todas las redes físicas están interconectadas, así que debe existir un ruteador conectado a cada una. Por lo tanto, el anfitrión de origen puede alcanzar un ruteador utilizando una sola red física. Una vez que la trama llega al ruteador, el software extrae el datagrama encapsulado, y el software IP selecciona el siguiente ruteador a lo largo del camino hacia el destino. De nuevo, se coloca el datagrama en una trama y se envía a través de la siguiente red física hacia un segundo ruteador, y así sucesivamente, hasta que se pueda entregar de forma directa. Estas ideas pueden resumirse así:

> *Los ruteadores en una red de redes TCP/IP forman una estructura cooperativa e interconectada. Los datagramas pasan de un ruteador a otro hasta que llegan a uno que los pueda entregar en forma directa.*

¿Cómo sabe un ruteador a dónde enviar cada datagrama? ¿Cómo puede saber un anfitrión qué ruteador utilizar para llegar a un destino determinado? Las dos preguntas están relacionadas, ya que comprenden el ruteo IP. Las contestaremos en dos fases, considerando en este capítulo el algoritmo básico de ruteo controlado por tablas y posponiendo el análisis sobre cómo los ruteadores aprenden sus rutas.

## 8.4 Ruteo IP controlado por tabla

El algoritmo usual de ruteo IP emplea una *tabla de ruteo Internet* (a veces, conocida como *tabla de ruteo IP*) en cada máquina que almacena información sobre posibles destinos y sobre cómo alcanzarlos. Debido a que tanto los ruteadores como los anfitriones rutean datagramas, ambos tienen tablas de ruteo IP. Siempre que el software de ruteo IP en un anfitrión necesita transmitir un datagrama, consulta la tabla de ruteo para decidir a dónde enviarlo.

¿Qué información se debe guardar en las tablas de ruteo? Si cada tabla de ruteo contuviera información sobre cada posible dirección de destino, sería imposible mantener actualizadas las tablas. Además, como el número de destinos posibles es muy grande, las máquinas no tendrían suficiente espacio para almacenar la información.

De manera conceptual, nos gustaría utilizar el principio de ocultación de información y permitir a las máquinas tomar decisiones de ruteo con una información mínima. Por ejemplo, nos gustaría aislar la información sobre anfitriones específicos del ambiente local en el que existen y hacer que las máquinas que están lejos ruteen paquetes hacia ellos sin saber dichos detalles. Por fortuna, el esquema de direccionamiento IP nos ayuda a lograr este objetivo. Recuerde que las direcciones IP se asignan de tal manera que todas las máquinas conectadas a una red física compartan un prefijo en común (la porción de red de la dirección). Ya hemos visto que una asignación de este tipo hace que la comprobación para la entrega directa sea eficiente. También significa que las tablas de ruteo sólo necesitan contener prefijos de red y no direcciones IP completas.

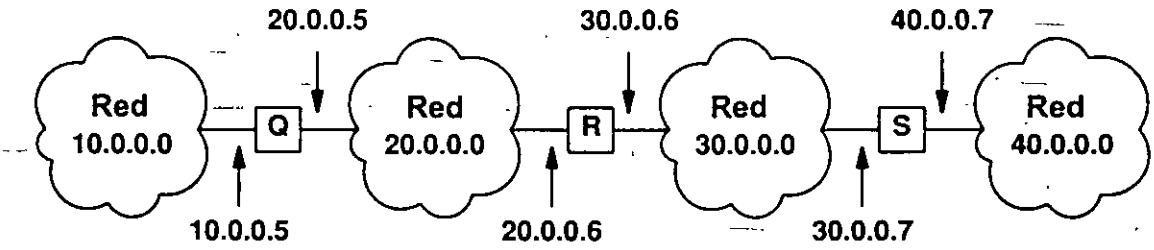
## 8.5 Ruteo con salto al siguiente

Utilizar la porción de red de una dirección de destino en vez de toda la dirección de anfitrión hace que el ruteo sea eficiente y mantiene reducidas las tablas de ruteo. También es importante, porque ayuda a ocultar información al mantener los detalles de los anfitriones específicos confinados al ambiente local en el que operan. Por lo común, una tabla de ruteo contiene pares $(N, R)$, donde $N$ es la dirección IP de una *red* de destino y $R$ la dirección IP del "siguiente" ruteador en el camino hacia la red $N$. El ruteador $R$ es conocido como el *salto siguiente* y la idea de utilizar una tabla de ruteo para almacenar un salto siguiente para cada destino es conocida como *ruteo con salto al siguiente*. Por lo tanto, la tabla de ruteo en el ruteador $R$ sólo especifica un paso a lo largo del camino de $R$ a su red de destino —el ruteador no conoce el camino completo hacia el destino.

Es importante entender que cada registro en una tabla de ruteo apunta hacia un ruteador que se puede alcanzar a través de una sola red. Esto es, que todos los ruteadores listados en la tabla de ruteo de la máquina $M$ deben residir en las redes con las que $M$ se conecta de manera directa. Cuando un datagrama está listo para dejar $M$, el software IP localiza la dirección IP de destino y extrae la porción de red. Luego, $M$ utiliza la porción de red para tomar una decisión de ruteo, seleccionando un ruteador que se pueda alcanzar directamente.

En la práctica, también aplicamos el principio de ocultación de información a los anfitriones. Insistimos que, aunque los anfitriones tengan tablas de ruteo IP, deben guardar información mínima en ellas. La idea es obligar a los anfitriones a que deleguen la mayor parte de sus funciones de ruteo a los ruteadores.

En la figura 8.2 se muestra un ejemplo concreto que nos ayuda a explicar las tablas de ruteo. La red de redes ejemplificada consiste en cuatro redes conectadas por tres ruteadores. En la figura, la tabla de ruteo proporciona las rutas que utiliza el ruteador *R*. Ya que *R* se conecta de manera directa a las redes 20.0.0.0 y 30.0.0.0, puede utilizar la entrega directa para llevar a cabo un envío a un anfitrión en cualquiera de esas redes (posiblemente utilizando ARP para encontrar las direcciones físicas). Teniendo un datagrama destinado para un anfitrión en la red 40.0.0.0, *R* lo rutea a la dirección 30.0.0.7, que es la dirección del ruteador *S*. Luego, *S* entregará el datagrama en forma directa. *R* puede alcanzar la dirección 30.0.0.7 debido a que tanto *R* como *S* se conectan directamente con la red 30.0.0.0.




(a)

| PARA ALCANZAR LOS ANFITRIONES EN LA RED | RUTEAR A ESTA DIRECCIÓN |
|---|---|
| 20.0.0.0 | ENTREGAR DIRECTAMENTE |
| 30.0.0.0 | ENTREGAR DIRECTAMENTE |
| 10.0.0.0 | 20.0.0.5 |
| 40.0.0.0 | 30.0.0.7 |

(b)

**Figura 8.2** (a) Ejemplo de una red con 4 redes y 3 ruteadores, y (b) la tabla de ruteo en *R*.

Como se demuestra en la figura 8.2, el tamaño de la tabla de ruteo depende del número de redes en la red; solamente crece cuando se agregan nuevas redes. Sin embargo, el tamaño y contenido de la tabla son independientes del número de anfitriones individuales conectados a las redes. Podemos resumir el principio subyacente:

*Para ocultar información, mantener reducidas las tablas de ruteo y tomar las decisiones de ruteo de manera eficiente, el software de ruteo IP sólo puede guar-*

*dar información sobre las direcciones de las redes de destino, no sobre las direcciones de anfitriones individuales.*

Escoger rutas basándose tan sólo en la identificación de la red de destino tiene muchas consecuencias. Primero, en la mayor parte de las implantaciones, significa que todo el tráfico destinado a una cierta red toma el mismo camino. Como resultado, aun cuando existen muchos caminos, quizá no se utilicen constantemente. De igual manera, todos los tipos de tráfico siguen el mismo camino sin importar el retraso o la generación de salida de las redes físicas. Segundo, debido a que sólo el último ruteador del camino intenta comunicarse con el anfitrión final, solamente el ruteador puede determinar si el anfitrión existe o está en operación. Por lo tanto, necesitamos encontrar una forma para que el ruteador envíe reportes sobre problemas de entrega, de vuelta a la fuente original. Tercero, debido a que cada ruteador rutea el tráfico de forma independiente, los datagramas que viajan del anfitrión *A* al *B* pueden seguir un camino totalmente distinto al que siguen los datagramas que viajan del anfitrión *B* al *A*. Necesitamos asegurarnos de que los ruteadores cooperen para garantizar que siempre sea posible la comunicación bidireccional.

## 8.6 Rutas asignadas por omisión

Otra técnica utilizada para ocultar información y mantener reducido el tamaño de las tablas de ruteo, es asociar muchos registros a un ruteador asignado por omisión. La idea es hacer que el software de ruteo IP busque primero la tabla de ruteo para encontrar la red de destino. Si no aparece una ruta en la tabla, las rutinas de ruteo envían el datagrama a un *ruteador asignado por omisión.*

El ruteo asignado por omisión es de gran ayuda cuando un sitio tiene pocas direcciones locales y sólo una conexión con el resto de la red de redes. Por ejemplo, las rutas asignadas por omisión trabajan bien en máquinas anfitriones que se conectan a una sola red física y alcanzan sólo un ruteador, que es la puerta hacia el resto de la red de redes. Toda la decisión de ruteo consiste en dos comprobaciones: una de la red local, y un valor asignado por omisión que apunta hacia el único ruteador posible. Inclusive si el sitio sólo contiene unas cuantas redes locales, el ruteo es sencillo ya que consiste en pocas comprobaciones de las redes locales, más un valor asignado por omisión para todos los demás destinos.

## 8.7 Rutas por anfitrión específico

Aunque hemos dicho que todo el ruteo está basado en redes y no en anfitriones individuales, la mayor parte del software de ruteo IP permite que se especifiquen rutas por anfitrión como caso especial. Tener rutas por anfitrión le da al administrador de red local un mayor control sobre el uso de la red, le permite hacer comprobaciones y también se puede utilizar para controlar el acceso por razones de seguridad. Cuando se depuran conexiones de red o tablas de ruteo, la capacidad para especificar una ruta especial hacia una máquina individual resulta ser especialmente útil.

## 8.8 El algoritmo de ruteo IP

Tomando en cuenta todo lo que hemos dicho, el algoritmo de ruteo IP es como sigue:

**Algoritmo:**

**RutaDatagrama (Datagrama, Tabla de Ruteo)**

**Extraer la dirección IP de destino, D, del datagrama y computar el prefijo de red, N;**
**si N corresponde a cualquier dirección de red directamente conectada entregar el datagrama al destino D sobre dicha red. (Esto comprende la transformación de D en una dirección física, encapsulando el datagrama y enviando la trama.)**
**De otra forma, si la tabla contiene una ruta con anfitrión específico para D, enviar el datagrama al salto siguiente especificado en la tabla;**
**de otra forma, si la tabla contiene una ruta para la red N, enviar el datagrama al salto siguiente especificado en la tabla;**
**de otra forma, si la tabla contiene una ruta asignada por omisión, enviar el datagrama al ruteador asignado por omisión especificado en la tabla;**
**de otra forma, declarar un error de ruteo;**

**Figura 8.3** Algoritmo que utiliza IP para direccionar un datagrama. Por medio de un datagrama IP y una tabla de ruteo, este algoritmo selecciona el salto siguiente al que se debe enviar el datagrama. Todas las rutas deben especificar un salto siguiente que resida en una red conectada directamente.

## 8.9 Ruteo con direcciones IP

Es importante entender que, a excepción de la disminución del tiempo de vida y de volver a computar la suma de verificación, el ruteo IP no altera el datagrama original. En particular, las direcciones de origen y destino del datagrama permanecen sin alteración; éstas siempre especifican la dirección IP de la fuente original y la dirección IP del último destino.[2] Cuando el IP ejecuta el algoritmo de ruteo, selecciona una nueva dirección IP, que es la dirección IP de la máquina a la que a continuación se tendrá que enviar el datagrama. La nueva dirección es parecida a la dirección de

[2] La única excepción ocurre cuando el datagrama contiene una opción de ruta de origen.

un ruteador. Sin embargo, si el datagrama se puede entregar directamente, la nueva dirección será la misma que la del último destino.

Dijimos que la dirección IP seleccionada por el algoritmo de ruteo IP se conoce como la dirección de *salto al siguiente*, pues indica a dónde se tiene que enviar después el datagrama (aunque quizá no sea el último destino). ¿Dónde almacena el IP la dirección del salto siguiente? No en el datagrama; no existe un lugar reservado para ella. De hecho, el IP no "almacena" la dirección del salto siguiente. Después de ejecutar el algoritmo de ruteo, el IP pasa el datagrama y la dirección del salto siguiente al software de interfaz de red, responsable de la red física sobre la que el datagrama se debe enviar. El software de interfaz de red transforma la dirección de salto siguiente en una dirección física, crea una trama utilizando la dirección física, pone el datagrama en la porción de datos de la trama y envía el resultado. Después de utilizar la dirección de salto siguiente para encontrar una dirección física, el software de interfaz de red la descarta.

Puede parecer extraño que las tablas de ruteo almacenen la dirección IP del salto siguiente para cada red de destino cuando dichas direcciones se tienen que traducir a sus direcciones físicas correspondientes, antes de que se pueda enviar el datagrama. Si nos imaginamos un anfitrión que envía una secuencia de datagramas a la misma dirección de destino, la utilización de direcciones IP nos parecería increíblemente ineficiente. El IP obedientemente extrae la dirección de destino en cada datagrama y utiliza la tabla de ruteo para producir una nueva dirección de salto siguiente. Luego pasa el datagrama y la dirección de salto siguiente a la interfaz de red, que recomputa la asignación para obtener una dirección física. Si la tabla de ruteo utilizó direcciones físicas, la transformación entre la dirección IP de salto siguiente y la dirección física se pueden llevar a cabo sólo una vez, evitando así cómputos innecesarios.

¿Por qué el software IP evita la utilización de direcciones físicas cuando almacena y computa las rutas? Como se muestra en la figura 8.4, existen dos razones importantes.

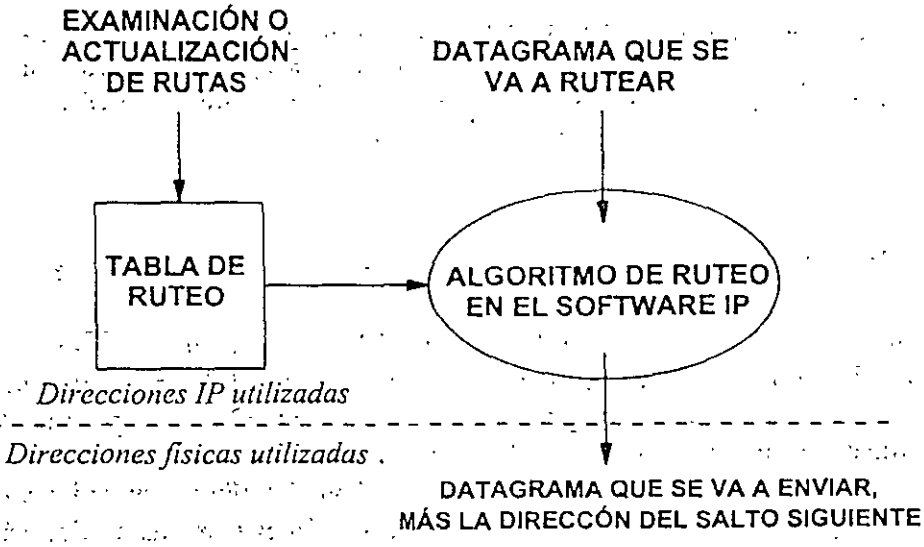



**Figura 8.4** El software IP y la tabla de ruteo que utiliza, residen arriba de la frontera de dirección. Utilizar sólo direcciones IP facilita la examinación o cambios de las rutas y oculta los detalles de las direcciones físicas.

Primero, la tabla de ruteo proporciona una interfaz muy transparente entre el software IP que rutea datagramas y el software de alto nivel que manipula las rutas. Para depurar problemas de ruteo, los administradores de red a menudo necesitan examinar las tablas de ruteo. La utilización de direcciones IP solamente en la tabla de ruteo facilita que los administradores las entiendan, lo mismo que ver dónde el software actualizó correctamente las rutas. Segundo, todo el sentido del Protocolo Internet es construir una abstracción que oculte los detalles de las redes subyacentes.

En la figura 8.4 se muestra la *frontera de direcciones*, importante división conceptual entre el software de bajo nivel que entiende las direcciones físicas y el software interno que sólo utiliza direcciones de alto nivel. Arriba de esta frontera, se puede escribir todo el software para que se comunique utilizando direcciones de red de redes; el conocimiento de las direcciones físicas se relega a unas cuantas rutinas de bajo nivel. Veremos que, al respetar la frontera, también se facilita la comprensión, prueba y modificación de la implantación de los restantes protocolos TCP/IP.

## 8.10 Manejo de los datagramas entrantes

Hasta ahora, hemos analizado el ruteo IP al describir cómo se toman las decisiones sobre los paquetes salientes. Sin embargo, debe quedar claro que el software IP también tiene que procesar los datagramas entrantes.

Cuando un datagrama IP llega a un anfitrión, el software de interfaz de red lo entrega al software IP para su procesamiento. Si la dirección de destino del datagrama corresponde a la dirección IP del anfitrión, el software IP del anfitrión acepta el datagrama y lo pasa al software de protocolo de alto nivel apropiado, para su procesamiento posterior. Si la dirección IP de destino no corresponde, se requiere que el anfitrión descarte el datagrama (por ejemplo, está prohibido que los anfitriones intenten direccionar datagramas que accidentalmente se ruteraron a la máquina equivocada).

A diferencia de los anfitriones, los ruteadores sí realizan el direccionamiento. Cuando llega un datagrama IP a un ruteador, éste lo entrega al software IP. De nuevo, surgen dos casos: que el datagrama haya podido llegar a su destino final o que, quizá, necesite viajar más. Como con los anfitriones, si la dirección de destino del datagrama corresponde a la dirección IP, el software IP pasa el datagrama a un software de protocolo de nivel más alto para su procesamiento.[3] Si el datagrama no ha llegado a su destino final; el IP lo rutea utilizando el algoritmo estándar así como la información en la tabla local de ruteo.

La determinación sobre si un datagrama IP alcanzó su destino final no es tan trivial como parece. Recuerde que hasta un anfitrión puede tener muchas conexiones físicas, cada una con su propia dirección IP. Cuando llega un datagrama IP, la máquina debe comparar la dirección de destino de red de redes con la dirección IP de cada una de sus conexiones de red. Si alguna corresponde, guarda el datagrama y lo procesa. Una máquina también debe aceptar datagramas que se transmitieron por difusión en la red física, si su dirección IP de destino es la dirección IP de difusión limitada, o es la dirección IP de difusión dirigida para esa red. Como veremos en los capítulos 10 y 17, las direcciones de subred y de multidifusión hacen que el reconocimiento de direcciones sea aún más complejo. En cualquier caso, si la dirección no corresponde a ninguna de las direcciones de la máquina local, el IP disminuye el campo de tiempo de vida en el encabezado del datagrama, des-

[3] Por lo general, los únicos datagramas destinados para un ruteador, son los utilizados para probar la conectividad o los que llevan comandos de manejo del ruteador.

cartándolo si el contador llega a cero, o computa una nueva suma de verificación y rutea el datagrama si la cuenta es positiva.

¿Todas las máquinas deben direccionar los datagramas IP que reciben? Obviamente, un ruteador debe direccionar datagramas entrantes ya que esa es su función principal. También hemos dicho que algunos anfitriones multi-homed actúan como ruteadores, aunque realmente son sistemas de computación multi-propósito. Aunque utilizar un anfitrión como ruteador por lo general no es una buena idea, si se elige utilizarlos de esa manera, el anfitrión debe configurarse para rutear datagramas igual que lo hace un ruteador. ¿Pero qué hay de los otros anfitriones, los que no están diseñados para ser ruteadores? La respuesta es que los anfitriones que no están diseñados para ello *no* deben rutear los datagramas que reciban, sino descartarlos.

Existen cuatro razones por las que un anfitrión que no esté diseñado para trabajar como ruteador debe abstenerse de realizar cualquier función de ruteo. Primero, cuando un anfitrión, de los antes mencionados, recibe un datagrama diseñado para alguna otra máquina, es que algo salió mal con el direccionamiento, ruteo o entrega en la red de redes. El problema puede no verse si el anfitrión toma una acción correctiva al rutear el datagrama. Segundo, el ruteo causará tráfico innecesario de red (y puede quitarle tiempo a la CPU para utilizar de forma legítima el anfitrión). Tercero, los errores simples pueden causar un caos. Suponga que cada anfitrión rutea tráfico e imagine lo que pasa si una máquina accidentalmente transmite por difusión un datagrama que está destinado al anfitrión *H*. Debido a que se llevó a cabo una difusión, cada anfitrión dentro de la red recibe una copia del datagrama. Cada anfitrión direcciona su copia hacia *H*, que se verá bombardeado con muchas copias. Cuarto, como se muestra en los siguientes capítulos, los ruteadores hacen mucho más que sólo rutear el tráfico. Como se mostrará en el siguiente capítulo, los ruteadores utilizan un protocolo especial para reportar errores y los anfitriones no (de nuevo, para evitar que muchos reortes de error saturen una fuente). Los ruteadores también propagan información de ruteo para asegurarse de que sus tablas están actualizadas. Si los anfitriones rutean datagramas sin participar por completo en todas las funciones de ruteo, se pueden presentar anormalidades inesperadas.

## 8.11 Establecimiento de tablas de ruteo

Hemos analizado cómo el IP rutea datagramas basándose en el contenido de las tablas de ruteo, sin indicar de qué manera inician o actualizan los sistemas sus tablas conforme cambia la red. En los capítulos posteriores, se tratarán estos temas y se analizarán los protocolos que permiten que los ruteadores mantengan sus tablas actualizadas. Por ahora, sólo es importante entender que el software IP utiliza la tabla de ruteo siempre que decide direccionar un datagrama, así que cambiar las tablas de ruteo cambiaría los caminos que siguen los datagramas.

## 8.12 Resumen

El ruteo IP consiste en decidir a dónde enviar un datagrama basándose en su dirección IP de destino. La entrega directa es posible si la máquina de destino reside en una red a la que se conecta la máquina transmisora; pensamos que ese es el paso final en la transmisión de datagramas. Si el

transmisor no puede alcanzar directamente al destino, debe direccionar el datagrama hacia un ruteador. El paradigma general es que todos los anfitriones envían datagramas de manera indirecta al ruteador más cercano; los datagramas viajan a través de la red de redes de un ruteador a otro hasta que pueden ser entregados de manera directa sobre una red física.

Cuando el software IP busca una ruta, el algoritmo genera la dirección IP de la siguiente máquina (por ejemplo, la dirección del salto siguiente) a la que se debe enviar el datagrama; el IP pasa el datagrama y la dirección del salto siguiente al software de interfaz de red. La transmisión de un datagrama de una máquina a la siguiente siempre comprende la encapsulación del datagrama en una trama física, transformando la dirección del salto siguiente en una dirección física y enviando la trama al utilizar el hardware subyacente.

El algoritmo de ruteo en una red de redes utiliza sólo direcciones IP y se controla por medio de tablas. Aunque es posible que una tabla de ruteo contenga una dirección de destino de un anfitrión específico, la mayor parte de ellas solamente contienen direcciones de red para mantenerse de un tamaño reducido. La utilización de una ruta asignada por omisión también puede ser útil para mantener reducida una tabla de ruteo, especialmente para los anfitriones que pueden accesar sólo un ruteador.

## PARA CONOCER MÁS

El ruteo es un tema importante. Frank y Chou (1971) y Schwartz y Stern (1980) tratan el ruteo en forma general; Postel (1980) analiza el ruteo en una red de redes. Braden y Postel (RFC 1009) proporcionan un resumen de cómo los ruteadores de Internet manejan los datagramas IP. Almquist (RFC 1716) ofrece un resumen sobre estudios más recientes. Narten (1989) contiene una encuesta sobre el ruteo en Internet. Fultz y Kleinrock (1971) analizan esquemas adaptables de ruteo; y McQuillan, Richer, y Rosen (1980) describen el algoritmo adaptable de ruteo ARPANET.

La idea de utilizar afirmaciones sobre políticas para formular reglas sobre el ruteo se considera a menudo. Leiner (RFC 1124) considera las políticas para redes interconectadas. Braun (RFC 1104) analiza modelos de políticas de ruteo para redes de redes; Rekhter (RFC 1092) relaciona las políticas de ruteo con la segunda columna vertebral NSFNET, y Clark (RFC 1102) describe la utilización de políticas de ruteo con el IP.

## EJERCICIOS

**8.1** Complete las tablas de ruteo para todos los ruteadores en la figura 8.2. ¿Qué ruteadores se beneficiarían más utilizando una ruta asignada por omisión?

**8.2** Examine el algoritmo de ruteo utilizado en su sistema local. ¿Están cubiertos todos los casos mencionados aquí? ¿Permite el algoritmo cualquier acción no mencionada?

**8.3** ¿Qué es lo que hace un ruteador con el valor de *tiempo de vida* en un encabezado IP?

**8.4** Considere una máquina con dos conexiones a redes físicas y con dos direcciones IP, $I_1$ e $I_2$. ¿Es posible que esa máquina reciba un datagrama destinado para $I_2$ sobre la red con la dirección $I_1$? Explíquelo.

8.5 Considere que dos anfitriones, *A* y *B*, se conectan a una misma red física, *N*. ¿Es posible que, al utilizar nuestro algoritmo de ruteo, *A* reciba un datagrama destinado para *B*? Explíquelo.

8.6 Modifique el algoritmo de ruteo para incorporar las opciones de ruteo de fuente IP que se trataron en el capítulo 7.

8.7 Un ruteador IP debe realizar un cómputo que toma tiempo, proporcional a la longitud del encabezado del datagrama, cada vez que procesa un datagrama. Explíquelo.

8.8 Un administrador de red arguye que, para facilitar el monitoreo y la depuración de su red local, quiere reescribir el algoritmo de ruteo para que compruebe las rutas de anfitrión específico *antes* de comprobar la entrega directa. ¿Se puede imaginar cómo podría utilizar el algoritmo revisado para diseñar un monitor de red?

8.9 ¿Es posible direccionar un datagrama a una dirección IP de un ruteador? ¿Tiene algún sentido hacerlo?

8.10 Considere un algoritmo modificado de ruteo que examine las rutas de anfitrión específico antes de comprobar la entrega en redes conectadas directamente. ¿Bajo qué circunstancias se desearía un algoritmo así? ¿Bajo qué circunstancias no?

8.11 Juegue al detective: una tarde, después de monitorear el tráfico IP en una red de área local por 10 minutos, alguien se da cuenta de que todas las tramas destinadas para la máquina *A* llevan datagramas IP que tienen un destino igual a la dirección IP de *A*, mientras que todas las tramas destinadas para la máquina *B* llevan datagramas IP que tienen un destino igual a la dirección IP de *B*. Los usuarios informan que tanto *A* como *B* se pueden comunicar. Explíquelo.

8.12 ¿Cómo podría cambiar el formato de datagrama IP de manera que pudiera aceptar la conmutación de datos en alta velocidad en los ruteadores? Pista: un ruteador debe recomputar la suma de verificación del encabezado después de disminuir el campo de tiempo de vida.

8.13 Compare el CLNP, protocolo ISO de entrega sin conexión (estándar ISO 8473) con el IP. ¿Qué tan bien aceptaría el protocolo ISO la conmutación a alta velocidad? Pista: los campos de longitud variable son caros.

# 10

# Extensiones de dirección de subred y superred

## 10.1 Introducción

En el capítulo 4, se analizó el esquema original de direccionamiento en Internet y se presentó los tres formatos principales de las direcciones IP. En este capítulo, se examinan cuatro extensiones del esquema de direcciones IP, que permiten que una localidad utilice una sola dirección IP para muchas redes físicas. En él, se considera la motivación para las extensiones de dirección y se describen los mecanismos básicos para cada una. En particular, en este capítulo se presentan los detalles del esquema de subred que actualmente es parte del estándar TCP/IP.

## 10.2 Reseña de hechos importantes

En el capítulo 4 se trató el direccionamiento en las redes de redes y se presentó los fundamentos del esquema actual de las direcciones IP. Se dijo que las direcciones de 32 bits se asignan con cuidado para que las direcciones IP de todos los anfitriones de una red física tengan un prefijo en común. En el esquema original de las direcciones IP, los diseñadores pensaron al prefijo como la definición de la porción de red de una dirección de red de redes, y al remanente como la porción de anfitrión. La consecuencia que nos interesa es que:

*En el esquema original de direccionamiento IP, cada red física tiene asignada una dirección única; cada anfitrión en la red tiene la dirección de red como prefijo de su dirección individual.*

La mayor ventaja de dividir una dirección IP en dos partes surge del tamaño de las tablas de ruteo que necesitan los ruteadores. En vez de almacenar un registro de ruteo por cada anfitrión de destino, un ruteador puede tener un registro por cada red y examinar sólo la porción de red de la dirección de destino cuando tome decisiones de ruteo.

Recuerde que el TCP/IP incorpora muchos tamaños de red por el hecho de tener tres tipos principales de direcciones. Las redes que tienen asignadas direcciones tipo *A* dividen los 32 bits en una porción de red de 8 bits y una porción de anfitrión de 24 bits. Las direcciones tipo *B* dividen los 32 bits en porciones de red y de anfitrión de 16 bits; y las direcciones tipo *C* dividen la dirección en una porción de red de 24 bits y una porción de anfitrión de 8 bits.

Para entender las extensiones de dirección de este capítulo, es importante darse cuenta que las localidades tienen la libertad de modificar las direcciones y las rutas, siempre y cuando dichas modificaciones permanezcan ocultas para las demás localidades. Esto es, una localidad puede asignar y utilizar internamente direcciones IP de manera no usual siempre y cuando:

- Todos los anfitriones y los ruteadores en dicha localidad estén de acuerdo en seguir el esquema de direccionamiento.
- Otras localidades en Internet puedan manejar las direcciones como en el esquema original.

## 10.3 Minimización de números de red

El esquema original de direccionamiento IP parece incluir todas las posibilidades, pero tiene una debilidad menor. ¿Cómo surgió esta debilidad? ¿Qué es lo que los diseñadores no vislumbraron? La respuesta es simple: el crecimiento. Debido a que los diseñadores trabajaban en un mundo de computadoras mainframe caras, visualizaron una red con cientos de redes y miles de anfitriones. No pensaron en las decenas de miles de redes pequeñas de computadoras personales que aparecerían de manera repentina en los años siguientes al diseño del TCP/IP.

El crecimiento es más visible en cuanto a las conexiones a Internet, cuyo tamaño se duplica cada nueve meses. La gran población de redes pequeñas resalta la importancia del esquema de Internet, ya que significa: (1) que se requiere mucho trabajo administrativo para manejar las direcciones de red, (2) que las tablas de ruteo de los ruteadores son muy grandes, y (3) que el espacio para las direcciones se acabará eventualmente. El segundo problema es importante porque significa que, cuando los ruteadores intercambian información de sus tablas de ruteo, la carga en la red de redes es alta, así como también lo es el esfuerzo computacional requerido por los ruteadores participantes. El tercer problema es crucial ya que el esquema original de direcciones no puede incorporar el número actual de redes en la red global Internet. En particular, no existen suficientes prefijos tipo B para cubrir todas las redes de tamaño mediano en Internet. La pregunta es: ¿cómo se puede minimizar el número de direcciones asignadas de red, en especial las de tipo B, sin destruir el esquema original de direccionamiento?

Para minimizar las direcciones de red, muchas redes físicas deben compartir el mismo prefijo IP de red. Para minimizar las direcciones tipo B, se deben utilizar direcciones tipo C. Claro está, se deben modificar los procedimientos de ruteo y todas las máquinas que se conectan a las redes afectadas deben entender las normas utilizadas.

La idea de compartir una dirección de red entre muchas redes físicas no es nueva y ha tomado muchas formas. Examinaremos tres de ellas: ruteadores transparentes, ARP sustituto (proxy ARP) y subredes IP estándar. También consideraremos el direccionamiento sin tipo, que es asignar muchas direcciones tipo C en vez de direcciones tipo B.

## 10.4 Ruteadores transparentes

El esquema de *ruteador transparente* se basa en la observación de que una red que tiene asignada una dirección IP tipo *A* se puede extender mediante un sencillo truco, ilustrado en la figura 10.1.

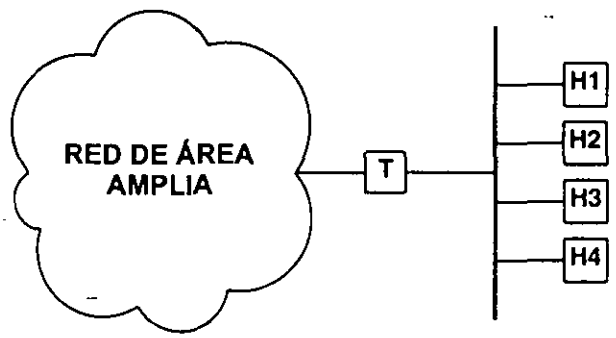



**Figura 10.1** Ruteador transparente *T* que extiende una red de área amplia a muchos anfitriones en una localidad. Cada anfitrión parece tener una dirección IP en la WAN.

El truco consiste en hacer que una red física, por lo general una WAN, realice el multiplexado de muchas conexiones de anfitrión a través de un solo puerto. Como se muestra en la figura 10.1, un ruteador *T*, de propósito especial, conecta un solo puerto de anfitrión de la red de área amplia a una red de área local. *T* se conoce como *ruteador transparente,* debido a que los otros anfitriones y ruteadores en la WAN no saben que existe.

La red de área local no posee su propio prefijo IP; los anfitriones conectados tienen asignadas direcciones como si se conectaran de manera directa con la WAN. El ruteador transparente realiza el demultiplexado de los datagramas que llegan de la WAN al enviarlos hacia el anfitrión apropiado (por ejemplo, utilizando una tabla de direcciones). El ruteador transparente también acepta datagramas de los anfitriones en la red de área local y los rutea a través de la WAN hacia su destino.

Para realizar de manera eficiente el demultiplexado, los ruteadores transparentes a menudo dividen la dirección IP en muchas partes y codifican la información dentro de las partes no utiliza-

das. Por ejemplo, ARPANET tenía asignada la dirección de red tipo *A 10.0.0.0.* Cada nodo de conmutación de paquetes (PSN) tenía una dirección única de números enteros. Internamente, ARPANET trataba cualquier dirección IP de 4 octetos con la forma *10.p.u.i,* como cuatro octetos separados que especificaban una red (*10*), un puerto específico en el PSN de destino (*p*), y un PSN de destino (*i*). El octeto *u* no tenía interpretación. Por consiguiente, tanto la dirección de ARPANET *10.2.5.37* como la *10.2.9.37*, se refieren al anfitrión *2* en el PSN *37.* Un ruteador transparente conectado al PSN *37* en el puerto *2* puede utilizar el octeto *u* para decidir qué anfitrión real debe recibir un datagrama. La WAN por sí misma no necesita enterarse de todos los anfitriones que se encuentran más allá del PSN.

Los ruteadores transparentes tienen ventajas y desventajas cuando se les compara con los ruteadores convencionales. La ventaja principal es que requieren menos direcciones de red, ya que la red de área local no necesita un prefijo IP por separado. Otra ventaja es que pueden incorporar el balanceo de carga. Esto es, si dos ruteadores transparentes se conectan a la misma red de área local, se puede dividir el tráfico hacia ellos. En comparación, los ruteadores convencionales sólo pueden manejar una ruta hacia cierta red.

Una desventaja de los ruteadores transparentes es que sólo trabajan con redes que tienen un espacio de direcciones grande, de donde escoger las de los anfitriones. Por lo tanto, trabajan bien con las redes tipo *A*, y no así con las redes tipo *C*. Otra desventaja es que, como no son ruteadores convencionales, los ruteadores transparentes no proporcionan los mismos servicios. En particular, los ruteadores transparentes quizá no participen del todo en los protocolos ICMP, o de manejo de red como SNMP. Por lo tanto, no generan respuestas de eco ICMP (por ejemplo, no se puede utilizar "ping" para determinar si un ruteador transparente está operando).

## 10.5 ARP sustituto (proxy ARP)

Los términos *ARP sustituto (proxy, ARP) promiscuo* y *ARP hack,* se refieren a la segunda técnica utilizada para transformar un solo prefijo IP de red en dos direcciones físicas. La técnica, que sólo se aplica en redes que utilizan ARP para convertir direcciones de red en direcciones físicas, se puede explicar mejor mediante un ejemplo. En la figura 10.2 se ilustra la situación.

En la figura, dos redes comparten una sola dirección IP. Imagine que la etiquetada como *Red Principal* era la red original y segunda, etiquetada como *Red Oculta*, se agregó después. *R*, que es el ruteador que conecta las dos redes, sabe qué anfitriones residen en cada red física y utiliza ARP para mantener la ilusión de que solamente existe una red. Para dar esa apariencia, *R* mantiene totalmente oculta la localización de los anfitriones, permitiendo que las demás máquinas en la red se comuniquen como si estuvieran conectadas de manera directa. En nuestro ejemplo, cuando el anfitrión $H_1$ necesita comunicarse con el anfitrión $H_4$, primero llama a ARP para convertir la dirección IP de $H_4$ en una dirección física. Una vez que tiene la dirección física, $H_1$ puede enviarle directamente el datagrama.

Debido a que el ruteador *R* corre software proxy ARP, *R* captura la solicitud transmitida por difusión de $H_1$, decide que la máquina en cuestión reside en la otra red física y responde la solicitud ARP enviando su propia dirección física. $H_1$ recibe la respuesta ARP, instala la asociación en su tabla ARP y la utiliza para enviar a *R* los datagramas destinados a $H_4$. Cuando *R* recibe un datagrama, busca en una tabla especial de ruteo para determinar cómo rutear el datagrama. *R* debe encami-

## 11.4 Funcionalidad de las capas

Una vez que se ha tomado la decisión de subdividir los problemas de comunicación en cuatro subproblemas y organizar el software de protocolo en módulos, de manera que cada uno maneje un subproblema, surge la pregunta: "¿qué tipo de funciones deben instalarse en cada módulo?" La pregunta no es fácil de responder por varias razones. En primer lugar, un conjunto de objetivos y condiciones determinan un problema de comunicación en particular, es posible elegir una organización que optimice el software de protocolo para ese problema. Segundo, incluso cuando se consideran los servicios generales a nivel de red, como un transporte confiable, es posible seleccionar entre distintas maneras de resolver el problema. Tercero, el diseño de una arquitectura de red (o de una red de redes) y la organización del software de protocolo están interrelacionados; no se puede diseñar a uno sin considerar al otro.

### 11.4.1 Modelo de referencia ISO de 7 capas

Existen dos ideas dominantes sobre la estratificación por capas de protocolos. La primera, basada en el trabajo realizado por la International Organization for Standardization (Organización Internacional para la Estandarización o ISO, por sus siglas en inglés), conocida como *Reference Model of Open System Interconnection* (*Modelo de referencia de interconexión de sistemas abiertos*) de ISO, denominada frecuentemente *modelo ISO*. El modelo ISO contiene 7 capas conceptuales organizadas como se muestra en la figura 11.4.

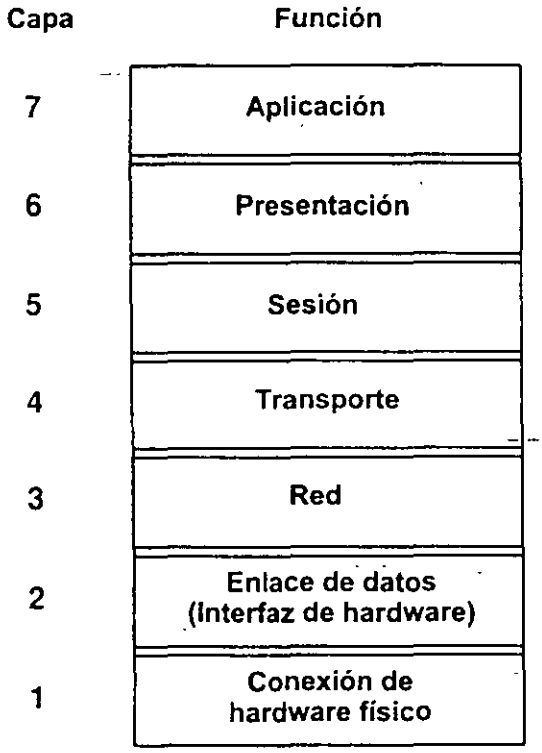



**Figura 11.4** Modelo de referencia de 7 capas ISO, para software de protocolo.

El modelo ISO, elaborado para describir protocolos para una sola red, no contiene un nivel específico para el ruteo en el enlace de redes, como sucede con el protocolo TCP/IP.

## 11.5 X.25 y su relación con el modelo ISO

Aun cuando fue diseñado para proporcionar un modelo conceptual y no una guía de implementación, el esquema de estratificación por capas de ISO ha sido la base para la implementación de varios protocolos. Entre los protocolos comúnmente asociados con el modelo ISO, el conjunto de protocolos conocido como X.25 es probablemente el mejor conocido y el más ampliamente utilizado. X.25 fue establecido como una recomendación de la *Telecommunications Section* de la *International Telecommunications Union*[1] (ITU-TS), una organización internacional que recomienda estándares para los servicios telefónicos internacionales. X.25 ha sido adoptado para las redes públicas de datos y es especialmente popular en Europa. Consideraremos a X.25 para ayudar a explicar la estratificación por capas de ISO.

Dentro de la perspectiva de X.25, una red opera en gran parte como un sistema telefónico. Una red X.25 se asume como si estuviera formada por complejos conmutadores de paquetes que tienen la capacidad necesaria para el ruteo de paquetes. Los anfitriones no están comunicados de manera directa a los cables de comunicación de la red. En lugar de ello, cada anfitrión se comunica con uno de los conmutadores de paquetes por medio de una línea de comunicación serial. En cierto sentido la comunicación entre un anfitrión y un conmutador de paquetes X.25 es una red miniatura que consiste en un enlace serial. El anfitrión puede seguir un complicado procedimiento para transferir paquetes hacia la red.

- *Capa física.* X.25 especifica un estándar para la interconexión física entre computadoras anfitrión y conmutadores de paquetes de red, así como los procedimientos utilizados para transferir paquetes de una máquina a otra. En el modelo de referencia, el nivel *1* especifica la interconexión física incluyendo las características de voltaje y corriente. Un protocolo correspondiente, X.21, establece los detalles empleados en las redes públicas de datos.
- *Capa de enlace de datos.* El nivel *2* del protocolo X.25 especifica la forma en que los datos viajan entre un anfitrión y un conmutador de paquetes al cual está conectado. X.25 utiliza el término *trama* para referirse a la unidad de datos cuando ésta pasa entre un anfitrión y un conmutador de paquetes (es importante entender que la definición de X.25 de *trama* difiere ligeramente de la forma en que la hemos empleado hasta aquí). Dado que el hardware, como tal, entrega sólo un flujo de bits, el nivel de protocolo *2* debe definir el formato de las tramas y especificar cómo las dos máquinas reconocen las fronteras de la trama. Dado que los errores de transmisión pueden destruir los datos, el nivel de protocolo *2* incluye una detección de errores (esto es, una suma de verificación de trama). Finalmente, dado que la transmisión es no confiable, el nivel de protocolo *2* especifica un intercambio de acuses de recibo que permite a las dos máquinas saber cuándo se ha transferido una trama con éxito.

Hay protocolo de nivel 2, utilizado comúnmente, que se conoce como *High Level Data Link Communication (Comunicación de enlace de datos de alto nivel)*, mejor conocido por sus siglas, *HDLC*. Existen varias versiones del HDLC, la más reciente es conocida como *HDLC/LAPB*. Es

[1] La International Telecommunications Union fue llamada formalmente *Consultative Committee on International Telephony and Telegraphy (CCITT).*

importante recordar que una transferencia exitosa en el nivel *2* significa que una trama ha pasado hacia un conmutador de paquetes de red para su entrega; esto no garantiza que el conmutador de paquetes acepte el paquete o que esté disponible para rutearlo.

- *Capa de red.* El modelo de referencia ISO especifica que el tercer nivel contiene funciones que completan la interacción entre el anfitrión y la red. Conocida como capa de *red* o *subred de comunicación,* este nivel define la unidad básica de transferencia a través de la red e incluye el concepto de direccionamiento de destino y ruteo. Debe recordarse que en el mundo de X.25 la comunicación entre el anfitrión y el conmutador de paquetes está conceptualmente aislada respecto al tráfico existente. Así, la red permitiría que paquetes definidos por los protocolos del nivel 3 sean mayores que el tamaño de la trama que puede ser transferida en el nivel *2*. El software del nivel 3 ensambla un paquete en la forma esperada por la red y utiliza el nivel *2* para transferirlo (quizás en fragmentos) hacia el conmutador de paquetes. El nivel 3 también debe responder a los problemas de congestionamiento en la red.
- *Capa de transporte.* El nivel 4 proporciona confiabilidad punto a punto y mantiene comunicados al anfitrión de destino con el anfitrión fuente. La idea aquí es que, así como en los niveles inferiores de protocolos se logra cierta confiabilidad verificando cada transferencia, la capa punto a punto duplica la verificación para asegurarse de que ninguna máquina intermedia ha fallado.
- *Capa de sesión.* Los niveles superiores del modelo ISO describen cómo el software de protocolo puede organizarse para manejar todas las funciones necesarias para los programas de aplicación. El comité ISO consideró el problema del acceso a una terminal remota como algo tan importante que asignó la capa 5 para manejarlo. De hecho, el servicio central ofrecido por las primeras redes públicas de datos consistía en una terminal para la interconexión de anfitriones. Las compañías proporcionaban en la red, mediante una línea de marcación, una computadora anfitrión de propósito especial, llamada *Packet Assembler and Disassembler (Ensamblador y desensamblador de paquetes o PAD,* por sus siglas en inglés). Los suscriptores, por lo general viajeros que transportaban su propia computadora y su módem, se ponían en contacto con la PAD local, haciendo una conexión de red hacia el anfitrión con el que deseaban comunicarse. Muchas compañías prefirieron comunicarse por medio de la red para su comunicación por larga distancia, porque resultaba menos cara que la marcación directa.
- *Capa de presentación.* La capa 6 de ISO está proyectada para incluir funciones que muchos programas de aplicación necesitan cuando utilizan la red. Los ejemplos comunes incluyen rutinas estándar que comprimen texto o convierten imágenes gráficas en flujos de bits para su transmisión a través de la red. Por ejemplo, un estándar ISO, conocido como *Abstract Syntax Notation 1* (*Notación de sintaxis abstracta 1* o *ASN.1,* por sus siglas en inglés), proporciona una representación de datos que utilizan los programas de aplicación. Uno de los protocolos TCP/IP, SNMP, también utiliza ASN.1 para representar datos.
- *Capa de aplicación.* Finalmente, la capa 7 incluye programas de aplicación que utilizan la red. Como ejemplos de esto se tiene al correo electrónico o a los programas de transferencia de archivos. En particular, el ITU-TS tiene proyectado un protocolo para correo electrónico, conocido como estándar *X.400*. De hecho, el ITU y el ISO trabajan juntos en el sistema de manejo de mensajes; la versión de ISO es conocida como *MOTIS.*

### 11.5.1 El modelo de estratificación por capas de TCP/IP de Internet

El segundo modelo mayor de estratificación por capas no se origina de un comité de estándares, sino que proviene de las investigaciones que se realizan respecto al conjunto de protocolos de TCP/IP. Con un poco de esfuerzo, el modelo ISO puede ampliarse y describir el esquema de estratificación por capas del TCP/IP, pero los presupuestos subyacentes son lo suficientemente distintos para distinguirlos como dos diferentes.

En términos generales, el software TCP/IP está organizado en cuatro capas conceptuales que se construyen sobre una quinta capa de hardware. La figura 11.5 muestra las capas conceptuales así como la forma en que los datos pasan entre ellas.

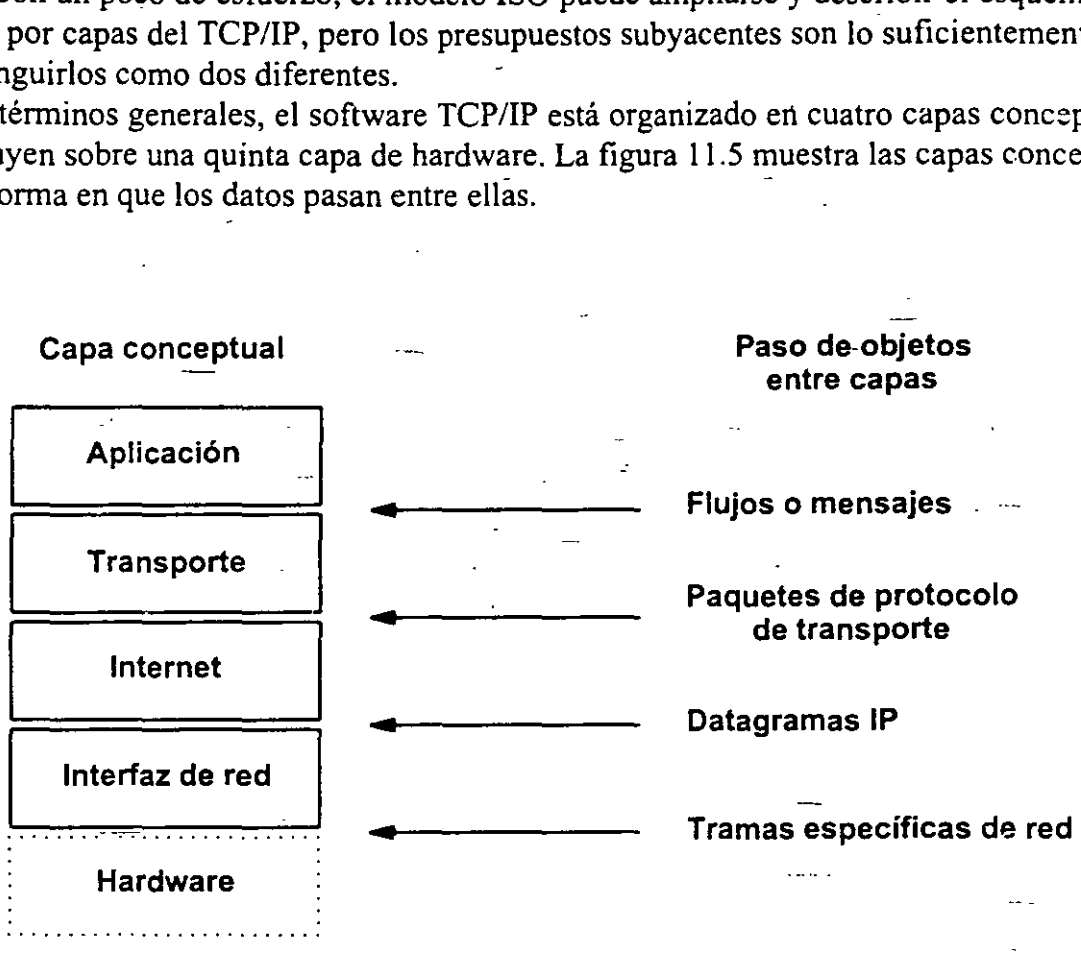



**Figura 11.5** Las cuatro capas conceptuales del software TCP/IP y la forma en que los objetos pasan entre capas. La capa con el nombre *interfaz de red* se conoce con frecuencia con el nombre de capa de *enlace de datos*.

- *Capa de aplicación.* En el nivel más alto, los usuarios llaman a una aplicación que accesa servicios disponibles a través de la red de redes TCP/IP. Una aplicación interactúa con uno de los protocolos de nivel de transporte para enviar o recibir datos. Cada programa de aplicación selecciona el tipo de transporte necesario, el cual puede ser una secuencia de mensajes individuales o un flujo continuo de octetos. El programa de aplicación pasa los datos en la forma requerida hacia el nivel de transporte para su entrega.
- *Capa de transporte.* La principal tarea de *la capa de transporte* es proporcionar la comunicación entre un programa de aplicación y otro. Este tipo de comunicación se conoce frecuentemente como comunicación *punto a punto*. La capa de transporte regula el flujo de información. Puede también proporcionar un transporte confiable, asegurando que los datos lleguen sin errores y en secuencia. Para hacer esto, el software de protocolo de transporte tiene el lado de recepción enviando

acuses de recibo de retorno y la parte de envío retransmitiendo los paquetes perdidos. El software de transporte divide el flujo de datos que se está enviando en pequeños fragmentos (por lo general conocidos como *paquetes*) y pasa cada paquete, con una dirección de destino, hacia la siguiente capa de transmisión.

Aun cuando en la figura 11.5 se utiliza un solo bloque para representar la capa de aplicación, una computadora de propósito general puede tener varios programas de aplicación accesando la red de redes al mismo tiempo. La capa de transporte debe aceptar datos desde varios programas de usuario y enviarlos a la capa del siguiente nivel. Para hacer esto, se añade información adicional a cada paquete, incluyendo códigos que identifican qué programa de aplicación envía y qué programa de aplicación debe recibir, así como una suma de verificación. La máquina de recepción utiliza la suma de verificación para verificar que el paquete ha llegado intacto y utiliza el código de destino para identificar el programa de aplicación en el que se debe entregar.

- *Capa Internet.* Como ya lo hemos visto, la capa Internet maneja la comunicación de una máquina a otra. Ésta acepta una solicitud para enviar un paquete desde la capa de transporte, junto con una identificación de la máquina, hacia la que se debe enviar el paquete. Encapsula el paquete en un datagrama IP, llena el encabezado del datagrama, utiliza un algoritmo de ruteo para determinar si puede entregar el datagrama directamente o si debe enviarlo a un ruteador y pasar el datagrama hacia la interfaz de red apropiada para su transmisión. La capa Internet también maneja la entrada de datagramas, verifica su validez y utiliza un algoritmo de ruteo para decidir si el datagrama debe procesarse de manera local o debe ser transmitido. Para el caso de los datagramas direccionados hacia la máquina local, el software de la capa de red de redes borra el encabezado del datagrama y selecciona, de entre varios protocolos de transporte, un protocolo con el que manejará el paquete. Por último, la capa Internet envía los mensajes ICMP de error y control necesarios y maneja todos los mensajes ICMP entrantes.
- *Capa de interfaz de red.* El software TCP/IP de nivel inferior consta de una capa de interfaz de red responsable de aceptar los datagramas IP y transmitirlos hacia una red específica. Una interfaz de red puede consistir en un dispositivo controlador (por ejemplo, cuando la red es una red de área local a la que las máquinas están conectadas directamente) o un complejo subsistema que utiliza un protocolo de enlace de datos propio (por ejemplo, cuando la red consiste de conmutadores de paquetes que se comunican con anfitriones utilizando HDLC).

## 11.6 Diferencias entre X.25 y la estratificación por capas de Internet

Hay dos diferencias importantes y sutiles entre el esquema de estratificación por capas del TCP/IP y el esquema X.25. La primera diferencia gira en torno al enfoque de la atención de la confiabilidad, en tanto que la segunda comprende la localización de la inteligencia en el sistema completo.

### 11.6.1 Niveles de enlace y confiabilidad punto a punto

Una de las mayores diferencias entre los protocolos TCP/IP y X.25 reside en su enfoque respecto a los servicios confiables de entrega de datos. En el modelo X.25, el software de protocolo detecta y maneja errores en todos los niveles. En el nivel de enlace, protocolos complejos garantizan que la

transferencia, entre un anfitrión y un conmutador de paquetes que están conectados, se realice correctamente. Una suma de verificación acompaña a cada fragmento de datos transferido y el receptor envía acuses de recibo de cada segmento de datos recibido. El protocolo de nivel de enlace incluye intervalos de tiempo y algoritmos de retransmisión que evitan la pérdida de datos y proporcionan una recuperación automática después de las fallas de hardware y su reiniciación.

Los niveles sucesivos de X.25 proporcionan confiabilidad por sí mismos. En el nivel *3*, X.25 también proporciona detección de errores y recuperación de transferencia de paquetes en la red mediante el uso de sumas de verificación así como de intervalos de tiempo y técnicas de retransmisión. Por último, el nivel *4* debe proporcionar confiabilidad punto a punto pues tiene una correspondencia entre la fuente y el destino final para verificar la entrega.

En contraste con este esquema, el TCP/IP basa su estratificación por capas de protocolos en la idea de que la confiabilidad punto a punto es un problema. La filosofía de su arquitectura es sencilla: una red de redes se debe construir de manera que pueda manejar la carga esperada, pero permitiendo que las máquinas o los enlaces individuales pierdan o alteren datos sin tratar repetidamente de recuperarlos. De hecho, hay una pequeña o nula confiabilidad en la mayor parte del software de las capas de interfaz de red. En lugar de esto, las capas de transporte manejan la mayor parte de los problemas de detección y recuperación de errores.

El resultado de liberar la capa de interfaz de la verificación hace que el software TCP/IP sea mucho más fácil de entender e implementar correctamente. Los ruteadores intermedios pueden descartar datagramas que se han alterado debido a errores de transmisión. Pueden descartar datagramas que no se pueden entregar o que, a su llegada, exceden la capacidad de la máquina y pueden rutear de nuevo datagramas a través de vías con retardos más cortos o más largos sin informar a la fuente o al destino.

Tener enlaces no confiables significa que algunos datagramas no llegarán a su destino. La detección y la recuperación de los datagramas perdidos se establece entre el anfitrión fuente y el destino final y se le llama verificación *end-to-end*.[2] El software extremo a extremo que se ubica en la capa de transporte utiliza sumas de verificación, acuses de recibo e intervalos de tiempo para controlar la transmisión. Así, a diferencia del protocolo X.25, orientado a la conexión, el software TCP/IP enfoca la mayor parte del control de la confiabilidad hacia una sola capa.

### 11.6.2 Localización de la inteligencia y la toma de decisiones

Otra diferencia entre el modelo X.25 y el modelo TCP/IP se pone de manifiesto cuando consideramos la localización de la autoridad y el control. Como regla general, las redes que utilizan X.25 se adhieren a la idea de que una red es útil porque proporciona un servicio de transporte. El vendedor que ofrece el servicio controla el acceso a la red y monitorea el tráfico para llevar un registro de cantidades y costos. El prestador de servicios de la red también maneja internamente problemas como el ruteo, el control de flujo y los acuses de recibo, haciendo la transferencia confiable. Este enfoque hace que los anfitriones puedan (o necesiten) hacer muy pocas cosas. De hecho, la red es un sistema complejo e independiente en el que se pueden conectar computadoras anfitrión relativamente simples; los anfitriones por sí mismos participan muy poco en la operación de la red.

---

[2] N del T: podría entenderse como *verificación punto a punto* o extremo a extremo, a diferencia de la *confiabilidad punto a punto* de la que se habla en párrafos anteriores).

En contraste con esto, el TCP/IP requiere que los anfitriones participen en casi todos los protocolos de red. Ya hemos mencionado que los anfitriones implementan activamente la detección y la corrección de errores de extremo a extremo. También participan en el ruteo puesto que deben seleccionar una ruta cuando envían datagramas y participan en el control de la red dado que deben manejar los mensajes de control ICMP. Así, cuando la comparamos con una red X.25, una red de redes TCP/IP puede ser vista como un sistema de entrega de paquetes relativamente sencillo, el cual tiene conectados anfitriones inteligentes.

## 11.7 El principio de la estratificación por capas de protocolos

Independientemente del esquema de estratificación por capas que se utilice o de las funciones de las capas, la operación de los protocolos estratificados por capas se basa en una idea fundamental. La idea, conocida como *principio de estratificación por capas* puede resumirse de la siguiente forma:

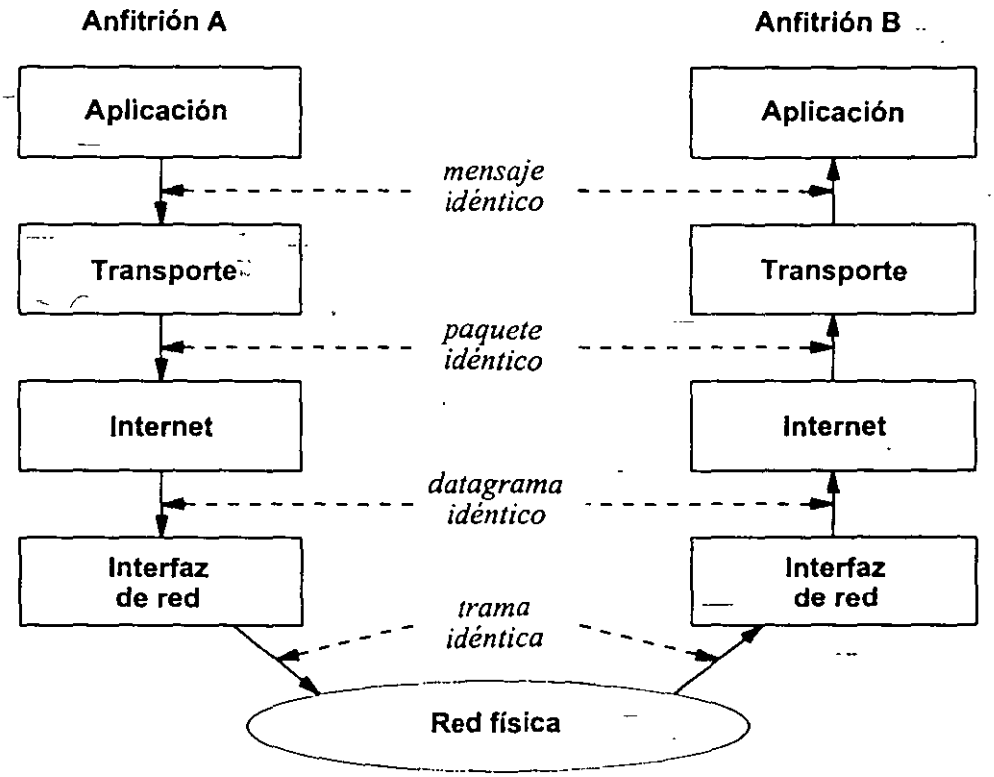



**Figura 11.6** Trayectoria de un mensaje cuando pasa de la aplicación en un anfitrión a la aplicación en otro. Una capa *n* en el anfitrión *B* recibe exactamente el mismo objeto que la capa *n* correspondiente del anfitrión emisor *A*.

*Los protocolos estratificados por capas están diseñados de modo que una capa* n *en el receptor de destino reciba exactamente el mismo objeto enviado por la correspondiente capa* n *de la fuente.*

El principio de estratificación por capas explica por qué la estratificación por capas es una idea poderosa. Ésta permite que el diseñador de protocolos enfoque su atención hacia una capa a la vez, sin preocuparse acerca del desempeño de las capas inferiores. Por ejemplo, cuando se construye una aplicación para transferencia de archivos, el diseñador piensa sólo en dos copias del programa de aplicación que se correrá en dos máquinas y se concentrará en los mensajes que se necesitan intercambiar para la transferencia de archivos. El diseñador asume que la aplicación en el anfitrión receptor es exactamente la misma que en el anfitrión emisor.

La figura 11.6 ilustra cómo trabaja el principio de estratificación por capas:

### 11.7.1 Estratificación por capas en un ambiente de Internet TCP/IP

Nuestro planteamiento sobre el principio de estratificación por capas es un tanto vago y la ilustración de la figura 11.6 toca un tema importante dado que permite distinguir entre la transferencia desde una fuente hasta un destino final y la transferencia a través de varias redes. La figura 11.7.

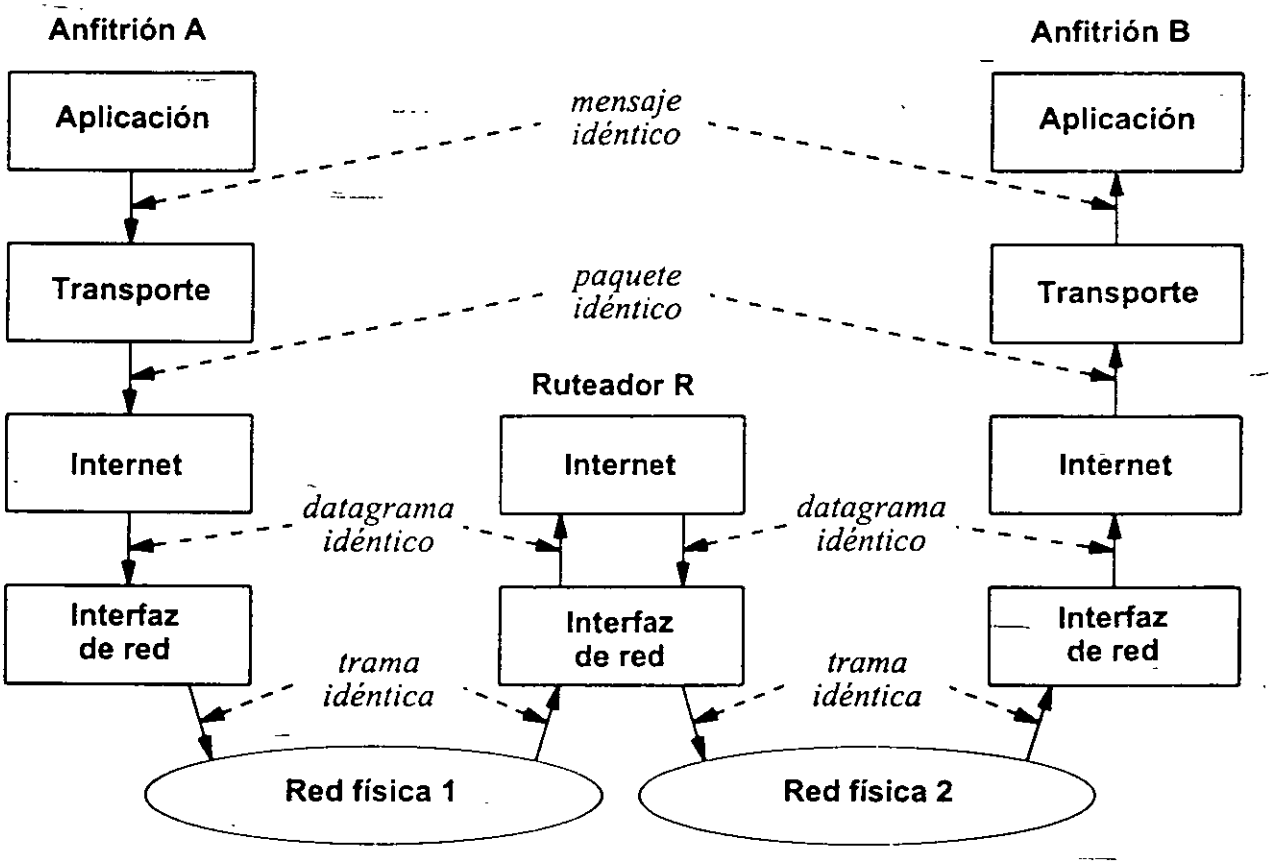



**Figura 11.7** Principio de estratificación por capas cuando se utiliza un ruteador. La trama entregada al ruteador *R* es exactamente la trama enviada desde el anfitrión *A*, pero difiere de la trama enviada entre *R* y *B*.

ilustra la distinción y muestra el trayecto de un mensaje enviado desde un programa de aplicación en un anfitrión hacia la aplicación en otro a través de un ruteador.

Como se muestra en la figura, la entrega del mensaje utiliza dos estructuras de red separadas, una para la transmisión desde el anfitrión *A* hasta el ruteador *R* y otra del ruteador *R* al anfitrión *B*. El siguiente principio de trabajo de estratificación de capas indica que el marco entregado a *R* es idéntico al enviado por el anfitrión *A*. En contraste, las capas de aplicación y transporte cumplen con la condición punto a punto y están diseñados de modo que el software en la fuente se comunique con su par en el destino final. Así, el principio de la estratificación por capas establece que el paquete recibido por la capa de transporte en el destino final es idéntico al paquete enviado por la capa de transporte en la fuente original.

Es fácil entender que, en las capas superiores, el principio de estratificación por capas se aplica a través de la transferencia punto a punto y que en las capas inferiores se aplica en una sola transferencia de máquina. No es tan fácil ver cómo el principio de estratificación de capas se aplica a la estratificación Internet. Por un lado, hemos dicho que los anfitriones conectados a una red de redes deben considerarse como una gran red virtual, con los datagramas IP que hacen las veces de tramas de red. Desde este punto de vista, los datagramas viajan desde una fuente original hacia un destino final y el principio de la estratificación por capas garantiza que el destino final reciba exactamente el datagrama que envió la fuente. Por otra parte, sabemos que el encabezado "datagram" contiene campos, como "*time to live*", que cambia cada vez que el "*datagram*" pasa a través de un ruteador. Así, el destino final no recibirá exactamente el mismo diagrama que envió la fuente. Debemos concluir que, a pesar de que la mayor parte de los datagramas permanecen intactos cuando pasan a través de una red de redes, el principio de estratificación por capas sólo se aplica a los datagramas que realizan transferencias de una sola máquina. Para ser precisos, no debemos considerar que las capas de Internet proporcionan un servicio punto a punto.

## 11.8 Estratificación por capas en presencia de una subestructura de red

Recordemos, del capítulo 2, que algunas redes de área amplia contienen varios conmutadores de paquetes. Por ejemplo, una WAN puede consistir en ruteadores conectados a una red local en cada localidad así como a otros ruteadores que utilizan líneas en serie arrendadas. Cuando un ruteador recibe un datagrama, éste puede entregar el datagrama en su destino o en la red local, o transferir el datagrama a través de una línea serial hacia otro ruteador. La cuestión es la siguiente: "¿cómo se ajusta el protocolo utilizado en una línea serial con respecto al esquema de estratificación por capas del TCP/IP?" La respuesta depende de cómo considere el diseñador la interconexión con la línea serial.

Desde la perspectiva del IP, el conjunto de conexiones punto a punto entre ruteadores puede funcionar como un conjunto de redes físicas independientes o funcionar colectivamente como una sola red física. En el primer caso, cada enlace físico es tratado exactamente como cualquier otra red en una red de redes. A ésta se le asigna un número único de red (por lo general de clase C) y los dos anfitriones que comparten el enlace tiene cada uno una dirección única IP asignada para su conexión. Los ruteadores se añaden a la tabla de ruteo IP como lo harían para cualquier otra red. Un nuevo módulo de software se añade en la capa de interfaz de red para controlar el nuevo enlace

de hardware, pero no se realizan cambios sustanciales en el esquema de estratificación por capas. La principal desventaja del enfoque de redes independientes es la proliferación de números de red (uno por cada conexión entre dos máquinas), lo que ocasiona que las tablas de ruteo sean tan grandes como sea necesario. Tanto la *línea serial IP* (*Serial Line IP o SLIP*) como el *protocolo punto a punto* (*Point to Point Protocol o PPP*) tratan a cada enlace serial como una red separada.

El segundo método para ajustar las conexiones punto a punto evita asignar múltiples direcciones IP al cableado físico. En lugar de ello, se tratan a todas las conexiones colectivamente como una sola red independiente IP con su propio formato de trama, esquema de direccionamiento de hardware y protocolos de enlace de datos. Los ruteadores que emplean el segundo método necesitan sólo un número de red IP para todas las conexiones punto a punto.

Usar el enfoque de una sola red significa extender el esquema de estratificación por capas de protocolos para añadir una nueva capa de ruteo dentro de la red, entre la capa de interfaz de red y los dispositivos de hardware. Para las máquinas con una sola conexión punto a punto, una capa adicional parece innecesaria. Para comprender por qué es necesaria, considere una máquina con varias conexiones físicas punto a punto y recuerde de la figura 11.2 cómo la capa de interfaz de red se divide en varios módulos de software que controlan cada uno una red. Necesitamos añadir una interfaz de red nueva para la nueva red punto a punto, pero la interfaz nueva debe controlar varios dispositivos de hardware. Además, dado un datagrama para envío, la nueva interfaz debe seleccionar el enlace correcto por el que el datagrama será enviado. La figura 11.8 muestra la organización.

El software de la capa Internet pasa hacia la interfaz de red todos los datagramas que deberán enviarse por cualquier conexión punto a punto. La interfaz los pasa hacia el módulo de ruteo dentro de la red que, además, debe distinguir entre varias conexiones físicas y rutear el datagrama a través de la conexión correcta.

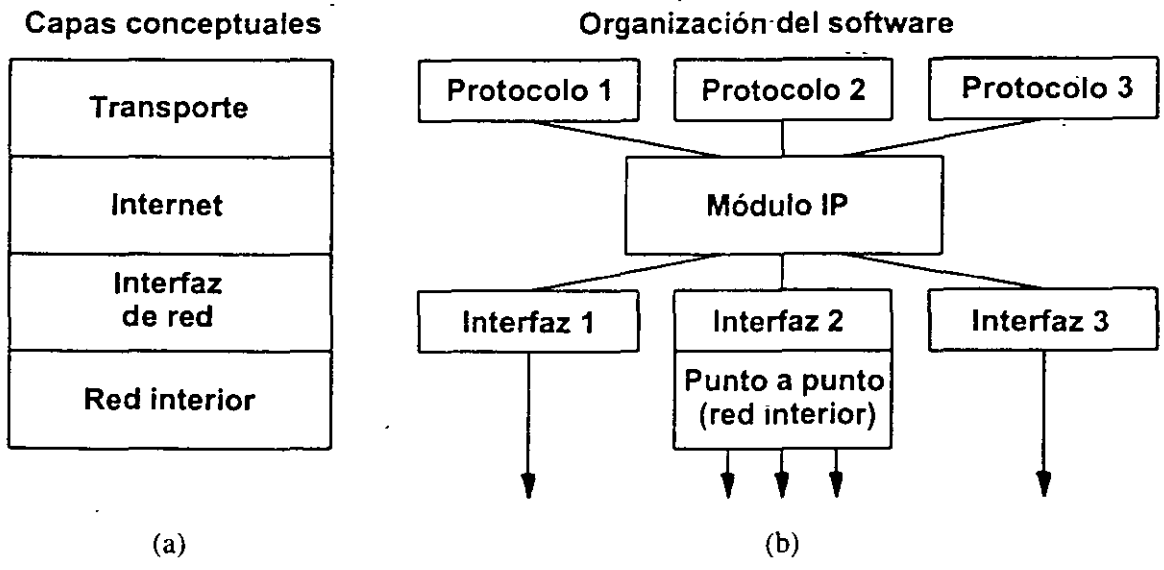



**Figura 11.8** (a) posición de los conceptos de un protocolo de red interior, para conexiones punto a punto, cuando el IP la trata como una sola red IP, y (b) diagrama detallado de los módulos de software correspondientes. Cada flecha corresponde a un dispositivo físico.

El programador que diseña software de ruteo dentro de la red determina exactamente cómo selecciona el software un enlace físico. Por lo general, el algoritmo conduce a una tabla de ruteo dentro de la red. La tabla de ruteo dentro de la red es análoga a una tabla de ruteo de una red de redes en la que se especifica una transformación de la dirección de destino hacia la ruta. La tabla contiene pares de enteros, *(D, L)*, donde *D* es una dirección de destino de un anfitrión y *L* especifica una de las líneas físicas utilizadas para llegar al destino.

Las diferencias entre una tabla de ruteo de red de redes y una tabla de ruteo dentro de la red es que esta última, es mucho más pequeña. Contiene solamente información de ruteo para los anfitriones conectados directamente a la red punto a punto. La razón es simple: la capa Internet realiza la transformación de una dirección de destino arbitraria hacia una ruta de dirección específica antes de pasar el datagrama hacia una interfaz de red. De esta manera, la capa dentro de la red sólo debe distinguir entre máquinas en una sola red punto a punto.

## 11.9 Dos fronteras importantes en el modelo TCP/IP

La estratificación por capas conceptual incluye dos fronteras que podrían no ser obvias: una frontera de dirección de protocolo que separa los direccionamientos de alto nivel y de bajo nivel, y una frontera de sistema operativo que separa al sistema de los programas de aplicación.

### 11.9.1 Frontera de dirección de protocolo de alto nivel

Ahora que hemos visto la capa de software TCP/IP, podemos precisar una idea introducida en el capítulo 8: la partición de una frontera conceptual entre el software que utiliza direcciones de bajo nivel (físicas), con respecto a un software que utiliza direcciones de alto nivel (IP). Como se muestra en la figura 11.9, la frontera aparece entre la capa de interfaz de red y la capa de Internet. Esto es,

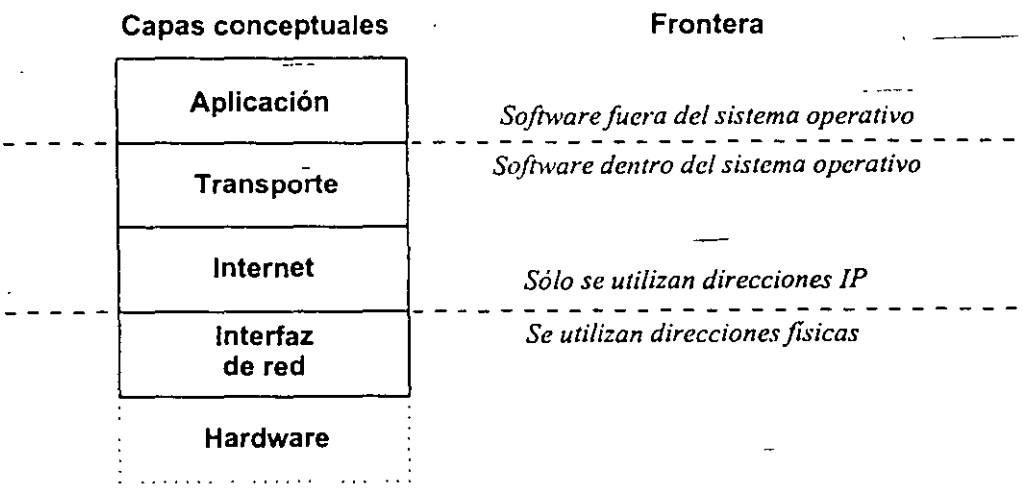



**Figura 11.9** Relación entre la estratificación por capas conceptual y las fronteras, para el sistema operativo y las direcciones de protocolo de alto nivel.

