El comité 802.3 decidió crear una Ethernet mejorada por tres razones principales:

1. La necesidad de compatibilidad hacia atrás con las LANs Ethernet existentes.
2. El miedo de que un nuevo protocolo tuviera problemas no previstos.
3. El deseo de terminar el trabajo antes de que la tecnología cambiara.

El trabajo se terminó rápidamente (mediante las normas de los comités de estándares), y el resultado, **802.3u,** fue aprobado oficialmente por el IEEE en junio de 1995. Técnicamente, 802.3u no es un nuevo estándar, sino un agregado al estándar existente 802.3 (para enfatizar su compatibilidad hacia atrás). Puesto que prácticamente todos lo llaman **Fast Ethernet,** en lugar de 802.3u, nosotros también lo haremos.

La idea básica detrás de Fast Ethernet era sencilla: mantener todos los formatos anteriores, interfaces y reglas de procedimientos, y sólo reducir el tiempo de bits de 100 nseg a 10 nseg. Técnicamente, habría sido posible copiar 10Base-5 o 10Base-2 y aún detectar colisiones a tiempo con sólo reducir la longitud máxima de cable por un factor de diez. Sin embargo, las ventajas del cableado 10Base-T eran tan abrumadoras que Fast Ethernet se basa por completo en este diseño. Por lo tanto, todos los sistemas Fast Ethernet utilizan concentradores y conmutadores; no se permiten cables con múltiples derivaciones vampiro ni conectores BNC.

Sin embargo, aún se tienen que tomar algunas decisiones, la más importante de las cuáles es qué tipos de cable soportar. Un contendiente era el cable de par trenzado categoría 3. El argumento a su favor era que prácticamente todas las oficinas en el mundo occidental tienen por lo menos cuatro cables de par trenzado categoría tres (o mejor) que van de la oficina hacia un gabinete de cableado telefónico dentro de una distancia de 100 metros. Algunas veces existen dos de esos cables. Por lo tanto, el uso del cable de par trenzado categoría 3 hace posible cablear las computadoras de escritorio mediante Fast Ethernet sin tener que volver a cablear el edificio, lo cual es una enorme ventaja para muchas organizaciones.

La principal desventaja del cable de par trenzado categoría 3 es su incapacidad de llevar señales de 200 megabaudios (100 Mbps con codificación Manchester) a una distancia de hasta100 metros, que es la distancia máxima de computadora a concentrador especificada para 10Base-T (vea la figura 4-13). En contraste, el cable de par trenzado categoría 5 puede manejar 100 metros con facilidad, y la fibra puede ir mucho más rápido. El arreglo elegido fue permitir las tres posibilidades, como se muestra en la figura 4-21, pero fortalecer la solución categoría 3 para darle la capacidad de transmisión adicional necesaria.

Nombre	Cable	Segmento máximo	Ventajas
100Base-T4	Par trenzado	100 m	Utiliza UTP categoría 3
100Base-TX	Par trenzado	100 m	Dúplex total a 100 Mbps (UTP cat 5)
100Base-FX	Fibra óptica	2000 m	Dúplex total a 100 Mbps; distancias largas

Figura 4-21. El cableado original de Fast Ethernet.

El esquema UTP categoría 3, llamado **100Base-T4,** utiliza una velocidad de señalización de 25 MHz, tan sólo 25 por ciento más rápida que los 20 MHz de la Ethernet estándar (recuerde que la codificación Manchester, como se muestra en la figura 4-16, requiere dos periodos de reloj para cada uno de los 10 millones de bits cada segundo). Sin embargo, para alcanzar el ancho de banda necesario, 100Base-T4 requiere cuatro cables de par trenzado. Debido a que el cableado telefónico estándar durante décadas ha tenido cuatro cables de par trenzado por cable, la mayoría de las oficinas puede manejar esto. Por supuesto, esto significa ceder su teléfono de la oficina, pero seguramente eso es un precio pequeño a cambio de correo electrónico más rápido.

De los cuatro cables de par trenzado, uno siempre va al concentrador, uno siempre sale del concentrador y los otros dos son intercambiables a la dirección actual de transmisión. Para obtener el ancho de banda necesario, no se utiliza la codificación Manchester, pero con relojes modernos y distancias cortas, ya no es necesaria. Además, se envían señales ternarias, para que durante un periodo de reloj el cable pueda contener un 0, un 1 o un 2. Con tres cables de par trenzado y la señalización ternaria, se puede transmitir cualquiera de 27 símbolos posibles, con lo que se pueden enviar 4 bits con algo de redundancia. Transmitir 4 bits en cada uno de los 25 millones de ciclos de reloj por segundo da los 100 Mbps necesarios. Además, siempre hay un canal de regreso de 33.3 Mbps que utiliza el resto del cable de par trenzado. No es probable que este esquema, conocido como **8B/6T** (8 bits se convierten en 6 trits), gane un premio por elegancia, pero funciona con la planta de cableado existente.

Para el cableado categoría 5, el diseño **100Base-TX** es más simple porque los cables pueden manejar velocidades de reloj de 125 MHz. Sólo se utilizan dos cables de par trenzado por estación, uno para enviar y otro para recibir. La codificación binaria directa no se utiliza; en su lugar se toma un esquema llamado **4B/5B** tomado de las redes FDDI, y compatible con ellas. Cada grupo de cinco periodos de reloj, cada uno de los cuales contiene uno de dos valores de señal, da 32 combinaciones. Dieciséis de estas combinaciones se utilizan para transmitir los cuatro grupos de bits 0000, 0001, 0010, ..., 1111. Algunos de los 16 restantes se utilizan para propósitos de control, como el marcado de límites de tramas. Las combinaciones utilizadas se han elegido cuidadosamente para proporcionar suficientes transiciones para mantener sincronización de reloj. El sistema 100Base-TX es de dúplex total; las estaciones pueden transmitir a 100 Mbps y recibir a 100 Mbps al mismo tiempo. Con frecuencia, 100Base-TX y 100Base-T4 se llaman en conjunto **100Base-T.**

La última opción, **100Base-FX,** utiliza dos filamentos de fibra multimodo, una para cada dirección, por lo que también es dúplex total con 100 Mbps en cada dirección. Además, la distancia entre una estación y el concentrador puede ser de hasta 2 km.

En respuesta a la demanda popular, en 1997 el comité 802 agregó un nuevo tipo de cableado, 100Base-T2, que permite que la Fast Ethernet se ejecute a través de dos pares de cables existentes de categoría 3. Sin embargo, se necesita un procesador de señales digital sofisticado para manejar el esquema de codificación requerido, lo que hace de esta opción algo muy costoso. Hasta ahora su uso es muy inusual debido a su complejidad y costo, así como al hecho de que muchos edificios de oficinas se han vuelto a cablear con UTP categoría 5.

Con 100Base-T son posibles dos tipos de dispositivos de interconexión: concentradores y conmutadores, como se muestra en la figura 4-20. En un concentrador, todas las líneas entrantes (o al menos todas las líneas que llegan a una tarjeta de conexión) se conectan lógicamente, formando

un solo dominio de colisión. Se aplican todas las reglas estándar, entre ellas el algoritmo de retroceso exponencial binario, por lo que el sistema funciona de la misma manera que la Ethernet antigua. En particular, sólo una estación a la vez puede transmitir. En otras palabras, los concentradores requieren comunicación semidúplex.

En un conmutador, cada trama entrante se almacena en el búfer de una tarjeta de conexión y se pasa a través de una matriz de conmutación de alta velocidad de la tarjeta de origen a la de destino, si es necesario. La matriz de conmutación no se ha estandarizado, ni lo necesita, debido a que está completamente oculta dentro del conmutador. Si la experiencia pasada sirve de algo, los fabricantes de conmutadores competirán con ardor para producir matrices de conmutación más veloces para mejorar la velocidad real de transporte del sistema. Debido a que los cables 100Base-FX son muy largos para el algoritmo de colisiones de la Ethernet, deben conectarse a conmutadores, de manera que cada uno sea un dominio de colisión en sí mismo. Los concentradores no están permitidos con 100Base-FX.

Como nota final, casi todos los conmutadores pueden manejar una mezcla de estaciones de 10 y 100 Mbps, para facilitar la actualización. Conforme un sitio adquiera más y más estaciones de trabajo de 100 Mbps, todo lo que tiene que hacer es comprar la cantidad necesaria de tarjetas de línea e insertarlas en el conmutador. De hecho, el estándar mismo proporciona una forma para que dos estaciones negocien de manera automática la velocidad óptima (10 o 100 Mbps) y el tipo de transmisión dúplex (semi o total). La mayoría de los productos de Fast Ethernet utilizan esta característica para autoconfigurarse.

4.3.8 Gigabit Ethernet

La tinta apenas se estaba secando en el estándar de la Fast Ethernet cuando el comité 802 comenzó a trabajar en una Ethernet aún más rápida (1995). Se conoció como **Gigabit Ethernet** y fue aprobada por el IEEE en 1998 bajo el nombre 802.3z. Este identificador sugiere que la Gigabit Ethernet va a ser el final de la línea, a menos que alguien invente rápidamente una letra después de la z. A continuación analizaremos algunas de las características principales de la Gigabit Ethernet. Para mayor información, vea (Seifert, 1998).

Los objetivos del comité 802.3z eran esencialmente los mismos que los del comité 802.3u: hacer que Ethernet fuera 10 veces más rápida y que permaneciera compatible hacia atrás con todos los estándares Ethernet existentes. En particular, Gigabit Ethernet tiene que ofrecer servicio de datagramas sin confirmación de recepción con difusión y multidifusión, utilizar el mismo esquema de direccionamiento de 48 bits que el actual y mantener el mismo formato de trama, incluyendo los tamaños mínimo y máximo de trama. El estándar final cumple con todos estos objetivos.

Todas las configuraciones de Gigabit Ethernet son de punto a punto en lugar de múltiples derivaciones como en el estándar original de 10 Mbps, ahora conocido como **Ethernet clásica.** En la configuración más simple de Gigabit Ethernet, que se muestra en la figura 4-22(a), dos computadoras están conectadas de manera directa entre sí. Sin embargo, el caso más común es tener un conmutador o un concentrador conectado a múltiples computadoras y posiblemente a conmutadores o concentradores adicionales, como se muestra en la figura 4-22(b). En ambas configuraciones cada cable Ethernet individual tiene exactamente dos dispositivos en él, ni más ni menos.

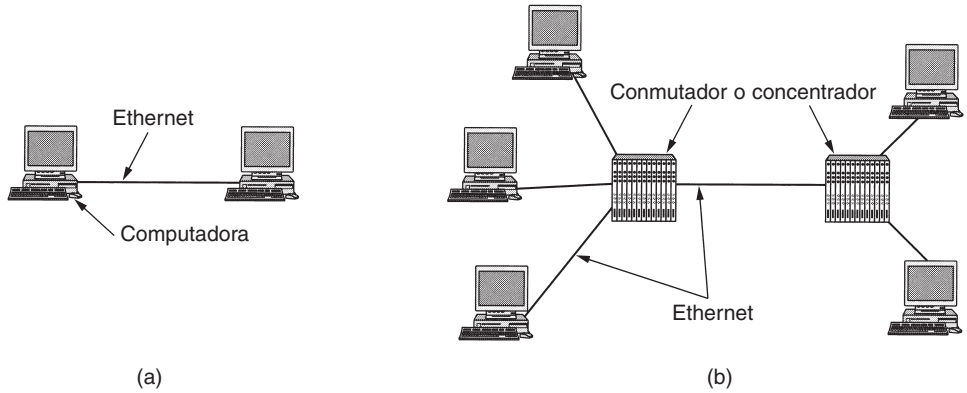


Figura 4-22. (a) Ethernet de dos estaciones. (b) Ethernet con múltiples estaciones.

Gigabit Ethernet soporta dos modos diferentes de funcionamiento: modo de dúplex total y modo de semidúplex. El modo "normal" es el de dúplex total, el cual permite tráfico en ambas direcciones al mismo tiempo. Este modo se utiliza cuando hay un conmutador central conectado a computadoras (o a otros conmutadores) en el periférico. En esta configuración, todas las líneas se almacenan en el búfer a fin de que cada computadora y conmutador pueda enviar tramas siempre que lo desee. El emisor no tiene que detectar el canal para ver si alguien más lo está utilizando debido a que la contención es imposible. En la línea entre una computadora y un conmutador, la computadora es el único emisor posible en esa línea al conmutador y la transmisión tiene éxito aun cuando el conmutador esté enviado actualmente una trama a la computadora (porque la línea es de dúplex total). Debido a que no hay contención, no se utiliza el protocolo CSMA/CD y la longitud máxima del cable se determina con base en la fuerza de la señal más que en el tiempo que tarda una ráfaga de ruido en regresar al emisor en el peor caso. Los conmutadores son libres de mezclar e igualar velocidades. La autoconfiguración se soporta al igual que en Fast Ethernet.

El otro modo de operación, semidúplex, se utiliza cuando las computadoras están conectadas a un concentrador en lugar de a un conmutador. Un concentrador no almacena en el búfer las tramas entrantes. En su lugar, conecta en forma eléctrica todas las líneas internamente, simulando el cable con múltiples derivaciones que se utiliza en la Ethernet clásica. En este modo las colisiones son posibles, por lo que es necesario el protocolo CSMA/CD estándar. Debido a que una trama mínima (de 64 bytes) ahora puede transmitirse 100 veces más rápido que en la Ethernet clásica, la distancia máxima es 100 veces menor, o 25 metros, para mantener la propiedad esencial de que el emisor aún transmita cuando la ráfaga de ruido vuelva a él, incluso en el peor caso. Con un cable de 2500 metros de longitud, el emisor de una trama de 64 bytes a 1 Gbps podría terminar su transmisión antes de que la trama recorra una décima del camino, y muchísimo antes de que llegue al otro extremo y regrese.

El comité 802.3z consideró un radio de 25 metros como inaceptable y agregó dos características al estándar para incrementar el radio. La primera, llamada **extensión de portadora,** esencialmente indica al hardware que agregue su propio relleno después de la trama normal para extenderla a 512 bytes. Puesto que este relleno es agregado por el hardware emisor y eliminado

por el hardware receptor, el software no toma parte en esto, lo que significa que no es necesario realizar cambios al software existente. Por supuesto, utilizar 512 bytes de ancho de banda para transmitir 46 bytes de datos de usuario (la carga útil de una trama de 64 bytes) tiene una eficiencia de línea de 9%.

La segunda característica, llamada **ráfagas de trama,** permite que un emisor transmita una secuencia concatenada de múltiples tramas en una sola transmisión. Si la ráfaga total es menor que 512 bytes, el hardware la rellena nuevamente. Si suficientes tramas están esperando la transmisión, este esquema es muy eficiente y se prefiere antes que la extensión de portadora. Estas nuevas características amplían el radio de red de 200 metros, que probablemente es suficiente para la mayoría de las oficinas.

Sin duda, una organización difícilmente enfrentará el problema de comprar e instalar tarjetas Gigabit Ethernet para obtener mayor rendimiento y después conectar las computadoras con un concentrador para simular una Ethernet clásica con todas sus colisiones. Si bien los concentradores son un tanto más baratos que los conmutadores, las tarjetas de interfaz Gigabit Ethernet aún son relativamente costosas. Por lo tanto, tratar de economizar comprando un concentrador barato y reducir drásticamente el desempeño del nuevo sistema es algo impensable. Además, la compatibilidad hacia atrás es sagrada en la industria de la computación, por lo que se le pidió al comité 802.3z que la añadiera.

Como se lista en la figura 4-23, Gigabit Ethernet soporta tanto el cableado de fibra óptica como el de cobre. Transmitir señales a o aproximadamente a 1 Gbps a través de fibra significa que la fuente de luz debe encenderse y apagarse en 1 nseg. Los LEDs simplemente no pueden funcionar con esta rapidez, por lo que se necesitan láseres. Se permiten dos longitudes de onda: 0.85 micras (Corto) y 1.3 micras (Largo). Los láseres a 0.85 micras son más económicos pero no funcionan en una fibra de modo sencillo.

Nombre	Cable	Segmento máximo	Ventajas
1000Base-SX	Fibra óptica	550 m	Fibra multimodo (50, 62.5 micras)
1000Base-LX	Fibra óptica	5000 m	Sencilla (10 μ) o multimodo (50, 62.5 μ)
1000Base-CX	2 pares de STP	25 m	Cable de par trenzado blindado
1000Base-T	4 Pares de UTP	100 m	UTP categoría 5 estándar

Figura 4-23. Cableado de Gigabit Ethernet.

Se permiten tres diámetros de fibra: 10, 50 y 62.5 micras. El primero es para el modo sencillo y los últimos dos son para multimodo. Sin embargo, no se permiten las seis combinaciones y la distancia máxima depende de la combinación que se utilice. Los números que se dan en la figura 4-23 son para el mejor de los casos. En particular, 5000 metros son alcanzables únicamente con láseres de 1.3 micras que funcionen a través de fibra de 10 micras en modo sencillo, pero ésta es la mejor opción para redes dorsales y se espera que sea popular, a pesar de ser la opción más costosa.

La opción 1000Base-CX utiliza cables de cobre blindados cortos. Su problema es que compite con la fibra de alto desempeño por una parte, y con el UTP más económico por la otra. No es probable que se utilice mucho, si es que se utiliza.

La última opción son paquetes de cuatro cables UTP categoría 5 que trabajan juntos. Debido a que la mayor parte de este cableado ya está instalada, es probable que sea la Gigabit Ethernet de la gente pobre.

Gigabit Ethernet utiliza nuevas reglas de codificación en las fibras. La codificación Manchester a 1 Gbps podría requerir una señal de 2 Gbaudios, lo cual era considerado muy difícil y también un desperdicio de ancho de banda. En su lugar se eligió un nuevo esquema, llamado **8B/10B,** que se basa en un canal de fibra. Cada byte de 8 bits está codificado en la fibra como 10 bits, de aquí el nombre 8B/10B. Debido a que hay 1024 palabras codificadas posibles para cada byte de entrada, hay algo de libertad al elegir cuáles palabras codificadas permitir. Las siguientes dos reglas se utilizaron al realizar las elecciones:

1. Ninguna palabra codificada podría tener más de cuatro bits idénticos en una fila.
2. Ninguna palabra codificada podría tener más de seis bits 0 o seis bits 1.

Estas elecciones se realizaron para mantener suficientes transiciones en el flujo para asegurarse de que el receptor continúe sincronizado con el emisor y también para mantener la cantidad de bits 0 y bits 1 en la fibra tan cerca del equilibrio como sea posible. Además, muchos bytes de entrada tienen dos palabras codificadas posibles asignadas a ellos. Cuando el codificador tiene la opción de palabras codificadas, siempre elige la palabra codificada que tiende a igualar la cantidad de 0s y 1s transmitidos hasta ese momento. Este énfasis en igualar 0s y 1s es necesario para mantener el componente DC de la señal tan bajo como sea posible para permitirle pasar a través de transformadores no modificados. Si bien los científicos de la computación no son afectos a que las propiedades de los transformadores dicten sus esquemas de codificación, algunas veces la vida es así.

Las Gigabit Ethernet que utilizan 1000Base-T emplean un esquema de codificación diferente debido a que cronometrar el tiempo de los datos en el cable de cobre en 1 nseg es muy difícil. Esta solución utiliza cuatro cables de par trenzado categoría 5 para permitir que se transmitan en paralelo cuatro símbolos. Cada uno de ellos se codifica utilizando uno de cinco niveles de voltaje. Este esquema permite que un solo símbolo codifique 00, 01, 10, 11 o un valor especial para propósitos de control. Por lo tanto, hay 2 bits de datos por cable de par tranzado u 8 bits de datos por ciclo de reloj. El reloj se ejecuta a 125 MHz, y permite una operación de 1 Gbps. La razón para permitir cinco niveles de voltaje en lugar de cuatro es tener combinaciones sobrantes para propósitos de entramado y control.

1 Gbps es una velocidad muy alta. Por ejemplo, si un receptor está ocupado con otra tarea por incluso un 1 mseg y no vacía el búfer de entrada en alguna línea, podrían haberse acumulado ahí hasta 1953 tramas en ese espacio de 1 ms. Además, cuando una computadora en una Gigabit Ethernet está enviando datos en la línea a una computadora en una Ethernet clásica, es muy probable que sucedan rebases de búfer. Como consecuencia de estas dos observaciones, Gigabit Ethernet soporta control de flujo (como lo hace la Fast Ethernet, aunque los dos son diferentes).

El control de flujo consiste en que un extremo envíe una trama de control especial al otro extremo indicándole que se detenga por algún tiempo. Las tramas de control son tramas comunes de Ethernet que contienen un tipo de 0x8808. Los primeros dos bytes del campo de datos dan el comando; los bytes exitosos proporcionan los parámetros, si es que hay. Para control de flujo, se utilizan las tramas PAUSE, en las que el parámetro indica cuánto tiempo detenerse, en unidades de tiempo de la trama más pequeña. Para la Gigabit Ethernet, la unidad de tiempo es 512 nseg, lo que permite pausas de 33.6 mseg.

Una vez que la Gigabit Ethernet se estandarizó, el comité 802 se aburrió y quiso volver al trabajo. El IEEE les dijo que iniciaran una Ethernet de 10 gigabits. Después de buscar arduamente una letra que siguiera a la z, el comité abandonó ese enfoque y se concentró en los sufijos de dos letras. Comenzó el trabajo y ese estándar fue aprobado por el IEEE en el 2002 como 802.3ae. ¿Es posible que le siga una Ethernet de 100 gigabits?

4.3.9. Estándar IEEE 802.2: control lógico del enlace

Tal vez ahora sea el momento de dar un paso atrás y comparar lo que hemos aprendido en este capítulo con lo que estudiamos en el anterior. En el capítulo 3 vimos la manera en que dos máquinas se podían comunicar de manera confiable a través de una línea inestable usando varios protocolos de enlace de datos. Estos protocolos proporcionaban control de errores (mediante confirmaciones de recepción) y control de flujo (usando una ventana corrediza).

En contraste, en este capítulo no hemos mencionado las comunicaciones confiables. Todo lo que ofrecen las Ethernet y los protocolos 802 es un servicio de datagramas de mejor esfuerzo. A veces, este servicio es adecuado. Por ejemplo, para transportar paquetes IP no se requieren ni se esperan garantías. Un paquete IP simplemente puede introducirse en un campo de carga 802 y enviarse a su destino; si se pierde, que así sea.

Sin embargo, también hay sistemas en los que se desea un protocolo de enlace de datos con control de errores y control de flujo. El IEEE ha definido uno que puede operar encima de todos los protocolos Ethernet y 802. Además, este protocolo, llamado **LLC (Control Lógico del Enlace),** esconde las diferencias entre los distintos tipos de redes 802, proporcionando un formato único y una interfaz con la capa de red. Este formato, interfaz y protocolo están basados estrechamente en HDLC que estudiamos en el capítulo 3. El LLC forma la mitad superior de la capa de enlace de datos, con la subcapa de MAC por debajo de él, como se muestra en la figura 4-24.

El uso típico del LLC es el siguiente. La capa de red de la máquina emisora pasa un paquete al LLC usando las primitivas de acceso del LLC. A continuación, la subcapa LLC agrega un encabezado LLC que contiene los números de secuencia y confirmación de recepción. La estructura resultante se introduce entonces en el campo de carga útil de una trama 802 y se transmite. En el receptor ocurre el proceso inverso.

El LLC proporciona tres opciones de servicio: servicio no confiable de datagramas, servicio de datagramas sin confirmación de recepción y servicio confiable orientado a la conexión. El encabezado LLC contiene tres campos: un punto de acceso de destino, un punto de acceso de origen y un campo de control. Los puntos de acceso indican de cuál proceso proviene la trama y en dónde

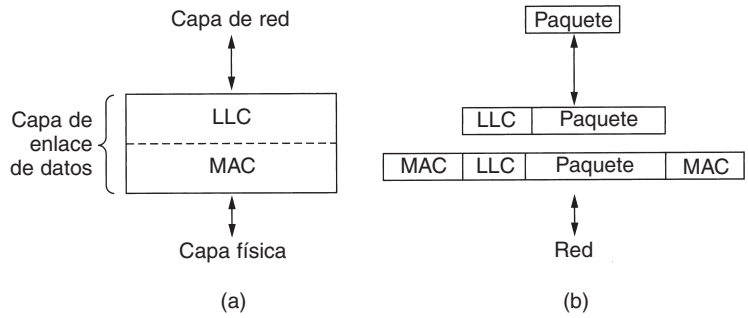


Figura 4-24. (a) Posición del LLC. (b) Formatos de protocolo.

se va a enviar, con lo que reemplazan el campo de *Tipo* DIX. El campo de control contiene números de secuencia y de confirmación de recepción, muy parecido a HDLC (vea la figura 3-24), pero no idéntico. Estos campos se utilizan principalmente cuando se necesita una conexión confiable en el nivel de enlace de datos, en cuyo caso se utilizarían protocolos similares a los que tratamos en el capítulo 3. Para Internet, los intentos de mejor esfuerzo para enviar los paquetes IP son suficientes, por lo que no se requieren confirmaciones de recepción en el nivel LLC.

4.3.10 Retrospectiva de Ethernet

Ethernet ha existido desde hace 20 años y no tiene competidores serios a la vista, por lo que es probable que exista por algunos años más. Pocas arquitecturas de CPU, sistemas operativos o lenguajes de programación han sido los reyes de la montaña por más de dos décadas. Claramente, Ethernet hizo algo bien, pero, ¿qué fue?

Probablemente la razón principal de su longevidad es que Ethernet es simple y flexible. En la práctica, simple se traduce como confiable, barato y fácil de mantener. Una vez que las derivaciones vampiro se reemplazaron con conectores BNC, las fallas eran menos frecuentes. Las personas dudaban en reemplazar algo que funcionaba bien todo el tiempo, especialmente porque sabían que muchas cosas funcionaban pobremente en la industria de la computación, por lo que muchas "actualizaciones" son peores que lo que reemplazan.

Simple también se traduce como barato. El cableado Ethernet delgado y el de par trenzado tienen un costo relativamente bajo. Las tarjetas de interfaz también tienen un costo bajo. Sólo cuando se introdujeron concentradores y conmutadores, se necesitaron inversiones considerables, pero para la época en que entraron en escena, Ethernet ya estaba bien establecida.

Ethernet es fácil de mantener. No hay software que instalar (sólo los controladores) y no hay tablas de configuración que manejar (con las cuales equivocarse). Además, agregar nuevos *hosts* es tan simple como conectarlos.

Otro punto es que Ethernet interactúa fácilmente con TCP/IP, el cual se ha vuelto dominante. IP es un protocolo sin conexión, porque se ajusta perfectamente con Ethernet, que tampoco es orientado a la conexión. IP no se ajusta tan bien con ATM, que es orientado a la conexión. Esta falta de ajuste afecta definitivamente las posibilidades de ATM.

Por último, Ethernet ha sido capaz de evolucionar en formas importantes. Las velocidades han aumentado en algunos niveles de magnitud y se han introducido los concentradores y conmutadores, pero estos cambios no requieren modificaciones en el software. Un vendedor de redes está en un grave problema cuando muestra una instalación grande y dice: "Tengo esta nueva red fantástica para usted. Lo único que tiene que hacer es tirar todo su hardware y reescribir todo su software". Cuando se introdujeron la FDDI, el canal de fibra y ATM, eran más rápidos que Ethernet, pero también eran incompatibles con Ethernet, mucho más complejos y difíciles de manejar. Con el tiempo, Ethernet los igualó en cuanto a velocidad, por lo que ya no tenían ventajas y poco a poco dejaron de utilizarse, excepto ATM, el cual se utiliza en el núcleo del sistema telefónico.

4.4 LANS INALÁMBRICAS

Aunque Ethernet se utiliza ampliamente, está a punto de tener un competidor fuerte. Las LANs inalámbricas se están volviendo muy populares, y más y más edificios de oficinas, aeropuertos y otros lugares públicos se están equipando con ellas. Las LANs inalámbricas pueden funcionar en una de dos configuraciones, como vimos en la figura 1-35: con una estación base y sin ninguna estación base. En consecuencia, el estándar de LAN 802.11 toma en cuenta esto y se previene para ambos arreglos, como veremos más adelante.

En la sección 1.5.4 proporcionamos información sobre 802.11. Ahora es tiempo de ver más de cerca esta tecnología. En las siguientes secciones veremos la pila de protocolos, las técnicas de transmisión de radio de la capa física, el protocolo de la subcapa MAC, la estructura de trama y los servicios. Para mayor información sobre 802.11, vea (Crow y cols., 1997; Geier, 2002; Heegard y cols., 2001; Kapp, 2002; O'Hara y Petrick, 1999, y Severance, 1999). Para obtener información de una fuente fidedigna, consulte el estándar publicado 802.11.

4.4.1 La pila de protocolos del 802.11

Los protocolos utilizados por todas las variantes 802, entre ellas Ethernet, tienen ciertas similitudes de estructura. En la figura 4-25 se muestra una vista parcial de la pila de protocolos del estándar 802.11. La capa física corresponde muy bien con la capa física OSI, pero la capa de enlace de datos de todos los protocolos 802 se divide en dos o más subcapas. En el estándar 802.11, la subcapa MAC determina la forma en que se asigna el canal, es decir, a quién le toca transmitir a continuación. Arriba de dicha subcapa se encuentra la subcapa LLC, cuyo trabajo es ocultar las diferencias entre las variantes 802 con el propósito de que sean imperceptibles para la capa de red. Anteriormente en este capítulo analizamos el LLC, cuando examinamos Ethernet, por lo que no repetiremos ese material aquí.

El estándar 802.11 de 1997 especifica tres técnicas de transmisión permitidas en la capa física. El método de infrarrojos utiliza en su mayor parte la misma tecnología que los controles remotos

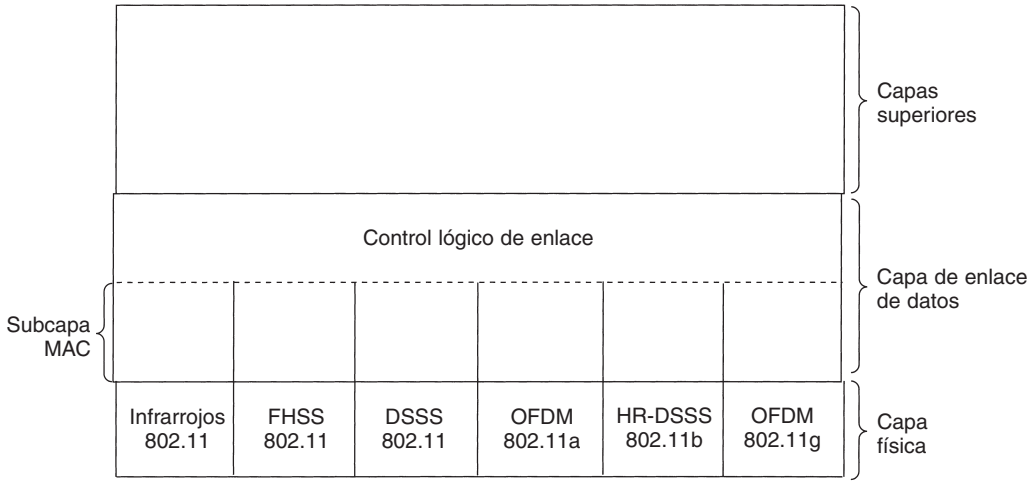


Figura 4-25. Parte de la pila de protocolos del 802.11.

de televisión. Los otros dos métodos utilizan el radio de corto alcance, mediante técnicas conocidas como FHSS y DSSS. Éstas utilizan parte del espectro que no necesita licencia (la banda ISM de 2.4 GHz). Los abridores de puertas de cocheras controlados por radio también utilizan esta parte del espectro, por lo que su computadora portátil podría encontrarse compitiendo con la puerta de la cochera. Los teléfonos inalámbricos y los hornos de microondas también utilizan esta banda. Todas estas técnicas funcionan a 1 o 2 Mbps y con poca energía por lo que no interfieren mucho entre sí. En 1999 se introdujeron dos nuevas técnicas para alcanzar un ancho de banda más alto. Éstas se conocen como OFDM y HRDSSS. Funcionan hasta 54 y 11 Mbps, respectivamente. En 2001 se introdujo una segunda modulación OFDM, pero en una banda de frecuencia diferente respecto a la primera. A continuación examinaremos con brevedad cada una de ellas. Técnicamente, pertenecen a la capa física y debieron examinarse en el capítulo 2, pero las trataremos aquí debido a que están estrechamente enlazadas a las LANs en general y a la subcapa MAC 802.11.

4.4.2 La capa física del 802.11

Cada una de las cinco técnicas permitidas de transmisión posibilitan el envío de una trama MAC de una estación a otra. Sin embargo, difieren en la tecnología utilizada y en las velocidades alcanzables. Un análisis detallado de estas tecnologías está más allá del alcance de este libro, pero es posible que algunas palabras sobre dichas tecnologías, junto con algunos términos clave, proporcionen a los lectores interesados algunos términos con los cuales buscar más información en Internet o en alguna otra parte.

La opción de infrarrojos utiliza transmisión difusa (es decir, no requiere línea visual) a 0.85 o 0.95 micras. Se permiten dos velocidades: 1 y 2 Mbps. A 1 Mbps se utiliza un esquema de codificación en el cual un grupo de 4 bits se codifica como una palabra codificada de 16 bits, que contiene quince 0s y un 1, mediante **código de Gray.** Este código tiene la propiedad de que un pequeño error en la sincronización en el tiempo lleva a un solo error de bits en la salida. A 2 Mbps, la codificación toma 2 bits y produce una palabra codificada de 4 bits, también con un solo 1, que es uno de 0001, 0010, 0100 o 1000. Las señales de infrarrojos no pueden penetrar las paredes, por lo que las celdas en los diferentes cuartos están bien aisladas entre sí. Sin embargo, debido al bajo ancho de banda (y al hecho de que la luz solar afecta las señales de infrarrojos), ésta no es una opción muy popular.

FHSS (**Espectro Disperso con Salto de Frecuencia**) utiliza 79 canales, cada uno de los cuales tiene un ancho de banda de 1 MHz, iniciando en el extremo más bajo de la banda ISM de 2.4 GHz. Para producir la secuencia de frecuencias a saltar, se utiliza un generador de números pseudoaleatorios. Siempre y cuando todas las estaciones utilicen la misma semilla para el generador de números pseudoaleatorios y permanezcan sincronizadas, saltarán de manera simultánea a la misma frecuencia. El tiempo invertido en cada frecuencia, el **tiempo de permanencia,** es un parámetro ajustable, pero debe ser menor que 400 mseg. La aleatorización de FHSS proporciona una forma justa de asignar espectro en la banda ISM no regulada. También proporciona algo de seguridad pues un intruso que no sepa la secuencia de saltos o el tiempo de permanencia no puede espiar las transmisiones. En distancias más grandes, el desvanecimiento de múltiples rutas puede ser un problema, y FHSS ofrece buena resistencia a ello. También es relativamente insensible a la interferencia de radio, lo que lo hace popular para enlaces de edificio en edificio. Su principal desventaja es su bajo ancho de banda.

El tercer método de modulación, **DSSS** (**Espectro Disperso de Secuencia Directa**), también está restringido a 1 o 2 Mbps. El esquema utilizado tiene algunas similitudes con el sistema CDMA que examinamos en la sección 2.6.2, pero difiere en otros aspectos. Cada bit se transmite como 11 *chips*, utilizando lo que se conoce como **secuencia Barker.** Utiliza modulación por desplazamiento de fase a 1 Mbaudio, y transmite 1 bit por baudio cuando opera a 1 Mbps, y 2 bits por baudio cuando opera a 2 Mbps. Durante mucho tiempo, la FCC exigió que todo el equipo de comunicación inalámbrica que operaba en la banda ISM en Estado Unidos utilizará el espectro disperso, pero en mayo de 2002 esa regla se eliminó conforme apareció nueva tecnología.

La primera de las LANs inalámbricas de alta velocidad, **802.11a,** utiliza **OFDM** (**Multiplexión por División de Frecuencias Ortogonales**) para enviar hasta 54 Mbps en la banda ISM más ancha de 5 GHz. Como lo sugiere el término FDM, se utilizan frecuencias diferentes —52 en total, 48 para datos y 4 para sincronización— al igual que ADSL. Debido a que las transmisiones están presentes en múltiples frecuencias al mismo tiempo, esta técnica se considera como una forma de espectro disperso, pero es diferente a CDMA y a FHSS. Dividir la señal en bandas más estrechas tiene más ventajas que el uso de una sola banda ancha, entre ellas mejor inmunidad a la interferencia de bandas estrechas y la posibilidad de utilizar bandas no contiguas. Se utiliza un sistema de codificación complejo, con base en la modulación por desplazamiento de fase para velocidades de hasta 18 Mbps, y en QAM para velocidades mayores. A 54 Mbps, se codifican 216 bits

de datos en símbolos de 288 bits. Parte del motivo para utilizar OFDM es la compatibilidad con el sistema europeo HiperLAN/2 (Doufexi y cols., 2002). La técnica tiene buena eficiencia de espectro en términos de bits/Hz y buena inmunidad al desvanecimiento de múltiples rutas.

A continuación analizaremos **HR-DSSS** (**Espectro Disperso de Secuencia Directa de Alta Velocidad),** otra técnica de espectro disperso, que utiliza 11 millones de chips/seg para alcanzar 11 Mbps en la banda de 2.4 GHz. Se llama **802.11b** pero no es la continuación de 802.11a. De hecho, su estándar se aprobó primero y apareció primero en el mercado. Las tasas de datos soportadas por 802.11b son 1, 2, 5.5 y 11 Mbps. Las dos tasas bajas se ejecutan a 1 Mbaudio, con 1 y 2 bits por baudio, respectivamente, utilizando modulación por desplazamiento de fase (por compatibilidad con DSSS). Las dos tasas más rápidas se ejecutan a 1.375 Mbaudios, con 4 y 8 bits por baudio, respectivamente, utilizando códigos **Walsh/Hadamard.** La tasa de datos puede ser adaptada de manera dinámica durante la operación para alcanzar la velocidad más óptima posible bajo las condiciones actuales de la carga y el ruido. En la práctica, la velocidad de operación de 802.11b siempre es de aproximadamente 11 Mbps. Aunque 802.11b es más lento que 802.11a, su rango es aproximadamente 7 veces mayor, lo que es más importante en muchas situaciones.

En noviembre de 2001, el IEEE aprobó una versión mejorada de 802.11b, **802.11g,** después de mucho politiqueo por cuál tecnología patentada podría utilizar. Utiliza el método de modulación OFDM de 802.11a pero opera en la banda ISM más estrecha 2.4 GHz ISM junto con 802.11b. En teoría, puede operar hasta a 54 Mbps. Aún no se ha decidido si esta velocidad se va a alcanzar en la práctica. Lo que esto significa es que el comité 802.11 ha producido tres LANs inalámbricas diferentes de alta velocidad: 802.11a, 802.11b y 802.11g (sin mencionar las tres LANs inalámbricas de baja velocidad). Uno se puede preguntar si es bueno que el comité de estándares haga esto. Tal vez el tres sea su número de suerte.

4.4.3 El protocolo de la subcapa MAC del 802.11

Regresemos ahora de la tierra de la ingeniería eléctrica a la de las ciencias de la computación. El protocolo de la subcapa MAC para el estándar 802.11 es muy diferente del de Ethernet debido a la complejidad inherente del entorno inalámbrico en comparación con el de un sistema cableado. Con Ethernet, una estación simplemente espera hasta que el medio queda en silencio y comienza a transmitir. Si no recibe una ráfaga de ruido dentro de los primeros 64 bytes, con seguridad la trama ha sido entregada correctamente. Esta situación no es válida para los sistemas inalámbricos.

Para empezar, existe el problema de la estación oculta mencionado con anterioridad, el cual se ilustra nuevamente en la figura 4-26(a). Puesto que no todas las estaciones están dentro del alcance de radio de cada una, las transmisiones que van en un lado de una celda podrían no recibirse en otro lado de la misma celda. En este ejemplo, la estación *C* transmite a la estación *B*. Si *A* detecta el canal, no escuchará nada y concluirá erróneamente que ahora puede comenzar a transmitir a *B*.