*Los programas de aplicación, así como todo el software del protocolo desde la capa de Internet hacia arriba, utiliza sólo direcciones IP; la capa de interfaz de red maneja direcciones físicas.*

Así, protocolos como ARP pertenecen a la capa de interfaz de red. Estos no son parte del IP.

### 11.9.2 Frontera de sistema operativo

La figura 11.9 muestra otra frontera también importante, la división entre el software que generalmente se considera parte del sistema operativo respecto al software que no lo es. En tanto que cada implantación del TCP/IP determina cómo se establece la distinción, muchos siguen el esquema mostrado. Dado que los colocan dentro el sistema operativo, el paso de datos entre las capas inferiores del software de protocolo es mucho menos caro que su paso entre un programa de aplicación y una capa de transporte. En el capítulo 20, se trata el problema con mayor detalle y se describe un ejemplo de la interfaz de un sistema operativo.

## 11.10 La desventaja de la estratificación por capas

Se ha mencionado el hecho de que la estratificación por capas es una idea fundamental que proporciona las bases para el diseño de protocolos. Permite al diseñador dividir un problema complicado en subproblemas y resolver cada parte de manera independiente. Por desgracia, el software resultante de una estratificación por capas estrictas puede ser muy ineficaz. Como ejemplo, considere el trabajo de la capa de transporte. Debe aceptar un flujo de octetos desde un programa de aplicación, dividir el flujo en paquetes y enviar cada paquete a través de la red de redes. Para optimizar la transferencia, la capa de transporte debe seleccionar el tamaño de paquete más grande posible que le permita a un paquete viajar en una trama de red. En particular, si la máquina de destino está conectada a una máquina de la misma red de la fuente, sólo la red física se verá involucrada en la transferencia, así, el emisor puede optimizar el tamaño del paquete para esta red. Si el software preserva una estricta estratificación por capas, sin embargo, la capa de transporte no podrá saber cómo ruteará el módulo de Internet el tráfico o qué redes están conectadas directamente. Más aún, la capa de transporte no comprenderá el datagrama o el formato de trama ni será capaz de determinar cómo deben ser añadidos muchos octetos de encabezado a un paquete. Así, una estratificación por capas estricta impedirá que la capa de transporte optimice la transferencia.

Por lo general, las implantaciones atenúan el esquema estricto de la estratificación por capas cuando construyen software de protocolo. Permiten que información como la selección de ruta y la MTU de red se propaguen hacia arriba. Cuando los búfers realizan el proceso de asignación, generalmente dejan espacio para encabezados que serán añadidos por los protocolos de las capas de bajo nivel y pueden retener encabezados de las tramas entrantes cuando pasan hacia protocolos de capas superiores. Tal optimización puede producir mejoras notables en la eficiencia siempre y cuando conserve la estructura básica en capas.

## 11.11 La idea básica detrás del multiplexado y el demultiplexado

Los protocolos de comunicación utilizan técnicas de *multiplexado* y *demultiplexado* a través de la jerarquía de capas. Cuando envía un mensaje, la computadora fuente incluye bits extras que codifican el tipo de mensaje, el programa de origen y los protocolos utilizados. Finalmente, todos los mensajes son colocados dentro de tramas de red para transferirse y combinarse en flujos de paquetes. En el extremo de recepción, la máquina destino se vale de información extra para guiar el proceso.

Considere el ejemplo de demultiplexado que se muestra en la figura 11.10.

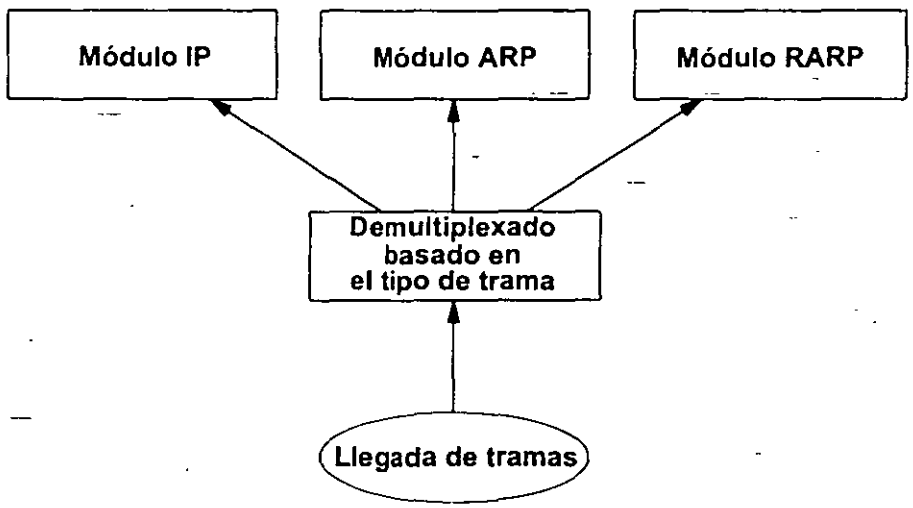



**Figura 11.10** Demultiplexado de tramas entrantes basado en el campo de tipo que se encuentra en el encabezado de la trama.

La figura muestra de qué manera utiliza el software, en la capa de interfaz de red, el tipo de trama para seleccionar un procedimiento que permita manejar las tramas entrantes. Se dice que la interfaz de red *demultiplexa* la trama con base en este tipo. Para hacer posible la selección, el software en la máquina fuente debe establecer el campo del tipo de trama antes de la transmisión. Así, cada módulo de software que envía tramas emplea el campo de_tipo para especificar el contenido de la trama.

El multiplexado y el demultiplexado se presentan en casi todas las capas de protocolo. Por ejemplo, luego de que la interfaz de red demultiplexa tramas y pasa las tramas que contienen datagramas IP hacia el módulo IP, el software IP extrae el datagrama y lo demultiplexa con base en el protocolo de transporte. La figura 11.11 muestra el multiplexado en la capa Internet.

Para decidir cómo manejar un datagrama, el software de red de redes examina el encabezado de un datagrama y, para su manejo, selecciona un protocolo con base en el tipo de datagrama. En el ejemplo, los tipos posibles de datagramas son: *ICMP*, que ya hemos examinado, y *UDP, TCP* y *EGP*, que examinaremos en capítulos posteriores.

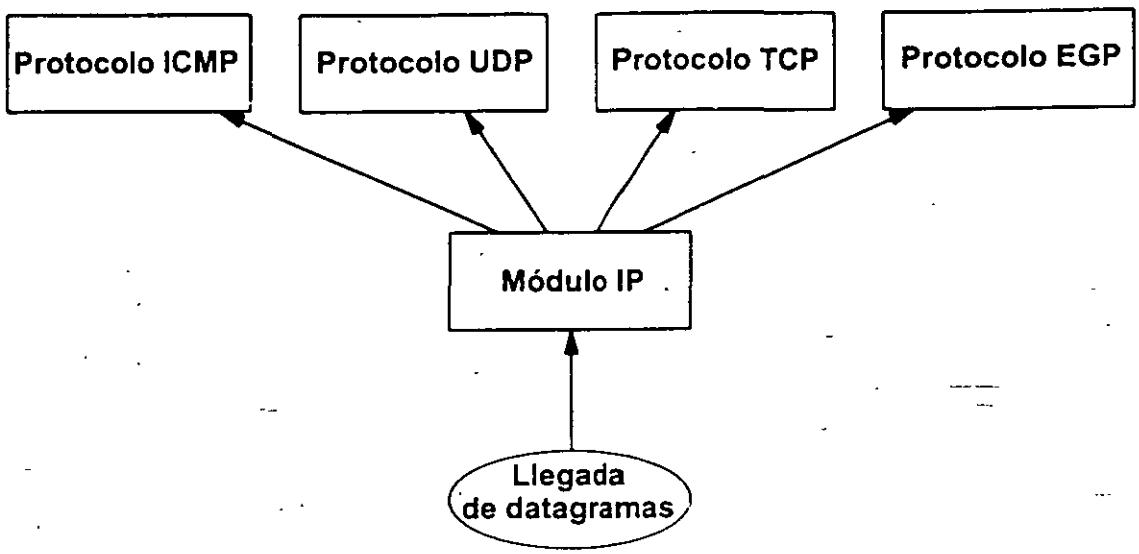



**Figura 11.11** Demultiplexado en la capa Internet. El software IP selecciona un procedimiento apropiado para manejar un datagrama, basándose en el campo de tipo de protocolo, localizado en el encabezado del datagrama.

## 11.12 Resumen

Los protocolos son los estándares que especifican cómo se representan los datos cuando son transferidos de una máquina a otra. También, especifican cómo se da la transferencia, cómo se detectan los errores y cómo se envían los acuses de recibo. Para simplificar el diseño y la implantación de los protocolos, los problemas de comunicación se transfieren hacia subproblemas que se pueden resolver de manera independiente. Cada problema se asigna a un protocolo separado.

La idea de la estratificación por capas es fundamental porque proporciona una estructura conceptual para el diseño de protocolos. En el modelo por capas, cada capa maneja una parte de los problemas de comunicación y generalmente se asocian a un protocolo. Los protocolos siguen el principio de la estratificación por capas, el cual establece que la implantación del software de una capa *n* en una máquina de destino recibe exactamente la implementación del software de la capa *n* en la fuente de la máquina emisora.

Examinamos el modelo de referencia de Internet de 4 capas, así como el modelo de referencia de 7 capas ISO. En ambos casos el modelo de estratificación por capas proporciona sólo una estructura conceptual para el software de protocolo. Los protocolos X.25 de ITU-TS siguen el modelo de referencia ISO y proporcionan un ejemplo de servicio de comunicación confiable ofrecido por una infraestructura comercial, mientras que los protocolos TCP/IP proporcionan un ejemplo diferente de un esquema de estratificación por capas.

En la práctica, el software de protocolo utiliza el multiplexado y el demultiplexado para distinguir entre varios protocolos dentro de una capa dada, haciendo el software de protocolo más complejo que como lo sugiere el modelo de estratificación por capas.

## PARA CONOCER MÁS

Postel (RFC 791) ofrece un bosquejo del esquema de estratificación por capas del Protocolo Internet y Clark (RFC 817) analiza los efectos de la estratificación por capas en las implantaciones. Saltzer, Reed y Clark (1984) plantean que la verificación extremo a extremo es importante. Chesson (1987) hace una exposición controvertida, según la cual la estratificación por capas produce un rendimiento total de la red malo e intolerable. En el volumen 2 de esta obra, se examina la estratificación por capas a detalle y se muestra un ejemplo de implantación con el que se logra la eficiencia mediante un compromiso entre la estratificación por capas estricta y el paso de punteros entre capas.

Los documentos de protocolo ISO (1987a) y (1987b) describen al ASN.1 en detalle. Sun (RFC 1014) describe XDR, un ejemplo de lo que podría llamarse un protocolo de presentación TCP/IP. Clark trata el paso de la información hacia arriba a través de las capas (Clark 1985).

## EJERCICIOS

**11.1** Estudie el modelo de estratificación por capas con mayor detalle. ¿Cómo describiría el modelo de comunicación en una red de área local como Ethernet?

**11.2** Elabore un caso en el que TCP/IP se mueva hacia una arquitectura de protocolo de cinco niveles que incluya una capa de presentación. (Sugerencia: varios programas utilizan el protocolo XDR, Courier y ASN.1)

**11.3** ¿Piensa usted que un solo protocolo de presentación emergería eventualmente remplazando a todos los demás? ¿Por qué sí, o por qué no?

**11.4** Compare y contraste el formato de datos etiquetado utilizado por el esquema de presentación ASN.1, con el formato no etiquetado, utilizado por XDR. Especifique situaciones en que uno sea mejor que el otro.

**11.5** Encuentre cómo utiliza un sistema UNIX la estructura *mbuf* para hacer eficiente el software de protocolo de estratificación por capas.

**11.6** Lea acerca del mecanismo *streams* del sistema UNIX V. ¿En qué forma ayuda a hacer más fácil la implantación del protocolo? ¿Cuál es su mayor desventaja?

# 12

# Protocolo de datagrama de usuario (UDP)

## 12.1 Introducción

En los capítulos anteriores, se describió una red de redes TCP/IP capaz de transferir datagramas IP entre computadores anfitriones, donde cada datagrama se rutea a través de la red, basándose en la dirección IP de destino. En la capa del Protocolo Internet, una dirección de destino identifica una computadora anfitrión; no se hace ninguna otra distinción con respecto a qué usuario o qué programa de aplicación recibirá el datagrama. En este capítulo se amplía el grupo de protocolos TCP/IP al agregar un mecanismo que distingue entre muchos destinos dentro de un anfitrión, permitiendo que varios programas de aplicación que se ejecutan en una computadora envíen y reciban datagramas en forma independiente.

## 12.2 Identificación del destino final

Los sistemas operativos de la mayor parte de las computadoras aceptan la multiprogramación, que significa permitir que varios programas de aplicación se ejecuten al mismo tiempo. Utilizando la jerga de los sistemas operativos, nos referimos a cada programa en ejecución como un *proceso, tarea, programa de aplicación* o *proceso a nivel de usuario*; a estos sistemas se les llama sistemas multitarea. Puede parecer natural decir que un proceso es el destino final de un mensaje. Sin embargo, especificar que un proceso en particular en una máquina en particular es el destino final para un datagrama es un poco confuso. Primero, por que los procesos se crean y destruyen de manera dinámica, los transmisores rara vez saben lo suficiente para identificar un proceso en otra máquina. Se-

;undo, nos gustaría poder reemplazar los procesos que reciben datagramas, sin tener que informar a todos los transmisores (por ejemplo, reiniciar una máquina puede cambiar todos los procesos, pero los transmisores no están obligados a saber sobre los nuevos procesos). Tercero, necesitamos identificar los destinos de las funciones que implantan sin conocer el proceso que implanta la función (por ejemplo, permitir que un transmisor contacte un servidor de archivos sin saber qué proceso en la máquina de destino implanta la función de servidor de archivos). También es importante saber que, en los sistemas que permiten que un solo proceso maneje dos o más funciones, es esencial que encontremos una forma para que un proceso decida exactamente qué función desea el transmisor.

En vez de pensar en un proceso como destino final, imaginaremos que cada máquina contiene un grupo de puntos abstractos de destino, llamados *puertos de protocolo*. Cada puerto de protocolo se identifica por medio de un número entero positivo. El sistema operativo local proporciona un mecanismo de interfaz que los procesos utilizan para especificar o accesar un puerto.

La mayor parte de los sistemas operativos proporciona un acceso síncrono a los puertos. Desde el punto de vista de un proceso en particular, el acceso síncrono significa que los cómputos se detienen durante una operación de acceso a puerto. Por ejemplo, si un proceso intenta extraer datos de un puerto antes de que llegue cualquier dato, el sistema operativo detiene (bloquea) temporalmente el proceso hasta que lleguen datos. Una vez que esto sucede, el sistema operativo pasa los datos al proceso y lo vuelve a iniciar. En general, los puertos tienen *memoria intermedia*, para que los datos que llegan antes de que un proceso esté listo para aceptarlos no se pierdan. Para lograr la colocación en memoria intermedia, el software de protocolo, localizado dentro del sistema operativo, coloca los paquetes que legan de un puerto de protocolo en particular en una cola de espera (finita) hasta que un proceso los extraiga.

Para comunicarse con un puerto externo, un transmisor necesita saber tanto la dirección IP de la máquina de destino como el número de puerto de protocolo del destino dentro de la máquina. Cada mensaje debe llevar el número del *puerto de destino* de la máquina a la que se envía, así como el número de *puerto de origen* de la máquina fuente a la que se deben direccionar las respuestas. Por lo tanto, es posible que cualquier proceso que recibe un mensaje conteste al transmisor.

## 12.3 Protocolo de datagrama de usuario

En el grupo de protocolos TCP/IP, el *Protocolo de datagrama de usuario o UDP* proporciona el mecanismo primario que utilizan los programas de aplicación para enviar datagramas a otros programas de aplicación. El UDP proporciona puertos de protocolo utilizados para distinguir entre muchos programas que se ejecutan en la misma máquina. Esto es, además de los datos, cada mensaje UDP contiene tanto el número de puerto de destino como el número de puerto de origen, haciendo posible que el software UDP en el destino entregue el mensaje al receptor correcto y que éste envíe una respuesta.

El UDP utiliza el Protocolo Internet subyacente para transportar un mensaje de una máquina a otra y proporciona la misma semántica de entrega de datagramas, sin conexión y no confiable que el IP. No emplea acuses de recibo para asegurarse de que llegan mensajes, no ordena los mensajes entrantes, ni proporciona retroalimentación para controlar la velocidad a la que fluye la información entre las máquinas. Por lo tanto, los mensajes UDP se pueden perder, duplicar o llegar sin orden.

Además, los paquetes pueden llegar más rápido de lo que el receptor los puede procesar. Podemos resumir que:

> *El protocolo de datagrama de usuario (UDP) proporciona un servicio de entrega sin conexión y no confiable, utilizando el IP para transportar mensajes entre máquinas. Emplea el IP para llevar mensajes, pero agrega la capacidad para distinguir entre varios destinos dentro de una computadora anfitrión.*

Un programa de aplicación que utiliza el UDP acepta toda la responsabilidad por el manejo de problemas de confiabilidad, incluyendo la pérdida, duplicación y retraso de los mensajes, la entrega fuera de orden y la pérdida de conectividad. Por desgracia, los programadores de aplicaciones a menudo olvidan estos problemas cuando diseñan software. Además, como los programadores a menudo prueban el software de red utilizando redes de área local, altamente confiables y de baja demora, el procedimiento de pruebas puede no evidenciar las fallas potenciales. Por lo tanto, muchos programas de aplicación que confían en el UDP trabajan bien en un ambiente local, pero fallan dramáticamente cuando se utilizan en una red de redes TCP/IP más grande.

## 12.4 Formato de los mensajes UDP

Cada mensaje UDP se conoce como *datagrama de usuario*. Conceptualmente, un datagrama de usuario consiste de dos partes: un encabezado UDP y un área de datos UDP. Como se muestra en la figura 12.1, el encabezado se divide en cuatro campos de 16 bits, que especifican el puerto desde el que se envió el mensaje, el puerto para el que se destina el mensaje, la longitud del mensaje y una suma de verificación UDP.

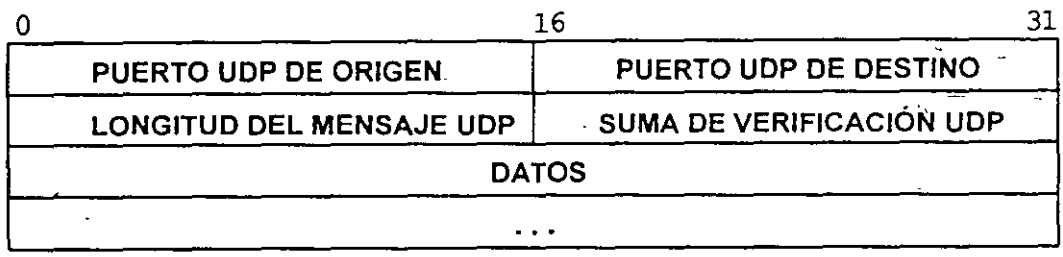



**Figura 12.1** Formato de los campos en un datagrama UDP.

Los campos *PUERTO DE ORIGEN* y *PUERTO DE DESTINO* contienen los números de puerto del protocolo UDP utilizados para el demultiplexado de datagramas entre los procesos que los esperan recibir. El *PUERTO DE ORIGEN* es opcional. Cuando se utiliza, especifica la parte a la que se deben enviar las respuestas, de lo contrario, puede tener valor de cero.

El campo de *LONGITUD* contiene un conteo de los octetos en el datagrama UDP, incluyendo el encabezado y los datos del usuario UDP. Por lo tanto, el valor mínimo para el campo *LONGITUD* es de ocho, que es la longitud del encabezado.

La suma de verificación UDP es opcional y no es necesario utilizarla; un valor de cero en el campo *SUMA DE VERIFICACIÓN* significa que la suma no se computó. Los diseñadores decidieron hacer opcional la suma de verificación a fin de permitir que las implantaciones operen con poco trabajo computacional cuando utilicen UDP en una red de área local altamente confiable. Sin embargo, recuerde que el IP no computa una suma de verificación de la porción de datos de un datagrama IP. Así que, la suma de verificación UDP proporciona la única manera de garantizar que los datos lleguen intactos, por lo que se debe utilizar.

Los principiantes, a menudo, se preguntan qué sucede con los mensajes UDP en los que la suma de verificación computada es cero. Un valor computacional de cero es posible debido a que el UDP utiliza el mismo algoritmo de suma de verificación que el IP: divide los datos en cantidades de 16 bits y computa el complemento de los unos de su suma de complementos de los unos. De manera sorprendente, el cero no es un problema debido a que la aritmética de los unos tiene dos representaciones para el cero: todos los bits como cero o todos los bits como uno. Cuando la suma de verificación computada es igual a cero, el UDP utiliza la representación con todos los bits como uno.

## 12.5 Pseudo-encabezado UDP

La suma de verificación UDP abarca más información de la que está presente en el datagrama UDP por sí solo. Para computar la suma de verificación, el UDP añade un *pseudo-encabezado* al datagrama UDP, adjunta un octeto de ceros para rellenar el datagrama y alcanzar exactamente un múltiplo de 16 bits, y computa la suma de verificación sobre todo el conjunto. El octeto utilizado como relleno y el pseudo-encabezado no se transmiten con el datagrama UDP, ni se incluyen en su longitud. Para computar una suma de verificación, el software primero almacena un cero en el campo de *SUMA DE VERIFICACIÓN*, luego, acumula una suma de complemento de 16 bits de todo el conjunto, incluyendo el pseudo-encabezado, el encabezado UDP y los datos del usuario.

El propósito de utilizar un pseudo-encabezado es para verificar que el datagrama UDP llegó a su destino correcto. La clave para entender el uso del pseudo-encabezado reside en darse cuenta de que el destino correcto consiste en una máquina específica y en un puerto de protocolo específico dentro de dicha máquina. Por sí mismo, el encabezado UDP sólo especifica el número de puerto de protocolo. Por lo tanto, para verificar un destino, el UDP en la máquina transmisora computa una suma de verificación que cubre tanto la dirección IP de destino como el datagrama UDP. En el destino final, el software UDP revisa la suma de verificación utilizando la dirección IP de destino, obtenida del encabezado del datagrama IP que transportó el mensaje UDP. Si la suma concuerda, debe ser verdad que el datagrama llegó al anfitrión de destino deseado, así como al puerto de protocolo correcto dentro del anfitrión.

El pseudo-encabezado utilizado en el cómputo de la suma de verificación UDP consiste en 12 octetos de datos, distribuidos como se muestra en la figura 12.2. Los campos en el pseudo-encabezado etiquetados como *DIRECCIÓN IP DE ORIGEN* y *DIRECCIÓN IP DE DESTINO*, contienen las direcciones IP que se utilizarán cuando se envíe el mensaje UDP. El campo *PROTO* contiene el código del tipo de protocolo IP (17 para UDP) y el campo *LONGITUD UDP* contiene la longitud del

datagrama UDP (sin incluir el pseudo-encabezado). Para revisar la suma de verificación, el receptor debe extraer estos campos del encabezado IP, ensamblarlos en el formato de pseudo-encabezado y recomputar la suma.

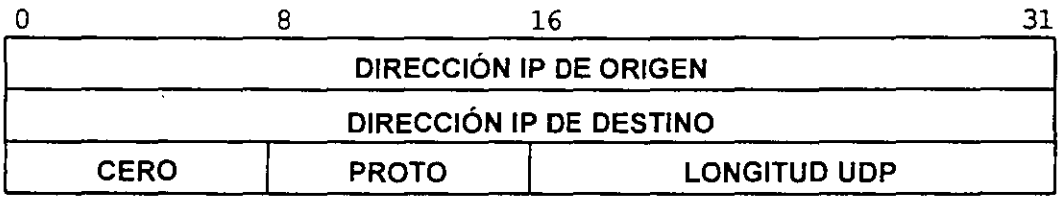



**Figura 12.2** Los 12 octetos de un pseudo-encabezado que se utilizan durante el cómputo de la suma de verificación UDP.

## 12.6 Encapsulación de UDP y estratificación por capas de protocolos

El UDP proporciona nuestro primer ejemplo de un protocolo de transporte. En el modelo de estratificación por capas del capítulo 11, el UDP reside sobre la capa del Protocolo Internet. Conceptualmente, los programas de aplicación accesan el UDP, que utiliza el IP para enviar y recibir datagramas como se muestra en la figura 12.3.

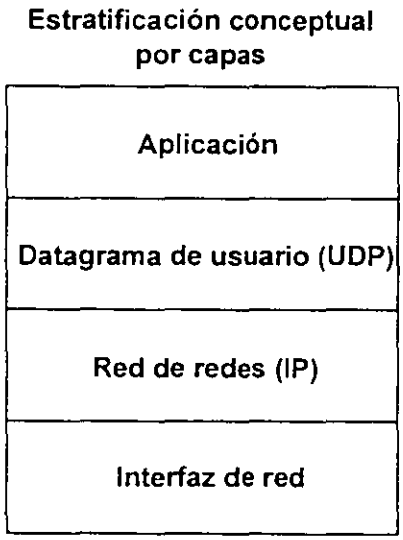



**Figura 12.3** Estratificación conceptual por capas de UDP entre programas de aplicación e IP.

Estratificar por capas el UDP por encima del IP significa que un mensaje UDP completo, incluyendo el encabezado UDP y los datos, se encapsula en un datagrama IP mientras viaja a través de una red de redes, tal como se muestra en la figura 12.4.

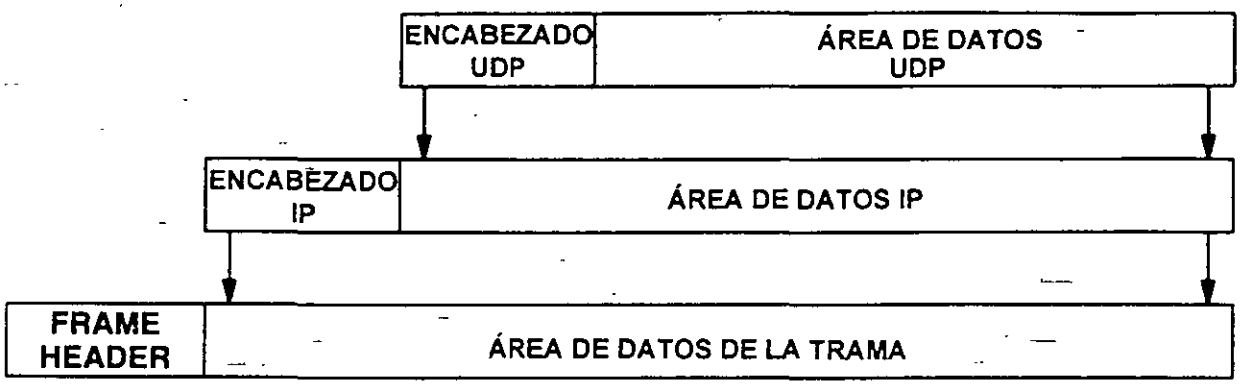



**Figura 12.4** Datagrama UDP encapsulado en un datagrama IP para su transmisión a través de una red de redes. El datagrama se encapsula en una trama cada vez que viaja a través de una red.

Para los protocolos que hemos examinado, la encapsulación significa que el UDP adjunta un encabezado a los datos que un usuario envía y lo pasa al IP. La capa IP adjunta un encabezado a lo que recibe del UDP. Y por último, la capa de interfaz de red introduce el datagrama en una trama antes de enviarlo de una máquina a otra. El formato de la trama depende de la tecnología subyacente de red. Por lo general, las tramas de red incluyen un encabezado adicional.

En la entrada, un paquete llega en la capa más baja del software de red y comienza su ascenso a través de capas sucesivamente más altas. Cada capa quita un encabezado antes de pasar el mensaje para que, en el momento en que el nivel más alto pasa los datos al proceso receptor, todos los encabezados se hayan removido. Por lo tanto, el encabezado exterior corresponde a la capa más baja de protocolo y el encabezado interior a la más alta de protocolo. Cuando se considera cómo se insertan y remueven los encabezados, es importante tener en cuenta el principio de la estratificación por capas. En lo particular, observe que este principio se aplica al UDP, así que el datagrama UDP que recibió el IP en la máquina de destino es idéntico al datagrama que el UDP pasó al IP en la máquina de origen. También, los datos que el UDP entrega a un proceso usuario en la máquina receptora serán los mismos que un proceso usuario pase al UDP en la máquina transmisora.

La división de funciones entre varias capas de protocolos es inflexible y clara:

> *La capa IP sólo es responsable de transferir datos entre un par de anfitriones dentro de una red de redes, mientras que la capa UDP solamente es responsable de diferenciar entre varias fuentes o destinos dentro de un anfitrión.*

Por lo tanto, sólo el encabezado IP identifica los anfitriones de origen y destino; sólo la capa UDP identifica los puertos de origen o destino dentro de un anfitrión.

## 12.7 Estratificación por capas y cómputo UDP de suma de verificación

Los lectores observadores habrán notado una contradicción aparente entre las reglas de la estratificación por capas y el cómputo de la suma de verificación UDP. Recuerde que la suma de verificación UDP incluye un pseudo-encabezado que tiene campos para las direcciones IP de origen y de destino. Se puede argüir que el usuario debe conocer la dirección IP de destino cuando envía un datagrama UDP y que éste la debe pasar a la capa UDP. Por lo tanto, la capa UDP puede obtener la dirección IP de destino sin interactuar con la capa IP. Sin embargo, la dirección IP de origen depende de la ruta que el IP seleccione para el datagrama, debido a que esta dirección identifica la interfaz de red sobre la que se transmite el datagrama. Por lo tanto, el UDP no puede conocer una dirección IP de origen a menos que interactúe con la capa IP.

Asumimos que el software UDP pide a la capa IP que compute la dirección IP de origen y (posiblemente) la de destino, las utiliza para construir un pseudo-encabezado, computa la suma de verificación, descarta el pseudo-encabezado y transfiere a la capa IP el datagrama UDP para su transmisión. Con un enfoque alternativo, que produce una mayor eficiencia, se logra que la capa UDP encapsule el datagrama UDP en un datagrama IP, obtenga del IP la dirección de origen, almacene las direcciones tanto de origen como de destino en los campos apropiados del encabezado del datagrama, compute la suma de verificación UDP y pase el datagrama IP a la capa IP, que sólo necesita llenar los campos restantes del encabezado IP.

¿La fuerte interacción entre el UDP y el IP viola nuestra premisa básica de que la estratificación por capas refleja la separación de funcionalidad? Sí. El UDP está fuertemente integrado al protocolo IP. Es claramente una transigencia de la separación pura, diseñado enteramente por razones prácticas. Deseamos pasar por alto la violación de estratificación por capas, ya que es imposible identificar plenamente un programa de aplicación de destino sin especificar la máquina de destino y porque queremos realizar, de manera eficaz, la transformación de direcciones utilizadas por el UDP y el IP. En uno de los ejercicios se examina este tema desde un punto de vista diferente y se pide al lector que considere si el UDP se debe separar de IP.

## 12.8 Multiplexado, demultiplexado y puertos de UDP

En el capítulo 11, vimos que el software a través de las capas de una jerarquía de protocolos debe multiplexar y demultiplexar muchos objetos en la capa siguiente. El software UDP proporciona otro ejemplo de multiplexado y demultiplexado. Acepta datagramas UDP de muchos programas de aplicación y los pasa a IP para su transmisión, también acepta datagramas entrantes UDP del IP y los transfiere al programa de aplicación apropiado.

Conceptualmente, todo el multiplexado y el demultiplexado entre el software UDP y los programas de aplicación ocurre a través del mecanismo de puerto. En la práctica, cada programa de aplicación debe negociar con el sistema operativo para obtener un puerto del protocolo y un número de puerto asociado, antes de poder enviar un datagrama UDP.[1] Una vez que se asigna el puerto,

[1] Por ahora, describiremos los puertos en forma abstracta; en el capítulo 20, se proporciona un ejemplo del sistema operativo que antiguamente se empleaba para crear y utilizar puertos.

ualquier datagrama que envíe el programa de aplicación a través de él, tendrá el número de puerto en el campo *PUERTO DE ORIGEN UDP*.

Mientras procesa la entrada, el UDP acepta datagramas entrantes del software IP y los demultiplexa, basándose en el puerto de destino UDP, como se muestra en la figura 12.5.

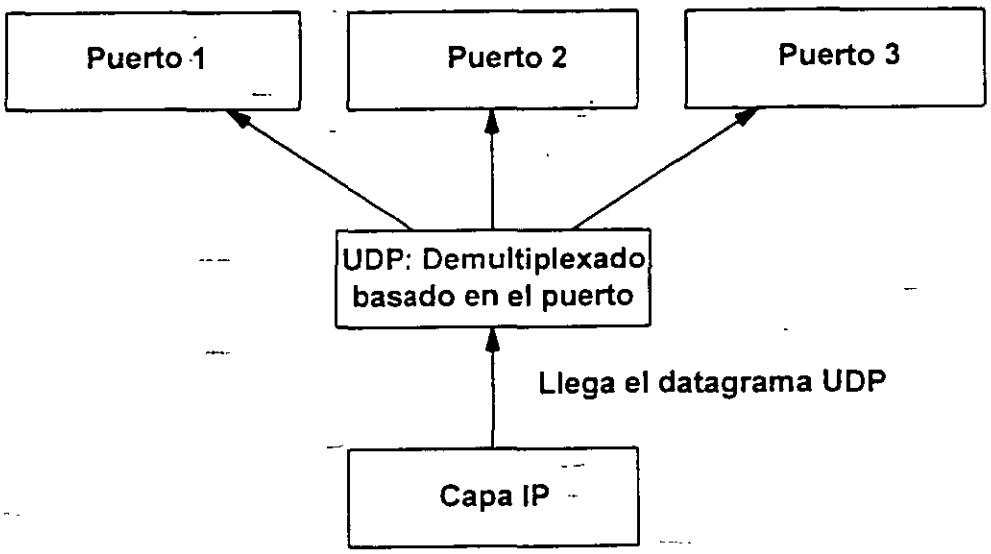



**Figura 12.5** Ejemplo del demultiplexado de una capa sobre el IP. El UDP utiliza el número de puerto UDP de destino para seleccionar el puerto apropiado de destino para los datagramas entrantes.

La forma más fácil de pensar en un puerto UDP es en una cola de espera. En la mayor parte de las implantaciones, cuando un programa de aplicación negocia con el sistema operativo la utilización de cierto puerto, el sistema operativo crea una cola de espera interna que puede almacenar los mensajes que lleguen. A menudo, la aplicación puede especificar o modificar el tamaño de la cola de espera. Cuando el UDP recibe un datagrama, verifica si el número de puerto de destino corresponde a uno de los puertos que están en uso. Si no, envía mensaje de error ICMP de *puerto no accesible* y descarta el datagrama. Si encuentra una correspondencia, el UDP pone en cola de espera el nuevo datagrama, en el puerto en que lo pueda accesar un programa de aplicación. Por supuesto, ocurre un error si el puerto se encuentra lleno y el UDP descarta el datagrama entrante.

## 12.9 Números de puerto UDP reservados y disponibles

¿Cómo se deben asignar los números de puerto de protocolo? El problema es importante ya que dos computadoras necesitan estar de acuerdo en los números de puerto antes de que puedan interoperar. Por ejemplo, cuando la computadora *A* quiere obtener un archivo de la computadora *B*, necesita saber qué puerto utiliza el programa de transferencia de archivos en la computadora *B*. Existen dos enfoques fundamentales para la asignación de puertos. El primero se vale de una autoridad central. Todos se ponen de acuerdo en permitir que una autoridad central asigne los números de puerto

conforme se necesitan y que publique la lista de todas las asignaciones. Entonces, todo el software se diseña de acuerdo con la lista. Este enfoque, a veces, se conoce como *enfoque universal* y las asignaciones de puerto especificadas por la autoridad se conocen como *asignaciones bien conocidas de puerto.*

El segundo enfoque para la asignación de puertos emplea la transformación dinámica. En este enfoque, los puertos no se conocen de manera global. En vez de eso, siempre que un programa necesita un puerto, el software de red le asigna uno. Para conocer la asignación actual de puerto en otra computadora, es necesario enviar una solicitud que pregunte algo así como "¿qué puerto está utilizando el servicio de transferencia de archivos?" La máquina objetivo responde al proporcionar el número de puerto correcto a utilizar.

Los diseñadores del TCP/IP adoptaron un enfoque híbrido que preasigna algunos números de puerto, pero que deja muchos de ellos disponibles para los sitios locales o programas de aplicación. Los números de puerto asignados comienzan con valores bajos y se extienden hacia arriba, dejando disponibles valores de números enteros altos para la asignación dinámica. En la tabla de la figura 12.6, se listan algunos de los números de puerto UDP actualmente asignados. La segunda columna contiene palabras clave asignadas como estándar de Internet y la tercera contiene palabras clave utilizadas en la mayor parte de los sistemas UNIX.

| Decimal | Palabra clave | Palabra clave UNIX | Descripción |
|---|---|---|---|
| 0 | - | - | Reservado |
| 7 | ECHO | echo | Eco |
| 9 | DISCARD | discard | Descartar |
| 11 | USERS | systat | Usuarios Activos |
| 13 | DAYTIME | daytime | Hora del día |
| 15 | - | netstat | Quién está ahí o NETSTAT |
| 17 | QUOTE | qotd | Cita del día |
| 19 | CHARGEN | chargen | Generador de caracteres |
| 37 | TIME | time | Hora |
| 42 | NAMESERVER | name | Servidor de nombres de anfitriones |
| 43 | NICNAME | whois | Quién es |
| 53 | DOMAIN | nameserver | Servidor de nombres de dominios |
| 67 | BOOTPS | bootps | Servidor de protocolo bootstrap |
| 68 | BOOTPC | bootpc | Cliente de protocolo bootstrap |
| 69 | TFTP | tftp | Transferencia trivial de archivos |
| 111 | SUNRPC | sunrpc | RPC de Sun Microsystems |
| 123 | NTP | ntp | Protocolo de tiempo de red |
| 161 | | snmp | monitor de red SNMP |
| 162 | | snmp-trap | interrupciones SNMP |
| 512 | | biff | comsat UNIX |
| 513 | | who | rwho daemon UNIX |
| 514 | | syslog | conexión de sistema |
| 525 | | timed | daemon de hora |

**Figura 12.6** Ejemplo ilustrativo de los puertos UDP actualmente asignados, que muestra la palabra clave estándar y su equivalente UNIX; la lista no es completa. En lo posible, otros protocolos de transporte que ofrecen los mismos servicios utilizan los mismos números de puerto que el UDP.

## 12.10 Resumen

La mayor parte de los sistemas de computadoras permite que varios programas de aplicaciones se ejecuten al mismo tiempo. Utilizando la jerga de los sistemas operativos, nos referimos a dicho programa en ejecución como un *proceso*. El protocolo de datagrama de usuario, UDP, distingue entre muchos procesos dentro de una máquina al permitir que los transmisores y los receptores agreguen números enteros de 16 bits, llamados números de puerto de protocolo, a cada mensaje UDP. Los números de puerto identifican el origen y el destino. Algunos números de puerto UDP, llamados *bien conocidos*, se asignan y mencionan permanentemente a través de Internet (por ejemplo, el puerto 69 está reservado para que lo utilice el protocolo simple de transferencia de archivos, *TFTP*, que se describe en el capítulo 24). Otros números de puerto están disponibles para que los utilicen programas arbitrarios de aplicación.

El UDP es un protocolo sencillo en el sentido de que no aumenta de manera significativa la semántica del IP. Sólo proporciona a los programas de aplicación la capacidad para comunicarse, mediante el uso del servicio de entrega de paquetes, sin conexión y no confiable. Por lo tanto, los mensajes UDP se pueden perder, duplicar, retrasar o entregar en desorden; el programa de aplicación que utiliza el UDP debe resolver estos problemas. Muchos programas que emplean el UDP no funcionan correctamente en una red de redes debido a que no manejan estas condiciones.

En el esquema de estratificación por capas de protocolo, el UDP reside en la capa de transporte, arriba de la capa del Protocolo Internet y bajo la capa de aplicación. Conceptualmente, la capa de transporte es independiente de la capa Internet, pero en la práctica interactúan estrechamente. La suma e verificación UDP incluye las direcciones IP de origen y destino, lo cual significa que el software UDP debe interactuar con el software IP para encontrar direcciones antes de enviar los datagramas.

## PARA CONOCER MÁS

Tanenbaum (1981) hace una comparación tutorial de los modelos de comunicación de datagrama y de circuito virtual. Ball *et. al.* (1979) describe los sistemas basados en mensajes sin tratar el protocolo de mensajes. El protocolo UDP que se describió aquí es un estándar para TCP/IP y lo define Postel (RFC 768).

## EJERCICIOS

**12.1** Utilice el UDP en su ambiente local. Mida la velocidad promedio de transferencia con mensajes de 256, 512, 1024, 2048, 4096, y 8192 octetos. ¿Puede explicar los resultados? Pista: ¿cuál es el MTU de su red?

**12.2** ¿Por qué la suma de verificación UDP está separada de la IP? ¿Objetaría un protocolo que utilizara una sola suma de verificación para todo el datagrama IP, incluyendo el mensaje UDP?

**12.3** No utilizar sumas de verificación puede ser peligroso. Explique cómo una sola difusión corrompida de paquetes ARP, realizada por la máquina *P*, puede ocasionar que sea imposible accesar otra máquina, *Q*.

¿Se debería incorporar al IP la noción de muchos destinos identificados por puertos de protocolos? ¿Por qué?

*Registro de Nombres.* Suponga que quiere permitir que pares de programas de aplicación establezcan comunicación con el UDP, pero no les quiere asignar números fijos de puerto UDP. En vez de eso, le gustaría que las correspondencias potenciales se identificaran por medio de una cadena de 64 o menos caracteres. Por lo tanto, un programa en la máquina *A* podría querer comunicarse con el programa de "id especial curiosamente larga" en la máquina *B* (puede asumir que un proceso siempre conoce la dirección IP del anfitrión con el que se quiere comunicar). Mientras tanto, un proceso en la máquina *C* se quiere comunicar "el programa id propio de comer" en la máquina *A*. Demuestre que solamente tiene que asignar un puerto UDP para hacer posible dicha comunicación al diseñar software en cada máquina que permita: (a) que un proceso local escoja una ID no utilizada de puerto UDP sobre la cual comunicarse, (b) que un proceso local registre el nombre de 64 caracteres al que responde, y (c) que un proceso exterior utilice el UDP para establecer comunicación utilizando solamente el nombre de 64 caracteres y la dirección de red de redes de destino.

Ponga en práctica el software de registro de nombres del ejercicio anterior.

¿Cuál es la principal ventaja de utilizar números preasignados de puerto UDP? ¿La principal desventaja?

¿Cuál es la principal ventaja de emplear puertos de protocolo en vez de identificadores de proceso para especificar el destino dentro de una máquina?

El UDP proporciona comunicación no confiable de datagramas debido a que no garantiza la entrega del mensaje. Vislumbre un protocolo confiable de datagramas que utilice terminación de tiempo y acuses de recibo para garantizar la entrega. ¿Qué tanto retraso y trabajo adicional provoca la confiabilidad?

Envíe datagramas IP a través de una red de área amplia y mida el porcentaje de datagramas perdidos y reordenados. ¿El resultado depende de la hora del día? ¿De la carga de la red?

# 13

# Servicio de transporte de flujo confiable (TCP)

## 13.1 Introducción

En los capítulos anteriores exploramos el servicio de entrega de paquetes sin conexión y no confiable, que forma la base para toda la comunicación en red de redes, así como el protocolo IP que lo define. En este capítulo, se introduce el segundo servicio más importante y mejor conocido de nivel de red, la entrega de flujo confiable, así como el *Protocolo de Control de Transmisión (TCP)* que lo define. Veremos que el TCP añade una funcionalidad substancial a los protocolos que ya hemos analizado, pero también veremos que su implantación es substancialmente más compleja.

Aunque aquí se presenta el TCP como parte del grupo de protocolos Internet TCP/IP, es un protocolo independiente de propósitos generales que se puede adaptar para utilizarlo con otros sistemas de entrega. Por ejemplo, debido a que el TCP asume muy poco sobre la red subyacente, es posible utilizarlo en una sola red como Ethernet, así como en una red de redes compleja. De hecho, el TCP es tan popular, que uno de los protocolos para sistemas abiertos de la Organización Internacional para la Estandarización, TP-4, se derivó de él.

## 13.2 Necesidad de la entrega de flujo

En el nivel más bajo, las redes de comunicación por computadora proporcionan una entrega de paquetes no confiable. Los paquetes se pueden perder o destruir cuando los errores de transmisión interfieren con los datos, cuando falla el hardware de red o cuando las redes se sobrecargan demasia-

do. Las redes que rutean dinámicamente los paquetes pueden entregarlos en desorden, con retraso o duplicados. Además, las tecnologías subyacentes de red pueden dictar un tamaño óptimo de paquete o formular otras obligaciones necesarias para lograr velocidades eficientes de transmisión.

En el nivel más alto, los programas de aplicación a menudo necesitan enviar grandes volúmenes de datos de una computadora a otra. Utilizar un sistema de entrega sin conexión y no confiable para las transferencias de gran volumen se vuelve tedioso, molesto y requiere que los programadores incorporen, en cada programa de aplicación, la detección y solución de errores. Debido a que es difícil diseñar, entender o modificar el software que proporciona confiabilidad, muy pocos programadores de aplicaciones tienen los antecedentes técnicos necesarios. Como consecuencia, una meta de la investigación de protocolos de red ha sido encontrar soluciones de propósito general para el problema de proporcionar una entrega de flujo confiable, lo que posibilita a los expertos a construir una sola instancia de software de protocolos de flujo que utilicen todos los programas de aplicación. Tener un solo protocolo de propósito general es útil para aislar los programas de aplicación de los detalles del trabajo con redes y permite la definición de una interfaz uniforme para el servicio de transferencia de flujo.

## 13.3 Características del servicio de entrega confiable

La interfaz entre los programas de aplicación y el servicio TCP/IP de entrega confiable se puede caracterizar por cinco funciones:

- *Orientación de flujo.* Cuando dos programas de aplicación (procesos de usuario) transfieren grandes volúmenes de datos, pensamos en los datos como un *flujo* de bits, divididos en *octetos* de 8 bits, que informalmente se conocen como *bytes*. El servicio de entrega de flujo en la máquina de destino pasa al receptor exactamente la misma secuencia de octetos que le pasa el transmisor en la máquina de origen.
- *Conexión de circuito virtual.* La transferencia de flujo es análoga a realizar una llamada telefónica. Antes de poder empezar la transferencia, los programas de aplicación, transmisor y receptor interactúan con sus respectivos sistemas operativos, informándose de la necesidad de realizar una transferencia de flujo. Conceptualmente, una aplicación realiza una "llamada" que la otra tiene que aceptar. Los módulos de software de protocolo en los dos sistemas operativos se comunican al enviarse mensajes a través de una red de redes, verificando que la transferencia esté autorizada y que los dos extremos estén listos. Una vez que se establecen todos los detalles, los módulos de protocolo informan a los programas de aplicación que se estableció una *conexión* y que la transferencia puede comenzar. Durante la transferencia, el software de protocolo en las dos máquinas continúa comunicándose para verificar que los datos se reciban correctamente. Si la comunicación no se logra por cualquier motivo (por ejemplo, debido a que falle el hardware de red a lo largo del camino entre las máquinas), ambas máquinas detectarán la falla y la reportarán a los programas apropiados de aplicación. Utilizamos el término *circuito virtual* para describir dichas conexiones porque aunque los programas de aplicación visualizan la conexión como un circuito dedicado de hardware, la confiabilidad que se proporciona depende del servicio de entrega de flujo.
- *Transferencia con memoria intermedia.* Los programas de aplicación envían un flujo de datos a través del circuito virtual pasando repetidamente octetos de datos al software de protocolo. Cuando transfieren datos, cada aplicación utiliza piezas del tamaño que encuentre adecuado, que

pueden ser tan pequeñas como un octeto. En el extremo receptor, el software de protocolo entrega octetos del flujo de datos en el mismo orden en que se enviaron, poniéndolos a disposición del programa de aplicación receptor tan pronto como se reciben y verifican. El software de protocolo puede dividir el flujo en paquetes, independientemente de las piezas que transfiera el programa de aplicación. Para hacer eficiente la transferencia y minimizar el tráfico de red, las implantaciones por lo general recolectan datos suficientes de un flujo para llenar un datagrama razonablemente largo antes de transmitirlo a través de una red de redes. Por lo tanto, inclusive si el programa de aplicación genera el flujo un octeto a la vez, la transferencia a través de una red de redes puede ser sumamente eficiente. De forma similar, si el programa de aplicación genera bloques de datos muy largos, el software de protocolo puede dividir cada bloque en partes más pequeñas para su transmisión.

Para aplicaciones en las que los datos se deben entregar aunque no se llene una memoria intermedia, el servicio de flujo proporciona un mecanismo de *empuje* (*push*) que las aplicaciones utilizan para forzar una transferencia. En el extremo transmisor, un empuje obliga al software de protocolo a transferir todos los datos generados sin tener que esperar a que se llene una memoria intermedia. Cuando llega al extremo receptor, el empuje hace que el TCP ponga los datos a disposición de la aplicación sin demora. Sin embargo, el lector debe notar que la función de empuje sólo garantiza que los datos se transferirán; no proporciona fronteras de registro. Por lo tanto, aun cuando la entrega es forzada, el software de protocolo puede dividir el flujo en formas inesperadas.

- *Flujo no estructurado.* Es importante entender que el servicio de flujo TCP/IP no está obligado a formar flujos estructurados de datos. Por ejemplo, no existe forma para que una aplicación de nómina haga que un servicio de flujo marque fronteras entre los registros de empleado o que identifique el contenido del flujo como datos de nómina. Los programas de aplicación que utilizan el servicio de flujo deben entender el contenido del flujo y ponerse de acuerdo sobre su formato antes de iniciar una conexión.
- *Conexión Full Duplex.* Las conexiones proporcionadas por el servicio de flujo TCP/IP permiten la transferencia concurrente en ambas direcciones. Dichas conexiones se conocen como *full duplex*. Desde el punto de vista de un proceso de aplicación, una conexión full duplex consiste en dos flujos independientes que se mueven en direcciones opuestas, sin ninguna interacción aparente. El servicio de flujo permite que un proceso de aplicación termine el flujo en una dirección mientras los datos continúan moviéndose en la otra dirección, haciendo que la conexión sea *half duplex*. La ventaja de una conexión full duplex es que el software subyacente de protocolo puede enviar en datagramas información de control de flujo al origen, llevando datos en la dirección opuesta. Este procedimiento de carga, transporte y descarga reduce el tráfico en la red.

## 13.4 Proporcionando confiabilidad

Hemos dicho que el servicio de entrega de flujo confiable garantiza la entrega de los datos enviados de una máquina a otra sin pérdida o duplicación. Surge la pregunta: "¿cómo puede el software de protocolo proporcionar una transferencia confiable si el sistema subyacente de comunicación sólo ofrece una entrega no confiable de paquetes?" La respuesta es complicada, pero la mayor parte de los protocolos confiables utilizan una técnica fundamental conocida como *acuse de recibo positivo con retransmisión*. La técnica requiere que un receptor se comunique con el origen y le en-

vie un mensaje de *acuse de recibo (ACK)* conforme recibe los datos. El transmisor guarda un registro de cada paquete que envía y espera un acuse de recibo antes de enviar el siguiente paquete. El transmisor también arranca un temporizador cuando envía un paquete y lo *retransmite* si dicho temporizador expira antes de que llegue un acuse de recibo.

En la figura 13.1 se muestra cómo transfiere datos el protocolo más sencillo de acuse de recibo positivo.

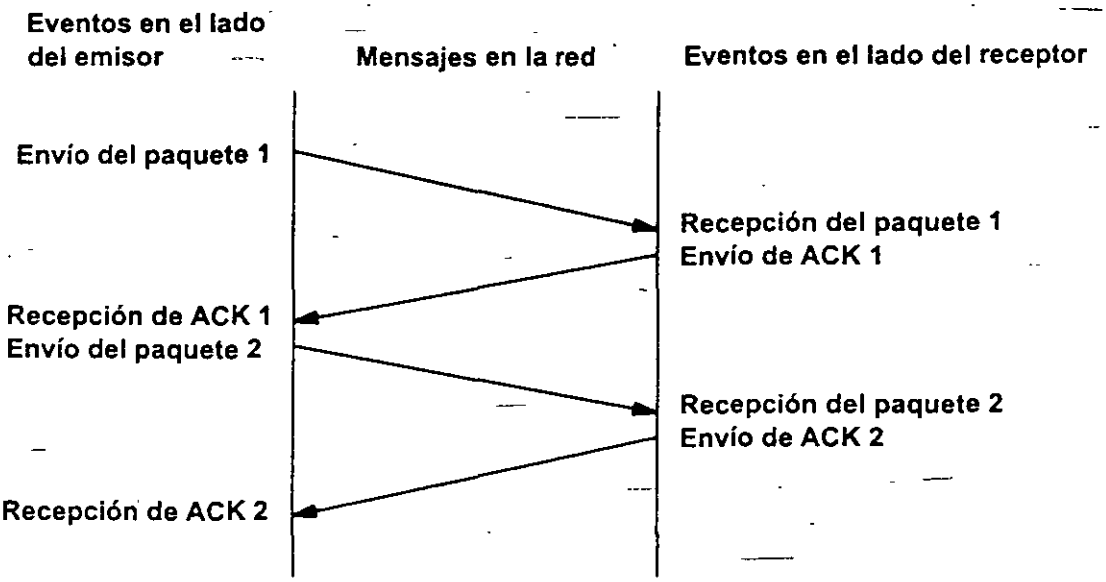



**Figura 13.1** Un protocolo que se vale de reconocimientos o acuses de recibo positivos, con retransmisión, en la cual el emisor espera un acuse de recibo para cada paquete enviado. La distancia vertical bajo la figura representa el incremento en el tiempo y las líneas que cruzan en diagonal representan la transmisión de paquetes de red.

En la figura, los eventos en el transmisor y receptor se muestran a la izquierda y derecha, respectivamente. Cada línea diagonal que cruza por el centro muestra la transferencia de un mensaje a través de la red.

En la figura 13.2 se utiliza el mismo diagrama de formato que en la figura 13.1 para mostrar qué sucede cuando se pierde o corrompe un paquete. El transmisor arranca un temporizador después de enviar el paquete. Cuando termina el tiempo, el transmisor asume que el paquete se perdió y lo vuelve a enviar.

El problema final de confiabilidad surge cuando un sistema subyacente de entrega de paquetes los duplica. Los duplicados también pueden surgir cuando las redes tienen grandes retrasos que provocan la retransmisión prematura. La solución de la duplicación requiere acciones cuidadosas ya que tanto los paquetes como los acuses de recibo se pueden duplicar. Por lo general, los protocolos confiables detectan los paquetes duplicados al asignar a cada uno un número de secuencia y al obligar al receptor a recordar qué números de secuencia recibe. Para evitar la confusión causada por acuses de recibo retrasados o duplicados, los protocolos de acuses de recibo positivos envían los números de secuencia dentro de los acuses, para que el receptor pueda asociar correctamen te los acuses de recibo con los paquetes.

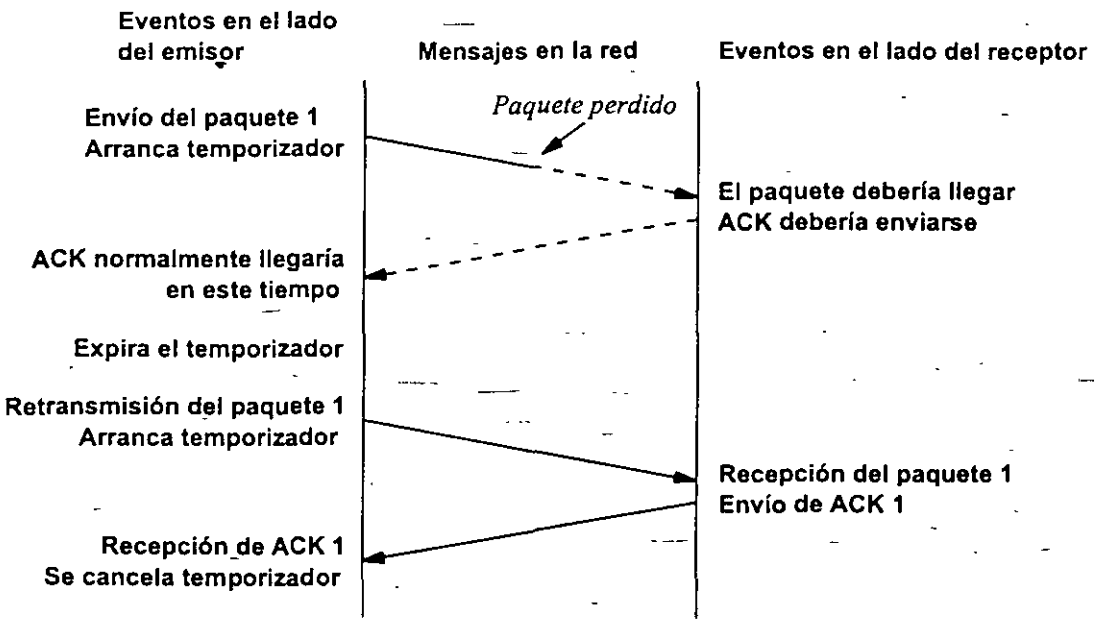



**Figura 13.2** Tiempo excedido y retransmisión que ocurre cuando un paquete se pierde. La línea punteada muestra el tiempo que podría ocuparse para la transmisión de un paquete y su acuse de recibo, si no se perdiera el paquete.

## 13.5 La idea detrás de las ventanas deslizables

Antes de examinar el servicio de flujo TCP, necesitamos explorar un concepto adicional que sirve de base para la transmisión de flujo. Este concepto, conocido como *ventana deslizable*, hace que la transmisión de flujo sea eficiente. Para entender lo que motiva a utilizar ventanas deslizables, recuerde la secuencia de eventos que se muestran en la figura 13.1. A fin de lograr la confiabilidad, el transmisor envía un paquete y espera un acuse de recibo antes de enviar otro. Como se muestra en la figura 13.1, los datos sólo fluyen entre las máquinas en una dirección a la vez, inclusive si la red tiene capacidad para comunicación simultánea en ambas direcciones. La red estará del todo ociosa durante el tiempo en que las máquinas retrasen sus respuestas (por ejemplo, mientras las máquinas computan rutas o sumas de verificación). Si nos imaginamos una red con altos retrasos en la transmisión, el problema es evidente:

> *Un protocolo simple de acuses de recibo positivos ocupa una cantidad sustancial de ancho de banda de red debido a que debe retrasar el envío de un nuevo paquete hasta que reciba un acuse de recibo del paquete anterior.*

La técnica de ventana deslizable es una forma más compleja de acuse de recibo positivo y retransmisión que el sencillo método mencionado antes. Los protocolos de ventana deslizable utilizan el ancho de banda de red de mejor forma, ya que permiten que el transmisor envíe varios paquetes sin esperar un acuse de recibo. La manera más fácil de visualizar la operación de ventana

deslizable es pensar en una secuencia de paquetes que se transmitirán como se muestra en la figura 13.3. El protocolo coloca una *ventana* pequeña y de tamaño fijo en la secuencia, y transmite todos los paquetes que residan dentro de la ventana.

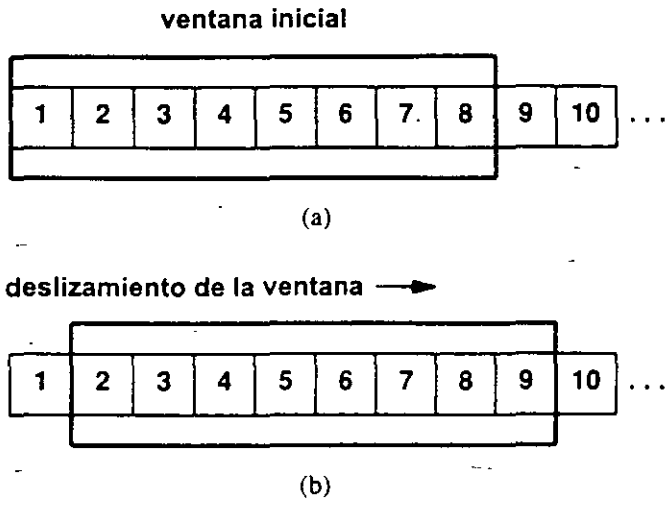



**Figura 13.3** (a) Un protocolo de ventana deslizante con ocho paquetes en la ventana, y (b) La ventana que se desliza hacia el paquete *9* puede enviarse cuando se recibe un acuse de recibo del paquete *1*. Unicamente se retransmiten los paquetes sin acuse de recibo.

Decimos que un paquete es *unacknowledged* o sin acuse de recibo[1] si se transmitió pero no se recibió ningún acuse de recibo. Técnicamente, el número de paquetes sin acuse de recibo en un tiempo determinado depende del *tamaño de la ventana* y está limitado a un número pequeño y fijo. Por ejemplo, en un protocolo de ventana deslizable con un tamaño de ventana de 8, se permite al transmisor enviar 8 paquetes antes de recibir un acuse de recibo.

Como se muestra en la figura 13.3, una vez que el transmisor recibe un acuse de recibo para el primer paquete dentro de la ventana, "mueve" la misma y envía el siguiente paquete. La ventana continuará moviéndose en tanto se reciban acuses de recibo.

El desempeño de los protocolos de ventana deslizable depende del tamaño de la ventana y de la velocidad en que la red acepta paquetes. En la figura 13.4, se muestra un ejemplo de la operación de un protocolo de ventana deslizable cuando se envían tres paquetes. Nótese que el transmisor los envía antes de recibir cualquier acuse de recibo.

Con un tamaño de ventana de *1*, un protocolo de ventana deslizable sería idéntico a un protocolo simple de acuse de recibo positivo. Al aumentar el tamaño de la ventana, es posible eliminar completamente el tiempo ocioso de la red. Esto es, en una situación estable, el transmisor puede enviar paquetes tan rápido como la red los pueda transferir. El punto principal es que:

*Como un protocolo de ventana deslizable bien establecido mantiene la red completamente saturada de paquetes, con él se obtiene una generación de salida substancialmente más alta que con un protocolo simple de acuse de recibo positivo.*

Conceptualmente, un protocolo de ventana deslizable siempre recuerda qué paquetes tienen acuse de recibo y mantiene un temporizador separado para cada paquete sin acuse de recibo. Si se pierde un paquete, el temporizador concluye y el transmisor reenvía el paquete. Cuando el receptor desliza su ventana, mueve hacia atrás todos los paquetes con acuse. En el extremo receptor, el software de protocolo mantiene una ventana análoga, que acepta y acusa como recibidos los paquetes conforme llegan. Por lo tanto, la ventana divide la secuencia de paquetes en tres partes: los paquetes a la izquierda de la ventana se transmitieron, recibieron y acusaron exitosamente; los paquetes a la derecha no se han transmitido; y los paquetes que quedan dentro de la ventana están en proceso de transmisión. El paquete con menor número en la ventana es el primer paquete en la secuencia para el que no se ha hecho un acuse de recibo.

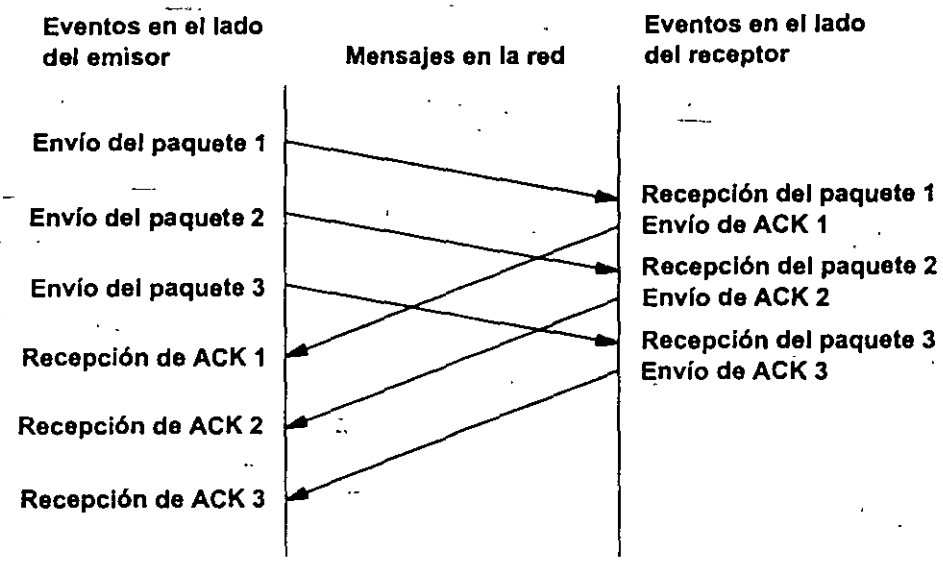



**Figura 13.4** Ejemplo de tres paquetes transmitidos mediante un protocolo de ventana deslizante. El concepto clave es que el emisor puede transmitir todos los paquetes de la ventana sin esperar un acuse de recibo.

## 13.6 Protocolo de control de transmisión

Ya que entendimos el principio de las ventanas deslizables, podemos examinar el servicio de flujo confiable proporcionado por el grupo de protocolos TCP/IP de Internet. El servicio lo define el *Protocolo de Control de Transmisión* o TCP. El servicio de flujo confiable es tan importante que todo el grupo de protocolos se conoce como TCP/IP. Es importante entender que:

*El TCP es un protocolo de comunicación, no una pieza de software.*

La diferencia entre un protocolo y el software que lo implementa es análoga a la diferencia entre la definición de un lenguaje de programación y un compilador. Al igual que en el mundo de los lenguajes de programación, la distinción entre definición e implantación a veces es imprecisa. Las personas encuentran software TCP mucho más frecuentemente que la especificación de protocolo, así que es natural pensar en una implantación en particular como en el estándar. No obstante, el lector debe tratar de distinguir entre las dos.

¿Qué proporciona el TCP exactamente? El TCP es complejo, por lo que no hay una respuesta sencilla. El protocolo especifica el formato de datos y los acuses de recibo que intercambian dos computadoras para lograr una transferencia confiable, así como los procedimientos que la computadora utiliza para asegurarse de que los datos lleguen de manera correcta. También, especifica cómo el software TCP distingue el correcto entre muchos destinos en una misma máquina, y cómo las máquinas en comunicación resuelven errores como la pérdida o duplicación de paquetes. El protocolo también especifica cómo dos computadoras inician una transferencia de flujo TCP y cómo se ponen de acuerdo cuando se completa.

Asimismo, es importante entender lo que el protocolo no incluye. Aunque la especificación TCP describe cómo utilizan el TCP los programas de aplicación en términos generales, no aclara los detalles de la interfaz entre un programa de aplicación y el TCP. Esto es, la documentación del protocolo sólo analiza las operaciones que el TCP proporciona; no especifica los procedimientos exactos que los programas de aplicación invocan para accesar estas operaciones. La razón para no especificar la interfaz del programa de aplicación es la flexibilidad. En particular, debido a que los programadores por lo general implantan el TCP en el sistema operativo de una computadora, necesitan emplear la interfaz que proporciona el sistema operativo, sea cual sea. Permitir que el implantador tenga flexibilidad hace posible tener una sola especificación para el TCP que pueda utilizarse para diseñar software en una gran variedad de máquinas.

Debido a que TCP asume muy poco sobre el sistema subyacente de comunicación, TCP se puede utilizar con una gran variedad de sistemas de entrega de paquetes, incluyendo el servicio de entrega de datagramas IP. Por ejemplo, el TCP puede implantarse para utilizar líneas de marcación telefónica, una red de área local, una red de fibra óptica de alta velocidad o una red de largo recorrido y baja velocidad. De hecho, la gran variedad de sistemas de entrega que puede utilizar el TCP es una de sus ventajas.

## 13.7 Puertos, conexiones y puntos extremos

Al igual que el Protocolo de Datagrama de Usuario (UDP) que vimos en el capítulo 12, el TCP reside sobre el IP en el esquema de estratificación por capas de protocolos. En la figura 13.5 se muestra la organización conceptual. El TCP permite que varios programas de aplicación en una máquina se comuniquen de manera concurrente y realiza el demultiplexado del tráfico TCP entrante entre los programas de aplicación. Al igual que el Protocolo de Datagrama de Usuario (UDP), el TCP utiliza números de *puerto de protocolo* para identificar el destino final dentro de una máquina. Cada puerto tiene asignado un número entero pequeño utilizado para identificarlo.[2]

[2] Aun cuando el TCP y el UDP se valen de identificadores enteros de puerto que comienzan en 1 para identificar puertos, no hay confusión entre ellos ya que un datagrama IP entrante identifica el protocolo en uso así como en número de

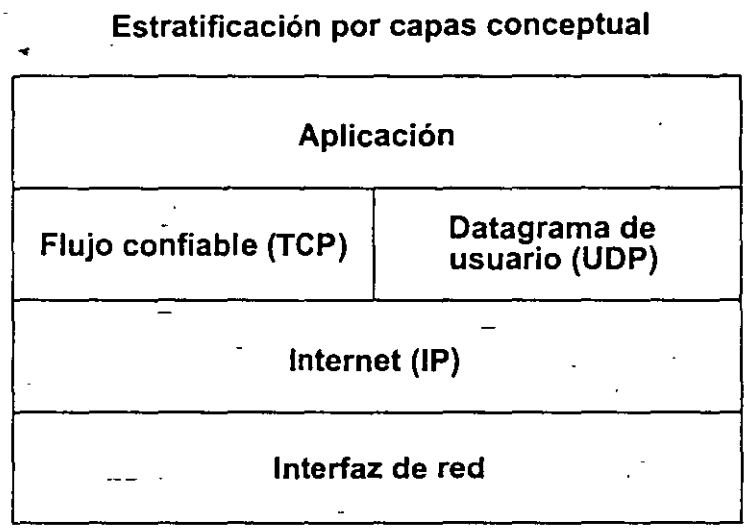



**Figura 13.5** La estratificación por capas conceptual del UDP y el TCP sobre el IP. El TCP proporciona un servicio de flujo confiable, mientras que el UDP proporciona un servicio de entrega de datagramas no confiable. Los programas de aplicación emplean ambos.

Cuando tratamos los puertos UDP, dijimos que se pensara de cada puerto como en una cola de salida en la que el software de protocolo coloca los datagramas entrantes. Sin embargo, los puertos TCP son mucho más complejos, ya que un número de puerto no corresponde a un solo objeto. De hecho, el TCP se diseñó según la *abstracción de conexión*, en la que los objetos que se van a identificar son conexiones de circuito virtual, no puertos individuales. Entender que el TCP utiliza la noción de conexiones es crucial, ya que nos ayuda a explicar el significado y la utilización de los números de puerto TCP:

> *El TCP utiliza la conexión, no el puerto de protocolo, como su abstracción fundamental; las conexiones se identifican por medio de un par de puntos extremos.*

¿Qué son exactamente los "puntos extremos" de una conexión? Hemos dicho que una conexión consiste en un circuito virtual entre dos programas de aplicación, por lo que puede ser natural asumir que un programa de aplicación sirve como el "punto extremo" de la conexión. Sin embargo, no es así. El TCP define que *un punto* extremo es un par de números enteros (*anfitrión, puerto*), en donde *anfitrión* es la dirección IP de un anfitrión y *puerto* es un puerto TCP en dicho anfitrión. Por ejemplo, el punto extremo (*128.10.2.3, 25*), se refiere al puerto TCP *25* en la máquina con dirección IP *128.10.2.3*.

Ahora que ya explicamos los puntos extremos, será fácil entender las conexiones. Recuerde que una conexión se define por sus dos puntos extremos. Por lo tanto, si existe una conexión entre la máquina (*18.26.0.36*) en el MIT y la máquina (*128.10.2.3*) en la Universidad de Purdue, la conexión se definiría por los puntos extremos:

(*18.26.0.36, 1069*) y (*128.10.2.3, 25*).

Mientras tanto, otra conexión se puede dar entre la máquina (*128.9.0.32*) en el Instituto de Ciencias de la Información y la misma máquina en Purdue, conexión identificada por sus puntos extremos:

(*128.9.0.32, 1184*) y (*128.10.2.3, 53*).

Hasta ahora, nuestros ejemplos de conexiones han sido directos, ya que los puertos utilizados en todos los puntos extremos han sido únicos. Sin embargo, la abstracción de conexión permite que varias conexiones compartan un punto extremo. Por ejemplo, podemos agregar otra conexión a las dos arriba mencionadas entre la máquina (*128.2.254.139*) en la CMU y la máquina en Purdue:

(*128.2.254.139, 1184*) y (*128.10.2.3, 53*).

Puede parecer extraño que dos conexiones utilicen al mismo tiempo el puerto TCP *53* en la máquina 128.10.2.3, pero no hay ambigüedad. Debido a que el TCP asocia los mensajes entrantes con una conexión en vez de hacerlo con un puerto de protocolo, utiliza ambos puntos extremos para identificar la conexión apropiada. La idea importante que se debe recordar es:

> *Como el TCP identifica una conexión por medio de un par de puntos extremos, varias conexiones en la misma máquina pueden compartir un número de puerto TCP.*

Desde el punto de vista de un programador, la abstracción de comunicación es importante. Significa que un programador puede diseñar un programa que proporcione servicio concurrente a varias conexiones al mismo tiempo, sin necesitar números únicos de puerto local para cada una. Por ejemplo, la mayor parte de los sistemas proporciona acceso concurrente a su servicio de correo electrónico, lo cual permite que varias computadoras les envíen correo electrónico de manera concurrente. Debido a que el programa que acepta correo entrante utiliza el TCP para comunicarse, sólo necesitan emplear un puerto TCP local, aun cuando permita que varias conexiones se realicen en forma concurrente.

## 13.8 Aperturas pasivas y activas

A diferencia del UDP, el TCP es un protocolo orientado a la conexión, el cual requiere que ambos puntos extremos estén de acuerdo en participar. Esto es, antes de que el tráfico TCP pueda pasar a través de una red de redes, los programas de aplicación en ambos extremos de la conexión deben estar de acuerdo en que desean dicha conexión. Para hacerlo, el programa de aplicación en un extremo realiza una función de *apertura pasiva* al contactar su sistema operativo e indicar que aceptará una conexión entrante. En ese momento, el sistema operativo asigna un número de puerto TCP a su extremo de la conexión. El programa de aplicación en el otro extremo debe contactar a su sistema operativo mediante una solicitud de *apertura activa* para establecer una conexión. Los dos módulos de software TCP se comunican para establecer y llevar a cabo la conexión. Una vez que

se crea ésta, los programas de aplicación pueden comenzar a transferir datos; los módulos de software TCP en cada extremo intercambian mensajes que garantizan la entrega confiable. Regresaremos a los detalles del establecimiento de conexiones después de examinar el formato de mensaje TCP.

## 13.9 Segmentos, flujos y números de secuencia

El TCP visualiza el flujo de datos como una secuencia de octetos (o bytes) que divide en *segmentos* para su transmisión. Por lo general, cada segmento viaja a través de una red de redes como un solo datagrama IP.

El TCP utiliza un mecanismo especializado de ventana deslizable para solucionar dos problemas importantes: la transmisión eficiente y el control de flujo. Al igual que el protocolo de ventana deslizable descrito anteriormente, el mecanismo de ventana del TCP hace posible enviar varios segmentos antes de que llegue un acuse de recibo. Hacerlo así aumenta la generación total de salida ya que mantiene ocupada a la red. La forma TCP de un protocolo de ventana deslizable también soluciona el problema de *control de flujo* de extremo a extremo, al permitir que el receptor restrinja la transmisión hasta que tenga espacio suficiente en memoria intermedia para incorporar más datos.

El mecanismo TCP de ventana deslizable opera a nivel de octeto, no a nivel de segmento ni de paquete. Los octetos del flujo de datos se numeran de manera secuencial, y el transmisor guarda tres apuntadores asociados con cada conexión. Los apuntadores definen una ventana deslizable, como se muestra en la figura 13.6. El primer apuntador marca el extremo izquierdo de la ventana deslizable, separa los octetos que ya se enviaron y envía el acuse de recibo de los octetos ya enviados. Un segundo apuntador marca el extremo derecho de la ventana deslizable y define el octeto más alto en la secuencia que se puede enviar antes de recibir más acuses de recibo. El tercer apuntador señala la frontera dentro de la ventana que separa los octetos que ya se enviaron de los que todavía no se envían. El software de protocolo envía sin retraso todos los octetos dentro de la ven-

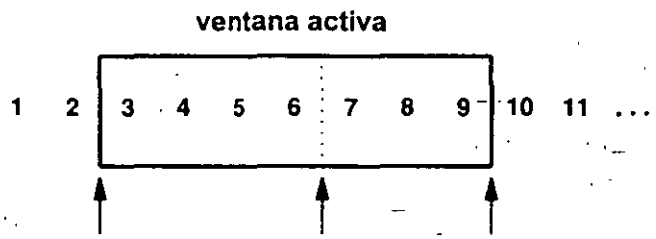



**Figura 13.6** Ejemplo de la ventana deslizante del TCP. Los octetos hasta el dos se han enviado y reconocido, los octetos del 3 al 6 han sido enviados pero no reconocidos, los octetos del 7 al 9 no se han enviado pero serán enviados sin retardo y los octetos del 10 en adelante no pueden ser enviados hasta que la ventana se mueva.

tana, por lo que en general, la frontera dentro de la ventana se mueve rápidamente de izquierda a derecha.

Hemos descrito cómo la ventana TCP del transmisor se desliza y hemos mencionado que el receptor debe tener una ventana similar para ensamblar de nuevo el flujo. Sin embargo, es importante entender que, como las conexiones TCP son de tipo full duplex, se llevan a cabo dos transferencias al mismo tiempo en cada conexión, una en cada dirección. Pensamos en las transferencias como en algo totalmente independientes porque, en cualquier momento, los datos pueden fluir a través de la conexión en una o en ambas direcciones. Por lo tanto, el software TCP en cada extremo mantiene dos ventanas por cada conexión (un total de cuatro), una se desliza a lo largo del flujo de datos que se envía, mientras la otra se desliza a lo largo de los datos que se reciben.

## 13.10 Tamaño variable de ventana y control de flujo

Una diferencia entre el protocolo TCP de ventana deslizable y el protocolo simplificado de ventana deslizable, presentado anteriormente, es que el TCP permite que el tamaño de la ventana varíe. Cada acuse de recibo, que informa cuántos octetos se recibieron, contiene un *aviso de ventana*, que especifica cuántos octetos adicionales de datos está preparado para aceptar el receptor. Pensemos el aviso de ventana como la especificación del tamaño actual de la memoria intermedia del receptor. En respuesta a un aumento en el aviso de ventana, el transmisor aumenta el tamaño de su ventana deslizable y procede al envío de octetos de los que todavía no se tiene un acuse de recibo. En respuesta a una disminución en el aviso de ventana, el transmisor disminuye el tamaño de su ventana y deja de enviar los octetos que se encuentran más allá de la frontera. El software TCP no debe contradecir los anuncios previos, reduciendo la posición aceptable de la ventana, que pasó anteriormente, en flujo de octetos. De hecho, los anuncios más pequeños acompañan a los acuses de recibo, así que el tamaño de la ventana cambia en el momento que se mueve hacia adelante.

La ventaja de utilizar una ventana de tamaño variable es que ésta proporciona control de flujo así como una transferencia confiable. Si la memoria intermedia del receptor se llena, no puede aceptar más paquetes, así que envía un anuncio de ventana más pequeño. En caso extremo, el receptor anuncia un tamaño de ventana igual a cero para detener toda la transmisión. Después, cuando hay memoria intermedia disponible, el receptor anuncia un tamaño de ventana distinto a cero para activar de nuevo el flujo de datos.[3]

Tener un mecanismo para el flujo de datos es esencial en un ambiente de red de redes, en donde las máquinas de varias velocidades y tamaños se comunican a través de redes y ruteadores de varias velocidades y capacidades. En realidad, existen dos problemas independientes de flujo. Primero, los protocolos de red de redes necesitan un control de flujo extremo a extremo entre la fuente y el destino final. Por ejemplo, cuando se comunican una minicomputadora y un gran mainframe, ambos necesitan regular la entrada de datos, o el software de protocolo se sobrecargaría rápidamente. Por lo tanto, el TCP debe implantar el control de flujo extremo a extremo para garantizar una entrega confiable. Segundo, los protocolos de red de redes necesitan un mecanismo de con-

[3] Hay dos excepciones para la transmisión cuando el tamaño de la ventana es cero. Primero, cuando un emisor está autorizado a transmitir un segmento con el bit de urgente activado para informar al receptor que está disponible un dato urgente. Segundo, para evitar un fin de cronometrado potencial si un anuncio diferente de cero se pierde luego de que el tama-

trol de flujo que permita que los sistemas intermedios (por ejemplo, los ruteadores) controlen una fuente que envíe más tráfico del que la máquina puede tolerar.

La sobrecarga de las máquinas intermedias se conoce como *congestionamiento* y los mecanismos que resuelven el problema se conocen como mecanismos de *control de congestionamiento*. El TCP emplea su esquema de ventana deslizable para resolver el problema de control de flujo extremo a extremo; no cuenta con un mecanismo explícito para el control de congestionamientos. Sin embargo, más adelante, veremos que una implantación TCP cuidadosamente programada puede detectar y resolver los congestionamientos, así como una implantación descuidada puede empeorarlos. En particular, aunque un esquema de retransmisión cuidadosamente seleccionado puede ser útil para evitar el congestionamiento, uno mal elegido puede empeorarlo.

## 13.11 Formato del segmento TCP

La unidad de transferencia entre el software TCP de dos máquinas se conoce como *segmento*. Los segmentos se intercambian para establecer conexiones, transferir datos, enviar acuses de recibo, anunciar los tamaños de ventanas y para cerrar conexiones. Debido a que el TCP utiliza acuses de recibo incorporados, un acuse que viaja de la máquina *A* a la máquina *B* puede viajar en el mismo segmento en el que viajan los datos de la máquina *A* a la máquina *B*, aun cuando el acuse de recibo se refiera a los datos enviados de *B* hacia *A*.[4] En la figura 13.7 se muestra el formato del segmento TCP.

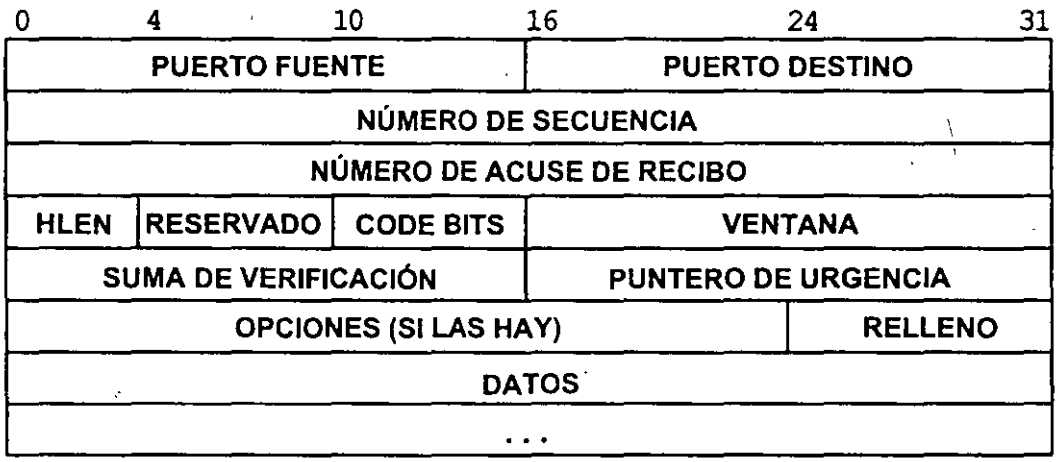



**Figura 13.7** Formato de un segmento TCP con un encabezado TCP seguido de datos. Los segmentos se utilizan para establecer conexiones, así como para transportar datos y acuses de recibo.

[4] En la práctica este tipo de incorporación no se presenta con frecuencia ya que la mayor parte de las aplicaciones no envía datos en ambas direcciones de manera simultánea.

Cada segmento se divide en dos partes: encabezado y datos. El encabezado, conocido como *encabezado TCP*, transporta la identificación y la información de control. Los campos *SOURCE PORT* (*PUERTO FUENTE*) y *DESTINATION PORT* (*PUERTO DESTINO*) contienen los números de puerto TCP que identifican a los programas de aplicación en los extremos de la conexión. El campo *SEQUENCE NUMBER* (*NÚMERO DE SECUENCIA*) identifica la posición de los datos del segmento en el flujo de datos del transmisor. El campo *ACKNOWLEDGEMENT NUMBER* (*NÚMERO DE ACUSE DE RECIBO*) identifica el número de octetos que la fuente espera recibir después. Observe que el número de secuencia se refiere al flujo que va en la misma dirección que el segmento, mientras que el número de acuse de recibo se refiere al flujo que va en la dirección opuesta al segmento.

El campo *HLEN*[5] contiene un número entero que especifica la longitud del encabezado del segmento, medida en múltiplos de 32 bits. Es necesario porque el campo *OPTIONS* (*OPCIONES*) varía en su longitud, dependiendo en qué opciones se haya incluido. Así, el tamaño del encabezado TCP varía dependiendo de las opciones seleccionadas. El campo de 6 bits marcado como *RESERVED* (*RESERVADO*), está reservado para usarse en el futuro.

Algunos segmentos sólo llevan un acuse de recibo y otros solamente llevan datos. Otros llevan solicitudes para establecer o cerrar una conexión. El software TCP utiliza el campo de 6 bits, etiquetado como *CODE BITS*, para determinar el propósito y contenido del segmento. Los seis bits indican cómo interpretar otros campos en el encabezado, de acuerdo con la tabla en la figura 13.8.

| Bit (de izquierda a derecha) | Significado si el bit está puesto a 1 |
|---|---|
| URG | El campo de puntero de urgente es válido |
| ACK | El campo de acuse de recibo es válido |
| PSH | Este segmento solicita una operación push |
| RST | Iniciación de la conexión |
| SYN | Sincronizar números de secuencia |
| FIN | El emisor ha llegado al final de su flujo de octetos |

**Figura 13.8** Bits del campo CODE en el encabezado TCP.

El software TCP informa sobre cuántos datos está dispuesto a aceptar cada vez que envía un segmento, al especificar su tamaño de memoria intermedia en el campo *WINDOW*. El campo contiene un número entero sin signo de 16 bits en el orden de octetos estándar de red. Los anuncios de ventana proporcionan otro ejemplo de acuse de recibo de carga, transporte y descarga ya que acompañan a todos los segmentos, tanto a los que llevan datos, como a los que sólo llevan un acuse de recibo.

## 13.12 Datos fuera de banda

Aunque el TCP es un protocolo orientado al flujo, algunas veces es importante que el programa en un extremo de la conexión envíe datos *fuera de banda*, sin esperar a que el programa en el otro extremo de la conexión consuma los octetos que ya están en flujo. Por ejemplo, cuando se utiliza el TCP para una sesión de acceso remoto, el usuario puede decidir si envía una secuencia de teclado que *interrumpa* o *aborte* el programa en el otro extremo. Dichas señales se necesitan aun más cuando un programa en la máquina remota no opera de manera correcta. Las señales se deben enviar sin esperar a que el programa lea los octetos que ya están en el flujo TCP (o no sería posible interrumpir programas que dejen de leer la entrada).

Para incorporar la señalización fuera de banda, el TCP permite que el transmisor especifique los datos como *urgentes*, dando a entender que se debe notificar su llegada al programa receptor tan pronto como sea posible, sin importar su posición en el flujo. El protocolo especifica que, cuando se encuentra con datos urgentes, el TCP receptor debe notificar al programa de aplicación, que esté asociado con la conexión, que entre en "modalidad urgente". Después de asimilar todos los datos urgentes, el TCP indica al programa de aplicación que regrese a su operación normal.

Por supuesto, los detalles exactos de cómo el TCP informa al programa de aplicación sobre datos urgentes dependen del sistema operativo de la máquina. El mecanismo utilizado para marcar los datos urgentes cuando se transmiten en un segmento consiste en un bit de código URG y en un campo *URGENT POINTER* (*PUNTERO DE URGENCIA*). Cuando se activa el bit URG, el indicador urgente especifica la posición dentro del segmento en la que terminan los datos urgentes.

## 13.13 Opción de tamaño máximo de segmento

No todos los segmentos que se envían a través de una conexión serán del mismo tamaño. Sin embargo, ambos extremos necesitan acordar el tamaño máximo de los segmentos que transferirán. El software TCP utiliza el campo *OPTIONS* para negociar con el software TCP en el otro extremo de la conexión; una de las opciones permite que el software TCP especifique el *tamaño máximo de segmento (MSS)* que está dispuesto a recibir. Por ejemplo, cuando un sistema incorporado que solamente tiene unos cuantos cientos de octetos de memoria intermedia se conecta con una gran supercomputadora, puede negociar un MSS que restrinja los segmentos para que quepan en la memoria intermedia. Para las computadoras conectadas por redes de área local de alta velocidad es especialmente importante escoger un tamaño máximo de segmento que llene los paquetes o no harán un buen uso del ancho de banda. Por lo tanto, si los dos puntos extremos residen en la misma red física, el TCP por lo general computará un tamaño máximo de segmento de tal forma que los datagramas IP resultantes correspondan con la MTU de la red. Si los puntos extremos no residen en la misma red física, pueden intentar descubrir la MTU mínima a lo largo del camino entre ellos o pueden escoger un tamaño máximo de segmento de *536* (tamaño máximo asignado por omisión de un datagrama IP, *576*, menos el tamaño estándar de los encabezados IP y TCP).

En un ambiente general de red de redes, escoger un tamaño máximo de segmento apropiado puede ser difícil, ya que el desempeño puede ser bajo tanto por tamaños de segmento demasiado grandes, como por tamaños muy pequeños. Por una parte, cuando el tamaño del segmento es pequeño, la utilización de la red permanece baja. Para entender por qué, recuerde que los segmentos

Cada segmento se divide en dos partes: encabezado y datos. El encabezado, conocido como *encabezado TCP*, transporta la identificación y la información de control. Los campos *SOURCE PORT* (*PUERTO FUENTE*) y *DESTINATION PORT* (*PUERTO DESTINO*) contienen los números de puerto TCP que identifican a los programas de aplicación en los extremos de la conexión. El campo *SEQUENCE NUMBER* (*NÚMERO DE SECUENCIA*) identifica la posición de los datos del segmento en el flujo de datos del transmisor. El campo *ACKNOWLEDGEMENT NUMBER* (*NÚMERO DE ACUSE DE RECIBO*) identifica el número de octetos que la fuente espera recibir después. Observe que el número de secuencia se refiere al flujo que va en la misma dirección que el segmento, mientras que el número de acuse de recibo se refiere al flujo que va en la dirección opuesta al segmento.

El campo *HLEN*[5] contiene un número entero que especifica la longitud del encabezado del segmento, medida en múltiplos de 32 bits. Es necesario porque el campo *OPTIONS* (*OPCIONES*) varía en su longitud, dependiendo en qué opciones se haya incluido. Así, el tamaño del encabezado TCP varía dependiendo de las opciones seleccionadas. El campo de 6 bits marcado como *RESERVED* (*RESERVADO*), está reservado para usarse en el futuro.

Algunos segmentos sólo llevan un acuse de recibo y otros solamente llevan datos. Otros llevan solicitudes para establecer o cerrar una conexión. El software TCP utiliza el campo de 6 bits, etiquetado como *CODE BITS*, para determinar el propósito y contenido del segmento. Los seis bits indican cómo interpretar otros campos en el encabezado, de acuerdo con la tabla en la figura 13.8.

| Bit (de izquierda a derecha) | Significado si el bit está puesto a 1 |
|---|---|
| URG | El campo de puntero de urgente es válido |
| ACK | El campo de acuse de recibo es válido |
| PSH | Este segmento solicita una operación push |
| RST | Iniciación de la conexión |
| SYN | Sincronizar números de secuencia |
| FIN | El emisor ha llegado al final de su flujo de octetos |

**Figura 13.8** Bits del campo CODE en el encabezado TCP.

El software TCP informa sobre cuántos datos está dispuesto a aceptar cada vez que envía un segmento, al especificar su tamaño de memoria intermedia en el campo *WINDOW*. El campo contiene un número entero sin signo de 16 bits en el orden de octetos estándar de red. Los anuncios de ventana proporcionan otro ejemplo de acuse de recibo de carga, transporte y descarga ya que acompañan a todos los segmentos, tanto a los que llevan datos, como a los que sólo llevan un acuse de recibo.

[5] La especificación indica que el campo *HLEN* es el desplazamiento del área de datos dentro del segmento.

TCP viajan encapsulados dentro de datagramas IP, que a su vez están encapsulados en tramas de red física. Por lo tanto, cada segmento tiene al menos 40 octetos de encabezados TCP e IP, además de los datos. Así pues, los datagramas que sólo llevan un octeto de datos utilizan como máximo 1/41 del ancho de banda de la red subyacente para los datos de usuario; en la práctica, las brechas mínimas entre paquetes y el hardware de red que ponen bits en tramas hacen que el rango sea aún más pequeño.

Por otro lado, los tamaños de segmento muy grandes también pueden producir un bajo desempeño. Los grandes segmentos resultan en grandes datagramas IP. Cuando dichos datagramas viajan a través de una red con una MTU pequeña, el IP debe fragmentarlos. A diferencia de un segmento TCP, un fragmento no se puede confirmar o retransmitir en forma independiente; todos los fragmentos deben llegar o de lo contrario se tendrá que retransmitir todo el datagrama. Debido a que la probabilidad de perder un fragmento no es de cero, aumentar el tamaño de segmento por arriba del umbral de fragmentación, disminuye la probabilidad de que lleguen los datagramas, lo que disminuye la producción de salida.

En teoría, el tamaño óptimo de segmento, *S*, ocurre cuando los datagramas IP que llevan los segmentos son tan grandes como sea posible sin requerir fragmentación en ninguna parte a lo largo del camino entre la fuente y el destino. En la práctica, encontrar *S* es difícil por muchas razones. Primero, la mayor parte de las implantaciones de TCP no incluye un mecanismo para hacerlo. Segundo, debido a que los ruteadores en una red de redes pueden cambiar las rutas en forma dinámica, el camino que siguen los datagramas entre un par de computadoras en comunicación puede cambiar también de manera dinámica, así como también puede cambiar el tamaño en que se tienen que fragmentar los datagramas. Tercero, el tamaño óptimo depende de los encabezados de protocolos de nivel más bajo (por ejemplo, el tamaño del segmento se debe reducir para incorporar opciones IP). La investigación sobre el problema de encontrar un tamaño óptimo de segmento continúa.

## 13.14 Cómputo de suma de verificación TCP

El campo *CHECKSUM* (*VERIFICACIÓN DE SUMA*) en el encabezado TCP contiene una suma de verificación de números enteros y 16 bits que se utiliza para verificar la integridad de los datos así como del encabezado TCP. Para computar la suma de verificación, el software TCP en la máquina transmisora sigue un procedimiento igual al descrito en el capítulo 12 para UDP. Coloca un *pseudo-encabezado* en el segmento, agrega suficientes bits en cero para lograr que el segmento sea un múltiplo de 16 bits y calcula la suma de 16 bits sobre todo el resultado. El TCP no cuenta el pseudo-encabezado ni los caracteres de relleno en la longitud del segmento, ni tampoco los transmite. También, asume que el campo de suma de verificación por sí mismo es de cero, para propósitos de la suma. Como en el caso de otras sumas de verificación, el TCP utiliza aritmética de 16 bits y toma el complemento a uno del complemento a uno de la suma. En la localidad receptora, el software TCP realiza el mismo cómputo para verificar que el segmento llega intacto.

El propósito de utilizar un pseudo-encabezado es exactamente el mismo que en el UDP. Permite que el receptor verifique que el segmento llegó a su destino correcto, que incluye tanto una dirección IP de anfitrión como un número de puerto de protocolo. Tanto la dirección IP de origen como la de destino son importantes para el TCP, ya que debe utilizarlas para identificar una conexión a la que pertenece el segmento. Así, cada vez que llega un datagrama que transporta un seg-

mento TCP, el IP debe pasar al TCP las direcciones IP de origen y destino, así como el segmento mismo. En la figura 13.9, se muestra el formato del pseudo-encabezado empleado en el cómputo de la suma de verificación.

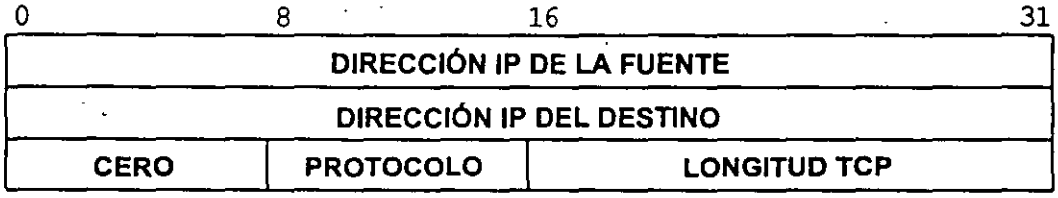



**Figura 13.9** Formato del pseudo-encabezado utilizado en el cálculo de la suma de verificación del TCP. En la localidad receptora, esta información se extrae del datagrama IP que transportaba el segmento.

El TCP transmisor asigna al campo *PROTOCOL* (*PROTOCOLO*) el valor que utilizará el sistema subyacente de entrega en su campo de tipo de protocolo. Para los datagramas IP que transporten TCP, el valor es 6. El campo *TCP LENGHT* (*LONGITUD TCP*) especifica la longitud total del segmento TCP, incluyendo el encabezado TCP. En el extremo receptor, la información utilizada en el pseudo-encabezado se extrae del datagrama IP que transportó el segmento y se incluye en el cómputo de la suma para verificar que el segmento llegó intacto al destino correcto.

## 13.15 Acuses de recibo y retransmisión

Como el TCP envía los datos en segmentos de longitud variable, y debido a que los segmentos retransmitidos pueden incluir más datos que los originales, los acuses de recibo no pueden remitirse fácilmente a los datagramas o segmentos. De hecho, se remiten a una posición en el flujo, utilizando los números de secuencia de flujo. El receptor recolecta octetos de datos de los segmentos entrantes y reconstruye una copia exacta del flujo que se envía. Como los segmentos viajan en datagramas IP, se pueden perder o llegar en desorden; el receptor utiliza los números de secuencia para reordenar los segmentos. En cualquier momento, el receptor tendrá cero o más octetos reconstruidos contiguamente desde el comienzo del flujo, pero puede tener piezas adicionales del flujo de datagramas que hayan llegado en desorden. El receptor siempre acusa recibo del prefijo contiguo más largo del flujo que se recibió correctamente. Cada acuse de recibo especifica un valor de secuencia mayor en una unidad, con respecto al octeto de la posición más alta en el prefijo contiguo que recibió. Por lo tanto, el transmisor recibe una retroalimentación continua del receptor conforme progresa el flujo. Podemos resumir esta idea importante de la siguiente manera:

> *Un acuse de recibo TCP especifica el número de secuencia del siguiente octeto que el receptor espera recibir.*

Al esquema TCP de acuse de recibo se le llama *acumulativo* porque reporta cuánto se ha acumulado del flujo. Los acuses de recibo acumulativos tienen ventajas y desventajas. Una ventaja es que los acuses de recibo son fáciles de generar y no son ambiguos. Otra es que los acuses de recibo perdidos no necesariamente forzarán la retransmisión. Una gran desventaja es que el receptor no obtiene información sobre todas las transmisiones exitosas, sino únicamente sobre una sola posición en el flujo que se recibió.

Para entender por qué la falta de información sobre todas las transmisiones exitosas hace que los acuses de recibo acumulativos sean menos eficientes, piense en una ventana que abarca *5000* octetos comenzando en la posición *101* en el flujo, y suponga que el transmisor envió todos los datos en la ventana al transmitir cinco segmentos. Suponga también que se pierde el primer segmento y que todos los demás llegan intactos. Conforme llega cada segmento, el receptor envía un acuse de recibo, pero todos los acuses especifican el octeto *101*, que es el octeto contiguo siguiente más alto que espera recibir. No hay forma para que el receptor indique al transmisor que llegó la mayor parte de los datos para la ventana actual.

Cuando ocurre una terminación de tiempo en el extremo transmisor, éste debe escoger entre dos esquemas potencialmente ineficaces. Puede retransmitir un segmento o retransmitir los cinco. En este caso, retransmitir los cinco segmentos no es eficaz. Cuando llega el primer segmento, el receptor tendrá todos los datos en la ventana, y en el acuse de recibo aparecerá *5101*. Si el transmisor sigue el estándar aceptado y retransmite sólo el primer segmento para el que no hay acuse, debe esperar a obtener el acuse de recibo antes de decidir qué y cuántos datos enviar. Por lo tanto, regresa a un protocolo simple de acuse de recibo positivo y puede perder las ventajas de tener una gran ventana.

## 13.16 Tiempo límite y retransmisión

Una de las ideas más importantes y complejas del TCP es parte de la forma en que maneja la terminación de tiempo (*time out*) y la retransmisión. Al igual que otros protocolos confiables, el TCP espera que el destino envíe acuses de recibo siempre que recibe exitosamente nuevos octetos del flujo de datos. Cada vez que envía un segmento, el TCP arranca un temporizador y espera un acuse de recibo. Si se termina el tiempo antes de que se acusen de recibidos los datos en el segmento, el TCP asume que dicho segmento se perdió o corrompió y lo retransmite.

Para entender por qué el algoritmo TCP de retransmisión difiere del algoritmo utilizado en muchos protocolos de red, necesitamos recordar que el TCP está diseñado para emplearse en un ambiente de red de redes. En una red de redes, un segmento que viaja entre dos máquinas puede atravesar una sola red de poco retraso (por ejemplo, una LAN de alta velocidad) o puede viajar a través de varias redes intermedias y de varios ruteadores. Por lo tanto, es imposible saber con anticipación qué tan rápido regresarán los acuses de recibo al origen. Además, el retraso en cada ruteador depende del tráfico, por lo que el tiempo total necesario para que un segmento viaje al destino y para que un acuse de recibo regrese al origen varía dramáticamente de un ejemplo a otro. En la figura 13.10, en la que se muestran medidas de tiempos de viaje redondo a través de la red Internet global para 100 paquetes consecutivos, se ilustra el problema. El software TCP debe incorporar las amplias diferencias en el tiempo necesario para llegar a varios destinos, así como los cambios en el tiempo necesario para llegar a cierto destino conforme varía la carga de tráfico.

El TCP maneja los retrasos variables en la red de redes al utilizar un *algoritmo adaptable de retransmisión*. En esencia, el TCP monitorea el desempeño de cada conexión y deduce valores razonables para la terminación de tiempo. Conforme cambia el desempeño de una conexión, el TCP revisa su valor de terminación de tiempo (por ejemplo, se adapta al cambio).

Para recolectar los datos necesarios para un algoritmo adaptable, el TCP registra la hora en la que se envía cada segmento y la hora en la que se recibe un acuse de recibo para los datos en el segmento. Considerando las dos horas, el TCP computa el tiempo transcurrido, conocido como *tiempo ejemplo de viaje redondo* o *ejemplo de viaje redondo*. Siempre que obtiene un nuevo ejemplo de viaje redondo, el TCP ajusta su noción del tiempo de viaje redondo promedio para la conexión. Por lo general, el software TCP almacena el tiempo estimado de viaje redondo, RTT (*round trip time*), como promedio calculado y utiliza nuevos ejemplos de viaje redondo para cambiar lentamente dicho promedio. Por ejemplo, para computar un nuevo cálculo de promedio, una técnica antigua para promediar se valía de un factor constante de cálculo, $\alpha$, donde $0 \leq \alpha < 1$, para calcular el promedio anterior contra el último ejemplo de viaje redondo:

$$\overline{RTT} = (\alpha * Old\_RTT) + ((1- \alpha)* New\_Round\_Trip\_Sample)$$

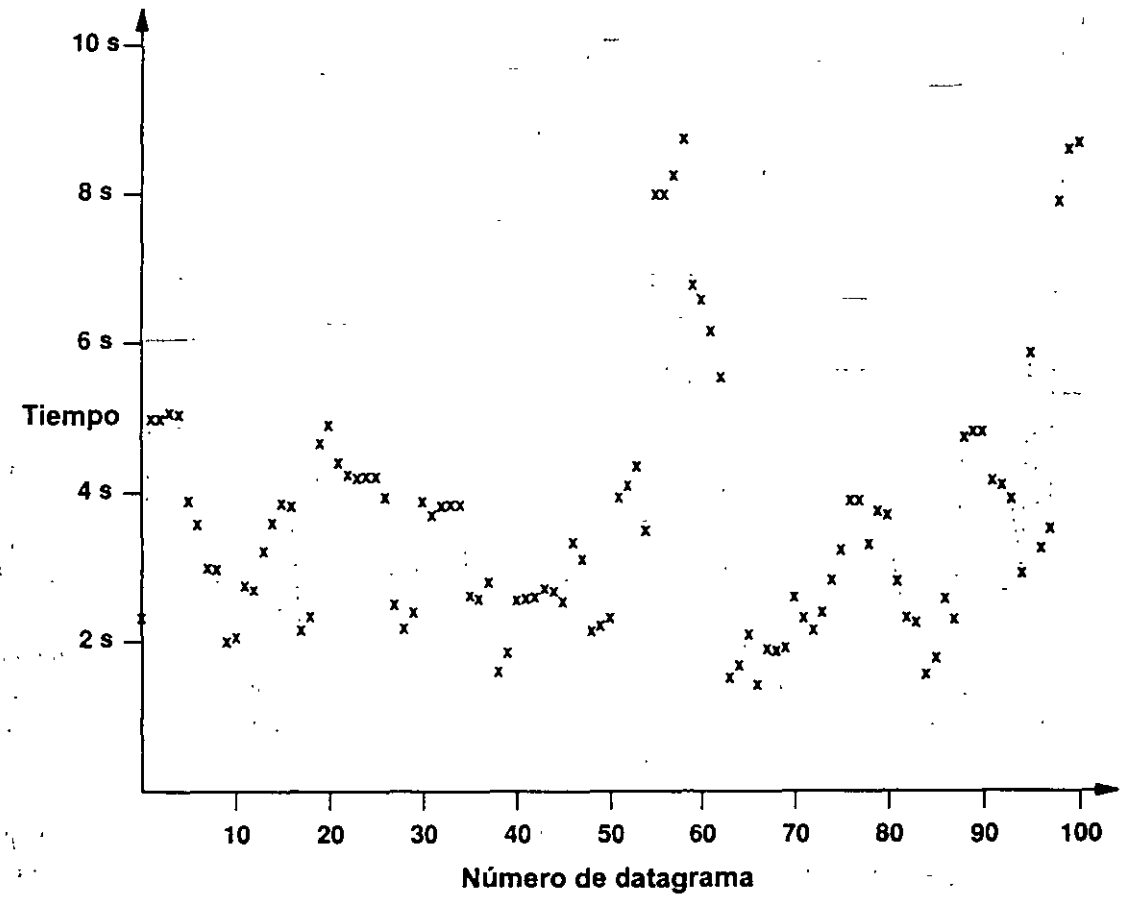



**Figura 13.10** Gráfico de tiempos de viaje redondo medidos para 100 datagramas IP sucesivos. Aun cuando la mayor parte de Internet opera ahora con retardos mucho menores, los retardos varían aún en tiempo.

Escoger un valor cercano a *1* para α, hace que el promedio calculado sea inalterable ante los cambios mínimos de tiempo (por ejemplo, un solo segmento que encuentra un gran retraso). Escoger un valor cercano a *0* para α, hace que el promedio calculado responda con rapidez a los cambios en el retraso.

Cuando envía un paquete, el TCP computa un valor de terminación de tiempo como una función de la estimación actual para viaje redondo. Las implantaciones antiguas del TCP se valían de un factor constante de cálculo, ß (ß > 1), y la terminación de tiempo era mayor que la estimación actual de viaje redondo:

$$\text{Terminación de tiempo} = ß * \text{RTT}$$

Escoger un valor para ß puede ser difícil. Por una parte, para detectar con rapidez la pérdida de paquetes, el valor de terminación de tiempo debe acercarse al tiempo actual de viaje redondo (por ejemplo, ß debe acercarse a *1*). La rápida detección de pérdida de paquetes mejora la producción de salida porque el TCP no esperará un tiempo innecesariamente largo para retransmitir. Por otra parte, si ß = *1*, el TCP se vuelve muy ansioso —cualquier retraso pequeño causará una retransmisión innecesaria, que desperdiciará el ancho de banda de la red. La especificación original recomendaba establecer ß = *2;* pero trabajos más recientes, descritos abajo, han producido mejores técnicas para el ajuste de la terminación de tiempo.

Podemos resumir las ideas hasta aquí presentadas de la siguiente manera:

> *Para manejar los retrasos variables que se encuentran en un ambiente de red de redes, el TCP utiliza un algoritmo adaptable de retransmisión que monitorea los retrasos en cada conexión y ajusta de acuerdo a ellos su parámetro de terminación de tiempo.*

## 13.17 Medición precisa de muestras de viaje redondo

En teoría, la medición de una muestra de viaje redondo es trivial —consiste en substraer la hora a la que se envía el segmento de la hora a la que llega el acuse de recibo. Sin embargo, surgen complicaciones debido a que el TCP se vale de un esquema de acuses de recibo acumulativos en el que un acuse se refiere a los datos recibidos y no al caso de un datagrama específico que transporta datos. Considere una retransmisión. El TCP forma un segmento, lo coloca en un datagrama y lo envía, el tiempo termina y el TCP vuelve a enviar el segmento en un segundo datagrama. Como ambos datagramas llevan exactamente los mismos datos, el receptor no tiene forma de saber si un acuse de recibo corresponde al datagrama original o al retransmitido. Este fenómeno se conoce como *ambigüedad de acuse de recibo* (*acknowledgement ambiguity*), y se dice que los acuses de recibo TCP son *ambiguos*.

¿Debe el TCP asumir que los acuses de recibo pertenecen a la primera transmisión (por ejemplo, al original) o a la última (por ejemplo, la transmisión más reciente)? De forma sorprendente, ninguno de los dos casos funciona. La asociación de los acuses de recibo con la transmisión original puede causar que el tiempo estimado de viaje redondo aumente sin medida en los casos en

los que una red de redes pierda datagramas.[6] Si llega un acuse de recibo después de una o más retransmisiones, el TCP medirá la muestra de viaje redondo de la transmisión original y computará un RTT nuevo utilizando la muestra excesivamente larga. Por lo tanto, el RTT crecerá ligeramente. En la siguiente ocasión que el TCP envíe un segmento, el RTT más largo resultará en terminaciones de tiempo ligeramente más grandes, por lo que si llega un acuse de recibo después de una o más retransmisiones, el siguiente tiempo de muestra de viaje redondo será aún más largo, y así sucesivamente.

La asociación de un acuse de recibo con la retransmisión más reciente también puede fallar. Considere lo que sucede cuando el retraso extremo a extremo aumenta repentinamente. Cuando el TCP envía un segmento, utiliza la estimación anterior de viaje redondo para computar una terminación de tiempo, que ahora es demasiado pequeña. El segmento llega y comienza el acuse de recibo, pero el aumento en el retraso significa que el tiempo termina antes de que llegue el acuse y el TCP retransmite el segmento. Poco después de que el TCP hace la retransmisión, llega el primer acuse de recibo y se asocia con la retransmisión. La muestra de viaje redondo será mucho más pequeño y resultará en una ligera disminución del tiempo estimado de viaje redondo, RTT. Por desgracia, disminuir la estimación de tiempo de viaje redondo garantiza que el TCP ajustará una terminación de tiempo demasiado pequeña para el siguiente segmento. Por último, la estimación del tiempo de viaje redondo se puede estabilizar en un valor, $T$, que sea de tal manera que el tiempo correcto de viaje redondo resulte ligeramente mayor que algunos múltiplos de $T$. Se ha observado que las implantaciones del TCP que asocian los acuses de recibo con la retransmisión más reciente llegan a un estado estable con el RTT ligeramente menor que la mitad del valor correcto (por ejemplo, el TCP envía cada segmento exactamente dos veces aunque no ocurra ninguna pérdida).

## 13.18 Algoritmo de Karn y anulación del temporizador

Si tanto la transmisión original como la más reciente fallan en proporcionar tiempos de viaje redondo, ¿qué debe hacer el TCP? La respuesta aceptada es sencilla: el TCP no debe actualizar la estimación de viaje redondo para los segmentos retransmitidos. Esta idea, conocida como *algoritmo de Karn*, evita el problema de todos los acuses de recibo ambiguos únicamente al ajustar la estimación de viaje redondo para acuses de recibo no ambiguos (acuses relacionados con segmentos que sólo se transmitieron una vez).

Por supuesto, una implantación simplista del algoritmo de Karn, que solamente ignore los tiempos para los segmentos retransmitidos, también puede conducir a fallas. Considere lo que sucede si el TCP envía un segmento después de un aumento significativo en el retraso. El TCP computa una terminación de tiempo mediante la estimación existente de viaje redondo. La terminación de tiempo será demasiado pequeña para el nuevo retardo y forzará la retransmisión. Si el TCP ignora los acuses de recibo para los segmentos retransmitidos, nunca actualizará la estimación y el ciclo continuará.

Para resolver dichas fallas, el algoritmo de Karn necesita que el transmisor combine las terminaciones de tiempo de transmisión con una estrategia de *anulación del temporizador* (timer backoff). La técnica de anulación computa una terminación de tiempo inicial por medio de una fórmu-

[6] La estimación sólo puede tener una longitud arbitrariamente grande si todos los segmentos se pierden al menos una vez.

la como la que se mostró anteriormente. Sin embargo, si se termina el tiempo y se provoca una retransmisión, el TCP aumenta el valor de terminación de tiempo. De hecho, cada vez que debe retransmitir un segmento, el TCP aumenta el valor de terminación de tiempo (para evitar que se vuelvan demasiado largos, la mayor parte de las implantaciones limitan los aumentos a una frontera mayor que es más larga que el retraso a lo largo de cualquier camino en la red de redes).

Las implantaciones utilizan varias técnicas para computar la anulación. La mayor parte escoge un factor multiplicativo, $\gamma$, y ajustan el nuevo valor a:

$$\text{new\_timeout} = \gamma * \text{timeout}$$

Por lo general, $\gamma$ es *2*. (Se ha argüido que los valores de $\gamma$ menores a 2 provocan inestabilidades.) Otras implantaciones utilizan una tabla de factores multiplicativos, lo que permite la anulación arbitraria en cada paso.[7]

El algoritmo de Karn combina la técnica de anulación con la estimación de viaje redondo para solucionar el problema de no incrementar las estimaciones de viaje redondo:

> *Algoritmo de Karn: Cuando se compute la estimación de viaje redondo, ignorar los ejemplos que correspondan a los segmentos retransmitidos, pero utilizar una estrategia de anulación, y mantener el valor de terminación de tiempo de un paquete retransmitido para los paquetes subsecuentes, hasta que se obtenga un ejemplo válido.*

Hablando en forma general, cuando una red de redes no se comporta adecuadamente, el algoritmo de Karn separa el cómputo del valor de terminación de tiempo de la estimación actual de viaje redondo. Utiliza la estimación antes mencionada para computar un valor inicial de terminación de tiempo, pero luego anula la terminación en cada retransmisión hasta que pueda transferir un segmento con éxito. Cuando envía segmentos subsecuentes, mantiene el valor de terminación de tiempo que resulta de la anulación. Por último, cuando llega un acuse de recibo correspondiente a un segmento que no requirió retransmisión, el TCP recalcula la estimación de viaje redondo y restablece la terminación de tiempo. La experiencia muestra que el algoritmo de Karn funciona bien, inclusive en las redes que tienen alta pérdida de paquetes.[8]

## 13.19 Respuesta a una variación alta en el retraso

La investigación sobre la estimación de viajes redondos ha mostrado que los cómputos descritos con anterioridad no se adaptan a un rango amplio de variación en el retraso. La teoría de poner en cola de espera sugiere que las variaciones en el tiempo de viaje redondo, $\sigma$, varían proporcionalmente a $1/(1-L)$, donde $L$ es la carga actual de red, $0 \leq L \leq 1$. Si una red de redes está corriendo a 50% de su capacidad, esperamos que el retraso de viaje redondo varíe por un factor de $\pm 2\sigma$, o *4*. Cuando la carga llega al 80%, esperamos una variación de *10*. El estándar TCP original especificaba la téc-

[7] El sistema Berkeley de UNIX es el sistema más notable de los que utiliza una tabla de factores, pero los valores activos en la tabla son equivalentes a usar $\gamma = 2$.

[8] Phil Karn es un radioaficionado entusiasta que ha desarrollado este algoritmo para permitir la comunicación TCP a través de conexiones de paquetes de radio de pérdidas altas.

nica para la estimación del tiempo de viaje redondo que describimos anteriormente. La utilización de esta técnica y la limitación de ß al valor sugerido de 2 significa que la estimación de viaje redondo se puede adaptar a cargas de hasta 30%.

La especificación de 1989 para el TCP necesita que las implantaciones estimen tanto el tiempo promedio de viaje redondo como la variación, y que utilicen la variación estimada en vez de la constante ß. Como resultado, las nuevas implantaciones del TCP se pueden adaptar a un rango más amplio de variación en el retraso y generar sustancialmente una salida más alta. Por fortuna, las aproximaciones requieren muy poca computación; se pueden derivar programas muy eficientes de las siguientes ecuaciones simples:

$$DIFF = SAMPLE - Old\_RTT$$

$$Smoothed\_RTT = Old\_RTT + \delta * DIFF$$

$$DEV = Old\_DEV + \rho(|DIFF| - Old\_DEV)$$

$$Timeout = Smoothed\_RTT + \eta * DEV$$

donde *DEV* es la desviación estimada deseada, $\delta$ es una fracción entre *0* y *1* que controla qué tan rápidamente afecta el nuevo ejemplo al promedio calculado, $\rho$ es una fracción entre *0* y *1* que controla qué tan rápidamente afecta el nuevo ejemplo la desviación estimada deseada, y $\eta$ es un factor que controla qué tanto afecta la desviación a la terminación de tiempo del viaje redondo. Para hacer el cómputo en forma eficiente, el TCP selecciona $\delta$ y $\rho$ para que cada una sea un inverso de una potencia de *2*, escala el cómputo por $2^n$ para lograr $n$ apropiadamente, y utiliza aritmética de números enteros. La investigación sugiere que los valores de $\delta = 1/2^3$, $\rho = 1/2^2$, y $n = 3$ funcionarán bien. El valor original para $\eta$ en 4.3BSD de UNIX era *2*; se cambió a *4* en 4.4BSD de UNIX.

## 13.20 Respuesta al congestionamiento

Parecería como si el software TCP se pudiera diseñar considerando la interacción entre dos puntos extremos de una conexión y los retrasos en la comunicación entre ellos. Sin embargo, en la práctica, el TCP también debe reaccionar al *congestionamiento* en la red de redes. El congestionamiento es una condición de retraso severo causada por una sobrecarga de datagramas en uno o más puntos de conmutación (por ejemplo, en ruteadores). Cuando ocurre un congestionamiento, los retrasos aumentan y los ruteadores comienzan a colocar en colas de salida a los datagramas hasta poderlos rutear. Debemos recordar que cada ruteador tiene una capacidad finita de almacenamiento y que los datagramas compiten por dicho almacenamiento (por ejemplo, en una red de redes basada en datagramas, no existe una prelocalización de recursos para conexiones TCP individuales). En el peor de los casos, el número total de datagramas entrantes a un ruteador congestionado, crece hasta que el ruteador alcanza su capacidad máxima y comienza a descartar datagramas.

Los puntos extremos por lo general no conocen los detalles sobre dónde ha ocurrido un congestionamiento o por qué. Para ellos, el congestionamiento tan sólo significa un mayor retraso. Por desgracia, la mayor parte de los protocolos de transporte utiliza la terminación de tiempo y la retransmisión, por lo que éstos responden a un aumento en el retraso retransmitiendo datagramas.

Las retransmisiones empeoran el congestionamiento en vez de solucionarlo. Si no se revisa, el incremento en el tráfico producirá mayor retraso, conduciendo a mayor tráfico, y así sucesivamente, hasta que la red no pueda utilizarse. La condición se conoce como *colapso por congestionamiento.*

Para evitar el colapso por congestionamiento, el TCP debe reducir la velocidad de transmisión cuando ocurre un congestionamiento. Los ruteadores verifican la longitud de sus colas de salida y utilizan técnicas como la solicitud de disminución ICMP para informar a los anfitriones que ha ocurrido un congestionamiento,[9] pero los protocolos de transporte como el TCP pueden ayudar a evitar el congestionamiento al reducir automáticamente la velocidad de transmisión siempre que ocurra un retraso. Por supuesto, los algoritmos para evitar los congestionamientos se deben diseñar con cuidado, ya que aun bajo condiciones normales de operación una red de redes tendrá amplias variaciones en los retrasos de viajes redondos.

Para evitar el congestionamiento, el estándar TCP ahora recomienda la utilización de dos técnicas: el *arranque lento* y la *disminución multiplicativa*. Estas técnicas están relacionadas y se pueden implantar con facilidad. Dijimos que para cada conexión, el TCP debe recordar el tamaño de la ventana del receptor (por ejemplo, el tamaño de la memoria intermedia, indicado en los acuses de recibo). Para controlar el congestionamiento, el TCP mantiene un segundo límite, llamado *límite de ventana de congestionamiento* o *ventana de congestionamiento*. En cualquier momento, el TCP actúa como si el tamaño de la ventana fuera:

```
Allowed_window = min(receiver_advertisement, congestion_window)
```

En un estado constante de una conexión no congestionada, la ventana de congestionamiento es del mismo tamaño que la ventana del receptor. La reducción de la ventana de congestionamiento reduce el tráfico que el TCP inyectará a la conexión. Para estimar el tamaño de la ventana de congestionamiento, el TCP asume que la mayor parte de la pérdida de datagramas viene del congestionamiento y se vale de la siguiente estrategia:

> *Prevención del Congestionamiento por Disminución Multiplicativa: Cuando se pierda un segmento, reducir a la mitad la ventana de congestionamiento (hasta un mínimo de un segmento). Para los segmentos que permanezcan en la ventana permitida, anular exponencialmente el temporizador para la retransmisión.*

Debido a que el TCP reduce a la mitad la ventana de congestionamiento por *cada* pérdida, disminuye la ventana exponencialmente si la pérdida continúa. En otras palabras, si el congestionamiento continúa, el TCP reduce exponencialmente el volumen de tráfico, así como la velocidad de retransmisión. Si la pérdida continúa, el TCP finalmente limita la transmisión a un solo datagrama y continúa duplicando los valores de terminación de tiempo antes de retransmitir. La idea es proporcionar una reducción rápida y significativa del tráfico, a fin de permitir que otros ruteadores tengan suficiente tiempo para deshacerse de los datagramas que ya tienen en sus colas de espera.

¿Cómo se puede recuperar el TCP cuando termina el congestionamiento? Usted puede sospechar que el TCP debe revertir la disminución multiplicativa y duplicar la ventana de congestionamiento cuando el tráfico comienza a fluir de nuevo. Sin embargo, hacerlo así produciría un sistema

[9] En una red congestionada, la longitud de las colas crece exponencialmente para un tiempo significativo.

inestable que oscilaría mucho entre poco tráfico y congestionamiento. Por el contrario, el TCP emplea una técnica, llamada *arranque lento*[10] para aumentar la transmisión:

> *Recuperación de arranque lento (aditiva): siempre que se arranque el tráfico en una nueva conexión o se aumente el tráfico después de un periodo de congestionamiento, activar la ventana de congestionamiento con el tamaño de un solo segmento y aumentarla un segmento cada vez que llegue un acuse de recibo.*

Los arranques lentos evitan saturar la red de redes con tráfico adicional, justamente después de que se libere un congestionamiento o cuando comienzan repentinamente nuevas conexiones.

El término *arranque lento* puede no estar bien aplicado porque bajo condiciones ideales, el arranque no es muy lento. El TCP inicia la ventana de congestionamiento con *1*, envía un segmento inicial y espera. Cuando llega el acuse de recibo, aumenta la ventana de congestionamiento a *2*, envía dos segmentos y espera. Cuando llegan los dos acuses de recibo, cada uno aumenta la ventana de congestionamiento en *1*, por lo que el TCP puede enviar *4* segmentos. Los acuses de recibo por estos *4* segmentos incrementarán a *8* la ventana de congestionamiento. Cuando ocurren cuatro viajes redondos, el TCP puede enviar *16* segmentos, que a veces es lo suficiente para llegar al límite de la ventana del receptor. Inclusive para ventanas muy largas, únicamente toma $\log_2 N$ viajes redondos antes de que el TCP pueda enviar N segmentos.

Para evitar el aumento demasiado rápido del tamaño de la ventana y no causar congestionamiento adicional, el TCP agrega una restricción más. Una vez que la ventana de congestionamiento llega a la mitad de su tamaño original, antes del congestionamiento, el TCP entra en una fase de *prevención de congestionamiento* y hace más lenta la velocidad de incremento. Durante la prevención de congestionamiento, aumenta el tamaño de la ventana por *1* solamente si, para todos los segmentos en la ventana, se tiene acuses de recibo.

Juntos, el incremento de arranque lento, la disminución multiplicativa, la prevención de congestionamiento, la medida de variación, y la anulación exponencial del temporizador mejoran notablemente el desempeño del TCP, sin agregar ningún trabajo computacional significativo del software de protocolo. Las versiones que utilizan estas técnicas han mejorado el desempeño de versiones anteriores en factores de *2* a *10*.

## 13.21 Establecimiento de una conexión TCP

Para establecer una conexión, el TCP utiliza un saludo (*handshake*) de tres etapas. En el caso más sencillo, este intercambio procede como se muestra en la figura 13.11.

El primer segmento del saludo se puede identificar porque tiene activo el bit SYN[11] en el campo de código. El segundo mensaje tiene tanto el bit SYN como el bit ACK activos, indicando tanto el acuse de recibo del primer segmento SYN como el hecho de que se continúa con el intercambio. El mensaje final del saludo es sólo un acuse de recibo y nada más se utiliza para informar al destino que ambos extremos están de acuerdo en establecer una conexión.

---

[10] El término *arranque lento* se atribuye a John Nagle; la técnica se conoció originalmente como *arranque suave*.

[11] SYN es una expresión que se emplea como abreviatura de *synchronization*; se pronuncia "sin".

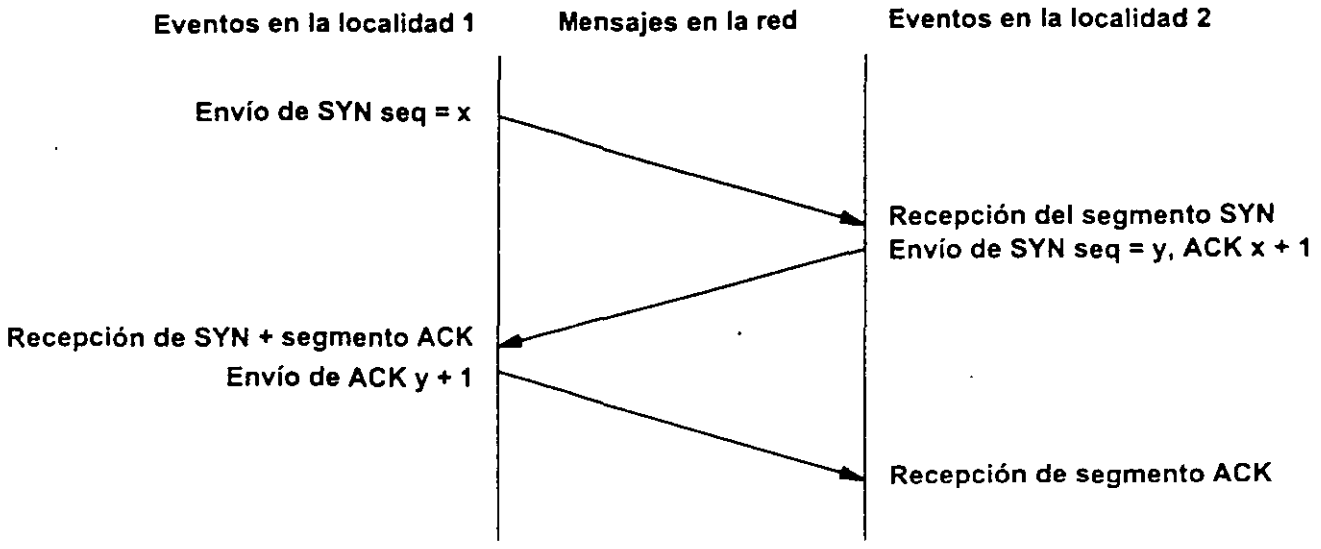



**Figura 13.11** Secuencia de mensajes en el saludo de tres etapas. En la representación, el tiempo transcurre hacia la parte inferior de la página; las líneas diagonales representan segmentos enviados entre localidades. Los segmentos SYN transportan información sobre el número de secuencia inicial.

Por lo general, el software TCP en una máquina espera de forma pasiva el intercambio de señales y el software TCP en otra máquina lo inicia. Sin embargo, el saludo (handshake) está cuidadosamente diseñado para funcionar aun cuando ambas máquinas intenten iniciar una conexión al mismo tiempo. Por lo tanto, se puede establecer una conexión desde cualquier extremo o desde ambos al mismo tiempo. Una vez que se establece la conexión, los datos pueden fluir en ambas direcciones por igual. No existe un maestro ni un esclavo.

El saludo de tres etapas es necesario y suficiente para la sincronización correcta entre los dos extremos de la conexión. Para entender por qué, recuerde que el TCP se construye sobre un servicio de entrega no confiable de paquetes, así que los mensajes pueden perderse, retrasarse, duplicarse o entregarse en desorden. Por lo tanto, el protocolo debe utilizar un mecanismo de terminación de tiempo y retransmitir las solicitudes perdidas. Sucederán algunos problemas si las solicitudes originales y retransmitidas llegan mientras se establece la conexión o si las solicitudes retransmitidas se retrasan hasta que se establezca, utilice y termine una conexión. Un saludo de tres etapas (más la regla de que el TCP ignore solicitudes adicionales de conexión después de que se establezca la misma), resuelve estos problemas.

## 13.22 Números de secuencia inicial

El saludo (*handshake*) de tres etapas realiza dos funciones importantes. Garantiza que ambos lados estén listos para transferir datos (y que tengan conocimiento de que ambos están listos) y permite, a ambas partes, acordar un número de secuencia inicial. Los números de secuencia son enviados y reconocidos durante el saludo. Cada máquina debe seleccionar un número de secuencia inicial en forma aleatoria que se utilizará para identificar octetos en el flujo que se está enviando. Los núme-

ros de secuencia no pueden comenzar siempre con el mismo valor. En particular, el TCP no puede seleccionar una secuencia *1* cada vez que crea una conexión (en uno de los ejercicios se examinan los problemas que se pueden originar si se hace de esta manera). Por supuesto, es importante que ambas partes acuerden un número inicial, así como el número de octetos empleados en un acuse de recibo de acuerdo a los utilizados en el segmento de datos.

Para entender cómo pueden acordar las máquinas un número de secuencia para dos flujos después de tres mensajes solamente, recordemos que cada segmento contiene un campo de número de secuencia y un campo de acuse de recibo. La máquina *A*, que inicia un saludo, transfiere un número de secuencia inicial, *x*, en el campo de secuencia del primer segmento SYN como parte del saludo de tres etapas. La segunda máquina, *B*, recibe el SYN, registra el número de secuencia y responde enviando su número de secuencia inicial en el campo de secuencia así como un reconocimiento que especifica el octeto $x+1$ esperado por *B*. En el mensaje final del saludo, *A* envía un "acuse de recibo" de la recepción del mensaje de *B* de todos los octetos a través de *y*. En todos los casos, los acuses de recibo siguen la convención de utilizar el número del *próximo* octeto esperado.

Hemos descrito cómo el TCP normalmente transporta el saludo de tres etapas intercambiando segmentos que contienen una cantidad mínima de información. Debido al diseño del protocolo es posible enviar datos junto con los números de secuencia iniciales en los segmentos de saludo. En cada caso el software TCP debe manejar los datos hasta que se complete el saludo. Una vez que la conexión se ha establecido, el software TCP puede liberar los datos manejados y entregarlos rápidamente al programa de aplicación. El lector deberá referirse a las especificaciones del protocolo para obtener más detalles.

## 13.23 Terminación de una conexión TCP

Dos programas que utilizan el TCP para comunicarse pueden terminar la conversación cortésmente valiéndose de la operación *close*. De manera interna, el TCP utiliza una modificación del saludo de tres etapas para cerrar conexiones. Recordemos que las conexiones TCP son de tipo full duplex y que hemos visto que éstas contienen dos transferencias de flujo independientes, una en cada dirección. Cuando un programa de aplicación informa al TCP que ya no tiene más datos para enviar, éste cerrará la conexión *en una dirección*. Para cerrar la mitad de una conexión, el emisor TCP termina de transmitir los datos restantes, espera la recepción de un acuse de recibo y, entonces, envía un segmento con el bit FIN activado. El receptor TCP reconoce el segmento FIN e informa al programa de aplicación en su extremo que no tiene más datos disponibles (por ejemplo, mediante el mecanismo de fin de archivo de sistema operativo).

Una vez que la conexión se ha cerrado en una dirección dada, TCP rechaza más datos en esta dirección. Mientras tanto, los datos pueden continuar fluyendo en la dirección opuesta hasta que el emisor se cierra. Por supuesto, los acuses de recibo continúan fluyendo hacia el emisor incluso después de que la conexión se ha cerrado. Cuando ambas direcciones se han cerrado, el software TCP en cada punto extremo borra sus registros de la conexión.

Los detalles del cierre de una conexión son más sutiles de lo que se ha sugerido anteriormente porque el TCP utiliza un saludo de tres etapas modificado para cerrar una conexión. La figura 13.12 ilustra el procedimiento.

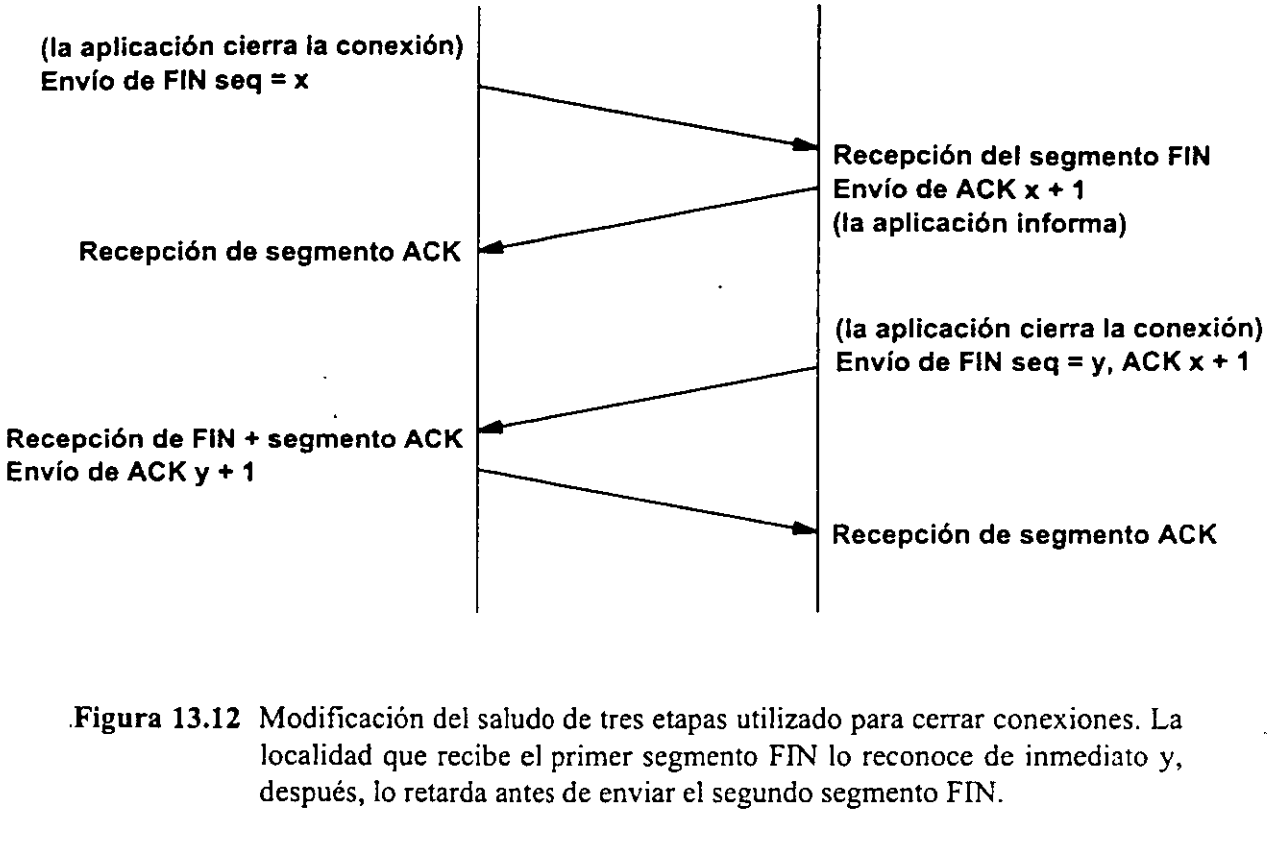



**Figura 13.12** Modificación del saludo de tres etapas utilizado para cerrar conexiones. La localidad que recibe el primer segmento FIN lo reconoce de inmediato y, después, lo retarda antes de enviar el segundo segmento FIN.

La diferencia entre el saludo de tres etapas empleado para establecer e interrumpir conexiones se presenta luego de que una máquina recibe el segmento FIN inicial. En lugar de generar un segundo segmento FIN inmediatamente, el TCP envía un acuse de recibo y luego informa a la aplicación de la solicitud de interrupción. Informar al programa de aplicación de la solicitud y obtener una respuesta, puede tomar un tiempo considerable (por ejemplo, si comprende la interacción humana). El acuse de recibo evita la retransmisión del segmento inicial FIN durante la espera. Por último, cuando el programa de aplicación instruye al TCP para que interrumpa la conexión completamente, el TCP envía el segundo segmento FIN y la localidad original responde con el tercer mensaje, un ACK.

## 13.24 Restablecimiento de una conexión TCP

Normalmente, un programa de aplicación se vale de la operación de cierre para interrumpir una conexión cuando termina de utilizarla. Así, el cierre de conexión es considerado como una parte normal de su uso, análogo al cierre de archivos. Algunas veces se presentan condiciones anormales que obligan a un programa de aplicación o al software de red a interrumpir una conexión. El TCP proporciona una capacidad de iniciación (*reset*) para estas desconexiones anormales.

Para iniciar una conexión, un lado inicia la interrupción enviando un segmento con el bit RST activado en el campo *CODE*. El otro lado responde a un segmento de iniciación inmediatamente interrumpiendo la conexión. El TCP también informa al programa de aplicación que se ha

presentado una iniciación. Una iniciación es una interrupción instantánea, lo cual significa que la transferencia en ambas direcciones se interrumpe de manera inmediata y se liberan recursos como los búfers.

## 13.25 Máquina de estado TCP

Como en la mayor parte de los protocolos, la operación del TCP se puede explicar mejor mediante un modelo teórico, llamado *máquina de estado finito*. La figura 13.13 muestra la máquina de estado finito TCP, en ella los círculos representan estados y las flechas representan transiciones entre éstos. El nombre en cada transición muestra qué recibe el TCP para generar la transición y qué envía como respuesta. Por ejemplo, el software TCP en cada extremo comienza en un estado *CLOSED* (*CERRADO*). El programa de aplicación debe emitir un comando *passive open* (apertura pasiva) (para esperar una conexión desde otra máquina) o un comando *active open* (apertura activa) (para iniciar una conexión). El comando active open obliga a que se dé una transición del estado *CLOSED* al estado *SYN SENT*. Cuando el TCP continúa con la transición, emite un segmento SYN. Cuando el otro extremo devuelve un segmento que contiene un SYN, más un ACK, el TCP cambia al estado *ESTABLISHED* (*ESTABLECIDO*) y comienza la transferencia de datos.

El estado *TIMED WAIT* (*ESPERA CRONOMETRADA*) revela cómo el maneja el TCP algunos de los problemas que se presentan con la entrega no confiable. El TCP conserva una noción de *máximo tiempo de vida del segmento*, el tiempo máximo en que un segmento puede mantenerse activo en una red de redes. Para evitar tener segmentos de una conexión previa interfiriendo con los actuales, el TCP cambia el estado *TIMED WAIT* después de cerrar una conexión. Se mantiene en este estado dos veces el máximo tiempo de vida del segmento antes de borrar sus registros de la conexión. Si algún segmento duplicado logra llegar a la conexión durante el intervalo de exceso de tiempo, el TCP lo rechazará. Sin embargo, para manejar casos cuyo el último acuse de recibo fue perdido, reconocerá los segmentos válidos y reiniciará el temporizador. Dado que el temporizador permite al TCP distinguir entre las conexiones anteriores de las nuevas, se evita que el TCP responda con un *RST* (iniciación) si el otro extremo retransmite una solicitud *FIN*.

## 13.26 Forzando la entrega de datos

Hemos dicho que el TCP es libre de dividir el flujo de datos en segmentos para su transmisión sin considerar el tamaño de transferencia que utiliza el programa de aplicación. La mayor ventaja de permitir que el TCP elija la forma de dividir es la eficiencia que se obtiene. Puede acumular suficientes octetos en una memoria intermedia para hacer los segmentos razonablemente largos, reduciendo las sobrecargas altas que se presentan cuando los segmentos contienen sólo unos cuantos octetos de datos.

Aun cuando el procesamiento en memoria intermedia mejora el desempeño de la red, puede interferir con algunas aplicaciones. Consideremos el uso de una conexión de TCP para transferir caracteres de una terminal interactiva a una máquina remota. El usuario espera una respuesta instantánea para cada pulsación de tecla. Si el emisor TCP pone en memoria intermedia los datos, la

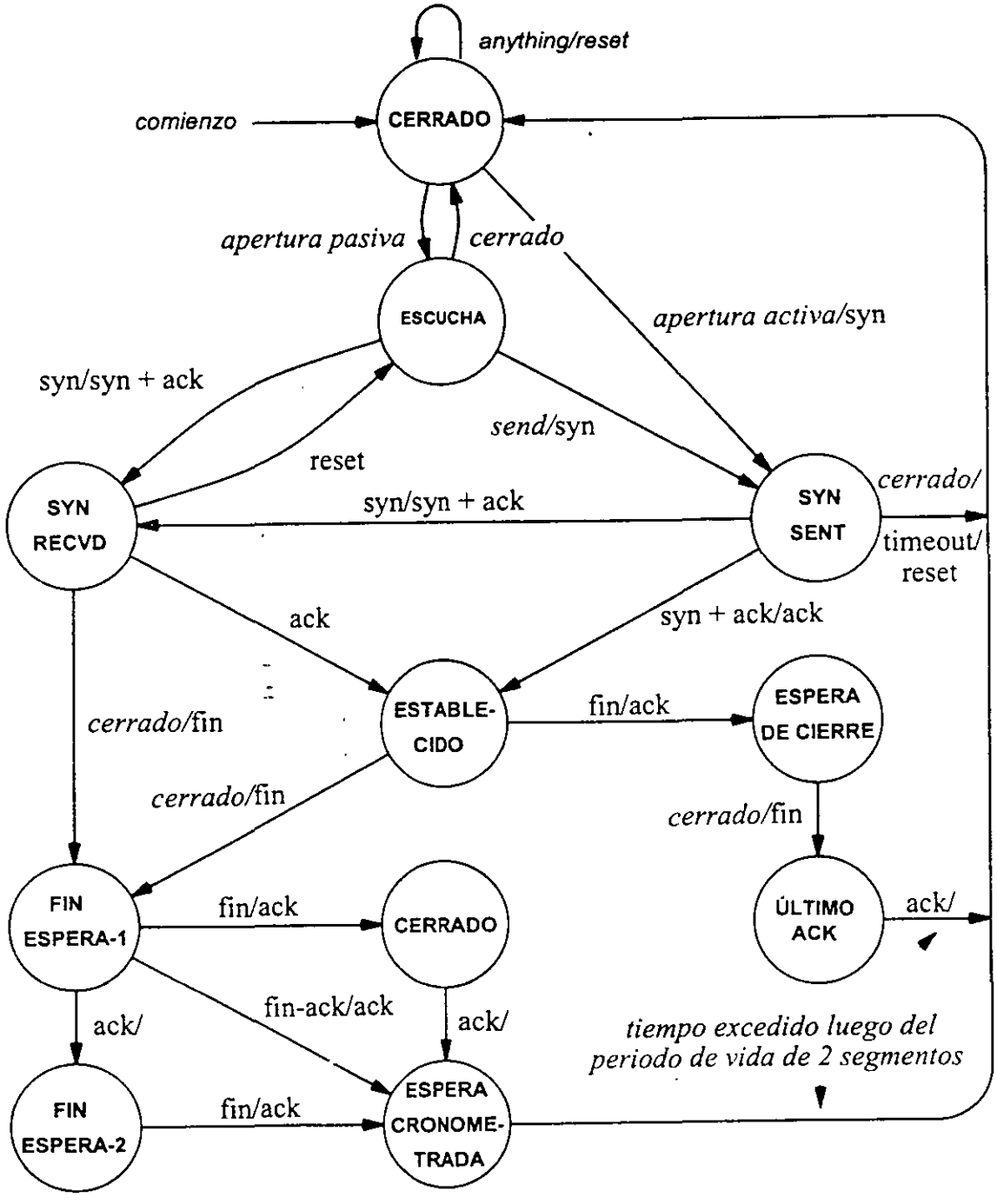



**Figura 13.13** Máquina de estado finito TCP. Cada punto final comienza en el estado *cerrado*. Los nombres de las transiciones muestran la entrada que ocasiona la transición, seguida por la salida, si la hay.

respuesta podría retrasarse, posiblemente por cientos de pulsaciones de teclas. De la misma forma, debido a que el receptor TCP puede poner en memoria intermedia los datos antes de que estén disponibles para el programa de aplicación en su extremo, forzar al emisor a transmitir los datos puede no ser suficiente para garantizar la entrega.

Para adaptarse a los usos interactivos, el TCP proporciona la operación *push (empujar)*, que un programa de aplicación puede utilizar para forzar la entrega de octetos actuales en el flujo de transmisión sin esperar a que se les almacene en memoria intermedia. La operación empujar hace más que forzar al TCP a enviar un segmento. También solicita al TCP que active el bit *PSH* en el segmento de campo de código, así los datos se entregarán al programa de aplicación en el extremo de recepción. Entonces, cuando se envían datos desde una terminal interactiva, la aplicación utiliza la función empujar luego de cada pulsación de tecla. De la misma forma, los programas de aplicación pueden forzar la salida para que sea enviada y desplegada en el indicador de la terminal llamando a la función empujar luego de escribir un carácter o una línea.

## 13.27 Números reservados de puerto TCP

Como el UDP, el TCP combina la asignación dinámica y estática de puertos mediante un conjunto de *asignación de puertos bien conocidos* para programas llamados con frecuencia (por ejemplo, el correo electrónico), pero la salida de la mayor parte de los números de puerto disponibles para el sistema operativo se asigna conforme los programas lo necesitan. Aun cuando el estándar original reservaba los números de puerto menores a *256* para utilizarlos como puertos bien conocidos, ahora se han asignado números superiores a 1,024. La figura 13.14 lista algunos de los puertos TCP asignados en la actualidad. Habría que puntualizar que, aunque los números de puerto TCP y UDP son independientes, los diseñadores han decidido utilizar el mismo número entero de puerto para cualquier servicio accesible desde UDP y TCP. Por ejemplo, un servidor de nombre de dominio puede acesarse con el TCP o el UDP. En ambos protocolos, el puerto número 53 ha sido reservado para los servidores en el sistema de nombre de dominio.

## 13.28 Desempeño del TCP

Como hemos visto, el TCP es un protocolo complejo que maneja las comunicaciones sobre una amplia variedad de tecnologías de red subyacentes. Mucha gente asume que, como el TCP aborda tareas mucho más complejas que otros protocolos de transporte, el código debe ser incómodo e ineficaz. Sorpresivamente, en general, lo que hemos analizado no parece entorpecer el desempeño del TCP. Experimentos realizados en Berkeley han mostrado que el mismo TCP que opera en forma eficiente en la red global de Internet puede proporcionar un desempeño sostenido a 8 Mbps con datos de usuario entre dos estaciones de trabajo en una red Ethernet a 10 Mbps.[12] Los investigadores de Cray Research, Inc. han demostrado que el desempeño del TCP se acerca a un gigabit por segundo.

[12] Encabezado Ethernet, IP, y TCP y el espacio requerido de "inter-packet", representan en amplitud de banda remanente.

| Decimal | Clave | Clave UNIX | Descripción |
|---|---|---|---|
| 0 | | | Reservado |
| 1 | TCPMUX | - | Multiplexor TCP |
| 5 | RJE | - | Introducción de función remota |
| 7 | ECHO | echo | Echo |
| 9 | DISCARD | discard | Abandonar |
| 11 | USERS | systat | Usuarios activos |
| 13 | DAYTIME | daytime | Fecha, hora |
| 15 | - | netstat | Programa de estado de red |
| 17 | QUOTE | qotd | Cita del día |
| 19 | CHARGEN | chargen | Generador de caracteres |
| 20 | FTP-DATA | ftp-data | Protocolo de transferencia de archivos (datos) |
| 21 | FTP | ftp | Protocolo de transferencia de archivos |
| 23 | TELNET | telnet | Conexión de Terminal |
| 25 | SMTP | smtp | Protocolo de transporte de correo sencillo |
| 37 | TIME | time | Hora |
| 42 | NAMESERVER | name | Nombre del anfitrión servidor |
| 43 | NICNAME | whois | ¿Quién está ahí? |
| 53 | DOMAIN | nameserver | Servidor de nombre de dominio |
| 77 | - | rje | Cualquier servicio RJE privado |
| 79 | FINGER | finger | Finger |
| 93 | DCP | - | Protocolo de control de dispositivo |
| 95 | SUPDUP | supdup | Protocolo SUPDUP |
| 101 | HOSTNAME | hostnames | Servidor de nombre de anfitrión NIC |
| 102 | ISO-TSAP | iso-tsap | ISO-TSAP |
| 103 | X400 | x400 | Servicio de correo X.400 |
| 104 | X400-SND | x400-snd | Envío de correo X.400 |
| 111 | SUNRPC | sunrpc | Llamada a procedimiento remoto de SUN |
| 113 | AUTH | auth | Servicio de autenticación |
| 117 | UUCP-PATH | uucp-path | Servicio de trayecto UUCP |
| 119 | NNTP | nntp | Protocolo de transferencia de noticias USENET |
| 129 | PWDGEN | - | Protocolo generador de clave de acceso |
| 139 | NETBIOS-SSN | - | Servicio de sesión NETBIOS |
| 160-223 | Reservado | | |

**Figura 13.14** Ejemplos de números de puerto TCP asignados actualmente. Para posibles extensiones, protocolos como el UDP utilizan los mismos números.

## 13.29 Síndrome de ventana tonta y paquetes pequeños

Los investigadores que participaron en el desarrollo del TCP observaron un serio problema de desempeño que puede presentarse cuando las aplicaciones del emisor y el receptor operan a velocidades diferentes. Para entender el problema, recordemos que el TCP almacena en memoria intermedia los datos de entrada y consideremos lo que sucedería si la aplicación de un receptor elige leer los datos de entrada un octeto a la vez. Cuando una conexión se establece por primera vez, el receptor TCP asigna un búfer de $K$ octetos y utiliza el campo *WINDOW*, en los segmentos de acuse de recibo, para anunciar el tamaño disponible del búfer al emisor. Si la aplicación del emisor gene-

ra datos rápidamente, el emisor TCP transmitirá segmentos con datos para toda la ventana. Finalmente, el emisor recibirá un acuse de recibo que especifique que toda la ventana está llena y que no queda espacio adicional en el búfer del receptor.

Cuando la aplicación de recepción lee un octeto de datos desde la memoria intermedia llena, queda disponible un espacio de un octeto. Hemos dicho que cuando un espacio queda disponible en el búfer, el TCP genera un acuse de recibo que utiliza el campo *WINDOW*, en la máquina de recepción, para informar al emisor. En el ejemplo, el receptor anunciará una ventana de un octeto. Cuando tenga conocimiento del espacio disponible, el emisor TCP responderá con la transmisión de un segmento que contenga un octeto de datos.

Aun cuando el anuncio de la ventana de un solo octeto trabaja de manera correcta conservando llena la memoria intermedia del receptor, el resultado es una serie de segmentos de datos pequeños. El emisor TCP debe componer un segmento que contenga un octeto de datos, colocar el segmento en un datagrama IP y transmitir el resultado. Cuando la aplicación de recepción lea otro octeto, el TCP generará otro acuse de recibo, lo cual ocasionará que el emisor transmita otro segmento que contenga un octeto de datos. La interacción resultante puede llegar a estabilizarse en un estado en el cual el TCP envíe un segmento separado para cada octeto de datos.

La transferencia de segmentos pequeños ocupa ancho de banda de la red innecesariamente e introduce una sobrecarga computacional. La transmisión de pequeños segmentos ocupa un ancho de banda pues cada datagrama transfiere sólo un octeto de datos; la cantidad de encabezados para los datos será muy extensa. La sobrecarga computacional se origina debido a que el TCP, tanto en el emisor como en el receptor, debe procesar cada segmento. El software TCP del emisor tiene que asignar espacio de memoria intermedia, formar un encabezado de segmento y calcular una suma de verificación para el segmento. De la misma forma, el software IP en la máquina emisora debe encapsular el segmento en un datagrama, calcular la suma de verificación del encabezado, rutear el datagrama y transferirlo hacia la interfaz de red apropiada. En la máquina de recepción, el IP debe verificar la suma de verificación del encabezado y transferir el segmento hacia el TCP. El TCP tiene que verificar la suma de verificación del segmento, examinar el número de secuencia, extraer el dato y colocarlo en una memoria intermedia.

Aun cuando hemos descrito lo que pueden provocar los segmentos pequeños cuando un receptor anuncia una ventana pequeña disponible, un emisor puede también ocasionar que cada segmento contenga una pequeña cantidad de datos. Por ejemplo, imagine una implantación del TCP que envía datos agresivamente cada vez que están disponibles y considere qué sucedería si una aplicación del emisor generara un octeto de datos por vez. Luego de que la aplicación generara un octeto de datos, el TCP crearía y transmitiría un segmento. El TCP puede también enviar un segmento pequeño si una aplicación genera datos en bloques de tamaños fijos de $B$ octetos y el emisor TCP extrae datos de la memoria intermedia en bloques del tamaño del segmento máximo $M$, donde $M$ es diferente de $B$, dado que el último bloque en una memoria intermedia puede ser pequeño.

Este problema se conoce como *síndrome de ventana tonta* (*silly window syndrome* o *SWS* por sus siglas en inglés) y se convirtió en una plaga en las primeras implantaciones del TCP. En resumen:

> *Las primeras implantaciones del TCP presentaron un problema conocido como* síndrome de ventana tonta *en el cual cada acuse de recibo anunciaba una pe-*

*queña cantidad de espacio disponible y cada segmento transportaba una pequeña cantidad de datos.*

## 13.30 Prevención del síndrome de ventana tonta

Las especificaciones del TCP incluyen ahora la heurística necesaria para prevenir el síndrome de ventana tonta. La heurística utilizada en una máquina emisora evita la transmisión de cantidades pequeñas de datos en cada segmento. Otra heurística empleada en la máquina receptora evita la emisión de incrementos pequeños en los anuncios de ventana que pueden activar paquetes de datos pequeños. Aun cuando las heurísticas juntas trabajan bien, tanto el receptor como el emisor evitan el síndrome de ventana tonta ayudando a asegurar un buen desempeño en caso de que uno de los extremos falle en la correcta implantación del procedimiento para evitar las ventanas tontas.