Además, existe el problema inverso, el de la estación expuesta, que se ilustra en la figura 4-26(b). Aquí *B* desea enviar a *C* por lo que escucha el canal. Cuando escucha una transmisión, concluye erróneamente que no debería transmitir a *C*, aunque *A* esté transmitiendo a *D* (lo cual no

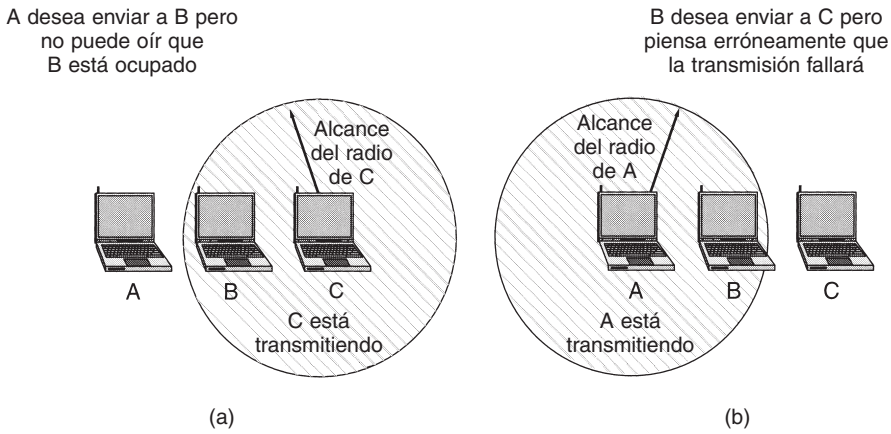


Figura 4-26. (a) El problema de la estación oculta. (b) El problema de la estación expuesta.

se muestra). Además, la mayoría de los radios son semidúplex, lo que significa que no pueden transmitir y escuchar ráfagas de ruido al mismo tiempo en una sola frecuencia. Como resultado de estos problemas, 802.11 no utiliza CSMA/CD, como lo hace Ethernet.

Para solucionar este problema, 802.11 suporta dos modos de funcionamiento. El primero, llamado **DCF** (**Función de Coordinación Distribuida**), no utiliza ningún tipo de control central (en ese aspecto, es similar a Ethernet). El otro, llamado **PCF** (**Función de Coordinación Puntual**), utiliza la estación base para controlar toda la actividad en su celda. Todas las implementaciones soportan DCF pero PCF es opcional. A continuación analizaremos estos dos modos a la vez.

Cuando se emplea DCF, 802.11 utiliza un protocolo llamado **CSMA/CA** (**CSMA con Evitación de Colisiones**). En este protocolo, se utiliza tanto la detección del canal físico como la del canal virtual. Los dos métodos de funcionamiento son soportados por CSMA/CA. En el primer método, cuando una estación desea transmitir, detecta el canal. Si está inactivo, comienza a transmitir. No detecta el canal mientras transmite pero emite su trama completa, la cual podría ser destruida en el receptor debido a interferencia. Si el canal está ocupado, el emisor espera hasta que esté inactivo para comenzar a transmitir. Si ocurre una colisión, las estaciones involucradas en ella esperan un tiempo aleatorio, mediante el algoritmo de retroceso exponencial binario de Ethernet, y vuelve a intentarlo más tarde.

El otro modo de la operación CSMA/CA se basa en MACAW y utiliza la detección de canal virtual, como se ilustra en la figura 4-27. En este ejemplo, *A* desea enviar a *B*. *C* es una estación que está dentro del alcance de *A* (y posiblemente dentro del alcance de *B*, pero eso no importa). *D* es una estación dentro del alcance de *B* pero no dentro del de *A*.

El protocolo inicia cuando *A* decide enviar datos a *B*. *A* inicia enviándole una trama RTS a *B* en la que le solicita permiso para enviarle una trama. Cuando *B* recibe esta solicitud, podría decidir otorgarle el permiso, en cuyo caso le regresa una trama CTS. Al recibir la CTS, *A* ahora envía su

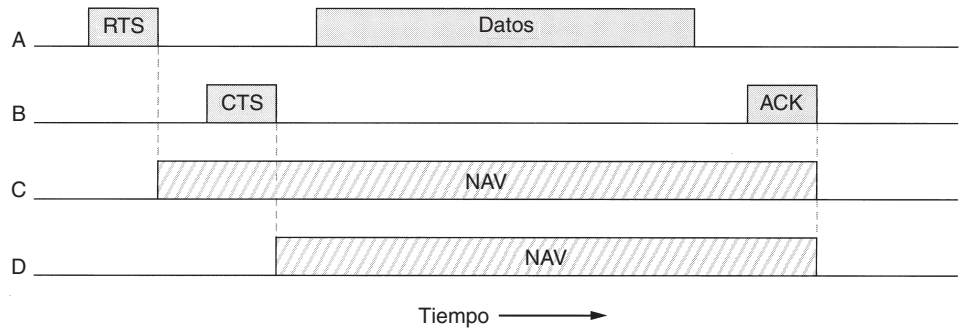


Figura 4-27. El uso de la detección de canal virtual utilizando CSMA/CA.

trama y comienza su temporizador de ACK. Al recibir correctamente la trama de datos, *B* responde con una trama de ACK, con lo que termina el intercambio. Si el temporizador de ACK de *A* termina antes de que el ACK regrese, todo el protocolo se ejecuta de nuevo.

Ahora consideremos este intercambio desde el punto de vista de *C* y *D*. *C* está dentro del alcance de *A*, por lo que podría recibir la trama RTS. Si pasa esto, se da cuenta de que alguien va a enviar datos pronto, así que por el bien de todos desiste de transmitir cualquier cosa hasta que el intercambio esté completo. A partir de la información proporcionada en la solicitud RTS, *C* puede estimar cuánto tardará la secuencia, incluyendo el ACK final, por lo que impone para sí misma un tipo de canal virtual ocupado, indicado por **NAV** (**Vector de Asignación de Red**) en la figura 4-27. *D* no escucha el RTS, pero sí el CTS, por lo que también impone la señal *NAV* para sí misma. Observe que las señales *NAV* no se transmiten; simplemente son recordatorios internos para mantenerse en silencio durante cierto periodo.

En contraste con las redes cableadas, las inalámbricas son ruidosas e inestables, en gran parte debido a los hornos de microondas, que también utilizan las bandas sin licencia ISM. Como consecuencia, la probabilidad de que una trama llegue a su destino se decrementa con la longitud de la trama. Si la probabilidad de que cualquier bit sea erróneo es p, entonces la probabilidad de que una trama de n bits se reciba por completo y correctamente es $(1 - p)^n$. Por ejemplo, para $p = 10^{-4}$, la probabilidad de recibir correctamente una trama Ethernet completa (12,144 bits) es menor que 30%. Si $p = 10^{-5}$, aproximadamente una trama de 9 estará dañada. Incluso si $p = 10^{-6}$, más de 1% de las tramas se dañará, lo que equivale a casi una docena por segundo, y más si se utilizan tramas más cortas que el máximo. En resumen, si una trama es demasiado grande, tiene muy pocas probabilidades de pasar sin daño y probablemente tenga que retransmitirse.

Para solucionar el problema de los canales ruidosos, 802.11 permite dividir las tramas en fragmentos, cada uno con su propia suma de verificación. Cada fragmento se numera de manera individual y su recepción se confirma utilizando un protocolo de parada y espera (es decir, el emisor podría no transmitir fragmentos de $k + 1$ hasta que haya recibido la confirmación de recepción del fragmento k). Una vez que se ha adquirido el canal mediante RTS y CTS, pueden enviarse múltiples

fragmentos en una fila, como se muestra en la figura 4-28. La secuencia de fragmentos se conoce como **ráfaga de fragmentos.**

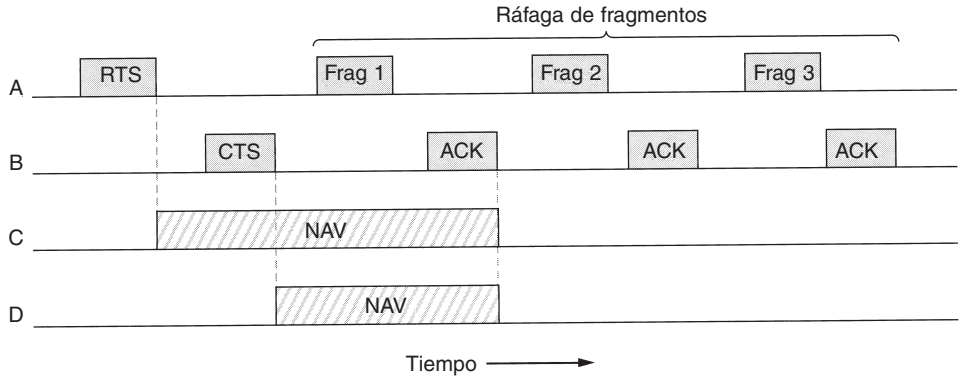


Figura 4-28. Una ráfaga de fragmentos.

La fragmentación incrementa la velocidad real de transporte restringiendo las retransmisiones a los fragmentos erróneos en lugar de la trama completa. El tamaño del fragmento no lo fija el estándar pero es un parámetro de cada celda y la estación base puede ajustarlo. El mecanismo NAV mantiene otras estaciones en silencio sólo hasta la siguiente confirmación de recepción, pero se utiliza otro mecanismo (descrito a continuación) para permitir que otra ráfaga de fragmentos completa se envíe sin interferencia.

Todo el análisis anterior se aplica al modo DCF 802.11. En él, no hay control central y la estación compite por tiempo aire, como en Ethernet. El otro modo permitido es PCF, en el que la estación base sondea las demás estaciones, preguntándoles si tienen tramas que enviar. Puesto que el orden de transmisión se controla por completo por la estación base en el modo PCF, no ocurren colisiones. El estándar prescribe el mecanismo para sondeo, pero no la frecuencia del sondeo, el orden del sondeo, ni el hecho de que las demás estaciones necesiten obtener un servicio igual.

El mecanismo básico consiste en que la estación base difunda una **trama de beacon** (trama guía o faro) de manera periódica (de 10 a 100 veces por segundo). Esta trama contiene parámetros de sistema, como secuencias de salto y tiempos de permanencia (para FHSS), sincronización de reloj, etcétera. También invita a las nuevas estaciones a suscribirse al servicio de sondeo. Una vez que una estación se inscribe para el servicio de sondeo a cierta tasa, se le garantiza de manera efectiva cierta fracción de ancho de banda, y se hace posible proporcionar garantías de calidad de servicio.

La duración de la batería siempre es un problema en los dispositivos inalámbricos móviles, por lo que 802.11 pone atención al asunto de la administración de energía. En particular, una estación base puede conducir una estación móvil al estado de hibernación hasta que dicha estación base o el usuario la saquen de él de manera explícita. Sin embargo, el hecho de indicar a una estación que entre en estado de hibernación significa que la estación base tiene la responsabilidad de almacenar en el búfer las tramas que vayan dirigidas a ella mientras la estación móvil esté hibernando. Posteriormente, esas tramas pueden colectarse.

PCF y DCF pueden coexistir dentro de una celda. Al principio podría parecer imposible tener control central y distribuido funcionando al mismo tiempo, pero 802.11 proporciona una forma de

alcanzar este objetivo. Funciona definiendo cuidadosamente el intervalo de tiempo entre tramas. Después de que se ha enviado una trama, se necesita cierta cantidad de tiempo muerto antes de que cualquier estación pueda enviar una trama. Se definen cuatro intervalos diferentes, cada uno con un propósito específico. Estos intervalos se describen en la figura 4-29.

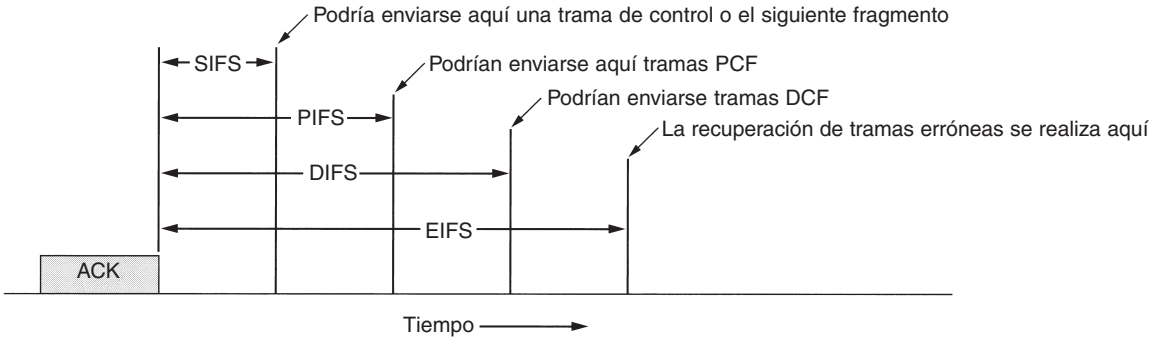


Figura 4-29. Espaciado entre tramas 802.11.

El intervalo más corto es **SIFS (Espaciado Corto Entre Tramas).** Se utiliza para permitir que las distintas partes de un diálogo transmitan primero. Esto incluye dejar que el receptor envíe un CTS para responder a una RTS, dejar que el receptor envíe un ACK para un fragmento o una trama con todos los datos y dejar que el emisor de una ráfaga de fragmentos transmita el siguiente fragmento sin tener que enviar una RTS nuevamente.

Siempre hay una sola estación que debe responder después de un intervalo SIFS. Si falla al utilizar su oportunidad y transcurre un tiempo **PIFS (Espaciado Entre Tramas PCF),** la estación base podría enviar una trama de *beacon* o una trama de sondeo. Este mecanismo permite que una estación base envíe una trama de datos o una secuencia de fragmentos para finalizar su trama sin que nadie interfiera, pero le da a la estación base la oportunidad de tomar el canal cuando el emisor anterior haya terminado, sin tener que competir con usuarios ansiosos.

Si la estación base no tiene nada que decir y transcurre un tiempo **DIFS (Espaciado Entre Tramas DCF),** cualquier estación podría intentar adquirir el canal para enviar una nueva trama. Se aplican las reglas de contención normales, y si ocurre una colisión, podría necesitarse el retroceso exponencial binario.

Sólo una estación que acaba de recibir una trama errónea o desconocida utiliza el último intervalo de tiempo, **EIFS (Espaciado Entre Tramas Extendido),** para reportar la trama errónea. La idea de dar a este evento la menor prioridad es que debido a que el receptor tal vez no tenga idea de lo que está pasando, debe esperar un tiempo considerable para evitar interferir con un diálogo en curso entre las dos estaciones.

4.4.4 La estructura de trama 802.11

El estándar 802.11 define tres clases diferentes de tramas en el cable: de datos, de control y de administración. Cada una de ellas tiene un encabezado con una variedad de campos utilizados

dentro de la subcapa MAC. Además, hay algunos encabezados utilizados por la capa física, pero éstos tienen que ver en su mayor parte con las técnicas de modulación utilizadas, por lo que no las trataremos aquí.

En la figura 4-30 se muestra el formato de la trama de datos. Primero está el campo de *Control de trama*. Éste tiene 11 subcampos. El primero es la *Versión de protocolo*, que permite que dos versiones del protocolo funcionen al mismo tiempo en la misma celda. Después están los campos de *Tipo* (de datos, de control o de administración) y de *Subtipo* (por ejemplo, RTS o CTS). Los bits *A DS* y *De DS* indican que la trama va hacia o viene del sistema de distribución entre celdas (por ejemplo, Ethernet). El bit *MF* indica que siguen más fragmentos. El bit *Retrans.* marca una retransmisión de una trama que se envió anteriormente. El bit de *Administración de energía* es utilizado por la estación base para poner al receptor en estado de hibernación o sacarlo de tal estado. El bit *Más* indica que el emisor tiene tramas adicionales para el receptor. El bit *W* especifica que el cuerpo de la trama se ha codificado utilizando el algoritmo **WEP (Privacidad Inalámbrica Equivalente).** Por último, el bit *O* indica al receptor que una secuencia de tramas que tenga este bit encendido debe procesarse en orden estricto.

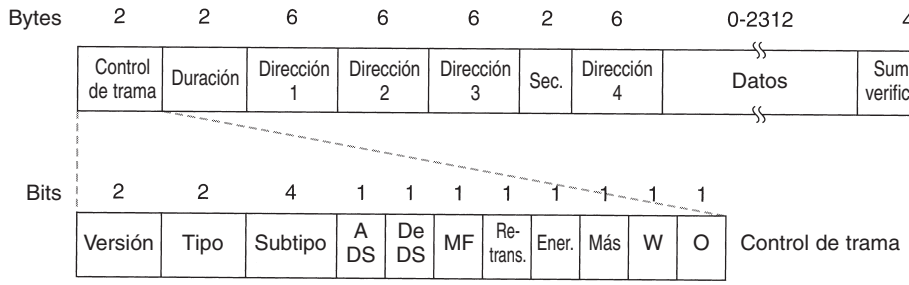


Figura 4-30. La trama de datos 802.11.

El segundo campo de la trama de datos, el de *Duración*, indica cuánto tiempo ocuparán el canal la trama y su confirmación de recepción. Este campo también está presente en las tramas de control y es la forma mediante la cual otras estaciones manejan el mecanismo NAV. El encabezado de trama contiene cuatro direcciones, todas en formato estándar IEEE 802. Obviamente se necesitan el origen y el destino, pero, ¿para qué son las otras dos? Recuerde que las tramas podrían entrar o dejar una celda a través de una estación base. Las otras dos direcciones se utilizan para las estaciones base de origen y destino para el tráfico entre celdas.

El campo de *Secuencia* permite que se numeren los fragmentos. De los 16 bits disponibles, 12 identifican la trama y 4 el fragmento. El campo de *Datos* contiene la carga útil, hasta 2312 bytes, y le sigue el campo común de *Suma de verificación*.

Las tramas de administración tienen un formato similar al de las tramas de datos, excepto que no tienen una de las direcciones de la estación base, debido a que las tramas de administración se restringen a una sola celda. Las tramas de control son más cortas; tienen una o dos direcciones, y no tienen ni campo de *Datos* ni de *Secuencia*. La información clave aquí se encuentra en el campo de *Subtipo*, que por lo general es RTS, CTS o ACK.

4.4.5 Servicios

El estándar 802.11 afirma que cada LAN inalámbrica que se apegue a él debe proporcionar nueve servicios. Éstos se dividen en dos categorías: cinco servicios de distribución y cuatro de estación. Los servicios de distribución se relacionan con la administración de membresías dentro de la celda y con la interacción con estaciones que están fuera de la celda. En contraste, los servicios de estación se relacionan con la actividad dentro de una sola celda.

Los cinco servicios de distribución son proporcionados por las estaciones base y tienen que ver con la movilidad de la estación conforme entran y salen de las celdas, conectándose ellos mismos a las estaciones base y separándose ellos mismos de dichas estaciones. Estos servicios son los siguientes:

1. **Asociación.** Este servicio es utilizado por las estaciones móviles para conectarse ellas mismas a las estaciones base. Por lo general, se utiliza después de que una estación se mueve dentro del alcance de radio de la estación base. Una vez que llega, anuncia su identidad y sus capacidades. Éstas incluyen las tasas de datos soportadas, necesarias para los servicios PCF (es decir, el sondeo), y los requerimientos de administración de energía. La estación base podría aceptar o rechazar la estación móvil. Si se acepta, dicha estación debe autenticarse.

2. **Disociación.** Es posible que la estación o la estación base se disocie, con lo que se rompería la relación. Una estación podría utilizar este servicio antes de apagarse o de salir, pero la estación base también podría utilizarlo antes de su mantenimiento.

3. **Reasociación.** Una estación podría cambiar su estación base preferida mediante este servicio. Esta capacidad es útil para estaciones móviles que se mueven de una celda a otra. Si se utiliza correctamente, no se perderán datos como consecuencia del cambio de estación base (*handover*). (Pero 802.11, al igual que Ethernet, es sólo un servicio de mejor esfuerzo.)

4. **Distribución.** Este servicio determina cómo enrutar tramas enviadas a la estación base. Si el destino es local para la estación base, las tramas pueden enviarse directamente a través del aire. De lo contrario, tendrán que reenviarse a través de la red cableada.

5. **Integración.** Si una trama necesita enviarse a través de una red no 802.11 con un esquema de direccionamiento o formato de trama diferentes, este servicio maneja la traducción del formato 802.11 al requerido por la red de destino.

Los cuatro servicios restantes son dentro de las celdas (es decir, se relacionan con acciones dentro de una sola celda). Se utilizan después de que ha ocurrido la asociación y son las siguientes:

1. **Autenticación.** Debido a que las estaciones no autorizadas pueden recibir o enviar con facilidad la comunicación inalámbrica, una estación debe autenticarse antes de que se le permita enviar datos. Una vez que la estación base asocia una estación móvil (es decir, la ha aceptado en su celda), le envía una trama especial de desafío para ver si dicha estación móvil sabe la clave secreta (contraseña) que se le ha asignado. La estación móvil prueba que sabe la clave secreta codificando la trama de desafío y regresándola a la estación base. Si el resultado es correcto, la estación móvil se vuelve miembro de la celda. En el estándar inicial, la estación base no tiene que probar su identidad a la estación móvil, pero se está realizando trabajo para reparar este defecto en el estándar.

2. **Desautenticación.** Cuando una estación previamente autenticada desea abandonar la red, se desautentica. Después de esto, tal vez ya no utilice la red.

3. **Privacidad.** Para que la información que se envía a través de una LAN inalámbrica se mantenga confidencial, debe codificarse. Este servicio maneja la codificación y la decodificación. El algoritmo de codificación especificado es RC4, inventado por Ronald Rivest del M.I.T.

4. **Entrega de datos.** Por último, la transmisión de datos es la parte esencial, por lo que el 802.11 naturalmente proporciona una forma de transmitir y recibir datos. Puesto que el 802.11 está basado en Ethernet y no se garantiza que la transmisión a través de Ethernet sea 100% confiable, tampoco se garantiza que la transmisión a través del 802.11 sea confiable. Las capas superiores deben tratar con la detección y la corrección de errores.

Una celda 802.11 tiene algunos parámetros que pueden inspeccionarse y, en algunos casos, ajustarse. Se relacionan con la codificación, intervalos de expiración de temporizador, tasas de datos, frecuencia de la trama de *beacon*, etcétera.

Las LANs inalámbricas basadas en 802.11 se están comenzando a distribuir en edificios de oficinas, aeropuertos, hoteles, restaurantes y universidades de todo el mundo. Se espera un crecimiento rápido. Para obtener información adicional acerca de la distribución extendida de 802.11 en CMU, vea (Hills, 2001).

4.5 BANDA ANCHA INALÁMBRICA

Hemos estado en casa mucho tiempo. Salgamos y veamos si hay algo interesante sobre redes por ahí. Pues sucede que hay bastantes novedades, y algunas de ellas tienen que ver con una característica llamada última milla. Con la desregulación del sistema telefónico en muchos países, en la actualidad a los competidores de la compañía telefónica arraigada con frecuencia se les permite ofrecer voz local y servicio de alta velocidad de Internet. Ciertamente hay mucha demanda. El problema es que el tendido de fibra óptica, cable coaxial o incluso cable de par

trenzado categoría 5 a millones de casas y oficinas es extremadamente costoso. ¿Qué debe hacer un competidor?

La respuesta es la banda ancha inalámbrica. Construir una antena enorme en una colina en las afueras del pueblo e instalar antenas que se dirijan a dicha antena en los techos de los clientes es más fácil y barato que cavar zanjas y ensartar cables. Por lo tanto, las compañías de telecomunicación en competencia tienen mucho interés en proporcionar un servicio de comunicación inalámbrica de multimegabits para voz, Internet, películas bajo demanda, etcétera. Como vimos en la figura 2-30, los LMDS se inventaron para este propósito. Sin embargo, hasta hace poco, cada portadora diseñaba su propio sistema. Esta falta de estándares significaba que el hardware y software no se podía producir en masa, por lo que los precios eran altos y la aceptación, baja.

Muchas personas en la industria se dieron cuenta de que tener un estándar de banda ancha inalámbrica era el elemento clave que faltaba, por lo que se le pidió a IEEE que formara un comité compuesto de personas de compañías clave y de academias para redactar el estándar. El siguiente número disponible en el espacio de numeración 802 era **802.16,** por lo que el estándar obtuvo este número. El trabajo se inició en julio de 1999, y el estándar final se aprobó en abril de 2002. Oficialmente el estándar se llama "Air Interface for Fixed Broadband Wireless Access Systems" (Interfaz de Aire para Sistemas Fijos de Acceso Inalámbrico de Banda Ancha). Sin embargo, algunas personas prefieren llamarlo **MAN (red de área metropolitana) inalámbrica** o **circuito local inalámbrico.** Nos referiremos a estos términos de manera indistinta.

Al igual que los otros estándares 802, el 802.16 estuvo influido fuertemente por el modelo OSI, incluyendo las (sub)capas, terminología, primitivas de servicios y más. Desgraciadamente, al igual que OSI, es muy complicado. En las siguientes secciones daremos una breve descripción de algunos de los puntos de importancia del estándar 802.16, pero este tratado no es completo y no trata muchos detalles. Para mayor información acerca de la banda ancha inalámbrica, vea (Bolcskei y cols., 2001, y Webb, 2001). Para mayor información sobre el estándar 802.16 en particular, vea (Eklund y cols., 2002).

4.5.1 Comparación entre los estándares 802.11 y 802.16

En este punto tal vez piense: ¿Por qué diseñar un nuevo estándar? ¿Por qué no simplemente utilizar 802.11? Hay algunas buenas razones para no utilizar 802.11, principalmente porque 802.11 y 802.16 resuelven diferentes problemas. Antes de introducirnos en la tecnología de 802.16, probablemente valga la pena dar algunos detalles de por qué es necesario un estándar completamente nuevo.

Los entornos en los que funcionan 802.11 y 802.16 son similares en algunas formas, principalmente en que fueron diseñados para proporcionar comunicaciones inalámbricas de alto ancho de banda. Pero también difieren en aspectos muy importantes. Para empezar, el protocolo 802.16 proporciona servicio a edificios, y éstos no son móviles. No migran de celda a celda con frecuencia. La mayor parte de 802.11 tiene que ver con la movilidad, y nada de eso es relevante aquí. Además, los edificios pueden tener más de una computadora en ellos, lo cual no ocurre cuando la estación final es una sola computadora portátil.

Debido a que los dueños de edificios por lo general están dispuestos a gastar mucho más dinero en artículos de comunicación que los dueños de computadoras portátiles, hay mejores radios disponibles. Esta diferencia significa que 802.16 puede utilizar comunicación de dúplex total, algo que 802.11 evita para mantener bajo el costo de los radios.

Puesto que el estándar 802.16 se usa en parte de la ciudad, las distancias involucradas pueden ser de varios kilómetros, lo que significa que la energía detectada en la estación base puede variar considerablemente de estación en estación. Esta variación afecta la relación señal a ruido, que, a su vez, fija múltiples esquemas de modulación. Además, la comunicación abierta a través de la ciudad significa que la seguridad y privacidad son esenciales y obligatorias.

Además, es probable que cada celda tenga muchos más usuarios que una celda 802.11 típica, y se espera que estos usuarios utilicen más ancho de banda que un usuario 802.11 típico. Después de todo es raro que una compañía reúna a 50 empleados con sus computadoras portátiles en un cuarto para ver si pueden saturar la red inalámbrica 802.11 al observar a la vez 50 películas por separado. Por esta razón es necesario más espectro del que las bandas ISM pueden proporcionar, con lo que se obliga al estándar 802.16 a funcionar en el rango de frecuencia más alto de 10 a 66 GHz, el único lugar en el que el espectro no utilizado aún está disponible.

Pero estas ondas milimétricas tienen propiedades físicas diferentes que las ondas más largas en las bandas ISM, que a su vez requieren una capa física completamente diferente. Una propiedad de las ondas milimétricas es que el agua (especialmente la lluvia y, en cierta medida, la nieve, el granizo y, con un poco de mala suerte, la niebla espesa) las absorbe por completo. En consecuencia, el control de errores es más importante que en un entorno interno. Las ondas milimétricas pueden enfocarse en rayos direccionales (802.11 es omnidireccional), por lo que las opciones realizadas en 802.11 relacionadas con la propagación de múltiples rutas son debatibles.

Otro aspecto es la calidad del servicio. Si bien el estándar 802.11 proporciona soporte para el tráfico en tiempo real (utilizando el modo PCF), realmente no se diseñó para uso extenso de telefonía y multimedia. En contraste, se espera que el estándar 802.16 soporte estas aplicaciones por completo porque está diseñado para uso residencial y de negocios.

En resumen, 802.11 se diseñó para ser una Ethernet móvil, mientras que el estándar 802.16 se diseñó para ser televisión por cable inalámbrica, pero estacionaria. Estas diferencias son tan grandes que los estándares resultantes son muy diferentes debido a que tratan de optimizar cosas distintas.

Vale la pena hacer una pequeña comparación con el sistema de teléfonos celulares. Al referirnos a los teléfonos celulares, hablamos de estaciones móviles de banda estrecha, baja potencia y con orientación a voz que se comunican mediante microondas de longitud media. Nadie ve (todavía) películas de 2 horas a alta resolución en teléfonos móviles GSM. Incluso UMTS tiene poca esperanza de cambiar esta situación. En resumen, el mundo de las MANs inalámbricas es más demandante que el de los teléfonos móviles, por lo que se necesita un sistema completamente diferente. El hecho de que en el futuro el estándar 802.16 pueda utilizarse para dispositivos móviles es una pregunta interesante. No fue optimizado para ellos, pero la posibilidad está abierta. Por el momento se enfoca en los sistemas inalámbricos fijos.

4.5.2 La pila de protocolos del estándar 802.16

En la figura 4-31 se ilustra la pila de protocolos del estándar 802.16. La estructura general es similar a la de las otras redes 802, pero con más subcapas. La subcapa inferior tiene que ver con la transmisión. El radio de banda estrecha tradicional se utiliza con esquemas de modulación tradicionales. Arriba de la capa de transmisión física está una subcapa de convergencia para ocultarle las diferentes tecnologías a la capa de enlace de datos. En la actualidad el estándar 802.11 también tiene algo parecido a esto, sólo que el comité eligió no formalizarlo con un nombre de tipo OSI.

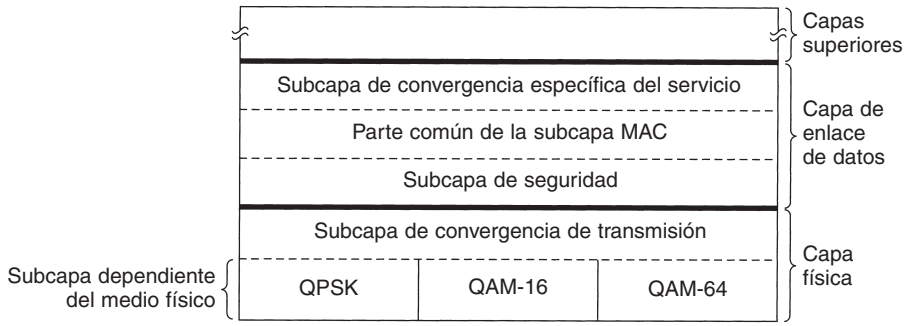


Figura 4-31. La pila de protocolos del 802.16.

Aunque no los mostramos en la figura, se está trabajando para agregar dos nuevos protocolos de capa física. El estándar 802.16a soportará a OFDM en el rango de frecuencia de 2 a 11 GHz. El estándar 802.16b operará en la banda ISM de 5 GHz. Estos dos son intentos para acercarse al estándar 802.11.

La capa de enlace de datos consta de tres subcapas. La inferior tiene que ver con la privacidad y seguridad, lo cual es más importante para las redes externas públicas que para las redes internas privadas. Maneja codificación, decodificación y administración de claves.

A continuación se encuentra la parte común de la subcapa MAC. Es aquí donde se encuentran los principales protocolos, como la administración de canal. El modelo consiste en que la estación base controla el sistema. Puede calendarizar de manera muy eficiente los canales de flujo descendente (es decir, de la estación base al suscriptor) y es muy importante en el manejo de los canales ascendentes (es decir, del suscriptor a la estación base). Una característica no muy común de la subcapa MAC es que, a diferencia de las subcapas de las otras redes 802, es completamente orientada a la conexión, para proporcionar garantías de calidad del servicio para la comunicación de telefonía y multimedia.

En los otros protocolos 802, la subcapa de convergencia específica del servicio toma el lugar de la subcapa de enlace lógico. Su función es interactuar con la capa de red. Un problema aquí es que el estándar 802.16 fue diseñado para integrarse sin ningún problema tanto con los protocolos

de datagramas (por ejemplo, PPP, IP y Ethernet) como con ATM. El problema es que los protocolos de paquetes no son orientados a la conexión y ATM sí lo es. Esto significa que cada conexión ATM se tiene que asignar a una conexión 802.16, que en principio es un asunto directo. ¿Pero en cuál conexión 802.16 debe asignarse un paquete IP entrante? El problema se soluciona en esta subcapa.

4.5.3 La capa física del estándar 802.16

Como se mencionó anteriormente, la banda ancha inalámbrica necesita mucho espectro y el único lugar para encontrarlo es en el rango de 10 a 66 GHz. Estas ondas milimétricas tienen una propiedad interesante que las microondas más largas no tienen: viajan en líneas rectas, a diferencia del sonido, pero en forma similar a la luz. Como consecuencia, la estación base puede tener múltiples antenas, cada una apuntando a un sector diferente del terreno circundante, como se muestra en la figura 4-32. Cada sector tiene sus propios usuarios y es completamente independiente de los sectores adyacentes, algo que no es verdad es el radio celular, el cual es omnidireccional.

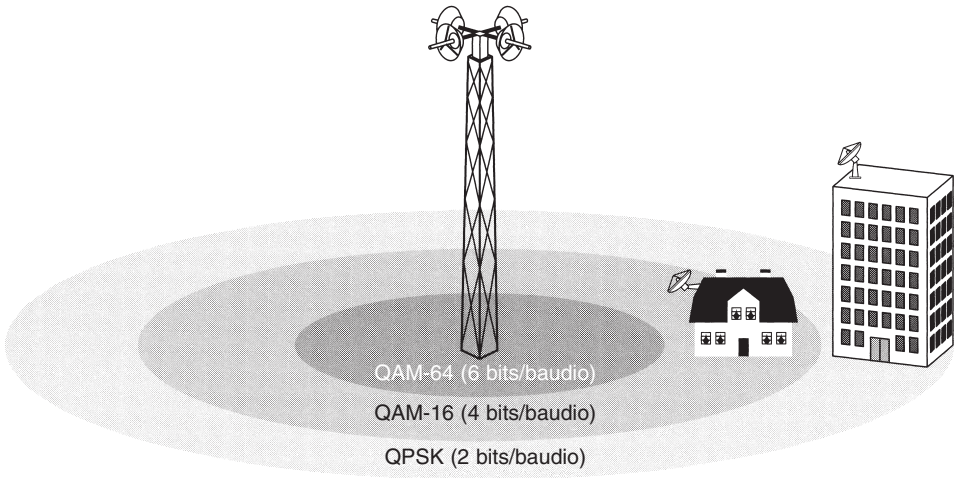


Figura 4-32. El entorno de transmisión 802.16.

Debido a que la fuerza de señal en la banda milimétrica desciende drásticamente con la distancia a partir de la estación base, la relación señal a ruido también desciende con la distancia a partir de la estación base. Por esta razón, el estándar 802.16 emplea tres esquemas de modulación diferentes, dependiendo de la distancia entre la estación suscriptora y la estación base. Para suscriptores cercanos se utiliza QAM-64, con 6 bits/baudio. Para suscriptores a distancias medias se utiliza QAM-16, con 4 bits/baudio. Para suscriptores distantes se utiliza QPSK, con 2 bits/baudio. Por ejemplo, para un valor típico de 25 MHz digno de espectro, QAM-64 da 150 Mbps, QAM-16 da 100 Mbps y QPSK da 50 Mbps. En otras palabras, entre más lejos esté el suscriptor de la estación base, será más baja la tasa de datos (algo similar a lo que vimos con ADSL en la figura 2-27). El diagrama de constelación de estas tres técnicas de modulación se mostró en la figura 2-25.

Dado el objetivo de producir un sistema de banda ancha, sujeto a las limitantes físicas mostradas anteriormente, los diseñadores del protocolo 802.16 trabajaron duro para utilizar eficientemente el espectro disponible. Los que no les gustaba era la forma en que funcionaban GSM y DAMPS. Ambos utilizan bandas de frecuencia diferentes pero iguales para el tráfico ascendente y descendente. Para voz, es probable que el tráfico sea simétrico en su mayor parte, pero para acceso a Internet por lo general hay más tráfico descendente que ascendente. En consecuencia, el estándar 802.16 proporciona una forma más flexible para asignar el ancho de banda. Se utilizan dos esquemas: **FDD (Duplexación por División de Frecuencia)** y **TDD (Duplexación por División de Tiempo).** Este último se ilustra en la figura 4-33. Aquí la estación base envía tramas periódicamente. Cada trama contiene ranuras de tiempo. Las primeras son para el tráfico descendente. Después se encuentra el tiempo de protección o guarda, el cual es utilizado por las estaciones para cambiar de dirección. Por último, están las ranuras para el tráfico ascendente. El número de ranuras de tiempo dedicadas para cada dirección se puede cambiar de manera dinámica con el fin de que el ancho de banda en cada dirección coincida con el tráfico.

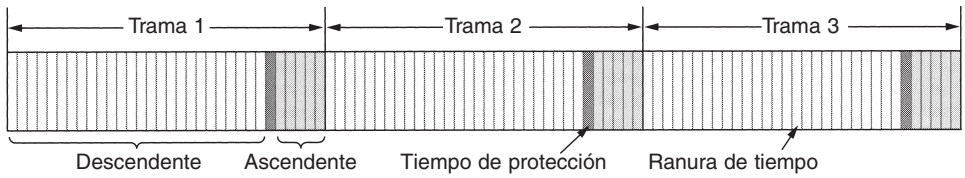


Figura 4-33. Tramas y ranuras de tiempo para duplexación por división de tiempo.

La estación base asigna el tráfico descendente en ranuras de tiempo. Además, controla por completo esta dirección. El tráfico ascendente es más complejo y depende de la calidad del servicio requerido. Más adelante volveremos a la asignación de ranuras, cuando analicemos la subcapa MAC.

Otra característica interesante de la capa física es su capacidad de empaquetar múltiples tramas MAC consecutivas en una sola transmisión física. Esta característica mejora la eficiencia espectral al reducir el número de preámbulos y encabezados de capa física necesarios.

Otro aspecto que vale la pena mencionar es el uso de los códigos de Hamming para realizar corrección de errores hacia delante en la capa física. La mayoría de las otras redes se basa simplemente en sumas de verificación para detectar errores y solicitar retransmisiones cuando se reciben tramas erróneas. Pero en el entorno de banda ancha de área amplia, se esperan tantos errores de transmisión que la corrección de errores se emplea en la capa física, además de sumas de verificación en las capas superiores. El objetivo de la corrección de errores es hacer que el canal luzca mejor de lo que realmente es (de la misma forma en que los CD-ROMs parecen ser muy confiables, pero sólo porque más de la mitad de los bits se destinan para la corrección de errores en la capa física).

4.5.4 El protocolo de la subcapa MAC del 802.16

La capa de enlace de datos se divide en tres subcapas, como vimos en la figura 4-31. Debido a que estudiaremos la criptografía hasta el capítulo 8 es difícil explicar ahora cómo funciona la

subcapa de seguridad. Basta decir que la codificación se utiliza para que todos los datos transmitidos se mantengan en secreto. Sólo las cargas útiles de las tramas se codifican; los encabezados no se codifican. Esta propiedad significa que un fisgón puede ver quién está hablándole a quién pero no puede saber qué se están diciendo.

Si usted ya sabe algo sobre criptografía, a continuación se muestra una explicación de un párrafo acerca de la subcapa de seguridad. Si no sabe nada sobre criptografía, es probable que el siguiente párrafo no sea muy ilustrativo para usted (pero podría considerar volver a leerlo después de terminar el capítulo 8).

Cuando un suscriptor se conecta a una estación base, realiza autenticación mutua con criptografía de clave pública RSA mediante certificados X.509. Las cargas útiles mismas se codifican mediante un sistema de clave simétrica, ya sea DES con cambio de bloques de código o triple DES con dos claves. Es probable que AES (Rijndael) se agregue pronto. La verificación de integridad utiliza SHA-1. Eso no es tan malo, ¿o sí?

Ahora veamos la parte común de la subcapa MAC. Las tramas MAC ocupan un número integral de ranuras de tiempo de la capa física. Cada trama se compone de subtramas, de las cuales las primeras dos son los mapas descendente y ascendente. Éstos indican lo que hay en cada ranura de tiempo y cuáles ranuras de tiempo están libres. El mapa descendente también contiene varios parámetros de sistema para informar de nuevas estaciones conforme entran en línea.

El canal descendente es muy directo. La estación base decide simplemente lo que se va a poner en cada subtrama. El canal ascendente es más complicado debido a que hay suscriptores no coordinados compitiendo por él. Su asignación está estrechamente relacionada con el aspecto de calidad del servicio. Hay cuatro clases de servicio definidas:

1. Servicio de tasa de bits constante.
2. Servicio de tasa de bits variable en tiempo real.
3. Servicio de tasa de bits variable no en tiempo real.
4. Servicio de mejor esfuerzo.