En la práctica el software TCP debe contener el código necesario para evitar el síndrome de ventana tonta, tanto en el emisor como en el receptor. Para entender por qué, recordemos que la conexión del TCP es de tipo full duplex los datos pueden fluir en ambas direcciones. Así, una implementación del TCP incluye tanto el código para enviar datos como el código para recibirlos.

### 13.30.1 Prevención de la ventana tonta en el lado del receptor

La heurística que utiliza un receptor para evitar las ventanas tontas es fácil de entender. En general, un receptor mantiene un registro interno de la ventana disponible en el momento, pero retarda los anuncios para incrementar el tamaño de la ventana del emisor hasta que la ventana pueda avanzar una cantidad significativa. La definición de "significativa" depende del tamaño de la memoria intermedia del receptor y del tamaño del segmento máximo. El TCP lo define como el mínimo de la mitad de la memoria intermedia del receptor o el número de octetos de datos en un segmento de tamaño máximo.

El procedimiento para evitar las ventanas tontas en el lado del receptor evita el anuncio de ventanas pequeñas en caso de que una aplicación de recepción extraiga octetos de datos lentamente. Por ejemplo, cuando la memoria intermedia de un receptor se llena por completo, envía un acuse de recibo que contiene un anuncio de ventana en 0. Conforme la aplicación del receptor extrae octetos de la memoria intermedia, el receptor TCP calcula nuevamente el espacio disponible en la memoria intermedia. En lugar de enviar el anuncio de una ventana inmediatamente, el receptor espera hasta que se logre un espacio disponible equivalente a la mitad del tamaño de la memoria intermedia o equivalente al segmento de tamaño máximo. Así, el emisor siempre recibirá incrementos extensos en la ventana actual, permitiendo la transferencia de segmentos grandes. La heurística puede resumirse en la siguiente forma:

> *Procedimiento para evitar ventanas tontas en el lado del receptor: antes de enviar el anuncio de una ventana actualizada, luego de anunciar una ventana igual a 0, esperar hasta que se obtenga un espacio disponible que sea equivalente a por lo menos el 50% del tamaño total de la memoria intermedia o igual al segmento de tamaño máximo.*

### 13.30.2 Acuses de recibo retardados

Se han tomado dos enfoques para implantar la prevención de las ventanas tontas en el lado del receptor. En el primer método, el TCP acusa de recibido cada segmento que llega pero no anuncia un incremento en sus ventanas hasta que la ventana alcanza el límite especificado por la heurística para la prevención de las ventanas tontas. En el segundo método, el TCP retarda el envío de un acuse de recibo cuando la prevención de las ventanas tontas especifica que la ventana no es lo suficientemente grande como para anunciarse. Los estándares recomiendan retrasar los acuses de recibo.

El retraso de los acuses de recibo tiene ventajas y desventajas. La mayor ventaja reside en que el retardo de los acuses de recibo puede reducir el tráfico y, por lo tanto, mejorar el desempeño. Por ejemplo, si llegan datos adicionales durante el periodo de retardo, un solo acuse de recibo reconocerá a todos los datos recibidos. Si la aplicación de recepción genera una respuesta inmediata después de la llegada de los datos (por ejemplo, un eco de caracteres en una sesión remota en línea), un pequeño retardo puede permitir que el acuse de recibo incorpore un segmento de datos. Sin embargo, el TCP no puede cambiar esta ventana hasta que la aplicación de recepción extraiga los datos desde la memoria intermedia. En los casos en que la aplicación de recepción lee los datos conforme éstos llegan, un corto retardo permite al TCP enviar un solo segmento de acuse de recibo de los datos y anunciar una actualización de ventana. Dentro del acuse de recibo retardado, el TCP reconocerá la llegada de datos inmediatamente y después enviará un acuse de recibo adicional para actualizar el tamaño de la ventana.

La desventaja del retardo en los acuses de recibo debería ser clara. Un aspecto importante es que, si un receptor retarda los acuses de recibo por mucho tiempo, el emisor TCP retransmitirá el segmento. Las retransmisiones innecesarias reducen el desempeño debido a que desperdician ancho de banda de la red. Además, las retransmisiones ocasionan una sobrecarga en las máquinas de emisión y recepción. Además, el TCP utiliza la llegada de los acuses de recibo para estimar los tiempos de viaje redondo; el retardo en los acuses de recibo puede provocar una estimación confusa y hacer que los tiempos de retransmisión sean demasiado largos.

Para evitar problemas potenciales, el estándar TCP define un límite para el tiempo de los retardos de acuse de recibo. Las implantaciones no pueden retrasar un acuse de recibo por más de 500 milisegundos. Además, para garantizar que el TCP reciba un número suficiente de estimaciones de viaje redondo, el estándar recomienda que un receptor debe acusar de recibido por lo menos cada segmento de datos diferente.

### 13.30.3 Prevención de la ventana tonta del lado del emisor

La heurística que un emisor TCP utiliza para evitar las ventanas tontas es sorprendente y elegante. Recordemos que el objetivo es evitar el envío de segmentos pequeños. También no olvidemos que una aplicación de emisión puede generar datos en bloques arbitrariamente pequeños (por ejemplo, de un octeto). Así, para lograr este objetivo, un emisor TCP debe permitir a la aplicación del emisor hacer múltiples llamadas a *write* y debe reunir los datos transferidos con cada llamada antes de transmitirlos a un solo segmento largo. Es decir que un emisor TCP debe retardar el envío de un segmento hasta que pueda acumular una cantidad razonable de datos. Esta técnica se conoce con el nombre de *clumping* (*agrupamiento*).

La cuestión es la siguiente, ¿qué tanto debe esperar el TCP antes de transmitir datos? Por un lado, si el TCP espera demasiado la aplicación tendrá retardos demasiado largos. Algo muy importante es que el TCP no puede saber si tiene que esperar pues no puede saber si la aplicación generará más datos en un futuro cercano. Por otro lado, si el TCP no espera lo suficiente, los segmentos serán pequeños y el desempeño se reducirá.

Los primeros protocolos diseñados para el TCP enfrentaron el mismo problema y utilizaron técnicas para agrupar datos en paquetes grandes. Por ejemplo, para obtener una transferencia eficaz a través de una red, los protocolos de terminal remota originales retrasaban la transmisión de cada pulsación de tecla por unos pocos cientos de milisegundos para determinar si el usuario continuaba presionando la tecla. Como el TCP está diseñado con propósitos generales, puede usarse para un conjunto diverso de aplicaciones. Los caracteres pueden viajar a través de una conexión TCP dado que un usuario pulsa una tecla o dado que un programa transfiere archivos. Un retraso fijo no es óptimo para todas las aplicaciones.

Como en los algoritmos del TCP utilizados para la retransmisión y el algoritmo de comienzo lento empleado para evitar el congestionamiento, la técnica que un emisor TCP utiliza para evitar el envío de paquetes pequeños es flexible —el retraso depende del desempeño actual de la red de redes. Como en el comienzo lento, la prevención de la ventana tonta en el lado del emisor se conoce como *self clocking* pues no se calculan retardos. Por el contrario, el TCP utiliza la llegada de un acuse de recibo para disparar la transmisión de paquetes adicionales. Esta heurística se puede resumir de la siguiente forma:

> *Procedimiento para evitar la ventana tonta del lado del emisor: cuando una aplicación de emisión genera datos adicionales para enviarse sobre una conexión por la que se han transmitido datos anteriormente, pero de los cuales no hay un acuse de recibo, debe colocarse los datos nuevos en la memoria intermedia de salida como se hace normalmente, pero no enviar segmentos adicionales hasta que se reúnan suficientes datos para llenar un segmento de tamaño máximo. Si todavía se está a la espera cuando llega un acuse de recibo, debe enviarse todos los datos que se han acumulado en la memoria intermedia. Aplíquese la regla incluso cuando el usuario solicita la operación de* empujar.

Si una aplicación genera datos un octeto por vez, el TCP enviará el primer octeto inmediatamente. Sin embargo, hasta que llegue el ACK, el TCP acumulará octetos adicionales en su memoria intermedia. Así, si la aplicación es razonablemente rápida, comparada con la red (por ejemplo, en una transferencia de archivo), los segmentos sucesivos contendrán muchos octetos. Si la aplicación es lenta en comparación con la red (por ejemplo, un usuario que pulsa un teclado), se enviarán segmentos pequeños sin retardos largos.

Conocida como *algoritmo Nagle,* en honor a quien la inventó, esta técnica es especialmente elegante pues requiere de una pequeña carga computacional. Un anfitrión no necesita conservar temporizadores separados para cada conexión ni hacer que el anfitrión examine un reloj cuando una aplicación genera datos. Algo muy importante, a través de esta técnica se logra la adaptación a combinaciones arbitrarias de retardos de red, tamaños de segmento máximo y velocidad de aplicaciones, lo cual no disminuye el desempeño en casos convencionales.

Para entender por qué el desempeño se conserva para las comunicaciones convencionales, observemos que las aplicaciones optimizadas para altos desempeños no generan datos un octeto a

la vez (de hacerlo así se incurriría en sobrecargas innecesarias del sistema operativo). De hecho, cada aplicación escribe grandes bloques de datos con cada llamada. Así, la memoria intermedia de salida del TCP tiene suficientes datos para al menos un segmento de tamaño máximo. Además, como la aplicación produce datos con mayor rapidez comparado a como el TCP los puede transferir, la memoria intermedia del emisor se mantiene casi llena y el TCP no tiene retardos de transmisión. Como resultado , el TCP continúa enviando segmentos en la medida en que la red de redes lo puede tolerar mientras la aplicación continúe llenando la memoria intermedia. En resumen:

> *El TCP ahora requiere que el emisor y el receptor implanten heurísticas que eviten el síndrome de ventana tonta. Un receptor evita anunciar ventanas pequeñas y un emisor utiliza un esquema flexible para retardar la transmisión y, así, agrupar datos dentro de segmentos largos.*

## 13.31 Resumen

El Protocolo de Control de Transmisión (TCP por sus siglas en inglés) define un servicio clave proporcionado para una red de redes llamado entrega de flujo confiable. El TCP proporciona una conexión tipo full duplex entre dos máquinas, lo que les permite intercambiar grandes volúmenes de datos de manera eficaz.

Dado que utiliza un protocolo de ventana deslizable, el TCP puede hacer eficiente el uso de la red. Como se hacen pocas suposiciones sobre el sistema de entrega subyacente, el TCP es lo suficientemente flexible como para operar sobre una gran variedad de sistemas de entrega. Ya que proporciona un control de flujo, el TCP permite que el sistema cuente con una amplia variedad de velocidades para la comunicación.

La unidad básica de transferencia utilizada por el TCP es un segmento. Los segmentos se emplean para transferir datos o información de control (por ejemplo, para permitir que el software TCP en dos máquinas establezca o interrumpa la comunicación). El formato de los segmentos permite a una máquina incorporar un acuse de recibo para datos que fluyen en una dirección, incluyéndolos en el encabezado del segmento de datos que fluyen en el sentido opuesto.

El TCP implanta el control de flujo estableciendo, en el anuncio del receptor, la cantidad de datos que está dispuesto a aceptar. También soporta mensajes fuera de banda utilizando una capacidad de datos urgentes y forzando la entrega por medio de un mecanismo de empuje.

El estándar TCP actual especifica un retroceso exponencial para los temporizadores de retransmisión y algoritmos de prevención de congestionamiento como el de arranque lento, disminución multiplicativa e incremento aditivo. Además, el TCP se vale de procedimientos heurísticos para evitar la transferencia de paquetes pequeños.

## PARA CONOCER MÁS

El estándar para TCP puede encontrarse en Postel (RFC 793); Braden (RFC 1122) contiene una actualización que aclara varios puntos. Clark (RFC 813) describe la administración de ventanas TCP,

Clark (RFC 816) describe las fallas en el aislamiento y la recuperación y Postel (RFC 879) reporta l tamaño de segmento máximo de TCP. Nagle (RFC 896) comenta la congestión en las redes TCP/IP y explica el efecto del cronometrado autónomo en el procedimiento para evitar las ventanas tontas. Karn y Partridge (1987) analizan la estimación del tiempo de viaje redondo y presentan el algoritmo de Karn. Jacobson (1988) presenta el algoritmo de control de congestión que ahora es una parte necesaria del estándar. Tomlinson (1975) considera el saludo de tres etapas con mayor detalle. Mills (RFC 889) reporta mediciones de retardos de viaje redondo de Internet. Jain (1986) describe el control de congestión, basado en el temporizador, en el ambiente de una ventana tonta. Borman (abril 1989) resume experimentos con el TCP de alta velocidad en computadoras Cray.

## EJERCICIOS

**13.1** El TCP utiliza un campo finito para contener números de secuencia de flujo. Estudie las especificaciones del protocolo para que descubra cómo éste permite a un flujo de longitud arbitraria pasar de una máquina a otra.

**13.2** Las notas de texto de una de las opciones del TCP permiten a un receptor especificar el tamaño de segmento máximo que está dispuesto a aceptar. ¿Por qué el TCP soporta una opción para especificar el tamaño de segmento máximo cuando también tiene un mecanismo de anuncio de ventana?

**13.3** ¿Bajo qué condiciones de retardo, ancho de banda, carga y pérdida de paquetes el TCP retransmitirá volúmenes de datos significativos innecesariamente?

**13.4** Los acuses de recibo del TCP perdidos no necesariamente obligan a una retransmisión. Explique por qué.

**13.5** Experimente con máquinas locales para determinar como el TCP maneja la reiniciación de una máquina. Establezca una conexión (por ejemplo, un enlace remoto) y déjela inactiva. Espere a que la máquina de destino interrumpa y reinicialice, y entonces fuerce a la máquina local a enviar un segmento TCP (por ejemplo, tecleando caracteres hacia la conexión remota).

**13.6** Imagine una implantación de TCP que descarte segmentos que llegan fuera de orden, incluso si éstos caen en la ventana actual. Esto es, la versión imaginada sólo aceptará segmentos que extiendan el flujo de octetos que ya ha recibido. ¿Trabajará? ¿Cómo se compara con una implantación TCP estándar?

**13.7** Considere el cálculo de una suma de verificación TCP. Asuma que, aun cuando el campo de suma de verificación en el segmento *no* ha sido puesto en 0, el resultado del cómputo de la suma de verificación es 0. ¿Qué se puede concluir de ello?

**13.8** ¿Cuáles son los argumentos a favor y en contra del cierre automático de una conexión inactiva?

**13.9** Si dos programas de aplicación utilizan el TCP para enviar datos, pero sólo envían un carácter por segmento (por ejemplo, utilizando la operación PUSH), ¿cuál es el máximo porcentaje del ancho de banda de la red que se tendrá para los datos?

**13.10** Supongamos que una implantación del TCP emplea un número de secuencia inicial *1* cuando se hace una conexión. Explique cómo un sistema que ha sido interrumpido y reiniciado puede confundir un sistema remoto en el supuesto de que la conexión anterior se mantiene abierta.

**13.11** Observe el algoritmo de estimación de tiempo de viaje redondo sugerido, en la especificación del protocolo ISO TP-4, y compárelo con el algoritmo TCP analizado en este capítulo. ¿Cuál preferiría utilizar?

**13.12** Averigüe cómo deben resolver las implantaciones del TCP el *problema de la superposición de segmentos*. El problema se presenta porque el receptor debe recibir sólo una copia de todos los octetos desde el flujo de datos, incluso si el emisor transmite dos segmentos que parcialmente se sobreponen uno sobre otro (por ejemplo, el primer segmento transfiere los octetos 100 a 200 y un segundo transporta los octetos 150 a 250).

**13.13** Siga la trayectoria de las transiciones de la máquina de estado finito TCP para las localidades que ejecutan una apertura pasiva y activa, asimismo siga los pasos a través del saludo de tres etapas.

**13.14** Lea la especificación TCP para encontrar las condiciones exactas bajo las que el TCP puede hacer la transición de *FIN WAIT-1* hacia *TIME WAIT*.

**13.15** Siga las transiciones de estado del TCP para dos máquinas que acuerdan cerrar una conexión cortésmente.

**13.16** Supongamos que el TCP está enviando segmentos mediante un tamaño de ventana máximo (64 Gigaoctetos) en un canal que tiene un ancho de banda infinito y un tiempo de viaje redondo promedio de 20 milisegundos. ¿Cuál es el máximo desempeño? ¿Cómo cambiará el desempeño si el tiempo de viaje redondo se incrementa a 40 milisegundos (mientras que el ancho de banda se mantiene infinito)?

**13.17** ¿Podría usted derivar una ecuación que exprese el desempeño máximo posible del TCP como una función del ancho de banda de la red, el retardo de la red y el tiempo para procesar un segmento y generar un acuse de recibo? Sugerencia: considere el ejercicio anterior.

**13.18** Describa las circunstancias (anormales) por las que el extremo de una conexión puede quedar indefinidamente en un estado *FIN WAIT-2*. Sugerencia: piense en pérdidas de datagramas y en caídas de sistemas.

incluyendo las rutas por omisión. Así, es posible hacer que dos ruteadores anuncien una ruta por omisión a diferentes métricas (esto es una ruta hacia el resto de la red de redes), haciendo una de ellas de ruta primaria y la otra de ruta de respaldo.

El campo final en cada entrada de información en un mensaje RIP, *DISTANCE TO NET i*, contiene un contador entero de la distancia hacia la red especificada. La distancia es medida en saltos de ruteador, pero los valores están limitados al rango entre *1* y *16*, con la distancia *16* utilizada para dar a entender una distancia infinita (esto significa que la ruta no existe).

### 16.3.4 Transmisión de mensajes RIP

Los mensajes RIP no contienen un campo de longitud explícito. De hecho, RIP asume que los mecanismos de entrega subyacentes dirán al receptor la longitud de un mensaje entrante. En particular, cuando se utiliza con el TCP/IP, los mensajes RIP dependen del UDP para informar al receptor la longitud del mensaje. RIP opera el puerto *520* en UDP. Aun cuando una solicitud RIP puede originar otro puerto UDP, el puerto de destino UDP para solicitudes es siempre *520*, que es el puerto de origen desde el cual en principio RIP difunde los mensajes.

El uso de RIP como protocolo de ruteo interior limita el ruteo a una métrica basada en contadores de saltos. Casi siempre los contadores de saltos proporcionan sólo una medición general de la respuesta de red o de la capacidad que no produce rutas óptimas. Además, calcular rutas con base en el conteo mínimo de saltos tiene la severa desventaja de que hace el ruteo relativamente estático, dado que las rutas no pueden responder a los cambios en las cargas de la red.

## 16.4 Protocolo Hello

El protocolo *HELLO* proporciona un ejemplo de un IGP que utiliza una métrica de ruteo basada en retardos en la red en lugar de contadores de saltos. A pesar de que ahora HELLO es obsoleto, es importante en la historia de Internet porque fue el IGP empleado entre los primeros ruteadores "fuzzball" de la columna vertebral NSF net. HELLO es importante para nosotros porque proporciona un ejemplo de un algoritmo vector-distancia que no utilizan contadores de saltos.

HELLO proporciona dos funciones: sincroniza los relojes entre un conjunto de máquinas y permite que cada máquina calcule las rutas de trayecto más corto hacia su destino. Así, los mensajes HELLO transportan información de sello de hora así como información de ruteo. La idea básica oculta o subyacente en HELLO es sencilla: cada máquina participante en el intercambio HELLO mantiene una tabla de sus mejores estimaciones de los relojes en las máquinas vecinas. Antes de transmitir un paquete, una máquina añade su sello de hora copiando el valor de reloj actual dentro del paquete. Cuando un paquete llega, el receptor calcula el retardo actual en el enlace. Para hacerlo, el receptor sustrae el sello de hora en el paquete entrante de su valor estimado para el reloj actual en el vecino. De manera periódica, las máquinas sondean a sus vecinos a fin de restablecer sus estimaciones para los relojes.

Los mensajes HELLO también permiten a las máquinas participantes calcular nuevas rutas. El algoritmo trabaja en forma parecida a RIP, pero utiliza retardos en lugar de contadores de salto. Cada máquina envía periódicamente a su vecino una tabla de los retardos estimados para todas las

otras máquinas. Supongamos que la máquina *A* envía a la máquina *B* una tabla de ruteo que especifica destinos y retardos. *B* examina cada entrada de información en la tabla. Si los retardos actuales de *B* para alcanzar un destino dado, *D*, son mayores que el retardo desde *A* hasta *B* más el retardo desde *B* hasta *A*, *B* cambia su ruta y envía el tráfico hacia *D* vía *A*. Esto es, *B* rutea el tráfico hacia *A* y toma la trayectoria de retraso más corto.

Como en cualquier algoritmo de ruteo, HELLO no puede cambiar rutas rápidamente o se volvería inestable. La inestabilidad en un algoritmo de ruteo produce un efecto de oscilación de dos estados en el cual el tráfico conmuta "de ida y de regreso" entre rutas alternas. En el primer estado, la máquina encuentra una trayectoria ligeramente cargada y de manera abrupta conmuta el tráfico hacia ésta, sólo para encontrar que comienza a estar completamente sobrecargada. En el segundo estado, la máquina conmuta el tráfico de regreso de la ruta sobrecargada, sólo para encontrar que la trayectoria comienza a sobrecargarse, y el ciclo continúa. Estas oscilaciones pueden presentarse. Para evitarlas, las implantaciones de HELLO seleccionan cambiar rutas sólo cuando la diferencia en el retardo es grande.

La figura 16.6 muestra el formato del mensaje HELLO. El protocolo es más complejo que el formato de mensaje indicado puesto que distingue las conexiones de redes locales de los saltos múltiples hacia afuera, los límites de tiempo caducan en entradas de información en las tablas de ruteo y utiliza identificadores locales para los anfitriones en lugar de direcciones IP completas.

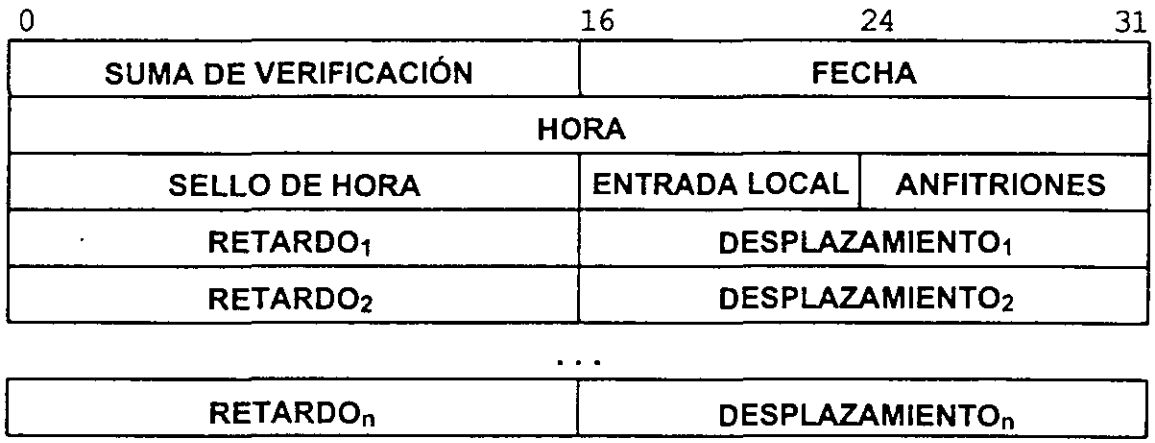



**Figura 16.6** Formato del mensaje HELLO. Cada mensaje transporta una entrada de datos para la fecha y la hora, así como un sello de hora que el protocolo utiliza para estimar los retardos de red.

El campo *CHECK SUM* (*VERIFICACIÓN DE SUMA*) contiene una suma de verificación relacionada con el mensaje, el campo *DATE* (*FECHA*) contiene la fecha local del emisor y el campo *TIME* contiene la hora local de acuerdo al reloj del emisor. El campo *TIMES STAMP* (*SELLO DE HORA*) se utiliza para calcular el ciclo completo o (viaje redondo).

El campo con el nombre *#HOSTS* especifica cuántas entradas de información siguen en la lista de anfitriones y el campo llamado *LOCAL ENTRY* (*ENTRADA LOCAL*) apunta hacia la lista para marcar el bloque de entradas de información utilizadas por la red local. Cada introducción de información contiene dos campos, *DELAY* (*RETRASO*) y *OFFSET* (*DESPLAZAMIENTO*), que

proporcionan el retraso para alcanzar un anfitrión y la estimación actual del emisor respecto a la diferencia entre el reloj del anfitrión y el reloj del emisor.

## 16.5 Combinación de RIP, Hello y EGP

Hemos observado ya que un solo ruteador puede usar tanto un IGP para asociar información de ruteo dentro de su sistema autónomo como un EGP para anunciar rutas hacia otros sistemas autónomos. En principio sería fácil construir una sola pieza de software que combinara los dos protocolos e hiciera posible asociar rutas y anunciarlas sin la intervención humana. En la práctica obstáculos técnicos y políticos hacen que esto sea muy complejo.

Técnicamente, tanto los protocolos IGP como RIP y HELLO son protocolos de ruteo. Un ruteador utiliza estos protocolos para actualizar su tabla de ruteo con base en la información que adquiere de otros ruteadores ubicados en su sistema autónomo. A diferencia de los protocolos de ruteo interior, el EGP trabaja además con la tabla de ruteo usual de un ruteador. Un ruteador utiliza el EGP para comunicar información de accesibilidad hacia otros sistemas autónomos, independientemente de su propia tabla de ruteo. Así, *routed*, el programa UNIX que implanta RIP, anuncia información desde la tabla de ruteo local y cambia la tabla de ruteo local cuando recibe actualizaciones. Dependen de estas máquinas que utilizan RIP para transferir los datos correctamente. En contraste, el programa que implanta el EGP no anuncia rutas desde la tabla de ruteo local; mantiene una base de datos separada de accesibilidad de redes.

Un ruteador que utiliza el EGP para anunciar accesibilidad debe tener cuidado de difundir sólo las rutas que tiene autorizado anunciar o podría afectar otras partes de la red de redes. Por ejemplo, si un ruteador en un sistema autónomo logra difundir una distancia de ruta *0* para una red en la Universidad de Purdue cuando no tiene esta ruta, RIP instalará la ruta en otras máquinas y comenzará a pasar el tráfico dirigido a Purdue hacia el ruteador que inició el error. Como resultado, será imposible para algunas máquinas en el sistema autónomo llegar a Purdue. Si el EGP propaga este error fuera del sistema autónomo, podría ser imposible alcanzar Purdue desde algunas partes de la red de redes.

El programa *gated*[6] combina múltiples IGP y EGP de acuerdo a un conjunto de reglas que restringen las rutas anunciadas hacia los ruteadores exteriores. Por ejemplo, *gated* puede aceptar mensajes RIP y modificar la tabla de ruteo de las computadoras locales como sucedería con el programa *routed*. Puede anunciar rutas desde el interior de su sistema autónomo utilizando el EGP. Las reglas permiten a un administrador de sistema especificar exactamente qué redes *gated* podrán y no podrán anunciar y cómo reportarán distancias para estas redes. Así, aun cuando *gated* no es un IGP, juega un papel importante en el ruteo porque demuestra que es factible construir un mecanismo automatizado enlazando un IGP con un EGP sin sacrificar la seguridad.

*Gated* realiza otra tarea útil al implantar transformaciones métricas. Recordemos del capítulo 15 que las extensiones para EGP permiten a los sistemas autónomos tomar decisiones de ruteo inteligentes, mientras todos los ruteadores que utilizan EGP acuerden una interpretación amplia de la métrica de distancias. En particular, el ruteador dentro de un sistema autónomo debe acordar el uso de valores de distancia más allá de un umbral fijo, digamos 128. Cada vez que un ruteador exterior

[6] El nombre *gated* se pronuncia "gate d" de "gate daemon".

# 20

# La interfaz socket

## 20.1 Introducción

Hasta aquí, nos hemos concentrado en tratar los principios y conceptos que sustentan los protocolos TCP/IP sin especificar la interfaz que existe entre los programas de aplicación y el software de protocolo. En este capítulo, veremos el ejemplo de una interfaz entre programas de aplicación y protocolos TCP/IP. Existen dos razones para posponer el análisis acerca de las interfaces. En primer lugar, debemos distinguir entre los protocolos de interfaz y el TCP/IP debido a que los estándares no especifican exactamente cómo es que interactúan los programas de aplicación con el software de protocolo. Por ello, la arquitectura de interfaz no está estandarizada; su diseño descansa fuera del campo de lo relacionado con el protocolo. En segundo lugar, en la práctica, es inapropiado unir a los protocolos con una interfaz en particular pues ninguna arquitectura de interfaz funciona bien en todos los sistemas. En particular, como el software de protocolo reside en el sistema operativo de una computadora, los detalles de la interfaz dependen del sistema operativo.

A pesar de la carencia de un estándar, la revisión de un ejemplo nos ayudará a comprender cómo es que emplean los programadores el TCP/IP. Aunque el ejemplo que hemos escogido es del sistema operativo de BSD de UNIX, se ha aceptado ampliamente y se usa en muchos otros sistemas. En particular, la interfaz *Winsock* proporciona la funcionalidad socket para Microsoft Windows. El lector deberá recordar que nuestra meta es tan sólo dar un ejemplo concreto y no prescribir cómo es que deberían estar diseñadas las intefaces. El lector no deberá olvidar tampoco que las operaciones listadas aquí no comprenden una estandarización en ningún sentido.

## 20.2 El paradigma E/S de UNIX y la E/S de la red

Unix fue desarrollado a finales de los años sesenta y principios de los setenta, y se diseñó originalmente como un sistema de tiempo compartido para computadoras de un solo procesador. Se trata de un sistema operativo orientado al proceso, en el que cada programa de aplicación se ejecuta como un proceso de nivel de usuario. Un programa de aplicación interactúa con el sistema operativo haciendo *llamadas de sistema*. Desde el punto de vista del programador, las llamadas de sistema se ven y comportan exactamente igual que las demás llamadas de procedimiento. Toman argumentos y devuelven uno o más resultados. Los argumentos pueden ser valores (por ejemplo, una operación de enteros) o punteros a objetos en el programa de aplicación (como un búfer que ha de ser llenado con caracteres).

Derivados de los Multics y los sistemas anteriores, los sistemas primitivos de entrada y salida (E/S, ENTRADA/SALIDA) de UNIX siguen un paradigma que algunas veces se denomina *open-read-write-close* (*abrir-leer-escribir-cerrar*). Antes de que un proceso de usuario pueda ejecutar operaciones de E/S, llama a *open* para especificar el archivo o dispositivo que se va a usar y obtiene el permiso. La llamada a *open* devuelve un pequeño entero *descriptor de archivo*[1] que el proceso utiliza cuando ejecuta las operaciones de E/S en el archivo o dispositivo abierto. Una vez que se ha abierto un objeto, el proceso de usuario hace una o más llamadas a *read* o *write* para transferir datos. *Read* transfiere datos dentro del proceso de usuario; *write* transfiere datos del proceso de usuario al archivo o dispositivo. Tanto *read* como *write* toman tres argumentos que especifican el descriptor de archivo que se ha de usar, la dirección de un búfer y el número de octetos que se han de transferir. Luego de completar todas las operaciones de transferencia, el proceso de usuario llama a *close* para informar al sistema operativo que ha terminado de usar el objeto (el sistema operativo cierra de manera automática todos los descriptores abiertos si ningún proceso llama a *close*).

## 20.3 Adición de la red E/S a UNIX

Originalmente, los diseñadores de UNIX agruparon todas las operaciones de E/S en el paradigma open-read-write-close descrito arriba. El esquema incluyó E/S para los dispositivos orientados a caracteres como los teclados y para los dispositivos orientados a bloques como los discos y los archivos de datos. Una de las primeras implantaciones del TCP/IP bajo UNIX también utilizó el paradigma open-read-write-close con un nombre de archivo especial, */dev/tcp*.

El grupo que añadió los protocolos de red a BSD de UNIX decidió que, como los protocolos de la red eran más complejos que los dispositivos convencionales de E/S, la interacción entre los procesos del usuario y los protocolos de red debía ser más compleja que las interacciones entre los procesos de usuario y las instalaciones convencionales de E/S. En particular, la interfaz de protocolo debía permitir a los programadores crear un código de servidor que esperara las conexiones pasivamente, así como también un código cliente que formara activamente las conexiones. Además, los programas de aplicación que mandaban datagramas podían especificar la dirección de destino

[1] El término "descriptor de archivo" surge porque, en UNIX, todos los dispositivos son transformados en el archivo de espacio de nombre de sistema. En la mayor parte de los casos, las operaciones de E/S en los archivos y dispositivos no se pueden distinguir

junto con cada datagrama en lugar de destinos enlazados en el momento en que llamaban a *open*. Para manejar todos estos casos, los diseñadores eligieron abandonar el paradigma tradicional de UNIX open-read-write-close y añadir varias llamadas nuevas del sistema operativo, así como también una nueva biblioteca de rutinas. La adición de protocolos de red a UNIX incrementó sustancialmente la complejidad de la interfaz de E/S.

Posteriormente surgió una mayor complejidad en la interfaz de protocolos de UNIX pues los diseñadores intentaron construir un mecanismo general para adaptar muchos protocolos. Por ejemplo, la generalidad hace posible, para el sistema operativo, incluir software para otros conjuntos de protocolos así como también el TCP/IP y permitir que un programa de aplicación utilice uno o más de ellos a la vez. Como consecuencia, el programa de aplicación no sólo puede proporcionar una dirección de 32 bits y esperar a que el sistema operativo la interprete de manera correcta. La aplicación debe especificar explícitamente que el número de 32 bits representa una dirección IP.

## 20.4 La abstracción de socket

La base para la E/S de red en BSD de UNIX se centra en una abstracción conocida como *socket*.[2] Pensamos al socket como una generalización del mecanismo de acceso a archivos de UNIX que proporciona un punto final para la comunicación. Al igual que con el acceso a archivos, los programas de aplicación requieren que el sistema operativo cree un socket cuando se necesita. El sistema devuelve un entero pequeño que utiliza el programa de aplicación para hacer referencia al socket recientemente creado. La diferencia principal entre los descriptores de archivos y los descriptores de socket es que el sistema operativo enlaza un descriptor de archivo a un archivo o dispositivo específico cuando la aplicación llama a *open*, pero puede crear sockets sin enlazarlos a direcciones de destino específicas. La aplicación puede elegir abastecer una dirección de destino cada vez que utiliza el socket (es decir, cuando se envían datagramas) o elegir enlazar la dirección de destino a un socket y evadir la especificación de destino repetidamente (es decir, cuando se hace una conexión TCP).

Cada vez que tenga sentido, los sockets hacen ejecuciones exactamente iguales a los archivos o dispositivos de UNIX, de manera que pueden utilizarse con operaciones tradicionales, como *read* y *write*. Por ejemplo, una vez que un programa de aplicación crea un socket y una conexión TCP del socket a un destino externo, el programa puede hacer uso de *write* para mandar un flujo de datos a través de la conexión (el programa de aplicación en el otro extremo puede utilizar *read* para recibirlo). Para hacer posible que se utilicen las primitivas *read* y *write* con archivos y sockets, el sistema operativo ubica los descriptores de socket y los descriptores de archivo del mismo conjunto de enteros y se asegura de que, si un entero dado se ha ubicado como descriptor de archivo, no será ubicado también como descriptor de socket.

[2] Por ahora describiremos a los sockets como parte del sistema operativo pues es la manera en que BSD de UNIX los proporciona; en secciones posteriores se describe cómo es que otros sistemas operativos utilizan las bibliotecas de rutinas para proporcionar una interfaz de socket.

## 20.5 Creación de un socket

El sistema de llamada *socket* crea sockets cuando se le pide. Toma tres argumentos enteros y devuelve un resultado entero:

resultado = socket (pf, tipo, protocolo)

El argumento *pf* especifica la familia del protocolo que se va a utilizar con el socket. Esto quiere decir que especifica cómo interpretar la dirección cuando se suministra. Las familias actuales incluyen la red de redes TCP/IP (PF_INET), la Red de redes PUP de Xerox Corporation (PF_PUP), la red Appletalk de Apple Computer Incorporated (PF_APPLETALK) y el sistema de archivos UNIX (PF_UNIX) así como también muchos otros.[3]

El argumento *tipo* especifica el tipo de comunicación que se desea. Los tipos posibles incluyen el servicio de entrega confiable de flujo (SOCK_STREAM) y el servicio de entrega de datagramas sin conexión (SOCK_DGRAM), así como también el tipo creado o no procesado (SOCK_RAW) que permite a los programas de privilegio accesar a los protocolos de bajo nivel o a las interfaces de red. Otros dos tipos adicionales ya se han planeado pero no han sido implantados.

Aunque el acercamiento general de separación de las familias y los tipos de protocolo puede parecer suficiente para manejar todos los casos con facilidad, no es así. En primer lugar puede ser que una familia de protocolos dada no soporte uno o mas de los tipos de servicio posibles. Por ejemplo, la familia UNIX tiene un mecanismo de comunicación de interproceso, llamado *pipe*, que utiliza un servicio de entrega de flujo confiable, pero no posee mecanismo de entrega de paquetes secuenciados. De este modo, no todas las combinaciones de familias de protocolos y tipos de servicio tienen sentido. En segundo lugar, algunas familias de protocolos poseen protocolos diversos que soportan un tipo de servicio. Por ejemplo podría ser que una sola familia de protocolos tuviera dos servicios de entrega de datagramas sin conexión. Para acomodar los diversos protocolos dentro de una familia, la llamada de *socket* tiene un tercer argumento, que se puede emplear para seleccionar un protocolo específico. Para usar el tercer argumento, el programador debe conocer la familia de protocolos lo suficientemente bien como para saber el tipo de servicio que cada protocolo brinda.

Como los diseñadores trataron de capturar muchas de las operaciones convencionales de UNIX en su diseño socket, necesitaban una manera de simular el mecanismo pipe. No es necesario comprender los detalles de los pipes; sólo una de las características es importante: los pipes difieren de las operaciones de red estándar pues el proceso de llamado crea ambos puntos de terminación para la comunicación de manera simultánea. Para acomodar los pipes, los diseñadores añadieron un sistema de llamado *socketpair* (*apareamiento de sockets*) que toma la siguiente forma:

socketpair(pf, tipo, protocolo, sarray)

*Socketpair* tiene un argumento más que en el procedimiento *socket*, se trata de *sarray*. Dicho argumento adicional da la dirección de un arreglo entero de dos elementos. *Socketpair* crea dos sockets de manera simultánea y coloca dos descriptores de socket en los dos elementos de *sarray*.

---

[3] En UNIX, los programas de aplicación contienen nombres simbólicos como *PF_INET*; los archivos de sistema contienen las definiciones que especifican valores numéricos para cada nombre.

Los lectores deberán comprender que *socketpair* no es significativo cuando se aplica a la familia de protocolos TCP/IP (lo incluimos aquí sólo para completar nuestra descripción de la interfaz).

## 20.6 Herencia y finalización del socket

UNIX utiliza las llamadas del sistema *fork* y *exec* para comenzar nuevos programas de aplicación. Se trata de un procedimiento de dos pasos. En el primer paso, *fork* crea una copia separada del programa de aplicación que se está ejecutando en ese momento. En el segundo paso, la nueva copia se reemplaza a sí misma con el programa de aplicación deseado. Cuando una programa llama a *fork*, la copia que se acaba de crear le hereda el acceso a todos los sockets abiertos, justo como si heredara el acceso a todos los archivos abiertos. Cuando el programa llama a *exec*, la nueva aplicación retiene el acceso para todos los sockets abiertos. Veremos que los servidores maestros utilizan la herencia socket cuando crean servidores esclavos o manejan una conexión específica. Internamente, el sistema operativo mantiene una cuenta de referencia asociada con cada socket de manera que se sabe cuántos programas de aplicación (procesos) han accedido a él.

Tanto los procesos viejos como los nuevos tienen los mismos derechos de acceso para los sockets existentes, y ambos tipos pueden accesar a los sockets. Luego pues, es responsabilidad del programador asegurar que los dos procesos empleen el significado del socket compartido.

Cuando un proceso termina de utilizar un socket, éste llama a *close* (*cerrar*). *Close* tiene la forma:

close(socket)

donde el argumento *socket* especifica al descriptor de un socket para que cierre. Cuando un proceso se termina por cualquier razón, el sistema cierra todos los sockets que permanecen abiertos. Internamente, una llamada a *close* (*cerrar*) disminuye la cuenta de referencia para un socket y destruye al socket si la cuenta llega a cero.

## 20.7 Especificación de una dirección local

Inicialmente, un socket se crea sin ninguna asociación hacia direcciones locales o de destino. Para los protocolos TCP/IP, esto significa que ningún número de puerto de protocolo local se ha asignado y que ningún puerto de destino o dirección IP se ha especificado. En muchos casos, los programas de aplicación no se preocupan por las direcciones locales que utilizan, ni están dispuestos a permitir que el software de protocolo elija una para ellos. Sin embargo los procesos del servidor que operan en un puerto bien conocido deben ser capaces de especificar dicho puerto para el sistema. Una vez que se ha creado un socket, el servidor utiliza una llamada del sistema *bind (enlace)* para establecer una dirección local para ello. *Bind* tiene la siguiente forma:

bind(socket, localaddr, addrlen)

El argumento *socket* es el descriptor de enteros del socket que ha de ser enlazado. El argumento *localaddr* es una estructura que especifica la dirección local a la que el socket deberá enlazarse, y el argumento *addrlen* es un entero que especifica la longitud de las direcciones medidas en octetos. En lugar de dar la dirección, solamente como una secuencia de octetos, los diseñadores eligieron utilizar una estructura para las direcciones, como se ilustra en la figura 20.1.

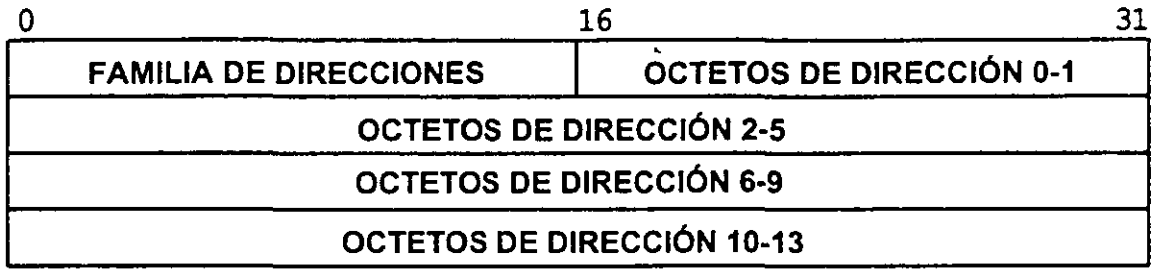



**Figura 20.1** La estructura *sockaddr* que se utiliza cuando se pasa una dirección TCP/IP para la interfaz socket.

La estructura, que genéricamente se denomina *sockaddr*, comienza como un campo *ADDRESS FAMILY* (*FAMILIA DE DIRECCIONES*) de 16 bits que identifica el conjunto de protocolos al que pertenece la dirección. Va seguido por una dirección de hasta *14* octetos. Cuando se declara en C, la dirección de la estructura de socket es una unión de estructuras para todas las familias de direcciones posibles.

El valor en el campo *ADDRESS FAMILY* determina el formato de los octetos de las direcciones restantes. Por ejemplo, el valor $2^4$ en el campo *ADDRESS FAMILY* significa que los octetos de direcciones restantes contienen una dirección TCP/IP. Cada familia de protocolos define cómo se usarán los octetos en el campo de dirección. Para las direcciones TCP/IP, la dirección de socket se conoce como *sockaddr_in*. Esto incluye una dirección del IP y un número de puerto de protocolo (es decir, una dirección de socket de red de redes cuya estructura puede contener direcciones IP y puertos de protocolo en la dirección). En la figura 20.2 se muestra el formato exacto de una dirección de socket TCP/IP.

Aunque es posible especificar valores arbitrarios en la estructura de la dirección cuando se llama a *bind*, no todos los enlaces posibles son válidos. Por ejemplo, quien llame podría pedir un puerto de protocolo local que ya esté en uso para otro programa, o podría pedir una dirección IP no válida. En tales casos, la llamada *bind* falla y se devuelve un código de error.

## 20.8 Conexión de socket con direcciones de destino

Inicialmente, un socket se crea en el *unconnected state* (*estado no conectado*), lo que significa que el socket no está asociado con ningún destino externo. La llamada de sistema *connect* (*conectar*)

[4] UNIX utiliza el nombre simbólico PF_INET para denotar direcciones TCP/IP.

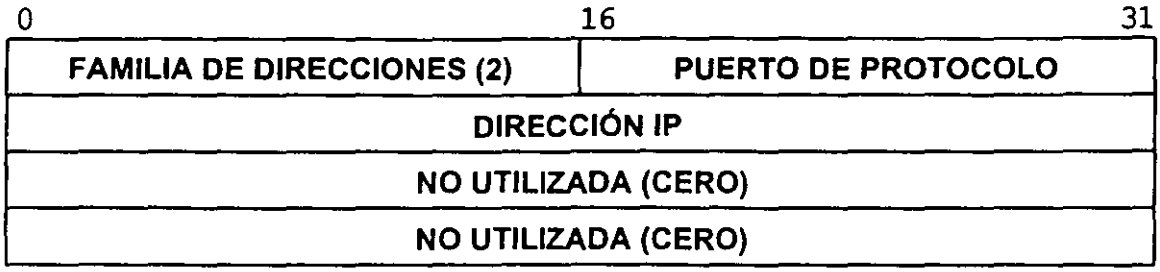



**Figura 20.2** El formato de una estructura de dirección de socket (*sockaddr_in*) cuando se usa con una dirección TCP/IP. La estructura incluye ambas, una dirección IP y un puerto de protocolo en la dirección.

enlaza un destino permanente a un socket, colocándolo en el *connected state* (*estado conectado*). Un programa de aplicación debe llamar a *connect* para establecer una conexión antes de que pueda transferir datos a través de un socket de flujo confiable. Los sockets utilizados con servicios de datagrama sin conexión necesitan no estar conectados antes de usarse, pero haciéndolo así, es posible transferir datos sin especificar el destino en cada ocasión.

La llamada de sistema *connect* tiene la forma:

connect(socket, destaddr, addrlen)

El argumento *socket* es el descriptor entero del socket que ha de conectarse. El argumento *destaddr* es una estructura de dirección socket en la que se especifica la dirección de destino a la que deberá enlazarse el socket. El argumento *addrlen* especifica la longitud de la dirección de destino medida en octetos.

El significado de *connect* depende de los protocolos subyacentes. La selección del servicio de entrega de flujo confiable en la familia PF_INET significa elegir al TCP. En tales casos, *connect* construye una conexión con el destino y devuelve un error si no puede. En el caso del servicio sin conexión, *connect* no hace nada más que almacenar localmente la dirección de destino.

## 20.9 Envío de datos a través de un socket

Una vez que el programa de aplicación ha establecido un socket, puede usar el socket para transmitir datos. Hay cinco llamadas posibles de sistemas operativos de entre las que se puede escoger: *send* (*mandar*), *sendto* (*mandar a*), *sendmsg* (*mandar mensaje*), *write* (*escribir*) y *writev* (*vector escribir*). *Send*, *write* y *writev* sólo trabajan con sockets conectados pues no permiten que quien llama especifique una dirección de destino. Las diferencias entre los tres son menores. *Write* toma tres argumentos:

write(socket, buffer, length)

El argumento *socket* contiene un descriptor de socket entero (*write* puede también usarse con otros tipos de descriptores). El argumento *buffer* contiene la dirección de datos que se han de man-

lar y el argumento *length* especifica el número de octetos que se han de mandar. La llamada a *write* se bloquea hasta que los datos se pueden transferir (es decir, se bloquea si los búfers del sistema interno para el socket están llenos). Como en la mayor parte de las llamadas de sistema en UNIX, *write* devuelve un código de error para la aplicación que llama, lo cual permite al programador saber si la operación tuvo éxito.

La llamada de sistema *writev* funciona como *write* salvo porque utiliza la forma "gather write (escritura de recolección)", la cual hace posible que el programa de aplicación escriba un mensaje sin copiar el mensaje en los octetos de memoria contiguos. *Writev* tiene la forma:

writev(socket, iovector, vectorlen)

El argumento *iovector* da la dirección de un arreglo de tipo *iovec*, el cual contiene una secuencia de apuntadores de los bloques de octetos que forman el mensaje. Como se muestra en la figura 20.3, a cada apuntador lo acompaña una longitud. El argumento *vectorlen* especifica el número de entradas en *iovector*.

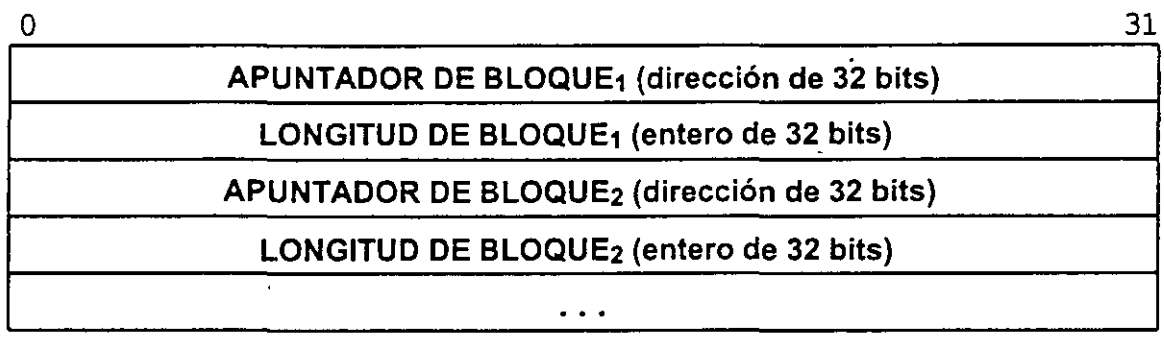



**Figura 20.3** Formato de un iovector (vector e/s) de tipo *iovec* que se usa con *writev (vector escribir)* y *readv (vector leer)*.

La llamada de sistema *send* tiene la forma:

send(socket, message, length, flags)

donde el argumento *socket* especifica el socket que se ha de usar, el argumento *message* (*mensaje*) da la dirección de datos a ser mandados, el argumento *length* (*longitud*) especifica el número de octetos que se va a enviar y el argumento *flags* (*banderas*) controla la transmisión. Un valor para *flags* permite que quien envía especifique que el mensaje se deberá enviar fuera de banda en los sockets que soporten tal noción. Por ejemplo, recordemos que, en el capítulo 13, se decía que los mensajes fuera de banda corresponden a la noción de datos urgentes del TCP. Otro valor para *flags* permite que quien llama pida que el mensaje se envíe sin necesidad de usar tablas de ruteo local. La intención es permitir a quien llama que tome el control del ruteo, haciendo posible que se escriba en el software de depuración de red. Por supuesto, no todos los sockets soportan todas las peticiones de cualquier programa. Algunas peticiones requieren que el programa tenga privilegios especiales y hay otros programas que sencillamente no se soportan en todos los sockets.

Las llamadas de sistema *sendto* y *sendmsg* permiten que quien llama envíe un mensaje a través de un socket no conectado, pues ambos requieren que quien llama especifique un destino. *Sendto*, que toma la dirección de destino como argumento, tiene la forma:

sendto(socket, message, length, flags, destaddr, addrlen)

Los primeros cuatro argumentos son exactamente los mismos que se usaron con la llamada de sistema *send*. Los dos argumentos finales especifican una dirección de destino y dan la longitud de dicha dirección. El argumento *destaddr* especifica la dirección de destino utilizando la estructura *sockaddr_in* como se define en la figura 20.2

Un programador puede elegir usar la llamada de sistema *sendmsg* en casos en los que la larga lista de argumentos requeridos para *sendto* haga que el programa sea ineficaz o difícil de leer. *Sendmsg* tiene la forma:

sendmsg(socket, messagestruct, flags)

donde el argumento *messagestruct* (*estructura de mensaje*) es una estructura de la forma ilustrada en la figura 20.4. La estructura contiene información acerca del mensaje que se ha de mandar, su longitud, la dirección de destino y la longitud de la dirección. Esta llamada es especialmente útil porque no hay una operación de entrada correspondiente (se describe abajo) que produzca una estructura de mensaje exactamente en el mismo formato.

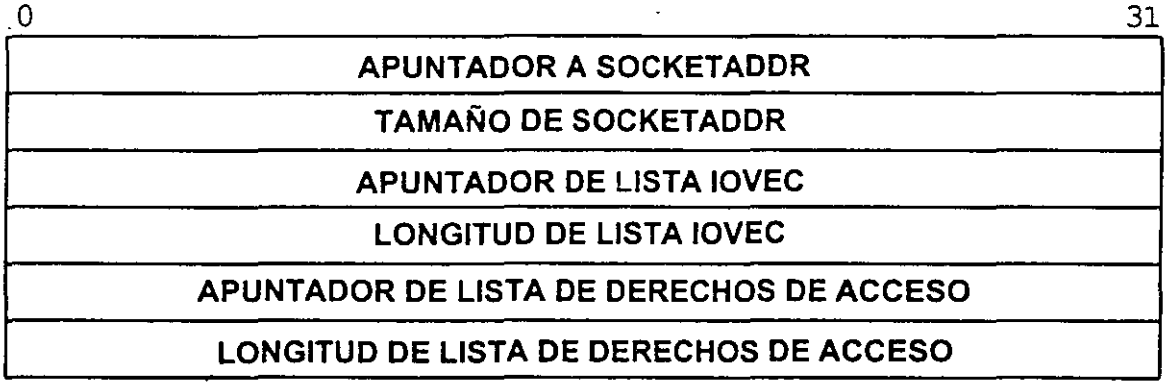



**Figura 20.4** Formato de la estructura de mensaje *messagestruct* utilizado por *sendmsg*.

## 20.10 Recepción de datos a través de un socket

BSD de UNIX ofrece cinco llamadas de sistema, análogas a las cinco diferentes operaciones de salida, que un proceso puede utilizar para recibir datos a través de un socket: *read* (*leer*), *ready* (*listo*), *recv* (*recibir*), *recvfrom* (*recibir de*) y *recvmsg (recibir mensaje)*. La operación de entrada convencional de UNIX, *read*, sólo se puede usar cuando un socket se conecta. Este tiene la forma:

read(descriptor, buffer, length)

donde el *descriptor* da el descriptor entero de un socket o el descriptor de archivo del que se puede leer los datos, el *buffer* especifica la dirección en memoria en la que se almacenan los datos y *length* especifica el número máximo de octetos que puedan leerse.

Una forma alternativa, *readv*, permite que quien llama utilice un estilo de interfaz de "scatter read" (lectura de diseminación) que coloque los datos que entran en ubicaciones no contiguas. *Readv* tiene la forma:

readv(descriptor, iovector, vectorlen)

El argumento *iovector* da la dirección de una estructura de tipo *iovec* (ver figura 20.3) que contiene una secuencia de apuntadores de bloques de memoria dentro de la que deberán almacenarse los datos que entran. El argumento *vectorlen* especifica el número de entradas en *iovector*.

Además de las operaciones de entrada convencionales, hay tres llamadas de sistema adicionales para la entrada de un mensaje a la red. Las llamadas de procesos *recv* son para recibir datos de un socket conectado. Tienen la forma:

recv(socket, buffer, length, flags)

El argumento *socket* especifica un descriptor de socket del que serán recibidos los datos. El argumento *buffer* especifica la dirección en memoria dentro de la que deberá colocarse el mensaje, y el argumento *length* especifica la longitud del área del búfer. Por último, el argumento *flags* permite que quien llame controle la recepción. Entre los valores posibles para el argumento *flags*, hay uno que permite que quien llame se adelante y extraiga una copia del siguiente mensaje que viene sin retirar el mensaje del socket.

La llamada de sistema *recvfrom* permite que quien llame especifique la entrada de un socket no conectado. Incluye argumentos adicionales que permiten a quien llame especificar dónde registrar la dirección de quien envía. La forma es:

recvfrom(socket, buffer, length, flags, fromaddr, addrlen)

Los dos argumentos adicionales, *fromaddr* y *addrlen*, son apuntadores de estructura de dirección de socket y un entero. El sistema operativo utiliza *fromaddr* para registrar las direcciones de quien envía el mensaje y utiliza *fromlen* para registrar la longitud de la dirección de quien lo envía. Obsérvese que la operación de salida *sendto*, que se analizó arriba, toma una dirección en forma exactamente igual a la que *recvfrom* genera. Así, enviar respuestas es fácil.

La última llamada de sistema utilizada para entrada, *recvmsg*, es análoga a la operación de salida *sendmsg*. *Recvmsg* opera como *recvfrom*, pero requiere de menos argumentos. Su forma es:

recvmsg(socket, messagestruct, flags)

donde el argumento *messagestruct* da la dirección de una estructura que sostiene la dirección para el mensaje que entra así como también las ubicaciones para la dirección de quien envía. La estruc-

tura producida por *recvmsg* es exactamente la misma que la estructura utilizada por *sendmsg*, lo que las hace que operen bien en conjunto.

## 20.11 Obtención de direcciones socket locales y remotas

Dijimos que los procesos que se crearon recientemente heredaron el conjunto de sockets abiertos del proceso que los creó. En algunas ocasiones, un proceso recién creado necesita determinar la dirección de destino a la que se conecta el socket. Un proceso puede también determinar la dirección local para un socket. Dos llamadas de sistema proporcionan información como: *getpeername* (*obtener nombre de pareja*) y *getsockname* (*obtener nombre de socket*) (sin importar sus nombres, ambos tratan con lo que pensamos como "direcciones").

Un proceso llama a *getpeername* para determinar la dirección de la pareja (es decir, el extremo remoto) al que se conecta el socket. Tiene la forma:

getpeername(socket, destaddr, addrlen)

El argumento *socket* especifica el socket para el que se desea la dirección. El argumento *destaddr* es un apuntador de estructura de tipo *sockaddr* (ver figura 20.1) que recibirá la dirección de socket. Por último, el argumento *addrlen* es un apuntador de entero que recibirá la longitud de la dirección. *Getpeername* sólo trabaja con sockets conectados.

La llamada de sistema *getsockname* devuelve la dirección local asociada con un socket. Tiene la forma:

getsockname(socket, localaddr, addrlen)

Como se esperaba, el argumento *socket* especifica el socket para el que se desea la dirección local. El argumento *localaddr* es un apuntador a una estructura de tipo *sockaddr* que contendrá la dirección y el argumento *addrlen* es un apuntador a entero que contendrá la longitud de dirección.

## 20.12 Obtención y definición de opciones de socket

Además de enlazar un socket a una dirección local o conectarla a una dirección de destino, surge la necesidad de un mecanismo que permita a los programas de aplicación controlar el socket. Por ejemplo, cuando se usan protocolos que emplean tiempos límite y retransmisión, el programa de aplicación podría obtener o designar los parámetros del tiempo límite. También, podría controlar la ubicación de espacio de búfer, determinar si el socket permite la transmisión de difusión o controlar el procesamiento de datos fuera de banda. En lugar de añadir nuevas llamadas de sistema para cada nueva operación de control, los diseñadores decidieron construir un mecanismo sencillo. El mecanismo tiene dos operaciones: *getsockopt* (*obtener opción socket*) y *setsockopt* (*definir opción socket*).

La llamada de sistema *getsockopt* permite que la aplicación solicite información sobre el socket. Quien llama especifica el socket, que es la opción de interés, y una ubicación en la que se almacena la información requerida. El sistema operativo examina su estructura de datos interna para el socket y transmite la información requerida a quien llama. La llamada tiene la forma:

getsockopt(socket, level, optionid, optionval, length)

El argumento *socket* especifica el socket para el que se necesita la información. El argumento *level* identifica si la operación se aplica al socket mismo o a los protocolos subyacentes que se están usando. El argumento *optionid* especifica una sola opción a la que la petición se aplica. El par de argumentos *optionval* y *length* especifican dos apuntadores. El primero nos da la dirección de un búfer dentro del cual el sistema coloca el valor requerido, y el segundo nos da la dirección de un entero dentro del que el sistema coloca la longitud del valor de opción.

La llamada de sistema *setsockopt* permite que un programa de aplicación configure una opción de socket mediante el conjunto de valores obtenidos con *getsockopt*. Quien llama especifica un socket para el que la opción deberá designarse, la opción a ser cambiada y un valor para tal opción. La llamada a *setsockopt* tiene la forma:

setsockopt(socket, level, optionid, optionval, length)

donde los argumentos como *getsockopt*, a excepción del argumento *length* contienen la longitud de la opción que se está transmitiendo al sistema. Quien llama debe proporcionar un valor legal para la opción así como también la longitud correcta para ese valor. Por supuesto, no todas las opciones se aplican a todos los sockets. La corrección y el significado de las peticiones individuales depende del estado actual del socket y de los protocolos subyacentes que se están usando.

## 20.13 Especificación de una longitud de cola para un servidor

Una de las opciones que se aplica a los sockets se utiliza con tanta frecuencia que se ha tenido que dedicar una llamada de sistema por separado a ella. Para comprender cómo surge, consideremos un servidor. El servidor crea un socket, lo enlaza a un puerto de protocolo bien conocido y espera las peticiones. Si el servidor emplea una entrega de flujo confiable, o si la computación de una respuesta se lleva cantidades no triviales de tiempo, puede suceder que una nueva petición llegue antes de que el servidor termine de responder a una petición anterior. Para evitar el rechazo de protocolos o la eliminación de las peticiones entrantes, el servidor debe indicar al software de protocolo subyacente que desea tener dichas peticiones en cola de espera hasta que haya tiempo para procesarlas.

La llamada de sistema *listen* (*escuchar*) permite que los servidores preparen un socket para las conexiones que vienen. En términos de protocolos subyacentes, *listen* pone al socket en modo pasivo listo para aceptar las conexiones. Cuando el servidor invoca a *listen*, también informa al sistema operativo que el software de protocolo deberá colocar en cola de espera las diversas peticiones simultáneas que llegaron al socket. La forma es:

listen(socket, qlength)

El argumento *socket* indica al descriptor de un socket que debe estar preparado para que lo use el servidor y el argumento *qlength* especifica la longitud de la cola de espera para ese socket. Después de la llamada, el sistema establece la cola de espera para que *qlength* solicite las conexiones. Si la cola de espera está llena cuando llega una petición, el sistema operativo rechaza la conexión descartando la petición. *Listen* se aplica sólo a los sockets que han seleccionado el servicio de entrega de flujo confiable.

## 20.14 Cómo acepta conexiones un servidor

Como hemos visto, un proceso de servidor utiliza las llamadas de sistema *socket, bind* y *listen* para crear un socket, enlazarlo a un puerto de protocolo bien conocido y especificar la longitud de cola para las peticiones de conexión. Obsérvese que la llamada a *bind* asocia al socket con un puerto de protocolo bien conocido, pero el socket no está conectado a un destino exterior específico. De hecho, el destino exterior debe especificar un *wildcard* (*comodín*), lo cual permite que el socket reciba peticiones de conexión de cualquier cliente.

Una vez que se ha establecido un socket, el servidor necesita esperar la conexión. Para hacerlo, se vale de la llamada de sistema *accept* (*aceptar*). Una llamada *accept* se bloquea hasta que la petición de conexión llega. Tiene la forma:

newsock = accept(socket, addr, addrlen)

El argumento *socket* especifica el descriptor del socket en el que va a esperar. El argumento *addr* es un apuntador a estructura de tipo *sockaddr* y *addrlen* es un apuntador a entero. Cuando llega una petición, el sistema llena un argumento *addr* con la dirección del cliente que ha colocado la petición y configura *addrlen* a la longitud de la dirección. Por último, el sistema crea un nuevo socket que tiene su destino conectado hacia el cliente que pide, y devuelve el nuevo descriptor de socket a quien llama. El socket original todavía tiene un comodín de destino externo y aún permanece abierto. De este modo, el servidor maestro puede continuar aceptando las peticiones adicionales en el socket original.

Cuando llega una petición de conexión la llamada a *accept* reaparece. El servidor puede manejar las peticiones de manera concurrente o iterativa. Desde un enfoque iterativo, el servidor mismo maneja la petición, cierra el nuevo socket y, después, llama a *accept* para obtener la siguiente petición de conexión. Desde el enfoque concurrente, después de que la llamada a *accept* reapareció, el servidor maestro crea un esclavo para manejar la petición (en terminología UNIX el proceso se bifurca en el proceso hijo para manejar la petición). El proceso esclavo hereda una copia del nuevo socket, de modo que puede proceder a servir a la petición. Cuando termina, el esclavo cierra el socket y termina. El proceso del servidor original (maestro) cierra su copia de un nuevo socket después de iniciar al esclavo. Después, llama a *accept* para obtener la siguiente petición de conexión.

El diseño concurrente para servidores podría parecer confuso debido a que los diversos procesos usarán el mismo número local de puerto de protocolo. La clave para comprender el mecanis-

mo descansa en la manera en que los protocolos subyacentes tratan a los puertos de protocolos. Recordemos que en el TCP, un par de extremos definen una conexión. De este modo, no importa cuántos procesos se usen en un número local de puerto de protocolo, mientras se conecten a diferentes destinos. En el caso de un servidor concurrente, hay un proceso por cliente y un proceso adicional por destino externo, lo cual permite que se conecte con cualquier localidad externa. Cada proceso restante tiene un destino externo específico. Cuando llega un segmento TCP, es enviado al socket conectado a la fuente de segmento. Si no existe tal socket, el segmento se mandará al socket que tenga un comodín para su destino externo. Además, como el socket con un comodín de destino externo como número de puerto no tiene una conexión abierta, sólo respetará a los segmentos TCP que pidan una nueva conexión.

## 20.15 Servidores que manejan varios servicios

La interfaz BSD de UNIX proporciona otra posibilidad interesante para el diseño de servidores porque permite que un solo proceso espere las conexiones de varios sockets. La llamada de sistema que hace que el diseño sea posible se denomina *select* (*seleccionar*) y se aplica en general a la E/S, no sólo para comunicación en los sockets. *Select* tiene la forma:

nready = select(ndesc, indesc, outdesc, excdesc, timeout)

En general, una llamada a *select* se bloquea y espera que uno de los conjuntos de descriptores de archivos se encuentre listo. El argumento *ndesc* especifica cuántos descriptores se deben examinar (los descriptores revisados son siempre *0* a *ndesc*-1). El argumento *indesc* es un apuntador a una máscara de bits que especifica los descriptores de archivo que se han de revisar para la entrada, el argumento *outdesc* es un apuntador a una máscara de bits que especifica los descriptores de archivo que se han de revisar para la salida, y el argumento *excdesc* es un apuntador a una máscara de bits que especifica los descriptores de archivo que se han de revisar en cuanto a condiciones de excepción. Por último, si el argumento *timeout* (*tiempo límite*) no es cero, será la dirección de un entero la que especifique cuánto esperará una conexión antes de regresarla a quien llama. Un valor de cero para el tiempo límite bloquea a quien llama hasta que el descriptor está listo. Debido a que el argumento *timeout* contiene la dirección del entero de tiempo límite y no al entero mismo, se puede pedir un proceso de demora cero, transmitiendo la dirección de un entero que contiene cero (es decir, un proceso puede ver si la E/S está lista).

Una llamada a *select* devuelve el número de descriptores del conjunto especificado que está listo para la E/S. También cambia las máscaras de bit especificadas por *indesc, outdesc* y *excdesc* para informar a la aplicación qué descriptores de archivo seleccionados están listos. De este modo, antes de llamar a *select*, quien llama debe activar los bits que correspondan a los descriptores que se han de revisar. Siguiendo la llamada, todos los bits que permanezcan con valor *1* corresponderán a un descriptor de archivo ya listo.

Para comunicarse con más de un socket a la vez mediante un proceso, primero se crean todos los sockets que se necesitan y después se usa *select* para determinar cuáles de ellos están listos primero para E/S. Una vez que encuentra un socket listo, el proceso se vale de los procedimientos de entrada o salida definidos arriba para la comunicación.

## 20.16 Obtención y especificación de nombres de anfitrión

El sistema operativo BSD de UNIX mantiene un nombre de anfitrión interno. Para las máquinas de Internet, el nombre interno suele elegirse como el nombre de dominio para la interfaz principal de la red de la máquina. La llamada de sistema *gethostname* (*conseguir nombre de anfitrión*) permite a los procesos del usuario que accedan al nombre de anfitrión, y la llamada de sistema *sethostname* (*definir nombre de anfitrión*) permite a los procesos de privilegio definir el nombre de anfitrión. *Gethostname* tiene la forma:

gethostname(name, length)

El argumento *name* (*nombre*) da la dirección de un arreglo de octetos en la que se ha de almacenar el nombre, y el argumento *length* (*longitud*) es un entero que especifica la longitud del arreglo *name*. Para definir el nombre de anfitrión un proceso privilegiado hace una llamada de la forma:

sethostname(name, length)

El argumento *name* da la dirección de un arreglo en la que se almacena el nombre, y el argumento *length* es un entero que da la longitud del arreglo del nombre.

## 20.17 Obtención y especificación del dominio de anfitrión interno

El sistema operativo mantiene una cadena que especifica el nombre de dominio al que pertenece la máquina. Cuando una localidad obtiene autoridad por parte de un espacio de nombre de dominio, implanta una cadena que identifica su porción de espacio y usa una cadena como el nombre del dominio. Por ejemplo, las máquinas del dominio

cs.purdue.edu

tienen nombres tomados de la leyenda del rey Arturo. De este modo, uno puede encontrar máquinas llamadas *merlín, arturo, guenevere* y *lancelot*. El dominio en sí se llama *camelot*, de manera que el sistema operativo en cada anfitrión del grupo debe estar informado de que reside en el dominio *camelot*. Para hacerlo, un proceso privilegiado utiliza la llamada de sistema *setdomainname*, que tiene la forma:

setdomainname(name, length)

El argumento *name* da la dirección de un arreglo de octetos que contiene el nombre de un dominio y el argumento *length* es un entero que da la longitud del nombre.

Los procesos del usuario llaman a *getdomainname* para recuperar el nombre del dominio del sistema. Getdomainname tiene la forma:

getdomainname(name, length)

donde el argumento *name* especifica la dirección de un arreglo en el que el nombre debe estar almacenado, y el argumento *length* es un entero que especifica la longitud del arreglo.

## 20.18 Llamadas de la biblioteca de red BSD de UNIX

Además de las llamadas de sistema descritas arriba, BSD de UNIX ofrece un conjunto de rutinas de biblioteca que ejecutan funciones útiles, relacionadas con el trabajo en red. En la figura 20.5 se ilustra la diferencia entre las llamadas de sistema y las rutinas de biblioteca. Las llamadas de sistema transmiten el control al sistema operativo, mientras que las rutinas de biblioteca son como otros procedimientos que el programador enlaza dentro de un programa.

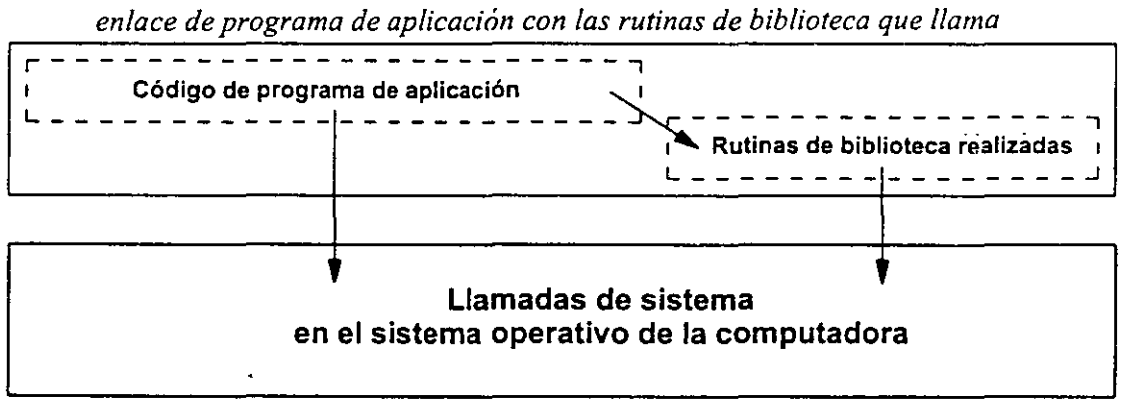



**Figura 20.5** Diferencia entre la biblioteca de rutinas, que están enlazadas dentro de un programa de aplicación y las llamadas de sistema, que son parte de un sistema operativo. Un programa puede llamar a ambos; una biblioteca de rutina puede llamar a otras bibliotecas de rutinas o a llamadas de sistema.

Muchas de las rutinas de biblioteca de BSD de UNIX proporcionan servicios de base de datos que permiten que un proceso determine los nombres de las máquinas y los servicios de red, los números de puertos de protocolos y demás información relacionada. Por ejemplo, un conjunto de rutinas de biblioteca proporciona acceso para la base de datos de los servicios de red. Pensamos en las entradas de la base de datos de los servicios como triadas (formadas por tres partes), donde cada triada contiene el nombre (legible para las personas) de un servicio de red de trabajo, el protocolo que soporta el servicio y un número de puerto de protocolo para el servicio. Existen rutinas de biblioteca que permiten que un proceso obtenga información de una entrada dada.

En la siguientes secciones se examinan grupos de rutinas de biblioteca, se explica sus propósitos y se proporciona información acerca de cómo pueden usarse. Como veremos, los conjuntos le rutinas de biblioteca que proporcionan acceso a bases de datos secuenciales siguen un patrón. Cada conjunto permite la aplicación para: establecer una conexión hacia la base de datos, obtener entradas una a la vez y cerrar la conexión. Las rutinas empleadas para estas tres operaciones se de-

nominan *setXent, getXent* y *endXent*, donde *X* es el nombre de la base de datos. Por ejemplo, las rutinas de biblioteca para la base de datos anfitrión se llaman *sethostent, gethostent* y *endhostent*. En las secciones en las que se describe estas rutinas, se resume las llamadas sin repetir los detalles de su uso.

## 20.19 Rutinas de conversión del orden de red de los octetos

Recordemos que las máquinas se diferencian por la forma en que almacenan las cantidades de enteros y que los protocolos TPC/IP definen un estándar independiente de máquina para orden de octetos. BSD de UNIX proporciona cuatro funciones de biblioteca que convierten el orden de octetos de la máquina local y el orden estándar de octetos de red. Para hacer que los programas sean portátiles deben escribirse para que llamen a rutinas de conversión cada vez que se copia un valor entero de la máquina local a un paquete de red de trabajo o cuando copian un valor de un paquete de red de trabajo a una máquina local.

Las cuatro rutinas de conversión son funciones que toman un valor como un argumento y devuelven un nuevo valor con los octetos reordenados. Por ejemplo, para convertir un entero pequeño (2 octetos) de un orden de octetos de red a un orden de octetos de anfitrión local, un programador llama a *ntohs* (*network to host short*). El formato es:

localshort = ntohs(netshort)

El argumento *netshort* es un entero de 2 octetos (16 bits) en el orden de octetos de la red de trabajo estándar y el resultado, *localshort*, es un orden de octetos de anfitrión local.

UNIX llama *longs* a los enteros de 4 octetos (32 bits). La función *ntohl* (network to host long) convierte los enteros largos de 4 octetos de un orden de octetos de red estándar en un orden de octetos del anfitrión local. Los programas invocan a *ntohl* como función dando un entero largo de un orden de octetos de red como argumento:

locallong = ntohl(netlong)

Dos funciones análogas son las que le permiten al programador convertir el orden de octetos del anfitrión local al orden de octetos de la red. La función *htons* convierte un entero (corto) de 2 octetos del orden de octetos del anfitrión local en un entero de 2 octetos de un orden de octetos de red estándar. Los programas invocan a *htons* como una función:

netshort = htons(localshort)

La rutina final de conversión, *htonl*, convierte a los enteros grandes de orden de octetos del anfitrión en orden de octetos de red. Como los otros, *htonl* es una función:

netlong = htonl(locallong)

Es obvio que las rutinas de conversión preservan las siguientes relaciones matemáticas:

netshort = htons( ntohs(netshort) )

y

localshort = ntohs( htons(localshort) )

Se sostienen relaciones similares para las rutinas de conversión de enteros grandes.

## 20.20 Rutinas de manipulación de direcciones IP

Debido a que muchos programas traducen de direcciones IP de 32 bits y la notación decimal con puntos correspondiente, la biblioteca BSD de UNIX incluye rutinas de herramientas que realizan la traducción. Los procedimientos *inet_addr* y *inet_network* traducen de formato decimal con puntos a direcciones IP de 32 bits en orden de octetos de la red. *Inet_addr* forma una dirección IP de anfitrión de 32 bits; *inet_network* forma la dirección de red con ceros para la parte anfitriona. Tienen la forma:

address = inet_addr(string)

y

address = inet_network(string)

donde el argumento *string* da la dirección de una cadena ASCII que contiene el número expresado en formato decimal con puntos. La forma decimal con puntos puede tener de 1 a 4 segmentos de dígitos separados por puntos. Si aparecen los cuatro, cada uno corresponde a un solo octeto del entero de 32 bits resultante. Si aparecen menos de 4, el último segmento se expande para llenar los octetos restantes.

El procedimiento *inet_ntoa* ejecuta lo inverso a *inet_addr,* pues transforman un entero de 32 bits en una cadena ASCII en formato decimal con puntos. Esto tiene la forma:

str = inet_ntoa(internetaddr)

donde el argumento *internetaddr* es una dirección IP de 32 bits en el orden de octetos de la red y *str* es la dirección de la versión ASCII resultante.

A menudo, los programas que manipulan direcciones IP deben combinar una dirección de red de trabajo con la dirección local de un anfitrión en una red de trabajo. El procedimiento *inet_makeaddr* realiza dicha combinación. Tiene la forma:

internetaddr = inet_makeaddr(net,local)

El argumento *net* es una dirección IP de red de 32 bits en el orden de octetos del anfitrión y el argumento *local* es el entero que representa una dirección de anfitrión local en la red, también en el orden de octetos para el anfitrión local.