Todos los servicios del estándar 802.16 son orientados a la conexión, y cada conexión toma una de las clases de servicio mostradas anteriormente, que se determina cuando se configura la conexión. Este diseño es muy diferente al de 802.11 o al de Ethernet, los cuales no tienen conexiones en la subcapa MAC.

El servicio de tasa de bits constante está diseñado para transmitir voz descomprimida, como en un canal T1. Este servicio necesita enviar una cantidad predeterminada de datos en intervalos de tiempo predeterminados. Se aloja mediante la dedicación de ciertas ranuras de tiempo a cada conexión de este tipo. Una vez que se ha asignado el ancho de banda, las ranuras de tiempo quedan disponibles automáticamente, sin necesidad de solicitar cada una.

El servicio de tasa de bits variable en tiempo real está destinado para la multimedia comprimida y otras aplicaciones en tiempo real en las que la cantidad de ancho de banda necesaria puede variar en cada instante. Es ajustada por la estación base sondeando al suscriptor a un intervalo fijo para saber cuánto ancho de banda se necesita esta vez.

El servicio de tasa de bits variable en tiempo no real es para las transmisiones pesadas que no son en tiempo real, como transmisiones grandes de archivos. Para este servicio, la estación base sondea al suscriptor con mucha frecuencia. Un cliente de tasa de bits constante puede establecer un bit en una de sus tramas, solicitando un sondeo para enviar tráfico adicional (tasa de bits variable).

Si una estación no responde a un sondeo *k* veces en una fila, la estación base la coloca en un grupo de multidifusión y elimina su sondeo personal. En su lugar, cuando se sondea el grupo de multidifusión, cualquiera de las estaciones que conforman el grupo puede responder, compitiendo por el servicio. De esta forma, las estaciones con poco tráfico no desperdician sondeos valiosos.

Por último, el servicio de mejor esfuerzo es para todo lo demás. No se realiza sondeo y el suscriptor debe competir por ancho de banda con otros suscriptores de mejor servicio. Las solicitudes por ancho de banda se realizan en ranuras de tiempo que están marcadas en el mapa ascendente como disponibles para competencia. Si una solicitud es exitosa, su éxito se notará en el siguiente mapa de bits descendente. Si no es exitosa, los suscriptores no exitosos deberán tratar más tarde. Para minimizar las colisiones, se utiliza el algoritmo de retroceso exponencial binario.

El estándar define dos formas de asignación de ancho de banda: por estación y por conexión. En el primer caso, la estación suscriptora agrega las necesidades de todos los usuarios del edificio y realiza solicitudes colectivas por ellos. Cuando se le concede el ancho de banda, lo asigna entre sus usuarios como considere necesario. En el último caso, la estación base administra cada conexión de manera directa.

4.5.5 La estructura de trama 802.16

Todas las tramas MAC comienzan con un encabezado genérico. A éste le sigue una carga útil y una suma de verificación (CRC) opcionales, como se ilustra en la figura 4-34. La carga útil no es necesaria en las tramas de control, por ejemplo, en las que solicitan ranuras de canal. La suma de verificación también es (sorprendentemente) opcional, debido a la corrección de errores en la capa física y al hecho de que nunca se realiza un intento por retransmitir tramas en tiempo real. Si estos intentos nunca ocurren, ¿para qué molestarse con una suma de verificación?

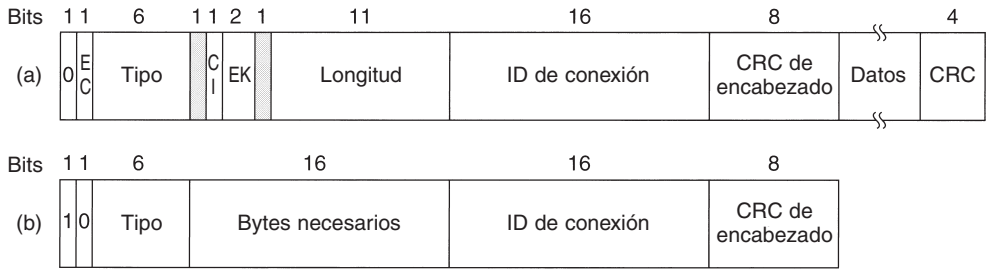


Figura 4-34. (a) Una trama genérica. (b) Una trama de solicitud de ancho de banda.

Un rápido vistazo a los campos de encabezado de la figura 4-34(a) es como sigue: el bit *EC* indica si la carga útil está encriptada. El campo *Tipo* identifica el tipo de la trama e indica principalmente si hay empaquetamiento y fragmentación. El campo *CI* indica la presencia o ausencia de la suma de verificación final. El campo *EK* indica cuál de las claves de encriptación se está utilizando (si es que se está utilizando alguna). El campo *Longitud* proporciona la longitud exacta de la trama, incluyendo la del encabezado. El *Identificador de conexión* indica a cuál conexión pertenece esta trama. Por último, el campo *CRC de encabezado* es la suma de verificación sólo del encabezado, que utiliza el polinomio $x^8 + x^2 + x + 1$.

En la figura 4-34(b) se muestra un segundo tipo de encabezado, para tramas que solicitan ancho de banda. Comienza con un bit 1 en lugar de uno 0 y es similar al encabezado genérico, excepto que el segundo y tercer bytes forman un número de 16 bits, lo que indica la cantidad de ancho de banda necesaria para transmitir el número de bytes especificados. Las tramas de solicitud de ancho de banda no transmiten datos útiles o un CRC de la trama completa.

Es posible decir muchísimas cosas más sobre el estándar 802.16, pero este no es el lugar para decirlo. Para mayor información, consulte el estándar mismo.

4.6 BLUETOOTH

En 1994, la empresa L. M. Ericsson se interesó en conectar sus teléfonos móviles y otros dispositivos (por ejemplo, PDAs,) sin necesidad de cables. En conjunto con otras cuatro empresas (IBM, Intel, Nokia y Toshiba), formó un SIG (grupo de interés especial, es decir, un consorcio) con el propósito de desarrollar un estándar inalámbrico para interconectar computadoras, dispositivos de comunicaciones y accesorios a través de radios inalámbricos de bajo consumo de energía, corto alcance y económicos. Al proyecto se le asignó el nombre **Bluetooth,** en honor de Harald Blaatand (Bluetooth) II (940-981), un rey vikingo que unificó (es decir, conquistó) Dinamarca y Noruega, también sin necesidad de cables. Aunque la idea original eran tan sólo prescindir de cables entre dispositivos, su alcance se expandió rápidamente al área de las LANs inalámbricas. Aunque esta expansión le dio más utilidad al estándar, también provocó el surgimiento de competencia con el 802.11. Para empeorar las cosas, los dos sistemas interfieren entre sí en el ámbito eléctrico. Asimismo, vale la pena hacer notar que Hewlett-Packard introdujo hace algunos años una red infrarroja para conectar periféricos de computadora sin cables, pero en realidad nunca alcanzó popularidad.

Sin desanimarse por esto, el SIG de Bluetooth emitió en julio de 1999 una especificación de 1500 páginas acerca de V1.0. Un poco después, el grupo de estándares del IEEE que se encarga de las redes de área personal inalámbricas, 802.15, adoptó como base el documento sobre Bluetooth y empezó a trabajar en él. A pesar de que podría parecer extraño estandarizar algo que ya cuenta con una especificación bien detallada, sin implementaciones incompatibles que tengan que armonizarse, la historia demuestra que al existir un estándar abierto manejado por un cuerpo neutral como el IEEE con frecuencia se estimula el uso de una tecnología. Para ser un poco más precisos, debe apuntarse que la especificación de Bluetooth está dirigida a un sistema completo, de la capa física a la capa de aplicación. El comité 802.15 del IEEE estandariza solamente las capas física y la de enlace de datos; el resto de la pila de protocolos está fuera de sus estatutos.

Aun cuando el IEEE aprobó en el 2002 el primer estándar para redes de área personal, 802.15.1, el SIG de Bluetooth continúa las mejoras. A pesar de que las versiones del SIG y del IEEE difieren, se espera que en breve coincidirán en un solo estándar.

4.6.1 Arquitectura de Bluetooth

Empecemos nuestro análisis del sistema Bluetooth con un rápido vistazo de sus elementos y de su propósito. La unidad básica de un sistema Bluetooth es una **piconet,** que consta de un nodo maestro y hasta siete nodos esclavos activos a una distancia de 10 metros. En una misma sala (grande) pueden encontrarse varias *piconets* y se pueden conectar mediante un nodo puente, como se muestra en la figura 4-35. Un conjunto de *piconets* interconectadas se denomina **scatternet.**

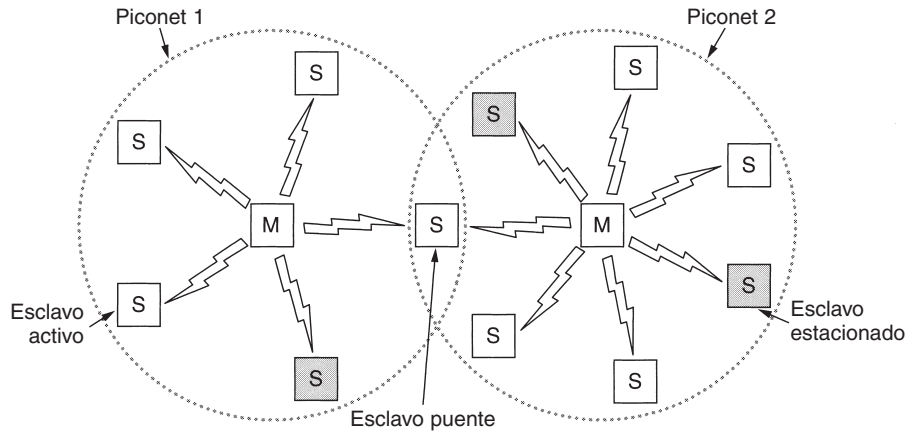


Figura 4-35. Dos piconets se pueden conectar para conformar una scatternet.

Además de los siete nodos esclavos activos de una *piconet*, puede haber hasta 255 nodos estacionados en la red. Éstos son dispositivos que el nodo maestro ha cambiado a un estado de bajo consumo de energía para reducir el desgaste innecesario de sus pilas. Lo único que un dispositivo en estado estacionado puede hacer es responder a una señal de activación por parte del maestro. También hay dos estados intermedios, *hold* y *sniff*, pero no nos ocuparemos de ellos aquí.

La razón para el diseño maestro/esclavo es que los diseñadores pretendían facilitar la implementación de chips Bluetooth completos por debajo de 5 dólares. La consecuencia de esta decisión es que los esclavos son sumamente pasivos y realizan todo lo que los maestros les indican. En esencia, una *piconet* es un sistema TDM centralizado, en el cual el maestro controla el reloj y determina qué dispositivo se comunica en un momento determinado. Todas las comunicaciones se realizan entre el maestro y el esclavo; no existe comunicación directa de esclavo a esclavo.

4.6.2 Aplicaciones de Bluetooth

La mayoría de los protocolos de red sólo proporcionan canales entre las entidades que se comunican y permiten a los diseñadores de aplicaciones averiguar para qué desean utilizarlos. Por ejemplo, el 802.11 no especifica si los usuarios deben utilizar sus computadoras portátiles para leer correo electrónico, navegar por la Red o cualquier otro uso. En contraste, la especificación Bluetooth V1.1 designa el soporte de 13 aplicaciones en particular y proporciona diferentes pilas de protocolos para cada una. Desgraciadamente, este enfoque conlleva una gran complejidad, que aquí omitiremos. En la figura 4-36 se describen las 13 aplicaciones, las cuales se denominan **perfiles.** Al analizarlas brevemente en este momento, veremos con mayor claridad lo que pretende el SIG de Bluetooth.

Nombre	Descripción
Acceso genérico	Procedimientos para el manejo de enlaces
Descubrimiento de servicios	Protocolo para descubrir los servicios que se ofrecen
Puerto serie	Reemplazo para un cable de puerto serie
Intercambio genérico de objetos	Define la relación cliente-servidor para el traslado de objetos
Acceso a LAN	Protocolo entre una computadora móvil y una LAN fija
Acceso telefónico a redes	Permite que una computadora portátil realice una llamada por medio de un teléfono móvil
Fax	Permite que un fax móvil se comunique con un teléfono móvil
Telefonía inalámbrica	Conecta un handset (teléfono) con su estación base local
Intercom (Intercomunicador)	Walkie-talkie digital
Headset (Diadema telefónica)	Posibilita la comunicación de voz sin utilizar las manos
Envío de objetos	Ofrece una manera de intercambiar objetos simples
Transferencia de archivos	Proporciona una característica para transferencia de archivos más general
Sincronización	Permite a un PDA sincronizarse con otra computadora

Figura 4-36. Los perfiles de Bluetooth.

El perfil de acceso genérico no es realmente una aplicación, sino más bien la base sobre la cual se construyen las aplicaciones; su tarea principal es ofrecer una manera para establecer y mantener enlaces (canales) seguros entre el maestro y los esclavos. El perfil de descubrimiento de servicios también es relativamente genérico; los dispositivos lo utilizan para descubrir qué servicios ofrecen otros dispositivos. Se espera que todos los dispositivos Bluetooth implementen estos dos perfiles. Los restantes son opcionales.

El perfil de puerto serie es un protocolo de transporte que la mayoría de los perfiles restantes utiliza. Emula una línea serie y es especialmente útil para aplicaciones heredadas que requieren una línea serie.

El perfil de intercambio genérico define una relación cliente-servidor para el traslado de datos. Los clientes inician operaciones, pero tanto un cliente como un servidor pueden fungir como esclavo. Al igual que el perfil de puerto serie, es la base para otros perfiles.

El siguiente grupo de tres perfiles está destinado a la conectividad. El perfil de acceso a LAN permite a un dispositivo Bluetooth conectarse a una red fija; este perfil es competencia directa del estándar 802.11. El perfil de acceso telefónico a redes fue el propósito original de todo el proyecto; permite a una computadora portátil conectarse a un teléfono móvil que contenga un módem integrado, sin necesidad de cables. El perfil de fax es parecido al de acceso telefónico a redes, excepto que posibilita a máquinas de fax inalámbricas enviar y recibir faxes a través de teléfonos móviles sin que exista una conexión por cable entre ambos.

Los tres perfiles siguientes son para telefonía. El perfil de telefonía inalámbrica proporciona una manera de conectar el *handset* (teléfono) de un teléfono inalámbrico a la estación base. En la actualidad, la mayoría de los teléfonos inalámbricos no se puede utilizar también como teléfonos móviles, pero quizá en el futuro se puedan combinar los teléfonos inalámbricos y los móviles. El perfil intercom hace posible que dos teléfonos se conecten como *walkie-talkies*. Por último, con el perfil *headset* (diadema telefónica) se puede realizar comunicación de voz entre la diadema telefónica y su estación base, por ejemplo, para comunicarse telefónicamente sin necesidad de utilizar las manos al manejar un automóvil.

Los tres perfiles restantes sirven para intercambiar objetos entre dos dispositivos inalámbricos, como tarjetas de presentación, imágenes o archivos de datos. En particular, el propósito del perfil de sincronización es cargar datos en un PDA o en una computadora portátil cuando se está fuera de casa y de recabar estos datos al llegar a casa.

¿Era realmente necesario explicar en detalle todas estas aplicaciones y proporcionar diferentes pilas de protocolos para cada una? Tal vez no, pero esto se debió a que fueron diversos grupos de trabajo los que diseñaron las diferentes partes del estándar y cada uno se enfocó en su problema específico y generó su propio perfil. La ley de Conway podría aplicarse en esta situación. (En el número de abril de 1968 de la revista *Datamation,* Melvin Conway apuntó que si se asignan *n* personas a escribir un compilador, se obtendrá un compilador de *n* pasos, o, en forma más general, la estructura del software reflejará la estructura del grupo que la produjo.) Quizá dos pilas de protocolos habrían sido suficientes en lugar de 13, una para la transferencia de archivos y otra para posibilitar el flujo continuo de la comunicación en tiempo real.

4.6.3 La pila de protocolos de Bluetooth

El estándar Bluetooth cuenta con muchos protocolos agrupados con poco orden en capas. La estructura de capas no sigue el modelo OSI, el modelo TCP/IP, el modelo 802 o algún otro modelo conocido. Sin embargo, el IEEE se encuentra modificando actualmente Bluetooth para ajustarlo al modelo 802. En la figura 4-37 se muestra la arquitectura básica de protocolos de Bluetooth tal como la modificó el comité 802.

La capa inferior es la capa de radio física, la cual es bastante similar a la capa física de los modelos OSI y 802. Se ocupa de la transmisión y la modulación de radio. Aquí, gran parte del interés se enfoca en el objetivo de lograr que el sistema tenga un costo bajo para que pueda entrar al mercado masivo.

La capa de banda base tiene algunos puntos en común con la subcapa MAC, aunque también incluye elementos de la capa física. Se encarga de la manera en que el maestro controla las ranuras de tiempo y de que éstas se agrupen en tramas.

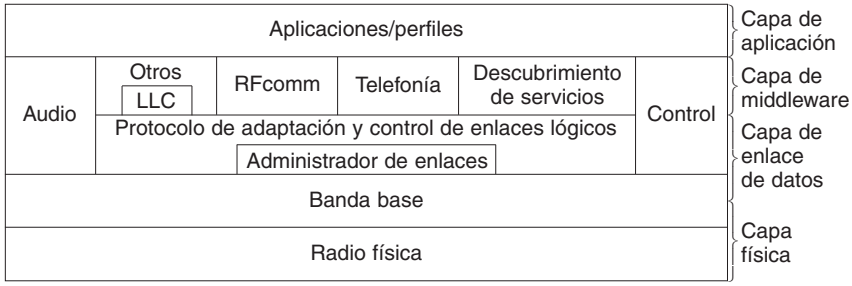


Figura 4-37. Versión 802.15 de la arquitectura de protocolos de Bluetooth.

A continuación se encuentra una capa con un grupo de protocolos un tanto relacionados. El administrador de enlaces se encarga de establecer canales lógicos entre dispositivos, incluyendo administración de energía, autenticación y calidad de servicio. El protocolo de adaptación y control de enlaces lógicos (también conocido como L2CAP) aísla a las capas superiores de los detalles de la transmisión. Es análogo a la subcapa LLC del estándar 802, pero difiere de ésta en el aspecto técnico. Como su nombre indica, los protocolos de audio y control se encargan del audio y el control, respectivamente. Las aplicaciones pueden acceder a ellos de manera directa sin necesidad de pasar por el protocolo L2CAP.

La siguiente capa hacia arriba es la de middleware, que contiene una mezcla de diferentes protocolos. El IEEE incorporó aquí la subcapa LLC del 802 para ofrecer compatibilidad con las redes 802. Los protocolos RFcomm, de telefonía y de descubrimiento de servicios son nativos. RFcomm (comunicación de Radio Frecuencia) es el protocolo que emula el puerto serie estándar de las PCs para la conexión de teclados, ratones y módems, entre otros dispositivos. Su propósito es permitir que dispositivos heredados lo utilicen con facilidad. El protocolo de telefonía es de tiempo real y está destinado a los tres perfiles orientados a voz. También se encarga del establecimiento y terminación de llamadas. Por último, el protocolo de descubrimiento de servicios se emplea para localizar servicios dentro de la red.

En la capa superior es donde se ubican las aplicaciones y los perfiles, que utilizan a los protocolos de las capas inferiores para realizar su trabajo. Cada aplicación tiene su propio subconjunto dedicado de protocolos. Por lo general, los dispositivos específicos, como las diademas telefónicas, contienen únicamente los protocolos necesarios para su aplicación.

En las siguientes secciones examinaremos las tres capas inferiores de la pila de protocolos de Bluetooth, ya que éstas corresponden más o menos con las subcapas física y MAC.

4.6.4 La capa de radio de Bluetooth

La capa de radio traslada los bits del maestro al esclavo, o viceversa. Es un sistema de baja potencia con un rango de 10 metros que opera en la banda ISM de 2.4 GHz. La banda se divide en 79 canales de 1 MHz cada uno. La modulación es por desplazamiento de frecuencia, con 1 bit por Hz, lo cual da una tasa de datos aproximada de 1 Mbps, pero gran parte de este espectro la consume la

sobrecarga. Para asignar los canales de manera equitativa, el espectro de saltos de frecuencia se utiliza a 1600 saltos por segundo y un tiempo de permanencia de 625 μseg. Todos los nodos de una *piconet* saltan de manera simultánea, y el maestro establece la secuencia de salto.

Debido a que tanto el 802.11 como Bluetooth operan en la banda ISM de 2.4 GHz en los mismos 79 canales, interfieren entre sí. Puesto que Bluetooth salta mucho más rápido que el 802.11, es más probable que un dispositivo Bluetooth dañe las transmisiones del 802.11 que en el caso contrario. Como el 802.11 y el 802.15 son estándares del IEEE, éste busca una solución para el problema, aunque no es tan sencilla porque ambos sistemas utilizan la banda ISM por la misma razón: no se requiere licencia para su uso. El estándar 802.11a emplea la otra banda ISM (5 GHz), pero tal estándar tiene un rango mucho más corto que el 802.11b (debido a la física de las ondas de radio), por lo cual el 802.11a no es una solución idónea en todos los casos. Algunas empresas han recurrido a la prohibición total de Bluetooth para solucionar el problema. Una solución de mercado es que la red más potente (en los aspectos político y económico, no eléctrico) solicite que la parte más débil modifique su estándar para dejar de interferir con ella. (Lansford y cols., 2001) dan algunas opiniones al respecto.

4.6.5 La capa de banda base de Bluetooth

La capa de banda base de Bluetooth es lo más parecido a una subcapa MAC. Esta capa convierte el flujo de bits puros en tramas y define algunos formatos clave. En la forma más sencilla, el maestro de cada *piconet* define una serie de ranuras de tiempo de 625 μseg y las transmisiones del maestro empiezan en las ranuras pares, y las de los esclavos, en las ranuras impares. Ésta es la tradicional multiplexión por división de tiempo, en la cual el maestro acapara la mitad de las ranuras y los esclavos comparten la otra mitad. Las tramas pueden tener 1, 3 o 5 ranuras de longitud.

La sincronización de saltos de frecuencia permite un tiempo de asentamiento de 250-260 μseg por salto para que los circuitos de radio se estabilicen. Es posible un asentamiento más rápido, pero a un mayor costo. Para una trama de una sola ranura, después del asentamiento, se desechan 366 de los 625 bits. De éstos, 126 se utilizan para un código de acceso y el encabezado, y 240 para los datos. Cuando se enlazan cinco ranuras, sólo se necesita un periodo de asentamiento y se utiliza uno ligeramente más corto, de tal manera que de los $5 \times 625 = 3125$ bits de las cinco ranuras de tiempo, 2781 se encuentran disponibles para la capa de banda base. Así, las tramas más grandes son mucho más eficientes que las de una sola ranura.

Cada trama se transmite por un canal lógico, llamado **enlace,** entre el maestro y un esclavo. Hay dos tipos de enlaces. El primero es el **ACL** (**Asíncrono no Orientado a la Conexión**), que se utiliza para datos conmutados en paquetes disponibles a intervalos irregulares. Estos datos provienen de la capa L2CAP en el nodo emisor y se entregan en la capa L2CAP en el nodo receptor. El tráfico ACL se entrega sobre la base de mejor esfuerzo. No hay garantías. Las tramas se pueden perder y tienen que retransmitirse. Un esclavo puede tener sólo un enlace ACL con su maestro.

El otro tipo de enlace es el **SCO (Síncrono Orientado a la Conexión),** para datos en tiempo real, como ocurre en las conexiones telefónicas. A este tipo de canal se le asigna una ranura fija en cada dirección. Por la importancia del tiempo en los enlaces SCO, las tramas que se envían a través de ellos nunca se retransmiten. En vez de ello, se puede utilizar la corrección de errores hacia delante (o corrección de errores sin canal de retorno) para conferir una confiabilidad alta. Un esclavo puede establecer hasta tres enlaces SCO con su maestro. Cada enlace de este tipo puede transmitir un canal de audio PCM de 64,000 bps.

4.6.6 La capa L2CAP de Bluetooth

La capa L2CAP tiene tres funciones principales. Primera, acepta paquetes de hasta 64 KB provenientes de las capas superiores y los divide en tramas para transmitirlos. Las tramas se reensamblan nuevamente en paquetes en el otro extremo.

Segunda, maneja la multiplexión y desmultiplexión de múltiples fuentes de paquetes. Cuando se reensambla un paquete, la capa L2CAP determina cuál protocolo de las capas superiores lo manejará, por ejemplo, el RFcomm o el de telefonía.

Tercera, la capa L2CAP se encarga de la calidad de los requerimientos de servicio, tanto al establecer los enlaces como durante la operación normal. Asimismo, durante el establecimiento de los enlaces se negocia el tamaño máximo de carga útil permitido, para evitar que un dispositivo que envíe paquetes grandes sature a uno que reciba paquetes pequeños. Esta característica es importante porque no todos los dispositivos pueden manejar paquetes de 64 KB.

4.6.7 Estructura de la trama de Bluetooth

Existen diversos formatos de trama, el más importante de los cuales se muestra en la figura 4-38. Empieza con un código de acceso que identifica al maestro, cuyo propósito es que los esclavos que se encuentren en el rango de alcance de dos maestros sepan cuál tráfico es para ellos. A continuación se encuentra un encabezado de 54 bits que contiene campos comunes de la subcapa MAC. Luego está el campo de datos, de hasta 2744 bits (para una transmisión de cinco ranuras). Para una sola ranura de tiempo, el formato es el mismo excepto que el campo de datos es de 240 bits.

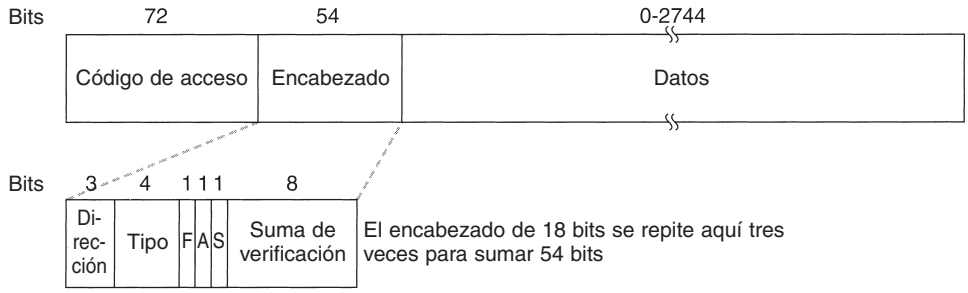


Figura 4-38. Trama de datos típica de Bluetooth.

Demos un rápido vistazo al encabezado. El campo *Dirección* identifica a cuál de los ocho dispositivos activos está destinada la trama. El campo *Tipo* indica el tipo de trama (ACL, SCO, de sondeo o nula), el tipo de corrección de errores que se utiliza en el campo de datos y cuántas ranuras de longitud tiene la trama. Un esclavo establece el bit *F* (de flujo) cuando su búfer está lleno y no puede recibir más datos. Ésta es una forma primitiva de control de flujo. El bit *A* (de confirmación de recepción) se utiliza para incorporar un ACK en una trama. El bit *S* (de secuencia) sirve para numerar las tramas con el propósito de detectar retransmisiones. El protocolo es de parada y espera, por lo que 1 bit es suficiente. A continuación se encuentra el encabezado *Suma de verificación* de 8 bits. Todo el encabezado de 18 bits se repite tres veces para formar el encabezado de 54 bits que se aprecia en la figura 4-38. En el receptor, un circuito sencillo examina las tres copias de cada bit. Si son las mismas, el bit es aceptado. De lo contrario, se impone la opinión de la mayoría. De esta forma, 54 bits de capacidad de transmisión se utilizan para enviar 10 bits de encabezado. Esto se debe a que es necesaria una gran cantidad de redundancia para enviar datos de manera confiable en un entorno con ruido mediante dispositivos de bajo costo y baja potencia (2.5 mW) con poca capacidad de cómputo.

En el campo de datos de las tramas ACL se utilizan varios formatos. Sin embargo, las tramas SCO son más sencillas: el campo de datos siempre es de 240 bits. Se definen tres variantes, que permiten 80, 160 y 240 bits de carga útil real, y el resto se utiliza para corrección de errores. En la versión más confiable (carga útil de 80 bits), el contenido se repite tres veces, al igual que el encabezado.

Debido a que el esclavo podría utilizar solamente las ranuras impares, obtiene 800 ranuras por segundo, de la misma manera que el maestro. Con una carga útil de 80 bits, la capacidad de canal del esclavo es de 64,000 bps y la del maestro también es de 64,000 bps, exactamente la necesaria para un canal de voz PCM dúplex total (razón por la cual se eligió una tasa de saltos de 1600 saltos por segundo). Estas cifras indican que un canal de voz dúplex total con 64,000 bps en cada dirección y el formato más confiable satura totalmente la *piconet* a pesar de un ancho de banda puro de 1 Mbps. En la variante menos confiable (240 bits por ranura sin redundancia a este nivel), se pueden soportar al mismo tiempo tres canales de voz dúplex total, debido a lo cual se permite un máximo de tres enlaces SCO por esclavo.

Hay mucho más por decir acerca de Bluetooth, pero ya no tenemos espacio aquí. Si desea más información, vea (Bhagwat, 2001; Bisdikian, 2001; Bray y Sturman, 2002; Haartsen, 2000; Johansson y cols., 2001; Miller y Bisdikian, 2001, y Sairam y cols., 2002).

4.7 CONMUTACIÓN EN LA CAPA DE ENLACE DE DATOS

Muchas organizaciones tienen varias LANs y desean interconectarlas. Este tipo de redes se puede conectar mediante dispositivos llamados **puentes,** que funcionan en la capa de enlace de datos. Los puentes examinan las direcciones de la capa de enlace de datos para enrutar los datos. Como no tienen que examinar el campo de carga útil de las tramas que enrutan, pueden transportar paquetes IPv4 (que se utilizan actualmente en Internet), IPv6 (que se utilizarán en el futuro en Internet), AppleTalk, ATM, OSI o de otros tipos. En contraste, los *enrutadores* examinan

las direcciones de los paquetes y realizan su trabajo de enrutamiento con base en ellas. Aunque ésta parece una clara división entre los puentes y los enrutadores, algunos desarrollos modernos, como el surgimiento de la Ethernet conmutada, han enturbiado las aguas, como veremos más tarde. En las siguientes secciones analizaremos los puentes y los conmutadores, en especial para conectar diferentes LANs 802. Para un análisis más completo sobre puentes, conmutadores y temas relacionados, vea (Perlman, 2000).

Antes de entrar de lleno a la tecnología de los puentes, vale la pena dar un vistazo a algunas situaciones comunes en las cuales se utilizan los puentes. Mencionaremos seis razones por las cuales una sola organización podría contar con varias LANs.

Primera, muchas universidades y departamentos corporativos tienen sus propias LANs, principalmente para interconectar sus propias computadoras personales, estaciones de trabajo y servidores. Dado que las metas de los distintos departamentos difieren, los departamentos escogen LANs diferentes, sin importarles lo que hagan los demás departamentos. Tarde o temprano surge la necesidad de interacción, y aquí es donde entran los puentes. En este ejemplo surgieron múltiples LANs debido a la autonomía de sus dueños.

Segunda, la organización puede estar distribuida geográficamente en varios edificios, separados por distancias considerables. Puede ser más económico tener LANs independientes en cada edificio y conectarlas mediante puentes y enlaces láser que tender un solo cable por toda la zona.

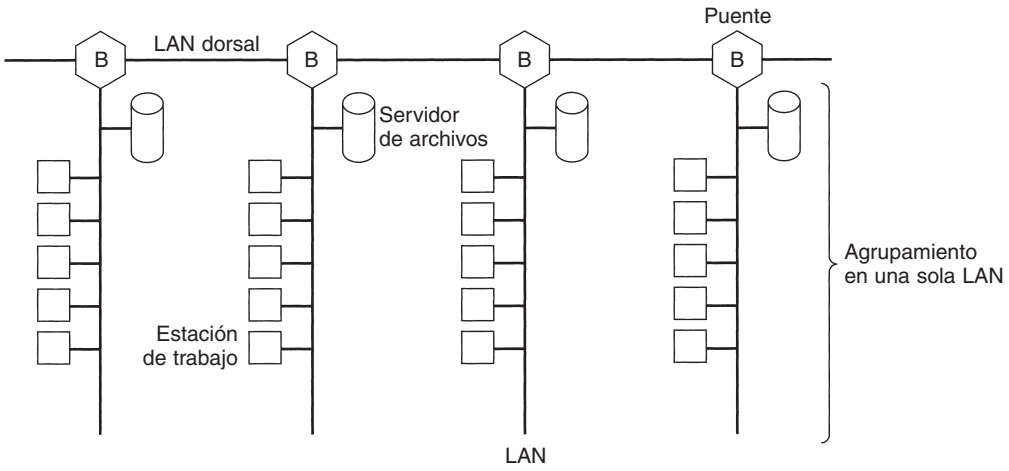


Figura 4-39. Varias LANs conectadas mediante una red dorsal para manejar una carga total mayor que la capacidad de una sola LAN.

Tercera, puede ser necesario dividir lo que lógicamente es una sola LAN en LANs individuales para manejar la carga. Por ejemplo, en muchas universidades miles de estaciones de trabajo están disponibles para los estudiantes y el cuerpo docente. Los archivos normalmente se guardan en máquinas servidoras de archivos, y son descargados a las máquinas de los usuarios a solicitud. La enorme escala de este sistema hace imposible poner todas las estaciones de trabajo en una sola

LAN, pues el ancho de banda total necesario es demasiado grande. En cambio, se usan varias LANs conectadas mediante puentes, como se muestra en la figura 4-39. Cada LAN contiene un grupo de estaciones de trabajo con su propio servidor de archivos, por lo que la mayor parte del tráfico está restringida a una sola LAN y no agrega carga a la red dorsal.

Vale la pena mencionar que aunque por lo general esquematizamos las LANs como cables con múltiples derivaciones como en la figura 4-39 (la apariencia clásica), hoy en día se implementan con mayor frecuencia mediante concentradores o conmutadores. Sin embargo, un cable largo con múltiples derivaciones para las máquinas, y un concentrador con las máquinas conectadas a él funcionan de manera idéntica. En ambos casos, todas las máquinas pertenecen al mismo dominio de colisión y todas utilizan el protocolo CSMA/CD para enviar tramas. No obstante, las LANs conmutadas son diferentes, como ya vimos y volveremos a ver en breve.

Cuarta, en algunas situaciones una sola LAN sería adecuada en términos de la carga, pero la distancia física entre las máquinas más distantes es demasiado grande (por ejemplo, mayor que 2.5 km para Ethernet). Aun si fuera fácil tender el cable, la red no funcionaría debido al retardo excesivamente grande del viaje de ida y vuelta. La única solución es segmentar la LAN e instalar puentes entre los segmentos. Con puentes, puede aumentarse la distancia física total cubierta.

Quinta, está la cuestión de la confiabilidad. En una sola LAN, un nodo defectuoso que envíe constantemente una cadena de basura echará a perder la LAN. Pueden introducirse puentes en lugares críticos, como puertas para bloquear el fuego en un edificio, y así evitar que un solo nodo enloquecido tire el sistema completo. A diferencia de un repetidor, que sólo copia lo que ve, un puente puede programarse para discriminar lo que envía y lo que no.

Sexta y última, los puentes pueden contribuir a la seguridad de la organización. La mayor parte de las interfaces LAN tienen un **modo promiscuo**, en el que *todas* las tramas se entregan a la computadora, no sólo las dirigidas a ella. Los espías y los intrusos aman esta característica. Al introducir puentes en varios lugares y tener cuidado de no reenviar tráfico delicado, es posible aislar partes de la red de manera que su tráfico no pueda escapar y caer en las manos equivocadas.

En el plano ideal, los puentes deberían ser totalmente transparentes, es decir, debería ser posible cambiar una máquina de un segmento a otro sin necesidad de modificar el hardware, software o tablas de configuración. Asimismo, debería ser posible que las máquinas de un segmento se comunicaran con las de cualquier otro segmento sin que importara el tipo de LAN que se utilizara en ambos segmentos o en los segmentos que hubiera entre ellos. Este objetivo se consigue en ocasiones, pero no siempre.

4.7.1 Puentes de 802.x a 802.y

Después de comprender por qué son necesarios los puentes, pasemos a la cuestión de su funcionamiento. En la figura 4-40 se ilustra la operación de un puente sencillo de dos puertos. El *host A* en una LAN (802.11) inalámbrica tiene un paquete por enviar a un *host* fijo, *B,* en una Ethernet (802.3) a la cual se encuentra conectada la LAN inalámbrica. El paquete desciende a la subcapa LLC y adquiere un encabezado LLC (aparece en negro en la figura). A continuación el paquete pasa a la subcapa MAC y se le antepone un encabezado 802.11 (también un terminador, que no se

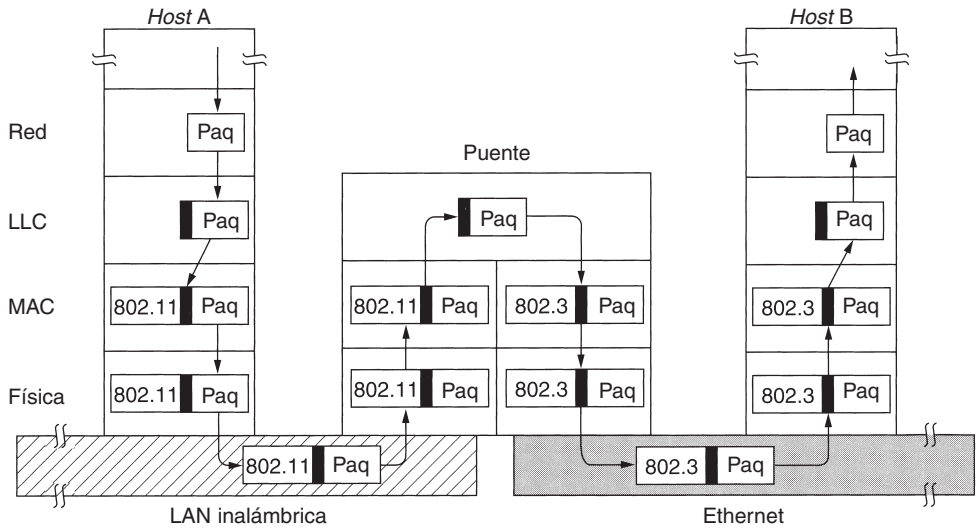


Figura 4-40. Operación de un puente entre una red 802.11 y una 802.3.

muestra en la figura). Esta unidad viaja a través del aire y es captada por la estación base, que se percata de que tiene que pasar por la Ethernet fija. Cuando el paquete llega al puente que conecta la red 802.11 con la 802.3, empieza en la capa física y realiza el recorrido hacia arriba. El encabezado 802.11 se elimina en la subcapa MAC del puente. El paquete recortado (con el encabezado LLC) se pasa a la subcapa LLC del puente. En este ejemplo, el paquete está destinado a una LAN 802.3, por lo que recorre la ruta hacia abajo en el lado 802.3 del puente y al terminar pasa a la red Ethernet. Observe que un puente que conecta *k* LANs diferentes tendrá *k* subcapas MAC diferentes y *k* capas físicas diferentes, una para cada tipo.

Hasta aquí pareciera que es muy sencillo desplazar una trama de una LAN a otra. No es así. En esta sección señalaremos algunos de los problemas que se enfrentan al intentar construir un puente entre las diversas LANs (y MANs) 802. Nos concentraremos en 802.3, 802.11 y 802.16, pero existen otras, cada una con sus propios problemas.

Para empezar, cada LAN utiliza un formato de trama distinto (vea la figura 4-41). En contraste con las diferencias entre Ethernet, token bus y token ring, que se originaron por la historia y el ego de las grandes corporaciones, aquí las diferencias son válidas hasta cierto punto. Por ejemplo, el campo *Duración* en 802.11 se justifica por el protocolo MACAW y no tiene sentido en Ethernet. En consecuencia, cualquier copia que se realice entre LANs distintas requiere de volver a dar formato, lo que lleva tiempo de CPU, una nueva suma de verificación y se presenta la posibilidad de errores sin detectar debido a bits erróneos en la memoria del puente.

Un segundo problema es que las LANs interconectadas no necesariamente operan a la misma tasa de datos. Al retransmitir una gran cantidad de tramas una tras otra, provenientes de una LAN rápida a otra más lenta, el puente será incapaz de despachar las tramas con la misma rapidez que

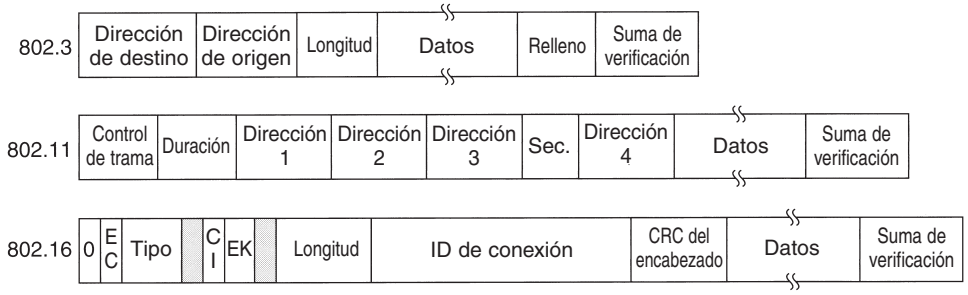


Figura 4-41. Formatos de trama de la redes 802. El dibujo no es a escala.

arriban. Por ejemplo, si una Gigabit Ethernet envía bits a su velocidad máxima a una LAN 802.11b de 11 Mbps, el puente tendrá que almacenarlos en búfer, con la esperanza de no agotar su memoria. Los puentes que conectan tres o más LANs tienen un problema similar cuando varias LANs intentan enviar datos a una misma LAN al mismo tiempo aun cuando todas operen a la misma velocidad.

Un tercer problema, y potencialmente el más grave de todos, es que distintas LANs 802 tienen diferentes longitudes máximas de trama. Un problema obvio surge cuando una trama grande tiene que reenviarse a una LAN que no puede aceptarla. En esta capa no es posible dividir la trama. Todos los protocolos dan por sentado que las tramas llegan o se pierden. No se considera el reensamblado de las tramas a partir de unidades más pequeñas. Lo anterior no significa que no se pueden diseñar tales protocolos. Es posible y se ha hecho. Es sólo que ningún protocolo de enlace de datos confiere esta característica, así que los puentes deben olvidarse de manipular la carga útil de la trama. En esencia, no hay solución. Las tramas demasiado grandes para reenviarse deben descartarse. Es suficiente sobre la transparencia.

Otro punto es la seguridad. Tanto el 802.11 como el 802.16 soportan encriptación en la capa de enlace de datos. Ethernet no. Esto significa que los diversos servicios de encriptación disponibles en las redes inalámbricas se pierden cuando el tráfico pasa sobre una Ethernet. Peor aún, si una estación inalámbrica emplea encriptación en la capa de enlace de datos, no habrá forma de desencriptar los datos cuando lleguen a la red Ethernet. Si la estación inalámbrica no utiliza encriptación, su tráfico quedará expuesto en el enlace aéreo. De cualquier manera hay un problema.

Una solución al problema de la seguridad es realizar la encriptación en una capa superior, pero en este caso la estación 802.11 tiene que saber si se está comunicando con otra estación sobre una red 802.11 (lo que significa que utilizará encriptación en la capa de enlace de datos) o con una distinta (en cuyo caso no utilizará encriptación). Al obligar a la estación a elegir se destruye la transparencia.

Un punto final es la calidad del servicio. Tanto el 802.11 como el 802.16 la ofrecen en diversas formas, el primero con el modo PCF y el último mediante conexiones a tasas de bits constantes. En Ethernet no existe el concepto de calidad del servicio, así que el tráfico proveniente de alguna de las anteriores perderá su calidad de servicio al pasar por una Ethernet.

4.7.2 Interconectividad local

La sección anterior examinó los problemas que surgen al conectar dos LANs IEEE 802 distintas mediante un solo puente. Sin embargo, en grandes organizaciones con muchas LANs, la sola interconexión entre todas da lugar a muchos problemas, aun cuando todas sean Ethernet. En un plano ideal, debería bastar con adquirir puentes diseñados para el estándar IEEE e insertar los conectores en el puente para que todo funcionara perfectamente al instante. No deberían ser necesarios cambios de hardware ni de software, ni configurar conmutadores de direcciones, descargar tablas de enrutamiento ni parámetros, nada. Tan sólo conectar los cables y empezar a trabajar. Más aún, los puentes no deberían afectar de ninguna manera el funcionamiento de LANs existentes. En otras palabras, los puentes deberían ser completamente transparentes (invisibles para todo el hardware y el software). Sorprendentemente, esto es posible. Echemos un vistazo a la manera en que se hace realidad esta magia.

En su forma más sencilla, un puente transparente funciona en modo promiscuo y acepta todas las tramas transmitidas sobre las LANs a las cuales está conectado. Tomemos como ejemplo la configuración de la figura 4-42. El puente B1 está conectado a las LANs 1 y 2, y el puente B2 está conectado a las LANs 2, 3 y 4. Una trama que llega al puente B1 en la LAN 1 con destino a *A* se puede descartar de inmediato porque se encuentra en la LAN correcta, pero una trama que llega a la LAN 1 con destino a *C* o *F* debe reenviarse.

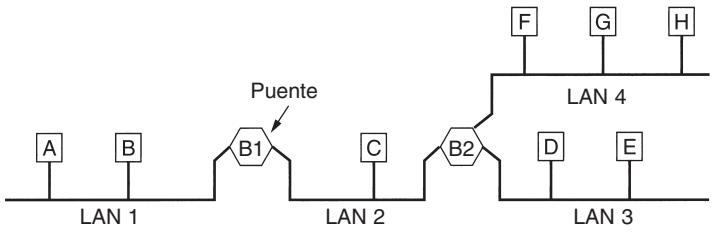


Figura 4-42. Configuración con cuatro LANs y dos puentes.

Cuando llega una trama, un puente debe decidir si la descarta o la reenvía, y si elige lo último, a cuál LAN la mandará. Esta decisión la toma consultando la dirección de destino en una enorme tabla (de *hash*) que se encuentra en su interior. La tabla lista cada posible destino e indica a cuál línea de salida (LAN) pertenece la trama. Por ejemplo, la tabla de B2 podría listar que *A* pertenece a LAN 2, ya que todo lo que B2 tiene que saber es a cuál LAN enviar las tramas para *A*. No le preocupa en absoluto el hecho de que posteriormente ocurran más reenvíos.

Cuando los puentes se conectan por primera vez, todas las tablas de *hash* están vacías. Ninguno de los puentes sabe dónde se encuentran los destinos, por lo que utilizan un algoritmo de inundación: todas las tramas que llegan con un destino desconocido se envían a todas las LANs a las cuales está conectado el puente, excepto a aquélla de la cual proceden. Con el paso del tiempo, los puentes aprenden dónde están los destinos, como se describe más adelante. Una vez conocido un destino, las tramas para él se reenvían solamente a la LAN apropiada en lugar de a todas las LANs.

El algoritmo que los puentes transparentes utilizan es **aprendizaje hacia atrás.** Como ya mencionamos, los puentes funcionan en modo promiscuo y de esta manera pueden ver todas las tramas que se envían a cualquiera de sus LANs. Al examinar la dirección del origen, pueden saber cuál máquina está disponible en cuál LAN. Por ejemplo, si el puente B1 de la figura 4-42 ve una trama proveniente de *C* en la LAN 2, sabe que es posible acceder a *C* por medio de la LAN 2, así que registra una entrada en su tabla de *hash* con la observación de que las tramas para *C* deben utilizar la LAN 2. Cualquier trama subsecuente dirigida a *C* que llegue desde la LAN 1 será reenviada, pero una trama para *C* que llegue desde la LAN 2 será descartada.

La topología puede cambiar conforme las máquinas y los puentes se enciendan y apaguen, o cuando se trasladen de un sitio a otro. Para manejar topologías dinámicas, siempre que se realiza una entrada en una tabla de *hash* se registra en la entrada la hora de llegada de una trama. Siempre que llega una trama cuyo origen ya está en la tabla, su entrada se actualiza con la hora actual. Por lo tanto, la hora asociada a cada entrada indica la última vez que se registró una trama proveniente de ese origen.

Un proceso del puente analiza periódicamente la tabla de *hash* y purga todas las entradas que tengan más de algunos minutos. De esta manera, si una computadora se desconecta de su LAN, se traslada a otro lugar del edificio y se vuelve a conectar en algún otro lugar, en pocos minutos volverá a funcionar con normalidad, sin necesidad de intervención manual. Este algoritmo también significa que si una máquina está inactiva durante algunos minutos, el tráfico destinado a ella será inundado hasta que la máquina misma envíe una trama.

El procedimiento de enrutamiento para una trama entrante depende de la LAN de que proceda (la LAN de origen) y de la LAN a la cual está destinada (la LAN de destino), como se puede ver a continuación:

1. Si las LANs de destino y de origen son la misma, descartar la trama.
2. Si las LANs de destino y de origen son diferentes, reenviar la trama.
3. Si se desconoce la LAN de destino, recurrir a la inundación.

Este algoritmo debe aplicarse cada vez que llega una trama. Chips VLSI especiales realizan la consulta y actualización de las entradas de la tabla en tan sólo algunos microsegundos.

4.7.3 Puentes con árbol de expansión

Para incrementar la confiabilidad, algunos sitios utilizan dos o más puentes en paralelo entre pares de LANs, como se muestra en la figura 4-43. Sin embargo, este arreglo también genera algunos problemas adicionales porque produce ciclos en la topología.

Un ejemplo simple de estos problemas lo tenemos al observar cómo se maneja una trama, *F*, con destino desconocido, en la figura 4-43. Cada puente, siguiendo las reglas normales para el manejo de destinos desconocidos, recurre a la inundación, que en este ejemplo es tan sólo copiar la trama a la LAN 2. Poco después, el puente 1 detecta a F_2, una trama con destino desconocido, y

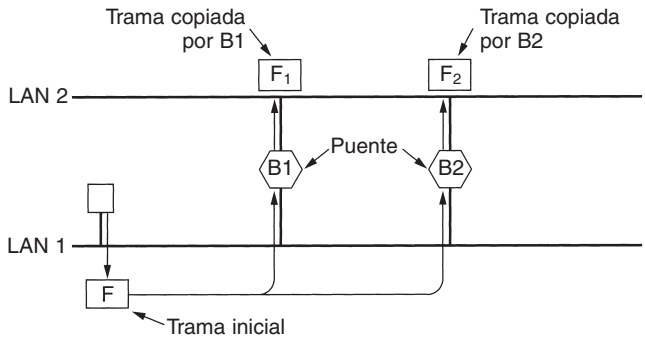


Figura 4-43. Dos puentes paralelos transparentes.

la copia a la LAN 1, lo cual genera a F_3 (no se muestra). De manera similar, el puente 2 copia a F_1 a la LAN 1 y genera a F_4 (tampoco se muestra). El puente 1 reenvía ahora a F_4 y el puente 2 copia a F_3. Este ciclo se repite una y otra vez.

La solución a este problema es que los puentes se comuniquen entre sí y cubran la topología existente con un árbol de expansión que llegue a todas las LANs. En realidad, algunas conexiones potenciales entre LANs se ignoran en el afán de construir una topología ficticia libre de ciclos. Por ejemplo, en la figura 4-44(a) vemos nueve LANs interconectadas por diez puentes. Esta configuración se puede abstraer en un grafo con las LANs como nodos. Un arco conecta dos LANs que estén unidas por un puente. El grafo puede reducirse a un árbol de expansión eliminando los arcos que se muestran como líneas punteadas en la figura 4-44(b). Con este árbol de expansión existe exactamente una ruta desde cada LAN hasta las demás LANs. Una vez que los puentes se ponen de acuerdo en el árbol de expansión, todos los reenvíos entre LANs se hacen a través del árbol de expansión. Puesto que existe sólo una ruta de cada origen a cada destino, es imposible que se generen ciclos.

Para construir el árbol de expansión, los puentes primero tienen que escoger un puente que funja como raíz del árbol. Toman esta decisión haciendo que cada uno difunda su número de serie, instalado por el fabricante y con garantía de ser único en el mundo. El puente con el menor número de serie se vuelve la raíz. A continuación, se construye un árbol con las rutas más cortas de la raíz a cada puente y LAN. Éste es el árbol de expansión. Si falla un puente o una LAN, se calcula un árbol nuevo.

El resultado de este algoritmo es que se establece una ruta única de cada LAN hasta la raíz y, por tanto, a todas las demás LANs. Aunque el árbol abarca todas las LANs, no necesariamente están presentes todos los puentes en el árbol (para evitar ciclos). Aun después de que se ha establecido el árbol de expansión, el algoritmo continúa operando a fin de detectar automáticamente cambios de topología y actualizar el árbol. El algoritmo distribuido que se usa para construir el árbol de expansión fue inventado por Radia Perlman y se describe en detalle en (Perlman, 2000). Se estandarizó en el IEEE 802.1D.

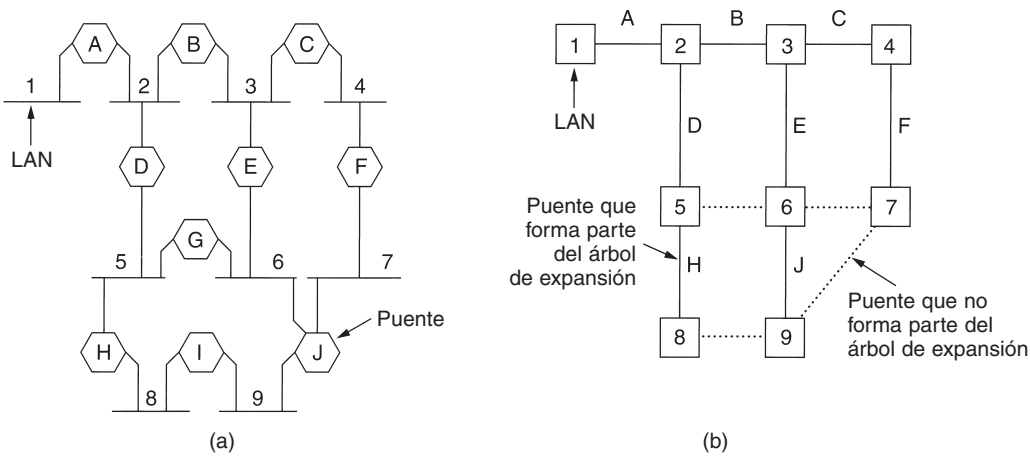


Figura 4-44. (a) LANs interconectadas. (b) Árbol de expansión que abarca las LANs. Las líneas punteadas no son parte del árbol de expansión.

4.7.4 Puentes remotos

Un uso común de los puentes es conectar dos (o más) LANs distantes. Por ejemplo, una empresa podría contar con plantas en varias ciudades, cada una con su propia LAN. En un plano ideal, todas las LANs deberían estar interconectadas de tal forma que funcionaran como una sola LAN grande.

Este objetivo se puede cumplir colocando un puente en cada LAN y conectando los puentes por pares con líneas punto a punto (por ejemplo, líneas alquiladas a una compañía telefónica). En la figura 4-45 se ilustra un sistema sencillo, con tres LANs. Aquí se aplican los algoritmos comunes de enrutamiento. La forma más sencilla de entender esto es considerar las tres líneas punto a punto como LANs sin *hosts*. A continuación tenemos un sistema normal de seis LANs interconectadas mediante cuatro puentes. En ninguna parte hasta aquí hemos dicho que una LAN debe contener *hosts*.

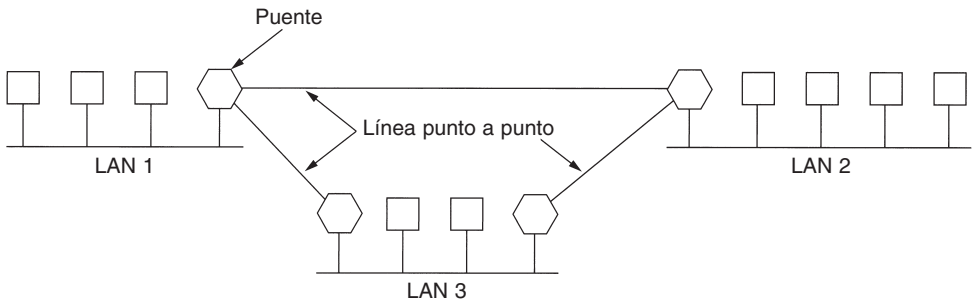


Figura 4-45. Los puentes remotos se pueden utilizar para interconectar LANs distantes.

En las líneas punto a punto se pueden utilizar diversos protocolos. Una opción es elegir algún protocolo de enlace de datos estándar de punto a punto como PPP y colocar tramas MAC completas en el campo de carga útil. Esta estrategia funciona mejor si todas las LANs son idénticas, y el único problema es conseguir que las tramas lleguen a la LAN correcta. Otra opción es eliminar tanto el encabezado como el terminador MAC en el puente de origen y agregar lo que queda en el campo de carga útil del protocolo de punto a punto. A continuación, en el puente de destino se pueden generar un nuevo encabezado y un nuevo terminador MAC. Una desventaja de este método consiste en que la suma de verificación que llega al *host* de destino no es la que se calculó en el *host* de origen, debido a lo cual tal vez no se detecten los errores ocasionados por bits defectuosos en la memoria de un puente.

4.7.5 Repetidores, concentradores, puentes, conmutadores, enrutadores y puertas de enlace

Hasta este punto hemos visto una gran variedad de formas para desplazar tramas y paquetes de un segmento de cable a otro. Hemos mencionado repetidores, puentes, conmutadores, concentradores, enrutadores y puertas de enlace. Todos estos dispositivos son de uso común, aunque difieren en formas sutiles y no tan sutiles. Puesto que son tantos, tal vez valga la pena analizarlos en conjunto para conocer sus similitudes y diferencias.

Para empezar, estos dispositivos operan en diferentes capas, como se muestra en la figura 4-46(a). La capa es importante porque los distintos dispositivos utilizan diferentes partes de información para decidir su modo de operación. En un escenario común, el usuario genera algunos datos que se enviarán a una máquina remota. Estos datos se pasan a la capa de transporte, que le agrega un encabezado, por ejemplo, un encabezado TCP, y pasa la unidad que resulta a la capa de red. Ésta incorpora su propio encabezado para obtener un paquete de capa de red, por ejemplo, un paquete IP. En la figura 4-46(b) podemos ver el paquete IP con un sombreado gris. A continuación, el paquete pasa a la capa de enlace de datos, que incorpora su propio encabezado y suma de verificación (CRC) y envía la trama resultante a la capa física para que desde ahí sea transmitida, por ejemplo, sobre una LAN.

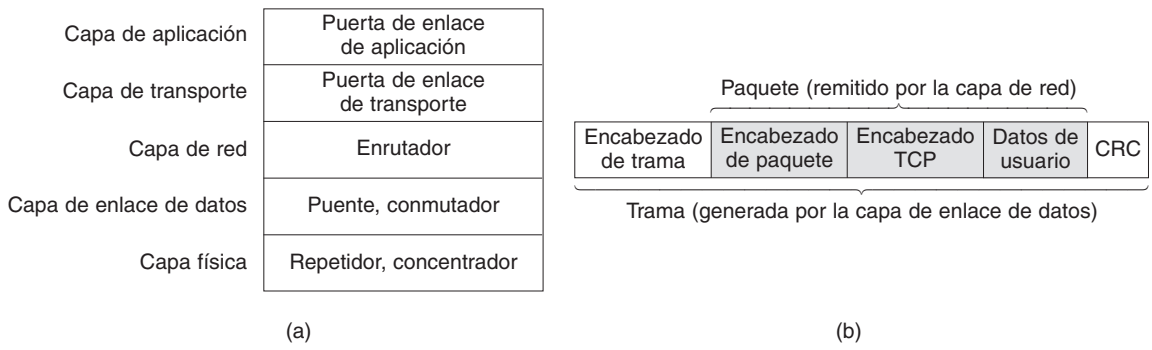


Figura 4-46. (a) Los dispositivos y sus capas correspondientes. (b) Tramas, paquetes y encabezados.

Ahora demos un vistazo a los dispositivos de conmutación y veamos cómo se relacionan con los paquetes y las tramas. Al fondo, en la capa física, se encuentran los repetidores. Éstos son dispositivos análogos conectados a dos segmentos de cable. Una señal que aparece en uno de ellos es amplificada y enviada al otro. Los repetidores no distinguen entre tramas, paquetes o encabezados. Manejan voltios. Por ejemplo, la Ethernet tradicional admite cuatro repetidores, con el propósito de extender la longitud máxima de cable de 500 a 2500 metros.

Pasemos ahora a los concentradores. Un concentrador tiene numerosos puertos de entrada que une de manera eléctrica. Las tramas que llegan a cualquiera de las líneas se envían a todas las demás. Si dos tramas llegan al mismo tiempo, chocarán, al igual que en un cable coaxial. En otras palabras, el concentrador constituye un solo dominio de colisión. Todas las líneas que convergen en un concentrador deben operar a la misma velocidad. A diferencia de los repetidores, los concentradores (por lo general) no amplifican las señales entrantes y su diseño les permite contener varias tarjetas de línea con múltiples entradas, aunque las diferencias son ligeras. Al igual que los repetidores, los concentradores no examinan las direcciones 802 ni las utilizan de ninguna manera. En la figura 4-47(a) se muestra un concentrador.

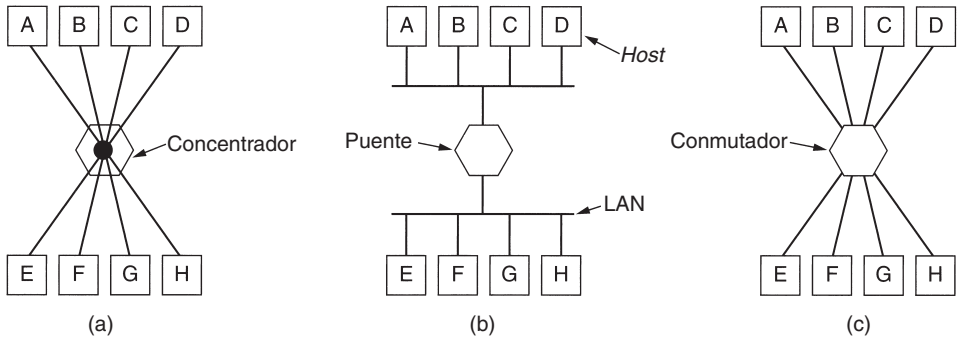


Figura 4-47. (a) Concentrador. (b) Puente. (c) Conmutador.

Veamos a continuación la capa de enlace de datos donde operan los puentes y los conmutadores. Ya hemos visto algo de los puentes. Un puente conecta dos o más LANs, como se puede ver en la figura 4-47(b). Cuando llega una trama, el software del puente extrae la dirección de destino del encabezado y la busca en una tabla para averiguar a dónde debe enviar la trama. En Ethernet, esta dirección es la dirección de destino de 48 bits que se muestra en la figura 4-17. De la misma manera que un concentrador, un puente moderno tiene tarjetas de línea, por lo general para cuatro u ocho puertos de entrada de un tipo determinado. Una tarjeta de línea para Ethernet no puede manejar tramas token ring debido a que no sabe dónde buscar la dirección de destino que viene en el encabezado de la trama. Sin embargo, un puente podría tener tarjetas de línea para diferentes tipos de red y diferentes velocidades. En contraste con un concentrador, en un puente cada puerto constituye su propio dominio de colisión.

Los conmutadores son similares a los puentes en el aspecto de que ambos enrutan tomando como base las direcciones de las tramas. De hecho, mucha gente se refiere a ellos de manera indistinta. La principal diferencia consiste en que un conmutador se utiliza con mayor frecuencia

para conectar computadoras individuales, como se puede ver en la figura 4-47(c). En consecuencia, cuando el *host A* de la figura 4-47(b) desea enviar una trama al *host B*, el puente toma la trama pero la descarta. Por el contrario, en la figura 4-47(c), el conmutador debe reenviar activamente la trama de *A* a *B* porque no existe otra forma para que ésta llegue ahí. Puesto que por lo general cada puerto del conmutador va hacia una sola computadora, éstos deben contar con espacio para muchas más tarjetas de línea que los puentes, cuyo propósito es conectar solamente LANs. Cada tarjeta de línea proporciona espacio de búfer para las tramas que llegan a sus puertos. Como cada puerto constituye su propio dominio de colisión, los conmutadores nunca pierden tramas por colisiones. Sin embargo, si las tramas llegan con mayor rapidez de la que pueden retransmitirse, el conmutador podría quedarse sin espacio de búfer y proceder a descartar tramas.

Para aliviar en parte este problema, los conmutadores modernos empiezan el reenvío de tramas tan pronto como llega el campo de encabezado del destino, antes de que el resto de la trama haya llegado (siempre y cuando el puerto de salida esté disponible, por supuesto). Estos conmutadores no utilizan la técnica de conmutación de almacenamiento y reenvío. En ocasiones se les menciona como **conmutadores cut-through.** Por lo general, este tipo de manejo se realiza por completo en hardware, en tanto que los puentes contienen tradicionalmente una CPU que realiza la conmutación de almacenamiento y reenvío en software. No obstante, debido a que todos los puentes y conmutadores modernos contienen circuitos integrados especiales para conmutación, la diferencia entre un conmutador y un puente es más un asunto de mercadotecnia que técnico.

Hasta aquí hemos visto repetidores y concentradores, que son bastante similares, así como puentes y conmutadores, que también son muy semejantes. Ahora pasaremos a los enrutadores, que son diferentes de todos los anteriores. Cuando un paquete llega a un enrutador, el encabezado y el terminador de la trama se eliminan y el paquete contenido en el campo de carga útil de la trama (sombreado en la figura 4-46) se pasa al software de enrutamiento. Este software se vale del encabezado del paquete para elegir un puerto de salida. En un paquete IP, el encabezado contendrá una dirección de 32 bits (IPv4) o 128 bits (IPv6), no una dirección 802 de 48 bits. El software de enrutamiento no analiza las direcciones de las tramas e incluso no sabe si el paquete proviene de una LAN o una línea punto a punto. En el capítulo 5 estudiaremos los enrutadores y el enrutamiento.

Una capa más arriba encontramos puertas de enlace de transporte. Estos dispositivos conectan dos computadoras que utilizan diferentes protocolos de transporte orientados a la conexión. Por ejemplo, imagine que una computadora que utiliza el protocolo TCP/IP orientado a la conexión necesita comunicarse con una computadora que emplea el protocolo de transporte ATM, también orientado a la conexión. La puerta de enlace de transporte puede copiar los paquetes de una conexión a la otra y darles el formato que necesiten.

Por último, las puertas de enlace de aplicación comprenden el formato y contenido de los datos y traducen los mensajes de un formato a otro. Por ejemplo, una puerta de enlace de correo electrónico puede traducir mensajes Internet en mensajes SMS para teléfonos móviles.

4.7.6 LANs virtuales

En los primeros días de las redes de área local, cables amarillos gruesos serpenteaban por los ductos de muchos edificios de oficinas. Conectaban a todas las computadoras por las que pasa-

ban. Con frecuencia había muchos cables, los cuales se conectaban a una red vertebral central (como en la figura 4-39) o a un concentrador central. No importaba cuál computadora pertenecía a cuál LAN. Todos los usuarios de oficinas cercanas se conectaban a la misma LAN aunque no estuvieran relacionados con ella. El aspecto geográfico se imponía al lógico.

Todo cambió con el surgimiento de 10Base-T y los concentradores en la década de 1990. El cableado de los edificios se renovó (a un costo considerable) para desechar todas las mangueras amarillas de jardín e instalar cables de par trenzado desde cada oficina hasta gabinetes centrales al final de cada pasillo o hasta salas centrales de máquinas, como se observa en la figura 4-48. Si el encargado del reemplazo del cableado era un visionario, se instalaba cable de par trenzado categoría 5; si era un simple administrador, se instalaba el cable telefónico (categoría 3) existente (que tenía que reemplazarse algunos años más tarde con la aparición de Fast Ethernet).

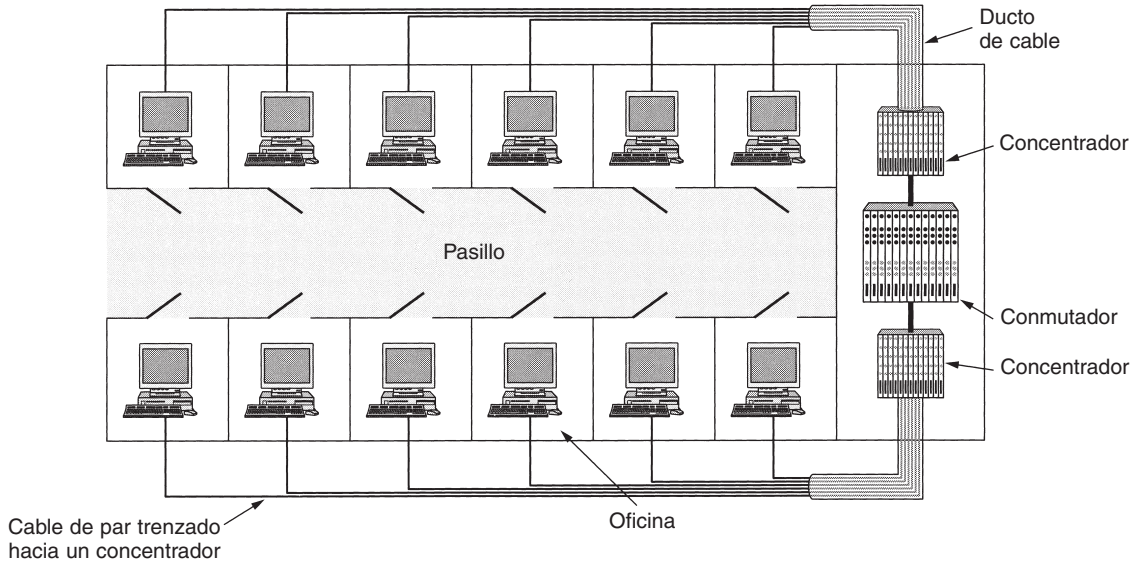


Figura 4-48. Edificio con cableado centralizado que utiliza concentradores y un conmutador.

El uso de concentradores (y de conmutadores posteriormente) con Ethernet hizo posible configurar las LANs con base en el aspecto lógico más que en el físico. Si una empresa necesita *k* LANs, compra *k* concentradores. Al elegir con cuidado qué conectores incorporar en qué concentradores, los usuarios de una LAN se pueden seleccionar de tal manera que tenga sentido para la organización, sin tomar mucho en cuenta el aspecto geográfico. Por supuesto, si dos personas del mismo departamento trabajan en diferentes edificios, es muy probable que estarán en diferentes concentradores y, en consecuencia, en diferentes LANs. No obstante, esto es mucho mejor que tener usuarios de una LAN con base totalmente en el aspecto geográfico.

¿Es importante quién está en qué LAN? Después de todo, en casi todas las organizaciones las LANs están interconectadas. Como veremos en breve, sí es importante. Por diversas razones, a los administradores de red les gusta agrupar a los usarios en LANs para reflejar la estructura de la or-

ganización más que el diseño físico del edificio. Un aspecto es la seguridad. Cualquier interfaz de red se puede operar en modo promiscuo para que copie todo el tráfico que llegue. Muchos departamentos, como los de investigación, patentes y contabilidad, manejan información que no debe salir de los límites de sus respectivas áreas. En casos como éste se justifica que todos los usuarios de un departamento sean asignados a una sola LAN y que no se permita que el tráfico salga de ésta. A los directivos no les agrada escuchar que un arreglo de este tipo sólo es posible si todos los usuarios de un departamento están en oficinas adyacentes, sin más gente entre ellos.

Un segundo aspecto es la carga. Algunas LANs se utilizan mucho más que otras, y en ocasiones podría ser necesario separarlas. Por ejemplo, si los usuarios de investigaciones realizan toda clase de experimentos que en ocasiones se les van de las manos y saturan su LAN, tal vez a los usuarios de contabilidad no les agrade tener que ceder parte de su capacidad para ayudarles.

Un tercer aspecto es la difusión. La mayoría de las LANs soporta la difusión, y muchos protocolos de la capa superior utilizan ampliamente esta característica. Por ejemplo, cuando un usuario desea enviar un paquete a una dirección IP *x*, ¿cómo sabe a cuál dirección MAC enviar la trama? En el capítulo 5 estudiaremos este asunto, pero en pocas palabras, la respuesta es que debe difundir una trama con la pregunta: ¿Quién posee la dirección IP *x*?, y esperar la respuesta. Existen muchos más ejemplos del uso de la difusión. Conforme se interconectan más y más LANs, la cantidad de difusiones (*broadcasts*) que pasan por cada máquina se incrementa de manera lineal con el número de máquinas.

Las difusiones tienen el problema asociado de que de vez en cuando las interfaces de red se averían y empiezan a generar flujos interminables de tramas de difusión. El resultado de una **tormenta de difusión** es que 1) las tramas ocupan toda la capacidad de la LAN, y 2) las máquinas de todas las LANs interconectadas se atascan procesando y descartando las tramas difundidas.

A primera vista parecería que la magnitud de las tormentas de difusión podría limitarse separando las LANs con puentes o conmutadores, pero si el objetivo es conseguir transparencia (es decir, que una máquina pueda cambiarse a una LAN distinta al otro lado del puente sin que nadie lo note), entonces los puentes tienen que reenviar las tramas difundidas.

Después de analizar por qué las empresas podrían requerir varias LANs con un alcance limitado, regresemos al problema de desacoplar la topología lógica de la física. Supongamos que un usuario es transferido de un departamento a otro de la misma empresa sin que se le cambie de oficina o que se le cambia de oficina pero no de departamento. En un entorno de concentradores con cables, cambiar al usuario a la LAN correcta implica que el administrador de la red debe desplazarse hasta el gabinete de cableado, quitar de un concentrador el conector de la máquina del usuario e insertar el mismo conector en otro concentrador.

En muchas empresas, los cambios organizacionales ocurren todo el tiempo, lo cual quiere decir que los administradores de sistemas desperdician mucho tiempo quitando y metiendo conectores de un lado a otro. Asimismo, en algunos casos el cambio no se puede realizar de ninguna manera porque el cable de par trenzado de la máquina del usuario está demasiado lejos del concentrador correcto (por ejemplo, en otro edificio).

En respuesta a la demanda de mayor flexibilidad por parte de los usuarios, los fabricantes de redes empezaron a trabajar en una forma de volver a cablear edificios completos mediante software.

El concepto que surgió se denomina **VLAN** (**LAN Virtual**) e incluso fue estandarizado por el comité 802. Ahora se encuentra funcionando en muchas organizaciones. Analicémoslo brevemente. Si desea información adicional, vea (Breyer y Riley, 1999, y Seifert, 2000).

Las VLANs se fundamentan en conmutadores especialmente diseñados para este propósito, aunque también podrían contar con algunos concentradores, como se muestra en la figura 4-48. Para configurar una red VLAN, el administrador de la red decide cuántas VLANs habrá, qué computadoras habrá en cuál VLAN y cómo se llamarán las VLANs. Es común nombrar mediante colores a las VLANs (de manera informal), ya que de esta manera es posible imprimir diagramas en color que muestren la disposición física de las máquinas, con los miembros de la LAN roja en rojo, los de la LAN verde en verde, etc. De esta forma, tanto el diseño físico como el lógico se pueden reflejar en un solo esquema.

Por ejemplo, considere las cuatro LANs de la figura 4-49(a), en la cual ocho de las máquinas pertenecen a la VLAN G (gris) y siete forman parte de la VLAN W (blanca). Dos puentes, *B1* y *B2*, conectan las cuatro LANs físicas. Si se utiliza cableado de par trenzado centralizado, también podría haber cuatro concentradores (que no se muestran), pero desde el punto de vista lógico un cable con múltiples derivaciones y un concentrador representan lo mismo. Al esquematizar la figura de esta forma se aprecia un poco menos amontonada. Asimismo, el término "puente" se emplea actualmente cuando hay varias máquinas en cada puerto, como es el caso de esta figura, pero de otra manera, "puente" y "conmutador" en esencia son indistintos. En la figura 4-49(b) se muestran las mismas máquinas y las mismas VLANs, aunque en esta ocasión con conmutadores y una sola máquina en cada puerto.

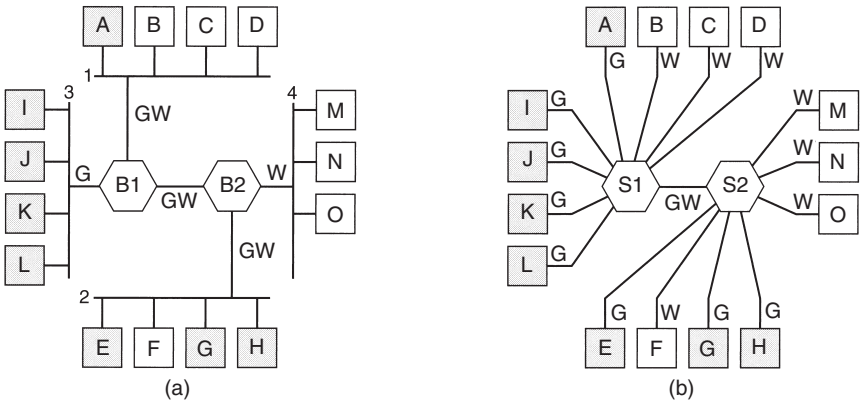


Figura 4-49. (a) Cuatro LANs físicas organizadas en dos VLANs, en gris y blanco, mediante dos puentes. (b) Las mismas 15 máquinas organizadas en dos VLANs mediante conmutadores.

Para que las VLANs funcionen correctamente, las tablas de configuración se deben establecer en los puentes o en los conmutadores. Estas tablas indican cuáles VLANs se pueden acceder a través de qué puertos (líneas). Cuando una trama llega procedente de, digamos, la VLAN gris, debe

reenviarse a todos los puertos G. De este modo se puede enviar tráfico ordinario (es decir, de unidifusión), así como de difusión y multidifusión.

Observe que un puerto puede marcarse con varios colores de VLAN. Esto se aprecia con más claridad en la figura 4-49(a). Suponga que la máquina *A* difunde una trama. El puente *B1* la recibe y detecta que proviene de una máquina de la VLAN gris, por lo cual la reenvía a todos los puertos G (excepto al puerto del que procede). Puesto que *B1* tiene sólo otros dos puertos y ambos son G, la trama se envía a ambos.

En *B2* la situación es distinta. Aquí el puerto sabe que no hay máquinas grises en la LAN 4, por lo que no envía la trama ahí. Sólo la manda a la LAN 2. Si uno de los usuarios de la LAN 4 debe cambiar de departamento y trasladarse a la VLAN gris, entonces las tablas de *B2* deben actualizarse y renombrar el puerto como GW en lugar de W. Si la máquina *F* cambia a gris, entonces el puerto de la LAN 2 debe renombrarse como G en lugar de GW.

Imaginemos ahora que todas las máquinas de las LANs 2 y 4 cambian a gris. En ese caso, no sólo los puertos de *B2* para las LANs 2 y 4 se renombran como G, sino que también el puerto de *B1* que va a *B2* debe renombrarse como G en lugar de GW porque las tramas blancas que llegan a *B1* procedentes de las LANs 1 y 3 ya no tienen que reenviarse a *B2*. En la figura 4-49(b) permanece la misma situación, sólo que aquí todos los puertos que van hacia una sola máquina se marcan con un solo color porque sólo hay una VLAN.

Hasta aquí hemos dado por sentado que los puentes y los conmutadores saben de alguna forma qué color tienen las tramas que llegan. ¿Cómo lo saben? Por medio de los tres métodos siguientes:

1. A cada puerto se le asigna un color de VLAN.
2. A cada dirección MAC se le asigna un color de VLAN.
3. A cada protocolo de la capa 3 o a cada dirección IP se le asigna un color de VLAN.

En el primer método, cada puerto se marca con un color de VLAN. Sin embargo, este método sólo funciona si todas las máquinas de un puerto pertenecen a la misma VLAN. En la figura 4-49(a), esta propiedad se aplica a *B1* para el puerto de la LAN 3 pero no al puerto de la LAN 1.

En el segundo método, el puente o el conmutador tienen una tabla con las direcciones MAC de 48 bits de cada máquina conectada a ellos, junto con la VLAN a la cual pertenece la máquina. Bajo estas condiciones, es factible mezclar VLANs en una LAN física, como en el caso de la LAN 1 de la figura 4-49(a). Cuando llega una trama, todo lo que tienen que hacer el puente o el conmutador es extraer la dirección MAC y buscarla en una tabla para averiguar de qué VLAN proviene.

En el tercer método el puente o el conmutador examinan el campo de carga útil de la trama, por ejemplo, para clasificar todas las máquinas IP en una VLAN y todas las máquinas AppleTalk en otra. En el primer caso, la dirección IP se puede utilizar también para identificar a la máquina. Esta estrategia es más útil cuando varias máquinas son computadoras portátiles que se pueden acoplar en cualquiera de diversos lugares. Puesto que cada estación de acoplamiento tiene su propia dirección MAC, el solo hecho de saber cuál estación de acoplamiento se utilizó no indica en absoluto en cuál VLAN se encuentra la computadora portátil.