Los procedimientos *inet-netof* y *inet-lnaof* proporcionan lo inverso a *inet_makeaddr* pues separan la red y las porciones locales de una dirección IP. Tienen la forma:

net = inet_netof(internetaddr)

y

local = inet_lnaof(internetaddr)

donde el argumento *internetaddr* es una dirección IP de 32 bits en el orden de octetos de la red y los resultados se devuelven en el orden de octetos del anfitrión.

## 20.21 Acceso al sistema de nomenclatura de dominios ANS[5]

Hay un conjunto de cinco procedimientos de biblioteca que comprende la interfaz BSD de UNIX para el sistema de nombre de dominio TCP/IP. Los programas de aplicación que llaman a estas rutinas se convierten en clientes de un sistema de nombre de dominio, mandando una o más peticiones de servicio y recibiendo respuestas.

La idea general es que un programa hace una solicitud, la manda a un servidor y espera una respuesta. Como existen muchas opciones, las rutinas tienen sólo unos cuantos parámetros básicos y utilizan una estructura global: *res*, para sostener a otras. Por ejemplo, un campo en la estructura *res* permite la depuración de mensajes mientras que otro controla si el código usa UDP o TCP para las solicitudes. La mayor parte de los campos en *res* comienza con datos por omisión razonables, de manera que las rutinas se pueden utilizar sin cambiar la estructura *res*.

Un programa llama a *res_init* antes de utilizar otros procedimientos. La llamada no toma argumentos:

res_init()

*Res_init* lee un archivo que contiene información como el nombre de la máquina que corre el servidor de nombre de dominio y almacena los resultados en la estructura global *res*.

El procedimiento *res_mkquery* forma una averiguación de nombre de dominio y la coloca en un búfer en la memoria. La forma de la llamada es:

res_mkquery(op, dname, class, type, data, datalen, newrr, buffer, buflen)

Los primeros siete argumentos corresponden directamente a los campos de la solicitud de nombre de dominio. El argumento *op* especifica la operación requerida, *dname* da la dirección de un arreglo de caracteres que contiene el nombre de dominio, *class* es un entero que da la clase de solicitud, *type* es un entero que da el tipo de solicitud, *data* da la dirección de un arreglo de datos que han de ser incluidos en la solicitud y *datalen* es un entero que da la longitud de los datos. Además de los procedimientos de biblioteca, UNIX proporciona programas de aplicación con definiciones de constantes simbólicas para valores importantes. De este modo, los programadores pueden utilizar el sistema de nombre de dominio sin comprender los detalles del protocolo. Los últimos dos argumentos, *buffer* y *buflen*, especifican la dirección de un área dentro de la que deberá colocarse

[5] En el capítulo 22, se considera detalladamente el Sistema de Nombre de Dominio.

la solicitud y la longitud del entero del área de búfer, respectivamente. Por último, en la implantación actual, el argumento *newrr* no se utiliza.

Una vez que el programa ha formado una búsqueda, llama a *res_send* para enviar a un nombre de servidor y obtener una respuesta. La forma es:

res_send(buffer, buflen, answer, anslen)

El argumento *buffer* es un apuntador a memoria que guarda el mensaje que se ha de enviar (presumiblemente, la aplicación llama al procedimiento *res_mkquery* para formar el mensaje). El argumento *buflen* es un entero que especifica la longitud. El argumento *answer* da la dirección en memoria dentro de la que se deberá escribir una respuesta, y el argumento entero *anslen* especifica la longitud de un área de respuesta.

Además de las rutinas que hacen y envían solicitudes, la biblioteca BSD de UNIX contiene dos rutinas que traducen nombres de dominio de ASCII convencional y del formato comprimido utilizado en las solicitudes. El procedimiento *dn_expand* expande un nombre de dominio comprimido convirtiéndolo en una versión completa en ASCII. Tiene la forma:

dn_expand(msg, eom, compressed, full, fullen)

El argumento *msg* da la dirección de un mensaje de nombre de dominio que contiene el nombre que se ha de expandir, con *eom* especificando el límite de fin de mensaje más allá del cual la expansión no puede ir. El argumento *compressed* es un apuntador para el primer octeto del nombre comprimido. El argumento *full* es un apuntador para un arreglo dentro del cual el nombre expandido deberá estar escrito, y el argumento *fullen* es un entero que especifica la longitud del arreglo.

Generar un nombre comprimido es más complejo que expandir un nombre comprimido porque la compresión comprende la eliminación de los sufijos comunes. Cuando se comprimen nombres, el cliente debe mantener un registro de los sufijos que han aparecido previamente. El procedimiento *dn_comp* comprime un nombre de dominio completo comparando para ello los sufijos con una lista de sufijos previamente utilizados y eliminando los sufijos más grandes posibles. Una llamada tiene la forma:

dn_comp(full, compressed, cmprlen, prevptrs, lastptr)

El argumento *full* da la dirección de un nombre de dominio completo. El argumento *compressed* apunta a un arreglo de octetos que mantendrá un nombre comprimido, con el argumento *cmprlen* especificando la longitud del arreglo. El argumento *prevptrs* es la dirección de un arreglo de apuntadores para los sufijos previamente comprimidos, con *lastptr* que apunta al extremo del arreglo. Normalmente, *dn_comp* comprime el nombre y actualiza *prevptrs* si se ha utilizado un nuevo sufijo.

El procedimiento *dn_comp* puede también usarse para traducir un nombre de dominio de ASCII a la forma interna sin compresión (es decir, sin quitar sufijos). Para hacerlo, el proceso invoca a *dn_comp* con el argumento *prevptrs* definido como *NULL* (es decir, cero).

## 20.22 Obtención de información sobre anfitriones

Existen procedimientos de biblioteca que permiten que un proceso recupere información de un anfitrión dado, ya sea que se tenga un nombre de dominio o una dirección IP. Cuando se emplea en una máquina que tiene acceso a un servidor de nombre de dominio, los procedimientos de la biblioteca realizan el proceso para el cliente del sistema de nombre de dominio enviando una petición a un servidor y esperando la respuesta. Cuando se utiliza en sistemas que no tienen acceso al sistema de nombre de dominio (es decir, un anfitrión que no está en Internet), las rutinas obtienen la información deseada de una base de datos que se mantiene en almacenamiento secundario.

La función *gethostbyname* (*obtener anfitrión por nombre*) toma un nombre de dominio y devuelve un apuntador a una estructura de información para ese anfitrión. Una llamada toma la forma:

```
ptr = gethostbyname(namestr)
```

El argumento *namestr* es un apuntador a una cadena de caracteres que contiene un nombre de dominio para el anfitrión. El valor devuelto, *ptr*, apunta a una estructura que contiene la siguiente información: el nombre oficial del anfitrión, una lista de alias que se han registrado para el anfitrión, el tipo de dirección del anfitrión (es decir, si la dirección es IP), la longitud de la dirección y una lista de una o más direcciones del anfitrión. Se pueden encontrar más detalles en el Manual del Programador de UNIX.

La función *gethostbyaddr* produce la misma información que *gethostbyname*. La diferencia entre las dos es que *gethostbyaddr* acepta la dirección de un anfitrión como un argumento:

```
ptr = gethostbhyaddr(addr, len, type)
```

El argumento *addr* es un apuntador a una secuencia de octetos que contiene una dirección de anfitrión. El argumento *len* es un entero que da la longitud de la dirección y el argumento *type* es un entero que especifica el tipo de la dirección (es decir, que es una dirección IP).

Como se mencionó al principio, los procedimientos *sethostent, gethostent* y *endhostent* proporcionan un acceso secuencial a la base de datos anfitrión.

## 20.23 Obtención de información sobre redes

Los anfitriones que utilizan BSD de UNIX o bien emplean el sistema de nombre de dominio o mantienen una base de datos sencilla de redes en su red de redes. Las rutinas de la biblioteca de la red incluyen cinco rutinas que permiten que un proceso acceda a la base de datos de la red. El procedimiento *getnetbyname* obtiene y da formato al contenido de una entrada de la base de datos una vez dado el nombre de dominio de una red de trabajo. Una llamada tiene la forma:

```
ptr = getnetbyname(name)
```

donde el argumento *name* es un apuntador a una cadena que contiene el nombre de la red de trabajo para la que se desea la información. El valor devuelto es un apuntador a una estructura que con-

tiene los campos para el nombre oficial de la red de trabajo, una lista de los alias registrados, una dirección de tipo entero y una dirección de red de 32 bits (es decir, una dirección IP con la porción de anfitrión puesta en cero).

Un proceso llama a la rutina de biblioteca *getnetbyaddr* cuando necesita buscar información acerca de una red de trabajo una vez dada su dirección. La llamada tiene la forma:

ptr = getnetbyaddr(netaddr, addrtype)

El argumento *netaddr* es una dirección de red de trabajo de 32 bits, y el argumento *addrtype* es un entero que especifica el tipo de *netaddr*. Los procedimientos *setnetent, getnetent* y *endnetent* proporcionan un acceso secuencial a una base de datos de la red.

## 20.24 Obtención de información sobre protocolos

Hay cinco rutinas de biblioteca que proporcionan el acceso a la base de datos de protocolos disponibles en una máquina. Cada protocolo tiene un nombre oficial, alias registrados y un número de protocolo oficial. El procedimiento *getprotobyname* permite que quien llama obtenga información acerca de un protocolo dando su nombre:

ptr = getprotobyname (name)

El argumento *name* es un apuntador a una cadena ASCII que contiene el nombre del protocolo para el que se desea la información. La función devuelve un apuntador a una estructura que tiene campos para el nombre oficial de protocolo, una lista de alias y un valor entero único asignado al protocolo.

El procedimiento *getprotobynumber* permite que un proceso busque la información del protocolo utilizando el número de protocolo como una clave:

ptr = getprotobynumber(number)

Finalmente, los procedimientos *getprotoent, setprotoente* y *endprotoent* proporcionan un acceso secuencial a la base de datos de protocolos.

## 20.25 Obtención de información sobre servicios de red

Recordemos que, en los capítulos 12 y 13, se mencionó que algunos números de puerto de protocolo de UDP y TCP están reservados para los servicios bien conocidos. Por ejemplo, el puerto *43* del TCP está reservado para el servicio *whois*. *Whois* permite a un cliente en una máquina ponerse en contacto con un servidor en otra y obtener información acerca de un usuario que tiene una cuenta en la máquina del servidor. La entrada para *whois* en la base de datos de servicios especifica el nombre de servicio, *whois*, el protocolo, *TCP*, y el número de puerto de protocolo *43*. Existen cin-

co rutinas de biblioteca que obtienen información acerca de los servicios y los puertos de protocolo que usan.

El procedimiento *getservbyname* transforma un servicio nombrado en un número de puerto.

ptr = getservbyname(name, proto)

El argumento *name* especifica la dirección de una cadena que contiene el nombre del servicio deseado, un argumento entero, *proto,* especifica el protocolo con el que el servicio se ha de utilizar. Por lo general, los protocolos están limitados a TCP y UDP. El valor devuelto es una apuntador hacia una estructura que contiene campos para el nombre del servicio, una lista de alias, una identificación del protocolo con el que se usa el servicio y un número entero de puerto de protocolo asignado para ese servicio.

El procedimiento *getservbyport* permite a quien llama obtener una entrada de la base de datos de servicios con sólo dar el número de puerto asignado para ello. Una llamada tiene la forma:

ptr = getservbyport(port, proto)

El argumento *port* es el número entero de puerto de protocolo asignado al servicio y el argumento *proto* especifica el protocolo para el que se desea el servicio. Al igual que con las otras bases de datos, un proceso puede accesar a la base de datos de servicios de manera secuencial utilizando *setservent, getservent* y *endservent.*

## 20.26 Ejemplo de un cliente

El siguiente ejemplo del programa C ilustra la interfaz entre el sistema operativo BSD de UNIX y TCP/IP. Se trata de una implantación sencilla de un *whois* cliente y servidor. Como se define en RFC 954, el servicio *whois* permite que un cliente en una máquina obtenga información acerca de un usuario en un sistema remoto. En esta implantación, el cliente es un programa de aplicación que un usuario invoca mediante dos argumentos: el nombre de una máquina remota y el nombre del usuario en esta máquina acerca de quien se desea la información. El cliente llama a *gethostbyname* para transformar el nombre de la máquina remota en una dirección IP y llama a *getservbyname* para encontrar el puerto bien conocido para el servicio *whois.* Una vez que ha transformado los nombres de anfitrión y servicio, el cliente crea un socket, especificando que el socket usará una cadena de entrega confiable (es decir, TCP). El cliente entonces enlaza el socket con el puerto de protocolo *whois* en la máquina de destino especificada.

```
/* whoisclient.c - main */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h
#include <netinet/in.h>
#include <netdb.h>
/*------------------------------------------------------------------
```

```
 * Programa:      whoisclient
 *
 * Propósito:     Programa de aplicación de UNIX que se convierte en
 *                cliente para el servicio "whois" de Internet.
 *
 * Usa:           whois hostname username
 *
 * Autor:         Barry Shein, Boston University
 *
 * Fecha:         Enero de 1987
 *
 *-------------------------------------------------------------------
 */
main(argc, argv)
int argc;          /* declaraciones de argumento estándar de UNIX*/
char *argv[ ];
{
  int s;                      /* descriptor de socket*/
  int len;                    /* longitud de datos recibidos*/
  struct sockaddr_in sa;      /* estruc. direc. socket Internet*/
  struct hosten *hp;          /* resultado búsqueda nombre anfi.*/
  struct servent *sp;         /* resultado servicio búsqueda*/
  char buf[ BUFSIZ+1] ;       /* búfer que lee inf. whois*/
  char *myname;               /* apuntador a nombre este prog.*/
  char *host;                 /* apuntador a nombre afi. remot.*/
  char *user;                 /* apuntador a nombre usuario rem.*/

  myname = argv[ 0] ;
  /*
   * Revisar que haya dos argumentos de línea de comandos
   */
  if(argc !=3) {
     fprintf(stderr, "usage: %s host username\n", myname);
     exit(1);
  }
  host = argv[ 1] ;
  user = argv[ 2] ;
  /*
   * Búsqueda del nombre de anfitrión especificado
   */
  if((hp = gethostbyname(host)) == NULL) {
     fprint(stderr,"%s: %s: no such host?\n", myname, host);
     exit(1);
  }
  /*
   * Poner la dirección del anfitrión en el tipo de anfitrión dentro de
     la estructura socket
   */
  bcopy((char *)hp-h_addr, (char *) &sa.sin_addr, hp-h_length);
```

```
/*
 * Búsqueda del número de socket para el servicio WHOIS
 */
if((sp = getservbhyname("whois","tcp")) == NULL) {
   fprintf(stderr, "%s: No existe servicio whois en este anfitrión\n",
   myname);
   exit(1);
}
/*
 * Poner el número de socket whois en la estructura de socket.
 */
sa.sin_port = sp-s_port;
/*
 * Ubicar un socket abierto
 */
if((s = socket(hp-h_addrtype, SOCK_STREAM, 0))  0) {
   perror("socket");
   exit(1);
}
/*
 * Conectar al servidor remoto
 */
if(connect(s, &sa, sizeof sa)  0) {
   perror("connect");
   exit(1);
}
/*
 * Enviar la petición
 */
if(write(s, user, strlen(user)) 1= strlen(user)) {
   fprintf(stderr, "%s: error de escritura\n", myname);
               exit(1);
}
/*
 * Leer la respuesta y ponerla a la salida del usuario
 */
while( (len = read(s, buf, BUFSIZ))  0)
   write(1, buf, len);
close(s);
exit(0);
}
```

## 20.27 Ejemplo de un servidor

El servidor de ejemplo es sólo un poco más complejo que el de cliente. El servidor escucha en el puerto "whois" bien conocido y devuelve la información requerida en respuesta a una petición de

cualquier cliente. La información se toma del archivo de claves de acceso de UNIX de la máquina del servidor.

```
/* whoisserver.c - main */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <pwd.h>

/*------------------------------------------------------------------
 * Programa:      whoisserver
 *
 * Propósito:     Programa de aplicación de UNIX que actúa como un
 *                servidor para el servicio "whois" de la máquina
 *                local. Escucha en el puerto WHOIS bien conocido (43)
 *                y contesta las solicitudes de los clientes. Este
 *                programa requiere un privilegio de super usuario para
 *                correrse.
 *
 * Usa:           whois hostname username
 *
 * Autor:         Barry Shein, Boston University
 *
 * Fecha:                    Enero de 1987
 *
 *------------------------------------------------------------------
*/

#define BACKLOG       5      /* # peticiones dispuestas para poner
                             en cola*/
#define MAXHOSTNAME   32     /* longitud máxima tolerable del nombre
                             de anfitrión*/

main(argc, argv)
int argc;                    /* declaraciones de argumento estándar
                             de UNIX*/
char *argv[];
{
  int s, t;                  /* descriptores de socket*/
  int I;                     /* entero de propósito general*/
  struct sockaddr_in sa, isa;/* estructura de la dirección de socket
                             de Internet*/
  struct hostent *hp;        /* resultado de la búsqueda de nombre
                             de anfitrión*/
  char *myname;              /* apuntador al nombre de este programa*/
  struct servent *sp;        /* resultado del servicio de búsqueda*/
```

```
char localhost[ MAXHOSTNAME+1] ;/* nombre del anfitrión local como
                                   cadena de caracteres*/

myname = argv[ 0] ;
/*
 * Búsqueda de la entrada al servicio WHOIS
 */
if((sp = getservbyname("whois","tcp")) == NULL)  {
   fprintf(stderr, "%s: No existe servicio whois en este anfitrión\n",
   myname);
   exit(1);
}
/*
 * Obtener nuestra propia información de anfitrión
 */
gethostname(localhost, MAXHOSTNAME);
if((hp = gethostbyname(localhost)) == NULL) {
   fprintf(stderr, "%s: no se puede obtener información del anfitrión
   local?\n", myname);
   exit(1);
}
/*
 * Poner el número de socket WHOIS y nuestra información de
 * dirección dentro de la estructura de socket
 */
sa.sin_port = sp-s_port;
bcopy((char *)hp-h_addr, (char *)&sa.sin_addr, hp-_length);
sa.sin_family = hp-h_addrtype;
/*
 * Ubicar un socket abierto para las conexiones que vienen
 */
si((s = socket(hp-h_addrtype, SOCK_STREAM, 0))  0) {
   perror("socket");
  exit(1);
}
/*
 * Enlazar el socket al puerto de servicio
 * de modo que escuchemos las conexiones entrantes
 */
if(bind(s, &sa, sizeof sa)  0) {
   perror("bind");
   exit(1);
}
/*
 * Definir las conexiones máximas que dejaremos atrás
 */
listen(s, BACKLOG);
/*
 * Caer en un ciclo infinito de espera de las nuevas conexiones
 */
```

```
  while(1) {
     l = sizeof isa;
     /*
      * Esperamos la aceptación () mientras se espera a nuevos clientes
      */
     if((t = accept(s, &isa, &I))  0) {
       perror("accept");
       exit(1);
     }
     whois(t);                  /* ejecuta el servicio WHOIS actual */
     cerrar(t);
  }
  }
  /*
   * Obtener la petición WHOIS del anfitrión remoto y dar formato
     a la respuesta.
   */
  whois(sock)
  int sock;
  {
  struct passwd *p;
  char buf[ BUFSIZ+1] ;
     int I;

  /*
   * Obtener una petición de línea
   */
  if( (I = read(sock, buf, BUFSIZ)) < = 0)
     return;
  buf[ i] = '\0' ;              /* Termina estado nulo */
  /*
   *Búsqueda del usuario requerido y formato a respuesta
   */
  if((p = getpwnam(buf)) == NULL)
     strcpy(buf,"Wser not found\n");
  else
     sprintf(buf, "%s: %s\n", p-pw_name, p-pw_gecos);
  /*
   * Se devuelve respuesta
   */
  write(sock, buf, strien(buf));
  return;
}
```

## 20.28 Resumen

Debido a que el software de protocolo TCP/IP reside dentro de un sistema operativo, la interfaz exacta entre un programa de aplicación y los protocolos TCP/IP dependen de los detalles del sistema operativo; no está especificada por el estándar de protocolo de TCP/IP. Examinamos la interfaz socket originalmente diseñada para BSD de UNIX y vimos que adoptó el paradigma open-read-write-close de UNIX. Para utilizar el TCP, un programa debe crear un socket, enlazar direcciones a éste, aceptar las conexiones que llegan y después comunicarse por medio de las primitivas *read* o *write*. Por último, cuando se termina de usar un socket, el programa debe cerrarlo. Además de la abstracción del socket y las llamadas de sistema que operan en los sockets, BSD de UNIX incluye rutinas de biblioteca que ayudan a los programadores a crear y manipular direcciones IP, convertir enteros entre el formato de la máquina local y el orden de octetos del estándar de red, y buscar información como direcciones de redes.

La interfaz socket se ha convertido en algo muy popular y es soportada ampliamente por muchos vendedores. Los vendedores que no ofrecen facilidades de socket en sus sistemas operativos a menudo proporcionan una biblioteca socket que les facilita a los programadores escribir aplicaciones mediante las llamadas de socket aun cuando el sistema operativo subyacente utilice un conjunto diferente de llamadas de sistema.

## PARA CONOCER MÁS

Se puede encontrar información detallada sobre las llamadas del sistema socket en la sección 2 del *UNIX Programmer's Manual* (*Manual del programador de Unix*), la cual contiene una descripción de cada llamada del sistema UNIX; en la sección 3 aparece una descripción de cada procedimiento de biblioteca. UNIX también proporciona copias en línea de las páginas del manual vía el comando *man*. Leffler, McKusick, Karels y Quarterman (1989) exploran con mayor detalle el sistema UNIX.

Los vendedores de sistemas operativos a menudo proporcionan bibliotecas de procedimientos que emulan a los sockets en sus sistemas. Además, Microsoft y otras compañías han cooperado para definir la interfaz *Winsock* que permite a los programas de aplicación que hacen llamadas socket que corran con Microsoft Windows; varios vendedores ofrecen productos compatibles con Winsock. Si requiere mayores detalles, consulte los manuales de programación de los vendedores.

La versión socket del volumen 3 de esta obra describe cómo están estructurados los programas de cliente y servidor y cómo utilizan la interfaz socket. En el volumen 3 podemos encontrar la versión TLI que nos proporciona una introducción a la *Transport Layer Interface (Interfaz de Capa de Transporte)*, que es una alternativa a los sockets utilizados en el Sistema V de UNIX.

# EJERCICIOS

**20.1** Trate de correr el ejemplo de *whois* cliente y servidor en su sistema local.

**20.2** Construya un servidor sencillo que acepte diversas conexiones concurrentes (para probarlo haga que el proceso maneje una conexión de impresión de un pequeño mensaje, demore un tiempo aleatorio, imprima otro mensaje y salga).

**20.3** ¿Cuándo es importante la llamada *listen* (*escuchar*)?

**20.4** ¿Qué procedimientos proporciona su sistema local para acceder al sistema de nombre de dominio?

**20.5** Diseñe un servidor que utilice un proceso UNIX sencillo, pero que maneje diversas conexiones concurrentes TCP. Sugerencia: piensa en *select* (*seleccionar*) (*poll* en SISTEMA V).

**20.6** Infórmese sobre la Interfaz de Biblioteca de Transporte (TLI) del Sistema V de AT&T y compárela con la interfaz socket. ¿Cuáles son las principales diferencias conceptuales?

**20.7** Cada sistema operativo limita el número de sockets que un programa dado puede usar en cualquier momento. ¿Cuántos sockets puede crear un programa en su sistema local?

**20.8** El mecanismo descriptor de socket/archivo y las operaciones asociadas *read* (*leer*) y *write* (*escribir*) se pueden considerar como una forma de objeto orientada al diseño. Explique por qué.

**20.9** Considere un diseño de interfaz alternativo que proporcione una interfaz para todas las capas del software de protocolo (es decir, que el sistema permita a un programa de aplicación enviar y recibir paquetes sin procesar, sin usar el IP, o enviar y recibir datagramas IP sin emplear el UDP o el TCP). ¿Cuáles son las ventajas y las desventajas de tener dicha interfaz?

**20.10** Un cliente y un servidor pueden correr en la misma computadora y valerse de un socket TCP para comunicarse. Explique de qué manera es posible construir un cliente-servidor que pueda comunicarse con una sola máquina sin enterarse de la dirección IP del anfitrión.

**20.11** Experimente con el servidor de muestra de este capítulo para ver si puede generar conexiones TCP que sean lo suficientemente rápidas como para exceder el trabajo atrasado que especifica el servidor. ¿Espera que las peticiones de conexión entrantes excedan el trabajo atrasado más rápido si el servidor opera en una computadora que tiene *1* procesador o en una computadora que tiene *5* procesadores? Explíquelo.

# 21

# Arranque y autoconfiguración (BOOTP, DHCP)

## 21.1 Introducción

Este capítulo muestra cómo se utiliza el paradigma cliente-servidor para el proceso de arranque. Cada computadora conectada a una red de redes TCP/IP necesita saber su dirección IP antes de que pueda enviar o recibir datagramas. Además, una computadora requiere información adicional como la dirección de un ruteador, la mascara de red en uso y la dirección de un servidor de nombres. En el capítulo 6, se describió cómo puede utilizar una computadora el protocolo RARP en el inicio de un sistema para determinar su dirección IP. En este capítulo, se analiza una alternativa: dos protocolos del proceso de arranque muy relacionados que permiten a un anfitrión determinar su dirección IP sin utilizar RARP. Sorprendentemente, el cliente y el servidor se comunican mediante el UDP, el protocolo de datagrama usuario descrito en el capítulo 12.

Lo que es sorprendente del procedimiento de arranque es que el UDP depende del protocolo IP para transferir mensajes, y podría parecer imposible que una computadora pudiera utilizar el UDP para localizar una dirección IP que utiliza ésta cuando se comunica. Examinaremos los protocolos que nos ayudarán a entender cómo puede utilizar una computadora la dirección IP especial mencionada en el capítulo 4 y la flexibilidad del mecanismo de transporte UDP/IP. También veremos de qué manera asigna un servidor una dirección IP a una computadora en forma automática. Esta asignación es especialmente importante en ambientes que permiten conexiones de red de redes temporales en los que las computadoras se transfieren de una red a otra (por ejemplo, un empleado con una computadora portátil puede moverse de una localidad hacia otra en una compañía).

## 21.2 La necesidad de una alternativa a RARP

El capítulo 6 presenta el problema de las computadoras sin disco durante el arranque de sistema. Estas máquinas por lo general contienen un programa de arranque en un medio de almacenamiento no volátil (por ejemplo, en ROM). Para minimizar costos y conservar partes intercambiables, el vendedor coloca exactamente el mismo protocolo en todas las máquinas. Como las computadoras con diferentes direcciones IP corren el mismo programa de arranque, el código no puede contener una dirección IP. Así, una máquina sin disco debe obtener su dirección IP desde otra fuente. De hecho, una computadora sin disco necesita conocer mucho más que su dirección IP. En general, la memoria ROM sólo contiene un pequeño programa de arranque, de manera que las computadoras sin disco deben también obtener una imagen de memoria inicial para su ejecución. Además, cada máquina sin disco puede determinar la dirección de un servidor de archivos en el que pueda almacenar y recuperar datos, así como la dirección del ruteador IP más cercano.

El protocolo RARP, descrito en el capítulo 6, tiene tres inconvenientes. En primer lugar, dado que RARP opera en un nivel bajo, su uso requiere de un acceso directo hacia el hardware de red. Así pues, puede resultar difícil o imposible para un programador de aplicaciones construir un servidor. En segundo lugar, aun cuando RARP necesita un intercambio de paquetes entre una máquina cliente y una computadora que responda a las solicitudes, la réplica contiene sólo una pequeña parte de información: la dirección IP del cliente de 4 octetos. Estos inconvenientes son especialmente molestos en redes como Ethernet que imponen un tamaño de paquete mínimo, ya que la información adicional puede enviarse en la respuesta sin costo adicional. En tercer lugar, como RARP emplea una dirección de hardware de computadora para identificar una máquina, no puede utilizarse en redes con una asignación dinámica de direcciones de hardware.

Para sortear algunas de las dificultades de RARP, los investigadores desarrollaron el *BOOTstrap Protocol* (*BOOTP*). Más recientemente, el *Dynamic Host Configuration Protocol* (*DHCP*) ha sido propuesto como sucesor del BOOTP. Dado que los dos protocolos se encuentran estrechamente relacionados, la mayor parte de las descripciones en este capítulo se aplica a ambos. Para simplificar el texto, describiremos primero BOOTP y luego veremos cómo extiende el DHCP su funcionalidad a fin de proporcionar una asignación de direcciones dinámica.

Dado que utiliza al UDP y al IP, BOOTP debe implantarse con un programa de aplicación. Como RARP, BOOTP opera dentro de un paradigma cliente-servidor y requiere sólo de un intercambio de paquetes. No obstante, BOOTP es más eficiente que RARP pues un sólo mensaje BOOTP especifica muchos aspectos necesarios para arranque, incluyendo una dirección IP de computadora, la dirección de un ruteador y la dirección de un servidor. BOOTP también incluye un campo de vendedor específico en la respuesta, que permite a los vendedores de hardware enviar información adicional utilizada sólo en sus computadoras.[1]

[1] Como veremos, el término "vendedor específico" es un nombre equivocado pues las especificaciones actuales también recomiendan utilizar el área vendedor-específico para información de propósito general, como máscaras de subred;

## 21.3 Utilización de IP para determinar una dirección IP

Dijimos que BOOTP se vale del UDP para transportar mensajes y que los mensajes UDP están encapsulados en los datagramas IP para su entrega. A fin de entender cómo puede enviar una computadora a BOOTP en un datagrama IP antes de que la computadora conozca su dirección IP, recordemos, del capítulo 4, que hay varias direcciones IP de casos especiales. En particular, cuando se usa como una dirección de destino, la dirección IP está formada sólo por *unos* (255.255.255.255), que especifican el límite para la difusión. El software IP puede aceptar y difundir datagramas que especifican la dirección de difusión límite, incluso antes de que el software haya descubierto la información de su dirección IP local. El punto es el siguiente:

> *Un programa de aplicación puede utilizar la dirección IP de difusión límite para obligar al IP a difundir un datagrama en la red local, antes de que el IP haya descubierto la dirección IP de la red local o la dirección IP de la máquina.*

Supongamos que la máquina cliente *A* desea utilizar BOOTP para localizar información de arranque (incluyendo su dirección IP) y que *B* es el servidor en la misma red física que responderá a la solicitud. Dado que *A* no conoce la dirección IP de *B* o la dirección IP de la red, debe difundir en su BOOTP inicial la solicitud para utilizar la dirección IP de difusión límite. ¿Cual será la réplica? ¿Puede *B* enviar una réplica directamente? Por lo general no. Aun cuando pueda no ser obvio, *B* quizás necesite utilizar la dirección de difusión límite para su réplica, aunque conozca la dirección IP de *A*. Para entender por qué, considere qué sucedería si un programa de aplicación en *B* logra enviar un datagrama utilizando la dirección IP de *A*. Después de rutear el datagrama, el software IP en *B* pasará el datagrama al software de interfaz de red. El software de interfaz debe transformar la siguiente dirección IP de salto hacia la dirección de hardware correspondiente, presumiblemente utilizando ARP como se describe en el capítulo 5. Sin embargo, debido a que *A* no ha recibido la réplica BOOTP, no reconocerá su dirección IP y no podrá responder a la solicitud ARP de *B*. Además, *B* tiene sólo dos alternativas: difundir la réplica o utilizar información del paquete de solicitud para añadir de manera manual una entrada a su memoria intermedia ARP. En los sistemas que no permiten a los programas de aplicación modificar la memoria intermedia ARP, la difusión es la única solución.

## 21.4 Política de retransmisión BOOTP

BOOTP confiere toda la responsabilidad de la confiabilidad de la comunicación al cliente. Sabemos que como el UDP utiliza al IP para la entrega, los mensajes pueden retrasarse, perderse, entregarse fuera de orden o duplicarse. Además, dado que el IP no proporciona una suma de verificación para los datos, el datagrama UDP puede llegar con algunos bits alterados. Para protegerse contra la alteración de datos, BOOTP requiere que el UDP utilice sumas de verificación. También, especifica que las solicitudes y las réplicas deben enviarse con el bit de *no fragmentar* activado a fin de adaptarse a los clientes que tengan una memoria pequeña para reensamblar datagramas. BOOTP también está construido para permitir réplicas múltiples; las acepta y procesa primero.

Para manejar datagramas perdidos, BOOTP utiliza la técnica convencional de *tiempo límite* (*time out*) y *retransmisión* (*retransmission*). Cuando el cliente transmite una solicitud, inicia un temporizador. Si no llega ninguna réplica antes de que el tiempo expire, el cliente debe retransmitir la solicitud. Por supuesto, después de una falla en el suministro de alimentación, todas las máquinas en la red deben arrancar de nuevo de manera simultánea, posiblemente sobrecargando el servidor BOOTP con solicitudes. Si todos los clientes emplean exactamente el mismo tiempo límite de retransmisión, muchos de ellos o todos retransmitirán simultáneamente. Para evitar las colisiones resultantes, las especificaciones BOOTP recomiendan utilizar un retardo aleatorio. Además, las especificaciones aconsejan comenzar con un tiempo límite aleatorio de entre *0* y *4* segundos y duplicar el temporizador después de cada retransmisión. Luego de que el temporizador alcanza un valor alto, *60* segundos, el cliente no incrementa el temporizador, pero continúa utilizando el procedimiento de establecer un valor aleatoriamente. Duplicar el tiempo límite después de cada retransmisión evita que BOOTP añada un tráfico excesivo y congestione la red; el procedimiento aleatorio ayuda a evitar las transmisiones simultáneas.

## 21.5 Formato de los mensajes BOOTP

Para mantener a una implantación tan simple como sea posible, BOOTP los mensajes tienen campos de longitud fija y las réplicas poseen el mismo formato que las solicitudes. Aun cuando dijimos que los clientes y los servidores son programas, el protocolo BOOTP emplea los términos con cierta vaguedad al referirse a la máquina que envía una solicitud BOOTP como el *cliente* y a cualquier máquina que envía una réplica como el *servidor*. La figura 21.1 muestra el formato del mensaje BOOTP.

El campo *OP* especifica si el mensaje es una solicitud (*valor 1*) o una réplica (*valor 2*). Como en ARP, los campos *HTYPE* y *HLEN* especifican el tipo de hardware de red y la longitud de la dirección de hardware (por ejemplo, Ethernet tiene definido un tipo *1* y una dirección de una longitud de 6).[2] El cliente coloca *0* en el campo *HOPS*. Si recibe la solicitud y decide transferir la solicitud hacia otra máquina (por ejemplo, permitir el arranque a través de varios ruteadores), el servidor BOOTP incrementará el contador *HOPS*. El campo *TRANSACTION ID (ID DE TRANSACCIÓN)* contiene un entero que la máquina sin disco utiliza para cotejar las respuestas con las solicitudes. El campo *SECONDS* (*SEGUNDOS*) reporta el número de segundos desde que el cliente comenzó el arranque.

El campo *CLIENT IP ADDRESS* (*DIRECCIÓN IP DEL CLIENTE*) y todos los campos que le siguen contienen la mayor parte de la información importante. Para permitir una flexibilidad creciente, los clientes llenan los campos con toda la información con la que cuentan y dejan los campos restantes puestos en cero. Por ejemplo, si un cliente conoce el nombre o la dirección de un servidor específico desde el que espera información, puede llenar los campos *SERVER IP ADDRESS* (*DIRECCIÓN IP DEL SERVIDOR*) o *SERVER HOST NAME* (*NOMBRE DEL ANFITRIÓN SERVIDOR*). Si estos campos no son iguales a cero, sólo el servidor con el nombre-dirección que concuerde responderá a la solicitud. Si los campos están puestos en cero, responderá cualquier servidor que reciba la solicitud.

BOOTP puede utilizarse desde un cliente que ya conozca su dirección IP (por ejemplo, para obtener información del archivo de arranque). Un cliente que conozca su dirección IP la colocará

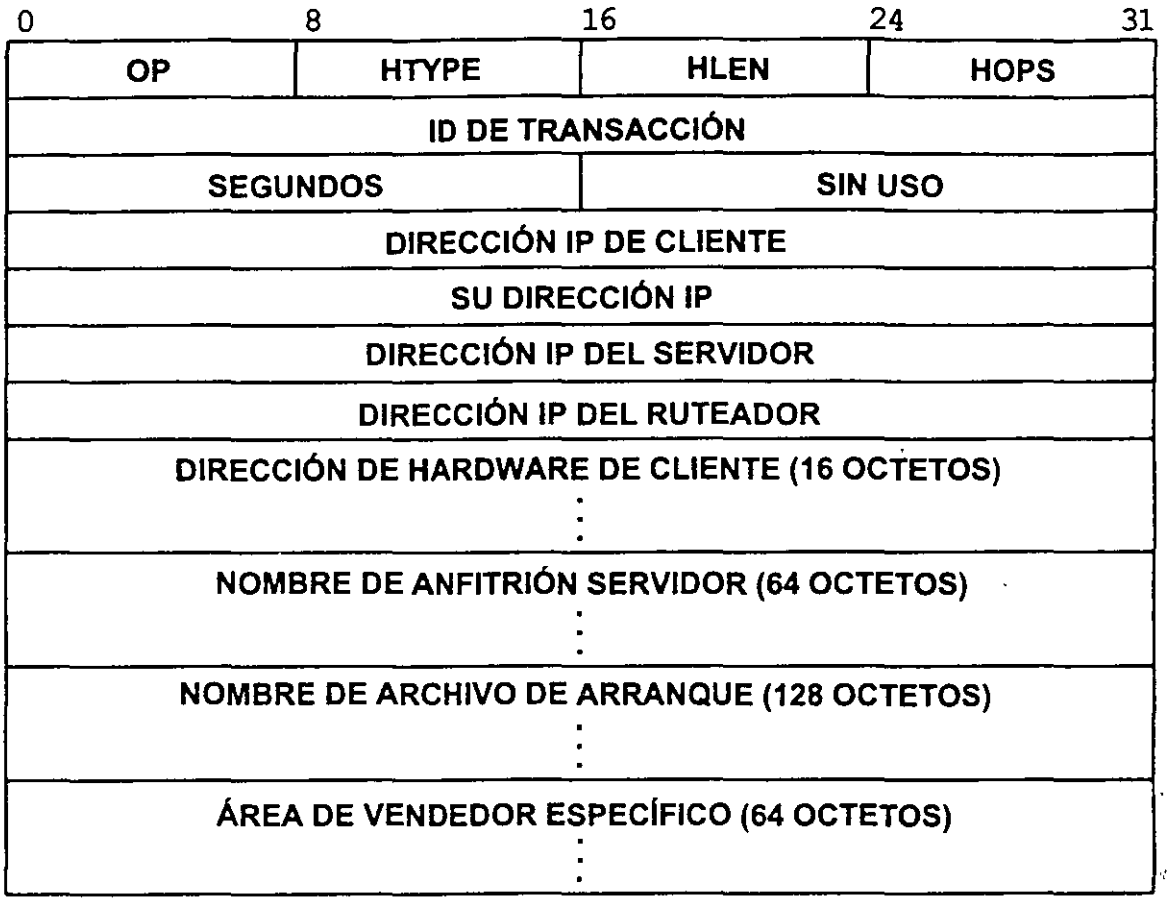



**Figura 21.1** Formato de un mensaje BOOTP. Para mantener las implementaciones lo suficientemente pequeñas como para ajustarse a una memoria ROM, todos los campos tienen longitudes fijas.

en el campo *CLIENT IP ADDRESS* (*DIRECCIÓN IP DEL CLIENTE*); otros clientes utilizarán cero en este campo. Si la dirección IP del cliente es cero en la solicitud, un servidor devolverá la dirección IP del cliente en el campo *YOUR IP ADDRESS* (*SU DIRECCIÓN IP*).

## 21.6 Procedimiento de arranque de dos pasos

BOOTP utiliza un procedimiento de arranque de dos pasos. No proporciona una imagen de memoria a los clientes —sólo proporciona al cliente información necesaria para obtener una imagen. El cliente entonces utiliza un segundo protocolo (por ejemplo, el TFTP considerado en el capítulo 24) para obtener la imagen de memoria. Aunque el procedimiento de dos pasos muchas veces parece innecesario, permite una clara separación de configuración y almacenamiento. Un servidor BOOTP no necesita correr en la misma máquina que almacena las imágenes de memoria. De hecho, el servidor BOOTP opera desde una simple base de datos que sólo conoce los nombres de las imágenes de memoria.

Mantener la configuración separada del almacenamiento es importante pues permite al administrador configurar conjuntos de máquinas para que éstas actúen en forma idéntica o de manera independiente. El campo *BOOT FILE NAME* de un mensaje BOOTP ilustra el concepto. Supongamos que un administrador tiene varias estaciones de trabajo con diferentes arquitecturas de hardware, y supongamos que cuando el usuario arranca una de las estaciones de trabajo, éstas seleccionan correr en UNIX o en un sistema operativo local. Debido a que el conjunto de estaciones de trabajo incluye múltiples arquitecturas de hardware, no operará en todas las máquinas una sola imagen de memoria. Para adaptarse a esta diversidad, BOOTP permite que el campo *BOOT FILE NAME* en una solicitud contenga un nombre genérico como "unix", lo cual significa "quiero arrancar el sistema operativo UNIX para esta máquina". El servidor BOOTP consulta su base de datos de configuración para transformar el nombre genérico en un nombre de archivo específico que contiene la imagen de memoria UNIX apropiada para el hardware del cliente y devuelve el nombre específico (es decir, completamente definido) en su réplica. Por supuesto, la base de datos de configuración también permite un arranque completamente automático mediante el cual el cliente coloca ceros en el campo *BOOT FILE NAME* y, con ello, BOOTP selecciona una imagen de memoria para la máquina. La ventaja del método automático es que permite al usuario especificar nombres genéricos que trabajen en cualquier máquina. No es necesario recordar nombres de archivos específicos o arquitecturas de hardware.

## 21.7 Campo área de vendedor específico

El campo *VENDOR-SPECIFIC AREA* (*ÁREA DE VENDEDOR ESPECÍFICO*) contiene información opcional para su transferencia del servidor al cliente. Aun cuando la sintaxis resulta intrincada, no es difícil. Los primeros 4 octetos del campo se llaman *magic cookie* y definen el formato de los temas restantes; el formato estándar descrito aquí utiliza un magic cookie con valor de 99.130.83.99 (notación decimal con puntos). A continuación de este campo, sigue una lista de términos, en la que cada aspecto contiene un octeto *type* (*tipo*), un octeto *length* (*longitud*) opcional y varios octetos *value* (*valor*). El estándar define los siguientes tipos que tienen longitudes de valores fijos predeterminados:

| Tipo | Código | Valor de Longitud | Valor de Contenidos |
|---|---|---|---|
| Relleno | 0 | - | Cero — utilizado sólo como relleno |
| Máscara de subred | 1 | 4 | Máscara de subred para red local |
| Hora del día | 2 | 4 | Hora del día en tiempo universal |
| Fin | 255 | - | Fin de la lista de aspectos |

**Figura 21.2** Contenido de la información de vendedor. Los campos fijos deben existir para los tipos *1* y *2*, pero no para los tipos *0* y *255*.

Aun cuando una computadora puede obtener información de máscara de su red mediante una solicitud ICMP, el estándar recomienda ahora que los servidores BOOTP proporcionen la máscara de su red en cada réplica para suprimir mensajes ICMP innecesarios.

Algunos aspectos adicionales en *VENDOR-SPECIFIC AREA* tienen un octeto *type*, un octeto *length* y uno *value*, como se muestra en la figura 21.3.

## 21.8 La necesidad de una configuración dinámica

Como RARP, BOOTP fue diseñado para un ambiente relativamente estático en el que cada anfitrión tiene una conexión de red permanente. Un administrador crea un archivo de configuración BOOTP que especifica un conjunto de parámetros BOOTP para cada anfitrión. El archivo no cambia con frecuencia pues la configuración generalmente se mantiene estable. Por lo común, una configuración no registra cambios durante semanas.

Con la llegada de redes inalámbricas y computadoras portátiles como las laptop y las notebook, se ha vuelto posible transportar a las computadoras de una localidad a otra rápida y fácilmente. BOOTP no se adapta a esta situación pues la información de configuración no puede cambiar rápidamente. Así pues, sólo proporciona una transformación estática desde un identificador de anfitrión hacia parámetros para el anfitrión. Además, un administrador debe introducir un conjunto de parámetros para cada anfitrión y luego almacenar la información en un archivo de configuración de servidor BOOTP —BOOTP no incluye una forma para asignar dinámicamente valores a máquinas individuales. En particular, un administrador debe asignar cada anfitrión a una dirección IP y configurar el servidor para entender la transformación del identificador de anfitrión a la dirección IP.

Los parámetros de asignación estática trabajan bien si las computadoras se mantienen en localidades fijas y el administrador tiene suficientes direcciones IP para asignar a cada computadora

| Tipo | Código | Longitud en octetos | Contenidos |
|---|---|---|---|
| Ruteadores | 3 | N | Direcciones IP de N/4 ruteadores |
| Servidor de hora | 4 | N | Direcciones IP de N/4 servidores de hora |
| Servidor IEN116 | 5 | N | Direcciones IP de N/4 servidores IEN116 |
| Servidor de dominio | 6 | N | Direcciones IP de N/4 servidores DNS |
| Servidor Log | 7 | N | Direcciones IP de N/4 servidores log |
| Servidor de citas | 8 | N | Direcciones IP de N/4 servidores de citas |
| Servidor Lpr | 9 | N | Direcciones IP de N/4 servidores lpr |
| Impress | 10 | N | Direcciones IP de N/4 servidores Impress |
| Servidor RLP | 11 | N | Direcciones IP de N/4 servidores RLP |
| Hostname | 12 | N | N bytes de nombre de anfitrión cliente |
| Tamaño de arranque | 13 | 2 | entero de 2 octetos para tamaño del archivo de arranque |
| RESERVADO | 128-254 | - | Reservado para usos específicos de la localidad |

**Figura 21.3** Tipo y contenido de aspecto del *VENDOR-SPECIFIC AREA* de una réplica BOOTP que tiene longitudes variables.

una dirección IP única. Sin embargo, en los casos en los que las computadoras se muevan con frecuencia o que el número de computadoras físicas exceda el de direcciones de anfitrión IP disponibles, la asignación estática generará sobrecargas excesivas.

Para entender cómo puede exceder el número de computadoras el de direcciones IP disponibles, consideremos una LAN, en el laboratorio de un colegio que ha sido asignado a direcciones clase C o a una subred de direcciones clase B con 255 direcciones. Supongamos que como el laboratorio sólo tiene sillas para 30 estudiantes, la cédula del laboratorio en 10 diferentes momentos durante una semana le da cabida a más de 300 estudiantes. Además, supongamos que cada estudiante transporta una computadora notebook personal que se utiliza en el laboratorio. En cualquier momento, la red tiene más de 30 computadoras activas. Sin embargo, ya que las direcciones de red pueden dar cabida a más de 255 anfitriones, un anfitrión no puede asignar una dirección única a cada computadora. Así, aunque recursos como las conexiones físicas limitan el número de conexiones simultáneas, el número de computadoras potencial que puede utilizar la instalación es elevado. Está claro que un sistema es inadecuado si requiere que el administrador cambie el archivo de configuración del servidor antes de que se añada una nueva computadora a la red y comience a comunicarse; se necesita pues un mecanismo automatizado.

## 21.9 Configuración dinámica de anfitrión

Para manejar la asignación de direcciones de manera automática, el IETF ha diseñado un nuevo protocolo. Conocido como *Dynamic Host Configuration Protocol* (*Protocolo de configuración dinámica de anfitrión* o *DHCP*), el nuevo protocolo extiende BOOTP de dos formas. En primer lugar, el DHCP permite que una computadora adquiera toda la información que necesita en un solo mensaje. Por ejemplo, además de una dirección IP, un mensaje DHCP puede tener una máscara de subred. En segundo lugar, el DHCP permite que una computadora posea una dirección IP en forma rápida y dinámica. Para utilizar el mecanismo de asignación de direcciones dinámico DHCP, un administrador debe configurar un servidor DHCP supliendo un conjunto de direcciones IP. Cada vez que una computadora nueva se conecta a la red, la computadora contacta al servidor y solicita una dirección. El servidor selecciona una de las direcciones especificadas por el administrador y la asigna a la computadora.

Para ser completamente general, el DHCP permite tres tipos de asignación de direcciones; un administrador selecciona cómo responderá el DHCP a cada red o a cada anfitrión. Como BOOTP, el DHCP permite la *configuración manual,* mediante la cual un administrador puede configurar una dirección específica para una computadora específica. El DHCP también permite la *configuración automática,* por medio de la cual el administrador permite a un servidor DHCP asignar una dirección permanente cuando una computadora es conectada por primera vez a la red. Por último, el DHCP permite una *configuración dinámica* completa, con la cual el servidor "presta" una dirección para una computadora por tiempo limitado.

Como en BOOTP, el DHCP utiliza la identidad del cliente para decidir cómo proceder. Cuando un cliente contacta un servidor DHCP, envía un identificador, por lo general, la dirección de hardware del cliente. El servidor utiliza el identificador del cliente y la red a la que el cliente se ha conectado para determinar cómo asignar el cliente y la dirección IP. Así, el administrador tiene un control completo sobre la forma en que se asignan las direcciones. Un servidor puede configu-

rarse para asignar direcciones a computadoras específicas de manera estática (como BOOTP), mientras permite a otras computadoras obtener dinámicamente direcciones de manera permanente o temporal.

## 21.10 Asignación dinámica de direcciones IP

La asignación dinámica de direcciones es el más significativo y novedoso aspecto del DHCP. A diferencia de la asignación de direcciones estática, utilizada en BOOTP, la asignación de direcciones dinámica no es una transformación uno a uno, y el servidor no necesita conocer la identidad de un cliente *a priori*. En particular, un servidor DHCP puede ser configurado para permitir que una computadora arbitraria obtenga una dirección IP y comience la comunicación. Así, el DHCP permite diseñar sistemas que se autoconfiguren. Luego de que una computadora ha sido conectada a la red, la computadora se vale del DHCP para obtener una dirección IP y entonces configura su software TCP/IP a fin de utilizar la dirección. Por supuesto, la autoconfiguración está sujeta a restricciones administrativas —es el administrador el que decide qué servidor DHCP puede realizar la autoconfiguración. En resumen:

> *Como el DHCP permite a un anfitrión obtener todos los parámetros necesarios para la comunicación, sin la intervención manual, también permite la autoconfiguración. Ésta se encuentra sujeta, por supuesto, a restricciones administrativas.*

Para hacer posible la autoconfiguración, un servidor del DHCP comienza con un conjunto de direcciones IP que el administrador de red asigna al servidor para su manejo. El administrador especifica las reglas bajo las que opera el servidor. Un cliente DHCP negocia el uso de una dirección intercambiando mensajes con un servidor. En el intercambio, el servidor proporciona una dirección para el cliente y el cliente verifica que la dirección sea aceptable. Una vez que el cliente ha aceptado una dirección, puede comenzar a utilizar la dirección para comunicarse.

A diferencia de la asignación de direcciones estática, que asigna permanentemente cada dirección IP a un anfitrión específico, la asignación de direcciones dinámica es temporal. Decimos que un servidor DHCP *arrienda* una dirección a un cliente por un período de tiempo finito. El servidor especifica el período de arrendamiento cuando asigna la dirección. Durante el período de arrendamiento, el servidor no arrendará la misma dirección a ningún otro cliente. Al final del período de arrendamiento, sin embargo, el cliente debe renovar el arrendamiento o dejar de usar la dirección.

¿Cuánto debe durar un arrendamientò DHCP? El tiempo óptimo de arrendamiento depende en particular de la red y de las necesidades de un anfitrión. Por ejemplo, para garantizar que las direcciones puedan reciclarse con rapidez, las computadoras en una red utilizadas por estudiantes en un laboratorio universitario deben tener un corto período de arrendamiento (por ejemplo, una hora). En contraste, la red de una compañía podría utilizar un período de arrendamiento de un día o de una semana. Para adaptarse a todos los posibles ambientes, el DHCP no especifica un período de arrendamiento fijo y constante. De hecho, el protocolo permite que un cliente solicite un período de arrendamiento específico y permite a un servidor informar al cliente que el período de arrenda-

miento está garantizado. Así, un administrador puede decidir durante cuánto tiempo podrá asignar cada servidor una dirección a un cliente. En el caso extremo, el DHCP reserva un valor *infinito* para permitir un arrendamiento por un período de tiempo indeterminadamente largo, como la asignación de direcciones permanente utilizada en BOOTP.

## 21.11 Obtención de direcciones múltiples

Una computadora multianfitriona (*multi-homed*) está conectada a más de una red. Cuando una computadora como ésta arranca, puede necesitar información de configuración para cada una de sus interfaces. Como un mensaje BOOTP, un mensaje DHCP suele proporcionar información acerca de una interfaz. Una computadora con varias interfaces debe manejar cada interfaz por separado. Así, aun cuando describimos del DHCP que una computadora necesita sólo una dirección, el lector debe recordar que cada interfaz de una computadora multianfitriona puede ser un punto diferente en el protocolo.

BOOTP y DHCP utilizan la noción *relay agent* (agente relevador) para permitir que una computadora contacte un servidor en una red no local. Cuando un agente relevador recibe una solicitud de difusión desde un cliente, envía la solicitud hacia un servidor y luego devuelve la réplica del servidor al anfitrión. Los agentes relevadores pueden complicar la configuración multianfitriona puesto que un servidor puede recibir varias solicitudes desde una misma computadora. Sin embargo, aun cuando BOOTP y DHCP utilizan el término *identificador de cliente*, asumimos que un cliente multianfitrión envía un valor que identifica a una interfaz muy particular (por ejemplo, a una dirección de hardware única). Así, un servidor siempre será capaz de distinguir entre solicitudes de un multianfitrión, aunque el servidor reciba tal solicitud por medio de un agente relevador.

## 21.12 Estados de adquisición de direcciones

Cuando se utiliza el DHCP para obtener una dirección IP, el cliente se encuentra en 1 de 6 estados. El diagrama de estados de transición en la figura 21.4 muestra los eventos y los mensajes que ocasionan que un cliente cambie de estado.

Cuando un cliente arranca por primera vez, entra en el estado *INITIALIZE* (*INICIALIZAR*). Para comenzar a adquirir una dirección IP, el cliente primero contacta a todos los servidores DHCP en la red local. Para hacerlo, el cliente difunde un mensaje *DHCPDISCOVER* y cambia al estado con el nombre *SELECT*. Dado que el protocolo es una extensión de BOOTP, el cliente envía el mensaje *DHCPDISCOVER* en un datagrama UDP con el puerto de destino activado para el puerto BOOTP (es decir, el puerto *67*). Todos los servidores DHCP de la red local reciben el mensaje y los servidores que hayan sido programados para responder a un cliente en particular enviarán un mensaje *DHCPOFFER*. Así, un cliente puede recibir ceros o más respuestas.

Mientras permanece en el estado *SELECT* (*SELECCIONADO*), el cliente reúne respuestas *DHCPOFFER* desde los servidores DHCP. Cada oferta contiene información de configuración para el cliente junto con una dirección IP que el servidor ofrece para arrendar al cliente. El cliente debe seleccionar una de las respuestas (por ejemplo, la primera en llegar) y negociar con el servi-

dor un arrendamiento. Para ello, el cliente envía al servidor un mensaje *DHCPREQUEST* y entra al estado *REQUEST*. A fin de enviar un acuse de recibo de la recepción de la solicitud y comenzar el arrendamiento, el servidor responde con el envío de un *DHCPACK*. El arribo y el acuse de recibo hacen que el cliente cambie al estado *BOUND*, en el cual el cliente procede a utilizar la dirección. En resumen:

*Para utilizar el DHCP, un anfitrión debe volverse cliente y difundir un mensaje a todos los servidores de la red local. El anfitrión entonces reunirá los ofrecimientos de los servidores, seleccionará uno de ellos y verificará su aceptación por parte del servidor.*

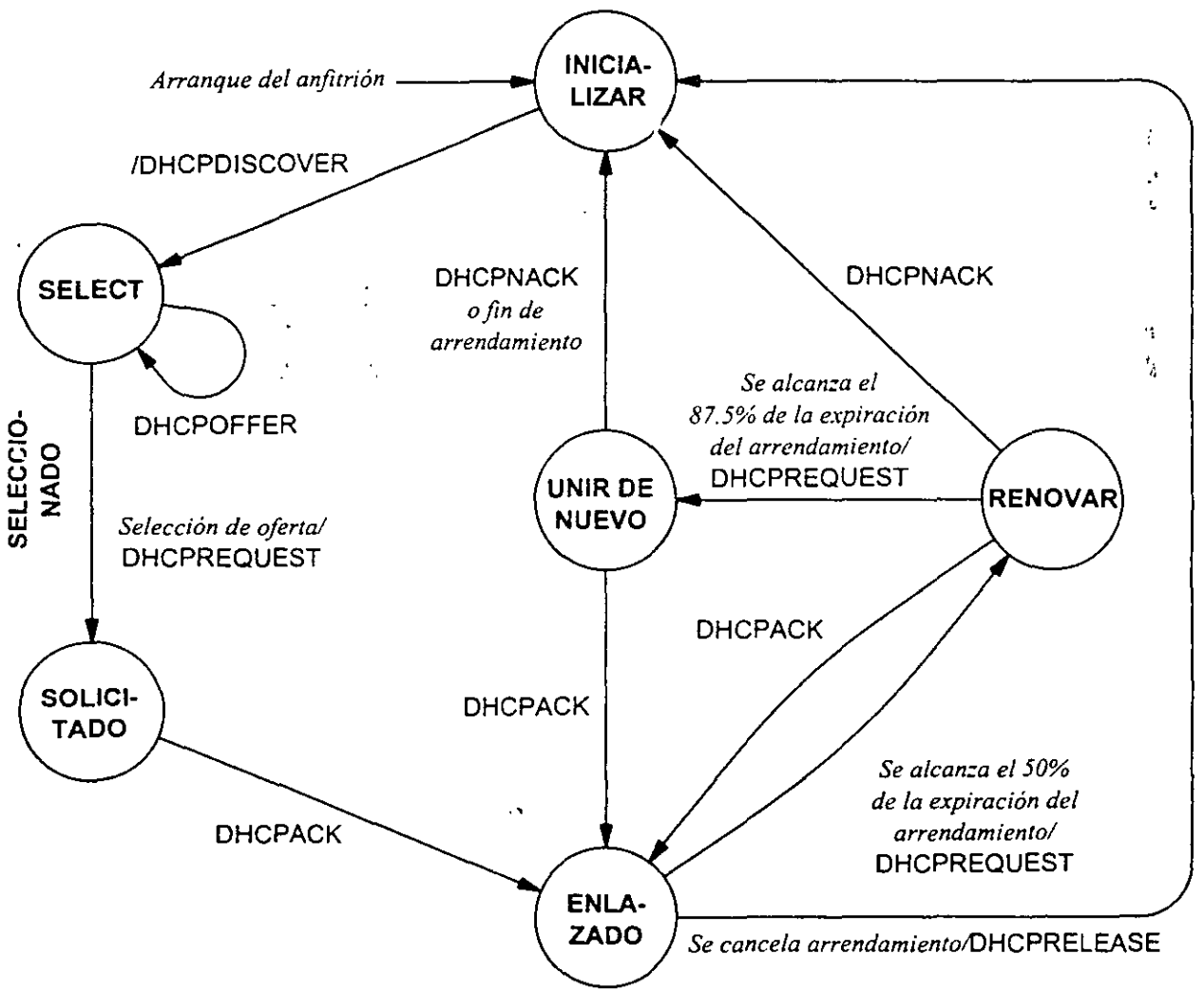



**Figura 21.4** Los seis estados principales de un cliente DHCP y las transiciones entre éstos. Cada nombre en una transición lista el mensaje entrante o el evento que ocasiona la transición, seguido por una diagonal y el mensaje que envía el cliente.

## 21.13 Terminación temprana de arrendamiento

Pensemos el estado *BOUND* como el estado normal de operación; un cliente por lo general se mantiene en un estado *BOUND* mientras utiliza la dirección IP que ha adquirido. Si un cliente tiene almacenamiento secundario (por ejemplo, un disco local), puede almacenar la dirección IP que le fue asignada y solicitar la misma dirección cuando arranque de nuevo. En algunos casos, sin embargo, un cliente en el estado *BOUND* puede descubrir que no necesita por más tiempo una dirección IP. Por ejemplo, supongamos que un usuario conecta una computadora portátil a una red, utiliza el DHCP para adquirir una dirección IP y, luego, se vale del TCP/IP para leer correo electrónico. El usuario puede no saber por cuánto tiempo leerá su correspondencia, o bien, la computadora portátil podría permitir al servidor seleccionar el período arrendado. En cualquier caso, el DHCP especifica un periodo de arrendamiento mínimo de 1 hora. Si después de obtener una dirección IP, el usuario descubre que no tiene mensajes de correo electrónico para leer podría optar por desconectar la computadora portátil y cambiar hacia otra localidad.

Cuando no es necesario un arrendamiento por más tiempo, el DHCP permite que el cliente lo termine sin esperar a que su tiempo expire. Tal terminación es muy útil en los casos en los que ni el cliente ni el servidor pueden determinar una terminación de arrendamiento apropiada, al mismo tiempo que se garantiza el arrendamiento pues le es posible a un servidor seleccionar un periodo de arrendamiento razonablemente largo. Una finalización temprana es importante en especial si el número de dirección IP que un servidor tiene disponible es mucho más pequeño que el número de computadoras que están conectadas a la red. Si cada cliente termina su arrendamiento en cuanto la dirección IP deja de ser necesaria, el servidor será capaz de asignar la dirección a otro cliente.

Para terminar un arrendamiento de manera temprana, el cliente envía un mensaje *DHCPRELEASE* al servidor. Liberar una dirección es una acción final que previene que el cliente continúe utilizando la dirección. Así, luego de transmitir el mensaje de liberación, el cliente no debe enviar ningún otro datagrama que utilice la dirección. En términos del diagrama de transición de estados de la figura 21.4, un anfitrión que envía una *DHCPRELEASE* deja el estado *BOUND* y debe comenzar de nuevo en el estado *INITIALIZE* antes de utilizar el IP.

## 21.14 Estado de renovación de arrendamiento

Dijimos que cuando se adquiere una dirección, un cliente DHCP cambia al estado *BOUND*. Al entrar al estado *BOUND*, el cliente instala tres temporizadores que controlan la renovación de arrendamiento, la reasignación y la expiración. Un servidor DHCP puede especificar valores explícitos para los temporizadores cuando asigna una dirección al cliente; si el servidor no especifica valores para el temporizador, el cliente empleará valores por omisión. El valor por omisión para el primer temporizador es la mitad del tiempo que tarda el arrendamiento. Cuando el primer temporizador expira, el cliente debe lograr la renovación de su arrendamiento. Para solicitar una renovación, el cliente envía un mensaje *DHCPREQUEST* hacia el servidor desde el que fue obtenido el arrendamiento. El cliente entonces cambia al estado *RENEW* (*RENOVAR*) en espera de una respuesta. *DHCPREQUEST* contiene la dirección IP del cliente que está utilizando actualmente e interroga al servidor para extender el arrendamiento en esa dirección. Como en la negociación inicial de arrendamiento, un cliente puede solicitar un período para la extensión, pero el servidor controla, en últi-

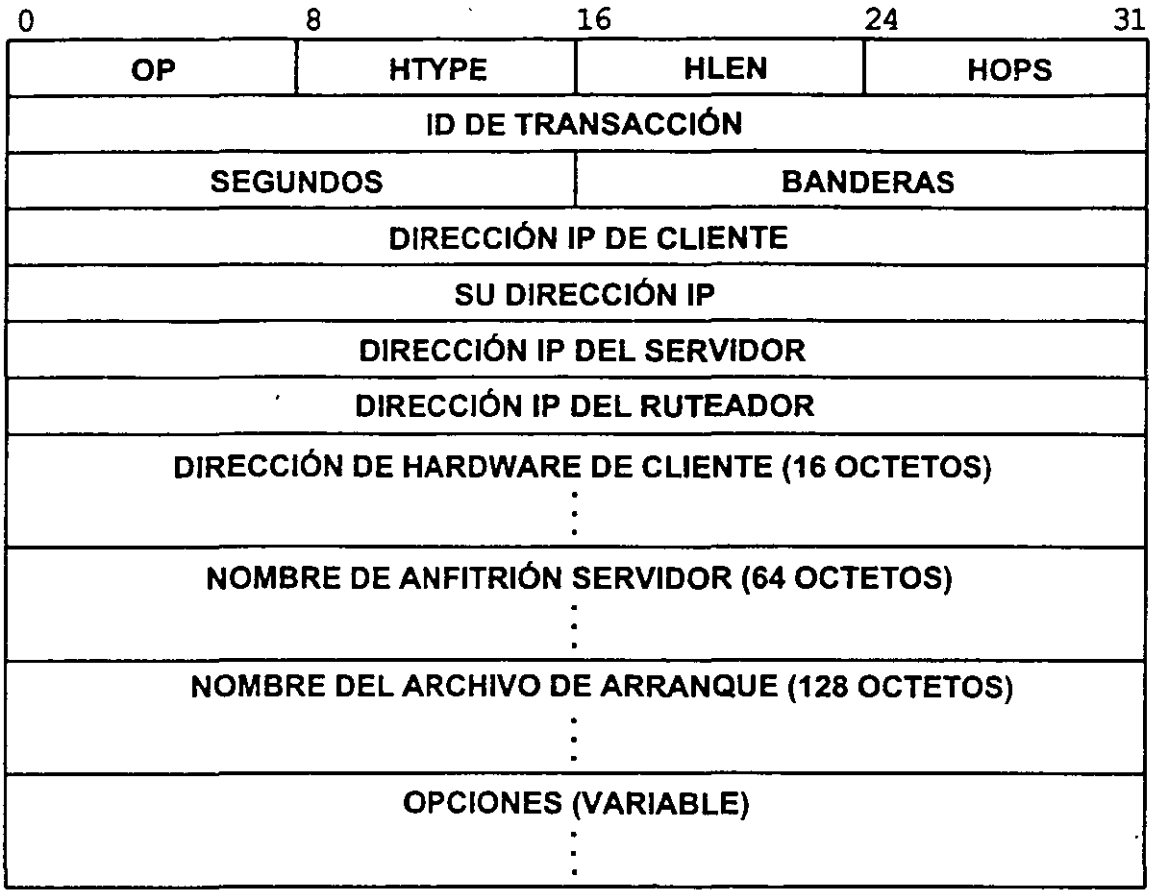



**Figura 21.5** Formato del mensaje DHCP, el cual es una extensión del mensaje BOOTP. El campo de opciones tiene una longitud variable; un cliente debe estar preparado para aceptar cuando menos 312 octetos por opción.

ma instancia, la renovación. Un servidor puede responder a la solicitud de renovación de un cliente de una de dos formas: puede instruir al cliente para que deje de usar la dirección o aprobar que la continúe utilizando. Si se aprueba esto último, el servidor envía un *DHCPACK*, el cual hace que el cliente regrese al estado *BOUND* y continúe utilizando la dirección. El *DHCPACK* puede también contener valores nuevos para los temporizadores del cliente. Si un servidor desaprueba que se continúe utilizando la dirección, el servidor enviará una *DHCPNACK* (acuse de recibo negativo), el cual hace que el cliente deje de utilizar la dirección inmediatamente y regrese al estado *INITIALIZE*.

Luego de enviar un mensaje *DHCPREQUEST* en el que solicite una extensión de su arrendamiento, el cliente se mantiene en el estado *RENEW* en espera de una respuesta. Si no se obtiene ninguna respuesta, el servidor que garantiza el arrendamiento se considera inactivo o inaccesible. Para manejar la situación, el DHCP libera un segundo temporizador, el cual fue instalado cuando el cliente entró al estado *BOUND*. El segundo temporizador expira luego de que se cumple el 87.5% del periodo de arrendamiento y hace que el cliente pase del estado *RENEW* al estado *REBIND* (*UNIR DE NUEVO*). Cuando se realiza la transición, el cliente asume que el anterior servidor DHCP no está disponible y comienza a difundir un mensaje *DHCPREQUEST* hacia cualquier servidor en la red local. Cualquier servidor configurado para proporcionar servicio a un cliente puede responder de manera positiva (por ejemplo, para extender el arrendamiento) o negativamente (esto

es, para no permitir que se siga usando la dirección IP). Si recibe una respuesta positiva, el cliente vuelve al estado *BOUND* y reinicializa los dos temporizadores. Si recibe una respuesta negativa, debe cambiar al estado *INITIALIZE*, dejar de usar inmediatamente la dirección IP y adquirir una nueva dirección IP antes de continuar utilizando el IP.

Luego de cambiar al estado *REBIND*, el cliente tendrá que interrogar al servidor original y a todos los servidores en la red local para una extensión del arrendamiento. En dado caso de que el cliente no reciba una respuesta de ningún servidor antes de que expire su tercer temporizador, el arrendamiento expirará. El cliente debe dejar de utilizar la dirección IP, regresar al estado *INITIALIZE* y comenzar la adquisición de una nueva dirección.

## 21.15 Formato de los mensajes DHCP

Como se muestra en la figura 21.5, el DHCP se vale del formato de mensaje BOOTP, pero modifica el contenido y el significado de algunos campos.

Como se muestra en la figura, casi todos los campos en un mensaje DHCP son idénticos a los de un mensaje BOOTP. De hecho, los dos protocolos son compatibles; un servidor DHCP puede ser programado para responder solicitudes BOOTP. Sin embargo, el DHCP cambia el significado de dos campos. Primero DHCP, interpreta el campo *UNUSED* (*SIN USO*) de BOOTP como un campo *FLAGS* (*BANDERAS*) de 16 bits. La figura 21.6 muestra que sólo el bit de orden superior del campo *FLAGS* tiene asignado un significado.

Como el mensaje de solicitud DHCP contiene la dirección de hardware del cliente, un servidor DHCP normalmente envía sus respuestas al cliente mediante una difusión de hardware. El cliente activa el bit de orden superior en el campo *FLAGS* para solicitar que el servidor responda por medio de la difusión y no de la unidifusión de hardware. Para entender por qué un cliente debe seleccionar una respuesta de difusión, recordemos que, cuando un cliente se comunica con un servidor DHCP, ya no tiene una dirección IP. Si un datagrama llega por medio de la unidifusión de hardware, y la dirección de destino no concuerda con la dirección de la computadora, el IP puede descartar el datagrama. Sin embargo, se requiere el IP para aceptar y manejar cualquier datagrama enviado hacia la dirección de difusión IP. Para asegurar que el software IP acepte y entregue mensajes DHCP que lleguen antes de que la dirección IP de la máquina se haya configurado, el cliente DHCP puede solicitar que el servidor envíe respuestas mediante la difusión IP.

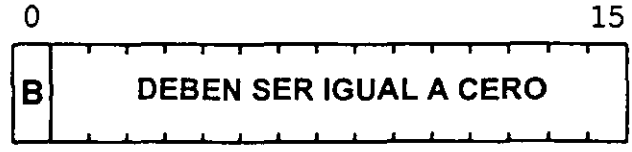



**Figura 21.6** Formato del campo *FLAGS* (*BANDERAS*) de 16 bits en un mensaje DHCP. El bit de la extremo izquierdo se interpreta como una solicitud de difusión;

## 21.16 Opciones y tipos de mensajes DHCP

Sorprendentemente, el DHCP no añade nuevos campos fijos para el formato de los mensajes BOOTP, ni cambia el significado de la mayor parte de los campos. Por ejemplo, el campo *OP* en un mensaje DHCP contiene los mismos valores que el campo *OP* en un mensaje BOOTP: el mensaje es una solicitud de arranque (valor *1*) o una réplica de arranque (valor *2*). Para codificar información como la duración del arrendamiento, el DHCP utiliza *opciones*. En particular, la figura 21.7 ilustra la opción *tipo de mensaje DHCP* utilizada para especificar qué mensaje DHCP se está enviando.

El campo opciones tiene el mismo formato que la *VENDOR SPECIFIC AREA*, asimismo el DHCP acepta todos los temas de información de vendedores específicos definidos para BOOTP. Como en BOOTP, cada opción consiste en un campo de código y de un campo de longitud de 1 octeto respectivamente, seguidos por los octetos de datos que comprenden la opción. Como se muestra en la figura, la opción utilizada para especificar el tipo de mensaje DHCP consiste exactamente en 3 octetos. El primer octeto contiene el código *53*, el segundo la longitud *1* y el tercero un valor utilizado para identificar uno de los posibles mensajes DHCP.


| CAMPO DE TIPO | Tipo de mensaje DHCP correspondiente |
|---|---|
| 1 | DHCPDISCOVER |
| 2 | DHCPOFFER |
| 3 | DHCPREQUEST |
| 4 | DHCPDECLINE |
| 5 | DHCPPACK |
| 6 | DHCPNACK |
| 7 | DHCPRELEASE |

**Figura 21.7** Formato de una opción de tipo de mensaje del DHCP utilizado para especificar el mensaje DHCP que se está enviando. La tabla lista posibles valores del tercer octeto y sus significados.

## 21.17 Opción Overload

Los campos *SERVER HOST NAME* y *BOOT FILE NAME* en el encabezado del mensaje DHCP ocupan muchos octetos. Si un mensaje dado no contiene información, en ninguno de estos campos, el espacio se desperdicia. Para permitir que un servidor DHCP utilice los dos campos para otras secciones, el DHCP define una opción *Option Overload*. Cuando está presente, la opción de sobrecarga informa al receptor que debe ignorar el significado usual de los campos *SERVER HOST NAME* y *BOOT FILE NAME*, y que debe considerar las opciones que están en lugar de los campos

## 21.18 DHCP y nombres de dominios[3]

Aun cuando puede asignar direcciones IP a una computadora que lo demande, el DHCP no automatiza por completo todo el procedimiento requerido para conectar un anfitrión permanente a una red de redes. En particular, el DHCP no interactúa con el sistema de nombre de dominio. Así, la asignación entre un nombre de anfitrión y la asignación DHCP de la dirección IP del anfitrión se deben manejar de manera independiente.

¿Qué nombre deberá recibir el anfitrión cuando tenga una dirección IP desde DHCP? Conceptualmente, hay tres posibilidades. En la primera, el anfitrión no recibe un nombre. Aun cuando es posible correr software del cliente en un anfitrión sin un nombre, utilizar una computadora sin nombre puede ser inconveniente. En segundo lugar, el anfitrión está asignado de manera automática a un nombre junto con una dirección IP. Este método es muy popular en la actualidad ya que los nombres pueden ser preasignados y no se requieren cambios para DNS. Por ejemplo, un administrador de sistema puede configurar el servidor de nombre de dominios local a fin de tener un nombre de anfitrión para cada dirección IP manejada por DHCP. Una vez que ha sido instalado en DNS, la asignación nombre-a-dirección se mantiene estática. La mayor desventaja de la asignación estática es que al anfitrión se le da un nombre nuevo cada vez que recibe una nueva dirección (por ejemplo, cuando un anfitrión cambia de una red física a otra). En tercer lugar, el anfitrión puede ser asignado a un nombre permanente que se mantiene sin cambios. Conservar un nombre de anfitrión de manera permanente es conveniente pues la computadora puede ser accesada siempre por medio de un solo nombre, independientemente de la localización actual de la computadora.

Se necesitan mecanismos adicionales para soportar nombres de anfitrión permanentemente. En particular, los nombres de anfitrión permanentes requieren de una coordinación entre DHCP y DNS. Un servidor DNS debe cambiar la asignación, nombre-a-dirección cada vez que un anfitrión reciba una dirección IP y retirar la asignación cuando expire el arrendamiento. No obstante, un grupo de trabajo IETF está considerando actualmente cómo hacer que interactúe DHCP con el sistema de nombre de dominios. De momento, no hay protocolos para actualizaciones DNS dinámicos. Así pues, hasta que se desarrolle un mecanismo de actualización dinámico, no habrá protocolo que mantenga nombres de anfitrión de manera permanente y que permita a al DHCP cambiar direcciones IP.

## 21.19 Resumen

El protocolo de arranque BOOTP proporciona una alternativa a RARP para computadoras que necesitan determinar su dirección IP. BOOTP es más general que RARP pues utiliza el UDP, lo que hace posible extender el proceso de arranque a través de un ruteador. BOOTP también permite a una máquina determinar una dirección de ruteador, una dirección (archivo) de servidor y el nombre de un programa que la computadora deberá correr. Finalmente, BOOTP permite a los administradores establecer la configuración de una base de datos que transforma un nombre genérico como

[3] En el capítulo 22, se considera el Sistema de Nombres de Dominio en detalle.

"unix" en un nombre de archivo completamente caracterizado que contiene la imagen de memoria apropiada para el hardware del cliente.

BOOTP está diseñado para ser lo suficientemente pequeño y simple como para residir en un arranque localizado en ROM. El cliente utiliza la dirección de difusión limitada para comunicarse con el servidor y tiene la responsabilidad de retransmitir solicitudes si el servidor no responde. La retransmisión emplea un procedimiento de retroceso exponencial similar a Ethernet para evitar el congestionamiento.

Diseñado como sucesor de BOOTP, el Dynamic Host Configuration Protocol (DHCP) amplía BOOTP de varias formas. Lo más importante es que el DHCP permite que un servidor localice direcciones IP automática o dinámicamente. La asignación dinámica es necesaria para ambientes como las redes inalámbricas cuyas computadoras pueden conectarse y desconectarse rápidamente. Para utilizar el DHCP, una computadora debe convertirse en cliente. La computadora difunde una solicitud para los servidores DHCP, selecciona una de las ofertas recibidas e intercambia mensajes con el servidor a fin de obtener un arrendamiento de la dirección IP anunciada.

Cuando un cliente obtiene una dirección IP, arranca 3 temporizadores. Luego de que el primer temporizador expira, el cliente debe intentar la renovación de su arrendamiento. Si un segundo temporizador expira antes de que se complete la renovación, el cliente debe tratar de reasignar su dirección a cualquier servidor. Si el último temporizador expira antes de que se haya obtenido un arrendamiento, el cliente deja de utilizar la dirección IP y regresa al estado inicial para adquirir una nueva dirección. Una máquina de estado finito explica la adquisición de arrendamiento y su renovación.

## PARA CONOCER MÁS

BOOTP es un protocolo estándar en la serie del TCP/IP. Se pueden obtener mayores detalles en Croft y Gilmore (RFC 951), en el cual se compara BOOTP con RARP y sirve como estándar oficial. Reynolds (RFC 1084) explica cómo interpretar el área de vendedor específico y Braden (RFC 1123) recomienda utilizar el área de vendedor específico para transferir la máscara de subred.

Droms (RFC 1541) presenta la última especificación para el DHCP, así como una descripción detallada de las transiciones de estado; se espera pronto otra revisión. En un documento relacionado, Alexander (RFC 1533) especifica la codificación de las opciones DHCP y la extensión de vendedor BOOTP. Por último, Droms (RFC 1534) analiza la interoperabilidad entre BOOTP y DHCP.

## EJERCICIOS

**21.1** BOOTP no contiene un campo explícito para volver a poner la hora del día del servidor al cliente, pero lo hace parte (opcional) de la información del vendedor específico. ¿La hora debería ser incluida en los campos requeridos? Explique por qué sí o por qué no.

**21.2** Exponga qué separación de configuración y almacenamiento *no* es buena. (Sugerencia: consulte el

**21.3** El formato de mensaje BOOTP es inconsistente pues tiene dos campos para la dirección IP de cliente y uno para el nombre de la imagen de arranque. Si el cliente deja su campo de dirección IP vacío, el servidor devuelve la dirección IP del cliente en el segundo campo. Si el cliente deja el campo de nombre del archivo de arranque vacío, el servidor lo *reemplaza* con un nombre explícito. ¿Por qué?

**21.4** Lea el estándar para encontrar cómo utilizan clientes y servidores el campo *HOPS*.

**21.5** Cuando un cliente BOOTP recibe una réplica por medio de la difusión de hardware, ¿cómo sabe si la réplica está dirigida a otro cliente BOOTP en la misma red física?

**21.6** Cuando una máquina obtiene una máscara de subred con BOOTP en lugar de ICMP, coloca la carga menor en *otras* computadoras anfitrión. Explíquelo.

**21.7** Lea el estándar para encontrar cómo pueden acordar un cliente DHCP y un servidor la duración de arrendamiento sin tener relojes sincronizados.

**21.8** Considere un anfitrión que tiene un disco y utiliza DHCP para obtener una dirección IP. Si el anfitrión almacena su dirección en un disco junto con la fecha en que expira el arrendamiento y luego se reinicializa dentro del periodo de arrendamiento, ¿puede utilizar la dirección? ¿Por qué sí o por qué no?

**21.9** El DHCP establece un arrendamiento de dirección mínimo de una hora. ¿Puede usted imaginar una situación en la que el arrendamiento mínimo del DHCP provoque inconvenientes? Explíquelo.

**21.10** Lea el RFC para encontrar cómo especifica el DHCP la renovación y la reasignación de temporizadores. ¿Un servidor debe establecer siempre uno sin el otro? ¿Por qué sí o por qué no?

**21.11** El diagrama de transición de estado no muestra la retransmisión. Lea el estándar para encontrar cómo muchas veces debe retransmitir un cliente una solicitud.

**21.12** ¿Puede el DHCP garantizar que un cliente no es "engañado" (es decir, puede el DHCP garantizar que no está enviando información de configuración del anfitrión *A* al anfitrión *B*)? ¿La respuesta difiere para BOOTP? Explique por qué sí o por qué no.

**21.13** El DHCP especifica que un anfitrión debe prepararse para manejar por lo menos *312* octetos de opciones. ¿Cómo se obtiene el número *312*?

**21.14** ¿Puede una computadora que utiliza al DHCP obtener una dirección IP operando un servidor? Si así es, ¿cómo accede un cliente al servidor?

# 29

# El futuro de TCP/IP (IPng, IPv6)

## 29.1 Introducción

La evolución de la tecnología TCP/IP está vinculada a la evolución de Internet por varias razones. En primer lugar, Internet es la red de redes del TCP/IP instalada más extensa, de manera que muchos problemas aparecen en Internet antes de que salgan a la superficie en otras redes de redes TCP/IP. En segundo lugar, los investigadores e ingenieros fundadores del TCP/IP provienen de compañías y dependencias gubernamentales que utilizan Internet, de manera que tienden a fundar proyectos que impactan a Internet. En tercer lugar, la mayoría de los investigadores participantes en el TCP/IP tienen conexiones con Internet y la utilizan diariamente. Así pues, tienen una motivación inmediata para resolver problemas que mejorarán el servicio y ampliarán su funcionalidad.

Con millones de usuarios en decenas de miles de localidades alrededor del mundo que dependen de la red global de Internet como parte de su ambiente diario de trabajo, puede parecer que Internet es una infraestructura de producción estable. Hemos pasado de las primeras etapas de desarrollo, en las que los usuarios eran también expertos, a una etapa en la cual pocos usuarios comprenden la tecnología. Sin embargo, a pesar de las apariencias, ni Internet ni el conjunto de protocolos TCP/IP son estáticos. Nuevos grupos conectan sus redes y descubren nuevas formas de utilizar la tecnología. Los investigadores resuelven nuevos problemas de redes y los ingenieros mejoran los mecanismos subyacentes. En pocas palabras, la tecnología continúa evolucionando.

El propósito de este capítulo es considerar el proceso de evolución actual y examinar uno de los más importantes esfuerzos de ingeniería. En particular, veremos una propuesta de revisión del IP. Si la propuesta es aprobada como estándar y adoptada por los vendedores, tendrá un mayor impacto en el TCP/IP y en Internet. Si el nuevo protocolo se hace parte del TCP/IP en los próximos meses, años o décadas es irrelevante; el objetivo es hacer que el lector comprenda el esfuerzo. El

lector deberá estar consciente de que la propuesta no es un estándar final y que los detalles pueden cambiar.

## 29.2 ¿Por qué cambiar TCP/IP e Internet?

La tecnología básica TCP/IP ha funcionado bien por una década. ¿Por qué debería cambiarse? En términos generales, los procesos que estimulan la evolución del TCP/IP y de la arquitectura de Internet se pueden clasificar dentro de cuatro categorías. Luego de describir cada categoría, examinaremos la propuesta de una nueva versión del IP y veremos cómo cada categoría afecta al diseño.

### 29.2.1 Nuevas tecnologías de comunicación y computación

Como en la mayoría de los grupos orientados hacia la tecnología, los investigadores e ingenieros que trabajan en los protocolos TCP/IP mantienen un agudo interés por las nuevas tecnologías. Tan pronto como un nueva computadora de alta velocidad está disponible, la utilizan en anfitriones y ruteadores. En cuanto una nueva tecnología de red emerge, la utilizan para transportar datagramas IP. Por ejemplo, además de las LAN y las líneas convencionales de comunicación serial arrendadas, los investigadores del TCP/IP han estudiado la comunicación punto a punto vía satélite, las estaciones múltiples de satélites sincronizados, los paquetes de radio y ATM. Más recientemente, los investigadores han estudiado las redes inalámbricas que se valen de luz infrarroja o las tecnologías de frecuencias de radio de espectro extendido.

### 29.2.2 Nuevas aplicaciones

Las nuevas aplicaciones constituyen una de las fronteras de investigación y desarrollo de Internet más interesantes y por lo general crean una demanda de infraestructura o servicios que los protocolos actuales no pueden proporcionar. Por ejemplo, el interés creciente en multimedios ha creado una demanda de protocolos que puedan transferir imágenes y sonido eficientemente. De la misma forma, el interés en la comunicación en tiempo real de audio y video ha creado una demanda de protocolos que puedan garantizar la entrega de la información con retardos fijos, así como protocolos que puedan sincronizar audio y video con flujos de datos.

### 29.2.3 Incrementos en el tamaño y en la carga

La red global de Internet ha tenido varios años de crecimiento exponencial, duplicando su tamaño cada nueve meses o más rápido. A principios de 1994, en promedio, un nuevo anfitrión aparecía cada 30 segundos, y la cantidad se incrementó de manera dramática. Sorpresivamente, la carga de tráfico en Internet ha crecido más rápido que el número de redes. El incremento en el tráfico puede atribuirse a varias causas. En primer lugar, la población de Internet está cambiando su composición respecto al público en general, deja de estar formada por académicos e investigadores. En conse-

cuencia, la gente ahora utiliza Internet luego de sus horas de trabajo para actividades comerciales y de entretenimiento. En segundo lugar, las nuevas aplicaciones que transfieren imágenes y video en tiempo real generan más tráfico que las aplicaciones que transfieren texto. En tercer lugar, las herramientas de búsqueda automatizada generan una cantidad sustancial de tráfico y lo hacen más lento al sondear en las localidades de Internet para encontrar datos.

### 29.2.4 Nuevas políticas

Conforme se expande hacia nuevas industrias y nuevos países, Internet cambia de forma fundamental: adquiere nuevas autoridades administrativas. Los cambios en la autoridad producen cambios en las políticas administrativas y se establecen nuevos mecanismos para reforzar tales políticas. Como hemos visto, la arquitectura de conexión de Internet y los protocolos que utiliza comprenden un modelo de núcleo centralizado. La evolución continúa conforme se conectan más columnas vertebrales de redes nacionales, produciendo un incremento complejo de políticas que regulan la interacción. Cuando diversas corporaciones interconectan redes TCP/IP privadas enfrentan problemas similares al tratar de definir políticas de interacción y encontrar mecanismos para reforzar estas políticas. Así, muchos de los esfuerzos de investigadores e ingenieros alrededor del TCP/IP continúan enfocados a encontrar formas de adaptarse a nuevos grupos administrativos.

## 29.3 Motivos para el cambio del IPv4

La versión *4* del protocolo de Internet (*IPv4*) proporciona los mecanismos de comunicación básicos del conjunto TCP/IP y la red global Internet; se ha mantenido casi sin cambio desde su inserción a fines de los años setenta.[1] La antigüedad de la versión *4* muestra que el diseño es flexible y poderoso. Desde el momento en que se diseñó el IPv4, el desempeño de los procesadores se ha incrementado en dos órdenes de magnitud, el tamaño de las memorias se ha incrementado por un factor de 32, el ancho de banda de la columna vertebral de la red Internet se ha incrementado en un factor de 800, las tecnologías LAN han emergido y el número de anfitriones en Internet ha crecido hasta llegar un total de 4 millones. Además, los cambios no ocurren de manera simultánea —el IP se ha adaptado a los cambios de una tecnología antes de adaptarse a los cambios de otras.

A pesar de su diseño, el IPv4 también debe ser reemplazado. En el capítulo 10, se describe las principales motivaciones para actualizar el IP: el inminente agotamiento del espacio de direcciones. Cuando el IP se diseñó, un espacio de 32 bits era más que suficiente. Sólo un puñado de organizaciones utilizaba las LAN; pocas tenían una WAN corporativa. Ahora, sin embargo, muchas corporaciones de tamaño mediano tienen varias LAN y varias de las grandes corporaciones cuentan con una WAN corporativa. En consecuencia, el espacio de direcciones IP de 32 bits que se usa actualmente no puede adaptarse al crecimiento proyectado de la red global de Internet.

Aun cuando la necesidad de un espacio de direcciones extenso está forzando un cambio inmediato en el IP, hay otros factores que también contribuyen. En particular, gran parte de éstos se refieren al soporte de nuevas aplicaciones. Por ejemplo, debido a que el audio y el video en tiempo

real necesitan determinadas garantías en los retardos, una nueva versión del IP debe proporcionar un mecanismo que haga posible asociar un datagrama con una reservación de fuente preasignada. Además, como varias de las nuevas aplicaciones de Internet necesitan comunicaciones seguras, una nueva versión del IP deberá incluir capacidades que hagan posible autenticar al emisor.

## 29.4 El camino hacia una nueva versión del IP

Los grupos en el IETF han estado trabajando para formular una nueva versión del IP por varios años. Como tratan de producir un estándar *abierto*, el IETF ha invitado a toda la comunidad a participar en el proceso de estandarización. En consecuencia, investigadores, fabricantes de computadoras, vendedores de hardware y software de red, programadores, administradores, usuarios, compañías telefónicas y televisoras por cable han especificado sus requerimientos para la próxima versión IP y han comentado todos sus propuestas específicas.

Se han propuesto muchos diseños para servir a un propósito en particular o a una comunidad en especial. Uno de los diseños propuestos haría al IP más sofisticado y el costo por el incremento en la complejidad de procesamiento se elevaría. Otro diseño propone utilizar una modificación del protocolo CLNS de OSI. Un tercer diseño mayor propone conservar la mayor parte de las ideas del IP, y hacer extensiones para adaptarlo a direcciones extensas. El diseño conocido como *SIP (Simple IP)* ha sido la base para una propuesta extendida que incluye ideas de otras propuestas. La versión extendida del SIP ha sido llamada *Simple IP Plus* (*SIPP*) y finalmente emerge como el diseño elegido como base para el próxima IP.

Seleccionar una nueva versión del IP no ha sido fácil. La popularidad de Internet hace que el mercado de productos IP alrededor del mundo se tambalee. Muchos grupos consideran esto como una oportunidad económica y tratan de que la nueva versión del IP les ayude a obtener ganancias sobre sus competidores. Además, se han involucrado algunas personalidades —algunas opiniones técnicas individuales se mantienen fuertemente; otros consideran la participación activa como una manera de hacerse promoción. En consecuencia, las discusiones han generado argumentaciones acaloradas.

## 29.5 Nombre del próximo IP

Al comienzo de las discusiones sobre el cambio del IP, el IAB publicó una declaración política que se refería a la próxima versión como *IP version 7*, el informe causó una confusión general. La gente preguntaba "¿qué sucedió con la versión *5* y *6*?" ¿El IAB se refiere a la *versión 5*, o se refiere al establecimiento de una política para un futuro a largo plazo. Evidentemente, el error ocurrió porque el protocolo ST estaba asignado como la versión número *5* y uno de los documentos disponibles por el IAB reportaba erróneamente a la versión actual como la versión *6*.

Para evitar la confusión, el IETF cambió el nombre. Retomando el nombre de una popular serie de televisión, el IETF eligió "IP — la próxima generación" y el esfuerzo comenzó a conocerse como *IPng*.

Formalmente, se ha decidido que a la próxima versión del IP se le asigne el número de versión *6*. Así, para distinguirlo de la versión actual del IP (*IPv4*), la próxima generación se llamará *IPv6*. En el pasado, el término *IPng* ha sido utilizado en un contexto amplio para referirse a todas las discusiones y propuestas para una próxima versión del IP, mientras que el término *IPv6* [2] se ha utilizado para referirse a una propuesta específica que proviene del IETF. La literatura actual a menudo trata a los dos términos como sinónimos y los emplea de manera indistinta. Para ayudar a distinguir la discusión general respecto de la propuesta actual, utilizaremos el término *IPv6* para referirnos al protocolo específico que ha sido propuesto e *IPng* para referirnos a todos los esfuerzos relacionados con el desarrollo de una nueva generación del IP.

## 29.6 Características del IPv6

El protocolo IPv6 propuesto conserva muchas de las características que contribuyeron al éxito del IPv4, de hecho, los diseñadores han caracterizado al IPv6 como si fuera básicamente el mismo que el IPv4 con unas cuantas modificaciones. Por ejemplo, el IPv6 todavía soporta la entrega sin conexión (es decir, permite que cada datagrama sea ruteado independientemente), permite al emisor seleccionar el tamaño de un datagrama y requiere que el emisor especifique el máximo número de saltos que un datagrama puede realizar antes de ser eliminado. Como veremos, el IPv6 también conserva la mayor parte de los conceptos proporcionados por la versión IPv4, incluyendo capacidades de fragmentación y ruteo de fuente.

A pesar de las similitudes conceptuales, el IPv6 cambia la mayor parte de los detalles de protocolo. Por ejemplo, el IPv6 utiliza direcciones largas y añade unas cuantas características nuevas. Algo muy importante, el IPv6 revisa completamente el formato de los datagramas, remplazando el campo de opción de longitud variable del IPv4 por una serie de encabezados de formato fijo. Examinaremos los detalles luego de considerar los cambios mayores y las motivaciones subyacentes para cada uno.

Los cambios introducidos para el IPv6 pueden agruparse en cinco categorías:

- *Direcciones más largas.* El nuevo tamaño de las direcciones es el cambio más notable. El IPv6 cuadruplica el tamaño de las direcciones del IPv4, va de 32 bits a 128 bits. El espacio de direcciones del IPv6 es tan grande que no podrá agotarse en un futuro previsible.
- *Formato de encabezados flexible.* El IPv6 utiliza un formato de datagrama incompatible y completamente nuevo. A diferencia del IPv4, que utiliza un encabezado de datagrama de formato fijo en el que todos los campos excepto las opciones ocupan un número fijo de octetos en un desplazamiento fijo, el IPv6 utiliza un conjunto de encabezados opcionales.
- *Opciones mejoradas.* Como el IPv4, el IPv6 permite que un datagrama incluya información de control opcional. El IPv6 incluye nuevas opciones que proporcionan capacidades adicionales no disponibles en el IPv4.

[2] Algunos autores utilizan la abreviatura *IP6*

- *Soporte para asignación de recursos.* El IPv6 reemplaza la especificación del tipo de servicio del IPv4 con un mecanismo que permite la preasignación de recursos de red. En particular, el nuevo mecanismo soporta aplicaciones como video en tiempo real que requieren de una garantía de ancho de banda y retardo.
- *Provisión para extensión de protocolo.* Posiblemente el cambio más significativo en el IPv6 es el cambio de un protocolo que especifica completamente todos los detalles a un protocolo que puede permitir características adicionales. La capacidad de extensión tiene la posibilidad de permitir que el IETF se adapte a los protocolos para cambiar al hardware de red subyacente o a nuevas aplicaciones.

## 29.7 Forma general de un datagrama IPv6

El IPv6 cambia completamente el formato de datagrama. Como se muestra en la figura 29.1, un datagrama IPv6 tiene un *encabezado base* de tamaño fijo, seguido por ceros o más *encabezados de extensión*, seguidos a su vez por datos.

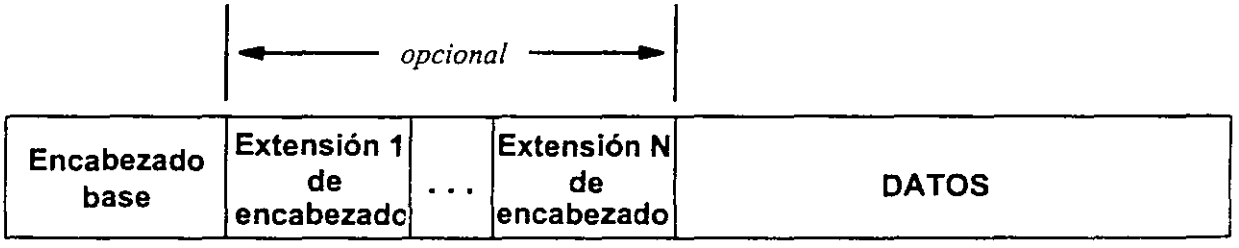



**Figura 29.1** Forma general de un datagrama IPv6 con varios encabezados. Sólo el encabezado base es indispensable, los encabezados de extensión son opcionales.

## 29.8 Formato del encabezado base del IPv6

Es interesante que, aun cuando debe adaptarse a direcciones extensas, un encabezado base IPv6 contiene menos información que un encabezado de datagrama IPv4. Las opciones y algunos de los campos fijos que aparecen en un encabezado de datagrama del IPv4 se han cambiado por encabezados de extensión en el IPv6. En general, el cambio en los encabezados en los datagramas refleja los cambios en el protocolo:

- La alineación se ha cambiado de múltiplos de 32 bits a múltiplos de 64 bits.
- Los campos de longitud de encabezado se han eliminado y el campo de longitud de datagrama ha sido reemplazado por el campo *PAYLOAD LENGTH* (*LONGITUD PAYLOAD*).

- El tamaño de los campos de dirección de fuente y destino se ha incrementado en 16 octetos cada uno.
- La información de fragmentación se ha movido de los campos fijos en el encabezado base, hacia un encabezado de extensión.
- El campo *TIME-TO-LIVE* (*LÍMITE DE SALTO*) ha sido reemplazado por el *HOP LIMIT.*
- El campo *SERVICE TYPE* ha sido reemplazado por el campo *FLOW LABEL* (*ETIQUETA DE FLUJO*).
- El campo *PROTOCOL* ha sido reemplazado por un campo que especifica el tipo del próximo encabezado.

La figura 29.2 muestra el contenido y el formato de un encabezado base IPv6. Varios campos en un encabezado base IPv6 corresponden directamente a los campos en un encabezado IPv4. Como en el IPv4, el campo inicial *VERS* de 4 bits especifica la versión del protocolo; *VERS* siempre contiene el número 6 en un datagrama IPv6. Como en el IPv4, los campos *SOURCE ADDRESS* (*DIRECCIÓN FUENTE*) y *DESTINATION ADDRESS* (*DIRECCIÓN DE DESTINO)* especifican la dirección del emisor y del recipiente. En el IPv6, sin embargo, cada dirección requiere 16 octetos. El campo *HOP LIMIT* corresponde al campo *TIME- TO-LIVE* del IPv4. A diferencia del IPv4, que interpreta un tiempo límite como una combinación de conteo de saltos y tiempo máximo, el IPv6 interpreta el valor como un límite estricto del máximo número de saltos que un datagrama puede realizar antes de ser desechado.

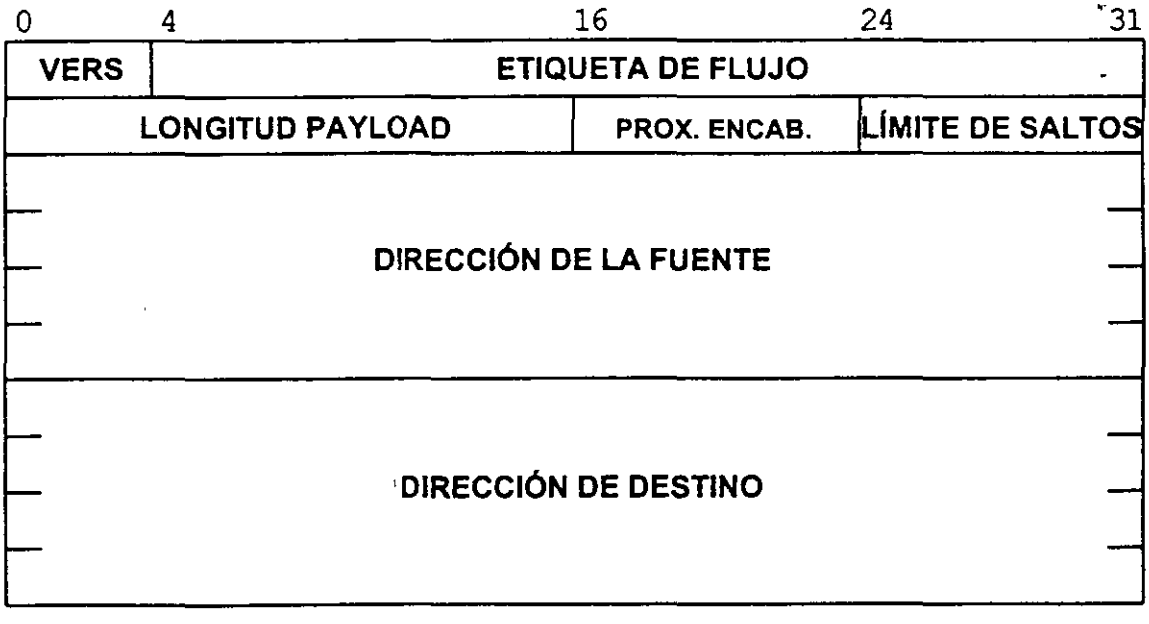



**Figura 29.2** Formato del encabezado base de 40 octetos del IPv6. Cada datagrama IPv6 comienza con un encabezado base.

El IPv6 maneja las especificaciones de longitud de datagramas de forma nueva. En primer lugar, debido a que el tamaño del encabezado base se fijó en 40 octetos, dicho encabezado no incluye un campo para la longitud de encabezado. En segundo lugar, el IPv6 reemplaza el campo de longitud de datagrama del IPv4 por un campo *PAY LOAD LENGTH* de 16 bits que especifica el número de octetos transportados en un datagrama, excluyendo al encabezado mismo. Así, un datagrama IPv6 puede contener 64K octetos de datos.

Un nuevo mecanismo en el IPv6 soporta reservación de recursos y permite a un ruteador asociar cada datagrama con una asignación de recursos dados. La abstracción subyacente, un *flujo*, consiste en una trayectoria a través de una red de redes a lo largo de la cual ruteadores intermedios garantizan una calidad de servicio específica. Por ejemplo, dos aplicaciones que necesitan enviar video pueden establecer un flujo en el que el retardo y el ancho de banda estén garantizados. Como alternativa, un proveedor de red puede requerir una suscripción para especificar la calidad de servicio deseado y, luego, utilizar un flujo para limitar el tráfico a una computadora específica o al envío de una aplicación específica. Observe que un flujo puede también utilizarse dentro de una organización determinada para administrar recursos de red y asegurar que todas las aplicaciones puedan compartir recursos de manera justa.

El campo *FLOW LABEL* en el encabezado base contiene información que los ruteadores utilizan para asociar un datagrama con una prioridad y un flujo específicos. El campo está subdividido en dos subcampos como se muestra en la figura 29.3.

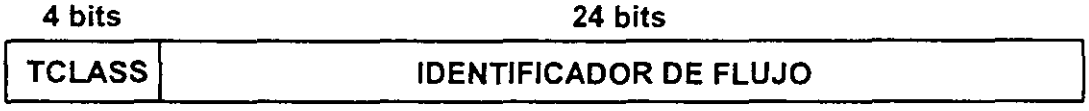



**Figura 29.3** Los dos subcampos de una etiqueta de flujo. Cada datagrama IPv6 transporta una etiqueta de flujo, la cual puede usarse para asociar el datagrama con una calidad específica de servicio.

Dentro de la etiqueta de flujo, el campo *TCLASS* de 4 bits especifica la clase de tráfico para el datagrama. Los valores del *0* al *7* se emplean para especificar la sensibilidad al tiempo del tráfico controlado por flujo; los valores del *8* al *15* se utilizan para especificar una prioridad para tráfico que no es de flujo. El campo de 24 bits restantes contiene el campo *FLOW IDENTIFIER (IDENTIFICADOR DE FLUJO)*. La fuente selecciona un identificador de flujo cuando establece el flujo (por ejemplo, en forma aleatoria). No hay conflicto potencial entre las computadoras debido a que un ruteador utilice la combinación de direcciones fuente de datagramas e identificadores de flujo cuando asocia un datagrama con un flujo específico. En resumen:

> *Cada datagrama IPv6 comienza con un encabezado base de 40 octetos que incluye campos para las direcciones de fuente y destino, el límite máximo de saltos, la etiqueta de flujo y el tipo del próximo encabezado. Así, un datagrama IPv6 debe contener cuando menos 40 octetos además de los datos.*

## 29.9 Encabezados de extensión del IPv6

El paradigma de un encabezado base fijo seguido por un conjunto de encabezados de extensión opcionales se eligió como un compromiso entre la generalidad y la eficiencia. Para ser totalmente general, el IPv6 necesita incluir mecanismos para soportar funciones como la fragmentación, el ruteo de fuente y la autenticación. Sin embargo, elegir la asignación de campos fijos en el encabezado de datagrama para todos los mecanismos es ineficiente pues la mayor parte de los datagramas no utiliza todos los mecanismos. El gran tamaño de las direcciones IPv6 aumenta la ineficiencia. Por ejemplo, cuando se envía un datagrama a través de una red de área local, un encabezado que contenga campos de dirección vacíos puede ocupar una fracción sustancial de cada trama. Algo muy importante, los diseñadores asumen que no se puede predecir qué recursos serán necesarios.

El paradigma de encabezado de extensión IPv6 funciona en forma similar a las opciones del IPv4 —un emisor puede elegir qué encabezados de extensión incluir en un datagrama determinado y cuáles omitir. Así, los encabezados de extensión proporcionan una flexibilidad máxima. Podemos resumir lo siguiente:

> *Los encabezados de extensión IPv6 son similares a las opciones IPv4. Cada datagrama incluye encabezados de extensión sólo para los recursos que el datagrama utilice.*

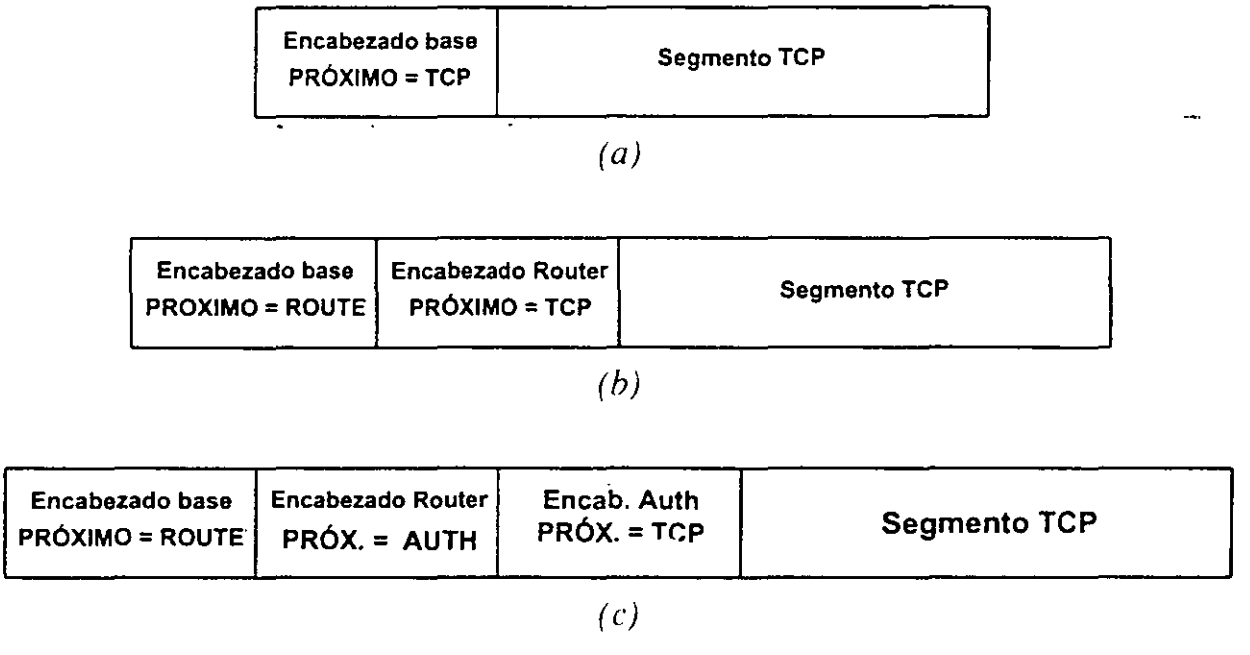



**Figura 29.4** Tres datagramas con (a) sólo un encabezado base, (b) un encabezado base y una extensión y (c) un encabezado base más dos extensiones. El campo *NEXT HEADER* (*PRÓXIMO ENCABEZADO*) en cada encabezado especifica el tipo de encabezado siguiente.

## 29.10 Análisis de un datagrama IPv6

Cada encabezado de base y extensiones contiene un campo *NEXT HEADER* (*PRÓXIMO ENCABEZADO*). El software en los ruteadores intermedios y en el destino final que necesitan procesar el datagrama deben utilizar el valor en el campo *NEXT HEADER* de cada encabezado para analizar el datagrama. Extraer toda la información del encabezado de un datagrama IPv6 requiere de una búsqueda secuencial a través de los encabezados. Por ejemplo, la figura 29.4 muestra el campo *NEXT HEADER* de 3 datagramas que contienen cero, uno y dos encabezados de extensión.

Por supuesto, analizar un datagrama IPv6 que sólo tiene un encabezado base y datos es tan eficaz como analizar un datagrama IPv4. Además, veremos que los ruteadores intermedios con frecuencia necesitan procesar todos los encabezados de extensión.

## 29.11 Fragmentación y reensamblaje del IPv6

Como el IPv4, el IPv6 prepara el destino final para realizar el reensamblaje de datagramas. Sin embargo, los diseñadores tomaron una decisión poco usual respecto a la fragmentación. Recordemos que el IPv4 requiere un ruteador intermedio para fragmentar cualquier datagrama que sea demasiado largo para la MTU de la red en la que viaja. En el IPv6, la fragmentación está restringida a la fuente original. Antes de enviar tráfico de información, una fuente debe realizar una técnica de *Path MTU Discovery* (*descubrir la MTU de la ruta*) para identificar la MTU (Maximum Transfer Unit) mínima a lo largo de la trayectoria hasta el destino. Antes de enviar un datagrama, la fuente fragmenta el datagrama de manera que cada fragmento sea menor que el Path MTU. Así, la fragmentación es de extremo a extremo; no son necesarias fragmentaciones adicionales en ruteadores intermedios.

El encabezado base IPv6 no contiene campos análogos a los campos utilizados para la fragmentación en un encabezado IPv4. Por el contrario, cuando la fragmentación es necesaria, la fuente inserta un pequeño encabezado de extensión luego del encabezado base en cada fragmento. La figura 29.5 muestra el contenido de un *encabezado de extensión de fragmento*.

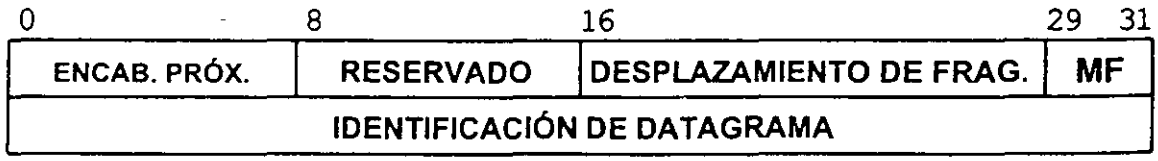



**Figura 29.5** Formato de un encabezado de extensión de fragmento.

El IPv6 conserva mucho de la fragmentación del IPv4. Cada fragmento debe ser un múltiplo de *8* octetos, un bit en el campo *MF* marca el último fragmento como el bit del IPv4 *MORE FRAG-*

*MENTS*, y el campo *DATAGRAM IDENTIFICATION* (*IDENTIFICACIÓN DE DATAGRAMA*) transporta una ID única que el receptor utiliza para el grupo de fragmentos.[3]

## 29.12 Consecuencia de la fragmentación de extremo a extremo

La motivación para utilizar la fragmentación de extremo a extremo radica en su capacidad para reducir la sobrecarga en los ruteadores y permitir que cada ruteador maneje más datagramas por unidad de tiempo. De hecho, la sobrecarga de CPU requerida por la fragmentación IPv4 puede ser significativa —en un ruteador convencional, la CPU puede alcanzar el 100% de su utilización si el ruteador fragmenta muchos o todos los datagramas que recibe. Sin embargo, la fragmentación de extremo a extremo tiene una consecuencia importante: cambia un supuesto fundamental respecto a Internet.

Para entender la consecuencia de la fragmentación de extremo a extremo, recordemos que el IPv4 está diseñado para permitir a los ruteadores cambiar en cualquier momento. Por ejemplo, si una red o un ruteador falla, el tráfico puede ser redireccionado hacia diferentes trayectorias. La mayor ventaja de este sistema es su flexibilidad —el tráfico puede rutearse hacia una trayectoria alternativa sin interrumpir el servicio y sin informar a la fuente o al destino. En el IPv6, sin embargo, los ruteadores no pueden cambiarse tan fácilmente pues un cambio en una ruta puede cambiar el Path MTU. Si el Path MTU a lo largo de una nueva ruta es menor que el Path MTU a lo largo de la ruta original, un ruteador intermedio debe fragmentar el datagrama original o la fuente original debe ser informada. El problema se puede resumir de la siguiente forma:

> *Un protocolo de red de redes que utilice la fragmentación de extremo a extremo requiere que el emisor descubra el Path MTU para cada destino y que fragmente cualquier datagrama que salga si es mayor que el Path MTU. La fragmentación de extremo a extremo no se adapta al cambio de rutas.*

Para resolver el problema de los cambios de ruta que afectan el Path MTU, el IPv6 permite a los ruteadores intermedios hacer un túnel de IPv6 a través del IPv6. Cuando un ruteador intermedio necesita fragmentar un datagrama, el ruteador no inserta un encabezado de extensión de fragmento ni cambia los campos en el encabezado base. En lugar de ello, el ruteador intermedio crea un datagrama completamente nuevo que encapsula el datagrama original como dato. El ruteador divide el nuevo datagrama en fragmentos reproduciendo el encabezado base e insertando un encabezado de extensión de fragmento en cada uno. Finalmente, el ruteador envía cada fragmento hacia el destino final. En el destino final, el datagrama original puede formarse recolectando los fragmentos entrantes en un datagrama y luego extrayendo la porción de datos. La figura 29.6 ilustra la encapsulación.

---

[3] El IPv6 expande el campo *IDENTIFICATION* a 32 bits para adaptarse a las redes de alta velocidad.

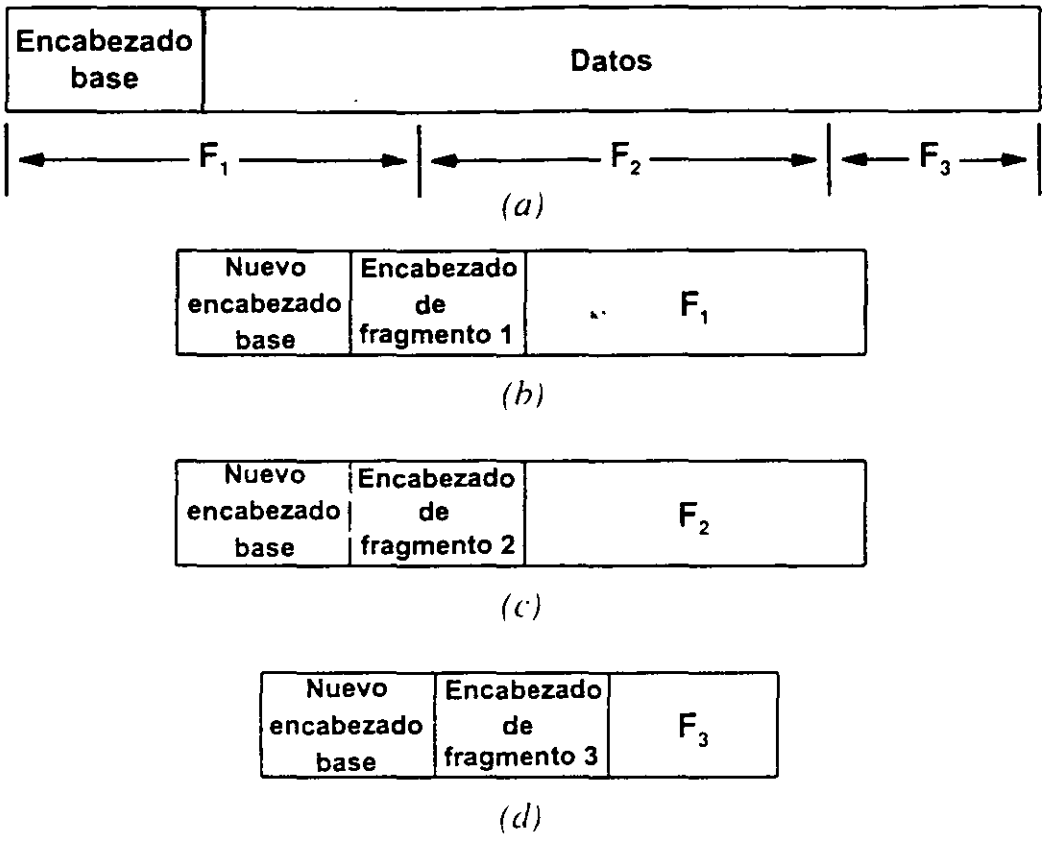



**Figura 29.6** (a) Un datagrama IPv6, y de (b) a (d) los tres fragmentos resultantes cuando un ruteador encapsula y fragmenta el datagrama. En el destino se reensamblará el datagrama original incluyendo el encabezado.

## 29.13 Ruteamiento de origen del IPv6

El IPv6 conserva la capacidad de un emisor para especificar una ruta fuente. A diferencia del IPv4, en el que el ruteo de fuente se proporciona mediante opciones, el IPv6 utiliza un encabezado de extensión separado. Como se muestra en la figura 29.7, los campos de encabezado de ruteo corresponden a los campos de una opción de ruteo de fuente del IPv4. El encabezado contiene una lista de direcciones que especifica ruteadores intermedios a través de los cuales debe pasar el datagrama. El campo *NUM ADDRS* (*NÚMERO DE DIRECCIONES*) especifica el número total de direcciones en la lista y el campo *NEXT ADDRESS* (*DIRECCIÓN PRÓXIMA*) la dirección siguiente hacia la que se enviará el datagrama.

## 29.14 Opciones del IPv6

Podría parecer que los encabezados de extensión IPv6 reemplazan por completo a las opciones IPv4. Sin embargo, los diseñadores propusieron 2 encabezados de extensión adicionales para adaptarse a cualquier tipo de información no incluida en otros encabezados de extensión. Los encabeza-

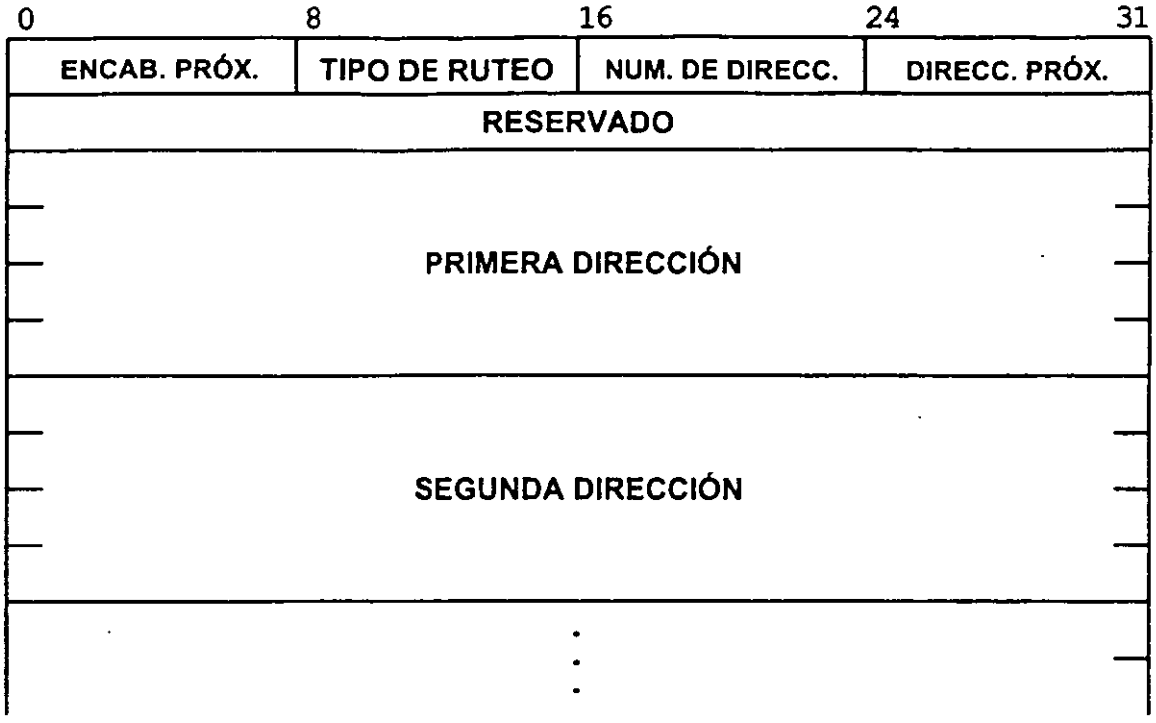



**Figura 29.7** Formato de un encabezado de ruteo IPv6. Los campos corresponden a los de una opción de ruta de fuente de IPv4.

dos adicionales consisten en un *Hop By Hop Extension Header* (*extensión de cabeza salto por salto*) y un *End To End Extension Header* (*extensión de cabeza extremo a extremo*). Como lo indican los nombres, los dos encabezados de opción separan el conjunto de opciones que serán examinados en cada salto por el conjunto que será interpretado en el destino.

Aun cuando cada uno de los 2 encabezados de opción tiene un código de tipo único, ambos encabezados utilizan el formato que se ilustra en la figura 29.8.

Como sucede normalmente, el campo *NEXT HEADER* proporciona el tipo de encabezado que sigue. Dado que un encabezado de opción no tiene un tamaño fijo, el campo con el nombre *HEADER LEN* (*LONGITUD DE ENCABEZADO)* especifica la longitud total del encabezado. El

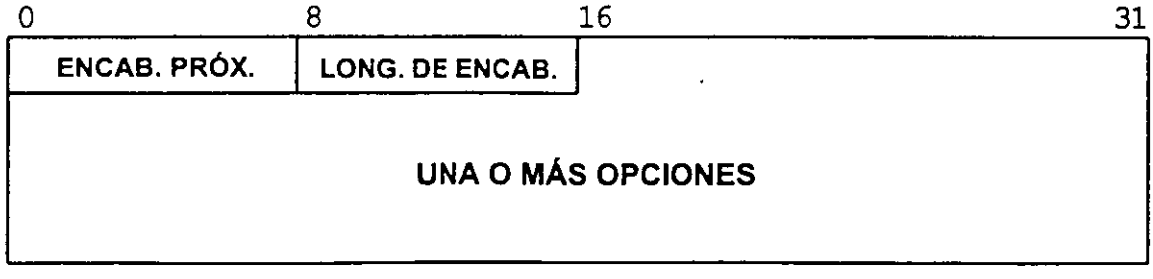



**Figura 29.8** Formato del encabezado de una extensión opcional IPv6. Los encabezados de las opciones *salto por salto* y *extremo a extremo* utilizan el mismo formato; el campo *NEXT HEADER* del encabezado anterior distingue entre los dos tipos.

área con el nombre *ONE OR MORE OPTIONS* (*UNA O MÁS OPCIONES*) representa una secuencia de opciones individuales. La figura 29.9 ilustra cómo está codificada cada opción individual con un tipo, longitud y valor;[4] las opciones no están alineadas ni tienen rellenos.

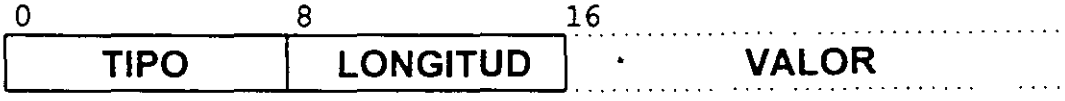



**Figura 29.9** Codificación de una opción individual en el encabezado de la extensión opcional IPv6. Cada opción consiste en un tipo de un octeto y una longitud de un octeto seguidos por cero o más octetos de datos para la opción.

Como se muestra en la figura, las opciones IPv6 tienen la misma forma que las opciones IPv4. Cada opción comienza con un campo de un octeto *TYPE* (*TIPO*) seguido de un campo *LENGTH* (*LONGITUD*) de un octeto. Si la opción requiere de datos adicionales, los octetos que comprenden *VALUE* (*VALOR*) se siguen de LENGTH.

Los dos bits de orden superior de cada opción de campo *TYPE* especifican cómo deberán disponer un anfitrión o un ruteador del datagrama si no comprende las opciones:

| Bits en tipo | Significado |
|---|---|
| 00 | Saltar esta opción |
| 01 | Desechar el datagrama; no enviar mensaje ICMP |
| 10 | Desechar datagrama; enviar mensaje ICMP a la fuente |
| 11 | Desechar datagrama; no enviar ICMP para multidifusión |

## 29.15 Tamaño del espacio de dirección del IPv6

En el IPv6, cada dirección ocupa 16 octetos, 4 veces el tamaño de una dirección IPv4. El amplio espacio de direcciones garantiza que el IPv6 puede tolerar cualquier esquema de asignación de direcciones razonable. De hecho, si los diseñadores deciden cambiar el esquema de direccionamiento más tarde, el espacio de direcciones es lo suficientemente extenso como para adaptarse a una reasignación.

Es difícil comprender el tamaño del espacio de direcciones del IPv6. Una forma de entenderlo es relacionando la magnitud con el tamaño de la población: el espacio de direcciones es tan grande que cada persona en el planeta puede tener direcciones suficientes como para poseer una red de redes tan grande como la Internet actual. Otra forma de entender el tamaño es relacionarlo con el agotamiento de direcciones. Por ejemplo, consideremos qué se necesitaría para asignar todas las posibles direcciones. Un entero de 16 octetos puede manejar $2^{128}$ valores. Así, el espacio de di-

[4] En la jerga, una codificación de tipo, longitud y valor se conoce en ocasiones como *codificación TLV*

recciones es mayor que $3.4 \times 10^{38}$. Si las direcciones se asignaran a razón de un millón de direcciones por milisegundo, tomaría alrededor de 20 años asignar todas las direcciones posibles.

## 29.16 Notación hexadecimal con dos puntos del IPv6

Aun cuando resuelve el problema de tener una capacidad insuficiente, el gran tamaño de direcciones plantea un problema nuevo: los usuarios que manejan redes de redes deben leer, introducir y manipular estas direcciones. Obviamente, la notación binaria no es práctica. Sin embargo, la notación decimal con puntos utilizada por el IPv4 tampoco hace las direcciones lo suficientemente compactas. Para entender por qué, consideremos el ejemplo de un número de 128 bits expresado en notación decimal con puntos:

```
104.230.140.100.255.255.255.255.0.0.17.128.150.10.255.255
```

Para ayudar a hacer la dirección ligeramente más compacta y fácil de introducir, los diseñadores del IPv6 proponen utilizar una *notación hexadecimal con dos puntos* (abreviado *colon hex*) en la cual el valor de cada cantidad de 16 bits se representa en forma hexadecimal separado por dos puntos. Por ejemplo, cuando el valor mostrado arriba en notación decimal se traduce a la notación hexadecimal con dos puntos e impresa, utilizando el mismo espaciado, se convierte en:

```
68E6:8C64:FFFF:FFFF:0:1180:96A:FFFF
```

La notación hexadecimal con dos puntos tiene la ventaja obvia de requerir menos dígitos y menos caracteres separadores que la notación decimal con puntos. Además, la notación hexadecimal con dos puntos incluye dos técnicas que la hacen muy útil. En la primera, la notación hexadecimal con dos puntos permite la *compresión 0* mediante la cual una cadena de ceros repetidos se reemplaza por un par de dos puntos. Por ejemplo, la dirección:

```
FF05:0:0:0:0:0:0:B3
```

puede escribirse:

```
FF05::B3
```

Para asegurar que la compresión cero produce una interpretación sin ambigüedades, la propuesta especifica que puede aplicarse sólo una en cualquier dirección. La compresión cero es especialmente útil cuando se emplea el esquema de asignación de direcciones propuesto ya que muchas direcciones contendrán cadenas contiguas de ceros. En segundo lugar, la notación hexadecimal con dos puntos incorpora sufijos decimales con punto; como veremos, esta combinación tiene el propósito de utilizarse durante la transición del IPv4 al IPv6. Por ejemplo, la siguiente cadena es una notación hexadecimal con dos puntos válida:

```
0:0:0:0:0:0:128.10.2.1
```

Obsérvese que, aun cuando los números están separados por dos puntos, cada uno especifica el valor de una cantidad de 16 bits, los números en la porción decimal con puntos especifican el valor de un octeto. Por supuesto, la compresión cero puede utilizarse con el número de arriba para producir una cadena hexadecimal con dos puntos equivalente que se vería muy similar a una dirección IPv4:

```
::128.10.2.1
```

## 29.17 Tres tipos básicos de dirección IPv6

Como el IPv4, el IPv6 asocia una dirección con una conexión de red específica, no con una computadora específica. Así, la asignación de direcciones es similar para el IPv4: un ruteador IPv6 tiene dos o más direcciones, y un anfitrión IPv6, con una conexión de red, necesita sólo una dirección. El IPv6 también conserva (y extiende) la jerarquía de direcciones del IPv4 en la que una red física es asignada a un prefijo. Sin embargo, para hacer la asignación de direcciones y la modificación más fácil, el IPv6 permite que varios prefijos sean asignados a una red dada y que una computadora tenga varias direcciones simultáneas asignadas hacia una interfaz determinada.

Además de permitir varias direcciones simultáneas por conexión de red, el IPv6 expande y, en algunos casos, unifica las direcciones especiales del IPv4. En general, una dirección de destino en un datagrama cae dentro de una de tres categorías:

| | |
|---|---|
| Unidifusión | La dirección de destino especifica una sola computadora (anfitrión o ruteador); el datagrama deberá rutearse hacia el destino a lo largo de la trayectoria más corta. |
| Grupo | El destino es un conjunto de computadoras en el que todas comparten un solo prefijo de dirección (por ejemplo, si están conectadas a la misma red física); el datagrama deberá rutearse hacia el grupo a través de la trayectoria más corta y, después, entregarse exactamente a un miembro del grupo (por ejemplo, el miembro más cercano). |
| Multidifusión | El destino es un conjunto de computadoras, posiblemente en múltiples localidades. Una copia del datagrama deberá entregarse a cada miembro del grupo que emplee hardware de multidifusión o de difusión si están disponibles. |

## 29.18 Dualidad de difusión y multidifusión

El IPv6 no emplea el término *difusión* o *difusión dirigida* para referirse a la entrega a todas las computadoras en una red física o a una subred IP lógica. En cambio, utiliza el término *multidifusión*, y trata a la difusión como una forma especial de multidifusión. La elección puede parecer ex-

traña para cualquiera que conozca el hardware de red ya que la mayor parte de las tecnologías de hardware soporta la difusión así como la multidifusión. De hecho, un ingeniero de hardware es probable que vea a la multidifusión como una forma restringida de difusión —el hardware envía un paquete de multidifusión hacia todas las computadoras en la red exactamente como un paquete de difusión, y el hardware de interfaz en cada computadora filtra todos los paquetes de multidifusión excepto los que el software ha definido para que los acepte el hardware de interfaz.

En teoría, la elección entre la multidifusión y una forma limitada de difusión es irrelevante pues se puede simular una con la otra. Esto es, la difusión y la multidifusión son dos formas diferentes que proporcionan la misma funcionalidad. Para entender por qué, consideremos cómo simular una con la otra. Si la difusión está disponible, un paquete puede entregarse a un grupo enviándolo a todas las máquinas y haciendo que el software en cada computadora decida si acepta o descarta el paquete entrante. Si la multidifusión está disponible, un paquete puede ser entregado a todas las máquinas haciendo que todas las máquinas escuchen el mismo grupo de multidifusión similar de *todos los anfitriones* (tratado en el capitulo 17).

## 29.19 Una elección de ingeniería y difusión simulada

Saber que la difusión y la multidifusión son teóricamente equivalentes no ayuda a la elección entre éstas. Para ver por qué los diseñadores del IPv6 eligieron la multidifusión como la abstracción central en lugar de la difusión, consideremos las abstracciones en lugar de observar el hardware subyacente. Una aplicación necesita comunicarse con otra aplicación o con otro grupo de aplicaciones. La comunicación directa se maneja mejor vía unidifusión; la comunicación en grupo se maneja mejor por medio de la multidifusión o la difusión. Para proporcionar la mayor flexibilidad, los miembros de un grupo no deben determinar las conexiones de red, ya que puede haber miembros que residan en localidades arbitrarias. Utilizar la difusión para la comunicación de todo el grupo no conduce a manejar una red de redes tan extensa como la red global de Internet.

No es sorprendente, pues, que los diseñadores predefinieran las direcciones de multidifusión que corresponden a las redes del IPv4 y a las direcciones de difusión de subred. Así, además de sus propias direcciones de unidifusión, cada anfitrión es requerido para aceptar paquetes direccionados hacia el grupo de multidifusión *todos los nodos* y hacia el grupo de multidifusión *todos los anfitriones* para su entorno local; también, existe la dirección *todos los ruteadores*.

## 29.20 Asignación propuesta de espacio de dirección IPv6

La cuestión sobre cómo dividir el espacio de direcciones ha generado muchas discusiones. Hay dos temas centrales: cómo administrar la asignación de direcciones y cómo transformar una dirección en una ruta. El primer tema se enfoca en el problema práctico de construir una jerarquía de autoridad. A diferencia de la Internet actual, la cual utiliza una jerarquía de dos niveles de prefijos de red (asignados por la autoridad de Internet) y sufijos de anfitrión (asignados por la organización), el gran espacio de direcciones en el IPv6 permite una jerarquía de multiniveles o jerarquías múltiples. El segundo tema se enfoca en la eficiencia computacional. Independientemente de la jerarquía de

autoridad que asigne direcciones, un ruteador debe examinar cada datagrama y elegir una trayectoria hacia el destino. Para mantener bajo el costo de los ruteadores de alta velocidad, el tiempo de procesamiento requerido para elegir una trayectoria debe mantenerse bajo.

Como se muestra en la figura 29.10, los diseñadores del IPv6 proponen asignar clases de direcciones en forma similar al esquema utilizado por el IPv4. Aun cuando los ocho primeros bits de una dirección son suficientes para especificar su tipo, el espacio de direcciones no se divide en secciones de igual tamaño.

| Prefijo binario | Tipo de dirección | Parte del espacio de dirección |
|---|---|---|
| 0000 0000 | Reservado (compatible con IPv4) | 1 /256 |
| 0000 0001 | Reservado | 1 /256 |
| 0000 001 | Direcciones NSAP | 1/128 |
| 0000 010 | Direcciones IPX | 1/128 |
| 0000 011 | Reservado | 1/128 |
| 0000 100 | Reservado | 1/128 |
| 0000 101 | Reservado | 1/128 |
| 0000 110 | Reservado | 1/128 |
| 0000 111 | Reservado | 1 /28 |
| 0001 | Reservado | 1/16 |
| 001 | Reservado | 1/8 |
| 010 | Unidifusión proveedor asignado | 1/8 |
| 011 | Reservado | 1/8 |
| 100 | Reservado (geográfico) | 1/8 |
| 101 | Reservado | 1/8 |
| 110 | Reservado | 1/8 |
| 1110 | Reservado | 1/16 |
| 1111 0 | Reservado | 1/32 |
| 1111 10 | Reservado | 1/64 |
| 1111 110 | Reservado | 1/128 |
| 1111 1110 | Disponible para uso local | 1/ 256 |
| 1111 1111 | Utilizado para multidifusión | 1/256 |

**Figura 29.10** División propuesta de las direcciones IPv6 en tres tipos, los cuales son análogos a las clases IPv4. Como en el IPv4, el prefijo de una dirección determina su tipo de dirección.

## 29.21 Codificación y transición de la dirección IPv4

Observe de la figura 29.10 que alrededor del *72%* del espacio de direcciones ha sido reservado para usos futuros, no incluyendo la sección reservada para direcciones geográficas. Aun cuando el

prefijo *0000 0000* tiene el nombre *reservado* en la figura, los diseñadores planean usar una pequeña fracción de direcciones en esta sección para codificar direcciones IPv4. En particular, cualquier dirección que comience con 80 bits puestos a cero, seguidos por 16 bits puestos a 1 o 16 bits puestos todos a cero, contiene una dirección IPv4 en los 32 bits de orden inferior. La codificación será necesaria durante la transición del IPv4 al IPv6 por dos razones. En primer lugar, una computadora puede elegir actualizar su software de IPv4 como IPv6 antes de tener asignada una dirección IPv6 válida. En segundo lugar, una computadora que corra software IPv6 puede necesitar comunicarse con una computadora que corra sólo software IPv4.

Teniendo una forma de codificar una dirección IPv4 en una dirección IPv6 no se resuelve el problema de lograr que las dos versiones interoperen. Además de la codificación de direcciones, es necesaria la traducción. Para utilizar un traductor, una computadora IPv6 genera un datagrama que contiene la codificación IPv6 de la dirección de destino IPv4. La computadora IPv6 envía el datagrama hacia un traductor, el cual utiliza IPv4 para comunicarse con el destino. Cuando el traductor recibe una réplica desde el destino, traduce el datagrama IPv4 a IPv6 y lo envía de regreso a la fuente IPv6.

Parecería como si el protocolo de traducción de direcciones fallara debido a que las capas superiores de los protocolos verifican la integridad de las direcciones. En particular, el TCP y el UDP utilizan un *pseudo encabezado* en su cálculo para la suma de verificación. El pseudo encabezado incluye la dirección del protocolo de la fuente y el destino, cambiar estas direcciones puede afectar el cálculo. Sin embargo los diseñadores planearon cuidadosamente que el TCP o el UDP en una máquina IPv4 se pudieran comunicar con el correspondiente protocolo de transporte en una máquina IPv6. Para evitar errores en la suma de verificación, la codificación IPv6 de una dirección IPv4 ha sido elegida de manera que el complemento a uno de los 16 bits de la suma de verificación para una dirección IPv4 y la codificación IPv6 de la dirección sean idénticos. El punto es el siguiente:

> *Además de seleccionar detalles técnicos de un nuevo protocolo de Internet, el IETF que trabaja en el IPng se ha enfocado en encontrar una forma de transición del protocolo actual al protocolo nuevo. En particular, la propuesta actual para el IPv6 permite codificar una dirección IPv4 en lugar de una dirección IPv6, de manera que la traducción de la dirección no cambie la suma de la verificación del pseudo encabezado.*

## 29.22 Proveedores, suscriptores y jerarquía de direcciones

Un ejemplo ayudará a entender cómo concibieron los diseñadores el uso de las direcciones IPv6. Consideremos la compañía *Network Access Provider* (*NAP*). Dicha compañía ofrece a sus clientes conectividad hacia Internet, a tales clientes los llamaremos *suscriptores*. Para permitir que cada proveedor asigne direcciones, la autoridad de Internet asigna a cada proveedor un identificador único. El proveedor puede entonces asignar a cada proveedor un identificador único y utilizar ambos identificadores cuando asigne un bloque de direcciones. El suscriptor puede asignar entonces un ID único para cada red física y a cada computadora en una red un ID de nodo único. La figura 29.11 ilustra la posible división de una dirección en subcampos.

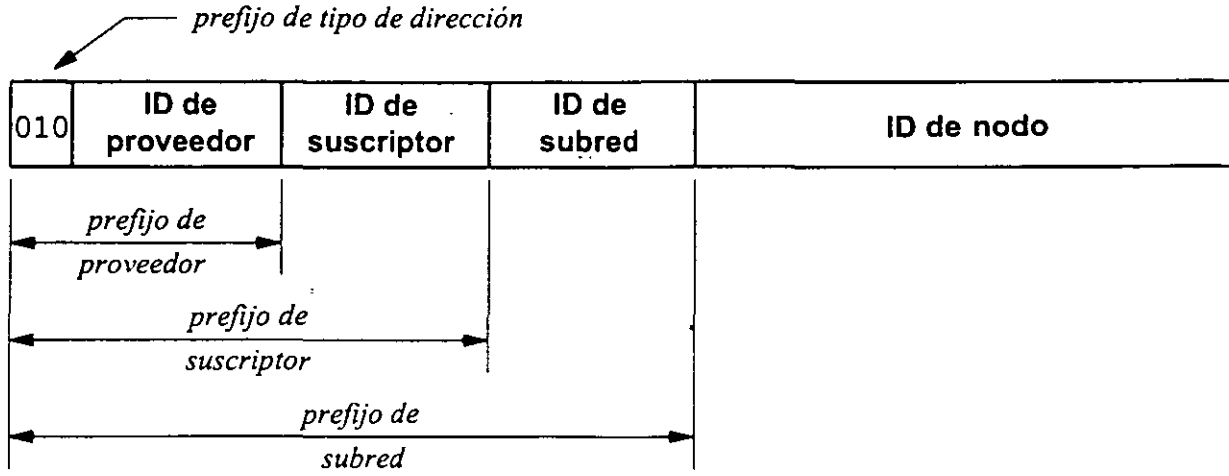



**Figura 29.11** Jerarquía de direcciones IPv6 para una dirección asignada por un proveedor de acceso a red. La autoridad de Internet asigna a cada proveedor una ID única, el proveedor asigna una ID única a cada suscriptor y el suscriptor asigna una única ID a cada subred en cada nodo.

Como se muestra en la figura 29.11, cada prefijo sucesivamente más largo tiene un nombre. La cadena inicial *010* identifica la dirección como el tipo de asignación del proveedor. Para cada dirección, el *prefijo de proveedor* incluye el tipo de dirección más el ID del proveedor. El *prefijo de suscriptor* cubre el prefijo del proveedor más el ID del suscriptor. Por último, el *prefijo de subred* incluye el prefijo de suscriptor más la información de subred.

Los campos en la figura 29.11 no están dibujados a escala. Por ejemplo, aun cuando los prefijos de direcciones parecen grandes en la figura, ocupan sólo tres de 128 bits. Los diseñadores recomiendan que el campo *ID del nodo* contenga por lo menos 48 bits para permitir que se utilice el direccionamiento de tipo 802 de IEEE. Así, será posible para un nodo IPv6 usar su dirección Ethernet como su ID de nodo.

## 29.23 Jerarquía adicional

Aunque el formato de direcciones mostrado arriba implica una jerarquía de cuatro niveles, una organización puede introducir niveles adicionales dividiendo el campo *Subnet ID* en varios campos. Por ejemplo, una organización puede elegir subdividir su subred en áreas y asignar subredes dentro de las áreas. Hacer esto es similar al esquema de direccionamiento de subred del IPv4, en el que la porción del anfitrión de una dirección es dividida en dos partes. El amplio espacio de direccionamiento del IPv6 permite la división en muchas partes.

## 29.24 Resumen

Ni la red global Internet ni los protocolos TCP/IP son estáticos. A través de su Engineering Task Force, la Internet Architecture Board se mantiene activa y realiza esfuerzos que hacen que la tecnología evolucione y mejore. Los procesos que conducen al cambio se manifiestan como un incremento en el tamaño y en la carga que obliga a mejorar los recursos para mantener el servicio, como aplicaciones nuevas que demandan más de la tecnología subyacente y como tecnologías nuevas que hacen posible proporcionar nuevos servicios.

Un esfuerzo para definir la próxima generación de protocolo de Internet (IPng) ha generado una gran polémica y varias propuestas. Ha surgido un acuerdo de IETF para adoptar una propuesta conocida como *Simple IP Plus* como estándar para el IPng. Debido que se deberá asignar el número de versión *6*, el protocolo propuesto se conoce a menudo como IPv6 para distinguirlo del protocolo actual, IPv4.

El IPv6 conserva muchos de los conceptos básicos del IPv4, pero cambia la mayor parte de los detalles. Como el IPv4, el IPv6 proporciona un servicio de entrega de datagramas sin conexión, con el mejor esfuerzo. Sin embargo el formato del datagrama IPv6 es completamente diferente del formato IPv4, y el IPv6 proporciona características nuevas como la autenticación, un mecanismo para flujos controlados de datagramas y soporte para seguridad.

El IPv6 revisa cada datagrama como una serie de encabezados seguidos por datos. Un datagrama siempre comienza con un encabezado base de 40 octetos, el cual contiene la dirección de fuente y destino y un identificador de flujo. El encabezado base puede estar seguido de ceros o por más encabezados de extensión, seguidos por datos. Los encabezados de extensión son opcionales —el IPv6 los utiliza para manejar gran parte de la información que el IPv4 codifica en opciones.

Una dirección IPv6 tiene una longitud de 128 bits, lo que hace que el espacio de dirección sea tan largo que cada persona en el planeta podría tener una red de redes tan extensa como la Internet actual. El IPv6 divide las direcciones en tipos en forma análoga a como el IPv4 las divide en clases. Un prefijo de la dirección determina la localización y la interpretación de los campos de dirección restantes. Muchas direcciones IPv6 serán asignadas por proveedores de servicio de red autorizados, dichas direcciones tienen campos que contienen un ID de proveedor, un ID de suscriptor, un ID de subred y un ID de nodo.

## PARA CONOCER MÁS

Han aparecido muchos RFC que contienen información relacionada con el IPng, incluyendo análisis sobre requerimientos, procedimientos y propósitos específicos. Bradner y Mankin (RFC 1550) hacen una invitación para propuestas y discusiones; muchos RFC responden. Por ejemplo, Britton y Tavs (RFC 1678) y Fleischman (RFC 1687) hacen comentarios sobre el IPng en redes corporativas amplias. Gross (RFC 1719) trata una dirección general, Partridge y Kastenholz (RFC 1726) analizan los criterios técnicos para elegir la tecnología IPng, y Brazdziunas (RFC 1680) comenta sobre el soporte IPng para ATM. Bellovin (RFC 1675) comenta la seguridad en el IPng.

Bradner y Mankin (RFC 1752) resumen propuestas y contienen las recomendaciones de los administradores de área de IETF para el IPng. Los lectores deben estar conscientes de que las pro-

puestas para el IPv6 no son ya un estándar probado a fondo y que más adelante algunos detalles probablemente cambien.

## EJERCICIOS

**29.1** El IPv6 propuesto no tiene suma de verificación del encabezado. ¿Cuáles son las ventajas y las desventajas de este método?

**29.2** ¿Cómo deberían ordenarse los encabezados de extensión para minimizar el tiempo de procesamiento?

**29.3** Aun cuando las direcciones del IPv6 son asignadas jerárquicamente, un ruteador no necesita analizar una dirección completamente para seleccionar una ruta. Construya un algoritmo y una estructura de datos para obtener un ruteo eficiente. Sugerencia: considere un enfoque de mayor semejanza.

**29.4** Demuestre que los 128 bits de direcciones son más de lo que se necesitaría y que 96 bits proporcionan capacidad suficiente.

**29.5** Suponga que su organización trata de adoptar el IPv6. Construya el esquema de direccionamiento que utilizaría la organización para asignar a cada anfitrión una dirección. ¿Seleccionaría una asignación jerárquica dentro de su organización? ¿Por qué sí o por qué no?

**29.6** ¿Cuál es la mayor ventaja de codificar una dirección Ethernet en una dirección IPv6? ¿Cuál es la mayor desventaja?

**29.7** Si usted tuviera que seleccionar tamaños para los campos ID de proveedores, suscriptores y subredes de una dirección IPv6, ¿de qué tamaño haría cada uno? ¿Por qué?

**29.8** Lea sobre los encabezados de autenticación y seguridad del IPv6. ¿Por qué se proponen dos encabezados?

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

Cursos Abiertos

# *Diplomado en Redes*

# *WAN*

Módulo IV

## TCP / IP, Protocolos de Internet

( CA 127 )

## Notas del Curso

**Octubre de 1998**


**Expositor:**

**Dr. James Kellegahn**



Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtemoc 06000 México, D.F APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

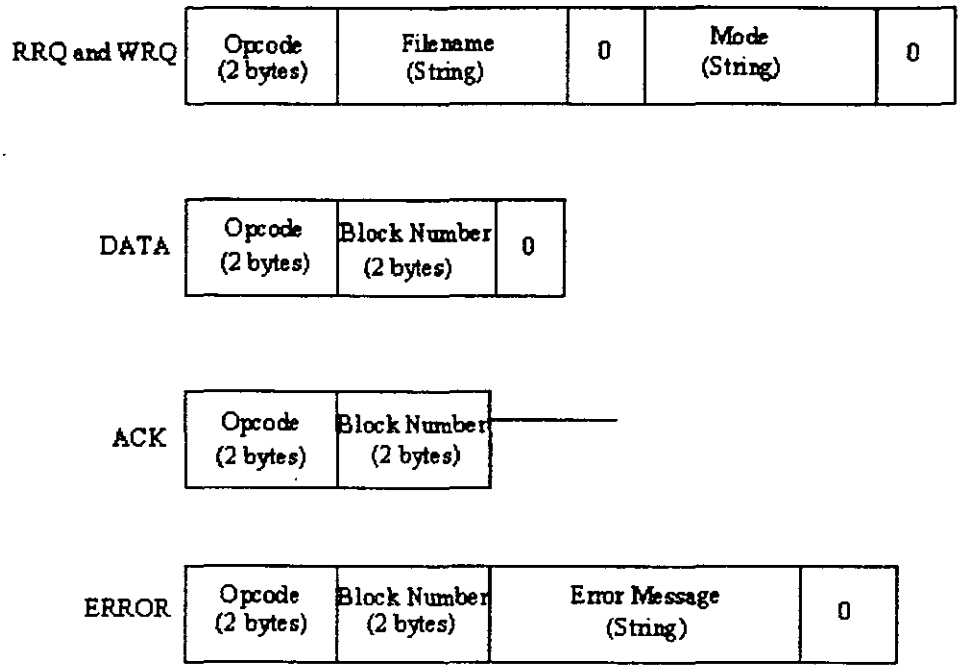



**Figure 6-14: TFTP Packet layouts**



TFTP packets

| | |
|---|---|
| NLST | Transfer a directory listing |
| NOOP | No operation |
| PASS | User password |
| PASV | Request a passive open |
| PORT | Port address |
| PWD | Display current directory |
| QUIT | Terminate the connection |
| REIN | Terminate and restart a connection |
| REST | Restart marker (restart transfer) |
| RETR | Transfer copy of file |
| RMD | Remove a directory |
| RNFR | Old pathname for rename command |
| RNTO | New pathname for rename command |
| SITE | Provides service specifics |
| SMNT | Mount a file system |
| STAT | Returns status |
| STOR | Accept and store data |
| STOU | Accept data and store under different name |
| STRU | File structure |
| SYST | Query to determine operating system |
| TYPE | Type of data |
| USER | User ID |

FTP also uses simple return codes to indicate transfer conditions. Each return code is a three-digit number, the first of which signifies a successful execution (the first digit is 1, 2, or 3) or a failure (the first digit is 4 or 5). The second and third digits specify the return code or error condition in more detail. The FTP return codes are shown in Table 6.4 and Table 6.5. The third-digit codes are not included here because there are many of them and they vary between implementations.

**Table 6.4. FTP reply code first digits.**

*interpreter*, or PI, and the *data transfer process*, or DTP. The PI transfers instructions between the two implementations using TCP command channel 21, and the DTP transfers data on TCP data channel 20. This is shown in Figure 6.7.

Figure 6.7. FTP channel connections.

FTP is similar to Telnet in that it uses a server program that runs continuously and a separate program that is executed on the client. On UNIX systems, these programs are named ftpd and ftp, respectively (similar to telnetd and telnet).

## FTP Commands

Before looking at how you can use FTP to transfer files, you should look at the commands behind the protocol itself. As with Telnet's commands, these are for the protocol's use only and should not be used by a user (although administrators sometimes use the FTP commands for debugging and diagnostic purposes).

FTP's internal protocol commands are four-character ASCII sequences terminated by a newline character. Some of the codes require parameters after them. One primary advantage to using ASCII characters for commands is that a user can observe the command flow and understand it easily. This helps considerably in the debugging process. Also, it enables a knowledgeable user to communicate directly with the FTP server component (ftpd).

FTP commands used by the protocol are summarized in Table 6.3. These commands provide for the connection process, password checking, and the actual file transfers. These are not to be confused with the commands available to a user.

**Table 6.3. FTP internal commands.**

| *Command* | *Description* |
|---|---|
| ABOR | Abort previous command |
| ACCT | User account ID |
| ALLO | Allocate storage for forthcoming operation |
| APPE | Append incoming data to an existing file |
| CDUP | Change to parent directory |
| CWD | Change working directory |
| DELE | Delete file |
| HELP | Retrieve information |
| LIST | Transfer list of directories |
| MKD | Make a directory |
| MODE | Set transfer mode |

command:

```
ftp 205.150.89.5
```

When FTP connects to the destination, you must be able to log into the system as a valid user (as you do when connecting through Telnet). Some systems enable an anonymous or guest login for FTP file transfers (usually using your login name as a password as a record of your access; see the section titled "Anonymous FTP Access"), but most require you to have regular access to the machine. The following extract shows the login process as a user provides a login and password for the remote machine:

```
ftp tpci_hpws4
Connected to tpci_hpws4.
220 tpci_hpws4 FTP server
Name (tpci_hpws4:tparker):
331 Password required for tparker.
Password:
230 User tparker logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
```

On large networks where a system such as Yellow Pages (YP) or Network Information Services (NIS) is used, FTP logins are usually permitted on most machines. If YP or NIS is not employed, you must be in the valid users file to obtain FTP access. As with Telnet, you can log into the remote with a different user ID from your local machine login. To transfer files, you must have the proper permissions on the remote, if file permissions are provided for by the operating system.

After logging into another machine using FTP, you are not actually on the remote machine. You are still logically on the client, so all instructions for file transfers and directory movement must be with respect to your local machine, not the remote one. Note that this is the opposite of Telnet (a distinction that causes considerable confusion among newcomers to FTP and Telnet).

Remember that all references to files and directories are relative to the machine that initiated the FTP session. If you are not careful, you can accidentally overwrite existing files.

The process followed by FTP when a connection is established is as follows:

1. **Login:** Verifies the user ID and password.
2. **Define directory:** Identifies the starting directory.
3. **Define file transfer mode:** Defines the type of transfer.
4. **Start data transfer:** Enables user commands.

| *First Digit* | *Description* |
|---|---|
| 1 | Action initiated. Expect another reply before sending a new command. |
| 2 | Action completed. Can send a new command. |
| 3 | Command accepted but on hold due to lack of information. |
| 4 | Command not accepted or completed. Temporary error condition exists. Command can be reissued |
| 5 | Command not accepted or completed. Reissuing the command will result in the same error (don't reissue). |

**Table 6.5. FTP reply code second digits.**

| *Second Digit* | *Description* |
|---|---|
| 0 | Syntax error or illegal command |
| 1 | Reply to request for information |
| 2 | Reply that refers to connection management |
| 3 | Reply for authentication command |
| 4 | Not used |
| 5 | Reply for status of server |

FTP enables file transfers in several formats, which are usually system-dependent. The majority of systems (including UNIX systems) have only two modes: text and binary. Some mainframe installations add support for EBCDIC, whereas many sites have a local type designed for fast transfers between local network machines (the local type might use 32- or 64-bit words).