El único problema de este enfoque es que transgrede la regla más elemental de la conectividad: independencia de las capas. A la capa de enlace de datos no le incumbe lo que esté en el campo de carga útil. No le corresponde analizar la carga útil ni tomar decisiones con base en el contenido. Una consecuencia del uso de este enfoque es que un cambio en el protocolo de la capa 3 (por ejemplo, una actualización de IPv4 a IPv6) ocasiona que los conmutadores fallen repentinamente. Por desgracia, en el mercado hay conmutadores que funcionan de esta manera.

Por supuesto, no hay nada de malo en enrutar con base en las direcciones IP —casi todo el capítulo 5 está dedicado al enrutamiento IP— pero al mezclar las capas se pueden propiciar problemas. Un fabricante de conmutadores podría minimizar esta situación argumentando que sus conmutadores comprenden tanto IPv4 como IPv6, así que no hay problema. ¿Pero qué pasará cuando surja IPv7? En tal caso, el fabricante tal vez dirá: Compre nuevos conmutadores, ¿cuál es el problema?

El estándar IEEE 802.1Q

Al ahondar un poco más en este asunto salta a la vista que lo importante es la VLAN de la trama, no la VLAN de la máquina emisora. Si hubiera alguna forma de identificar la VLAN en el encabezado de la trama, se desvanecería la necesidad de examinar la carga útil. Para una LAN nueva como 802.11 u 802.16 habría sido bastante fácil tan sólo agregar un campo de VLAN en el encabezado. De hecho, el campo *Identificador de conexión* del estándar 802.16 es muy parecido a un identificador VLAN. ¿Pero qué se puede hacer con Ethernet, que es la LAN dominante y no tiene campos disponibles para el identificador VLAN?

El comité IEEE 802 se enfrentó a este problema en 1995. Después de muchas discusiones, hizo lo impensable y cambió el encabezado de Ethernet. El nuevo formato se publicó en el estándar **802.1Q** del IEEE, emitido en 1998. El nuevo formato contiene una etiqueta VLAN; en breve la examinaremos. No es de sorprender que el cambio de algo ya bien establecido como el encabezado de Ethernet no sea nada sencillo. Algunas de las preguntas que surgen son:

1. ¿Tenemos que tirar a la basura cientos de millones de tarjetas Ethernet existentes?
2. Si no es así, ¿quién generará los nuevos campos?
3. ¿Qué sucederá con las tramas que ya tienen el tamaño máximo?

Por supuesto, el comité 802 estaba consciente de estos problemas y tenía que encontrar soluciones, lo cual hizo.

La clave para la solución consiste en comprender que los campos VLAN sólo son utilizados por los puentes y los conmutadores, no por las máquinas de los usuarios. De ahí que en la figura 4-49 no sea realmente necesario que estén presentes en las líneas que van hacia las estaciones finales siempre y cuando se encuentren en la línea entre los puentes o los conmutadores. Así, para utilizar VLANs, los puentes o los conmutadores deben tener soporte para VLAN, pero ese ya era un requisito. Ahora sólo estamos agregando el requisito adicional de que tengan soporte para 802.1Q, requisito que los nuevos ya cubren.

Respecto a la cuestión de si es necesario desechar todas las tarjetas Ethernet existentes, la respuesta es no. Recuerde que el comité 802.3 no pudo conseguir que la gente cambiara el campo *Tipo* por un campo *Longitud.* Ya podrá imaginar la reacción ante el anuncio de que todas las tarjetas Ethernet existentes tuvieran que desecharse. Sin embargo, se espera que las nuevas tarjetas Ethernet que salgan al mercado tendrán compatibilidad con el 802.1Q y llenarán correctamente el campo VLAN.

Por lo tanto, si el emisor no generará los campos VLAN, ¿quién lo hará? La respuesta es que el primer puente o conmutador con soporte de VLAN en recibir una trama los agregará y el último que los reciba los eliminará. ¿Pero cómo sabrán cuál trama corresponde a cuál VLAN? Bueno, el primer puente o conmutador podría asignar un número de VLAN a un puerto, analizar la dirección MAC o (¡Dios no lo quiera!) examinar la carga útil. Mientras todas las tarjetas Ethernet no se apeguen al estándar 802.1Q, estaremos en donde empezamos. La esperanza real es que todas las tarjetas Gigabit Ethernet se apegarán a 802.1Q desde el principio y que en tanto la gente se actualiza a Gigabit Ethernet, el 802.1Q se introducirá automáticamente. En cuanto al problema de las tramas mayores a 1518 bytes, el 802.1Q tan sólo incrementó el límite a 1522 bytes.

Durante el proceso de transición, muchas instalaciones tendrán algunas máquinas heredadas (en su mayoría, clásicas o Fast Ethernet) que no soportarán VLAN y otras (por lo general, Gigabit Ethernet) que sí lo harán. Esta situación se ilustra en la figura 4-50, en donde los símbolos sombreados representan máquinas que soportan VLAN y los vacíos no las soportan. Por simplicidad, damos por sentado que todos los conmutadores soportan VLAN. Si no es así, el primer conmutador que soporte VLAN puede incorporar las etiquetas con base en las direcciones MAC o IP.

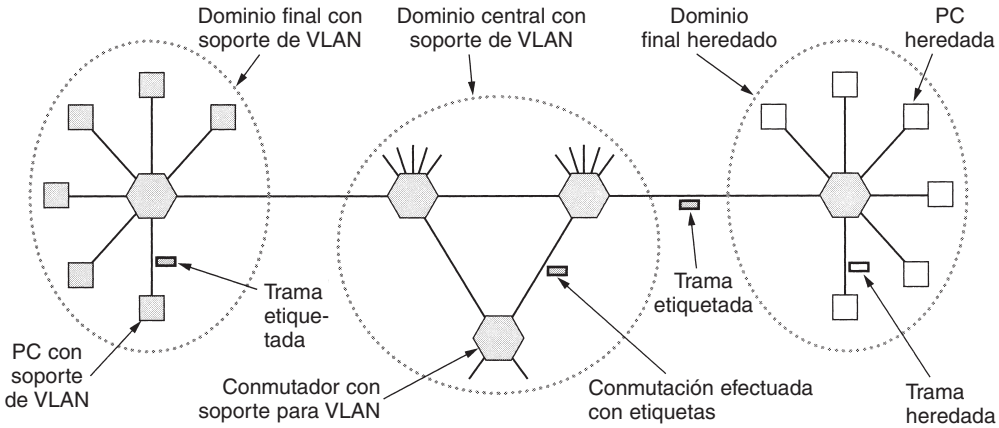


Figura 4-50. Transición de Ethernet heredada a Ethernet con soporte para VLAN. Los símbolos sombreados representan soporte para VLAN, a diferencia de los vacíos.

En esta figura, las tarjetas Ethernet con soporte para VLAN generan directamente tramas etiquetadas (es decir, 802.1Q) y la conmutación posterior se vale de estas etiquetas. Para realizar esta conmutación, los conmutadores tienen que saber cuáles VLANs están al alcance en cada puerto, lo mismo que antes. El hecho de saber que una trama pertenece a la VLAN gris no es de mucha

ayuda sino hasta que el conmutador sabe cuáles puertos tienen conexión con las máquinas de la VLAN gris. De esta forma, el conmutador necesita una tabla indexada por VLAN que le indique cuáles puertos puede utilizar y si éstos tienen soporte para VLAN o son heredados.

Cuando una PC heredada envía una trama a un conmutador con soporte para VLAN, el conmutador genera una trama etiquetada apoyándose en el conocimiento que tiene de la VLAN del emisor (utilizando el puerto, la dirección MAC o la dirección IP). De ahí en adelante, no importa que el emisor sea una máquina heredada. Asimismo, un conmutador que necesita entregar una trama etiquetada a una máquina heredada tiene que darle a la trama el formato heredado antes de entregarla.

Demos ahora un vistazo al formato de la trama 802.1Q, que se muestra en la figura 4-51. El único cambio es la adición de un par de campos de dos bytes. El primero es la *ID del protocolo de VLAN.* Siempre tiene el valor 0x8100. Como esta cifra es mayor que 1500, todas las tarjetas Ethernet lo interpretan como un tipo más que como una longitud. Lo que una tarjeta heredada hace con una trama como ésta es discutible porque dichas tramas no deberían enviarse a tarjetas heredadas.

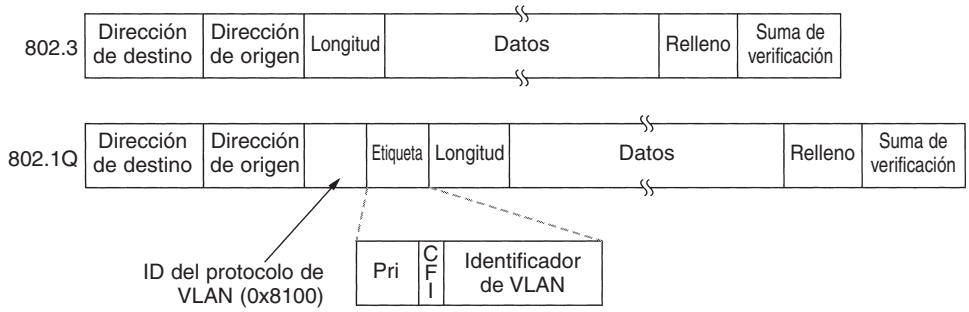


Figura 4-51. Formatos de trama Ethernet 802.3 (heredada) y 802.1Q.

El segundo campo de dos bytes contiene tres subcampos. El principal es el *Identificador de VLAN*, que ocupa los 12 bits de orden menor. Éste es el punto central de la cuestión: ¿a qué VLAN pertenece la trama? El campo *Prioridad* de 3 bits no tiene absolutamente nada que ver con las VLANs, pero como el cambio del encabezado Ethernet es un suceso poco frecuente que tarda tres años y ocupa a un ciento de personas, ¿por qué no incorporarle algunas otras cosas buenas en el proceso? Este campo permite distinguir el tráfico en tiempo real estricto del tráfico en tiempo real flexible y del tráfico no sensible al retardo, con el propósito de ofrecer una mejor calidad de servicio sobre Ethernet. Esto es necesario para el transporte de voz sobre Ethernet (aunque, para ser justos, IP tiene un campo similar desde hace un cuarto de siglo y nadie lo utiliza).

El último bit, *CFI* (*Indicador del Formato Canónico*), debió haberse llamado *CEI* (*Indicador del Ego Corporativo*). Su propósito original era indicar las direcciones MAC *little endian* en comparación con las *big endian*, pero esto ha cambiado con el tiempo. Actualmente indica que la carga útil contiene una trama 802.5 congelada-seca que se espera encuentre otra LAN 802.5 en el destino cuando se transmita a través de Ethernet. Por supuesto, este arreglo no tiene absolutamente

nada que ver con las VLANs. Pero la política de los comités de estándares no difiere mucho de la política común: si votas por mi bit, votaré por el tuyo.

Como ya mencionamos, cuando una trama etiquetada llega a un conmutador con soporte para VLAN, éste utiliza el ID de la VLAN como índice de tabla para averiguar a cuáles puertos enviar la trama. ¿Pero de dónde proviene la tabla? Si se elabora en forma manual, tenemos que empezar desde cero: la configuración manual de los puentes. La ventaja de los puentes transparentes es que son *plug and play* y no requieren configuración manual. Sería un terrible retroceso perder esa propiedad. Por fortuna, los puentes con soporte para VLAN se pueden autoconfigurar al observar las etiquetas que arriben a ellos. Si una trama etiquetada como VLAN 4 llega al puerto 3, entonces aparentemente una máquina en el puerto 3 se encuentra en la VLAN 4. El estándar 802.1Q explica cómo construir las tablas de manera dinámica, en su mayor parte referenciando porciones apropiadas del algoritmo de Perlman estandarizado en el 802.1D.

Antes de abandonar el tema del enrutamiento para VLAN, vale la pena hacer una última observación. Mucha gente de Internet y Ethernet es fanática de las redes no orientadas a la conexión y se oponen terminantemente a todo lo que huela a conexiones en las capas de enlace de datos y de red. No obstante, las VLANs incluyen un aspecto que es sorprendentemente similar a una conexión. Para utilizar las VLANs de manera apropiada, cada trama lleva un identificador especial nuevo que se utiliza como índice en una tabla dentro del conmutador para averiguar el destino al que se debe enviar la trama. Esto es precisamente lo que se hace en las redes orientadas a la conexión. En las redes no orientadas a la conexión, la dirección de destino es la que se utiliza para el enrutamiento, no un tipo de identificador de conexión. En el capítulo 5 abundaremos en este conexionismo gradual.

4.8 RESUMEN

Algunas redes tienen un solo canal que se usa para todas las comunicaciones. En estas redes, el aspecto clave del diseño es la asignación del canal entre las estaciones competidoras que desean usarlo. Se han desarrollado muchos algoritmos de asignación de canal. En la figura 4-52 se presenta un resumen de algunos de los métodos de asignación de canal más importantes.

Los métodos de asignación más sencillos son la FDM y la TDM; son eficientes con un número de estaciones pequeño y fijo y tráfico continuo. Ambos esquemas se usan ampliamente en estas circunstancias; por ejemplo, para dividir el ancho de banda de las troncales telefónicas.

Con un número grande y variable de estaciones, o con un tráfico en ráfagas, la FDM y la TDM son soluciones pobres. Se ha propuesto como alternativa el protocolo ALOHA, con y sin ranuras y control. El ALOHA y sus muchas variantes y derivaciones han sido ampliamente estudiados, analizados y usados en sistemas reales.

Cuando puede detectarse el estado del canal, las estaciones pueden evitar el comienzo de una transmisión mientras otra estación está transmitiendo. Esta técnica, la detección de portadora, ha producido varios protocolos que pueden usarse en LANs y MANs.

Método	Descripción
FDM	Dedica una banda de frecuencia a cada estación
WDM	Esquema FDM dinámico para fibra
TDM	Dedica una ranura de tiempo a cada estación
ALOHA puro	Transmisión asíncrona en cualquier momento
ALOHA ranurado	Transmisión aleatoria en ranuras de tiempo bien definidas
CSMA persistente-1	Acceso múltiple con detección de portadora estándar
CSMA no persistente	Retardo aleatorio cuando se detecta que el canal está ocupado
CSMA persistente-p	CSMA, pero con una probabilidad de persistencia p
CSMA/CD	CSMA, pero aborta al detectar una colisión
Mapa de bits	Calendarización *round robin* mediante mapa de bits
Conteo descendente binario	La estación disponible con el número más alto toma el turno
Recorrido de árbol	Contención reducida mediante habilitación selectiva
MACA, MACAW	Protocolos de LAN inalámbrica
Ethernet	CSMA/CD con retraso exponencial binario
FHSS	Espectro disperso con salto de frecuencia
DSSS	Espectro disperso de secuencia directa
CSMA/CA	Acceso múltiple con detección de portadora y evitación de colisiones

Figura 4-52. Métodos de asignación de canal y sistemas para canal común.

Se conoce una clase de protocolos que eliminan por completo la contención, o cuando menos la reducen considerablemente. El conteo binario descendente elimina por completo la contención. El protocolo de recorrido de árbol la reduce dividiendo dinámicamente las estaciones en dos grupos separados, uno que puede transmitir y otro que no. Se intenta hacer la división de tal manera que sólo una estación lista para transmitir pueda hacerlo.

Las LANs inalámbricas tienen sus propios problemas y soluciones. El problema principal lo causan las estaciones ocultas, por lo que el CSMA no funciona. Una clase de soluciones, tipificadas por MACA y MACAW, intenta estimular las transmisiones en las cercanías del destino, para hacer que el CSMA funcione mejor. También se usan el espectro disperso con salto de frecuencia y el espectro disperso de secuencia directa. El IEEE 802.11 combina CSMA y MACAW para producir CSMA/CA.

Ethernet predomina en el campo de las redes de área local. Utiliza CSMA/CD para la asignación de canal. Las primeras versiones empleaban un cable que serpenteaba entre las máquinas, pero en la actualidad son más comunes los cables de par trenzado hacia concentradores y conmutadores. Las velocidades se han incrementado de 10 Mbps a 1 Gbps y siguen en aumento.

Las LANs inalámbricas se están popularizando, y el 802.11 domina el campo. Su capa física permite cinco diferentes modos de transmisión, entre ellos el infrarrojo, diversos esquemas de

espectro disperso y un sistema FDM multicanal con una estación base en cada celda, aunque también puede funcionar sin ninguna. El protocolo es una variante de MACAW, con detección de portadora virtual.

Las MANs inalámbricas están empezando a aparecer. Son sistemas de banda amplia que utilizan radio para reemplazar la última milla en conexiones telefónicas. También utilizan técnicas tradicionales de modulación de banda estrecha. La calidad de servicio es importante, y el estándar 802.16 define cuatro clases (tasa de bits constante, dos tasas variables de bits y una de mejor esfuerzo).

El sistema Bluetooth también es inalámbrico, aunque está más enfocado a los sistemas de escritorio, para conectar diademas telefónicas y otros periféricos a las computadoras sin necesidad de cables. También se utiliza para conectar periféricos, como máquinas de fax, a los teléfonos móviles. Al igual que el 801.11, utiliza espectro disperso con saltos de frecuencia en la banda ISM. Debido al nivel de ruido esperado en muchos entornos y a la necesidad de interacción en tiempo real, sus diversos protocolos incorporan una sofisticada corrección de errores hacia delante.

Con tantas LANs diferentes, es necesaria una forma para interconectarlas. Los puentes y los conmutadores tienen este propósito. El algoritmo de árbol de expansión se utiliza para construir puentes *plug and play*. La VLAN es un nuevo desarrollo del mundo de la interconexión de LANs, que separa la topología lógica de la topología física de las LANs. Se ha introducido un nuevo formato para las tramas Ethernet (802.1Q), cuyo propósito es facilitar la utilización de las VLANs en las organizaciones.

PROBLEMAS

1. Para este problema, utilice una fórmula de este capítulo, pero primero enúnciela. Las tramas arriban de manera aleatoria a un canal de 100 Mbps para su transmisión. Si el canal está ocupado cuando arriba una trama, ésta espera su turno en una cola. La longitud de la trama se distribuye exponencialmente con una media de 10,000 bits/trama. Para cada una de las siguientes tasas de llegada de tramas, dé el retardo experimentado por la trama promedio, incluyendo tanto el tiempo de encolamiento como el de transmisión.
   (a) 90 tramas/seg.
   (b) 900 tramas/seg.
   (c) 9000 tramas/seg.

2. Un grupo de N estaciones comparte un canal ALOHA puro de 56 kbps. La salida de cada estación es una trama de 1000 bits en promedio cada 100 seg aun si la anterior no ha sido enviada (por ejemplo, las estaciones pueden almacenar en búfer las tramas salientes). ¿Cuál es el valor máximo de N?

3. Considere el retardo del ALOHA puro comparándolo con el ALOHA ranurado cuando la carga es baja. ¿Cuál es menor? Explique su respuesta.

4. Diez mil estaciones de reservaciones de una aerolínea compiten por un solo canal ALOHA ranurado. La estación promedio hace 18 solicitudes/hora. Una ranura dura 125 μseg. ¿Cuál es la carga aproximada total del canal?

5. Una gran población de usuarios de ALOHA genera 50 solicitudes/seg incluidas tanto originales como retransmisiones. El tiempo se divide en ranuras de 40 mseg.
(a) ¿Cuál es la oportunidad de éxito en el primer intento?
(b) ¿Cuál es la probabilidad exacta de k colisiones y después tener éxito?
(c) ¿Cuál es el número esperado de intentos de transmisión necesarios?

6. Mediciones en un canal ALOHA ranurado con una cantidad infinita de usuarios muestra que 10% de las ranuras están inactivas.
(a) ¿Qué carga, G, tiene el canal?
(b) ¿Cuál es la velocidad real de transporte?
(c) ¿El canal está subcargado o sobrecargado?

7. En un sistema ALOHA ranurado de población infinita, la cantidad media de ranuras que espera una estación entre una colisión y su retransmisión es de 4. Grafique la curva de retardo contra velocidad real de transporte de este sistema.

8. ¿Cuánto debe esperar una estación, s, en el peor de los casos, antes de empezar a transmitir su trama sobre una LAN que utiliza
(a) el protocolo básico de mapa de bits?
(b) el protocolo de Mok y Ward con cambio de números virtuales de estación?

9. Una LAN usa la versión de Mok y Ward del conteo descendente binario. En cierto momento, las 10 estaciones tienen los números de estación virtual 8, 2, 4, 5, 1, 7, 3, 6, 9 y 0. Las tres estaciones siguientes que van a enviar son: 4, 3 y 9, en ese orden. ¿Cuáles son los nuevos números de estación virtual una vez que las tres han terminado sus transmisiones?

10. Dieciséis estaciones contienden por un canal compartido que usa el protocolo de recorrido de árbol. Si todas las estaciones cuyas direcciones son números primos de pronto quedaran listas al mismo tiempo, ¿cuántas ranuras de bits se necesitan para resolver la contención?

11. Un conjunto de 2^n estaciones usa el protocolo de recorrido de árbol adaptable para arbitrar el acceso a un cable compartido. En cierto momento, dos de ellas quedan listas. ¿Cuál es el número de ranuras mínimo, máximo y medio para recorrer el árbol si $2^n \gg 1$?

12. Las LANs inalámbricas que estudiamos usaban protocolos como MACA en lugar de CSMA/CD. ¿En qué condiciones sería posible usar CSMA/CD?

13. ¿Qué propiedades tienen en común los protocolos de acceso a canal WDMA y GSM? Consulte el GSM en el capítulo 2.

14. Seis estaciones, de A a F, se comunican mediante el protocolo MACA. ¿Es posible que dos transmisiones tengan lugar de manera simultánea? Explique su respuesta.

15. Un edificio de oficinas de siete pisos tiene 15 oficinas adyacentes por piso. Cada oficina contiene un enchufe de pared para una terminal en la pared frontal, por lo que los enchufes forman una retícula triangular en el plano vertical, con una separación de 4 m entre enchufes, tanto vertical como horizontalmente. Suponiendo que es factible tender un cable recto entre cualquier par de enchufes, horizontal, vertical o diagonalmente, ¿cuántos metros de cable se necesitan para conectar todos los enchufes usando
(a) una configuración en estrella con un solo enrutador en medio?
(b) una LAN 802.3?

16. ¿Cuál es la tasa de baudios de la Ethernet de 10 Mbps estándar?

17. Bosqueje la codificación Manchester para el flujo de bits: 0001110101.

18. Bosqueje la codificación diferencial Manchester para el flujo de bits del problema anterior. Suponga que la línea se encuentra inicialmente en el estado bajo.

19. Una LAN CSMA/CD (no la 802.3) de 10 Mbps y 1 km de largo tiene una velocidad de propagación de 200 m/μseg. En este sistema no se permiten los repetidores. Las tramas de datos tienen 256 bits de longitud, incluidos 32 bits de encabezado, suma de verificación y un poco más de sobrecarga. La primera ranura de bits tras una transmisión exitosa se reserva para que el receptor capture el canal y envíe una trama de confirmación de recepción de 32 bits. ¿Cuál es la tasa de datos efectiva, excluyendo la sobrecarga, suponiendo que no hay colisiones?

20. Dos estaciones CSMA/CD intentan transmitir archivos grandes (multitrama). Tras el envío de cada trama, contienden por el canal usando el algoritmo de retroceso exponencial binario. ¿Cuál es la probabilidad de que la contención termine en la ronda k, y cuál es la cantidad media de rondas por periodo de contención?

21. Considere la construcción de una red CSMA/CD que opere a 1 Gbps a través de un cable de 1 km de longitud sin repetidores. La velocidad de la señal en el cable es de 200,000 km/seg. ¿Cuál es el tamaño mínimo de trama?

22. Un paquete IP que se transmitirá a través de Ethernet tiene 60 bytes de longitud, incluyendo todos los encabezados. Si no se utiliza LLC, ¿se necesita relleno en la trama Ethernet, y de ser así, cuántos bytes?

23. Las tramas Ethernet deben tener al menos 64 bytes de longitud para asegurar que el transmisor permanezca en línea en caso de que ocurra una colisión en el extremo más lejano del cable. Fast Ethernet tiene el mismo tamaño mínimo de trama de 64 bytes pero puede recibir los bits diez veces más rápido. ¿Cómo es posible mantener el mismo tamaño mínimo de trama?

24. Algunos libros citan que el tamaño máximo de una trama Ethernet es de 1518 bytes en lugar de 1500. ¿Están en un error? Explique su respuesta.

25. La especificación 1000Base-SX indica que el reloj debe correr a 1250 MHz, aun cuando se supone que Gigabit Ethernet funciona a 1 Gbps. ¿Esta velocidad más alta confiere un margen adicional de seguridad? Si no es así, ¿qué está pasando?

26. ¿Cuántas tramas por segundo puede manejar Gigabit Ethernet? Reflexione con cuidado y tome en cuenta todos los casos relevantes. *Sugerencia:* es importante el hecho de que se trata de *Gigabit* Ethernet.

27. Mencione dos redes que permitan empaquetar tramas una tras otra. ¿Por qué es importante esta característica?

28. En la figura 4.27 se muestran cuatro estaciones, *A, B, C* y *D.* ¿Cuál de las dos últimas estaciones cree que está más cerca de *A* y por qué?

29. Suponga que una LAN 802.11b de 11 Mbps transmite tramas de 64 bytes una tras otra sobre un canal de radio con una tasa de error de 10^{-7}. ¿Cuántas tramas por segundo en promedio resultarán dañadas?

30. Una red 802.16 tiene un ancho de canal de 20 MHz. ¿Cuántos bits/seg se pueden enviar a una estación suscrita?

31. El IEEE 802.16 soporta cuatro clases de servicio. ¿Cuál es la mejor clase de servicio para enviar vídeo sin comprimir?

32. Dé dos razones por las cuales las redes podrían usar un código de corrección de errores en lugar de detección de errores y retransmisión.

33. En la figura 4-35 podemos ver que un dispositivo Bluetooth puede estar en dos *piconets* al mismo tiempo. ¿Hay alguna razón por la cual un dispositivo no pueda fungir como maestro en ambas al mismo tiempo?

34. La figura 4-25 muestra varios protocolos de capa física. ¿Cuál de éstos está más cercano al protocolo de capa física Bluetooth? ¿Cuál es la principal diferencia entre ambos?

35. Bluetooth soporta dos tipos de enlaces entre un maestro y un esclavo. ¿Cuáles son y para qué se utiliza cada uno?

36. Las tramas de *beacon* en el espectro disperso con salto de frecuencia, variante del 802.11, contienen el tiempo de permanencia. ¿Cree que las tramas de *beacon* análogas de Bluetooth también contienen tiempo de permanencia? Explique su respuesta.

37. Considere las LANs interconectadas que se muestran en la figura 4-44. Suponga que los *hosts a* y *b* se encuentran en la LAN 1, *c* está en la LAN 2 y *d* está en la LAN 8. En principio, las tablas de *hash* de todos los puentes están vacías y se utiliza el árbol de expansión que se muestra en la figura 4-44(b). Muestre la manera en que cambian las tablas de *hash* de los diversos puentes después de que cada uno de los siguientes sucesos ocurren en secuencia, primero (a) y a continuación (b), y así sucesivamente.
(a) *a* envía a *d.*
(b) *c* envía a *a.*
(c) *d* envía a *c.*
(d) *d* pasa a la LAN 6.
(e) *d* envía a *a.*

38. Una consecuencia del uso de un árbol de expansión para reenviar tramas en una LAN extendida es que algunos puentes tal vez no participen en absoluto en el reenvío de tramas. Identifique tres puentes que se encuentren en esta situación en la figura 4-44. ¿Hay alguna razón para conservar estos puentes, aun cuando no se utilicen para el reenvío?

39. Suponga que un conmutador tiene tarjetas de línea para cuatro líneas de entrada. Con frecuencia, una trama que llega en una de las líneas tiene que salir en otra línea de la misma tarjeta. ¿A qué decisiones se enfrenta el diseñador del conmutador como resultado de esta situación?

40. Un conmutador diseñado para Fast Ethernet tiene una tarjeta madre que puede transportar 10 Gbps. ¿Cuántas tramas/seg puede manejar en el peor de los casos?

41. Considere la red de la figura 4-49(a). Si la máquina *J* tuviera que volverse blanca repentinamente, ¿serían necesarios cambios para el etiquetado? Si es así, ¿cuáles?

42. Describa brevemente la diferencia entre los conmutadores de almacenamiento y reenvío y los *cut-through*.

43. Los conmutadores de almacenamiento y reenvío tienen una ventaja sobre los *cut-through* en relación con las tramas dañadas. Explique cuál es.

44. Las tablas de configuración son necesarias en los conmutadores y los puentes para que las VLANs funcionen. ¿Qué pasaría si las VLANs de la figura 4-49(a) utilizaran concentradores en vez de cables con múltiples derivaciones? ¿Los concentradores también necesitarían tablas de configuración? ¿Por qué sí o por qué no?

45. En la figura 4-50 el conmutador del dominio final heredado en la parte derecha tiene soporte para VLAN. ¿Sería posible utilizar ahí un conmutador heredado? Si es así, ¿cómo funcionaría? En caso contrario, ¿por qué no?

46. Escriba un programa para simular el comportamiento del protocolo CSMA/CD sobre Ethernet cuando hay *N* estaciones listas para transmitir en el momento en que se está transmitiendo una trama. El programa deberá informar las veces que cada estación inicia exitosamente el envío de su trama. Suponga que un pulso de reloj ocurre una vez cada ranura de tiempo (51.2 microsegundos) y que la detección de una colisión y el envío de una secuencia atorada tarda una ranura de tiempo. Todas las tramas tienen la longitud máxima permitida.

5

LA CAPA DE RED

La capa de red se encarga de llevar los paquetes desde el origen hasta el destino. Llegar al destino puede requerir muchos saltos por enrutadores intermedios. Esta función ciertamente contrasta con la de la capa de enlace de datos, que sólo tiene la meta modesta de mover tramas de un extremo del cable al otro. Por lo tanto, la capa de red es la capa más baja que maneja la transmisión de extremo a extremo.

Para lograr su cometido, la capa de red debe conocer la topología de la subred de comunicación (es decir, el grupo de enrutadores) y elegir las rutas adecuadas a través de ella; también debe tener cuidado al escoger las rutas para no sobrecargar algunas de las líneas de comunicación y los enrutadores y dejar inactivos a otros. Por último, cuando el origen y el destino están en redes diferentes, ocurren nuevos problemas. La capa de red es la encargada de solucionarlos. En este capítulo estudiaremos todos estos temas y los ilustraremos principalmente valiéndonos de Internet y de su protocolo de capa de red, IP, aunque también veremos las redes inalámbricas.

5.1 ASPECTOS DE DISEÑO DE LA CAPA DE RED

En las siguientes secciones presentaremos una introducción a algunos de los problemas que deben enfrentar los diseñadores de la capa de red. Estos temas incluyen el servicio proporcionado a la capa de transporte y el diseño interno de la subred.

5.1.1 Conmutación de paquetes de almacenamiento y reenvío

Antes de iniciar una explicación sobre los detalles de la capa de red, probablemente valga la pena volver a exponer el contexto en el que operan los protocolos de esta capa. En la figura 5-1 se muestra dicho contexto. Los componentes principales del sistema son el equipo de la empresa portadora (enrutadores conectados mediante líneas de transmisión), que se muestra dentro del óvalo sombreado, y el equipo del cliente, que se muestra fuera del óvalo. El *host H1* está conectado de manera directa al enrutador de una empresa portadora, *A*, mediante una línea alquilada. En contraste, *H2* se encuentra en una LAN con un enrutador, *F*, el cual es propiedad de un cliente, quien lo maneja. Este enrutador también tiene una línea alquilada hacia el equipo de la empresa portadora. Mostramos *F* fuera del óvalo porque no pertenece a la empresa portadora, pero en términos de construcción, software y protocolos, tal vez no sea diferente de los enrutadores de la empresa portadora. Si bien es debatible el hecho de que este enrutador pertenezca a la subred, para los propósitos de este capítulo, los enrutadores del cliente son considerados como parte de la subred porque se valen de los mismos algoritmos que los enrutadores de la empresa portadora (y nuestro interés principal aquí son los algoritmos).

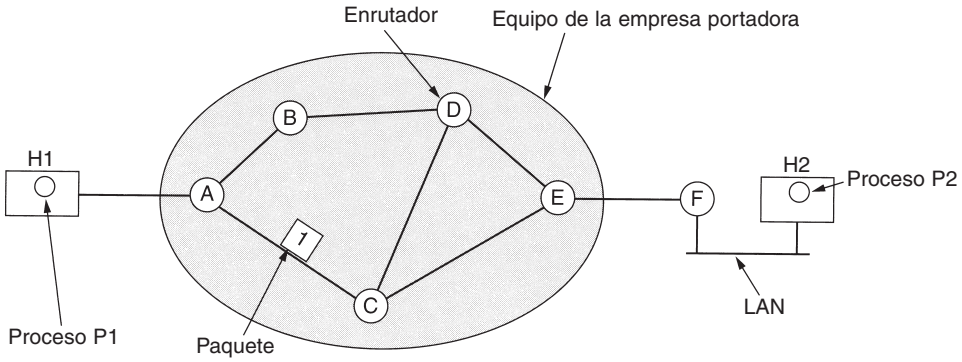


Figura 5-1. El entorno de los protocolos de la capa de red.

Este equipo se utiliza como sigue. Un *host* transmite al enrutador más cercano un paquete que tiene por enviar, ya sea en su propia LAN o a través de un enlace punto a punto con la empresa portadora. El paquete se almacena ahí hasta que haya llegado por completo, a fin de que la suma de verificación pueda comprobarse. Después se reenvía al siguiente enrutador de la ruta hasta que llegue al *host* de destino, donde se entrega. Este mecanismo se conoce como conmutación de paquetes de almacenamiento y reenvío, como vimos en capítulos anteriores.

5.1.2 Servicios proporcionados a la capa de transporte

La capa de red proporciona servicios a la capa de transporte en la interfaz capa de red/capa de transporte. Una pregunta importante es qué tipo de servicios proporciona la capa de red a la capa de transporte. Los servicios de la capa de red se han diseñado con los siguientes objetivos en mente.

1. Los servicios deben ser independientes de la tecnología del enrutador.
2. La capa de transporte debe estar aislada de la cantidad, tipo y topología de los enrutadores presentes.
3. Las direcciones de red disponibles para la capa de transporte deben seguir un plan de numeración uniforme, aun a través de varias LANs y WANs.

Dadas estas metas, los diseñadores de la capa de red tienen mucha libertad para escribir especificaciones detalladas de los servicios que se ofrecerán a la capa de transporte. Con frecuencia esta libertad degenera en una batalla campal entre dos bandos en conflicto. La discusión se centra en determinar si la capa de red debe proporcionar servicio orientado o no orientado a la conexión.

Un bando (representado por la comunidad de Internet) alega que la tarea del enrutador es mover bits de un lado a otro, y nada más. Desde su punto de vista (basado en casi 30 años de experiencia con una red de computadoras real y operativa), la subred es inherentemente inestable, sin importar su diseño. Por lo tanto, los *hosts* deben aceptar este hecho y efectuar ellos mismos el control de errores (es decir, detección y corrección de errores) y el control de flujo.

Este punto de vista conduce directamente a la conclusión de que el servicio de red no debe ser orientado a la conexión, y debe contar tan sólo con las primitivas SEND PACKET y RECEIVE PACKET. En particular, no debe efectuarse ningún ordenamiento de paquetes ni control de flujo, pues de todos modos los *hosts* lo van a efectuar y probablemente se ganaría poco haciéndolo dos veces. Además, cada paquete debe llevar la dirección de destino completa, porque cada paquete enviado se transporta de manera independiente de sus antecesores, si los hay.

El otro bando (representado por las compañías telefónicas) argumenta que la subred debe proporcionar un servicio confiable, orientado a la conexión. Afirman que una buena guía son 100 años de experiencia exitosa del sistema telefónico mundial. Desde este punto de vista, la calidad del servicio es el factor dominante, y sin conexiones en la subred, tal calidad es muy difícil de alcanzar, especialmente para el tráfico de tiempo real como la voz y el vídeo.

Estas dos posturas se ejemplifican mejor con Internet y ATM. Internet ofrece servicio de capa de red no orientado a la conexión; las redes ATM ofrecen servicio de capa de red orientado a la conexión. Sin embargo, es interesante hacer notar que conforme las garantías de calidad del servicio se están volviendo más y más importantes, Internet está evolucionando. En particular, está empezando a adquirir propiedades que normalmente se asocian con el servicio orientado a la conexión, como veremos más adelante. En el capítulo 4 dimos un breve indicio de esta evolución en nuestro estudio sobre las VLANs.

5.1.3 Implementación del servicio no orientado a la conexión

Puesto que ya vimos las dos clases de servicios que la capa de red puede proporcionar a sus usuarios, es tiempo de analizar la manera en que funciona internamente esta capa. Se pueden realizar dos formas de organización distintas, dependiendo del tipo de servicio que se ofrezca. Si se ofrece el servicio no orientado a la conexión, los paquetes se colocan individualmente en la subred y se enrutan de manera independiente. No se necesita una configuración avanzada. En este

contexto, por lo general los paquetes se conocen como **datagramas** (en analogía con los telegramas) y la subred se conoce como **subred de datagramas.** Si se utiliza el servicio orientado a la conexión, antes de poder enviar cualquier paquete de datos, es necesario establecer una ruta del enrutador de origen al de destino. Esta conexión se conoce como **CV (circuito virtual),** en analogía con los circuitos físicos establecidos por el sistema telefónico, y la subred se conoce como **subred de circuitos virtuales.** En esta sección examinaremos las subredes de datagramas; en la siguiente analizaremos las subredes de circuitos virtuales.

A continuación veamos cómo funciona una subred de datagramas. Suponga que el proceso *P1* de la figura 5-2 tiene un mensaje largo para *P2*. Dicho proceso entrega el mensaje a la capa de transporte y le indica a ésta que lo envíe al proceso *P2* que se encuentra en el *host H2*. El código de la capa de transporte se ejecuta en *H1*, por lo general dentro del sistema operativo. Dicho código agrega un encabezado de transporte al mensaje y entrega el resultado a la capa de red, quizá otro procedimiento dentro del sistema operativo.

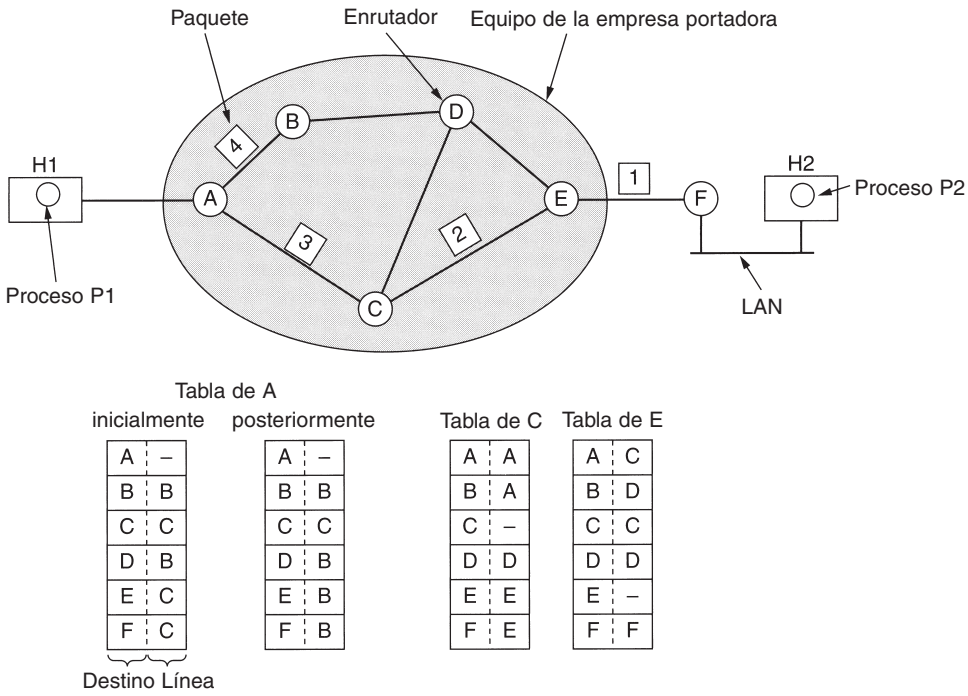


Figura 5-2. Enrutamiento dentro de una subred de datagramas.