Text transfers use ASCII characters separated by carriage-return and newline characters, whereas binary enables transfer of characters with no conversion or formatting. Binary mode is faster than text and also enables for the transfer of all ASCII values (necessary for nontext files). On most systems, FTP starts in text mode, although many system administrators now set FTP to binary mode as a default for their users' convenience. FTP cannot transfer file permissions, because these are not specified as part of the protocol.

Before transferring files with FTP, make sure you are using the correct transfer mode. Transferring a binary file as ASCII results in garbage! Check with your system administrator if you are unsure of the mode, or watch the messages FTP returns to see the mode used.

## FTP Connections

FTP is usually started with the name or address of the target machine. As with Telnet, the name must be resolvable into an IP address for the command to succeed. The target machine can also be specified from the FTP command line. For example, to connect to the IP address 205.150.89.5, you would issue this

```
Password:
230 User tparker logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pwd
257 "/u1/tparker" is current directory.
ftp> get mandelfile1.gif
remote: mandelfile1.gif local: mandelf1.gif
200 PORT command successful
150 Opening BINARY mode data connection for mandelfile1.gif
226 File transfer complete
1192354 bytes sent in 0.99 seconds
ftp> <Ctrl+d>
tpci_hpws1-2>
```

In this short sample, I transferred a file called mandelfile1.gif from a UNIX machine (the server) to the local machine (the client). You might have noticed that the filename was truncated automatically by the server to fit the DOS filesystem naming conventions. Also, note that I used binary mode (which was the system default). If the default had been ASCII mode, I would have just transferred over a megabyte of total garbage that couldn't be used for anything.

A debugging option is available from the command line by adding -d to the command. This displays the command channel instructions. Instructions from the client are shown with an arrow as the first character, whereas instructions from the server have three digits in front of them. A PORT in the command line indicates the address of the data channel on which the client is waiting for the server's reply. If no PORT is specified, channel 20 (the default value) is used. Unfortunately, the progress of data transfers cannot be followed in the debugging mode. A sample session with the debug option enabled is shown here:

```
tpci_hpws1-1> ftp -d
ftp> open tpci_hpws4
Connected to tpci_hpws4.
220 tpci_hpws4 FTP server Name (tpci_hpws4:tparker):
---> USER tparker
331 Password required for tparker.
Password:
---> PASS qwerty5
230 User tparker logged in.
```

5. **Stop data transfer:** Closes the connection.

The steps are performed in sequence for each connection. A user has several commands available to control FTP; the most frequently used commands are summarized in Table 6.6.

**Table 6.6. FTP user commands.**

| *FTP Command* | *Description* |
|---|---|
| ascii | Switch to ASCII transfer mode |
| binary | Switch to binary transfer mode |
| cd | Change directory on the server |
| close | Terminate the connection |
| del | Delete a file on the server |
| dir | Display the server directory |
| get | Fetch a file from the server |
| hash | Display a pound character for each block transmitted |
| help | Display help |
| lcd | Change directory on the client |
| mget | Fetch several files from the server |
| mput | Send several files to the server |
| open | Connect to a server |
| put | Send a file to the server |
| pwd | Display the current server directory |
| quote | Supply an FTP command directly |
| quit | Terminate the FTP session |

Using FTP is similar to Telnet, except that all movements of files are relative to the client. Therefore, putting a file is moving it from the client to the server, whereas getting a file is the reverse. A sample FTP session follows:

```
tpci_hpws1-1> ftp tpci_hpws4

Connected to tpci_hpws4.

220 tpci_hpws4 FTP server (Version 1.7.109.2 Tue Jul 28 23:32:34 GMT 1992) ready.

Name (tpci_hpws4:tparker):

331 Password required for tparker.
```

| *Code* | *Value* | *Description* |
|---|---|---|
| Abort Output (AO) | 245 | Runs process to completion but does not send the output |
| Are you there (AYT) | 246 | Queries the other end to ensure that an application is functioning |
| Break (BRK) | 243 | Sends a break instruction |
| Data Mark | 242 | Data portion of a Sync |
| Do | 253 | Asks for the other end to perform or an acknowledgment that the other end is to perform |
| Don't | 254 | Demands that the other end stop performing or confirms that the other end is no longer performing |
| Erase Character (EC) | 247 | Erases a character in the output stream |
| Erase Line (EL) | 248 | Erases a line in the output stream |
| Go Ahead (GA) | 249 | Indicates permission to proceed when using half-duplex (no echo) communications |
| Interpret as Command (IAC) | 255 | Interprets the following as a command |
| Interrupt Process (IP) | 244 | Interrupts, suspends, aborts, or terminates the process |
| NOP | 241 | No operation |
| SB | 250 | Subnegotiation of an option |
| SE | 240 | End of the subnegotiation |
| Will | 251 | Instructs the other end to begin performing or confirms that this end is now performing |
| Won't | 252 | Refuses to perform or rejects the other end performing |

Telnet commands are sent in a formal package called a *command*, as shown in Figure 6.5. Typically the commands contain two or three bytes: the Interpret as Command (IAC) instruction, the command code being sent, and any optional parameter to the command. The options supported by Telnet are shown in Table 6.2.

Figure 6.5. The Telnet command structure.

**Table 6.2. Supported Telnet option codes.**

| *Code* | *Description* |
|---|---|
| 0 | Binary transmission |
| 1 | Echo |
| 2 | Reconnection |
| 3 | Suppress Go Ahead (GA) |

```
Connected to tpci_hpws4.
Escape character is '^]'.
SENT do SUPPRESS GO AHEAD
SENT will TERMINAL TYPE (don't reply)
SEND will NAWS (don't reply)
RCVD do 36 (reply)
sent won't 36 (don't reply)
RECD do TERMINAL TYPE (don't reply)
RCVD will SUPPRESS GO AHEAD (don't reply)
RCVD do NAWS (don't reply)
Sent suboption NAWS 0 80 (80) 0 37 (37)
Received suboption Terminal type - request to send.
RCVD will ECHO (reply)
SEND do ECHO (reply)
RCVD do ECHO (reply)
SENT won't ECHO (don't reply)
HP-UX tpci_hpws4 A.09.01 A 9000/720 (ttys2)
login:
```

**Note**

The Telnet commands are used by the protocol, not by users (although you can issue them during a Telnet session, but this is usually used only for diagnostic purposes). There are no inherent Telnet user commands, other than the command mode toggle, because Telnet's role is to connect you to a remote system and let you use it directly.

A partial set of Telnet command codes is shown in Table 6.1. Additional codes are used to represent printer functions such as horizontal and vertical tabs and form feeds, but these have been left off the table for brevity's sake. Part of the Telnet command code set includes six terminal functions (IP, AO, AYT, EC, EL, and GA) that are common across most terminal definitions, so they are formally defined in the Telnet standard.

**Table 6.1. Telnet command codes.**

echoing.

---

**Note**

The use of ASCII characters and small tables of commands and options make it relatively easy to follow Telnet communications.

---

### TN3270

Many mainframes use EBCDIC, whereas most smaller machines rely on ASCII. This can cause a problem when trying to Telnet from EBCDIC-based machines to ASCII-based machines and vice-versa, because the codes being transferred are not accurate. To correct this, a Telnet application called TN3270 was developed, which provides translation between the two formats.

When TN3270 is used to connect between two machines, Telnet itself establishes the initial connection, and then one end sets itself up for translation. If an ASCII machine is calling an EBCDIC machine, the translation between the two formats is conducted at the EBCDIC (server) end unless there is a gateway between them, in which case the gateway can perform the translation.

Many TCP/IP application suites that include a Telnet program also include a TN3270 program. For example, Figure 6.6 shows a TN3270 window from the NetManage ChameleonNFS suite in the process of connecting to a mainframe EBCDIC-based machine. The mainframe's IP address is used to initiate the connection.

Figure 6.6. TN3270 provides conversion between ASCII and EBCDIC.

### File Transfer Protocol (FTP)

*File Transfer Protocol*, usually called FTP, is a utility for managing files across machines without having to establish a remote session with Telnet. FTP enables you to transfer files back and forth, manage directories, and access electronic mail. FTP is not designed to enable access to another machine to execute programs, but it is the best utility for file transfers.

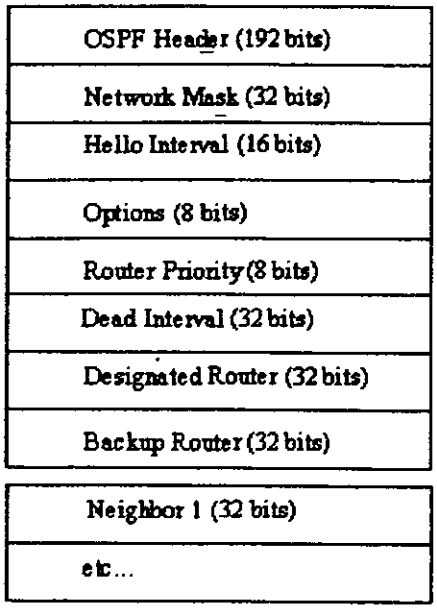
FTP uses two TCP channels. TCP port 20 is the data channel, and port 21 is the command channel. FTP is different from most other TCP/IP application programs in that it does use two channels, enabling simultaneous transfer of FTP commands and data. It also differs in one other important aspect: FTP conducts all file transfers in the foreground, instead of the background. In other words, FTP does not use spoolers or queues, so you are watching the transfer process in real time. By using TCP, FTP eliminates the need to worry about reliability or connection management, because FTP can rely on TCP to perform these functions properly.

In FTP parlance, the two channels that exist between the two machines are called the *protocol*

| | |
|---|---|
| 4 | Approximate message size negotiation |
| 5 | Status |
| 6 | Timing mark |
| 7 | Remote controlled transmission and echo |
| 8 | Output line width |
| 9 | Output page length |
| 10 | Output carriage-return action |
| 11 | Output horizontal tab stop setting |
| 12 | Output horizontal tab stop action |
| 13 | Output form feed action |
| 14 | Output vertical tab stop setting |
| 15 | Output vertical tab stop action |
| 16 | Output line feed action |
| 17 | Extended ASCII characters |
| 18 | Logout |
| 19 | Bytes macro |
| 20 | Data entry terminal |
| 21 | SUPDUP |
| 22 | SUPDUP output |
| 23 | Send location |
| 24 | Terminal type |
| 25 | End of Record |
| 26 | TACACS user identification |
| 27 | Output marking |
| 28 | Terminal location number |
| 29 | 3270 regime |
| 30 | X 3 PAD (Packet assembly and disassembly) |
| 31 | Window size |

If you refer to the previous code listing with the options toggled on, some of the commands can be understood more clearly now. For example, will ECHO (which would be transmitted as values 255 251 1) instructs the other end to begin echoing back characters it receives. The command won't ECHO (the command would be 255 252 1) indicates that the sender will not echo back characters or wants to stop

OSPF Hello




Note: Fields are not to scale!

OSPF Link state advertisement

| Link State Age (16 bits) |
| --- |
| Options (8 bits) |
| Link State Type (8 bits) |
| Link State ID (32 bits) |
| Advertising Router (32 bits) |
| Link State Sequence Number (32 bits) |
| Link State Checksum (16 bits) |
| Length (16 bits) |

other GAP messages format

| Vers | Type | Code | Status |
|---|---|---|---|
| Checksum | | System No. | |
| Sequence No. | | Hello Interval | |
| Poll Interval | | | |

Neighbor Acquisition

| Vers | Type | Code | Status |
|---|---|---|---|
| Checksum | | System No. | |
| Sequence No. | | Not Used | |
| Source Network IP Address | | | |

Poll

| Vers | Type | Code | Status |
|---|---|---|---|
| Checksum | | System No. | |
| Sequence No. | | | |

Neighbor Reachability

| Vers | Type | Code | Status |
|---|---|---|---|
| Checksum | | System No. | |
| Sequence No. | | Reason | |
| Error Message | | | |
| | | | |

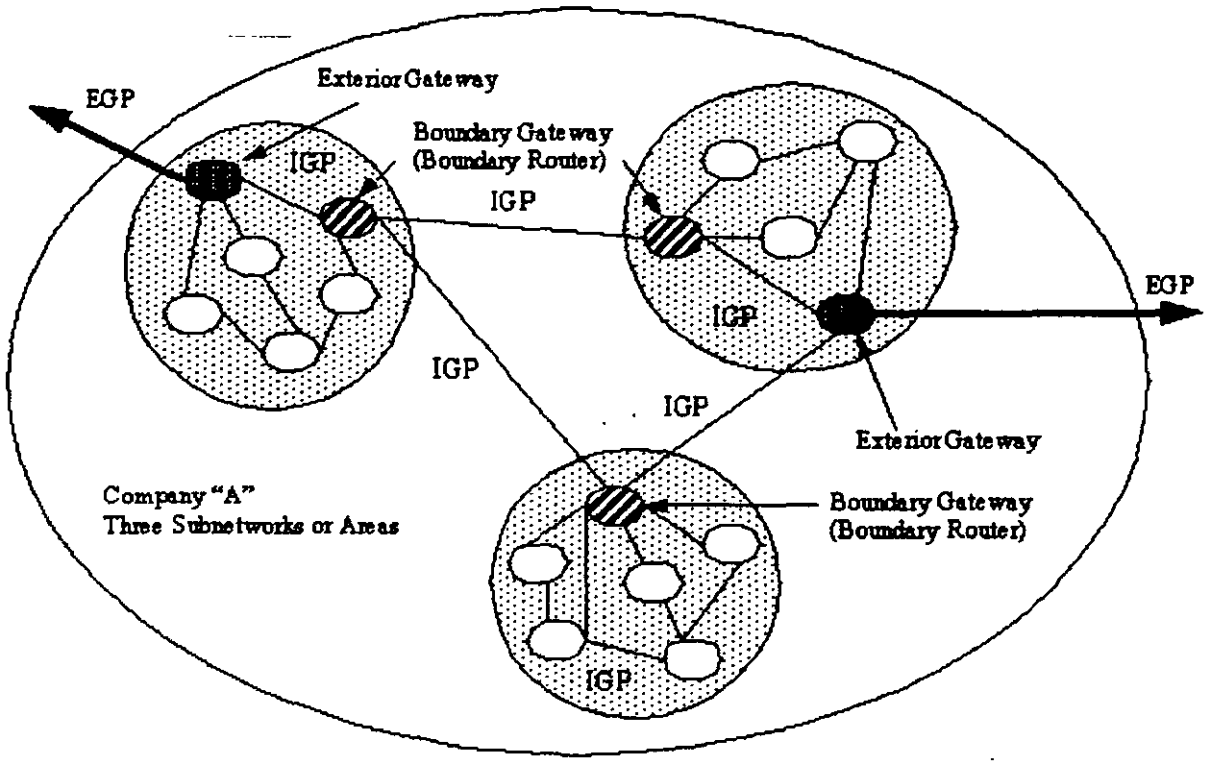
Error Message

Update message format

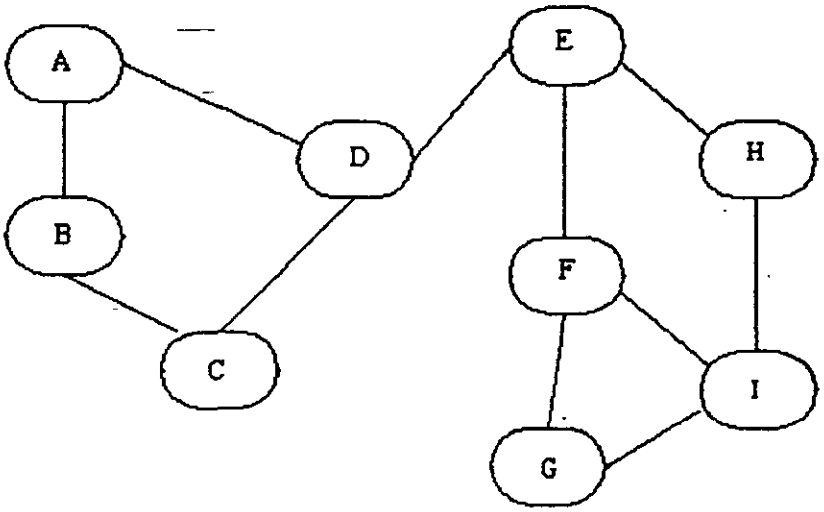
| Version (8 bits) | Message Type (8 bits) |
|---|---|
| Code (8 bits) | Status (8 bits) |
| Checksum (16 bits) | |
| System Number (16 bits) | |
| Sequence Number (16 bits) | |
| Number of Internal Gateways (8 bits) | Number of External Gateways (8 bits) |
| IP Source Network Address (32 bits) | |

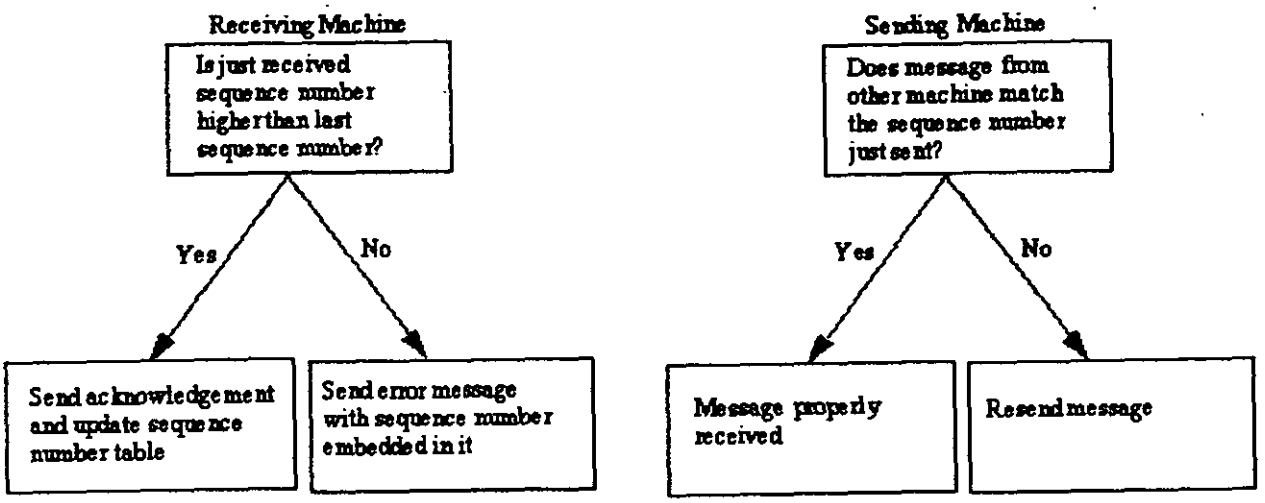
| Gateway 1 IP Address (8 to 24 bits) | |
|---|---|
| Number of Distances (8 bits) | Distance 1 (8 bits) |
| Network 1 IP Address (8 to 24 bits) | |

etc...

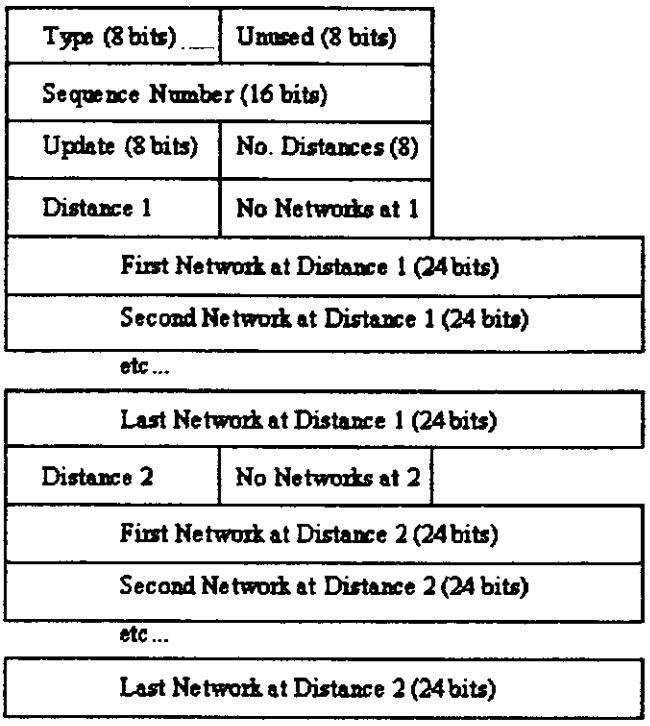

EGP
Exterior Gateway
Boundary Gateway
(Boundary Router)
IGP
IGP
IGP
IGP
IGP
EGP
Exterior Gateway
Company "A"
Three Subnetworks or Areas
Boundary Gateway
(Boundary Router)
IGP

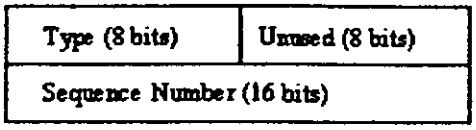
Routing Distance



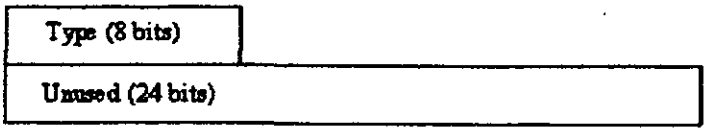

Receiving Machine
Is just received sequence number higher than last sequence number?
Yes
No
Send acknowledgement and update sequence number table
Send error message with sequence number embedded in it
Sending Machine
Does message from other machine match the sequence number just sent?
Yes
No
Message properly received
Resend message

GRP

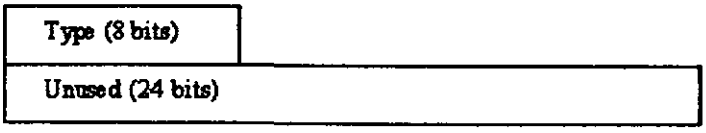

Type (8 bits)
Unused (8 bits)
Sequence Number (16 bits)
Update (8 bits)
No. Distances (8)
Distance 1
No Networks at 1
First Network at Distance 1 (24 bits)
Second Network at Distance 1 (24 bits)
etc...
Last Network at Distance 1 (24 bits)
Distance 2
No Networks at 2
First Network at Distance 2 (24 bits)
Second Network at Distance 2 (24 bits)
etc...
Last Network at Distance 2 (24 bits)
etc...




GGP Acknowledgement




GGP Echo Request and Echo Reply



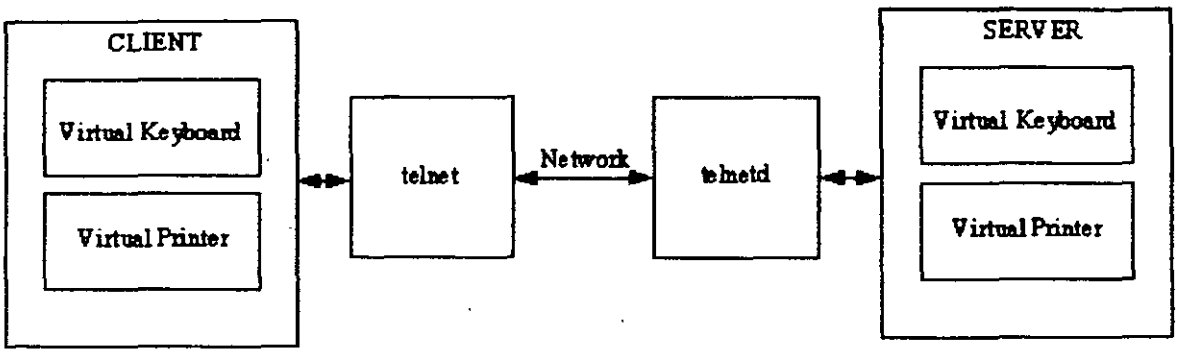

GGP Network Interface Status

OSPF format

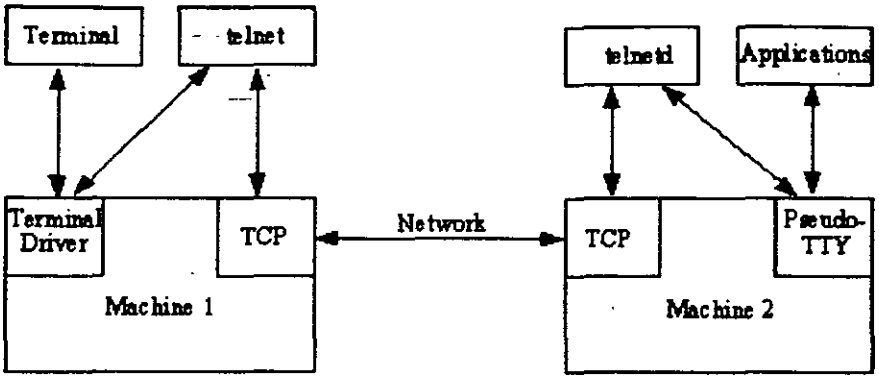
| Version (8 bits) |
| --- |
| Type (8 bits) |
| Packet Length (16 bits) |
| Router ID (32 bits) |
| Area ID (32 bits) |
| Checksum (16 bits) |
| Authentication Type (16 bits) |
| Authentication (64 bits) |

Note: Fields are not to scale!

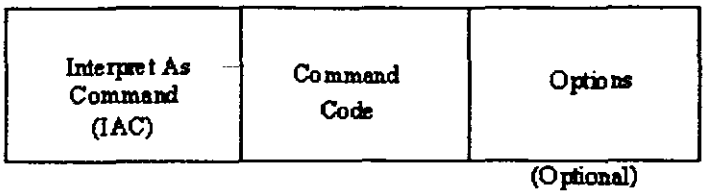
TNVT

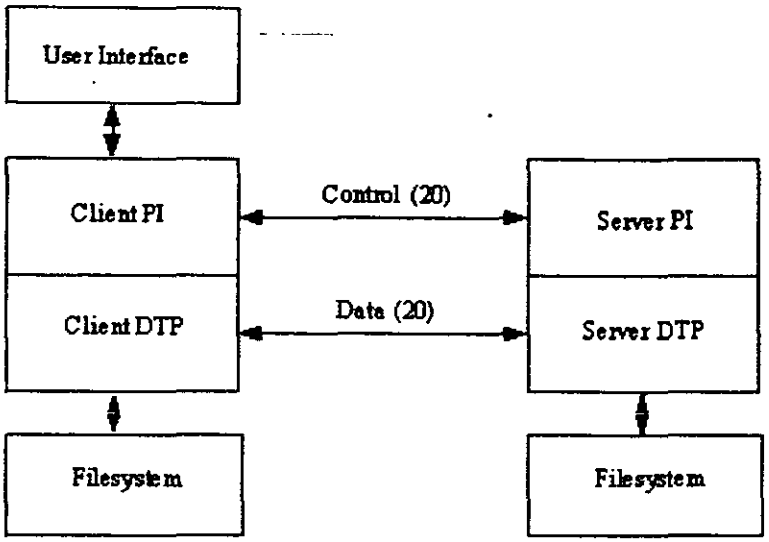

CLIENT
Virtual Keyboard
Virtual Printer
telnet
Network
telnetd
SERVER
Virtual Keyboard
Virtual Printer

Telnet


Terminal
telnet
telnetd
Applications
Terminal Driver
TCP
Network
TCP
Pseudo-TTY
Machine 1
Machine 2

Telnet command



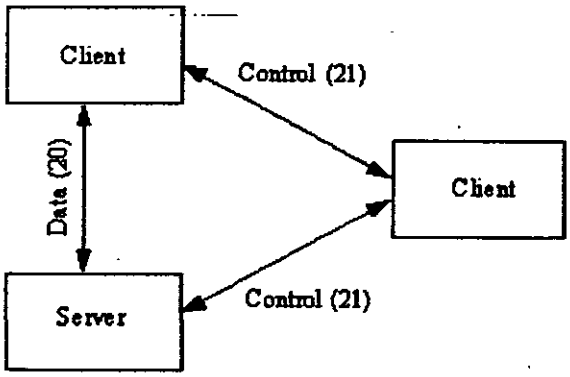



FTP channel connection

Prot Interface

Data Transfer Protocol

Third party FTP



number in the destination machine's packet does not match, the packet that would have been next in sequence from the last correctly received packet is resent.

**What is a core gateway?**

A core gateway is one that resides as an interface between a network and the internetwork. A non-core gateway is between two LANs that are not connected to the larger internetwork.

**Protocol conversion takes place in which of the following: gateways, routers, bridges, or brouters?**

Gateways perform protocol conversion. They have to because they can join two dissimilar network types. Some recent routers and brouters are capable of protocol conversion.

**What are the three types of routing table?**

Routing tables can be fixed (a table that is modified manually every time there is a change), dynamic (one that modifies itself based on network traffic), or fixed central (one downloaded at intervals from a central repository, which can be dynamic).

## Quiz

1. Define the role of gateways, routers, bridges, and brouters.
2. What is a packet-switched network?
3. What is the difference between interior and exterior neighbor gateways?
4. What are the advantages and disadvantages of the three types of routing tables?
5. What is the HELLO protocol used for?

When this type of message is received by another router and it has been validated as containing no errors, the neighbor information can be processed into the neighbor data table.

Another message that is used to initialize the database of a router is the database description packet. It contains information about the topology of the network (either in whole or in part). To provide database description packet service, one router is set as the master, and the other is the slave. The master sends the database description packets, and the slave acknowledges them with database description responses.

The format of the database description packet is shown in Figure 5.16. After the OSPF header is a set of unused bits, followed by three 1-bit flags. When the I (initial) bit is set to 0, it indicates that this packet is the first in a series of packets. The M (more) bit, when set to 1, means that more database description packets follow this one. The MS (master/slave) bit indicates the master/slave relationship. When it has a value of 1 it means that the router that sent the packet is the master. A 0 indicates that the sending machine is the slave. The Data Descriptor Sequence Number is an incrementing counter. The rest of the packet contains Link State advertisements as seen in Figure 5.14.

Figure 5.16. The database description packet layout.

## Link State Request and Update Packets

The Link State Request packet asks for information about a topological table from a database, whereas the Update packet can provide topological information of the types shown in Table 5.11. The Request packet is usually sent when an entry in the router's topological table is corrupted, missing, or out of date. The format of the Link State Request packet is shown in Figure 5.17. The Link State Request packet contains the OSPF header and a block of three repeating fields for the Link State Type, Link State ID, and Advertising Router.

Figure 5.17. OSPF Link State Request packet format.

The Link State Update packet has four formats, depending on the link state type: router links, network links, summary links, or autonomous systems external links. The Router Links advertisement packet is sent to neighbors periodically and contains fields for each router link and the type of service provided in each link, as shown in Figure 5.18.

Figure 5.18. OSPF Router Links advertisement packet format.

After the OSPF header and the Link State advertisement header are two single bit flags surrounded by 6- and 8-bit unused fields. The E (external) flag, when set to 1, indicates that the router is an autonomous systems (AS) boundary router. The B (border) flag, when set to 1, indicates that the router is an area border router. Following the unused 8-bit area is a field for the number of links (advertisements) in the message. Following this, the links are provided in sequence, one link to a block.

Each Link State advertisement block in the Router Links advertisement packet has a field for the Link ID (the type of router, although the value is dependent on the Type field later in the block), the Link Data (whose value is an IP address or a network mask, depending on the Type field's setting), the Type field (a value of 1 indicates a connection to another router, 2 a connection to a transit network, and 3 a connection to a stub network), and the Number of TOS field, which shows the number of metrics for the link (at least one must be provided, which is called TOS 0). Then, a repeating block is appended for each

border routers to their entire area.

- An Autonomous System Extended Links advertisement contains information on routes in external autonomous systems. It is used by boundary routers but covers the entire system.

OSPF maintains several tables for determining routes, including the protocol data table (the high-level protocol in use in the autonomous system), the area data table or backbone data table (which describes the area), the interface data table (information on the router-to-network connections), the neighbor data table (information on the router-to-router connections), and a routing data table (which contains the route information for messages). Each table has a structure of its own, the details of which are not needed for this level of discussion. Interested readers are referred to the RFC for complete specifications.

## OSPF Packets

As mentioned earlier, OSPF uses IP for the network layer. The OSPF specifications provide for two reserved multicast addresses: one for all routers that support OSPF (224.0.0.5) and one for a designated router and a backup router (224.0.0.6). The IP protocol number 89 is reserved for OSPF. When IP sends an OSPF message, it uses the protocol number and a Type of Service (TOS) field value of 0. Usually, the IP precedence field is set higher than normal IP messages, also.

OSPF uses two header formats. The primary OSPF message header format is shown in Figure 5.13. Note that the fields are not shown in their scale lengths in this figure for illustrative purposes. The Version Number field identifies the version of the OSPF protocol in use (currently version 1). The Type field identifies the type of message and might contain a value from those shown in Table 5.11.

Figure 5.13. OSPF message header format.

**Table 5.11. OSPF header Type values.**

| *Type* | *Description* |
|---|---|
| 1 | Hello |
| 2 | Database description |
| 3 | Link state request |
| 4 | Link state update |
| 5 | Link state acknowledgment |

The Packet Length field contains the length of the message, including the header. The Router ID is the identification of the sending machine, and the Area ID identifies the area the sending machine is in. The Checksum field uses the same algorithm as IP to verify the entire message, including the header.

The Authentication Type (AUType) field identifies the type of authentication to be used. There are currently only two values for this field: 0 for no authentication, and 1 for a password. The Authentication field contains the value that is used to authenticate the message, if applicable.

The timers involved with RIP are devoted to each possible route in the routing table. A time-out timer is set when the route is initialized and each time the route is updated. If the timer expires (the default setting is 180 seconds) before another update, the route is considered unreachable. A second timer, called the garbage-collection timer, takes over after the time-out timer and marks when the route is completely expunged from the routing table. The garbage-collection timer has a default value of 120 seconds. If a request for a routing update is received after the time-out timer has expired but before the garbage-collection timer has expired, the entry for that gateway is sent but with the maximum value for the route value. After the garbage-collection timer has expired, the route is not sent at all.

A response timer triggers a set of messages every 30 seconds to all neighboring machines, in an attempt to update routing tables. These messages are composed of the machine's IP address and the distance to the recipient machine.

## The HELLO Protocol

The HELLO protocol is used often, especially where TCP/IP installations are involved. It is different from RIP in that HELLO uses time instead of distance as a routing factor. This requires the network of machines to have reasonably accurate timing, which is synchronized with each machine. For this reason, the HELLO protocol depends on clock synchronization messages.

The format of a HELLO message is shown in Figure 5.12. The primary header fields are as follows:

- A checksum of the entire message
- The current date of the sending machine
- The current time of the sending machine
- A timestamp used to calculate round-trip delays
- An offset that points to the following entries
- The number of hosts that follow as a list

Figure 5.12. HELLO message format.

Following the header are several entries with a delay estimate to the machine and an offset, which is an estimate of the difference between the sending and receiving clocks. The offsets are important because HELLO is a time-critical protocol, so the offset enables correction between times on different machines.

The timestamp on messages is used by machines that the message passes through to calculate delays in the network. In this manner, a routing table based on realistic delivery times can be constructed.

## The Open Shortest Path First (OSPF) Protocol

The Open Shortest Path First protocol was developed by the Internet Engineering Task Force, with the

Hello msg

| Checksum |
| --- |
| Date |
| Time |
| Timestamp |
| Offset |
| Number of hosts |

| Delay for Host 1 |
| --- |
| Offset for Host 1 |

etc...

RIP MSG

| Command Value |
| --- |
| Version Number |
| Reserved |

| Family |
| --- |
| Network Address |
| Network Address |
| Network Address |
| Metric (Distance) |

etc...

| *Name* | *Description* |
|---|---|
| M | Hello polling mode. |
| P1 | Minimum interval acceptable between successive received Hello commands. Default is 30 seconds. |
| P2 | Minimum interval acceptable between successive received Poll commands. Default is 120 seconds. |
| P3 | Interval between Request or Cease command retransmissions. Default is 30 seconds. |
| P4 | Interval during which the state variables are maintained without receiving an incoming message when in the up or down state. Default is one hour. |
| P5 | Interval during which the state variables are maintained without receiving an incoming message when in the cease or acquisition state. Default is 2 minutes. |
| R | Receive sequence number. |
| S | Send sequence number. |
| T1 | Interval between Hello command retransmissions. |
| T2 | Interval between Poll command retransmissions. |
| T3 | Interval during which reachability attempts are counted. |
| t1 | Retransmission timer for Request, Hello, and Cease messages. |
| t2 | Retransmission timer for Poll messages. |
| t3 | Abort timer |

Many of the state parameters are set during the initial establishment of a connection between neighbors. The exceptions are the P1 through P5 values, which are established by the host system and are not modified by the neighbors. The send sequence number is determined only after a message has been received from the other gateway.

A full discussion of the changes between EGP states and the events that occur when a state change occurs is longer than this book and is not of relevance to this level of discussion. Therefore, the original RFC should be consulted for full state condition information. It is useful at this point simply to be aware of the state-driven nature of EGP and to understand that the state can be changed by a message reception, lack of a reply to a message, or expiration of a timer.

## Interior Gateway Protocols (IGP)

There are several IGPs in use, none of which have proven themselves dominant. Usually, the choice of an IGP is made on the basis of network architecture and suitability to the network's software requirements. Earlier today, RIP and HELLO were mentioned. Both are examples of IGPs. Together with a third protocol called Open Shortest Path First (OSPF), these IGPs are now examined in more detail.

Both RIP and HELLO calculate distances to a destination, and their messages contain both a machine

autonomous network, one gateway usually assumes the responsibility for handling this reachability information. In Figure 5.10, gateway A is responsible for sending information about the three networks that lead from it, as well as the two non-core gateways.

EGPs use a polling process to keep themselves aware of their neighbors as they become active or go down, and to exchange routing and status information with all their neighbors. EGP is also a state-driven protocol, meaning that it depends on a state table containing values that reflect gateway conditions and a set of operations that must be performed when a state table entry changes. There are five states, as shown in Table 5.8.

**Table 5.8. EGP states.**

| *State* | *Description* |
|---|---|
| 0 | Idle |
| 1 | Acquisition |
| 2 | Down |
| 3 | Up |
| 4 | Cease |

The meanings of each of these EGP states follow:

- An idle state means the gateway is not involved in any activity and has no resources allocated to it. It usually responds to a message to initialize itself but ignores all other messages unless it switches to either the down or acquisition states.
- An acquisition state enables a gateway to transmit messages but not act as a full messaging gateway. It can receive messages and change to either down or idle states.
- The down state is when the gateway is not operational as far as polling operations are concerned. Messages are neither received nor generated.
- The up state is used whenever a gateway is processing and responding to all EGP messages it receives and can transmit messages.
- A gateway is in cease state when the gateway ceases all updating operations but can still send and receive Cease and Cease Acknowledgment messages.

All EGP messages fall into one of three categories: commands, responses, or indications. A command usually requires an action to be performed, whereas a response is a reply to a command to perform some action. An indication shows current status. Command-response signals are shown in Table 5.9.

**Table 5.9. EGP commands and their responses.**

message. Each gateway tracks its own sequence number for sending to every other gateway it is connected to, as well as the incoming sequence numbers from that gateway. They are not necessarily the same, because more messages might flow one way than the other, although usually each message should have an acknowledgment or reply of some type.

Sequence numbers have important meanings for the messages and are not just for the sake of keeping an incremental count of the traffic volume. When a gateway receives a message from another gateway, it compares the sequence number in that message to the last received sequence number in its internal tables. If the latest message has a higher sequence number than the last message received, the gateway accepts the message and updates its sequence number to the latest received value. If the number was less than the last received sequence number, the message is considered old and is ignored, with an error message containing the just-received message sent back. This process is shown in Figure 5.4.

Figure 5.4. Processing sequence numbers in GGP.

The receiving gateway acknowledges the received message by sending a return message that contains the sequence number of the just-received message. The other gateway compares that number with the number of its last sent message, and if they are the same, the gateway knows that the message was properly received. If the numbers do not match, the gateway knows an error occurred and transmits the message again.

When a message is ignored by the recipient gateway, the sending gateway receives a message with the sequence number of the ignored message. It can then determine which messages were skipped and adjust itself accordingly, resending messages that need to be sent.

The GGP message format is shown in Figure 5.5. After it is constructed, it is encapsulated into an IP datagram that includes source and target addresses. The first field is a message type, which is set to a value of 12 for routing information. The sequence number was discussed earlier and provides an incremental counter for each message. The Update field is set to a value of 0 unless the sending gateway wants a routing update for the provided destination address, in which case it is set to a value of 1. The Number of Distances field holds the number of groups of addresses contained in the current message.

Figure 5.5. The GGP message format.

For each distance group in the message, a distance value and the number of networks that can be reached at that distance are provided, followed by all the network address identifications. According to the GGP standard, not all the distances need to be reported, but the more information supplied, the more useful the message is to each gateway.

GGP does not deal with full Internet addresses specifically, so the host portion of the address does not necessarily have to be included in the address, although the network address is always provided. This can result in different lengths of addresses in the identification field (8, 16, or 24 bits, depending on the type of address).

Three other formats are used with GGP messages, as shown in Figure 5.6. The acknowledgment message uses the Type field to indicate whether the message is a positive acknowledgment (type is set to 2) or a negative acknowledgment (type is set to 10) . The sequence number, as mentioned earlier today, is used to identify the message to which the acknowledgment applies.

Figure 5.6. Other GGP message formats.

The echo request and echo reply formats are passed between gateways to inform the gateways of status changes and to ensure the gateway is up. An echo request has the Type field set to the value 8, whereas an echo reply has the Type field set to a value of 0. Because the address of the sending gateway is embedded in the IP header, it is not duplicated in the GGP message. The remaining 24 bits of the message are unused.

The network interface status message is used by a gateway to ensure that it is able to send and receive messages properly. This type of message can be sent to the originating gateway itself, with the type field set to a value of 9 and the IP address in the header set to the network interface's address

## The External Gateway Protocol (EGP)

As mentioned earlier, an EGP is used to transfer information between non-core neighboring gateways. Non-core gateways contain complete details about their immediate neighbors and the machines attached to them, but they lack information about the rest of the network. Core gateways know about all the other core gateways but often lack the details of the machines beyond a gateway.

EGP is usually restricted to information within the gateway's autonomous system. This prevents too much information from passing through the networks, especially when most of the information that relates to external autonomous systems would be unusable to another gateway. EGP therefore imposes restrictions on the gateways about the machines EGP passes routing information about.

### Neighbors and EGP

Because EGP was developed to enable remote systems to exchange routing information and status messages, the protocol is heavily based in requests or commands followed by replies. The four EGP commands and their possible responses are shown in Table 5.2

**Table 5.2. EGP commands.**

| *Command Name* | *Command Description* | *Response Name* | *Response Description* |
|---|---|---|---|
| Request | Request that a neighbor become a gateway | Confirm/Refuse | Agree or refuse the request |
| Cease | Request the termination of a neighbor | Cease-Ack | Agree to termination |
| Hello | Request confirmation of routing to neighbor (neighbor reachability) | IHU | Confirms the routing |
| Poll | Request that the neighbor provide network information (network reachability) | Update | Provides network information |

To understand Table 5.2 properly, you must understand the concept of *neighbor* to an internetwork. Gateways are neighbors if they share the same subnetwork. They might be gateways to the same network (such as the Internet) or work with different networks. When the two want to exchange information, they must first establish communications between each other; the two gateways are essentially agreeing to exchange routing information. This process is called *neighbor acquisition.*

Neighbor doesn't mean the networks have to be next to each other. They are connected by a gateway, but the networks can be on different continents. The term neighbor has to do with connections, not geography.

The process of becoming neighbors is formal, because one gateway might not want to become a neighbor at that particular time (for any number of reasons, but usually because the gateway is busy). It begins with a Request, which is followed by either an acceptance (Confirm) or refusal (Refuse) from the second machine. If the two gateways are neighbors, either can break the relationship with a Cease message.

After two gateways become neighbors, they assure each other that they are still in contact by occasionally sending a Hello message, to which the second gateway responds with an IHU (I Heard You) message as soon as possible. These Hello/IHU messages can be sent at any time. With several gateways involved on a network, the number of Hello messages can become appreciable as the gateways continue to remain in touch. This process is called *neighbor reachability.*

The other message pair sent by EGP is network reachability, in which case one gateway sends a Poll message and expects an Update message in response. The response contains a list of networks that can be reached through that gateway, with a number representing the number of hops that must be made to reach the networks. By assembling the Update messages from different neighbors, a gateway can decide the best route to send a datagram.

Finally, an error message is returned whenever the gateway cannot understand an incoming EGP message.

## EGP Messages

The layout of the different messages used by EGP are shown in Figure 5.7. The fields have the following meanings:

- The Version field holds the EGP version number of the sending machine (the current version is 2).
- The Type field identifies the type of EGP messages. There are ten message types in EGP.
- The Code field contains a value that identifies the subtype of the message.
- The Status field is used with the Type and Code fields to reflect the current status of the gateway's state.
- The Checksum is calculated for the EGP message in the same manner as other TCP/IP headers.
- The System Number is an identification of the autonomous system that the sending gateway belongs to.

- The Sequence Number of the message is an incrementing counter for each message, also used to identify a reply to a previous message.

Figure 5.7. EGP message format.

The Reason field of the Error message can contain one of the following integers:

0—Unspecified error

1—Bad EGP header

2—Bad EGP data field

3—Reachability information not available

4—Excessive polling

5—No response received to a poll

Through a combination of the message Type, Code, and Status fields, the purpose and meaning of the EGP message can be more accurately determined. Table 5.3 shows all codes and status values.

**Table 5.3. EGP messages.**

all physically connected together. These gateways communicate through an EGP.

There are fewer rules governing IGPs than EGPs simply because the IGP can handle custom-developed applications and protocols within its local network. When the Internet is used for gateway-to-gateway communications, the messages must conform to the internetwork standards. Also, when connecting two subnetworks, it is possible to send only one message to the subnetwork gateway through EGP, which can then be duplicated, modified, and propagated to all gateways on the internal system using IGP. EGP has formalized rules governing its use.

## Gateway-to-Gateway Protocol (GGP)

GGP is used for communications between core gateways. A recent improvement of the protocol, called SPREAD, is starting to be used but is not yet as common as GGP. Even if GGP is phased out in favor of SPREAD, it is a useful illustration of gateway-to-gateway protocols.

GGP is a vector-distance protocol, meaning that messages tend to specify a destination (vector) and the distance to that destination. Vector-distance protocols are also called Bellman-Ford protocols, after the researchers who first published the idea. For a vector-distance protocol to be effective, a gateway must have complete information about all the gateways on the internetwork; otherwise, computing a distance with a fewest-hops type of protocol cannot succeed.

---


You might recall from earlier today that core gateways have complete information about all other core gateways, so a vector-distance protocol works. Non-core gateways don't have a complete internetwork map, so GGP-type messages are not useful.

---

A gateway establishes its connections to other gateways by sending out messages, waiting for replies, and then building a table. This is initially accomplished when a gateway is installed and has no routing information at all. This aspect of communications is not defined within GGP but relies on network-specific messages. Once the initial table has been defined, GGP is used for all messages.

Connectivity with another gateway on the Internet is determined using the K-out-of-N method. In this procedure, a gateway sends an echo message to another gateway and waits for a reply. It repeats this every fifteen seconds. According to the Internet standards, if the gateway does not receive three (K) replies out of four (N) requests, the other gateway is considered down, or unusable, and routing messages are not sent to that gateway. This process can be repeated at regular intervals.

If a down gateway becomes active again, the Internet standards require two out of four echo messages to be acknowledged. This is called J-out-of-M, where J is two and M is four. The Internet-assigned values for J, K, M, and N can be changed for autonomous networks, but the standard defines the values for use on the Internet itself.

Each message between gateways has a sequence number that is incremented with each transmitted

Suppose that a gateway link within one of the networks was broken due to a machine or connection failure, such as that between gateway C and machine X in Figure 5.3. Gateway C would find out about the problem through an IGP message and update its routing table to reflect the break, usually by putting the largest legal value for routing length in that entry. (Remember that IGP is a general term for any internal network protocol for gateway communications, such as RIP or HELLO.) Gateway C transfers its new copy of the routing table to gateway A.

Routing a message to machine Y would now be impossible through the C–X connection. However, because gateway A has the routing information from C, and it exchanges routing information with gateway B, which also exchanges with gateway D, any message passing through either D or B for machine Y could be rerouted up through gateway A, then C, and finally to Y. An EGP message between B–D and A–C would indicate that the new route costs less than the maximum value assigned going through C–X (which is broken), so the round-about transfer through the four gateways can be used.

EGP messages between gateways are usually sent whenever a connection problem exists and the routing information is set to its maximum (worst) value, or when a better connection alternative has been discovered for some reason. This can be because of an update from a remote gateway's routing table, or the addition of new connections, machines, or networks to the system. Whichever happens, an EGP message informs all the connected gateways of the changes.

## The IGP and EGPGateway Protocols

Gateways need to know what is happening to the rest of the network in order to route datagrams properly and efficiently. This includes not only routing information but also the characteristics of subnetworks. For example, if one gateway is particularly slow but is the only access method to a subnetwork, other gateways on the network can tailor the traffic to suit.

A GGP is used to exchange routing information between devices. It is important not to confuse routing information, which contains addresses, topology, and details on routing delays, with the algorithms used to make routing information. Usually the routing algorithms are fixed within a gateway and not modified. Of course, as the routing information changes, the algorithm adapts the chosen routes to reflect the new information.

GGPs are primarily for autonomous (self-complete) networks. An autonomous system uses gateways that are connected in one large network, such as one might find in a large corporation. Two kinds of gateways must be considered in an autonomous network. The gateways between smaller subnetworks help tie the small systems into the larger corporate network, but the gateways for each subnetwork are usually under the control of one system (usually in the IS department). These gateways are considered autonomous because the connections between gateways are constant and seldom change. These gateways communicate through an IGP.

Large internetworks like the Internet are not as static as corporate systems. Gateways can change constantly as the subsidiary networks make changes, and the communications routes between gateways are more subject to change, too. For widely spread companies, there might be gateways spread throughout the country (or the world) that are all part of the same corporate network but use the Internet to communicate. The communications between these gateways are slightly different than when they are

This type of routing depends on the type of routing service available from gateway to gateway. This is called *type of service* (TOS) routing. It is also more formally called *quality of service* (QOS) by OSI. TOS includes consideration for the speed and reliability of connections, as well as security and route-specific factors.

To effect TOS routing, most systems use dynamic updating of tables that reflect traffic and link conditions. They also take into account current queue lengths at each gateway, because the fastest theoretical route might not matter if the message is backlogged in a queue. This information is obtained through the frequent transfer of status messages between gateways, especially when conditions deteriorate.

Dynamic updating of tables can have a disadvantage in that if tables are updated too frequently, a message might circulate through a section of the internetwork without proper routing to its destination, or proceed through a long and convoluted path. For this reason, dynamic updating occurs at regular but not too frequent intervals. To prevent stray datagrams from circulating on the internetwork too long, the Time to Live information in the IP message header is important.


The IP header's Time to Live (TTL) field is very important to dynamic gateway routing protocols, which is why it is a mandatory field. Without it, datagrams could circulate throughout the network indefinitely.

The dynamic nature of TOS routing can sometimes cause a message's fragments to be routed in different ways to a destination. For example, if a long message of 10 datagrams is being sent by one route, but the routing tables are changed during transmission to reflect a backlog, the remainder of the datagrams might be sent via an alternate route. This doesn't matter, of course, because the receiving machine reassembles the message in the proper order as the datagrams are received.

## Updating Gateway Routing Information

A somewhat simplified example of a dynamic update is useful at this stage. The exact communications protocols between gateways are examined in more detail later today.

Assume that two autonomous networks are connected to each other at two locations, as shown in Figure 5.3, with connections to different autonomous networks at other locations. The A–C connection and the B–D connection can both be used for routing from within the networks, depending on which is the optimal path. Gateway C has a copy of gateway A's routing table, and vice versa. Gateways B and D each have copies of the other's routing tables, as well. These copies are transmitted at intervals so the gateways can maintain an up-to-date picture of the connections available through the other gateway. The gateways use EGP to send the messages. (They would use GGP if they were core gateways.)

Figure 5.3. Two interconnected networks.

## Fewest-Hops Routing

Most networks and gateways to internetworks work on the assumption that the shortest route (in terms of machines traveled through) is the best way to route messages. Each machine that a message passes through is called a *hop*, so this routing method is known as *fewest hops*. Although experimentation has shown that the fewest-hops method is not necessarily the fastest method (because it doesn't take into account transmission speed between machines), it is one of the easiest routing methods to implement.

To provide fewest-hops routing, a table of the distance between any two machines is developed, or an algorithm is available to help calculate the number of hops required to reach a target machine. This is shown using the sample internetwork of gateways in Figure 5.2 and its corresponding table of distances between the gateways in the figure, which is shown in Table 5.1.

Figure 5.2. An internetwork of gateways.

**Table 5.1. Table of fewest hops from Figure 5.2.**

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* |
|---|---|---|---|---|---|---|---|---|---|
| **A** | | 1 | 2 | 1 | 2 | 3 | 4 | 3 | 4 |
| **B** | 1 | | 1 | 2 | 3 | 4 | 5 | 4 | 5 |
| **C** | 2 | 1 | | 1 | 2 | 3 | 4 | 3 | 4 |
| **D** | 1 | 2 | 1 | | 1 | 2 | 3 | 2 | 3 |
| **E** | 2 | 3 | 2 | 1 | | 1 | 2 | 1 | 2 |
| **F** | 3 | 4 | 3 | 2 | 1 | | 1 | 2 | 1 |
| **G** | 4 | 5 | 4 | 3 | 2 | 1 | | 2 | 1 |
| **H** | 3 | 4 | 3 | 2 | 1 | 2 | 2 | | 1 |
| **I** | 4 | 5 | 4 | 3 | 2 | 1 | 1 | 1 | |

When a message is to be routed using the fewest-hops approach, the table of distances is consulted, and the route with the fewest number of hops is selected. The message is then routed to the gateway that is closest to the destination network. When intermediate gateways receive the message, they perform the same type of table lookup and forward to the next gateway on the route.

There are several problems with the fewest-hops approach. If the tables of the gateways through which a message travels to its destination have different route information, it is conceivable that a message that left the source machine on the shortest route could end up following a more circuitous path because of differing tables in the intervening gateways. The fewest-hops method also doesn't account for transfer speed, line failures, or other factors that could affect the overall time to travel to the destination; it is merely concerned with the shortest apparent distance, assuming that all connections are equal. To accommodate these factors, another routing method must be used.

## Type of Service Routing

/etc/gateways.

It has become common practice to allow a default network Internet address of 0.0.0.0, which refers to a gateway on the network that should be capable of resolving an unknown address. (This is included in the previous sample configuration file as *proto default*.) The default route is used when the local machine cannot resolve the address properly. Because the routing tables on a gateway are usually more complete than those on a local machine, this helps send packets to their intended destination. If the default address gateway cannot resolve the address, an Internet Control Message Protocol (ICMP) error message is returned to the sender.

## Routing

Routing refers to the transmission of a packet of information from one machine through another. Each machine that the packet enters analyzes the contents of the packet header and decides its action based on the information within the header. If the destination address of the packet matches the machine's address, the packet should be retained and processed by higher-level protocols. If the destination address doesn't match the machine's, the packet is forwarded further around the network. Forwarding can be to the destination machine itself, or to a gateway or bridge if the packet is to leave the local network.

Routing is a primary contributor to the complexity of packet-switched networks. It is necessary to account for an optimal path from source to destination machines, as well as to handle problems such as a heavy load on an intervening machine or the loss of a connection. The route details are contained in a routing table, and several sophisticated algorithms work with the routing table to develop an optimal route for a packet.

Creating a routing table and maintaining it with valid entries are important aspects of a protocol. Here are a few common methods of building a routing table:

- A fixed table is created with a map of the network, which must be modified and reread every time there is a physical change anywhere on the network.
- A dynamic table is used that evaluates traffic load and messages from other nodes to refine an internal table.
- A fixed central routing table is used that is loaded from the central repository by the network nodes at regular intervals or when needed.

Each method has advantages and disadvantages. The fixed table approach, whether located on each network node or downloaded at regular intervals from a centrally maintained fixed table, is inflexible and can't react to changes in the network topology quickly. The central table is better than the first option, simply because it is possible for an administrator to maintain the single table much more easily than a table on each node.

The dynamic table is the best for reacting to changes, although it does require better control, more complex software, and more network traffic. However, the advantages usually outweigh the disadvantages, and a dynamic table is the method most frequently used on the Internet.

```
traceoptions general kernel icmp egp protocol ;
autonomcussystem 519 ;
rip no;
egp yes {
     group ASin 519 {
          neighbor  128.212.64.1 ;
     } ;
} ;
static {
     default gateway 128.212.64.1 pref 100 ;
} ;
propagate proto egp as 519 {
     proto rip gateway 128.212.64.1 {
          announce 128.212 metric 2 ;
     } ;
     proto direct {
          announce 128.212 metric 2 ;
     } ;
} ;
propagate proto rip {
     proto default {
          announce 0.0.0.0 metric 1 ;
     } ;
     proto rip {
          noannounce all ;
     } ;
} ;
```

The code above shows a number of configuration details. It starts with a number of options and the switch that turns EGP on and sets the neighbor IP address. This is followed by code that defines the way EGP behaves. Most of the details are of little interest and are seldom (if ever) modified by a user. Instead, configuration routines tend to manage this file's contents.

The UNIX system administrator also has a program called route that enables direct entry of routing table information. The information on a UNIX system regarding routing is usually stored in the file

gateways to update routing tables.


The three gateway protocols are intertwined: EGP is used between gateways of autonomous systems, whereas the IGPs RIP and HELLO are used within the network itself. GGP is used between core gateways.

Why not use GGP for all internetwork communications, dropping the need for EGPs? The answer lies in the fact that core gateways that use GGP know about all the other core gateways on the internetwork. This simplifies their messaging and provides complete routing tables. However, core gateways usually lead into many complex networks of more autonomous networks, most of which the core gateways don't know about. However, the exterior gateways must know about all the networks directly connected to it, but not all the networks on the entire internetwork, so the routing tables and routing algorithms for a core and non-core gateway are different. This also means that messages can have different formats, because routing information for a non-core gateway has some connections that are hidden from other gateways.

It is possible for a large autonomous system to be composed of several subnetworks or areas, each of which communicates with the other areas through an IGP. Each subnetwork or area has a designated gateway, called a *border gateway*, or *border router*, to indicate that it is within an area. Border routers communicate with each other using IGP. A commonly encountered term is *boundary gateway*, which is the same as an exterior gateway or a path to another autonomous network. This is illustrated in Figure 5.1, which shows three subnetworks or areas that communicate with each other through boundary gateways or routers using IGP, and two exterior gateways (also called boundary gateways) that communicate with the rest of the internetwork using EGP.

Figure 5.1. Interior and exterior gateways.

## Routing Daemons

To handle the routing tables, most UNIX systems use a daemon called routed. A few systems run a daemon called gated. Both routed and gated can exchange RIP messages with other machines, updating their route tables as necessary. The gated program can also handle EGP and HELLO messages, updating tables for the internetwork. Both routed and gated can be managed by the system administrator to select favorable routes, or to tag a route as not reliable.

The configuration information for gated and routed is usually stored as files named gated.cfg, gated.conf, or gated.cf. Some systems specify gated information files for each protocol, resulting in the files gated.egp, gated.hello, and gated.rip. A sample configuration file for EGP used by the gated process is shown here:

```
#     @(#)gated.egp 4.1 Lachman System V STREAMS TCP  source
#   sample EGP config file
```

line-switched (fixed or dedicated connections) networks, but these are rarely used with TCP/IP. Packet-switched networks tend to be faster overall than message-switched networks, but they are also considerably more complex.

## Gateway Protocols

Gateway protocols are used to exchange information with other gateways in a fast, reliable manner. Using gateway protocols, transmission time over large internetworks has been shown to increase, although there is considerable support for the idea of having only one protocol across the entire Internet (which would eliminate gateway protocols in favor of TCP/IP throughout).

The Internet provides for two types of gateways: core and non-core. All core gateways are administered by the Internet Network Operations Center (INOC). Non-core gateways are not administered by this central authority but by groups outside the Internet hierarchy (who might still be connected to the Internet but administer their own machines). Typically, corporations and educational institutions use non-core gateways.

The origin of core gateways arose from the ARPANET, where each node was under the control of the governing agency. ARPANET called them *stub gateways*, whereas any gateway not under direct control (non-core in Internet terms) was called a *nonrouting gateway*. The move to the Internet and its proliferation of gateways required the implementation of the Gateway-to-Gateway Protocol (GGP), which was used between core gateways. The GGP was usually used to spread information about the non-core gateways attached to each core gateway, enabling routing tables to be built.

As the Internet grew, it became impossible for any one gateway to hold a complete map of the entire internetwork. This was solved by having each gateway handle only a specific section of the internetwork, relying on neighboring gateways to know more about their own attached networks when a message was passed. One problem that frequently occurred was a lack of information for complete routing decisions, so default routes were used.

Earlier in this book, the term *autonomous system* was introduced. An autonomous system is one in which the structure of the network it is attached to is not visible to the rest of the internetwork. Usually, a gateway leads into the network, so all traffic for that network must go through the gateway, which hides the internal structure of the local network from the rest of the internetwork.

If the local network has more than one gateway and they can talk to each other, they are considered interior neighbors. (The term interior neighbor is sometimes applied to the machines within the network, too, not just the gateways.) If the gateways belong to different autonomous systems, they are exterior gateways. Thus, when default routes are required, it is up to the exterior gateways to route messages between autonomous systems. Interior gateways are used to transfer messages into an autonomous system.

Within a network, the method of transferring routing information between interior gateways is usually the Routing Information Protocol (RIP) or the less common HELLO protocol, both of which are Interior Gateway Protocols (IGPs). These protocols are designed specifically for interior neighbors. On the Internet, messages between two exterior gateways are through the Exterior Gateway Protocol (EGP). RIP, HELLO, and EGP all rely on a frequent (every thirty seconds) transfer of information between

detail by looking at the manner in which gateways transfer routing information between themselves.

The routing method used to send a message from its origin to destination is important, but the method by which the routing information is transferred depends on the role of the network gateways. There are special protocols developed specifically for different kinds of gateways, all of which function with TCP.

## Gateways, Bridges, and Routers

To forward messages through networks, a machine's IP layer software compares the destination address of the message (contained in the Protocol Data Unit, or PDU) to the local machine's address. If the message is not for the local machine, the message is passed on to the next machine. Moving messages around small network is quite easy, but large networks and internetworks add to the complexity, requiring gateways, bridges, and routers, which try to establish the best method of moving the message to its destination.

Defining the meaning of these terms is relatively easy:

- A gateway is a device that performs routing functions, usually as a stand-alone device, that also can perform protocol translation from one network to another.
- A bridge is a network device that connects two or more networks that use the same protocol.
- A router is a network node that forwards datagrams around the network.

The gateway's protocol conversion capability is important (otherwise, the machine is no different from a bridge). Protocol conversion usually takes place in the lower layers, sometimes including the transport layer. Conversion can occur in several forms, such as when moving from a local area network format to Ethernet (in which case the format of the packet is changed) or from one proprietary file convention to another (in which case the file specifications are converted).

Bridges act as links between networks, which often have a bridge at either end of a dedicated communications line (such as a leased line) or through a packet system such as the Internet. There might be a conversion applied between bridges to increase the transmission speed. This requires both ends of the connection to understand a common protocol.

Routers operate at the network level, forwarding packets to their destination. Sometimes a protocol change can be performed by a router that has several delivery options available, such as Ethernet or serial lines.

A term you might occasionally see is *brouter*, a contraction of both bridge and router. As you might expect, brouters perform the functions of both a bridge and a router, although sometimes not all functions are provided. The term brouter is often applied for any device that performs some or all of the functions of both a bridge and a router.

A term in common use when dealing with routes is *packet-switching*. A packet-switched network is one in which all transfers are based on a self-contained packet of data (like that of TCP/IP's datagrams). There are also message-switched (self-contained complete messages, as with UNIX's UUCP system) and




— 5 —

# Gateway and Routing Protocols

TCP/IP functions perfectly well on a local area network, but its development was spurred by internetworks (more specifically by the Internet itself), so it seems logical that TCP/IP has an architecture that works well with internetwork operations. Today I examine these internetwork specifics in more



# — 6 —
# Telnet and FTP

In the last five days you have seen the architecture of TCP/IP, as well as both the Internet Protocol and the Transmission Control Protocol in considerable detail. Building on these two protocols is a layer of application-layer protocols that are commonly associated with TCP/IP. Today I look at the most common application layer protocols: Telnet, File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP), and Simple Mail Transfer Protocol (SMTP), as well as a suite of tools called the Berkeley r-utilities.

To cover all four protocols in complete detail would require several hundred pages, so today I examine the protocols' most important aspects, including their purposes, their relations to TCP and IP, their control codes and behavior, and their typical usage. Each of the four application layer protocols has advantages that make it ideally suited for a particular purpose. I hope that by the end of the day you will understand why they are used and how they fit into the TCP/IP world.

## Telnet

The Telnet (telecommunications network) program is intended to provide a remote login or virtual terminal capability across a network. In other words, a user on machine A should be able to log into machine B anywhere on the network, and as far as the user is concerned, it appears that the user is seated in front of machine B. The Telnet service is provided through TCP's port number 23 (see Table 4.1 or Appendix D, "Well Known Port Numbers," for the TCP port numbers). The term Telnet is used to refer to both the program and the protocol that provide these services.

Telnet was developed because at one time the only method of enabling one machine to access another machine's resources (including hard drives and programs stored there) was to establish a link using communications devices such as modems or networks into dedicated serial ports or network adapters. This is a little more complicated than might appear at first glance because of the wide diversity of terminals and computers, each with their own control codes and terminal characteristics. When directly connected to another machine, the machine's CPU must manage the translation of terminal codes between the two, which puts a hefty load on the CPU. With several remote logins active, a machine's CPU can spend an inordinate amount of time managing the translations. This is especially a problem with servers that can handle many connections at once: if each had to be handled with full terminal translation, the server CPU could be bogged down just performing this function.

Telnet alleviates this problem by embedding the terminal characteristic sequences within the Telnet protocol. When two machines communicate using Telnet, Telnet itself can determine and set the communications and terminal parameters for the session during the connection phase. The Telnet protocol includes the capability not to support a service that one end of the connection cannot handle. When a connection has been established by Telnet, both ends have agreed upon a method for the two machines to exchange information, taking the load off the server CPU for a sizable amount of this work.

Usually, Telnet involves a process on the server that accepts incoming requests for a Telnet session. On UNIX systems, this process is called telnetd. On Windows NT and other PC-based operating systems, a Telnet Server program is usually involved. The client (the end doing the calling) runs a program, usually called telnet, that attempts the connection to the server. A relative of the telnet program is the program rlogin, which is common on UNIX machines and which I look at later today; see the section titled "The Berkeley Utilities."

---


The rlogin program provides almost identical functionality to Telnet and adds support for the UNIX environment. Many machines, especially UNIX workstations, act as both client and server simultaneously, enabling a user to log

into other machines on the network and other users to log into the user's machine.

## Telnet Connections

The Telnet protocol uses the concept of a *network virtual terminal,* or NVT, to define both ends of a Telnet connection. Each end of the connection (each NVT) has a logical keyboard and printer. The logical printer can display characters, and the logical keyboard can generate characters. The logical printer is usually a terminal screen, whereas the logical keyboard is usually the user's keyboard, although it could be a file or other input stream. These terms are also used in the File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP). Figure 6.1 illustrates the NVT and logical keyboard and printer.

Figure 6.1. A network virtual terminal for Telnet.

The Telnet protocol treats the two ends of the connection as NVTs. The two programs at either end (telnet and telnetd for a UNIX server) manage the translation from virtual terminals to actual physical devices. The concept of virtual terminals enables Telnet to interconnect to any type of device, as long as a mapping is available from the virtual codes to the physical device. One advantage of this approach is that some physical devices cannot support all operations, so the virtual terminal does not have those codes. When the two ends are establishing the connection, the lack of these codes is noted, and sequences that would use them are ignored. This process is straightforward: one end asks whether the function is supported, and the other replies either positively or negatively. If it is supported, the necessary codes are sent. The list of supported functions is covered quickly in this manner.

When a connection is established through Telnet, telnetd (or whatever program is acting as the Telnet server) starts a process on the server for running applications. Every keystroke in a Telnet session must go through several different processes, as shown in Figure 6.2. Each keystroke goes through telnet, telnetd, and the applications that are used during the Telnet session. Some applications want to communicate through a terminal device, so the remote system runs a pseudo-TTY driver that acts like a terminal to the application. If a windowed interface such as X or Motif is used on the host and remote machines, the systems must be instructed to enable windowing information to be passed back and forth; otherwise, the remote machine tries to open the windows on the server.

Figure 6.2. A Telnet connection.

To start Telnet, you must provide either the name or the IP address of the machine to be connected with. The name can be used only if the system has a means of resolving the name into its IP address, such as with the Domain Name System. A port name can usually be used to connect to a specific service, but this is used infrequently. For example, to connect to a machine with the IP address 205.150.89.1, you would enter this command:

```
telnet 205.150.89.1
```

If the system had the name darkstar, which was resolvable into its IP address, you could issue this command:

```
telnet darkstar
```

If no name, address, or port is specified, Telnet enters its command mode and waits for specific instructions. When the connection is established, a user ID and password are requested. You can log in with any user ID that is valid on the remote system (it does not have to be the same user ID you have on the local system). A typical connection to a UNIX server looks like this:

```
telnet 205.150.89.1

Trying...

Connected to tpci

Escape character is '^]'.

HP-UX tpci A.09.01 A 9000/720 (ttys2)

login: tparker

password: xxxxxxxx

$
```

As you can see in the preceding code, Telnet tried to connect to the remote system, told you it was connected, then set up the communications parameters between the two systems. When that was done, the login prompt was displayed (as on any UNIX terminal), followed by a password request. If the login and password are enabled, the UNIX shell prompt (a dollar sign) is shown to indicate that the remote machine is now active.

You can use a machine name as part of the Telnet command only if the system has a means of resolving the name to its IP address. If not, no connection is established, although Telnet might remain in command mode. To exit, use Ctrl+D or the break sequence displayed as part of the start-up message.

You can enter Telnet's command mode at any time, usually by using the Ctrl+] key combination (hold down Ctrl and press the right bracket key). If you are currently connected to an active session when you enter command mode, Telnet waits for you to issue a command, execute it, and then return to the session automatically. Command mode lets you enter commands relative to the client (the machine you are physically in front of) instead of the server. You might need to do this to change directories or run a local application, for example.

Once the connection is successfully established, your session behaves as though you were on the remote machine, with all valid commands of that operating system. All instructions are relative to the server, so a directory command shows the current directory on the server, not the client. To see the client's directory, you would have to enter command mode. A sample Telnet login and logout session, calling from one UNIX workstation (merlin) to a server (tpci_hpws4, a name that can be resolved by the name server) follows:

```
merlin> telnet tpci_hpws4

Trying...
```

```
Connected to tpci_hpws4.

Escape character is '^]'.

HP-UX tpci_hpws4 A.09.01 A 9000/720 (ttys2)

login: tparker

password: xxxxxxxx

tpci_hpws4-1> pwd

/u1/tparker

tpci_hpws4-2> cd docs

tpci_hpws4-3> pwd

/u1/tparker/docs

tpci_hpws4-2> <Ctrl+d>

Connection closed by foreign host.

merlin>
```

Once you are connected to the remote machine, the session behaves exactly as if you were on that machine. To log out of the remote session, simply issue the logout command (in the previous example, the UNIX Ctrl+D combination), and you are returned to your local machine. The telnet program is useful when you are on an under-powered machine or terminal and you want to use another machine's processing capabilities, or if another machine has a particular tool that you don't want to load on your local machine.

Telnet utilities are available for many different operating systems. Figure 6.3 shows a Windows for Workgroups Telnet application (part of a larger TCP/IP application suite from NetManage called ChameleonNFS, which I look at in much more detail on Day 10, "Setting Up a Sample TCP/IP Network: DOS and Windows Clients") logging into an SCO UNIX server. Even when the local machine has a graphical interface such as Windows, you can most likely connect to remote machines using a character-based interface.

Figure 6.3. Using Telnet from a Windows for Workgroups machine.

If the calling and receiving workstations use a graphical user interface (GUI) such as Motif or X, and you want to use them instead of a character-based interface, you must instruct both ends to use the local terminal for windowing (because you can't see a window on the remote terminal). Locally, a program is run that instructs the operating system to enable other machines to display directly onto the screen, and the remote must have an instruction to redirect windowing commands to the local screen. Many UNIX systems perform this function like this:

```
tpci_server-1> xhost +

tpci_server-2> telnet tpci_hpws4

Trying...

Connected to tpci_hpws4.
```



SG

```
Escape character is '^]'.

HP-UX tpci_hpws4 A.09.01 A 9000/720 (ttys2)

login: tparker

password: xxxxxxxx

tpci_hpws4-1> setenv DISPLAY tpci_server:0.0

tpci_hpws4-2> <Ctrl+d>

Connection closed by foreign host.

tpci_server-3>
```

The UNIX xhost + instruction tells the local machine to enable the remote system to control windows on the local screen (which it normally is not allowed to do). The instruction setenv DISPLAY *machine_name* executed on the remote UNIX machine sets the UNIX shell environment variable DISPLAY to the local screen. Whenever a window must be opened (as when a Motif application is run), the windowing appears on the local screen, and the processing is conducted on the remote. These examples are for UNIX, but a similar sequence works on other machines and GUIs.

Complete applications that provide this capability to run local X and Motif windows on a Windows, Windows 95, or Windows NT machine are available from several commercial vendors. For example, Figure 6.4 shows an application running on a remote server called mandel that draws Mandelbrot figures. The server has been instructed to display the window on the local Windows for Workgroups machine using an X client package for Windows machines. The server passes all information about the size, position, and colors of the window, as well as instructions for drawing the contents to the local X client. The window appears on the Windows for Workgroups machine exactly as it would on the UNIX server.

Figure 6.4. Using an X client to show UNIX X windows on a PC.

## Telnet Commands

Several service options are available when a Telnet session is established. Their values can be changed during the course of a Telnet session if both ends agree (one end might be prevented from enabling or disabling a service because of administrator or resource settings). There are four verbs used by the Telnet protocol to offer, refuse, request, and prevent services: will, won't, do, and don't, respectively. The verbs are designed to be paired (will/won't and do/don't). To illustrate how these are used, consider the following Telnet session, which has the display of these verbs turned on using the telnet command toggle options:

```
tpci_server-1> telnet

telnet> toggle options

Will show option processing.

telnet> open tpci_hpws4

Trying...
```

```
---> SYST
215 UNIX Type: L8
Remote system type is UNIX.
---> Type I
200 Type set to I.
Using binary mode to transfer files.
ftp> ls
---> PORT 47,80,10,28,4,175
200 PORT command successful.
---> TYPE A
200 Type set to A.
---> LIST
150 Opening ASCII mode data connection for /bin/ls.
total 4
-rw-r-----  1 tparker  tpci    2803  Apr 29 10:46 file1
-rw-rw-r--  1 tparker  tpci    1286  Apr 14 10:46 file5_draft
-rwxr-----  2 tparker  tpci   15635  Mar 14 23:23 test_comp_1
-rw-r-----  1 tparker  tpci      52  Apr 22 12:19 xyzzy
Transfer complete.
---> TYPE I
200 Type set to I.
ftp> <Ctrl+d>
tpci_hpws1-2>
```

You might notice in the previous code how the mode changes from binary to ASCII to send the directory listing, and then back to binary (the system default value). You can see how the two systems communicate to display the status messages that appear without the debugging option active.

When FTP is used in a graphical user environment, you might be able to use a GUI-based tool. For example, NetManage's ChameleonNFS provides the FTP utility shown in Figure 6.8. In this case, the NFS client on the Windows for Workgroups machine has connected to a UNIX server. The Local side of the window shows the Windows machine, and the Remote side of the window shows the UNIX box's current filesystem contents. When using a GUI-based utility like this one, you can use the mouse and various buttons to transfer files back and forth between machines.

**Figure 6.8. Many operating systems have a GUI-based FTP client.**

## FTP Third-Party Transfers

FTP enables a transfer to occur through a third machine positioned between the client and the server. This procedure is known as a *third-party transfer* and is sometimes necessary to obtain proper permissions to access the remote machine. Figure 6.9 shows the schematic of a third-party transfer, with the control connection made through a third machine.

Figure 6.9. A third-party FTP transfer.

When setting up a third-party connection, the client opens the control connections between the remote machine and the second client that handles the control channel. Only the control channel goes through the second client, whereas the data channel goes directly between the two ends.

When a transfer request is submitted, it is transferred through the second client, which checks permissions and then forwards the request to the server. The data transfer can take place directly, because the permissions were checked on the control channel.

## Anonymous FTP Access

FTP requires a user ID and password to enable file transfer capabilities, but there is a more liberal method of enabling general access to a file or directory, called *anonymous FTP*. Anonymous FTP removes the requirement for a login account on the remote machine, usually enabling the login anonymous with a password of either guest or the user's actual login name. The following session shows the use of an anonymous FTP system:

```
tpci_hpws4-1> ftp uofo.edu
Connected to uofo.edu.
220 uofo.edu FTP server (Version 1.7.109.2 Tue Jul 26 23:32:34 GMT 1992) ready.
Name (uofo:username): anonymous
331 Guest login ok, send userID as password.
Password: tparker
230 Guest login ok, access restrictions apply.
ftp> <Ctrl+d>
tpci_hpws4-2>
```

If the remote system is set to enable anonymous logins, you are prompted for a password and then given a warning about access limitations. If there is a file on the remote system you require, a get command transfers it. Anonymous FTP sites are becoming common, especially with the popularity of the Internet.

## FTP Servers

Most UNIX machines act as FTP servers by default. To provide FTP server facilities, they run the daemon ftpd when the operating system is booted. The daemon is usually handled by the UNIX inetd process. When you start using inetd, the inetd daemon watches the TCP command port (channel 21) for an arriving request for a connection, then starts ftpd to service that request. If you want to ensure that your UNIX or Linux system can handle FTP requests, make sure the ftpd daemon can be started when needed by inetd by checking the inetd configuration file (usually /etc/inetd.config or /etc/inetd.conf) for a line that looks like this:

```
ftp stream  tcp  nowait  root  /usr/etc/ftpd   ftpd -l
```

If this line doesn't exist, you should add it. With most UNIX systems this line is already in the inetd configuration file, although it might be commented out, in which case you should remove the comment symbol.