Supongamos que el mensaje es cuatro veces más largo que el tamaño máximo de paquete, por lo que la capa de red tiene que dividirlo en cuatro paquetes, 1, 2, 3 y 4, y envía cada uno de ellos a la vez al enrutador *A* mediante algún protocolo punto a punto; por ejemplo, PPP. En este momento entra en acción la empresa portadora. Cada enrutador tiene una tabla interna que le indica a dónde enviar paquetes para cada destino posible. Cada entrada de tabla es un par que consiste en un

destino y la línea de salida que se utilizará para ese destino. Sólo se pueden utilizar líneas conectadas directamente. Por ejemplo, en la figura 5-2, *A* sólo tiene dos líneas de salida —a *B* y *C*—, por lo que cada paquete entrante debe enviarse a uno de estos enrutadores, incluso si el destino final es algún otro enrutador. En la figura, la tabla de enrutamiento inicial de *A* se muestra abajo de la leyenda "inicialmente".

Conforme los paquetes 1, 2 y 3 llegaron a *A*, se almacenaron unos momentos (para comprobar sus sumas de verificación). Después cada uno se reenvió a *C* de acuerdo con la tabla de *A*. Posteriormente, el paquete 1 se reenvió a *E* y después a *F*. Cuando llegó a *F*, se encapsuló en una trama de capa de enlace de datos y se envió a *H2* a través de la LAN. Los paquetes 2 y 3 siguieron la misma ruta.

Sin embargo, pasó algo diferente con el paquete 4. Cuando llegó a *A*, se envió al enrutador *B*, aunque también estaba destinado a *F*. Por alguna razón, *A* decidió enviar el paquete 4 por una ruta diferente a la de los primeros tres paquetes. Tal vez se enteró de que había alguna congestión de tráfico en alguna parte de la ruta *ACE* y actualizó su tabla de enrutamiento, como se muestra bajo la leyenda "posteriormente". El algoritmo que maneja las tablas y que realiza las decisiones de enrutamiento se conoce como **algoritmo de enrutamiento.** Los algoritmos de enrutamiento son uno de los principales temas que estudiaremos en este capítulo.

5.1.4 Implementación del servicio orientado a la conexión

Para servicio orientado a la conexión necesitamos una subred de circuitos virtuales. Veamos cómo funciona. El propósito de los circuitos virtuales es evitar la necesidad de elegir una nueva ruta para cada paquete enviado, como en la figura 5-2. En su lugar, cuando se establece una conexión, se elige una ruta de la máquina de origen a la de destino como parte de la configuración de conexión y se almacena en tablas dentro de los enrutadores. Esa ruta se utiliza para todo el tráfico que fluye a través de la conexión, exactamente de la misma forma en que funciona el sistema telefónico. Cuando se libera la conexión, también se termina el circuito virtual. Con el servicio orientado a la conexión, cada paquete lleva un identificador que indica a cuál circuito virtual pertenece.

Como ejemplo, considere la situación que se muestra en la figura 5-3. En ésta, el *host H1* ha establecido una conexión 1 con el *host H2*. Se recuerda como la primera entrada de cada una de las tablas de enrutamiento. La primera línea de la tabla *A* indica que si un paquete tiene el identificador de conexión 1 viene de *H1*, se enviará al enrutador *C* y se le dará el identificador de conexión 1. De manera similar, la primera entrada en *C* enruta el paquete a *E*, también con el identificador de conexión 1.

Ahora consideremos lo que sucede si *H3* también desea establecer una conexión con *H2*. Elige el identificador de conexión 1 (debido a que está iniciando la conexión y a que ésta es su única conexión) y le indica a la subred que establezca el circuito virtual. Esto nos lleva a la segunda fila de las tablas. Observe que aquí surge un problema debido a que aunque *A* sí puede saber con facilidad cuáles paquetes de conexión 1 provienen de *H1* y cuáles provienen de *H3*, *C* no puede hacerlo. Por esta razón, *A* asigna un identificador de conexión diferente al tráfico saliente para la segunda conexión. Con el propósito de evitar conflictos de este tipo, los enrutadores requieren

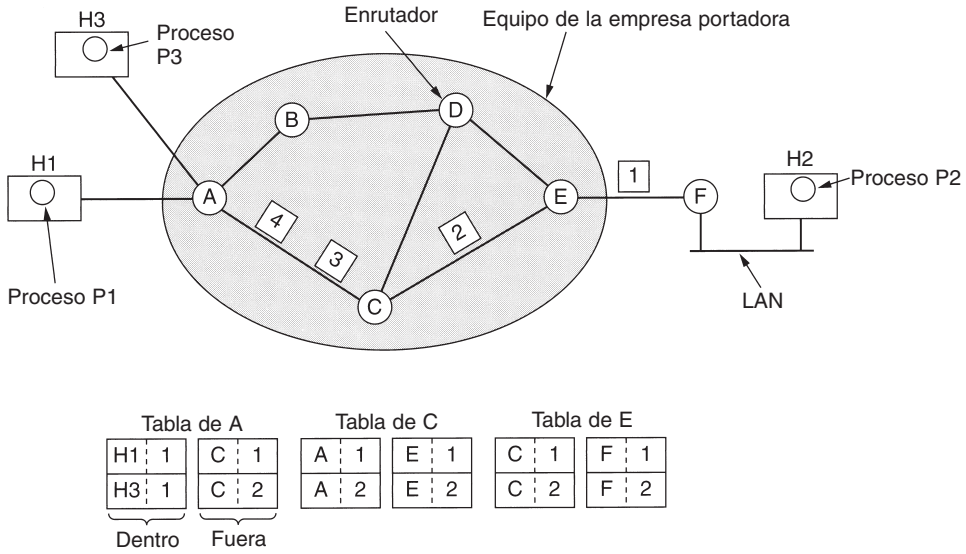


Figura 5-3. Enrutamiento dentro de una subred de circuitos virtuales.

la capacidad de reemplazar identificadores de conexión en los paquetes salientes. En algunos contextos a esto se le conoce como conmutación de etiquetas.

5.1.5 Comparación entre las subredes de circuitos virtuales y las de datagramas

Tanto los circuitos virtuales como los datagramas tienen sus seguidores y sus detractores. Ahora intentaremos resumir los argumentos de ambos bandos. Los aspectos principales se listan en la figura 5-4, aunque los puristas probablemente podrán encontrar ejemplos contrarios para todo lo indicado en la figura.

Dentro de la subred hay varios pros y contras entre los circuitos virtuales y los datagramas. Uno de ellos tiene que ver con el espacio de memoria del enrutador y el ancho de banda. Los circuitos virtuales permiten que los paquetes contengan números de circuito en lugar de direcciones de destino completas. Si los paquetes suelen ser bastante cortos, una dirección de destino completa en cada paquete puede representar una sobrecarga significativa y, por lo tanto, ancho de banda desperdiciado. El precio que se paga por el uso interno de circuitos virtuales es el espacio de tabla en los enrutadores. La mejor elección desde el punto de vista económico depende del costo relativo entre los circuitos de comunicación y la memoria de los enrutadores.

Otro punto por considerar es el del tiempo de configuración contra el tiempo de análisis de la dirección. El uso de circuitos virtuales requiere una fase de configuración, que consume tiempo y recursos. Sin embargo, determinar lo que hay que hacer con un paquete de datos en una subred de

Asunto	Subred de datagramas	Subred de circuitos virtuales
Configuración del circuito	No necesaria	Requerida
Direccionamiento	Cada paquete contiene la dirección de origen y de destino	Cada paquete contiene un número de CV corto
Información de estado	Los enrutadores no contienen información de estado de las conexiones	Cada CV requiere espacio de tabla del enrutador por conexión
Enrutamiento	Cada paquete se enruta de manera independiente	Ruta escogida cuando se establece el CV; todos los paquetes siguen esta ruta
Efecto de fallas del enrutador	Ninguno, excepto para paquetes perdidos durante una caída	Terminan todos los CVs que pasan a través del enrutador
Calidad del servicio	Difícil	Fácil si se pueden asignar suficientes recursos por adelantado para cada CV
Control de congestión	Difícil	Fácil si pueden asignarse por adelantado suficientes recursos a cada CV

Figura 5-4. Comparación de las subredes de datagramas y de circuitos virtuales.

circuitos virtuales es fácil: el enrutador simplemente usa el número de circuito para buscar en una tabla y encontrar hacia dónde va el paquete. En una subred de datagramas se requiere un procedimiento más complicado para localizar el destino del paquete.

Otra cuestión es la cantidad requerida de espacio de tabla en la memoria del enrutador. Una subred de datagramas necesita tener una entrada para cada destino posible, mientras que una subred de circuitos virtuales sólo necesita una entrada por cada circuito virtual. Sin embargo, esta ventaja es engañosa debido a que los paquetes de configuración de conexión también tienen que enrutarse, y a que utilizan direcciones de destino, de la misma forma en que lo hacen los datagramas.

Los circuitos virtuales tienen algunas ventajas en cuanto a la calidad del servicio y a que evitan congestiones en la subred, pues los recursos (por ejemplo, búferes, ancho de banda y ciclos de CPU) pueden reservarse por adelantado al establecerse la conexión. Una vez que comienzan a llegar los paquetes, estarán ahí el ancho de banda y la capacidad de enrutamiento necesarios. En una subred de datagramas es más difícil evitar las congestiones.

En los sistemas de procesamiento de transacciones (por ejemplo, las tiendas que llaman para verificar pagos con tarjeta de crédito), la sobrecarga requerida para establecer y terminar un circuito virtual puede ocupar mucho más tiempo que el uso real del circuito. Si la mayor parte del tráfico esperado es de este tipo, el uso de circuitos virtuales dentro de la subred tiene poco sentido. Por otra parte, aquí pueden ser de utilidad los circuitos virtuales permanentes, establecidos manualmente y con duración de meses o años.

Los circuitos virtuales también tienen un problema de vulnerabilidad. Si se cae un enrutador y se pierde su memoria, todos los circuitos virtuales que pasan por él tendrán que abortarse,

aunque se recupere un segundo después. Por el contrario, si se cae un enrutador de datagramas, sólo sufrirán los usuarios cuyos paquetes estaban encolados en el enrutador en el momento de la falla y, dependiendo de si ya se había confirmado o no su recepción, tal vez ni siquiera todos ellos. La pérdida de una línea de comunicación es fatal para los circuitos virtuales que la usan, pero puede compensarse fácilmente cuando se usan datagramas. Éstos también permiten que los enrutadores equilibren el tráfico a través de la subred, ya que las rutas pueden cambiarse a lo largo de una secuencia larga de transmisiones de paquetes.

5.2 ALGORITMOS DE ENRUTAMIENTO

La función principal de la capa de red es enrutar paquetes de la máquina de origen a la de destino. En la mayoría de las subredes, los paquetes requerirán varios saltos para completar el viaje. La única excepción importante son las redes de difusión, pero aun aquí es importante el enrutamiento si el origen y el destino no están en la misma red. Los algoritmos que eligen las rutas y las estructuras de datos que usan constituyen un aspecto principal del diseño de la capa de red.

El **algoritmo de enrutamiento** es aquella parte del software de la capa de red encargada de decidir la línea de salida por la que se transmitirá un paquete de entrada. Si la subred usa datagramas de manera interna, esta decisión debe tomarse cada vez que llega un paquete de datos, dado que la mejor ruta podría haber cambiado desde la última vez. Si la subred usa circuitos virtuales internamente, las decisiones de enrutamiento se toman sólo al establecerse un circuito virtual nuevo. En lo sucesivo, los paquetes de datos simplemente siguen la ruta previamente establecida. Este último caso a veces se llama **enrutamiento de sesión,** dado que una ruta permanece vigente durante toda la sesión de usuario (por ejemplo, durante una sesión desde una terminal, o durante una transferencia de archivos).

Algunas veces es útil distinguir entre el enrutamiento, que es el proceso consistente en tomar la decisión de cuáles rutas utilizar, y el reenvío, que consiste en la acción que se toma cuando llega un paquete. Se puede considerar que un enrutador realiza dos procesos internos. Uno de ellos maneja cada paquete conforme llega, buscando en las tablas de enrutamiento la línea de salida por la cual se enviará. Este proceso se conoce como **reenvío.** El otro proceso es responsable de llenar y actualizar las tablas de enrutamiento. Es ahí donde entra en acción el algoritmo de enrutamiento.

Sin importar si las rutas para cada paquete se eligen de manera independiente o sólo cuando se establecen nuevas conexiones, hay ciertas propiedades que todo algoritmo de enrutamiento debe poseer: exactitud, sencillez, robustez, estabilidad, equidad y optimización. La exactitud y la sencillez apenas requieren comentarios, pero la necesidad de robustez puede ser menos obvia a primera vista. Una vez que una red principal entra en operación, cabría esperar que funcionara continuamente durante años sin fallas a nivel de sistema. Durante ese periodo habrá fallas de hardware y de software de todo tipo. Los *hosts*, enrutadores y líneas fallarán en forma repetida y la topología cambiará muchas veces. El algoritmo de enrutamiento debe ser capaz de manejar los cambios de topología y tráfico sin requerir el aborto de todas las actividades en todos los *hosts* y el reinicio de la red con cada caída de un enrutador.

La estabilidad también es una meta importante del algoritmo de enrutamiento. Existen algoritmos de enrutamiento que nunca alcanzan el equilibrio, sin importar el tiempo que permanezcan operativos. Un algoritmo estable alcanza el equilibrio y lo conserva. La equidad y la optimización pueden parecer algo obvias (ciertamente nadie se opondrá a ellas), pero resulta que con frecuencia son metas contradictorias. En la figura 5-5 se muestra un ejemplo sencillo de este conflicto. Suponga que hay suficiente tráfico entre A y A', entre B y B' y entre C y C' para saturar los enlaces horizontales. A fin de aumentar al máximo el flujo total, el tráfico de X a X' debe suspenderse por completo. Por desgracia, X y X' podrían inconformarse. Evidentemente se requiere un punto medio entre la eficiencia global y la equidad hacia las conexiones individuales.

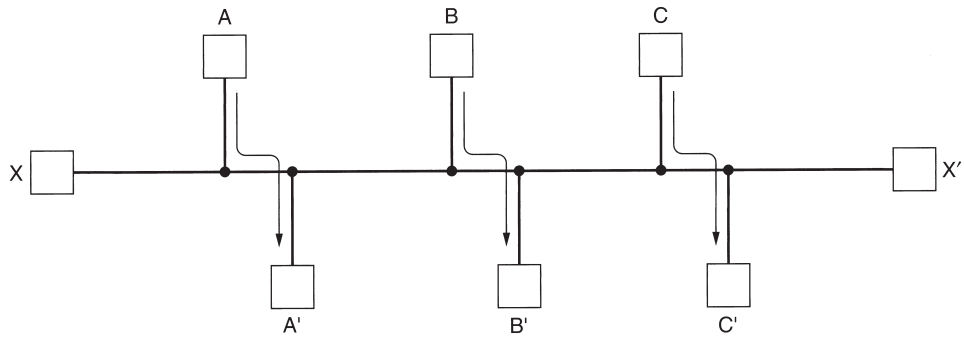


Figura 5-5. El conflicto entre equidad y optimización.

Antes de que podamos siquiera intentar encontrar el punto medio entre la equidad y la optimización, debemos decidir qué es lo que buscamos optimizar. Un candidato obvio es la minimización del retardo medio de los paquetes, pero también lo es el aumento al máximo de la velocidad real de transporte de la red. Además, estas dos metas también están en conflicto, ya que la operación de cualquier sistema de colas cerca de su capacidad máxima implica un retardo de encolamiento grande. Como término medio, muchas redes intentan minimizar el número de saltos que tiene que dar un paquete, puesto que la reducción de la cantidad de saltos reduce el retardo y también el consumo de ancho de banda, lo que a su vez mejora la velocidad real de transporte.

Los algoritmos de enrutamiento pueden agruparse en dos clases principales: no adaptativos y adaptativos. Los **algoritmos no adaptativos** no basan sus decisiones de enrutamiento en mediciones o estimaciones del tráfico y la topología actuales. En cambio, la decisión de qué ruta se usará para llegar de I a J (para todas las I y J) se toma por adelantado, fuera de línea, y se carga en los enrutadores al arrancar la red. Este procedimiento se conoce como **enrutamiento estático.**

En contraste, los **algoritmos adaptativos** cambian sus decisiones de enrutamiento para reflejar los cambios de topología y, por lo general también el tráfico. Los algoritmos adaptativos difieren en el lugar de donde obtienen su información (por ejemplo, localmente, de los enrutadores adyacentes o de todos los enrutadores), el momento de cambio de sus rutas (por ejemplo, cada ΔT segundos, cuando cambia la carga o cuando cambia la topología) y la métrica usada para la optimización (por ejemplo, distancia, número de saltos o tiempo estimado de tránsito). En las siguientes

secciones estudiaremos una variedad de algoritmos de enrutamiento, tanto estáticos como dinámicos.

5.2.1 Principio de optimización

Antes de entrar en algoritmos específicos, puede ser útil señalar que es posible hacer un postulado general sobre las rutas óptimas sin importar la topología o el tráfico de la red. Este postulado se conoce como **principio de optimización,** y establece que si el enrutador *J* está en ruta óptima del enrutador *I* al enrutador *K*, entonces la ruta óptima de *J* a *K* también está en la misma ruta. Para ver esto, llamemos r_1 a la parte de la ruta de *I* a *J*r_1 y r_2 al resto de la ruta. Si existiera una ruta mejor que r_2 entre *J* y *K*, podría conectarse con r_1 para mejorar la ruta entre *I* y *K*, contradiciendo nuestra aseveración de que r_1r_2 es óptima.

Como consecuencia directa del principio de optimización, podemos ver que el grupo de rutas óptimas de todos los orígenes a un destino dado forman un árbol con raíz en el destino. Tal árbol se conoce como **árbol sumidero** (o árbol divergente) y se ilustra en la figura 5-6, donde la métrica de distancia es el número de saltos. Observe que un árbol sumidero no necesariamente es único; pueden existir otros árboles con las mismas longitudes de rutas. La meta de todos los algoritmos de enrutamiento es descubrir y utilizar los árboles sumideros de todos los enrutadores.

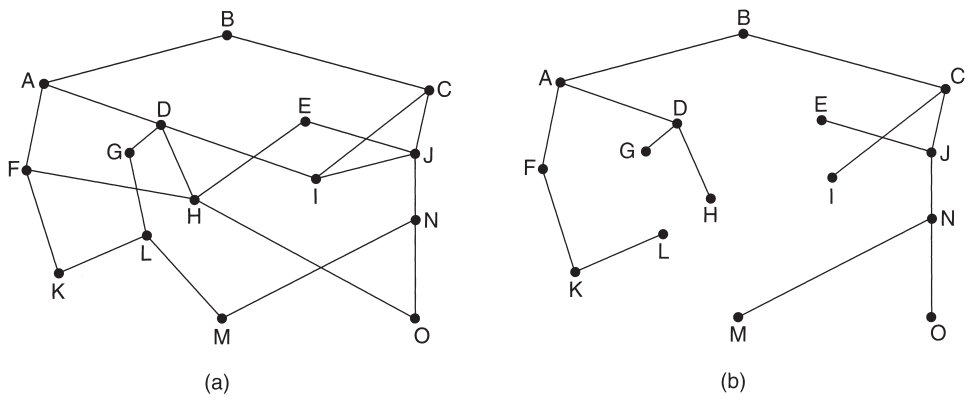


Figura 5-6. (a) Una subred. (b) Árbol sumidero para el enrutador *B*.

Puesto que un árbol sumidero ciertamente es un árbol, no contiene ciclos, por lo que cada paquete será entregado en un número de saltos finito y limitado. En la práctica, la vida no es tan fácil. Los enlaces y los enrutadores pueden caerse y reactivarse durante la operación, por lo que los diferentes enrutadores pueden tener ideas distintas sobre la topología actual. Además hemos evadido calladamente la cuestión de si cada enrutador tiene que adquirir de manera individual la información en la cual basa su cálculo del árbol sumidero, o si esta información se obtiene por otros

medios. Regresaremos a estos asuntos pronto. Con todo, el principio de optimización y el árbol sumidero proporcionan parámetros contra los que se pueden medir otros algoritmos de enrutamiento.

5.2.2 Enrutamiento por la ruta más corta

Comencemos nuestro estudio de los algoritmos de enrutamiento con una técnica de amplio uso en muchas formas, porque es sencilla y fácil de entender. La idea es armar un grafo de la subred, en el que cada nodo representa un enrutador y cada arco del grafo una línea de comunicación (con frecuencia llamada enlace). Para elegir una ruta entre un par dado de enrutadores, el algoritmo simplemente encuentra en el grafo la ruta más corta entre ellos.

El concepto de **ruta más corta** merece una explicación. Una manera de medir la longitud de una ruta es por la cantidad de saltos. Usando esta métrica, las rutas *ABC* y *ABE* de la figura 5-7 tienen la misma longitud. Otra métrica es la distancia geográfica en kilómetros, en cuyo caso *ABC* es claramente mucho mayor que *ABE* (suponiendo que la figura está dibujada a escala).

Sin embargo, también son posibles muchas otras métricas además de los saltos y la distancia física. Por ejemplo, cada arco podría etiquetarse con el retardo medio de encolamiento y transmisión de un paquete de prueba estándar, determinado por series de prueba cada hora. Con estas etiquetas en el grafo, la ruta más corta es la más rápida, en lugar de la ruta con menos arcos o kilómetros.

En el caso más general, las etiquetas de los arcos podrían calcularse como una función de la distancia, ancho de banda, tráfico medio, costo de comunicación, longitud media de las colas, retardo medio y otros factores. Cambiando la función de ponderación, el algoritmo calcularía la ruta "más corta" de acuerdo con cualquiera de varios criterios, o una combinación de ellos.

Se conocen varios algoritmos de cálculo de la ruta más corta entre dos nodos de un grafo. Éste se debe a Dijkstra (1959). Cada nodo se etiqueta (entre paréntesis) con su distancia al nodo de origen a través de la mejor ruta conocida. Inicialmente no se conocen rutas, por lo que todos los nodos tienen la etiqueta infinito. A medida que avanza el algoritmo y se encuentran rutas, las etiquetas pueden cambiar, reflejando mejores rutas. Una etiqueta puede ser tentativa o permanente. Inicialmente todas las etiquetas son tentativas. Una vez que se descubre que una etiqueta representa la ruta más corta posible del origen a ese nodo, se vuelve permanente y no cambia más.

Para ilustrar el funcionamiento del algoritmo de etiquetado, observe el grafo ponderado no dirigido de la figura 5-7(a), donde las ponderaciones representan, por ejemplo, distancias. Queremos encontrar la ruta más corta posible de *A* a *D*. Comenzamos por marcar como permanente el nodo *A*, indicado por un círculo rellenado. Después examinamos, por turno, cada uno de los nodos adyacentes a *A* (el nodo de trabajo), reetiquetando cada uno con la distancia desde *A*. Cada vez que reetiquetamos un nodo, también lo reetiquetamos con el nodo desde el que se hizo la prueba, para poder reconstruir más tarde la ruta final. Una vez que terminamos de examinar cada uno de los nodos adyacentes a *A*, examinamos todos los nodos etiquetados tentativamente en el grafo

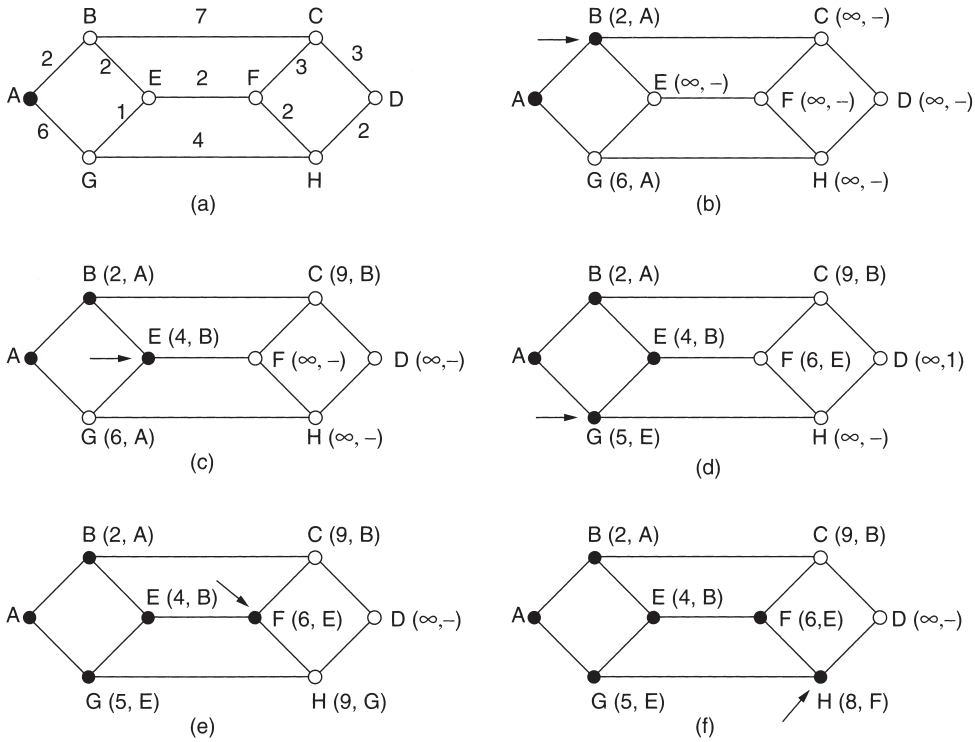


Figura 5-7. Los primeros cinco pasos del cálculo de la ruta más corta de *A* a *D*. Las flechas indican el nodo de trabajo.

completo y hacemos permanente el de la etiqueta más pequeña, como se muestra en la figura 5-7(b). Éste se convierte en el nuevo nodo de trabajo.

Ahora comenzamos por *B*, y examinamos todos los nodos adyacentes a él. Si la suma de la etiqueta de *B* y la distancia desde *B* al nodo en consideración es menor que la etiqueta de ese nodo, tenemos una ruta más corta, por lo que reetiquetamos ese nodo.

Tras inspeccionar todos los nodos adyacentes al nodo de trabajo y cambiar las etiquetas tentativas (de ser posible), se busca en el grafo completo el nodo etiquetado tentativamente con el menor valor. Este nodo se hace permanente y se convierte en el nodo de trabajo para la siguiente ronda. En la figura 5-7 se muestran los primeros cinco pasos del algoritmo.

Para ver por qué funciona el algoritmo, vea la figura 5-7(c). En ese punto acabamos de hacer permanente a *E*. Suponga que hubiera una ruta más corta que *ABE*, digamos *AXYZE*. Hay dos posibilidades: el nodo *Z* ya se hizo permanente, o no se ha hecho permanente. Si ya es permanente, entonces *E* ya se probó (en la ronda que siguió a aquella en la que se hizo permanente *Z*), por lo que la ruta *AXYZE* no ha escapado a nuestra atención y, por lo tanto, no puede ser una ruta más corta.

Ahora considere el caso en el que *Z* aún está etiquetado tentativamente. O bien la etiqueta de *Z* es mayor o igual que la de *E*, en cuyo caso *AXYZE* no puede ser una ruta más corta que *ABE*, o es menor que la de *E*, en cuyo caso *Z*, y no *E*, se volverá permanente primero, lo que permitirá que *E* se pruebe desde *Z*.

Este algoritmo se da en la figura 5-8. Las variables globales *n* y *dist* describen el grafo y son inicializadas antes de que se llame a *shortest_path*. La única diferencia entre el programa y el algoritmo antes descrito es que, en la figura 5-8, calculamos la ruta más corta posible comenzando por el nodo terminal, *t*, en lugar de en el nodo de origen, *s*. Dado que la ruta más corta posible desde *t* a *s* en un grafo no dirigido es igual a la ruta más corta de *s* a *t*, no importa el extremo por el que comencemos (a menos que haya varias rutas más cortas posibles, en cuyo caso la inversión de la búsqueda podría descubrir una distinta). La razón de una búsqueda hacia atrás es que cada nodo está etiquetado con su antecesor, en lugar de con su sucesor. Al copiar la ruta final en la variable de salida, *path*, la ruta de salida se invierte. Al invertir la búsqueda, ambos efectos se cancelan y la respuesta se produce en el orden correcto.

5.2.3 Inundación

Otro algoritmo estático es la **inundación,** en la que cada paquete de entrada se envía por cada una de las líneas de salida, excepto aquella por la que llegó. La inundación evidentemente genera grandes cantidades de paquetes duplicados; de hecho, una cantidad infinita a menos que se tomen algunas medidas para limitar el proceso. Una de estas medidas es integrar un contador de saltos en el encabezado de cada paquete, que disminuya con cada salto, y el paquete se descarte cuando el contador llegue a cero. Lo ideal es inicializar el contador de saltos a la longitud de la ruta entre el origen y el destino. Si el emisor desconoce el tamaño de la ruta, puede inicializar el contador al peor caso, es decir, el diámetro total de la subred.

Una técnica alterna para ponerle diques a la inundación es llevar un registro de los paquetes difundidos, para evitar enviarlos una segunda vez. Una manera de lograr este propósito es hacer que el enrutador de origen ponga un número de secuencia en cada paquete que recibe de sus *hosts*. Cada enrutador necesita una lista por cada enrutador de origen que indique los números de secuencia originados en ese enrutador que ya ha visto. Si un paquete de entrada está en la lista, no se difunde.

Para evitar que la lista crezca sin límites, cada lista debe incluir un contador, *k*, que indique que todos los números de secuencia hasta *k* ya han sido vistos. Cuando llega un paquete, es fácil comprobar si es un duplicado; de ser así, se descarta. Es más, no se necesita la lista completa por debajo de *k*, pues *k* la resume efectivamente.

Una variación de la inundación, un poco más práctica, es la **inundación selectiva.** En este algoritmo, los enrutadores no envían cada paquete de entrada por todas las líneas, sino sólo por aquellas que van aproximadamente en la dirección correcta. Por lo general, no tiene mucho caso enviar un paquete dirigido al oeste a través de una línea dirigida al este, a menos que la topología sea extremadamente peculiar y que el enrutador esté seguro de este hecho.

```
#define MAX_NODES 1024                /* número máximo de nodos */
#define INFINITY 1000000000           /* un número mayor que cualquier ruta
                                         máxima */
int n, dist[MAX_NODES][MAX_NODES];    /* dist[i][j] es la distancia entre
                                         i y j */

void shortest_path(int s, int t, int path[])
{ struct state {                      /* la ruta con la que se está
                                         trabajando */
     int predecessor;                 /* nodo previo */
     int length;                      /* longitud del origen a este nodo */
     enum {permanent, tentative} label;/* estado de la etiqueta */
 } state[MAX_NODES];

 int i, k, min;
 struct state *p;

 for (p = &state[0]; p < &state[n]; p++) {    /* estado de inicialización */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
 }
 state[t].length = 0; state[t].label = permanent;
 k = t;                               /* k es el nodo de trabajo inicial */
 do{                                  /* ¿hay una ruta mejor desde k? */
   for (i = 0; i < n; i++)            /* este grafo tiene n nodos */
          if (dist[k][i] != 0 && state[i].label == tentative) {
                 if (state[k].length + dist[k][i] < state[i].length) {
                       state[i].predecessor = k;
                       state[i].length = state[k].length + dist[k][i];
                 }
          }
   /* Encuentra el nodo etiquetado tentativamente con la etiqueta menor. */
   k = 0; min = INFINITY;
   for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
                min = state[i].length;
                k = i;
          }
   state[k].label = permanent;
 } while (k != s);

 /* Copia la ruta en el arreglo de salida. */
 i = 0; k = s;
 do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Figura 5-8. Algoritmo de Dijkstra para calcular la ruta más corta a través de un grafo.

La inundación no es práctica en la mayoría de las aplicaciones, pero tiene algunos usos. Por ejemplo, en aplicaciones militares, donde grandes cantidades de enrutadores pueden volar en pedazos en cualquier momento, es altamente deseable la excelente robustez de la inundación. En las aplicaciones distribuidas de bases de datos a veces es necesario actualizar concurrentemente todas las bases de datos, en cuyo caso la inundación puede ser útil. En las redes inalámbricas, algunas estaciones que se encuentren dentro del alcance de radio de una estación dada pueden recibir los mensajes que ésta trasmita, lo cual es, de hecho, inundación, y algunos algoritmos utilizan esta propiedad. Un cuarto posible uso de la inundación es como métrica contra la que pueden compararse otros algoritmos de enrutamiento. La inundación siempre escoge la ruta más corta posible, porque escoge en paralelo todas las rutas posibles. En consecuencia, ningún otro algoritmo puede producir un retardo más corto (si ignoramos la sobrecarga generada por el proceso de inundación mismo).

5.2.4 Enrutamiento por vector de distancia

Las redes modernas de computadoras por lo general utilizan algoritmos de enrutamiento dinámico en lugar de los estáticos antes descritos, pues los algoritmos estáticos no toman en cuenta la carga actual de la red. En particular, dos algoritmos dinámicos, el enrutamiento por vector de distancia y el enrutamiento por estado del enlace, son los más comunes. En esta sección veremos el primer algoritmo. En la siguiente estudiaremos el segundo.

Los algoritmos de **enrutamiento por vector de distancia** operan haciendo que cada enrutador mantenga una tabla (es decir, un vector) que da la mejor distancia conocida a cada destino y la línea que se puede usar para llegar ahí. Estas tablas se actualizan intercambiando información con los vecinos.

El algoritmo de enrutamiento por vector de distancia a veces recibe otros nombres, incluido el de algoritmo de enrutamiento **Bellman-Ford** distribuido y el de algoritmo **Ford-Fulkerson,** por los investigadores que los desarrollaron (Bellman, 1957, y Ford y Fulkerson, 1962). Éste fue el algoritmo original de enrutamiento de ARPANET y también se usó en Internet con el nombre RIP.

En el enrutamiento por vector de distancia, cada enrutador mantiene una tabla de enrutamiento indizada por, y conteniendo un registro de, cada enrutador de la subred. Esta entrada comprende dos partes: la línea preferida de salida hacia ese destino y una estimación del tiempo o distancia a ese destino. La métrica usada podría ser la cantidad de saltos, el retardo de tiempo en milisegundos, el número total de paquetes encolados a lo largo de la ruta, o algo parecido.

Se supone que el enrutador conoce la "distancia" a cada uno de sus vecinos. Si la métrica es de saltos, la distancia simplemente es un salto. Si la métrica es la longitud de la cola, el enrutador simplemente examina cada cola. Si la métrica es el retardo, el enrutador puede medirlo en forma directa con paquetes especiales de ECO que el receptor simplemente marca con la hora y lo regresa tan rápido como puede.

Por ejemplo, suponga que el retardo se usa como métrica y que el enrutador conoce el retardo a cada uno de sus vecinos. Una vez cada T mseg, cada enrutador envía a todos sus vecinos una

lista de sus retardos estimados a cada destino. También recibe una lista parecida de cada vecino. Imagine que una de estas tablas acaba de llegar del vecino X, siendo X_i la estimación de X respecto al tiempo que le toma llegar al enrutador i. Si el enrutador sabe que el retardo a X es de m mseg, también sabe que puede llegar al enrutador i a través de X en $X_i + m$ mseg. Efectuando este cálculo para cada vecino, un enrutador puede encontrar la estimación que parezca ser la mejor y usar esa estimación, así como la línea correspondiente, en su nueva tabla de enrutamiento. Observe que la vieja tabla de enrutamiento no se usa en este cálculo.

Este proceso de actualización se ilustra en la figura 5-9. En la parte (a) se muestra una subred. En las primeras cuatro columnas de la parte (b) aparecen los vectores de retardo recibidos de los vecinos del enrutador *J*. *A* indica tener un retardo de 12 mseg a *B*, un retardo de 25 mseg a *C*, un retardo de 40 mseg a *D*, etc. Suponga que *J* ha medido o estimado el retardo a sus vecinos *A*, *I*, *H* y *K* en 8, 10, 12 y 6 mseg, respectivamente.

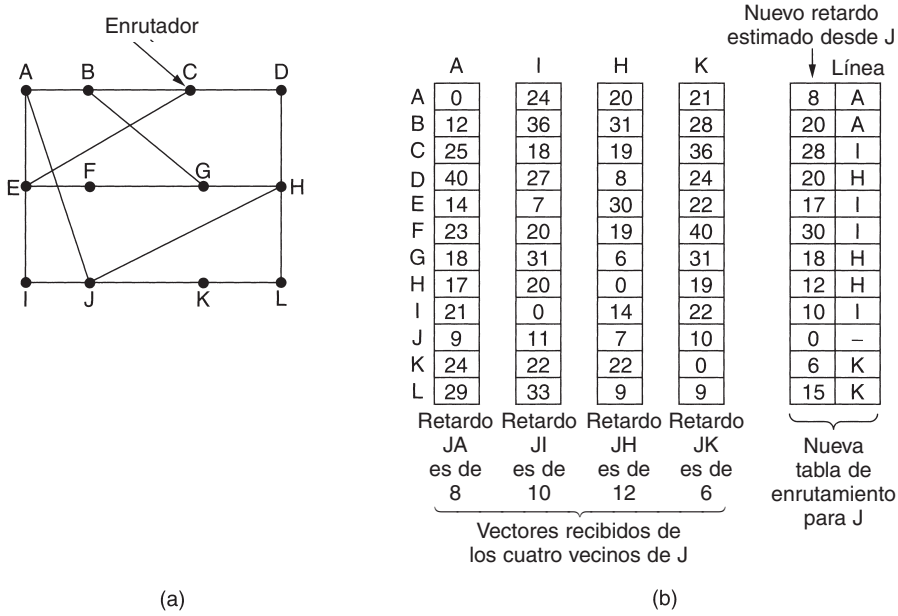


Figura 5-9. (a) Subred. (b) Entrada de A, I, H, K y la nueva tabla de enrutamiento de J.

Considere la manera en que *J* calcula su nueva ruta al enrutador *G*. Sabe que puede llegar a *A* en 8 mseg, y *A* indica ser capaz de llegar a *G* en 18 mseg, por lo que *J* sabe que puede contar con un retardo de 26 mseg a *G* si reenvía a través de *A* los paquetes destinados a *G*. Del mismo modo, *J* calcula el retardo a *G* a través de *I*, *H* y *K* en 41 (31 + 10), 18 (6 + 12) y 37 (31 + 6) mseg, respectivamente. El mejor de estos valores es el 18, por lo que escribe una entrada en su tabla de enrutamiento indicando que el retardo a *G* es de 18 mseg, y que la ruta que se utilizará es vía *H*. Se lleva a cabo el mismo cálculo para los demás destinos, y la nueva tabla de enrutamiento se muestra en la última columna de la figura.

El problema de la cuenta hasta infinito

El enrutamiento por vector de distancia funciona en teoría, pero tiene un problema serio en la práctica: aunque llega a la respuesta correcta, podría hacerlo lentamente. En particular, reacciona con rapidez a las buenas noticias, pero con lentitud ante las malas. Considere un enrutador cuya mejor ruta al destino *X* es larga. Si en el siguiente intercambio el vecino *A* informa repentinamente un retardo corto a *X*, el enrutador simplemente se conmuta a modo de usar la línea a *A* para enviar tráfico hasta *X*. En un intercambio de vectores, se procesan las buenas noticias.

Para ver la rapidez de propagación de las buenas noticias, considere la subred de cinco nodos (lineal) de la figura 5-10, en donde la métrica de retardo es el número de saltos. Suponga que *A* está desactivado inicialmente y que los otros enrutadores lo saben. En otras palabras, habrán registrado como infinito el retardo a *A*.

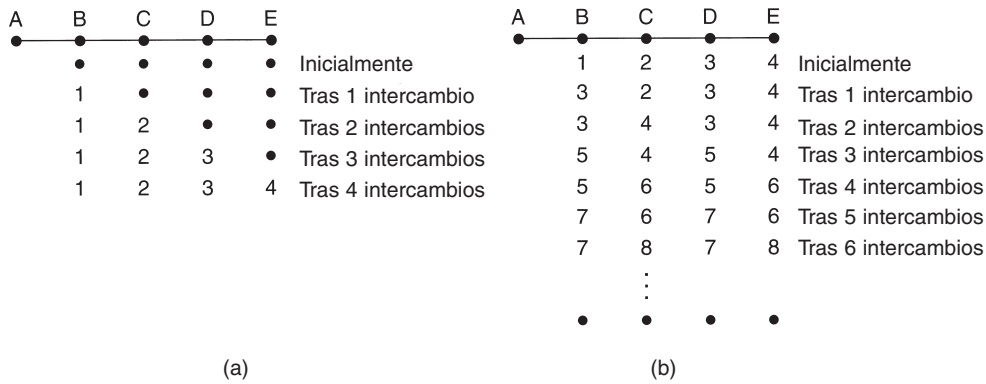


Figura 5-10. El problema de la cuenta hasta infinito.

Al activarse *A*, los demás enrutadores saben de él gracias a los intercambios de vectores. Por sencillez, supondremos que hay un gong gigantesco en algún lado, golpeando periódicamente para iniciar de manera simultánea un intercambio de vectores entre todos los enrutadores. En el momento del primer intercambio, *B* se entera de que su vecino de la izquierda tiene un retardo de 0 hacia *A*. *B* crea entonces una entrada en su tabla de enrutamiento, indicando que *A* está a un salto de distancia hacia la izquierda. Los demás enrutadores aún piensan que *A* está desactivado. En este punto, las entradas de la tabla de enrutamiento de *A* se muestran en la segunda fila de la figura 5-10(a). Durante el siguiente intercambio, *C* se entera de que *B* tiene una ruta a *A* de longitud 1, por lo que actualiza su tabla de enrutamiento para indicar una ruta de longitud 2, pero *D* y *E* no se enteran de las buenas nuevas sino hasta después. Como es evidente, las buenas noticias se difunden a razón de un salto por intercambio. En una subred cuya ruta mayor tiene una longitud de *N* saltos, en un lapso de *N* intercambios todo mundo sabrá sobre las líneas y enrutadores recientemente revividos.

Ahora consideremos la situación de la figura 5-10(b), en la que todas las líneas y enrutadores están activos inicialmente. Los enrutadores *B*, *C*, *D* y *E* tienen distancias a *A* de 1, 2, 3 y 4, respectivamente. De pronto, *A* se desactiva, o bien se corta la línea entre *A* y *B*, que de hecho es la misma cosa desde el punto de vista de *B*.

En el primer intercambio de paquetes, *B* no escucha nada de *A*. Afortunadamente, *C* dice: "No te preocupes. Tengo una ruta a *A* de longitud 2". *B* no sabe que la ruta de *C* pasa a través de *B* mismo. Hasta donde *B* sabe, *C* puede tener 10 líneas, todas con rutas independientes a *A* de longitud 2. Como resultado, *B* ahora piensa que puede llegar a *A* por medio de *C*, con una longitud de ruta de 3. *D* y *E* no actualizan sus entradas para *A* en el primer intercambio.

En el segundo intercambio, *C* nota que cada uno de sus vecinos indica tener una ruta a *A* de longitud 3. *C* escoge una de ellas al azar y hace que su nueva distancia a *A* sea de 4, como se muestra en la tercera fila de la figura 5-10(b). Los intercambios subsecuentes producen la historia mostrada en el resto de la figura 5-10(b).

A partir de esta figura debe quedar clara la razón por la que las malas noticias viajan con lentitud: ningún enrutador jamás tiene un valor mayor en más de una unidad que el mínimo de todos sus vecinos. Gradualmente, todos los enrutadores elevan cuentas hacia el infinito, pero el número de intercambios requerido depende del valor numérico usado para el infinito. Por esta razón, es prudente hacer que el infinito sea igual a la ruta más larga, más 1. Si la métrica es el retardo de tiempo, no hay un límite superior bien definido, por lo que se necesita un valor alto para evitar que una ruta con un retardo grande sea tratada como si estuviera desactivada. Este problema se conoce como el problema de la **cuenta hasta el infinito,** lo cual no es del todo sorprendente. Se han realizado algunos intentos por resolverlo (como el horizonte dividido con rutas inalcanzables [*poisoned reverse*] en el RFC 1058), pero ninguno funciona bien en general. La esencia del problema consiste en que cuando *X* indica a *Y* que tiene una ruta en algún lugar, *Y* no tiene forma de saber si él mismo está en la ruta.

5.2.5 Enrutamiento por estado del enlace

El enrutamiento por vector de distancia se usó en ARPANET hasta 1979, cuando fue reemplazado por el enrutamiento por estado del enlace. Dos problemas principales causaron su desaparición. Primero, debido a que la métrica de retardo era la longitud de la cola, no tomaba en cuenta el ancho de banda al escoger rutas. Inicialmente, todas las líneas eran de 56 kbps, por lo que el ancho de banda no era importante, pero una vez que se modernizaron algunas líneas a 230 kbps y otras a 1.544 Mbps, el no tomar en cuenta el ancho de banda se volvió un problema importante. Por supuesto, habría sido posible cambiar la métrica de retardo para considerar el ancho de banda, pero también existía un segundo problema: que el algoritmo con frecuencia tardaba demasiado en converger (el problema de la cuenta hasta el infinito). Por estas razones, el algoritmo fue reemplazado por uno completamente nuevo, llamado **enrutamiento por estado del enlace.** Hoy en día se usan bastante algunas variantes del enrutamiento por estado del enlace.

El concepto en que se basa el enrutamiento por estado del enlace es sencillo y puede enunciarse en cinco partes. Cada enrutador debe:

1. Descubrir a sus vecinos y conocer sus direcciones de red.
2. Medir el retardo o costo para cada uno de sus vecinos.
3. Construir un paquete que indique todo lo que acaba de aprender.
4. Enviar este paquete a todos los demás enrutadores.
5. Calcular la ruta más corta a todos los demás enrutadores.

De hecho, toda la topología y todos los retardos se miden experimentalmente y se distribuyen a cada enrutador. Entonces puede usarse el algoritmo de Dijkstra para encontrar la ruta más corta a los demás enrutadores. A continuación veremos con mayor detalle estos cinco pasos.

Conocimiento de los vecinos

Cuando un enrutador se pone en funcionamiento, su primera tarea es averiguar quiénes son sus vecinos; esto lo realiza enviando un paquete HELLO especial a cada línea punto a punto. Se espera que el enrutador del otro extremo regrese una respuesta indicando quién es. Estos nombres deben ser globalmente únicos puesto que, cuando un enrutador distante escucha después que tres enrutadores están conectados a *F*, es indispensable que pueda determinar si los tres se refieren al mismo *F*.

Cuando se conectan dos o más enrutadores mediante una LAN, la situación es ligeramente más complicada. En la figura 5-11(a) se ilustra una LAN a la que están conectados directamente tres enrutadores, *A*, *C* y *F*. Cada uno de estos enrutadores está conectado a uno o más enrutadores adicionales.

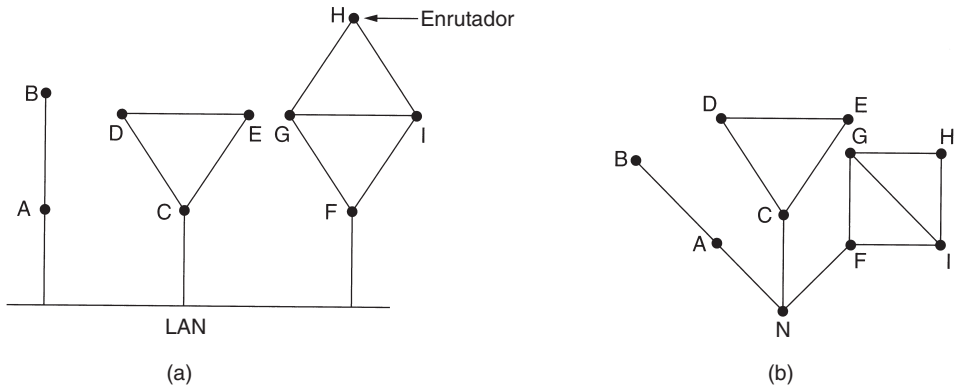


Figura 5-11. (a) Nueve enrutadores y una LAN. (b) Modelo de grafo de (a).

Una manera de modelar la LAN es considerarla como otro nodo, como se muestra en la figura 5-11(b). Aquí hemos introducido un nodo artificial nuevo, *N*, al que están conectados *A*, *C* y *F*. El hecho de que sea posible ir de *A* a *C* a través de la LAN se representa aquí mediante la ruta *ANC*.

Medición del costo de la línea

El algoritmo de enrutamiento por estado del enlace requiere que cada enrutador sepa, o cuando menos tenga una idea razonable, del retardo a cada uno de sus vecinos. La manera más directa de determinar este retardo es enviar un paquete ECHO especial a través de la línea y una vez que llegue al otro extremo, éste debe regresarlo inmediatamente. Si se mide el tiempo de ida y vuelta y se divide entre dos, el enrutador emisor puede tener una idea razonable del retardo. Para

obtener todavía mejores resultados, la prueba puede llevarse a cabo varias veces y usarse el promedio. Por supuesto que este método asume de manera implícita que los retardos son simétricos, lo cual no siempre es el caso.

Un aspecto interesante es si se debe tomar en cuenta la carga al medir el retardo. Para considerar la carga, el temporizador debe iniciarse cuando el paquete ECHO se ponga en la cola. Para ignorar la carga, el temporizador debe iniciarse cuando el paquete ECHO alcance el frente de la cola.

Pueden citarse argumentos a favor de ambos métodos. La inclusión de los retardos inducidos por el tráfico en las mediciones implica que cuando un enrutador puede escoger entre dos líneas con el mismo ancho de banda, una con carga alta continua y otra sin ella, considerará como ruta más corta la de la línea sin carga. Esta selección resultará en un mejor desempeño.

Desgraciadamente, también hay un argumento en contra de la inclusión de la carga en el cálculo del retardo. Considere la subred de la figura 5-12, dividida en dos partes, este y oeste, conectadas por dos líneas, *CF* y *EI*.

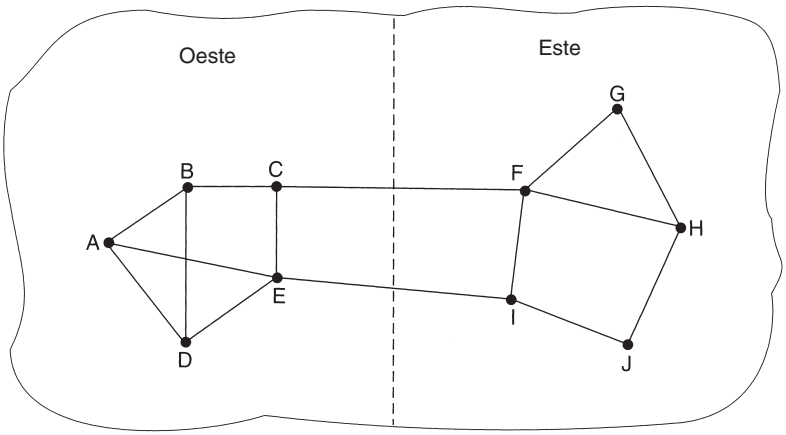
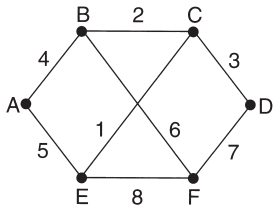


Figura 5-12. Subred en la que las partes este y oeste están conectadas por dos líneas.

Suponga que la mayor parte del tráfico entre el este y el oeste usa la línea *CF* y, como resultado, esta línea tiene tráfico alto con retardos grandes. La inclusión del retardo por encolamiento en el cálculo de la ruta más corta hará más atractiva a *EI*. Una vez instaladas las nuevas tablas de enrutamiento, la mayor parte del tráfico este-oeste pasará ahora por *EI*, sobrecargando esta línea. En consecuencia, en la siguiente actualización, *CF* aparecerá como la ruta más corta. Como resultado, las tablas de enrutamiento pueden oscilar sin control, lo que provocará un enrutamiento errático y muchos problemas potenciales. Si se ignora la carga y sólo se considera el ancho de banda, no ocurre este problema. De manera alterna, puede distribuirse la carga entre ambas líneas, pero esta solución no aprovecha al máximo la mejor ruta. No obstante, para evitar oscilaciones en la selección de la mejor ruta, podría ser adecuado dividir la carga entre múltiples líneas, con una fracción conocida de la carga viajando sobre cada una de ellas.

Construcción de los paquetes de estado del enlace

Una vez que se ha recabado la información necesaria para el intercambio, el siguiente paso es que cada enrutador construya un paquete que contenga todos los datos. El paquete comienza con la identidad del emisor, seguida de un número de secuencia, una edad (que se describirá después) y una lista de vecinos. Se da el retardo de vecino. En la figura 5-13(a) se da un ejemplo de subred, y los retardos se muestran como etiquetas en las líneas. En la figura 5-13(b) se muestran los paquetes de estado del enlace de los seis enrutadores.



(a)

Enlace Estado Paquetes

A		B		C		D		E		F	
Sec.		Sec.		Sec.		Sec.		Sec.		Sec.	
Edad		Edad		Edad		Edad		Edad		Edad	
B	4	A	4	B	2	C	3	A	5	B	6
E	5	C	2	D	3	F	7	C	1	D	7
		F	6	E	1			F	8	E	8

(b)

Figura 5-13. (a) Subred. (b) Paquetes de estado del enlace para esta subred.

Es fácil construir los paquetes de estado del enlace. La parte difícil es determinar cuándo construirlos. Una posibilidad es construirlos de manera periódica, es decir, a intervalos regulares. Otra posibilidad es construirlos cuando ocurra un evento significativo, como la caída o la reactivación de una línea o de un vecino, o el cambio apreciable de sus propiedades.

Distribución de los paquetes de estado del enlace

La parte más complicada del algoritmo es la distribución confiable de los paquetes de estado del enlace. A medida que se distribuyen e instalan los paquetes, los enrutadores que reciban los primeros cambiarán sus rutas. En consecuencia, los distintos enrutadores podrían estar usando versiones diferentes de la topología, lo que puede conducir a inconsistencias, ciclos, máquinas inalcanzables y otros problemas.

Primero describiremos el algoritmo básico de distribución y luego lo refinaremos. La idea fundamental es utilizar inundación para distribuir los paquetes de estado del enlace. A fin de mantener controlada la inundación, cada paquete contiene un número de secuencia que se incrementa con cada paquete nuevo enviado. Los enrutadores llevan el registro de todos los pares (enrutador de origen, secuencia) que ven. Cuando llega un paquete de estado del enlace, se verifica contra la lista de paquetes ya vistos. Si es nuevo, se reenvía a través de todas las líneas, excepto aquella por la que llegó. Si es un duplicado, se descarta. Si llega un paquete con número de secuencia menor que el mayor visto hasta el momento, se rechaza como obsoleto debido que el enrutador tiene datos más recientes.

Este algoritmo tiene algunos problemas, pero son manejables. Primero, si los números de secuencia vuelven a comenzar, reinará la confusión. La solución aquí es utilizar un número de secuencia de 32 bits. Con un paquete de estado del enlace por segundo, el tiempo para volver a empezar será de 137 años, por lo que puede ignorarse esta posibilidad.

Segundo, si llega a caerse un enrutador, perderá el registro de su número de secuencia. Si comienza nuevamente en 0, se rechazará como duplicado el siguiente paquete.

Tercero, si llega a corromperse un número de secuencia y se escribe 65,540 en lugar de 4 (un error de 1 bit), los paquetes 5 a 65,540 serán rechazados como obsoletos, dado que se piensa que el número de secuencia actual es 65,540.

La solución a todos estos problemas es incluir la edad de cada paquete después del número de secuencia y disminuirla una vez cada segundo. Cuando la edad llega a cero, se descarta la información de ese enrutador. Generalmente, un paquete nuevo entra, por ejemplo, cada 10 segundos, por lo que la información de los enrutadores sólo expira cuando el enrutador está caído (o cuando se pierden seis paquetes consecutivos, evento poco probable). Los enrutadores también decrementan el campo de *edad* durante el proceso inicial de inundación para asegurar que no pueda perderse ningún paquete y sobrevivir durante un periodo indefinido (se descarta el paquete cuya edad sea cero).

Algunos refinamientos de este algoritmo lo hacen más robusto. Una vez que un paquete de estado del enlace llega a un enrutador para ser inundado, no se encola para transmisión inmediata. En vez de ello, entra en un área de almacenamiento donde espera un tiempo breve. Si antes de transmitirlo entra otro paquete de estado del enlace proveniente del mismo origen, se comparan sus números de secuencia. Si son iguales, se descarta el duplicado. Si son diferentes, se desecha el más viejo. Como protección contra los errores en las líneas enrutador-enrutador, se confirma la recepción de todos los paquetes de estado del enlace. Cuando se desactiva una línea, se examina el área de almacenamiento en orden *round-robin* para seleccionar un paquete o confirmación de recepción a enviar.

En la figura 5-14 se describe la estructura de datos usada por el enrutador *B* para la subred de la figura 5-13(a). Cada fila aquí corresponde a un paquete de estado del enlace recién llegado, pero aún no procesado por completo. La tabla registra dónde se originó el paquete, su número de secuencia y edad, así como los datos. Además, hay banderas de transmisión y de confirmación de recepción para cada una de las tres líneas de *B* (a *A*, *C* y *F*, respectivamente). Las banderas de envío significan que el paquete debe enviarse a través de la línea indicada. Las banderas de confirmación de recepción significan que su confirmación debe suceder ahí.

En la figura 5-14, el paquete de estado del enlace de *A* llegó directamente, por lo que debe enviarse a *C* y *F*, y debe confirmarse la recepción a *A*, como lo muestran los bits de bandera. De la misma manera, el paquete de *F* tiene que reenviarse a *A* y a *C*, y debe enviarse a *F* la confirmación de su recepción.

Sin embargo, la situación del tercer paquete, de *E*, es diferente. Llegó dos veces, la primera a través de *EAB* y la segunda por medio de *EFB*. En consecuencia, este paquete tiene que enviarse sólo a *C*, pero debe confirmarse su recepción tanto a *A* como a *F*, como lo indican los bits.

Si llega un duplicado mientras el original aún está en el búfer, los bits tienen que cambiar. Por ejemplo, si llega una copia del estado de *C* desde *F* antes de que se reenvíe la cuarta entrada de la tabla, cambiarán los seis bits a 100011 para indicar que debe enviarse a *F* una confirmación de recepción del paquete, sin enviarle el paquete mismo.

			Banderas de envío			Banderas de ACK			
Origen	Sec.	Edad	A	C	F	A	C	F	Datos
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Figura 5-14. El búfer de paquetes para el enrutador *B* de la figura 5-13.

Cálculo de las nuevas rutas

Una vez que un enrutador ha acumulado un grupo completo de paquetes de estado del enlace, puede construir el grafo de la subred completa porque todos los enlaces están representados. De hecho, cada enlace se representa dos veces, una para cada dirección. Los dos valores pueden promediarse o usarse por separado.

Ahora puede ejecutar localmente el algoritmo de Dijkstra para construir la ruta más corta a todos los destinos posibles. Los resultados de este algoritmo pueden instalarse en las tablas de enrutamiento, y la operación normal puede reiniciarse.

Para una subred con n enrutadores, cada uno de los cuales tiene k vecinos, la memoria requerida para almacenar los datos de entrada es proporcional a kn. En las subredes grandes éste puede ser un problema. También puede serlo el tiempo de cómputo. Sin embargo, en muchas situaciones prácticas, el enrutamiento por estado del enlace funciona bien.

Sin embargo, problemas con el hardware o el software pueden causar estragos con este algoritmo (lo mismo que con otros). Por ejemplo, si un enrutador afirma tener una línea que no tiene, u olvida una línea que sí tiene, el grafo de la subred será incorrecto. Si un enrutador deja de reenviar paquetes, o los corrompe al hacerlo, surgirán problemas. Por último, si al enrutador se le acaba la memoria o se ejecuta mal el algoritmo de cálculo de enrutamiento, surgirán problemas. A medida que la subred crece en decenas o cientos de miles de nodos, la probabilidad de falla ocasional de un enrutador deja de ser insignificante. Lo importante es tratar de limitar el daño cuando ocurra lo inevitable. Perlman (1988) estudia detalladamente estos problemas y sus soluciones.

El enrutamiento por estado del enlace se usa ampliamente en las redes actuales, por lo que son pertinentes unas pocas palabras sobre algunos protocolos que lo usan. El protocolo OSPF, que se emplea cada vez con mayor frecuencia en Internet, utiliza un algoritmo de estado del enlace. Describiremos OSPF en la sección 5.6.4.

Otro protocolo de estado del enlace importante es el **IS-IS** (**sistema intermedio-sistema intermedio),** diseñado por DECnet y más tarde adoptado por la ISO para utilizarlo con su protocolo de capa de red sin conexiones, CLNP. Desde entonces se ha modificado para manejar también

otros protocolos, siendo el más notable el IP. El IS-IS se usa en varias redes dorsales de Internet (entre ellas la vieja red dorsal NSFNET) y en muchos sistemas celulares digitales, como CDPD. El Novell NetWare usa una variante menor del IS-IS (NLSP) para el enrutamiento de paquetes IPX.

Básicamente, el IS-IS distribuye una imagen de la topología de enrutadores, a partir de la cual se calculan las rutas más cortas. Cada enrutador anuncia, en su información de estado del enlace, las direcciones de capa de red que puede alcanzar de manera directa. Estas direcciones pueden ser IP, IPX, AppleTalk o cualquier otra dirección. El IS-IS incluso puede manejar varios protocolos de red al mismo tiempo.

Muchas de las innovaciones diseñadas para el IS-IS fueron adoptadas por el OSPF (el cual se diseñó varios años después que el IS-IS). Entre éstas se encuentran un método autorregulable para inundar actualizaciones de estado del enlace, el concepto de un enrutador designado en una LAN y el método de cálculo y soporte de la división de rutas y múltiples métricas. En consecuencia, hay muy poca diferencia entre el IS-IS y el OSPF. La diferencia más importante es que el IS-IS está codificado de tal manera que es fácil y natural llevar de manera simultánea información sobre varios protocolos de capa de red, característica que no está presente en el OSPF. Esta ventaja es especialmente valiosa en entornos multiprotocolo grandes.

5.2.6 Enrutamiento jerárquico

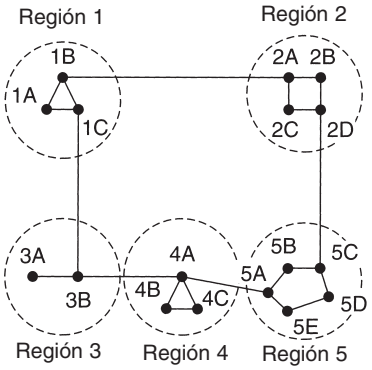
A medida que crece el tamaño de las redes, también lo hacen, de manera proporcional, las tablas de enrutamiento del enrutador. Las tablas que siempre crecen no sólo consumen memoria del enrutador, sino que también se necesita más tiempo de CPU para examinarlas y más ancho de banda para enviar informes de estado entre enrutadores. En cierto momento, la red puede crecer hasta el punto en que ya no es factible que cada enrutador tenga una entrada para cada uno de los demás enrutadores, por lo que el enrutamiento tendrá que hacerse de manera jerárquica, como ocurre en la red telefónica.

Cuando se utiliza el enrutamiento jerárquico, los enrutadores se dividen en lo que llamaremos **regiones,** donde cada enrutador conoce todos los detalles para enrutar paquetes a destinos dentro de su propia región, pero no sabe nada de la estructura interna de las otras regiones. Cuando se interconectan diferentes redes, es natural considerar cada una como región independiente a fin de liberar a los enrutadores de una red de la necesidad de conocer la estructura topológica de las demás.

En las redes enormes, una jerarquía de dos niveles puede ser insuficiente; tal vez sea necesario agrupar las regiones en clústeres, los clústeres en zonas, las zonas en grupos, etcétera, hasta que se nos agoten los nombres para clasificarlos. Como ejemplo de jerarquía multinivel, considere una posible forma de enrutar un paquete de Berkeley, California, a Malindi, Kenya. El enrutador de Berkeley conocería la topología detallada de California, pero podría enviar todo el tráfico exterior al enrutador de Los Ángeles. El enrutador de Los Ángeles podría enrutar el tráfico a otros enrutadores del país, pero enviaría el tráfico internacional a Nueva York. El enrutador de Nueva York tendría la programación para dirigir todo el tráfico al enrutador del país de destino encargado

del manejo de tráfico internacional, digamos en Nairobi. Por último, el paquete encontraría su camino por el árbol de Kenya hasta llegar a Malindi.

En la figura 5-15 se da un ejemplo cuantitativo de enrutamiento en una jerarquía de dos niveles con cinco regiones. La tabla de enrutamiento completa para el enrutador 1*A* tiene 17 entradas, como se muestra en la figura 5-15(b). Si el enrutamiento es jerárquico, como en la figura 5-15(c), hay entradas para todos los enrutadores locales, igual que antes, pero las demás regiones se han condensado en un solo enrutador, por lo que todo el tráfico para la región 2 va a través de la línea 1*B*-2*A*, pero el resto del tráfico remoto viaja por la línea 1*C*-3*B*. El enrutamiento jerárquico redujo la tabla de 17 entradas a 7. A medida que crece la razón entre la cantidad de regiones y el número de enrutadores por región, aumentan los ahorros de espacio de tabla.



(a)

Tabla completa para 1A

Dest.	Línea	Saltos
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Tabla jerárquica para 1A

Dest.	Línea	Saltos
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Figura 5-15. Enrutamiento jerárquico.

Desgraciadamente, estas ganancias de espacio no son gratuitas. Se paga un precio, que es una longitud de ruta mayor. Por ejemplo, la mejor ruta de 1*A* a 5*C* es a través de la región 2 pero con el enrutamiento jerárquico, todo el tráfico a la región 5 pasa por la región 3, porque eso es mejor para la mayoría de los destinos de la región 5.

Cuando una red se vuelve muy grande, surge una pregunta interesante: ¿cuántos niveles debe tener la jerarquía? Por ejemplo, considere una subred con 720 enrutadores. Si no hay jerarquía, cada enrutador necesita 720 entradas en la tabla de enrutamiento. Si dividimos la subred en 24 regiones de 30 enrutadores cada una, cada enrutador necesitará 30 entradas locales más 23 entradas remotas, lo que da un total de 53 entradas. Si elegimos una jerarquía de tres niveles, con ocho

clústeres, cada uno de los cuales contiene 9 regiones de 10 enrutadores, cada enrutador necesita 10 entradas para los enrutadores locales, 8 entradas para el enrutamiento a otras regiones dentro de su propio clúster y 7 entradas para clústeres distantes, lo que da un total de 25 entradas. Kamoun y Kleinrock (1979) descubrieron que el número óptimo de niveles para una subred de N enrutadores es de ln N, y se requieren un total de e ln N entradas por enrutador. También han demostrado que el aumento en la longitud media efectiva de ruta causado por el enrutamiento jerárquico es tan pequeño que por lo general es aceptable.

5.2.7 Enrutamiento por difusión

En algunas aplicaciones, los *hosts* necesitan enviar mensajes a varios otros *hosts* o a todos los demás. Por ejemplo, el servicio de distribución de informes ambientales, la actualización de los precios de la bolsa o los programas de radio en vivo podrían funcionar mejor difundiéndolos a todas las máquinas y dejando que las que estén interesadas lean los datos. El envío simultáneo de un paquete a todos los destinos se llama **difusión;** se han propuesto varios métodos para llevarla a cabo.

Un método de difusión que no requiere características especiales de la subred es que el origen simplemente envíe un paquete distinto a todos los destinos. El método no sólo desperdicia ancho de banda, sino que también requiere que el origen tenga una lista completa de todos los destinos. En la práctica, ésta puede ser la única posibilidad, pero es el método menos deseable.

La inundación es otro candidato obvio. Aunque ésta es poco adecuada para la comunicación punto a punto ordinaria, para difusión puede merecer consideración seria, especialmente si no es aplicable ninguno de los métodos descritos a continuación. El problema de la inundación como técnica de difusión es el mismo que tiene como algoritmo de enrutamiento punto a punto: genera demasiados paquetes y consume demasiado ancho de banda.

Un tercer algoritmo es el **enrutamiento multidestino.** Con este método, cada paquete contiene una lista de destinos o un mapa de bits que indica los destinos deseados. Cuando un paquete llega al enrutador, éste revisa todos los destinos para determinar el grupo de líneas de salida que necesitará. (Se necesita una línea de salida si es la mejor ruta a cuando menos uno de los destinos.) El enrutador genera una copia nueva del paquete para cada línea de salida que se utilizará, e incluye en cada paquete sólo aquellos destinos que utilizarán la línea. En efecto, el grupo de destinos se divide entre las líneas de salida. Después de una cantidad suficiente de saltos, cada paquete llevará sólo un destino, así que puede tratarse como un paquete normal. El enrutamiento multidestino es como los paquetes con direccionamiento individual, excepto que, cuando varios paquetes deben seguir la misma ruta, uno de ellos paga la tarifa completa y los demás viajan gratis.

Un cuarto algoritmo de difusión usa explícitamente el árbol sumidero para el enrutador que inicia la difusión, o cualquier otro árbol de expansión adecuado. El **árbol de expansión** es un subgrupo de la subred que incluye todos los enrutadores pero no contiene ciclos. Si cada enrutador sabe cuáles de sus líneas pertenecen al árbol de expansión, puede copiar un paquete de entrada difundido en todas las líneas del árbol de expansión, excepto en aquella por la que llegó. Este método utiliza de manera óptima el ancho de banda, generando la cantidad mínima de paquetes necesarios para llevar a cabo el trabajo. El único problema es que cada enrutador debe tener cono-

cimiento de algún árbol de expansión para que este método pueda funcionar. Algunas veces esta información está disponible (por ejemplo, con el enrutamiento por estado del enlace), pero a veces no (por ejemplo, con el enrutamiento por vector de distancia).

Nuestro último algoritmo de difusión es un intento de aproximar el comportamiento del anterior, aun cuando los enrutadores no saben nada en lo absoluto sobre árboles de expansión. La idea, llamada **reenvío por ruta invertida** (*reverse path forwarding*), es excepcionalmente sencilla una vez planteada. Cuando llega un paquete difundido a un enrutador, éste lo revisa para ver si llegó por la línea normalmente usada para enviar paquetes *al* origen de la difusión. De ser así, hay excelentes posibilidades de que el paquete difundido haya seguido la mejor ruta desde el enrutador y, por lo tanto, sea la primera copia en llegar al enrutador. Si éste es el caso, el enrutador reenvía copias del paquete a todas las líneas, excepto a aquella por la que llegó. Sin embargo, si el paquete difundido llegó por una línea diferente de la preferida, el paquete se descarta como probable duplicado.

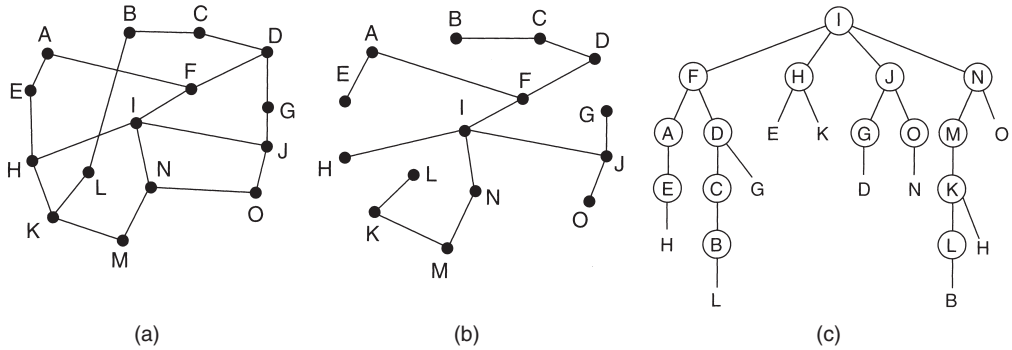


Figura 5-16. Reenvío por ruta invertida. (a) Subred. (b) Árbol sumidero. (c) Árbol construido mediante reenvío por ruta invertida.

En la figura 5-16 se muestra un ejemplo del reenvío por ruta invertida**.** En la parte (a) se muestra una subred, en la parte (b) se muestra un árbol sumidero para el enrutador *I* de esa subred, y en la parte (c) se muestra el funcionamiento del algoritmo de ruta invertida. En el primer salto, *I* envía paquetes a *F*, *H*, *J* y *N*, como lo indica la segunda fila del árbol. Cada uno de estos paquetes llega a *I* por la ruta preferida (suponiendo que la ruta preferida pasa a través del árbol sumidero), como lo indica el círculo alrededor de la letra. En el segundo salto, se generan ocho paquetes, dos por cada uno de los enrutadores que recibieron un paquete en el primer salto. Como resultado, los ocho llegan a enrutadores no visitados previamente, y cinco llegan a través de la línea preferida. De los seis paquetes generados en el tercer salto, sólo tres llegan por la ruta preferida (a *C, E* y *K*); los otros son duplicados. Después de cinco saltos y 24 paquetes, termina la difusión, en comparación con cuatro saltos y 14 paquetes si se hubiera seguido exactamente el árbol sumidero.

La ventaja principal del reenvío por ruta invertida es que es razonablemente eficiente y fácil de implementar. No requiere que los enrutadores conozcan los árboles de expansión ni tiene la sobrecarga de una lista de destinos o de un mapa de bits en cada paquete de difusión, como los

tiene el direccionamiento multidestino. Tampoco requiere mecanismos especiales para detener el proceso, como en el caso de la inundación (ya sea un contador de saltos en cada paquete y un conocimiento previo del diámetro de la subred, o una lista de paquetes ya vistos por origen).

5.2.8 Enrutamiento por multidifusión

Algunas aplicaciones requieren que procesos muy separados trabajen juntos en grupo; por ejemplo, un grupo de procesos que implementan un sistema de base de datos distribuido. En estos casos, con frecuencia es necesario que un proceso envíe un mensaje a todos los demás miembros del grupo. Si el grupo es pequeño, simplemente se puede transmitir a cada uno de los miembros un mensaje punto a punto. Si el grupo es grande, esta estrategia es costosa. A veces puede usarse la difusión, pero su uso para informar a 1000 máquinas de una red que abarca un millón de nodos es ineficiente porque la mayoría de los receptores no están interesados en el mensaje (o, peor aún, definitivamente están interesados pero no deben verlo). Por lo tanto, necesitamos una manera de enviar mensajes a grupos bien definidos de tamaño numéricamente grande, pero pequeños en comparación con la totalidad de la red.