Windows, Windows for Workgroups, and Windows 95 all lack an FTP server program as part of their distribution software (although Windows 95 does have an FTP client), so you have to add a commercial package. Many commercial TCP/IP suites include an FTP server process. Figure 6.10 shows the NetManage ChameleonNFS program group, which includes an FTP server program you can use as an example for Windows for Workgroups and Windows 3.*x* machines.

**Figure 6.10. The NetManage FTP Server dialog handles the FTP server process.**

To start the NetManage FTP Server software, double-click the FTP server icon in the NetManage program group. A dialog, shown in Figure 6.10, appears. The FTP server process is now active, and anyone on another machine on your local area network can now connect to your machine, assuming they have access.

Access to your FTP service is controlled through the user lists maintained by the FTP Server package. Selecting the Users menu option from the NetManage FTP Server dialog opens the Users dialog, shown in Figure 6.11. This lets you add user names to your system. If another user on a different machine tries to connect to your FTP server software, the server verifies that their login name and password match the name and password you enter in this dialog. This lets you set up a list of users who can transfer files to and from your system, as long as the FTP server is running.

**Figure 6.11. Access to your machine is controlled through the FTP Server Users dialog.**

If you are running an FTP server process, it is often a good idea to create a directory just for FTP. Many users prefer to create a directory called public, which is where all files to be transferred in and out of the local system are placed. This lets you prevent accidental deletion or transfer of files in other directories on your system, as well as providing you with the opportunity to filter incoming material for suitability, viruses, and so on. If you use a transfer directory, check it regularly and make sure all users who have access to your system can work only in that directory.

If you want to provide an anonymous or guest account for users on your LAN or any other network that can connect to your machine, you should set up an account with either no password or a simple password like guest. It is very important to restrict the areas a guest or anonymous login can use.

As mentioned earlier, Windows 95 is supplied with client FTP software but not an FTP server. You can

use other aspects of Windows 95 as a file transfer system, such as file and print sharing over any existing network, but these do not use FTP. If you want to set up an FTP server on your Windows 95 machine, you have to install third-party commercial software for this purpose.

A popular Windows 95 FTP server package called FTP Serv-U was written by Rob Beckers and is provided as shareware. An executable file called Serv-U starts the server. To control access to your Windows 95 system, you can set up logins using the Serv-U Users menu option. This displays a screen that lets you add logins and passwords, as well as the directories and drives the user has access to. Figure 6.12 shows the Edit User/Group dialog with a user being added. When a user from another system logs into your Windows 95 machine, they are asked for a login and password.

Figure 6.12. Set up all users of your FTP server with the Edit User/Group dialog.

## Trivial File Transfer Protocol (TFTP)

The Trivial File Transfer Protocol (TFTP) is one of the simplest file transfer protocols in use. It differs from FTP in two primary ways: it does not log onto the remote machine, and it uses the User Datagram Protocol (UDP) connectionless transport protocol instead of TCP. By using UDP, TFTP does not monitor the progress of the file transfer, although it does have to employ more complex algorithms to ensure proper data integrity. By avoiding logging onto the remote, user access and file permission problems are avoided. TFTP uses the TCP port identifier number 69, even though TCP is not involved in the protocol.

TFTP has few advantages over FTP. It is not usually used for file transfers between machines where FTP could be used instead, although TFTP is useful when a diskless terminal or workstation is involved. Typically, TFTP is used to load applications and fonts into these machines, as well as for bootstrapping. TFTP is necessary in these cases because the diskless machines cannot execute FTP until they are fully loaded with an operating system. TFTP's small executable size and memory requirements make it ideal for inclusion in a bootstrap, where the system requires only TFTP, UDP, and a network driver, all of which can be provided in a small EPROM.

TFTP handles access and file permissions by imposing restraints of its own. On most UNIX systems, for example, a file can be transferred only if it is accessible to all users on the remote (both read and write permissions are set). Because of the lax access regulations, most system administrators impose more control over TFTP (or ban its use altogether); otherwise, it is quite easy for a knowledgeable user to access or transfer files that could constitute a security violation.

TFTP transfers can fail for many reasons, because practically any kind of error encountered during a transfer operation causes a complete failure. TFTP does support some basic error messages, but it cannot handle simple errors such as insufficient resources for a file transfer or even a failure to locate a requested file.

### TFTP Commands

The important instructions in TFTP's command set are shown in Table 6.7. The TFTP command set is similar to FTP's, but it differs in several important aspects because of the connectionless aspect of the

protocol. Most noticeable is the connect command, which simply determines the remote's address instead of initiating a connection.

**Table 6.7. TFTP's command set.**

| *TFTP Command* | *Description* |
|---|---|
| binary | Use binary mode for transfers |
| connect | Determine the remote's address |
| get | Retrieve a file from the remote |
| put | Transfer a file to the remote |
| trace | Display protocol codes |
| verbose | Display all information |

TFTP enables both text and binary transfers, as does FTP. As with both Telnet and FTP, TFTP uses a server process (tftpd on a UNIX system) and an executable, usually called tftp. A sample TFTP session on a UNIX host is shown here, with full trace options and binary transfers turned on:

```
tpci_hpws1-1> tftp

tftp> connect tpci_hpws4

tftp> trace

Packet tracing on.

tftp> binary

Binary mode on.

tftp> verbose

Verbose mode on.

tftp> status

Connected to tpci_hpws4.

Mode: octet Verbose: on Tracing: on

Rexmt-interval: 5 seconds, Max-timeout: 25 seconds

tftp> get /usr/rmaclean/docs/draft1

getting from tpci_hpws4:/usr/rmaclean/docs/draft1 to /tmp/draft1 [octet]

sent RRQ <file=/usr/rmaclean/docs/draft1, mode=octet>

received DATA <block1, 512 bytes>

send ACK <block=1>

received DATA <block2, 512 bytes>
```

```
send ACK <block=3>

received DATA <block=4, 128 bytes>

send ACK <block=3>

Received 1152 bytes in 0.2 second 46080 bits/s]

tftp> quit

tpci_hpws1-2>
```

In the session above, you can see that the trace and verbose commands turn on the echoing of the instructions flowing between the two machines during a file transfer. Every time a block of data is sent after the get command is issued (the send ACK instruction shown on the session above), a received instruction is returned to acknowledge the ACK.

TFTP is available on all UNIX systems as well as in TCP/IP suites for other operating systems. Figure 6.13 shows the TFTP utility from ChameleonNFS, which lets you enter the remote host name, the remote and local filenames, and the type of transfer you want. The file transfer is then performed in the background using UDP.

Figure 6.13. Using TFTP to transfer a file.

## TFTP Packets

TFTP uses UDP as a transport protocol, so TFTP can use the UDP header to encapsulate TFTP protocol information. TFTP uses the UDP source and destination port fields to set the two ends of the connection. It accomplishes this by the use of *TFTP Transfer Identifiers,* or TIDs, which are created by TFTP and passed to UDP, which then places them in the headers.

As with Telnet and FTP, TFTP uses port binding, where the sending machine selects a TID, and the remote is set to port number 69 (TFTP's port number). The remote machine responds with an acknowledgment of a connection request, a source port of 69, and the destination TID sent in the request.

TFTP uses five types of Protocol Data Units, which are referred to as packets in the TFTP lexicon. These packets are listed in Table 6.8. Their layout is shown in Figure 6.14. Error messages supported by TFTP are shown in Table 6.9.

Figure 6.14. TFTP packet layouts.

**Table 6.8. TFTP Protocol Data Unit codes.**

| *Code* | *OpCode* | *Description* |
|---|---|---|
| ACK | 4 | Acknowledgment |
| DATA | 3 | Send Data |
| Error | 5 | Error |
| RRQ | 1 | Read request |
| WRQ | 2 | Write request |

**Table 6.9. TFTP error messages and codes.**

| *Code* | *Description* |
|---|---|
| 0 | Not defined |
| 1 | File not found |
| 2 | Permissions prevent access |
| 3 | Disk full or allocation limit exceeded |
| 4 | Illegal TFTP operation requested |
| 5 | Unknown transfer number |

The layouts for both RRQ and WRQ packets have a Mode field, which indicates the type of transfer. There are three modes currently available to TFTP:

- **NetASCII:** Standard ASCII codes.
- **Byte:** 8-bit bytes and binary information.
- **Mail:** Indicates that the destination is a user, not a file (information is transferred as NetASCII).

The last block in all packets contains between 0 and 511 bytes of data, labeled 0 in Figure 6.14. This pads out the block of data to 512 bytes.

The communications process used by TFTP begins with the client sending an RRQ or WRQ request to the server through UDP. As part of the request, a transaction number, the filename, and a code to identify the transmission mode to be used are specified. The transaction number is used to identify future transactions in the sequence.

Because there is no connection between the two, the client sets a timer and waits for a reply from the server. If one doesn't arrive before the timer expires, another request is sent. After an ACK is received, a DATA packet is transmitted, for which another ACK or an ERROR is received. If there are several packets to be transferred, they are constructed so they have a length of 512 bytes and an incrementing sequence number. The process terminates when a DATA packet with a length of less than 512 bytes is received by the server. For each packet sent, TFTP waits for an acknowledgment before sending the next,

a system known as a *flip-flop protocol.*

## Simple Mail Transfer Protocol (SMTP)

The Simple Mail Transfer Protocol (SMTP) is the defined Internet method for transferring electronic mail. SMTP is similar to FTP in many ways, including the same simplicity of operation. SMTP uses TCP port number 25.

Most UNIX systems use programs called sendmail or mmdf to implement SMTP (as well as several other mail protocols). The sendmail program, for example, acts as both a client and a server, usually running in the background as a daemon. Users do not interact with sendmail directly but use a front-end mail program such as mail, mailx, or Mail. These mail system interfaces pass the message to sendmail for forwarding.

SMTP uses spools or queues. When a message is sent to SMTP, it places it in a queue. SMTP attempts to forward the message from the queue whenever it connects to remote machines. If it cannot forward the message within a specified time limit, the message is returned to the sender or removed.

### SMTP Commands

SMTP data transmissions use a simple format. All the message text is transferred as 7-bit ASCII characters. The end of the message is indicated by a single period on a line by itself. If for some reason a line in the message begins with a period, a second one is added by the protocol to avoid confusion with the end-of-message indicator.

SMTP has a simple protocol command set, listed in Table 6.10. Using these protocol elements, mail is transferred with a minimum of effort.

**Table 6.10. The SMTP protocol command set.**

| *Command* | *Description* |
|---|---|
| DATA | Message text |
| EXPN | Expansion of a distribution list |
| HELO | Use in connection establishment to exchange identifiers |
| HELP | Request for help |
| MAIL | The sender's address |
| NOOP | No operation |
| RCPT | The message destination address (more than one can be provided) |
| RSET | Terminate the current transaction |
| SAML | Send a message to the user's terminal and send mail |
| SEND | Send a message to the user's terminal |
| SOML | Either send a message to the user's terminal or send mail |
| TURN | Change the sending direction (reverse sending and receiving roles) |
| VRFY | Verify the user name |

When a connection is established, the two SMTP systems exchange authentication codes. Following this, one system sends a MAIL command to the other to identify the sender and provide information about the message. The receiving SMTP returns an acknowledgment, after which a RCPT is sent to identify the recipient. If more than one recipient at the receiver location is identified, several RCPT messages are sent, but the message itself is transmitted only once. After each RCPT there is an acknowledgment. A DATA command is followed by the message lines, until a single period on a line by itself indicates the end of the message. The connection is closed with a QUIT command.

The sender and recipient address fields use standard Internet formats, involving the user name and domain name (such as tparker@tpci.com). The domain can be replaced by other information if a direct connection is established, or if there is a forwarding machine in the path. SMTP uses the Domain Name System (DNS) for all addresses.

## The Berkeley Utilities

The University of California at Berkeley was instrumental in the development of TCP/IP and contributed many utility programs to the application tool set. These are usually known by the term *Berkeley r-Utilities.* They are called r-utilities because they all start with the letter r (for remote). Most of the utilities are UNIX-specific, although they have since all been ported to other operating systems.

### The *hosts.equiv* and *.rhosts* Files

To enable machines to communicate correctly over networks, access rights for machines and users must

be set. Usually, when logging into another machine, a user must supply a user ID and a password. When you log into many machines, retyping this information can be tedious and time-consuming. It can also be a security problem, because it is easy to write a program that monitors network connections for this information. A way to enable fast access without actually logging in and preventing interception of passwords is clearly useful in some cases.

The system administrator can decide that all login names used on other machines whose names are in the file hosts.equiv are allowed access on the local machine. This enables a protocol that queries a machine for access to check the hosts.equiv file for the requesting machine's name, and if it is found, to grant access to the user on the remote machine. The user has the same access rights as on his or her home machine.

If the protocol doesn't find an entry in the hosts.equiv file, it can check another file maintained in a user's home directory, called .rhosts. A user can control who has access to their login name with the file .rhosts in their home directory, enabling other users to log in as if they were that user. The .rhosts file must be owned by the user (or root) and not allow write access to all users (on a UNIX system, the other permission cannot be write). An .rhosts file consists of a line for each user to be allowed into the home directory. The line consists of a machine name and a login name. A sample .rhosts file is shown here:

```
tpci_hpws1 rmaclean
tpci_hpws1 bsmallwood
tpci_hpws3 ychow
tpci_hpws3 bsmallwood
tpci_hpws4 glessard
tpci_hpws4 bsmallwood
tpci_sunws1 chatton
merlin tparker
merlin ahoyt
merlin lrainsford
```

This file allows user bsmallwood to log in from three different machines.

### *rlogin*

The rlogin command (for *remote login*) enables a user to log into another machine. It is very similar in functionality to Telnet, although the protocol is much simpler. There is a background program running on the server called rlogind, and the program rlogin resides on the client.

The rlogin protocol begins a session by sending three character strings, separated by 0s. The first string is the user's login ID (on the client), the second string is the login name for the server (usually but not always the same as the login name on the client), and the third string is the login name and transmission rate of the user's terminal (such as wyse60/19200). When received on the server, the strings can be

converted to environment variables (such as UNIX's TERM terminal variable). You cannot log into the remote machine with a different user ID, because the system does not prompt for the login name. It does prompt for a password, however.

After the login process is completed, rlogin doesn't use any protocol. Every character you type on the client machine is sent to the server, whereas every server-generated character is displayed on your console. The only exit to your local machine is by closing the connection by using Ctrl+D or entering the escape character on a line by itself. By default, the escape character is a tilde (~).

Some versions of rlogin enable a shell escape, a temporary suspension of the rlogin session and a return to the operating system, by using ~!.

### *rsh*

The rsh utility (remote shell) lets you execute commands on a remote machine. As with most Berkeley utilities, a background process called rshd is involved. Executing a command on a remote machine is a matter of adding rsh and the machine name to the front of the command line, such as rsh tpci_hpws3 who or rsh tpci_sunws1 tar xvf /dev/rct0 (using UNIX examples). The rsh utility depends on the presence of either hosts.equiv or .rhosts to enable login; otherwise, access is not granted.

The rsh utility is not a shell in the sense that it does not interpret commands like the UNIX C shell or Bourne shell. Instead, a command entered is sent to the server's standard input and output, executing the command as a local process through the TCP connection. The primary advantage of this is that a shell script that executes on your local machine can be submitted to the remote machine with no modification, where it runs just as if it were local (except using the remote's file system). Unfortunately, any return codes generated by the remote system are not sent back to your local machine. Also, most screen-oriented applications do not function properly, because they have no terminal output to write to.

### *rcp*

The Berkeley rcp (remote copy) command is similar to the UNIX cp command, except that it works across the network. The command syntax and option lists for rcp are the same as cp, although a machine name is usually specified as part of the filename by the addition of the machine name followed by a colon (see the following examples). Even recursive copying of directories is supported (a useful and attractive feature of rcp that isn't available under FTP or TFTP). The rcp program acts as both server and client, initiated when a request arrives.

```
rcp tpci_hpws4:/user/tparker/doc/draft1 .

rcp file2 merlin:/u1/bsmallwood/temp/file2

rcp -r merlin:/u2/tparker/tcp_book tpci_server/tcp_book
```

```
rcp merlin:/u1/ychow/iso9000_doc tpci_server:/u1/iso/doc1/iso_doc_from_ychow

rcp file4 tparker@tpci.com:new_info
```

As the examples indicate, the filenames at both the local and remote machines are specified, with standard UNIX conventions supported. The third example shows a file being transferred from one machine to another, neither of which is the machine from which the command is initiated. The last example shows the use of a full DNS-style name for the destination address.

The rcp utility is a faster method of transferring files than FTP, although rcp requires access permission through an .rhosts file (not hosts.equiv). Without an entry in this file, access is refused and FTP or TFTP must be used.

### *rwho*

The rwho (remote who) command uses the rwhod daemon to display a list of users on the network. It shows all network users, compiled from a regularly sent packet of information from all running rwhod programs. The frequency of this packet broadcast is system-dependent but is usually in the order of every one to three minutes. When an rwhod program receives a broadcast from another machine, it places it in a system file for future use. (The file on a UNIX system is called /usr/spool/rwho.)

When a machine has not sent a broadcast message within a time limit (usually eleven minutes), it is assumed that the machine has disconnected from the network, and all users listed as active on that machine in the system file are ignored. The rwhod program drops a user ID from its broadcast if nothing has been entered at the user's terminal in the last hour

The output from an rwho request is shown in the following example. For each user, it shows their login name, their machine and terminal name, and the time and date of their login.

```
bsmallwood merlin:tty2p      Feb 29 09:01
etreijs    tpci_hpws2:tty01  Feb 29 12:12
rmaclean   goofus:tty02      Feb 29 25:52
tparker    merlin:tty01      Feb 29 11:43
ychow      prudie:tty2a      Feb 28 11:57
```

The rcp program has one major problem on large networks: the continuous sending of update packets by each machine creates a considerable amount of network traffic. For this reason, some implementations directly request the user names only when an rwho request is received.

### *ruptime*

The ruptime utility displays a list of all machines on the network, their status, the number of active users, current load, and elapsed time since the system was booted. The program uses the same information as

the rwho command.

A sample output from a ruptime command follows:

```
merlin      up    3:15,12 users, load 0.90, 0.50, 0.09
prudie      down  9:12
tpci_hpws1 up    11:05, 3 users, load 0.10, 0.10, 0.00
tpci_hpws2 up    23:59, 5 users, load 0.30, 0.25, 0.08
tpci_hpws3 down  6:45
tpci_hpws4 up    9:05, 1 user,  load 0.12, 0.05, 0.01
```

### *rexec*

The rexec (remote execution) program is a holdover from earlier versions of the UNIX operating system. It was designed to enable remote execution of a command through a server process called rexecd. The utility uses the dedicated TCP port number 512.

The protocol used by rexec is very similar to rsh, except that an encrypted password is sent with the request and there is a full login process. The rexec utility is seldom used because rsh is a faster and more convenient method for executing a command remotely.

## Summary

Today I looked at the primary application protocols that use TCP/IP, as well as the Berkeley utilities. Now that you can see how protocols work on top of the TCP and IP protocols, the layered structure of TCP/IP becomes more pronounced. Future days' texts build on this information.

## Q&A

**What is the purpose of Telnet and FTP?**

Telnet provides a remote login capability, whereas FTP enables you to transfer files across the network.

**What channels (port numbers) are used by Telnet, FTP, and SMTP?**

Telnet uses port number 23. FTP uses port number 21 for the control information and port number 20 for data. SMTP uses port number 25.

**When you issue a get command in FTP, is it moving a file from the local to remote, or vice versa?**

FTP commands are relative to the remote, so a get command moves a file from the local to the remote.

**How does TFTP differ from FTP?**

TFTP does not require logging in. It sends a request over UDP. With FTP, you must log into the destination either directly or through a third-party device.

**Does rlogin differ from telnet?**

The rlogin program was developed earlier and for most users has no difference. There are some version dependencies with some releases of rlogin, reflecting its earlier (and less full-featured) origins.

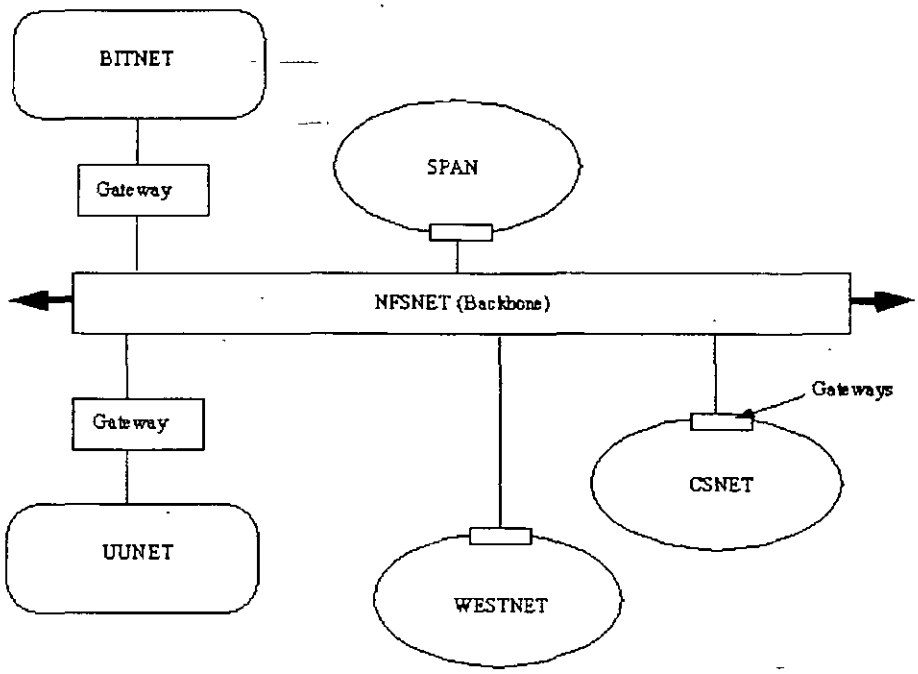
## Quiz

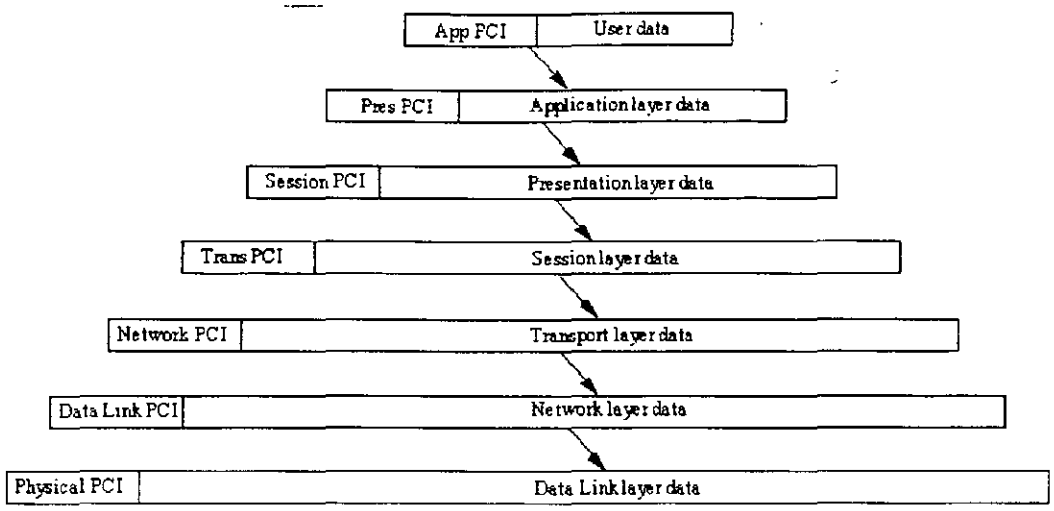
1. Explain what a network virtual terminal is.
2. Draw diagrams showing two- and three-party FTP sessions, indicating the port numbers used by each machine.
3. Why would you want to enable anonymous FTP access? Are there any reasons for disallowing it?
4. TFTP enables files to be transferred without logging in. What problems can this cause?
5. What are the Berkeley Utilities?

## Workshop

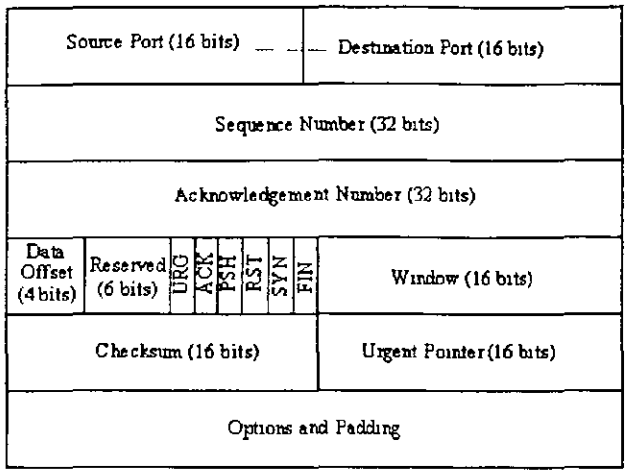
If you have access to a Telnet or FTP session, try logging into a remote machine and transferring files back and forth. Try to recognize that Telnet does everything relative to the local machine, whereas FTP is relative to the remote.

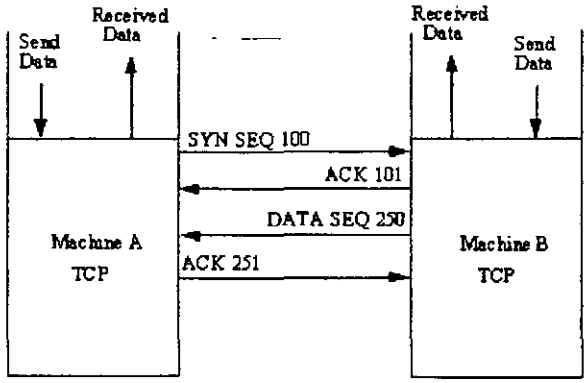
Us Internet Network




App PCI
User data
Pres PCI
Application layer data
Session PCI
Presentation layer data
Trans PCI
Session layer data
Network PCI
Transport layer data
Data Link PCI
Network layer data
Physical PCI
Data Link layer data

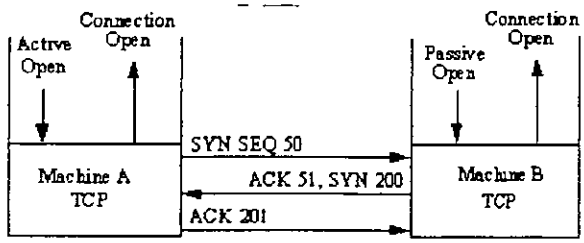
Protocol Data Unit PDU
(Header)

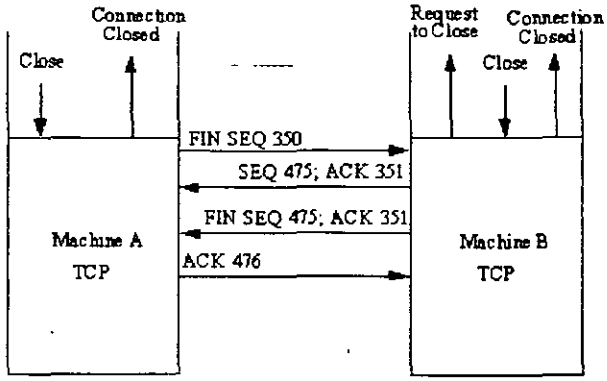





TCP
Send Data

Establishing a TCP connection




Connection Closed
Close
Request to Close
Connection Closed
Close
FIN SEQ 350
SEQ 475; ACK 351
FIN SEQ 475; ACK 351
ACK 476
Machine A
TCP
Machine B
TCP

TCP close

| Type | Code | Checksum |
|---|---|---|
| Unused | | |
| Original IP header + 64 bits | | |

Destination unreachable, Source Quench, Time Exceeded

| Type | Code | Checksum |
|---|---|---|
| Identifier | | Sequence No. |
| Originating Timestamp | | |

Timestamp Request

| Type | Code | Checksum |
|---|---|---|
| Ptr | Unused | |
| Original IP header + 64 bits | | |

Parameter Problem

| Type | Code | Checksum |
|---|---|---|
| Identifier | | Sequence No. |
| Originating Timestamp | | |
| Receiving Timestamp | | |
| Transmitting Timestamp | | |

Timestamp Reply

| Type | Code | Checksum |
|---|---|---|
| Gatewat IP Address | | |
| Original IP header + 64 bits | | |

Redirect

| Type | Code | Checksum |
|---|---|---|
| Identifier | | Sequence No. |

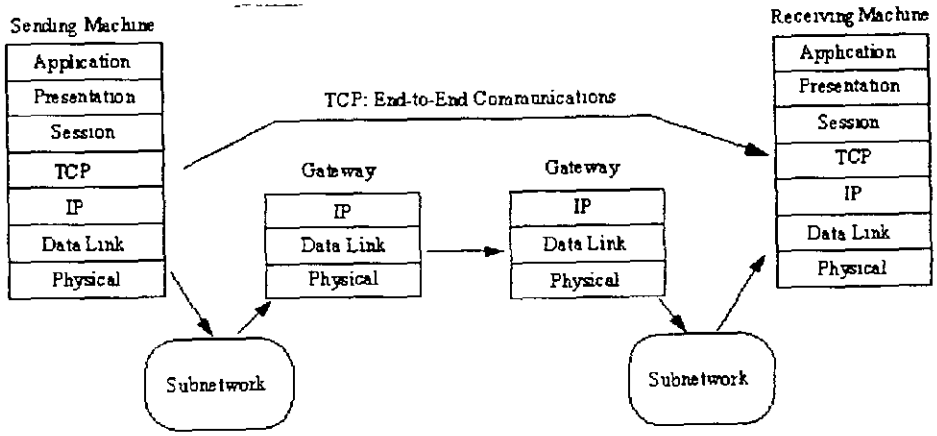
Information Request and Reply, Address Mask Request

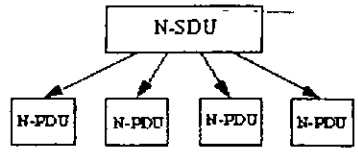
| Type | Code | Checksum |
|---|---|---|
| Identifier | | Sequence No |
| Original IP header + 64 bits | | |

Echo Request and Echo Reply

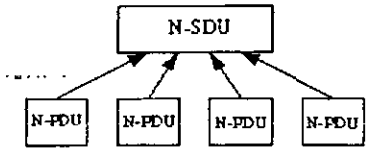
| Type | Code | Checksum |
|---|---|---|
| Identifier | | Sequence No |
| Address Mask | | |

Address Mask Reply

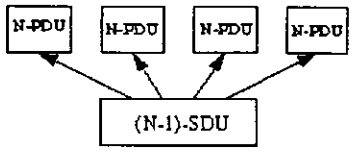

Sending Machine
Application
Presentation
Session
TCP
IP
Data Link
Physical
TCP: End-to-End Communications
Gateway
IP
Data Link
Physical
Gateway
IP
Data Link
Physical
Receiving Machine
Application
Presentation
Session
TCP
IP
Data Link
Physical
Subnetwork
Subnetwork



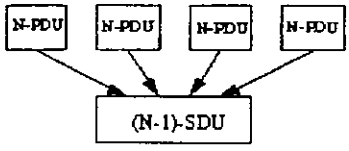
Segmentation



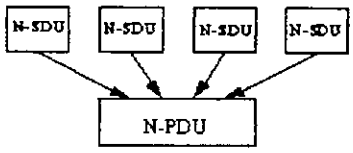
Reassembly



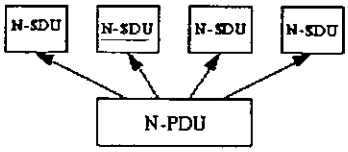
Separation



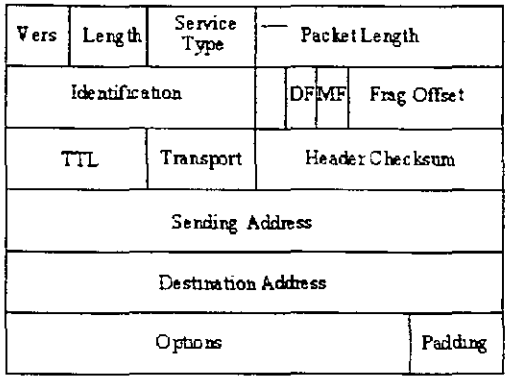
Concatenation



Blocking



Unblocking


Vers
Length
Service Type
Packet Length
Identification
DF
MF
Frag Offset
TTL
Transport
Header Checksum
Sending Address
Destination Address
Options
Padding

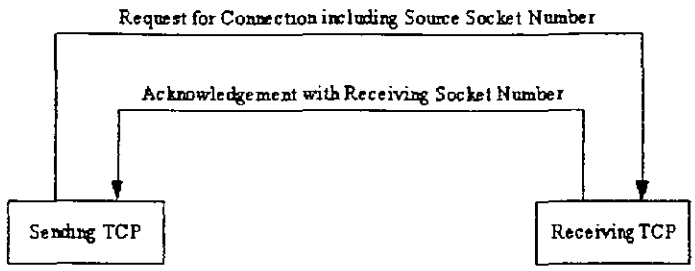
File Edit Format Special View Page Graphics Table Window Help

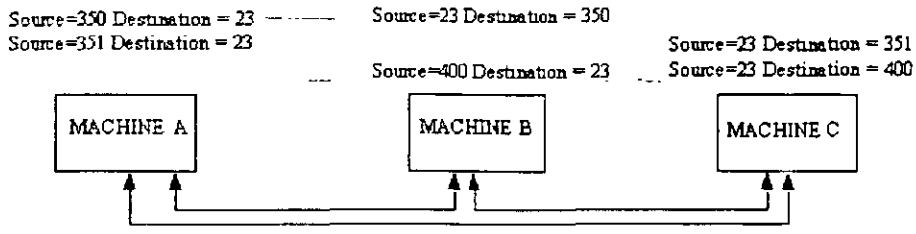
| Precedence (3 bits) | Delay | Thru | Rel | Not Used |
|---|---|---|---|---|

Figure 3-2: The 8-bit Service Type Field Layout

Flow: A ¶: Body 1 of 1

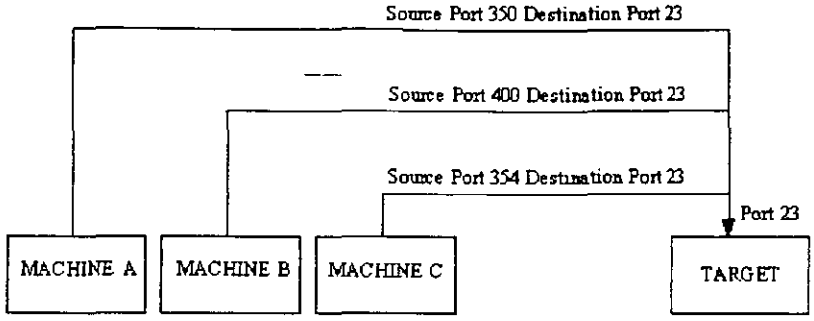
setup
virtual Circuit

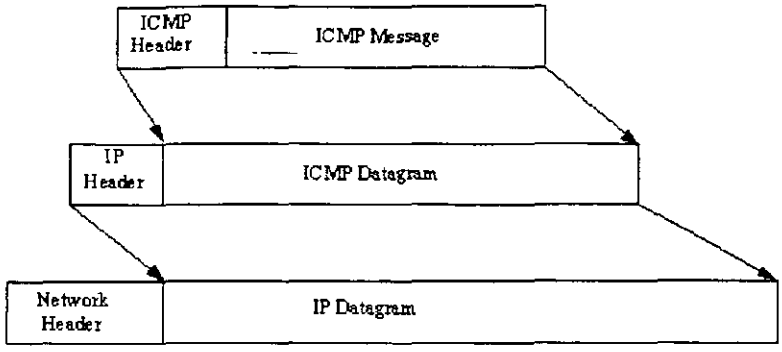


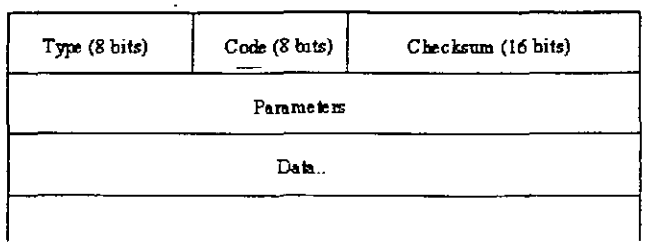

Source=350 Destination = 23
Source=351 Destination = 23
Source=23 Destination = 350
Source=400 Destination = 23
Source=23 Destination = 351
Source=23 Destination = 400
MACHINE A
MACHINE B
MACHINE C

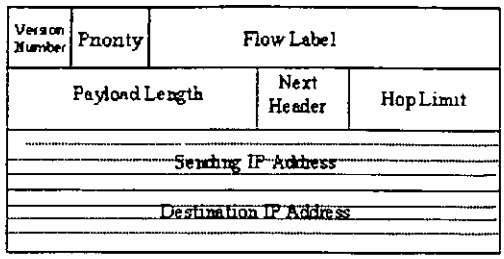
Binding

Multiplexing Ports




ICMP Header
ICMP Message
IP Header
ICMP Datagram
Network Header
IP Datagram


Type (8 bits)
Code (8 bits)
Checksum (16 bits)
Parameters
Data...

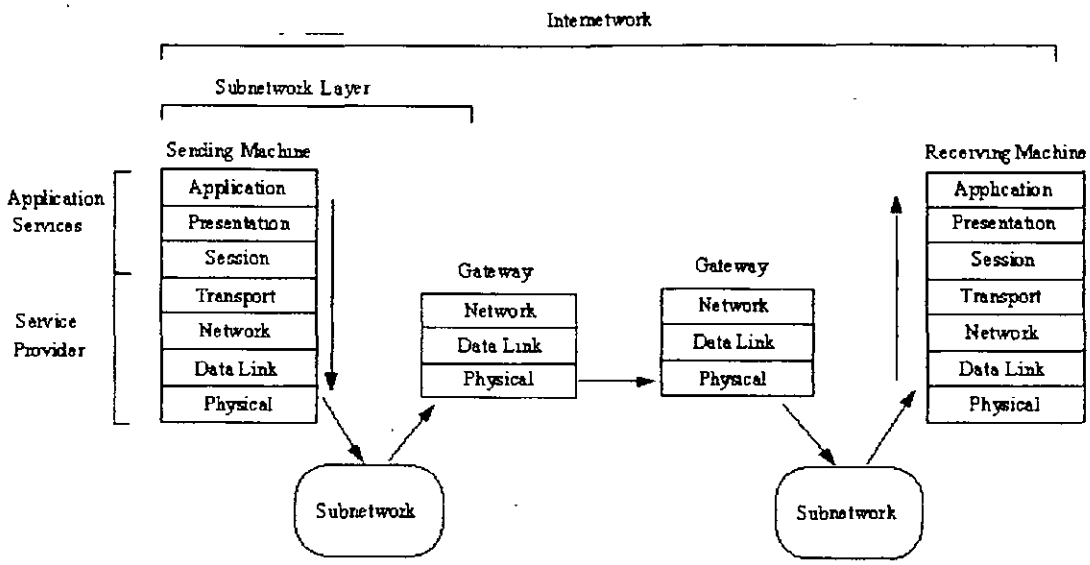

Version Number
Priority
Flow Label
Payload Length
Next Header
Hop Limit
Sending IP Address
Destination IP Address

TCP connection Table

| | STATE | LOCAL ADDRESS | LOCAL PORT | REMOTE ADDRESS | REMOTE PORT |
|---|---|---|---|---|---|
| Connection 1 | | | | | |
| Connection 2 | | | | | |
| Connection 3 | | | | | |
| Connection n | | | | | |

| Source Port (16 bits) | Destination Port (16 bits) |
|---|---|
| Length (16 bits) | Checksum (16 bits) |
| Data.. | |

UDP Headder


Internetwork
Subnetwork Layer
Sending Machine
Application Services
Service Provider
Application
Presentation
Session
Transport
Network
Data Link
Physical
Gateway
Network
Data Link
Physical
Gateway
Network
Data Link
Physical
Receiving Machine
Application
Presentation
Session
Transport
Network
Data Link
Physical
Subnetwork
Subnetwork

**Transmission Control Blocks and Flow Control**

TCP has to keep track of a lot of information about each connection. It does this through a Transmission Control Block (TCB), which contains information about the local and remote socket numbers, the send and receive buffers, security and priority values, and the current segment in the queue. The TCB also manages send and receive sequence numbers.

The TCB uses several variables to keep track of the send and receive status and to control the flow of information. These variables are shown in Table 4.3.

Table 4.3. TCP send and receive variables

| Variable Name | Description |
|---|---|
| Send Variables | |
| SND.UNA | Send Unacknowledged |
| SND.NXT | Send Next |
| SND.WND | Send Window |
| SND.UP | Sequence number of last urgent set |
| SND.WL1 | Sequence number for last window update |
| SND.WL2 | Acknowledgment number for last window update |
| SND.PUSH | Sequence number of last pushed set |
| ISS | Initial send sequence number |
| Receive Variables | |
| RCV.NXT | Sequence number of next received set |
| RCV.WND | Number of sets that can be received |
| RCV.UP | Sequence number of last urgent data |
| RCV.IRS | Initial receive sequence number |

**PDU**

Source port: A 16-bit field that identifies the local TCP user (usually an upper-layer application program).

Destination port: A 16-bit field that identifies the remote machine's TCP user.

Sequence number: A number indicating the current block's position in the overall message. This number is also used between two TCP implementations to provide the initial send sequence (ISS) number.

Acknowledgment number: A number that indicates the next sequence number expected. In a backhanded manner, this also shows the sequence number of the last data received; it shows the last sequence number received plus 1.

Data offset: The number of 32-bit words that are in the TCP header. This field is used to identify the start of the data field.

Reserved: A 6-bit field reserved for future use. The six bits must be set to 0.

Urg flag: If on (a value of 1), indicates that the urgent pointer field is significant

Ack flag: If on, indicates that the Acknowledgment field is significant

Psh flag: If on, indicates that the push function is to be performed.

Rst flag: If on, indicates that the connection is to be reset.

Syn flag: If on, indicates that the sequence numbers are to be synchronized. This flag is used when a connection is being established.

Fin flag: If on, indicates that the sender has no more data to send. This is the equivalent of an end-of-transmission marker.

Window: A number indicating how many blocks of data the receiving machine can accept.

Checksum: Calculated by taking the 16-bit one's complement of the one's complement sum of the 16-bit words in the header (including pseudo-header) and text together. (A rather lengthy process required to fit the checksum properly into the header.)

Urgent pointer. Used if the urg flag was set; it indicates the portion of the data message that is urgent by specifying the offset from the sequence number in the header. No specific action is taken by TCP with respect to urgent data; the action is determined by the application.

| | |
|---|---|
| ALLOCATE | Local connection name, data length |
| CLOSE | Local connection name |
| FULL-PASSIVE-OPEN | Local port, destination socket |
| | Optional: ULP timeout, timeout action, precedence, security, options |
| RECEIVE | Local connection name, buffer address, byte count, push flag, urgent flag |
| SEND | Local connection name, buffer address, data length, push flag, urgent flag |
| | Optional: ULP timeout, timeout action |
| STATUS | Local connection name |
| UNSPECIFIED-PASSIVE-OPEN | Local port |
| | Optional: ULP timeout, timeout action, precedence, security, options |
| TCP to ULP Service Request Primitives | |
| CLOSING | Local connection name |
| DELIVER | Local connection name, buffer address, data length, urgent flag |
| ERROR | Local connection name, error description |
| OPEN-FAILURE | Local connection name |
| OPEN-ID | Local connection name, remote socket, destination address |
| OPEN-SUCCESS | Local connection name |
| STATUS RESPONSE | Local connection name, source port, source address, remote socket, connection state, receive window, send window, amount waiting ACK, amount waiting receipt, urgent mode, precedence, security, timeout, timeout action |
| TERMINATE | Local connection name, description |

**Passive and Active Ports**

**The Retransmission Timer**

The retransmission timer manages retransmission timeouts (RTOs), which occur when a preset interval between the sending of a datagram and the returning acknowledgment is exceeded. The value of the timeout tends to vary, depending on the network type, to compensate for speed differences. If the timer expires, the datagram is retransmitted with an adjusted RTO, which is usually increased exponentially to a maximum preset limit. If the maximum limit is exceeded, connection failure is assumed, and error messages are passed back to the upper-layer application.

Values for the timeout are determined by measuring the average time that data takes to be transmitted to another machine and the acknowledgment received back, which is called the round-trip time, or RTT. From experiments, these RTTs are averaged by a formula that develops an expected value, called the smoothed round-trip time, or SRTT. This value is then increased to account for unforeseen delays.

**The Quiet Timer**

After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port. The quiet timer is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams.

The quiet timer is usually set to twice the maximum segment lifetime (the same value as the Time to Live field in an IP header), ensuring that all segments still heading for the port have been discarded. Typically, this can result in a port being unavailable for up to 30 seconds, prompting error messages when other applications attempt to access the port during this interval.

**The Persistence Timer**

The persistence timer handles a fairly rare occurrence. It is conceivable that a receive window might have a value of 0, causing the sending machine to pause transmission. The message to restart sending might be lost, causing an infinite delay. The persistence timer waits a preset time and then sends a one-byte segment at predetermined intervals to ensure that the receiving machine is still clogged.

The receiving machine resends the zero window-size message after receiving one of these status segments, if it is still backlogged. If the window is open, a message giving the new value is returned, and communications are resumed.

**The Keep-Alive Timer and the Idle Timer**

Both the keep-alive timer and the idle timer were added to the TCP specifications after their original definition. The keep-alive timer sends an empty packet at regular intervals to ensure that the connection to the other machine is still active. If no response has been received after sending the message by the time the idle timer has expired, the connection is assumed to be broken.

The keep-alive timer value is usually set by an application, with values ranging from 5 to 45 seconds. The idle timer is usually set to 360 seconds

Options: Similar to the IP header option field, this is used for specifying TCP options. Each option consists of an option number (one byte), the number of bytes in the option, and the option values. Only three options are currently defined for TCP:

0 End of option list

1 No operation

2 Maximum segment size

Padding: Filled to ensure that the header is a 32-bit multiple.

The TCP data transport service actually embodies six **subservices:**

Full duplex: Enables both ends of a connection to transmit at any time, even simultaneously.

Timeliness: The use of timers ensures that data is transmitted within a reasonable amount of time.

Ordered: Data sent from one application is received in the same order at the other end. This occurs despite the fact that the datagrams might be received out of order through IP, because TCP reassembles the message in the correct order before passing it up to the higher layers

Labeled: All connections have an agreed-upon precedence and security value.

Controlled flow: TCP can regulate the flow of information through the use of buffers and window limits.

Error correction: Checksums ensure that data is free of errors (within the checksum algorithm's limits).

Source port: An optional field with the port number. If a port number is not specified, the field is set to 0.

Destination port: The port on the destination machine.

Length: The length of the datagram, including header and data.

Checksum: A 16-bit one's complement of the one's complement sum of the datagram, including a pseudoheader similar to that of TCP.

The UDP checksum field is optional, but if it isn't used, no checksum is applied to the data segment because IP's checksum applies only to the IP header. If the checksum is not used, the field should be set to 0.

**TCP**

TCP receives the stream of bytes and assembles them into TCP segments, or packets. In the process of assembling the segment, header information is attached at the front of the data. Each segment has a checksum calculated and embedded within the header, as well as a sequence number if there is more than one segment in the entire message. The length of the segment is usually determined by TCP or by a system value set by the system administrator. (The length of TCP segments has nothing to do with the IP datagram length, although there is sometimes a relationship between the two.)

If two-way communications are required (such as with Telnet or FTP), a connection (virtual circuit) between the sending and receiving machines is established prior to passing the segment to IP for routing. This process starts with the sending TCP software issuing a request for a TCP connection with the receiving machine. In the message is a unique number (called a socket number) that identifies the sending machine's connection. The receiving TCP software assigns its own unique socket number and sends it back to the original machine. The two unique numbers then define the connection between the two machines until the virtual circuit is terminated. (I look at sockets in a little more detail in a moment.)

After the virtual circuit is established, TCP sends the segment to the IP software, which then issues the message over the network as a datagram. IP can perform any of the changes to the segment that you saw in yesterday's material, such as fragmenting it and reassembling it at the destination machine. These steps are completely transparent to the TCP layers, however. After winding its way over the network, the receiving machine's IP passes the received segment up to the recipient machine's TCP layer, where it is processed and passed up to the applications above it using an upper-layer protocol.

If the message was more than one TCP segment long (not IP datagrams), the receiving TCP software reassembles the message using the sequence numbers contained in each segment's header. If a segment is missing or corrupt (which can be determined from the checksum), TCP returns a message with the faulty sequence number in the body. The originating TCP software can then resend the bad segment.

If only one segment is used for the entire message, after comparing the segment's checksum with a newly calculated value, the receiving TCP software can generate either a positive acknowledgment (ACK) or a request to resend the segment and route the request back to the sending layer.

The receiving machine's TCP implementation can perform a simple flow control to prevent buffer overload. It does this by sending a buffer size called a window value to the sending machine, following which the sender can send only enough bytes to fill the window. After that, the sender must wait for another window value to be received. This provides a handshaking protocol between the two machines, although it slows down the transmission time and slightly increases network traffic.

> The use of a sliding window is more efficient than a single block send and acknowledgment scheme because of delays waiting for the acknowledgment. By implementing a sliding window, several blocks can be sent at once. A properly configured sliding window protocol provides a much higher throughput.

As with most connection-based protocols, timers are an important aspect of TCP. The use of a timer ensures that an undue wait is not involved while waiting for an ACK or an error message. If the timers expire, an incomplete transmission is assumed. Usually an expiring timer before the sending of an acknowledgment message causes a retransmission of the datagram from the originating machine.

Timers can cause some problems with TCP The specifications for TCP provide for the acknowledgment of only the highest datagram number that has been received without error, but this cannot properly handle fragmentary reception. If a message is composed of several datagrams that arrive out of order, the specification states that TCP cannot acknowledge the reception of the message until all the datagrams have been received. So even if all but one datagram in the middle of the sequence have been successfully received, a timer might expire and cause all the datagrams to be resent. With large messages, this can cause an increase in network traffic.

If the receiving TCP software receives duplicate datagrams (as can occur with a retransmission after a timeout or due to a duplicate transmission from IP), the receiving version of TCP discards any duplicate datagrams, without bothering with an error message. After all, the sending system cares only that the message was received—not how many copies were received.

TCP does not have a negative acknowledgment (NAK) function; it relies on a timer to indicate lack of acknowledgment. If the timer has expired after sending the datagram without receiving an acknowledgment of receipt, the datagram is assumed to have been lost and is retransmitted. The sending TCP software keeps copies of all unacknowledged datagrams in a buffer until they have been properly acknowledged. When this happens, the retransmission timer is stopped, and the datagram is removed from the buffer.

TCP supports a push function from the upper-layer protocols. A push is used when an application wants to send data immediately and confirm that a message passed to TCP has been successfully transmitted. To do this, a push flag is set in the ULP connection, instructing TCP to forward any buffered information from the application to the destination as soon as possible (as opposed to holding it in the buffer until it is ready to transmit it).

Ports & Sockets

Table 4.1. Frequently used TCP port numbers.

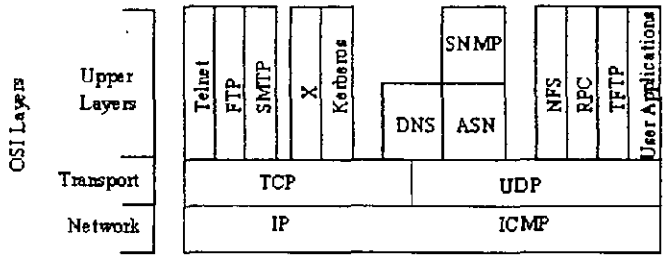
| Port Number | Process Name | Description |
|---|---|---|
| 1 | TCPMUX | TCP Port Service Multiplexer |
| 5 | RJE | Remote Job Entry |
| 7 | ECHO | Echo |
| 9 | DISCARD | Discard |
| 11 | USERS | Active Users |
| 13 | DAYTIME | Daytime |
| 17 | Quote | Quotation of the Day |
| 19 | CHARGEN | Character generator |
| 20 | FTP-DATA | File Transfer Protocol•Data |
| 21 | FTP | File Transfer Protocol•Control |
| 23 | TELNET | Telnet |
| 25 | SMTP | Simple Mail Transfer Protocol |
| 27 | NSW-FE | NSW User System Front End |
| 29 | MSG-ICP | MSG-ICP |
| 31 | MSG-AUTH | MSG Authentication |
| 33 | DSP | Display Support Protocol |
| 35 | | Private Print Servers |
| 37 | TIME | Time |
| 39 | RLP | Resource Location Protocol |
| 41 | GRAPHICS | Graphics |
| 42 | NAMESERV | Host Name Server |
| 43 | NICNAME | Who Is |
| 49 | LOGIN | Login Host Protocol |
| 53 | DOMAIN | Domain Name Server |
| 67 | BOOTPS | Bootstrap Protocol Server |
| 68 | BOOTPC | Bootstrap Protocol Client |
| 69 | TFTP | Trivial File Transfer Protocol |
| 79 | FINGER | Finger |
| 101 | HOSTNAME | NIC Host Name Server |
| 102 | ISO-TSAP | ISO TSAP |
| 103 | X400 | X.400 |
| 104 | X400SND | X 400 SND |
| 105 | CSNET-NS | CSNET Mailbox Name Server |
| 109 | POP2 | Post Office Protocol v2 |
| 110 | POP3 | Post Office Protocol v3 |
| 111 | RPC | Sun RPC Portmap |
| 137 | NETBIOS-NS | NETBIOS Name Service |
| 138 | NETBIOS-DG | NETBIOS Datagram Service |
| 139 | NETBIOS-SS | NETBIOS Session Service |
| 146 | ISO-TP0 | ISO TP0 |
| 147 | ISO-IP | ISO IP |
| 150 | SQL-NET | SQL NET |
| 153 | SGMP | SGMP |
| 156 | SQLSRV | SQL Service |
| 160 | SGMP-TRAPS | SGMP TRAPS |
| 161 | SNMP | SNMP |
| 162 | SNMPTRAP | SNMPTRAP |
| 163 | CMIP-MANAGE | CMIP/TCP Manager |
| 164 | CMIP-AGENT | CMIP/TCP Agent |
| 165 | XNS-Courier | Xerox |
| 179 | BGP | Border Gateway Protocol |

Table 4.2. ULP-TCP service primitives.

| Command Primitives | Parameters Expected ULP to TCP Service Request |
|---|---|
| ABORT | Local connection name |
| ACTIVE-OPEN | Local port, remote socket |
| | Optional: ULP timeout, timeout action, precedence, security, options |
| ACTIVE-OPEN-WITH-DATA | Source port, destination socket, data, data length, push flag, urgent flag |
| | Optional: ULP timeout, timeout action, precedence, security |

Telnet - RemoteLogin
FTP - File Transfer Protocol
SMTP - Simple Mail Transfer Protocol
X - X Windows System
Kerberos - Security
DNS - Domain Name System
ASN - Abstract Syntax Notation
SNMP - Simple Network Management Protocol

NFS - Network File Server
RPC - Remote Procedure Calls
TFTP - Trivial File Transfer Protocol
TCP - Transmission Control Protocol
User Datagram Protocol
IP - Internet Protocol
ICMP - Internet Control Message Protocol



The service definitions are formally developed from the bottom layer (physical) upward to the top layer. The advantage of this approach is that the design of the N+1 layer can be based on the functions performed in the N layer, avoiding two functions that accomplish the same task in two adjacent layers.

An entire set of variations on the service name has been developed to apply these definitions, some of which are in regular use:

An **N-service user** is a user of a service provided by the N layer to the next higher (N+1) layer.

An **N-service provider** is the set of N-entities that are involved in providing the N layer service.

An **N-service access point** (often abbreviated to N-SAP) is where an N-service is provided to an (N+1)-entity by the N-service provider.

**N-service data** is the packet of data exchanged at an N-SAP.

**N-service data units** (N-SDUs) are the individual units of data exchanged at an N-SAP (so that N-service data is made up of N-SDUs).

. Another common term is encapsulation, which is the addition of control information to a packet of data. The control data contains addressing details, checksums for error detection, and protocol control functions.

**Segmentation** is the process of breaking an N-service data unit (N-SDU) into several N-protocol data units (N-PDUs).

**Reassembly** is the process of combining several N-PDUs into an N-SDU (the reverse of segmentation).

**Blocking** is the combination of several SDUs (which might be from different services) into a larger PDU within the layer in which the SDUs originated.

**Unblocking** is the breaking up of a PDU into several SDUs in the same layer.

**Concatenation** is the process of one layer combining several N-PDUs from the next higher layer into one SDU (like blocking except occurring across a layer boundary).

**Separation** is the reverse of concatenation, so that a layer breaks a single SDU into several PDUs for the next layer higher (like unblocking except across a layer boundary).

**Multiplexing** is when several connections are supported by a single connection in the next lower layer (so three presentation service connections could be multiplexed into a single session connection).

**Demultiplexing** is the reverse of multiplexing, in which one connection is split into several connections for the layer above it.

**Splitting** is when a single connection is supported by several connections in the layer below (so the data link layer might have three connections to support one network layer connection).

**Recombining** is the reverse of splitting, so that several connections are combined into a single one for the layer above.

**Multiplexing and splitting** (and their reverses, demultiplexing and recombining) are different in the manner in which the lines are split. With multiplexing, several connections combine into one in the layer below. With splitting, however, one connection can be split into several in the layer below. As you might expect, each has its importance within TCP and OSI.

**Multiplexing** is when several connections are supported by a single connection. According to the formal definition, this applies to layers (so that three presentation service connections could be multiplexed into a single session connection). However, it is a term generally used for all kinds of connections, such as putting four modem calls down a single modem line. Demultiplexing is the reverse of multiplexing, in which one connection is split into several connections.

Multiplexing is a key to supporting many connections at once with limited resources. A typical example is a remote office with twenty terminals, each of which is connected to the main office by a telephone line. Instead of requiring twenty lines, they can all be multiplexed into three or four. The amount of multiplexing possible depends on the maximum capacity of each physical line.

How many protocol headers are added by the time an OSI-based e-mail application (in the application layer) has sent a message to the physical layer for transmission?

Seven, one for each OSI layer. More protocol headers can be added by the actual physical network system. As a general rule, each layer adds its own protocol information.

**Packets**

To transfer data effectively, many experiments have shown that creating a uniform chunk of data is better than sending characters singly or in widely varying sized groups. Usually these chunks of data have some information ahead of them (the header) and sometimes an indicator at the end (the trailer). These chunks of data are called packets in most synchronous communications systems.

The amount of data in a packet and the composition of the header can change depending on the communications protocol as well as some system limitations, but the concept of a packet always refers to the entire set (including header and trailer). The term packet is used often in the computer industry, sometimes when it shouldn't be.

You often see the word packet used as a generic reference to any group of data packaged for transmission. As an application's data passes through the layers of the architecture, each adds more information. The term packet is frequently used at each stage. Treat the term packet as a generalization for any data with additional information, instead of the specific result of only one layer's addition of header and trailer. This goes against the efforts of both OSI and the TCP governing bodies, but it helps keep your sanity intact!

**Subsystems**

A subsystem is the collective of a particular layer across a network. For example, if 10 machines are connected together, each running the seven-layer OSI model, all 10 application layers are the application subsystem, all 10 data link layers are the data link subsystem, and so on. As you might have already deduced, with the OSI Reference Model there are seven subsystems.

It is entirely possible (and even likely) that all the individual components in a subsystem will not be active at one time. Using the 10-machine example again, only three might have the data link layer actually active at any moment in time, but the cumulative of all the machines makes up the subsystem.

**Entities**

A layer can have more than one part to it. For example, the transport layer can have routines that verify checksums as well as routines that handle resending packets that didn't transfer correctly. Not all these routines are active at once, because they might not be required at any moment. The active routines, though, are called entities. The word entity was adopted in order
to find a single term that could not be confused with another computer term such as module, process, or task.

**N Notation**

The notations N, N+1, N+2, and so on are used to identify a layer and the layers that are related to it. Referring to Figure 1.7, if the transport layer is layer N, the physical layer is N–3 and the presentation layer is N+2. With OSI, N always has a value of 1 through 7 inclusive.

One reason this notation was adopted was to enable writers to refer to other layers without having to write out their names every time. It also makes flow charts and diagrams of interactions a little easier to draw. The terms N+1 and N–1 are commonly used in both OSI and TCP for the layers above and below the current layer, respectively, as you will see

To make things even more confusing, many OSI standards refer to a layer by the first letter of its name. This can lead to a real mess for the casual reader, because "S-entity," "5-entity," and "layer 5" all refer to the session layer.

**N-Functions**

Each layer performs N-functions The functions are the different things the layer does. Therefore, the functions of the transport layer are the different tasks that the layer provides. For most purposes in this book, functions and entities mean the same thing.

**N-Facilities**

This uses the hierarchical layer structure to express the idea that one layer provides a set of facilities to the next higher layer. This is sensible, because the application layer expects the presentation layer to provide a robust, well-defined set of facilities to it. In OSI-speak, the (N+1)-entities assume a defined set of N-facilities from the N-entity.

**Services**

The entire set of N-facilities provided to the (N+1)-entities is called the N-service In other words, the service is the entire set of N-functions provided to the next higher layer. Services might seem like functions, but there is a formal difference between the two. The OSI documents go to great lengths to provide detailed descriptions of services, with a "service definition standard" for each layer This was necessary during the development of the OSI standard so that the different tasks involved in the communications protocol could be assigned to different layers, and so that the functions of each layer are both well-defined and isolated from other layers.

5 Chaos
6 IEEE 802.X
7 ARCnet

**The Protocol Type Field**

The protocol type identifies the type of protocol the sending device is using. With TCP/IP, these protocols are usually an EtherType, for which the legal values are as follows:

| Decimal | Description |
|---|---|
| 512 | XEROX PUP |
| 513 | PUP Address Translation |
| 1536 | XEROX NS IDP |
| 2048 | Internet Protocol (IP) |
| 2049 | X.75 |
| 2050 | NBS |
| 2051 | ECMA |
| 2052 | Chaosnet |
| 2053 | X.25 Level 3 |
| 2054 | Address Resolution Protocol (ARP) |
| 2055 | XNS |
| 4096 | Berkeley Trailer |
| 21000 | BBN Simnet |
| 24577 | DEC MOP Dump/Load |
| 24578 | DEC MOP Remote Console |
| 24579 | DEC DECnet Phase IV |
| 24580 | DEC LAT |
| 24582 | DEC |
| 24583 | DEC |
| 32773 | HP Probe |
| 32784 | Excelan |
| 32821 | Reverse ARP |
| 32824 | DEC LANBridge |
| 32823 | AppleTalk |

If the protocol is not EtherType, other values are allowed.

**ARP and IP Addresses**

Two (or more) networks connected by a gateway can have the same network address. The gateway has to determine which network the physical address or IP address corresponds with. The gateway can do this with a modified ARP, called the Proxy ARP (sometimes called Promiscuous ARP). A proxy ARP creates an ARP cache consisting of entries from both networks, with the gateway able to transfer datagrams from one network to the other. The gateway has to manage the ARP requests and replies that cross the two networks.

An obvious flaw with the ARP system is that if a device doesn't know its own IP address, there is no way to generate requests and replies. This can happen when a new device (typically a diskless workstation) is added to the network. The only address the device is aware of is the physical address set either by switches on the network interface or by software. A simple solution is the Reverse Address Resolution Protocol (RARP), which works the reverse of ARP, sending out the
physical address and expecting back an IP address The reply containing the IP address is sent by an RARP server, a machine that can supply the information. Although the originating device sends the message as a broadcast, RARP rules stipulate that only the RARP server can generate a reply. (Many networks assign more than one RARP server, both to spread the processing load and to act as a backup in case of problems )

**DNS**

Seven first-level domain names have been established by the NIC so far. These are as follows.

| | |
|---|---|
| .arpa | An ARPANET-Internet identification |
| .com | Commercial company |
| .edu | Educational institution |
| .gov | Any governmental body |
| .mil | Military |
| .net | Networks used by Internet Service Providers |
| .org | Anything that doesn't fall into one of the other categories |

The adoption of TCP/IP didn't conflict with the OSI standards because the two developed concurrently. In some ways, TCP/IP contributed to OSI, and vice-versa. Several important differences do exist, though, which arise from the basic requirements of TCP/IP which are:

A common set of applications

Dynamic routing

Connectionless protocols at the networking level

Universal connectivity.

Packet-switching

There is a considerable amount of pressure from the user community to abandon the OSI model (and any future communications protocols developed that conform to it) in favor of TCP/IP. The argument hinges on some obvious reasons

TCP/IP is up and running and has a proven record

TCP/IP has an established, functioning management body.

Thousands of applications currently use TCP/IP and its well-documented application programming interfaces.

TCP/IP is the basis for most UNIX systems, which are gaining the largest share of the operating system market (other than desktop single-user machines such as the PC and Macintosh).

TCP/IP is vendor-independent.

Arguing rather strenuously against TCP/IP, surprisingly enough, is the US government—the very body that sponsored it in the first place. Their primary argument is that TCP/IP is not an internationally adopted standard, whereas OSI has that recognition. The Department of Defense has even begun to move its systems away from the TCP/IP protocol set. A compromise will probably result, with some aspects of OSI adopted into the still-evolving TCP/IP protocol suite.

Ethernet is a hardware system providing for the data link and physical layers of the OSI model. As part of the Ethernet standards, issues such as cable type and broadcast speeds are established. There are several different versions of Ethernet, each with a different data transfer rate. The most common is Ethernet version 2, also called 10Base5, Thick Ethernet, and
IEEE 802.3 (after the number of the standard that defines the system adopted by the Institute of Electrical and Electronic Engineers). This system has a 10 Mbps rate.

There are several commonly used variants of Ethernet, such as Thin Ethernet (called 10Base2), which can operate over thinner cable (such as the coaxial cable used in cable television systems), and Twisted-Pair Ethernet (10BaseT), which uses simple twisted-pair wires similar to telephone cable. The latter variant is popular for small companies because it is inexpensive, easy to wire, and has no strict requirements for distance between machines.

The **NFSNET backbone** is comprised of approximately 3,000 research sites, connected by T-3 leased lines running at 44.736 Megabits per second. Tests are currently underway to increase the operational speed of the backbone to enable more throughput and accommodate the rapidly increasing number of users. Several technologies are being field-tested, including Synchronous Optical Network (SONET), Asynchronous Transfer Mode (ATM), and ANSI's proposed High-Performance
Parallel Interface (HPPI). These new systems can produce speeds approaching 1 Gigabit per second.

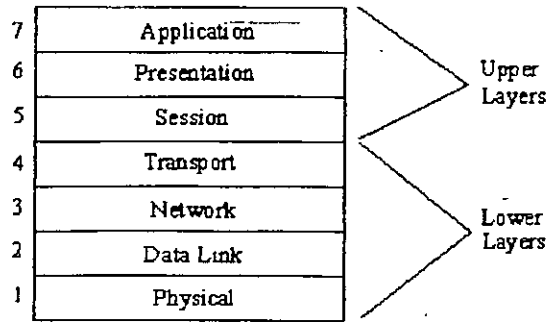
**IP**

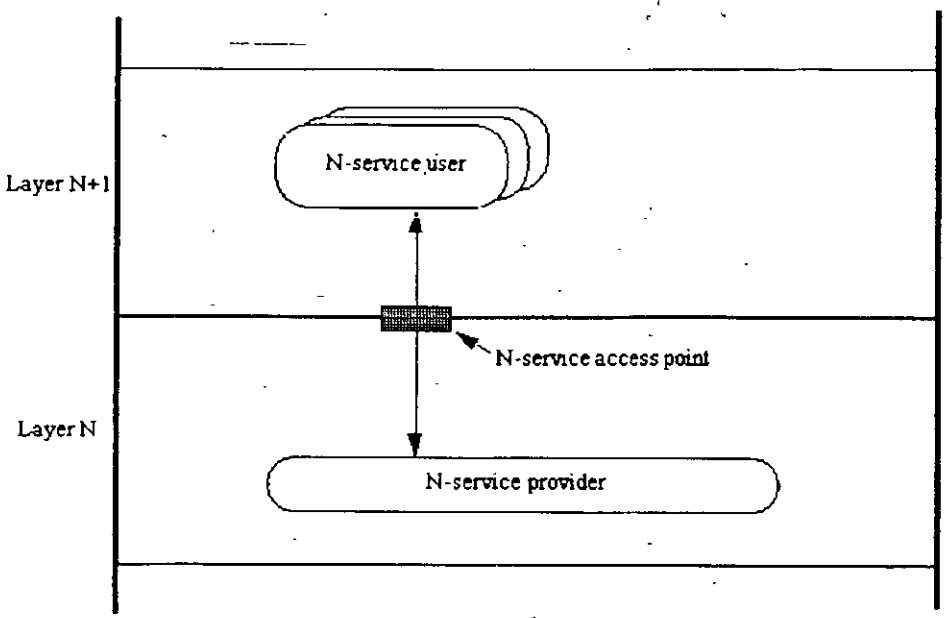
A name is a specific identification of a machine, a user, or an application. It is usually unique and provides an absolute target for the datagram. An address typically identifies where the target is located, usually its physical or logical location in a network. A route tells the system how to get a datagram to the address

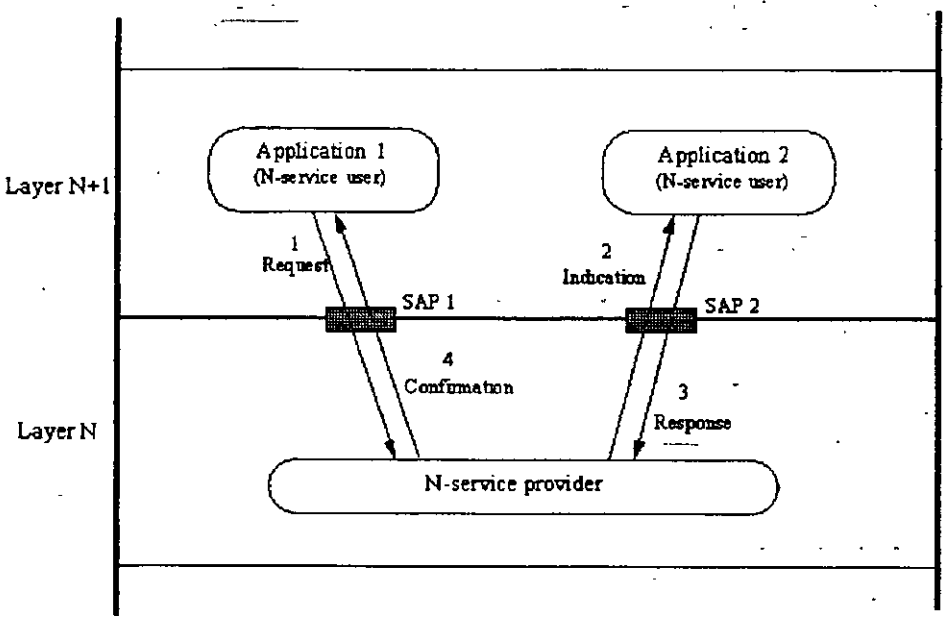
The **Hardware Type Field**

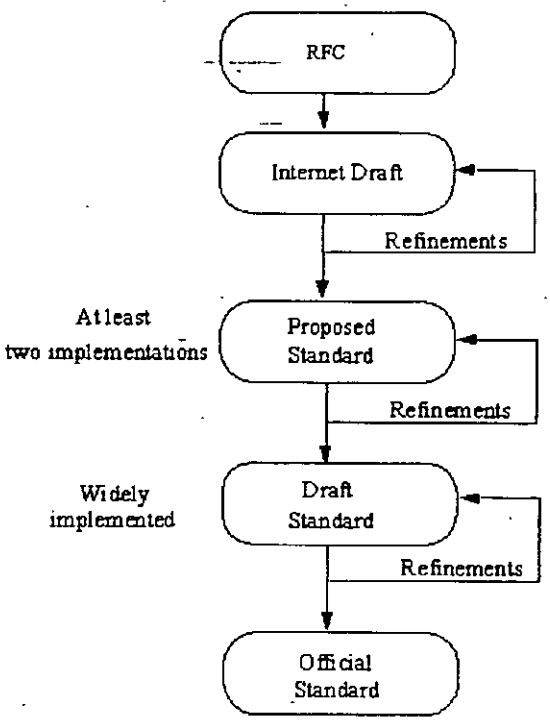
The hardware type identifies the type of hardware interface. Legal values are as follows.

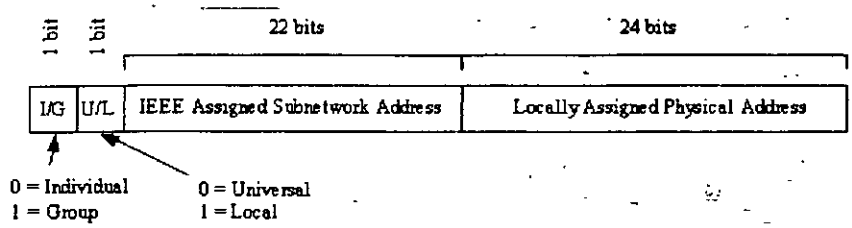
| Type | Description |
|---|---|
| 1 | Ethernet |
| 2 | Experimental Ethernet |
| 3 | X.25 |
| 4 | Proteon ProNET (Token Ring) |


7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data Link
1
Physical
Upper Layers
Lower Layers

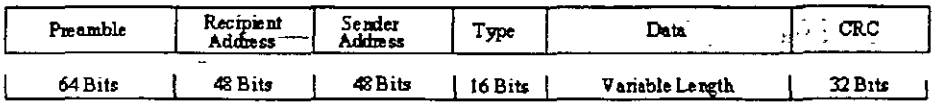

Layer N+1
N-service user
N-service access point
Layer N
N-service provider

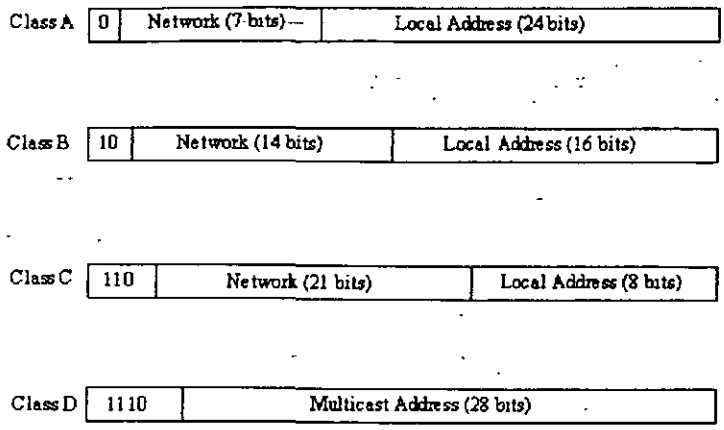

Layer N+1
Application 1
(N-service user)
Application 2
(N-service user)
1
Request
2
Indication
SAP 1
SAP 2
4
Confirmation
3
Response
Layer N
N-service provider


RFC
Internet Draft
Refinements
Atleast
two implementations
Proposed
Standard
Refinements
Widely
implemented
Draft
Standard
Refinements
Official
Standard

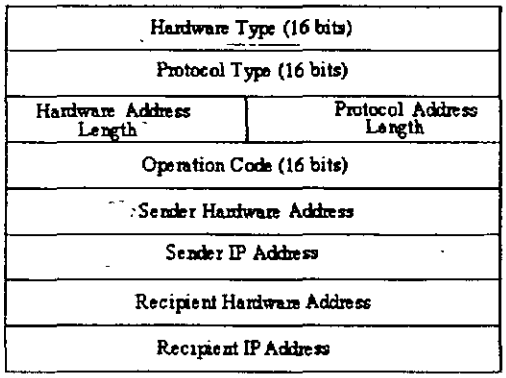

1 bit
1 bit
22 bits
24 bits
I/G
U/L
IEEE Assigned Subnetwork Address
Locally Assigned Physical Address
0 = Individual
1 = Group
0 = Universal
1 = Local

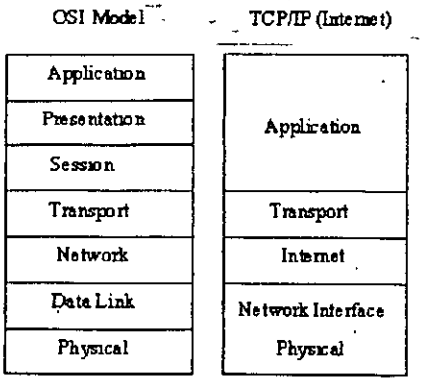
Ethernet Frame

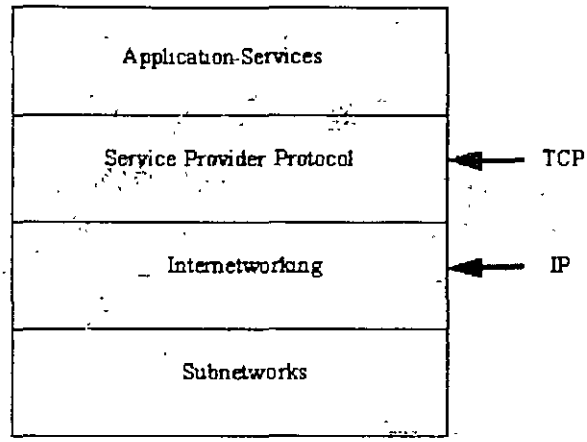



Class A
0
Network (7 bits)
Local Address (24 bits)
Class B
10
Network (14 bits)
Local Address (16 bits)
Class C
110
Network (21 bits)
Local Address (8 bits)
Class D
1110
Multicast Address (28 bits)

ARP Request & Reply layout




OSI Model
TCP/IP (Internet)
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Transport
Internet
Network Interface
Physical


Application-Services
Service Provider Protocol
TCP
Internetworking
IP
Subnetworks