El envío de un mensaje a uno de tales grupos se llama **multidifusión,** y su algoritmo de enrutamiento es el **enrutamiento por multidifusión.** En esta sección describiremos una manera de lograr el enrutamiento por multidifusión. Para información adicional, vea (Chu y cols., 2000; Costa y cols., 2001; Kasera y cols., 2000; Madruga y García-Luna-Aceves, 2001, y Zhang y Ryu, 2001).

Para la multidifusión se requiere administración de grupo. Se necesita alguna manera de crear y destruir grupos, y un mecanismo para que los procesos se unan a los grupos y salgan de ellos. La forma de realizar estas tareas no le concierne al algoritmo de enrutamiento. Lo que sí le concierne es que cuando un proceso se una a un grupo, informe a su *host* este hecho. Es importante que los enrutadores sepan cuáles de sus *hosts* pertenecen a qué grupos. Los *hosts* deben informar a sus enrutadores de los cambios en los miembros del grupo, o los enrutadores deben enviar de manera periódica la lista de sus *hosts*. De cualquier manera, los enrutadores aprenden qué *hosts* pertenecen a cuáles grupos. Los enrutadores les dicen a sus vecinos, de manera que la información se propaga a través de la subred.

Para realizar enrutamiento de multidifusión, cada enrutador calcula un árbol de expansión que cubre a todos los demás enrutadores de la subred. Por ejemplo, en la figura 5-17(a) tenemos una subred con dos grupos, 1 y 2. Algunos enrutadores están conectados a *hosts* que pertenecen a uno o ambos grupos, como se indica en la figura. En la figura 5-17(b) se muestra un árbol de expansión para el enrutador de la izquierda.

Cuando un proceso envía un paquete de multidifusión a un grupo, el primer enrutador examina su árbol de expansión y lo recorta, eliminando todas las líneas que conduzcan a *hosts* que no sean miembros del grupo. En el ejemplo de la figura 5-17(c) se muestra el árbol de expansión recortado del grupo 1. De la misma manera, en la figura 5-17(d) se presenta el árbol de expansión recortado del grupo 2. Los paquetes de multidifusión se reenvían sólo a través del árbol de expansión apropiado.

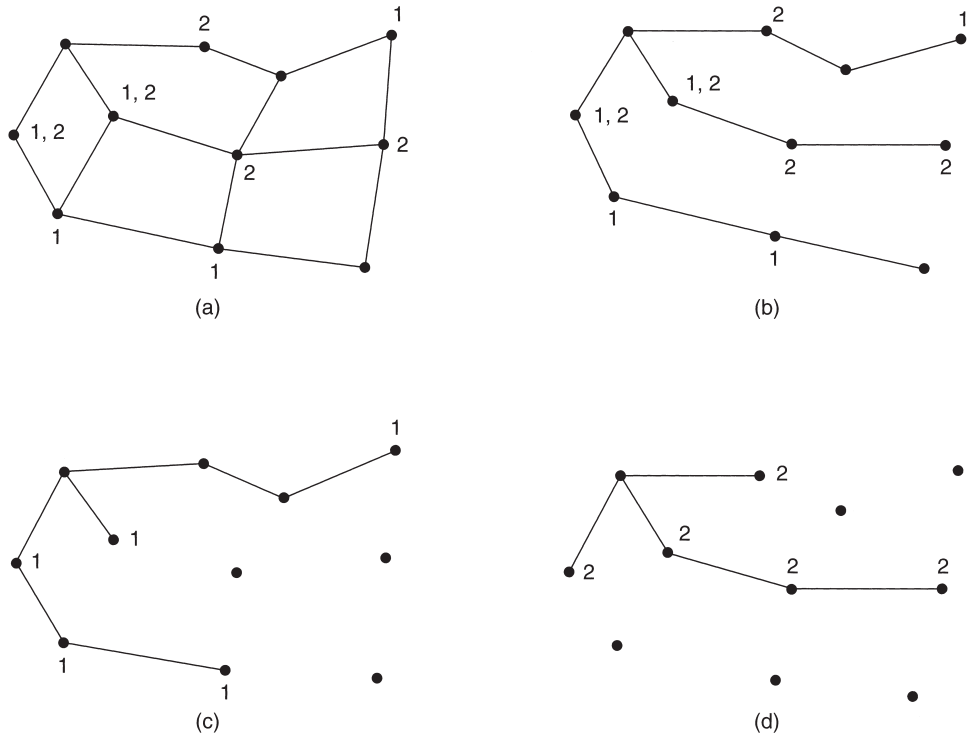


Figura 5-17. (a) Subred. (b) Árbol de expansión del enrutador del extremo izquierdo. (c) Árbol de multidifusión del grupo 1. (d) Árbol de multidifusión para el grupo 2.

Hay varias maneras de recortar el árbol de expansión. La más sencilla puede utilizarse si se maneja enrutamiento por estado del enlace, y cada enrutador está consciente de la topología completa de la subred, incluyendo qué *hosts* pertenecen a cuáles grupos. Después puede recortarse el árbol, comenzando por el final de cada ruta y trabajando hacia la raíz, eliminando todos los enrutadores que no pertenecen al grupo en cuestión.

Con el enrutamiento por vector de distancia se puede seguir una estrategia de recorte diferente. El algoritmo básico es el reenvío por ruta invertida. Sin embargo, cuando un enrutador sin *hosts* interesados en un grupo en particular y sin conexiones con otros enrutadores recibe un mensaje de multidifusión para ese grupo, responde con un mensaje de recorte (PRUNE), indicando al emisor que no envíe más multidifusiones para ese grupo. Cuando un enrutador que no tiene miembros del grupo entre sus propios *hosts* recibe uno de tales mensajes por todas las líneas, también puede responder con un mensaje de recorte. De esta forma, la subred se recorta recursivamente.

Una desventaja potencial de este algoritmo es que no escala bien en redes grandes. Suponga que una red tiene n grupos, cada uno con un promedio de m miembros. Por cada grupo se deben almacenar m árboles de expansión recortados, lo que da un total de mn árboles. Cuando hay muchos grupos grandes, se necesita bastante espacio para almacenar todos estos árboles.

Un diseño alterno utiliza **árboles de núcleo** (*core-based trees*) (Ballardie y cols., 1993). Aquí se calcula un solo árbol de expansión por grupo, con la raíz (el núcleo) cerca de la mitad del grupo. Para enviar un mensaje de multidifusión, un *host* lo envía al núcleo, que entonces hace la multidifusión a través del árbol de expansión. Aunque este árbol no será óptimo para todos los orígenes, la reducción de costos de almacenamiento de *m* árboles a un árbol por grupo representa un ahorro significativo.

5.2.9 Enrutamiento para *hosts* móviles

En la actualidad millones de personas tienen computadoras portátiles, y generalmente quieren leer su correo electrónico y acceder a sus sistemas de archivos normales desde cualquier lugar del mundo. Estos *hosts* móviles generan una nueva complicación: para enrutar un paquete a un *host* móvil, la red primero tiene que encontrarlo. El tema de la incorporación de *hosts* móviles en una red es muy reciente, pero en esta sección plantearemos algunos de los problemas relacionados y sugeriremos una posible solución.

En la figura 5-18 se muestra el modelo del mundo que usan generalmente los diseñadores de red. Aquí tenemos una WAN que consiste en enrutadores y *hosts*. Conectadas a la WAN hay varias LANs, MANs y celdas inalámbricas del tipo que estudiamos en el capítulo 2.

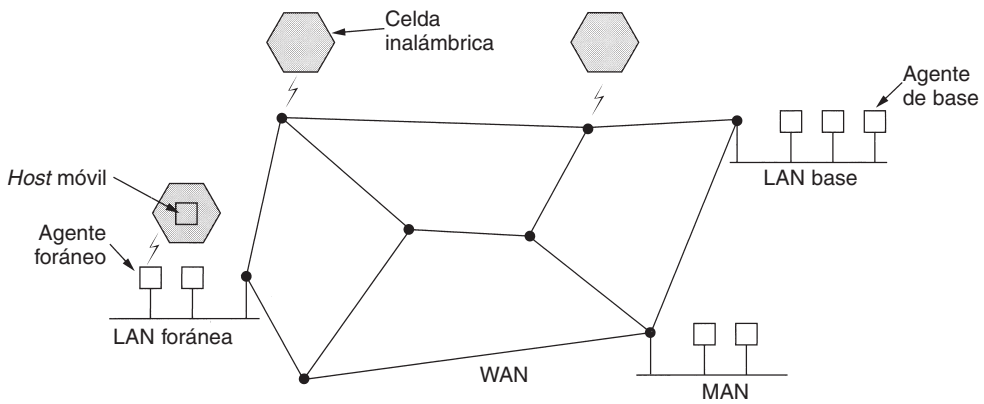


Figura 5-18. WAN a la que están conectadas LANs, MANs y celdas inalámbricas.

Se dice que los *hosts* que nunca se mueven son estacionarios; se conectan a la red mediante cables de cobre o fibra óptica. En contraste, podemos distinguir otros dos tipos de *hosts*. Los *hosts* migratorios básicamente son *hosts* estacionarios que se mueven de un lugar fijo a otro de tiempo en tiempo, pero que usan la red sólo cuando están conectados físicamente a ella. Los *hosts* ambulantes hacen su cómputo en movimiento, y necesitan mantener sus conexiones mientras se trasladan de un lado a otro. Usaremos el término ***hosts* móviles** para referirnos a cualquiera de las dos últimas categorías, es decir, a todos los *hosts* que están lejos de casa y que necesitan seguir conectados.

Se supone que todos los *hosts* tienen una **localidad base** que nunca cambia. Los *hosts* también tienen una dirección base permanente que puede servir para determinar su localidad base, de manera análoga a como el número telefónico 1-212-5551212 indica Estados Unidos (código de país 1) y Manhattan (212). La meta de enrutamiento en los sistemas con *hosts* móviles es posibilitar el envío de paquetes a *hosts* móviles usando su dirección base, y hacer que los paquetes lleguen eficientemente a ellos en cualquier lugar en el que puedan estar. Lo difícil, por supuesto, es encontrarlos.

En el modelo de la figura 5-18, el mundo se divide (geográficamente) en unidades pequeñas, a las que llamaremos áreas. Un área por lo general es una LAN o una celda inalámbrica. Cada área tiene uno o más **agentes foráneos,** los cuales son procesos que llevan el registro de todos los *hosts* móviles que visitan el área. Además, cada área tiene un **agente de base,** que lleva el registro de todos los *hosts* cuya base está en el área, pero que actualmente están visitando otra área.

Cuando un nuevo *host* entra en un área, ya sea al conectarse a ella (por ejemplo, conectándose a la LAN), o simplemente al entrar en la celda, su computadora debe registrarse con el agente foráneo de ese lugar. El procedimiento de registro funciona típicamente de esta manera:

1. Periódicamente, cada agente foráneo difunde un paquete que anuncia su existencia y dirección. Un *host* móvil recién llegado puede esperar uno de estos mensajes, pero si no llega ninguno con suficiente rapidez, el *host* móvil puede difundir un paquete que diga: "¿hay agentes foráneos por ahí?"
2. El *host* móvil se registra con el agente foráneo, dando su dirección base, su dirección actual de capa de enlace de datos y cierta información de seguridad.
3. El agente foráneo se pone en contacto con el agente de base del *host* móvil y le dice: "uno de tus *hosts* está por aquí". El mensaje del agente foráneo al agente de base contiene la dirección de red del agente foráneo, así como la información de seguridad, para convencer al agente de base de que el *host* móvil en realidad está ahí.
4. El agente de base examina la información de seguridad, que contiene una marca de tiempo, para comprobar que fue generada en los últimos segundos. Si está conforme, indica al agente foráneo que proceda.
5. Cuando el agente foráneo recibe la confirmación de recepción del agente de base, hace una entrada en sus tablas e informa al *host* móvil que ahora está registrado.

Idealmente, cuando un *host* sale de un área, este hecho también se debe anunciar para permitir que se borre el registro, pero muchos usuarios apagan abruptamente sus computadoras cuando terminan.

Cuando un paquete se envía a un *host* móvil, se enruta a la LAN base del *host*, ya que eso es lo indicado en la dirección, como se ilustra en el paso 1 de la figura 5-19. El emisor, que se encuentra en la ciudad noreste de Seattle, desea enviar un paquete a un *host* que se encuentra normalmente en Nueva York. Los paquetes enviados al *host* móvil en su LAN base de Nueva York son interceptados por el agente de base que se encuentra ahí. A continuación, dicho agente busca la

nueva ubicación (temporal) del *host* móvil y encuentra la dirección del agente foráneo que maneja al *host* móvil, en Los Ángeles.

El agente de base entonces hace dos cosas. Primero, encapsula el paquete en el campo de carga útil de un paquete exterior y envía este último al agente foráneo (paso 2 de la figura 5-19). Este mecanismo se llama entunelamiento y lo veremos con mayor detalle después. Tras obtener el paquete encapsulado, el agente foráneo extrae el paquete original del campo de carga útil y lo envía al *host* móvil como trama de enlace de datos.

Segundo, el agente de base indica al emisor que en lo futuro envíe paquetes al *host* móvil encapsulándolos en la carga útil de paquetes explícitamente dirigidos al agente foráneo, en lugar de simplemente enviarlos a la dirección base del *host* móvil (paso 3). Los paquetes subsiguientes ahora pueden enrutarse en forma directa al usuario por medio del agente foráneo (paso 4), omitiendo la localidad base por completo.

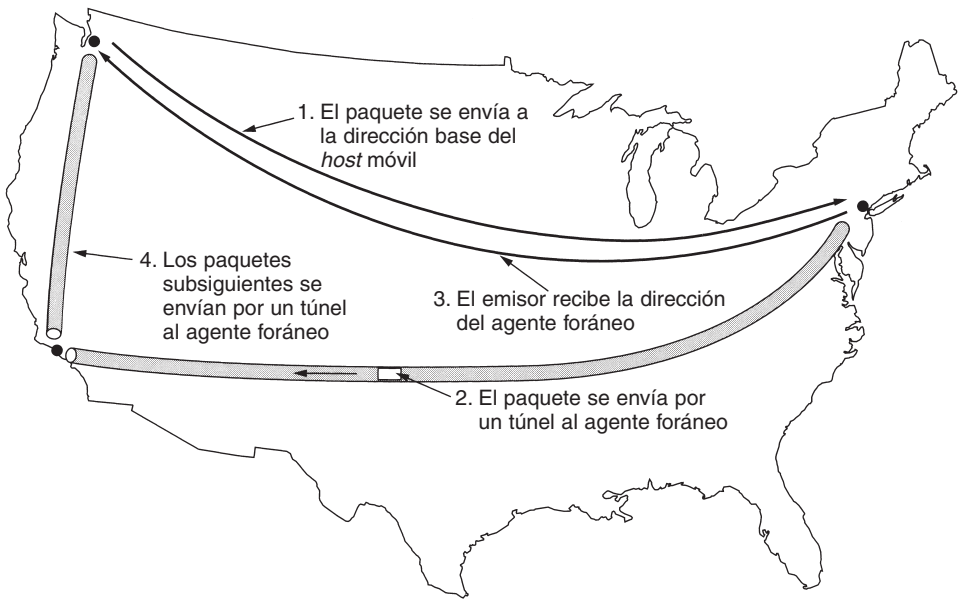


Figura 5-19. Enrutamiento de paquetes para *hosts* móviles.

Los diferentes esquemas propuestos difieren en varios sentidos. Primero está el asunto de qué parte del protocolo es llevada a cabo por los enrutadores y cuál por los *hosts* y, en este último caso, por cuál capa de los *hosts*. Segundo, en unos cuantos esquemas, los enrutadores a lo largo del camino registran las direcciones asignadas, para poder interceptarlas y redirigir el tráfico aún antes de que llegue a la dirección base. Tercero, en algunos esquemas, cada visitante recibe una dirección temporal única; en otros, la dirección temporal se refiere a un agente que maneja el tráfico de todos los visitantes.

Cuarto, los esquemas difieren en la manera en que logran realmente que los paquetes dirigidos a un destino lleguen a uno diferente. Una posibilidad es cambiar la dirección de destino y simplemente retransmitir el paquete modificado. Como alternativa, el paquete completo, con dirección base y todo, puede encapsularse en la carga útil de otro paquete enviado a la dirección temporal. Por último, los esquemas difieren en sus aspectos de seguridad. Por lo general, cuando un *host* o un enrutador recibe un mensaje del tipo "a partir de este momento, envíame por favor todo el correo de Genoveva", puede tener dudas acerca de con quién está hablando y de si se trata de una buena idea. En (Hac y Guo, 2000; Perkins, 1998a; Snoeren y Balakrishnan, 2000; Solomon, 1998, y Wang y Chen, 2001), se analizan y comparan varios protocolos para *hosts* móviles.

5.2.10 Enrutamiento en redes *ad hoc*

Ya vimos cómo realizar el enrutamiento cuando los *hosts* son móviles y los enrutadores son fijos. Un caso aún más extremo es uno en el que los enrutadores mismos son móviles. Entre las posibilidades se encuentran:

1. Vehículos militares en un campo de batalla sin infraestructura.
2. Una flota de barcos en el mar.
3. Trabajadores de emergencia en un área donde un temblor destruyó la infraestructura.
4. Una reunión de personas con computadoras portátiles en un área que no cuenta con 802.11.

En todos estos casos, y en otros, cada nodo consiste en un enrutador y un *host*, por lo general en la misma computadora. Las redes de nodos que están cerca entre sí se conocen como **redes *ad hoc*** o **MANETs (Redes *ad hoc* Móviles).** A continuación las examinaremos con brevedad. Para mayor información, vea (Perkins, 2001).

Lo que distingue a las redes *ad hoc* de las redes cableadas es que en las primeras se eliminaron todas las reglas comunes acerca de las topologías fijas, los vecinos fijos y conocidos, la relación fija entre direcciones IP y la ubicación, etcétera. Los enrutadores pueden ir y venir o aparecer en nuevos lugares en cualquier momento. En una red cableada, si un enrutador tiene una ruta válida a algún destino, esa ruta continúa siendo válida de manera indefinida (excepto si ocurre una falla en alguna parte del sistema que afecte a esa ruta). En una red *ad hoc*, la topología podría cambiar todo el tiempo, por lo que la necesidad o la validez de las rutas puede cambiar en cualquier momento, sin previo aviso. No es necesario decir que estas circunstancias hacen del enrutamiento en redes *ad hoc* algo diferente del enrutamiento en sus contrapartes fijas.

Se ha propuesto una variedad de algoritmos de enrutamiento para las redes *ad hoc*. Uno de los más interesantes es el algoritmo de enrutamiento **AODV (Vector de Distancia *ad hoc* bajo Demanda**) (Perkins y Royer, 1999). Es pariente lejano del algoritmo de vector de distancia Bellman-Ford pero está adaptado para funcionar en entornos móviles y toma en cuenta el ancho de banda

limitado y la duración corta de la batería en esos entornos. Otra característica inusual es que es un algoritmo bajo demanda, es decir, determina una ruta a algún destino sólo cuando alguien desea enviar un paquete a ese destino. A continuación veremos lo que eso significa.

Descubrimiento de ruta

En cualquier instante dado, una red *ad hoc* puede describirse mediante un grafo de los nodos (enrutadores + *hosts*). Dos nodos se conectan (es decir, tienen un arco entre ellos en el grafo) si se pueden comunicar de manera directa mediante sus radios. Debido a que uno de los dos podría tener un emisor más poderoso que el otro, es posible que *A* esté conectado a *B*, pero *B* no está conectado a *A*. Sin embargo, por simplicidad, asumiremos que todas las conexiones son simétricas. También debe hacerse notar que el simple hecho de que dos nodos estén dentro del alcance de radio entre sí no significa que estén conectados. Puede haber edificios, colinas u otros obstáculos que bloqueen su comunicación.

Para describir el algoritmo, considere la red *ad hoc* de la figura 5-20, en la que un proceso del nodo *A* desea enviar un paquete al nodo *I*. El algoritmo AODV mantiene una tabla en cada nodo, codificada por destino, que proporciona información acerca de ese destino, incluyendo a cuál vecino enviar los paquetes a fin de llegar al destino. Suponga que *A* busca en sus tablas y no encuentra una entrada para *I*. Ahora tiene que descubrir una ruta a *I*. Debido a esta propiedad de descubrir rutas sólo cuando es necesario este algoritmo se conoce como “bajo demanda”.

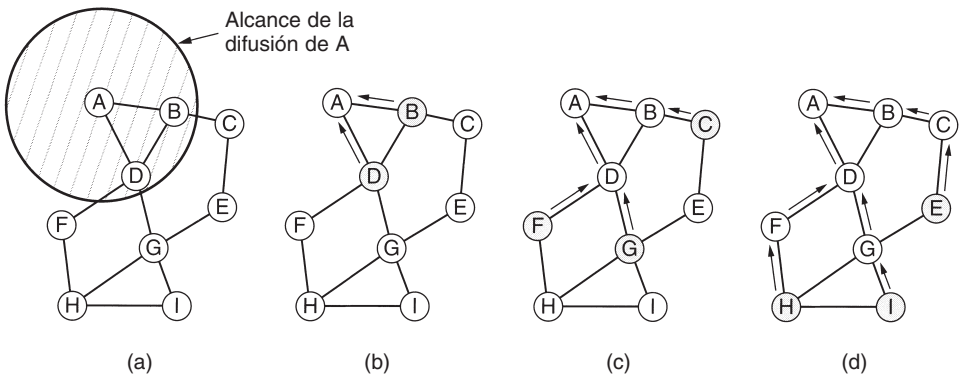


Figura 5-20. (a) Alcance de la difusión de *A*. (b) Después de que *B* y *D* reciben la difusión de *A*. (c) Después de que *C*, *F* y *G* reciben la difusión de *A*. (d) Después de que *E*, *H* e *I* reciben la difusión de *A*. Los nodos sombreados son nuevos receptores. Las flechas muestran las rutas invertidas posibles.

Para localizar a *I*, *A* construye un paquete especial de solicitud de ruta (ROUTE REQUEST) y lo difunde. El paquete llega a *B* y *D*, como se ilustra en la figura 5-20(a). De hecho, la razón por la que *B* y *D* se conectan a *A* en el grafo es que pueden recibir comunicación de *A*. Por ejemplo,

F no se muestra con un arco a *A* porque no puede recibir la señal de radio de *A*. Por lo tanto, *F* no se conecta a *A*.

En la figura 5-21 se muestra el formato del paquete de solicitud de ruta. Contiene las direcciones de origen y destino, por lo general sus direcciones IP, mediante las cuales se identifica quién está buscando a quién. También contiene un *ID de solicitud*, que es un contador local que se mantiene por separado en cada nodo y se incrementa cada vez que se difunde un paquete de solicitud de ruta. En conjunto, los campos *Dirección de origen* e *ID de solicitud* identifican de manera única el paquete de solicitud de ruta a fin de que los nodos descarten cualquier duplicado que pudieran recibir.

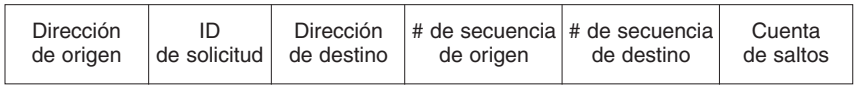


Figura 5-21. Formato de un paquete ROUTE REQUEST.

Además del contador *ID de solicitud*, cada nodo también mantiene un segundo contador de secuencia que se incrementa cada vez que se envía un paquete de solicitud de ruta (o una respuesta al paquete de solicitud de ruta de alguien más). Funciona de forma muy parecida a un reloj y se utiliza para indicar nuevas rutas a partir de rutas anteriores. El cuarto campo de la figura 5-21 es un contador de secuencia de *A*; el quinto campo es el valor más reciente del número de secuencia de *I* que *A* ha visto (0 si nunca lo ha visto). El uso de estos campos se aclarará pronto. El último campo, *Cuenta de saltos*, mantendrá un registro de cuántos saltos ha realizado el paquete. Se inicializa a 0.

Cuando un paquete de solicitud de ruta llega a un nodo (*B* y *D* en este caso), se procesa mediante los siguientes pasos.

1. El par (*Dirección de origen*, *ID de solicitud*) se busca en una tabla de historia local para ver si esta solicitud ya se había visto y procesado. Si es un duplicado, se descarta y el procesamiento se detiene. Si no es un duplicado, el par se introduce en la tabla de historia a fin de que se puedan rechazar futuros duplicados, y el procesamiento continúa.
2. El receptor busca el destino en su tabla de enrutamiento. Si se conoce una ruta reciente al destino, se regresa un paquete de respuesta de ruta (ROUTE REPLY) al origen que le indica cómo llegar al destino (básicamente: utilízame). Reciente significa que el *Número de secuencia de destino* almacenado en la tabla de enrutamiento es mayor que o igual al *Número de secuencia de destino* del paquete de solicitud de ruta. Si es menor, la ruta almacenada es más antigua que la que el origen tenía para el destino, por lo que se ejecuta el paso 3.
3. Puesto que el receptor no conoce una ruta reciente al destino, incrementa el campo *Cuenta de saltos* y vuelve a difundir el paquete de solicitud de ruta. También extrae los datos del paquete y los almacena como una entrada nueva en su tabla de rutas invertidas. Esta información se utilizará para construir la ruta invertida a fin de que la respuesta pueda

regresar posteriormente al origen. Las flechas que se muestran en la figura 5-20 se utilizan para construir la ruta invertida. También se inicia un temporizador para la nueva entrada de ruta invertida. Si expira, la entrada se borra.

Ni *B* ni *D* saben en dónde está *I*, por lo tanto, cada uno de estos nodos crea una entrada de ruta invertida que apunta a *A*, como lo muestran las flechas de la figura 5-20, y difunde el paquete con la *Cuenta de saltos* establecida a 1. La difusión de *B* llega a *C* y *D*. *C* crea una entrada para él en su tabla de rutas invertidas y la vuelve a difundir. En contraste, *D* la rechaza como un duplicado. De manera similar, *B* rechaza la difusión de *D*. Sin embargo, *F* y *G* aceptan la difusión de *D* y la almacenan como se muestra en la figura 5-20(c). Después de que *E*, *H* e *I* reciben la difusión, el paquete de solicitud de ruta finalmente alcanza un destino que sabe dónde está *I*, en este caso *I* mismo, como se ilustra en la figura 5-20(d). Observe que aunque mostramos las difusiones en tres pasos separados, las difusiones de nodos diferentes no se coordinan de ninguna forma.

En respuesta a la solicitud entrante, *I* construye un paquete de respuesta de ruta, como se muestra en la figura 5-22. La *Dirección de origen*, *Dirección de destino* y *Cuenta de saltos* se copian de la solicitud entrante, pero el *Número de secuencia de destino* se toma de su contador en memoria. El campo *Cuenta de saltos* se establece a 0. El campo *Tiempo de vida* controla cuánto tiempo es válida la ruta. Este paquete se difunde únicamente al nodo de donde proviene la solicitud de ruta, que en este caso es *G*. Después sigue la ruta invertida a *D* y, finalmente, a *A*. En cada nodo se incrementa *Cuenta de saltos* de modo que el nodo puede ver a qué distancia está del destino (*I*).

Dirección de origen	Dirección de destino	# de secuencia de origen	Cuenta de saltos	Tiempo de vida

Figura 5-22. Formato de un paquete de respuesta de ruta.

El paquete se inspecciona en cada nodo intermedio del camino de regreso. Se introduce en la tabla de enrutamiento local como una ruta a *I* si se cumple una o más de las siguientes tres condiciones:

1. No se conoce una ruta a *I*.
2. El número de secuencia para *I* en el paquete de respuesta de ruta es mayor que el valor en la tabla de enrutamiento.
3. Los números de secuencia son iguales pero la nueva ruta es más corta.

De esta forma, todos los nodos de la ruta invertida aprenden gratis la ruta a *I*, gracias al descubrimiento de ruta de *A*. Los nodos que obtuvieron el paquete original de solicitud de ruta pero que no estaban en la ruta invertida (*B*, *C*, *E*, *F* y *H* en este ejemplo) descartan la entrada de la tabla de ruta invertida cuando el temporizador asociado termina.

En una red grande, el algoritmo genera muchas difusiones, incluso para los destinos que están cerca. El número de difusiones se puede reducir como se muestra a continuación. El campo *Tiempo de vida* del paquete IP se inicializa al diámetro esperado de la red y se decrementa en cada salto. Si llega a 0, el paquete se descarta en lugar de difundirse.

El proceso de descubrimiento se modifica como se muestra a continuación. Para localizar un destino, el emisor difunde un paquete de solicitud de ruta con el campo *Tiempo de vida* establecido a 1. Si dentro de un tiempo razonable no se obtiene respuesta alguna, se envía otro paquete, pero esta vez con el campo *Tiempo de vida* establecido a 2. Los intentos subsiguientes utilizan 3, 4, 5, etcétera. De esta forma, la búsqueda primero se intenta de manera local y, después, en anillos cada vez más grandes.

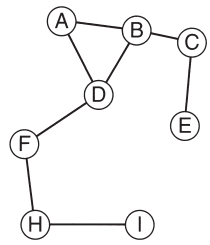
Mantenimiento de rutas

Debido a que es posible mover o apagar los nodos, la topología puede cambiar de manera espontánea. Por ejemplo, en la figura 5-20, si *G* se apaga, *A* no se dará cuenta de que la ruta a *I* (*ADGI*) que estaba utilizando ya no es válida. El algoritmo necesita ser capaz de manejar esto. Cada nodo difunde de manera periódica un mensaje de saludo (*Hello*). Se espera que cada uno de sus vecinos responda a dicho mensaje. Si no se recibe ninguna respuesta, el difusor sabe que el vecino se ha movido del alcance y ya no está conectado a él. De manera similar, si el difusor trata de enviar un paquete a un vecino que no responde, se da cuenta de que el vecino ya no está disponible.

Esta información se utiliza para eliminar rutas que ya no funcionan. Para cada destino posible, cada nodo, *N*, mantiene un registro de sus vecinos que le han proporcionado un paquete para ese destino durante los últimos ΔT segundos. Éstos se llaman **vecinos activos** de *N* para ese destino. *N* hace esto mediante una tabla de enrutamiento codificada por destino y al contener el nodo de salida a utilizar para llegar al destino, la cuenta de saltos al destino, el número de secuencia de destino más reciente y la lista de vecinos activos para ese destino. En la figura 5-23(a) se muestra una tabla de enrutamiento posible para el nodo *D* en nuestra topología de ejemplo.

Dest.	Siguiente salto	Distancia	Vecinos activos	Otros campos
A	A	1	F, G	
B	B	1	F, G	
C	B	2	F	
E	G	2		
F	F	1	A, B	
G	G	1	A, B	
H	F	2	A, B	
I	G	2	A, B	

(a)



(b)

Figura 5-23. (a) La tabla de enrutamiento de *D* antes de que *G* se apague. (b) El grafo después de que *G* se ha apagado.

Cuando cualquiera de los vecinos de *N* se vuelve inalcanzable, *N* verifica su tabla de enrutamiento para ver cuáles destinos tienen rutas en las que se incluya al vecino ahora perdido. Para cada una de estas rutas, se les informa a los vecinos activos que su ruta a través de *N* ahora es inválida y que se debe eliminar de sus tablas de enrutamiento. A continuación los vecinos activos indican este hecho a sus vecinos activos, y así sucesivamente y de manera recursiva, hasta que las rutas que dependen del nodo perdido se eliminan de todas las tablas de enrutamiento.

Como ejemplo del mantenimiento de ruta, considere nuestro ejemplo previo, pero ahora *G* se apaga de manera repentina. En la figura 5-23(b) se ilustra la topología modificada. Cuando *D* descubre que *G* ha desaparecido, busca en su tabla de enrutamiento y ve que *G* se utilizó en rutas a *E*, *G* e *I*. La unión de los vecinos activos para estos destinos es el conjunto {*A*, *B*}. En otras palabras, *A* y *B* dependen de *G* para algunas de sus rutas, y por esto se les tiene que informar que estas rutas ya no funcionan. *D* les indica este hecho enviándoles paquetes que causan que actualicen sus propias tablas de enrutamiento de manera acorde. *D* también elimina las entradas de *E*, *G* e *I* de su tabla de enrutamiento.

En nuestra descripción tal vez no sea obvio, pero una diferencia importante entre AODV y Bellman-Ford es que los nodos no envían difusiones periódicas que contengan su tabla de enrutamiento completa. Esta diferencia ahorra ancho de banda y duración de batería.

AODV también es capaz de realizar enrutamiento de difusión y multidifusión. Para mayores detalles, consulte (Perkins y Royer, 2001). El enrutamiento *ad hoc* es un área de investigación reciente. Se ha escrito bastante sobre este tema. Algunas de las obras son (Chen y cols., 2002; Hu y Johnson, 2001; Li y cols., 2001; Raju y Garcia-Luna-Aceves, 2001; Ramanathan y Redi, 2002; Royer y Toh, 1999; Spohn y Garcia-Luna-Aceves, 2001; Tseng y cols., 2001, y Zadeh y cols., 2002).

5.2.11 Búsqueda de nodos en redes de igual a igual

Las redes de igual a igual son un fenómeno relativamente nuevo, en las cuales una gran cantidad de personas, por lo general con conexiones cableadas permanentes a Internet, están en contacto para compartir recursos. La primera aplicación de uso amplio de la tecnología de igual a igual sirvió para cometer un delito masivo: 50 millones de usuarios de Napster intercambiaron canciones con derechos de autor sin el permiso de los poseedores de tales derechos hasta que las cortes cerraron Napster en medio de una gran controversia. Sin embargo, la tecnología de igual a igual tiene muchos usos interesantes y legales. También tiene algo similar a un problema de enrutamiento, aunque no como los que hemos estudiado hasta el momento. No obstante, vale la pena echarle un vistazo breve.

Lo que hace que los sistemas de igual a igual sean interesantes es que son totalmente distribuidos. Todos los nodos son simétricos y no hay control central o jerarquía. En un sistema típico de igual a igual, cada usuario tiene algo de información que podría ser de interés para otros usuarios. Esta información podría ser software (del dominio público), música, fotografías, etcétera, todos gratuitos. Si hay una gran cantidad de usuarios, éstos no se conocerán entre sí y no sabrán en dónde buscar lo que desean. Una solución es una base de datos central enorme, pero tal vez esto no sea tan factible por alguna razón (por ejemplo, nadie está dispuesto a albergarla ni a mantenerla). Por

lo tanto, el problema se reduce a qué debe hacer el usuario para encontrar un nodo que contiene lo que desea cuando no hay una base de datos centralizada o incluso un índice centralizado.

Supongamos que cada usuario tiene uno o más elementos de datos, como canciones, fotografías, programas, archivos, etcétera, que otros usuarios podrían querer leer. Cada elemento tiene una cadena ASCII que lo nombra. Un usuario potencial sólo conoce la cadena ASCII y desea averiguar si una o más personas tienen copias, y de ser así, cuáles son sus direcciones IP.

Como ejemplo, considere una base de datos genealógica distribuida. Cada genealogista tiene algunos registros en línea de sus predecesores y parientes, posiblemente con fotos, audio o incluso videoclips de las personas. Varias personas pueden tener el mismo bisabuelo, por lo que un predecesor podría tener registros en múltiples nodos. El nombre del registro es el nombre de la persona de alguna forma canónica. En algún punto, un genealogista descubre el testamento de su bisabuelo en un archivo, en el que su bisabuelo hereda su reloj de bolsillo de oro a su sobrino. El genealogista ahora sabe el nombre del sobrino y desea averiguar si otros genealogistas tienen un registro de dicho sobrino. ¿De qué manera sería posible, sin una base de datos central, saber quién, en caso de que lo hubiera, lo tiene?

Se han propuesto varios algoritmos para resolver este problema. El que examinaremos se llama Chord (Dabek y cols., 2001a, y Stoica y cols., 2001). A continuación se proporciona una explicación simplificada de cómo funciona. El sistema Chord consiste de n usuarios participantes, cada uno de los cuales podría contar con algunos registros almacenados y además está preparado para almacenar bits y fragmentos del índice para que otros usuarios los utilicen. Cada nodo de usuario tiene una dirección IP que puede generar un código de *hash* de m bits mediante una función de *hash*. Chord utiliza SHA-1 para *hash*. SHA-1 se utiliza en la criptografía; en el capítulo 8 analizaremos este tema. Por ahora, sólo es una función que toma como argumento una cadena de bytes de longitud variable y produce un número completamente aleatorio de 160 bits. Por lo tanto, podemos convertir cualquier dirección IP a un número de 160 bits llamado **identificador de nodo.**

Conceptualmente, todos los 2^{160} identificadores de nodos están organizados en orden ascendente en un gran círculo. Algunos de ellos corresponden a nodos participantes, pero la mayoría no. En la figura 5-24(a) mostramos el círculo de identificador de nodo de $m = 5$ (por el momento ignore el arco de en medio). En este ejemplo, los nodos con identificadores 1, 4, 7, 12, 15, 20 y 27 corresponden a nodos reales y están sombreados en la figura; el resto no existe.

A continuación definiremos la función *sucesor*(k) como el identificador de nodo del primer nodo real que sigue a k alrededor del círculo en el sentido de las manecillas del reloj. Por ejemplo, *sucesor*(6) = 7, *sucesor*(8) = 12 y *sucesor*(22) = 27.

A los nombres de los registros (nombres de canciones, nombres de los predecesores, etcétera) también se les aplica la función de *hash* (es decir, SHA-1) para generar un número de 160 bits, llamado **clave.** Por lo tanto, para convertir *nombre* (el nombre ASCII del registro) a su clave, utilizamos *clave* = *hash*(*nombre*). Este cálculo es simplemente una llamada de procedimiento local a *hash*. Si una persona que posee un registro genealógico para *nombre* desea ponerlo a disposición de todos, primero construye una tupla que consiste de (*nombre*, *mi-dirección-IP*) y después solicita a *sucesor*(*hash*(*nombre*)) que almacene la tupla. Si existen múltiples registros (en nodos diferentes) para este nombre, su tupla se almacenará en el mismo nodo. De esta forma, el índice se

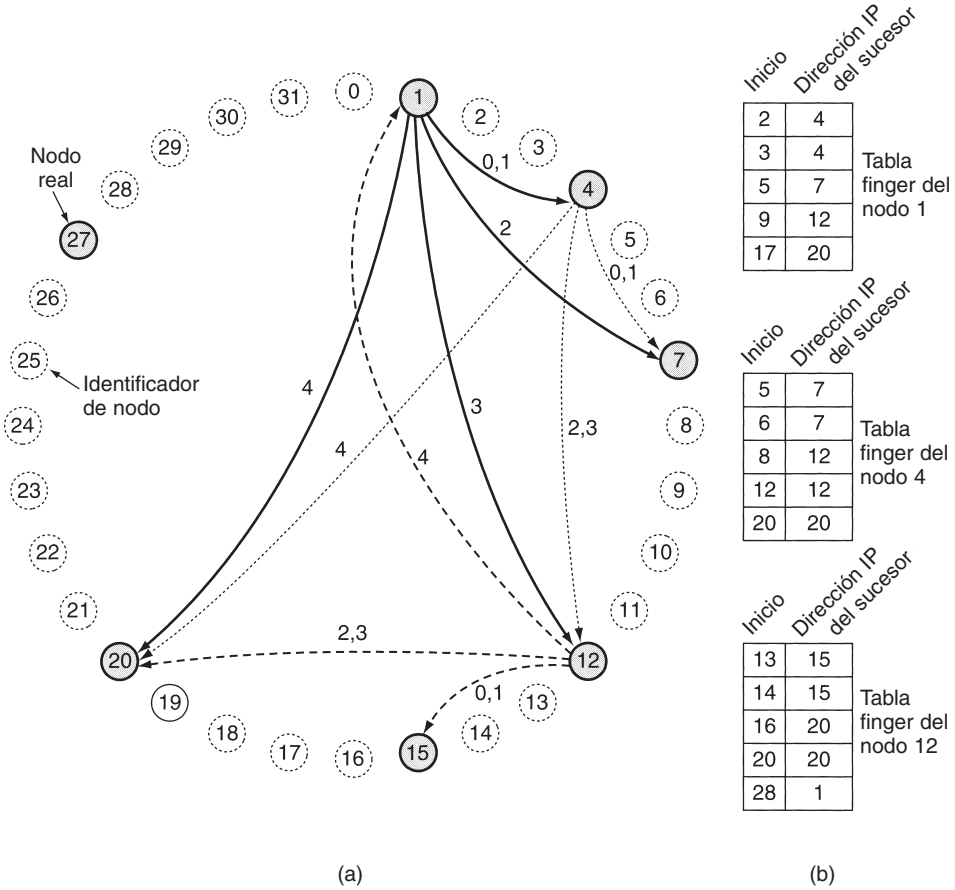


Figura 5-24. (a) Conjunto de 32 identificadores de nodos ordenados en círculo. Los sombreados corresponden a máquinas reales. Los arcos muestran los fingers de los nodos 1, 4 y 12. Las etiquetas de los arcos son los índices de las tablas. (b) Ejemplos de las tablas finger.

distribuye al azar sobre los nodos. Para tolerancia a fallas, podrían utilizarse *p* funciones de *hash* diferentes para almacenar cada tupla en *p* nodos, pero esto no lo tomaremos en cuenta aquí.

Si posteriormente algún usuario desea buscar *nombre*, le aplica la función de *hash* para obtener *clave* y después utiliza *sucesor*(*clave*) para encontrar la dirección IP del nodo que almacena sus tuplas de índice. El primer paso es fácil; el segundo no lo es. Para poder encontrar la dirección IP del nodo que corresponde a cierta clave, cada nodo debe mantener ciertas estructuras de datos administrativas. Una de ellas es la dirección IP de su nodo sucesor a lo largo del círculo identificador de nodo. Por ejemplo, en la figura 5-24, el sucesor del nodo 4 es 7 y el sucesor del nodo 7 es 12.

La búsqueda ahora puede ser como se muestra a continuación. El nodo solicitante envía un paquete a su sucesor, el cual contiene su dirección IP y la clave que está buscando. El paquete se propaga alrededor del anillo hasta que localiza al sucesor del identificador de nodo que se está buscando. Ese nodo verifica si tiene alguna información que corresponda con la clave y, de ser así, la regresa directamente al nodo solicitante, del cual tiene la dirección IP.

Como primera optimización, cada nodo puede contener las direcciones IP tanto de su sucesor como de su predecesor, por lo que las consultas pueden enviarse en dirección de las manecillas del reloj o en dirección contraria, dependiendo de cuál ruta se considere más corta. Por ejemplo, el nodo 7 de la figura 5-24 puede ir en dirección de las manecillas del reloj para encontrar el identificador de nodo 10, pero en dirección contraria para encontrar el identificador de nodo 3.

Incluso con dos opciones de dirección, la búsqueda lineal de todos los nodos es muy ineficiente en un sistema grande de igual a igual debido a que el número promedio de nodos requeridos por búsqueda es $n/2$. Para incrementar en forma considerable la velocidad de la búsqueda, cada nodo también mantiene lo que Chord llama una **tabla finger.** Ésta tiene m entradas, indexadas desde 0 hasta $m - 1$, y cada una apunta a un nodo real diferente. Cada una de estas entradas tiene dos campos: *inicio* y la dirección IP de *sucesor*(*inicio*), como se muestra para tres nodos de ejemplo de la figura 5-24(b). Los valores de los campos para la entrada i en el nodo k son:

$inicio = k + 2^i$ ($módulo 2^m$)
Dirección IP de *sucesor*(*inicio* [i])

Observe que cada nodo almacena las direcciones IP de una cantidad de nodos relativamente pequeña y que la mayoría de éstos están muy cercanos en términos de identificador de nodo.

Mediante la tabla *finger*, la búsqueda de *clave* en el nodo k se realiza como se muestra a continuación. Si *clave* está entre k y *sucesor*(k), el nodo que contiene información acerca de *clave* es *sucesor* (k) y la búsqueda termina. De lo contrario, en la tabla *finger* se busca la entrada cuyo campo *inicio* sea el predecesor más cercano de *clave*. A continuación se envía una solicitud directamente a la dirección IP de esa entrada de la tabla *finger* para pedirle que continúe la búsqueda. Debido a que dicha dirección está más cerca de la *clave,* aunque todavía está por debajo, es muy probable que pueda regresar la respuesta con tan sólo algunas consultas adicionales. De hecho, debido a que cada búsqueda divide en dos la distancia restante al destino, puede mostrarse que el número promedio de búsquedas es $\log_2 n$.

Como primer ejemplo, considere la búsqueda de *clave* = 3 en el nodo 1. Debido a que el nodo 1 sabe que 3 está entre él y su sucesor, 4, el nodo deseado es 4 y la búsqueda termina, regresando la dirección IP del nodo 4.

Como segundo ejemplo, considere la búsqueda de *clave* = 14 en el nodo 1. Debido a que 14 no está entre 1 y 4, se consulta la tabla *finger*. El predecesor más cercano a 14 es 9, por lo que la solicitud se reenvía a la dirección IP de la entrada 9, es decir, la del nodo 12. Este nodo ve que 14 está entre él y su sucesor (15), por lo que regresa la dirección IP del nodo 15.

Como tercer ejemplo, considere la búsqueda de *clave* = 16 en el nodo 1. Nuevamente, se envía una solicitud al nodo 12, pero esta vez dicho nodo no sabe la respuesta. Busca el nodo más cercano que preceda a 16 y encuentra 14, lo que resulta en la dirección IP del nodo 15. A continuación

se envía una consulta ahí. El nodo 15 observa que 16 está entre él y su sucesor (20), por lo que regresa la dirección IP de 20 al invocador, que en este caso es el nodo 1.

Puesto que los nodos se unen y separan todo el tiempo, Chord necesita una forma de manejar estas operaciones. Suponemos que cuando el sistema comenzó a operar era tan pequeño que los nodos apenas podían intercambiar información directamente para construir el primer círculo y las primeras tablas *finger*. Después de eso se necesita un procedimiento automatizado, como el que se muestra a continuación. Cuando un nuevo nodo, *r*, desea unirse, debe contactar a algún nodo existente y pedirle que busque la dirección IP de *sucesor*(*r*). A continuación, el nuevo nodo solicita a *sucesor*(*r*) su predecesor. Después pide a ambos que inserten *r* entre ellos en el círculo. Por ejemplo, si el nodo 24 de la figura 5-24 desea unirse, pide a cualquier nodo que busque *sucesor*(24), que es 27. A continuación pide a 27 su predecesor (20). Después de que les informa a estos dos de su existencia, 20 utiliza 24 como su sucesor y 27 utiliza 24 como su predecesor. Además, el nodo 27 entrega esas claves en el rango 21–24, que ahora pertenece a 24. En este punto, 24 está completamente insertado.

Sin embargo, ahora muchas tablas *finger* son erróneas. Para corregirlas, cada nodo ejecuta un proceso en segundo plano que recalcula de manera periódica cada *finger* invocando a *sucesor*. Cuando una de estas consultas coincide con un nuevo nodo, se actualiza la entrada correspondiente del *finger*.

Cuando un nodo se separa con éxito, entrega sus claves a su sucesor e informa a su predecesor su partida a fin de que dicho predecesor pueda enlazarse con el sucesor del nodo que se va. Cuando falla un nodo, surge un problema pues su predecesor ya no tiene un sucesor válido. Para solucionarlo, cada nodo lleva un registro no sólo de su sucesor directo sino también de sus *s* sucesores directos, a fin de saltar hasta $s - 1$ nodos erróneos consecutivos y volver a conectar el círculo.

Chord se ha utilizado para construir un sistema de archivos distribuido (Dabek y cols., 2001b) y otras aplicaciones, y las investigaciones continúan. En (Rowstron y Druschel, 2001a, y Rowstron y Druschel, 2001b), se describe un sistema de igual a igual diferente, Pastry, así como sus aplicaciones. En (Clarke y cols., 2002) se analiza un tercer sistema de igual a igual, Freenet. En (Ratnasamy y cols., 2001) se describe un cuarto sistema de este tipo.

5.3 ALGORITMOS DE CONTROL DE CONGESTIÓN

Cuando hay demasiados paquetes presentes en la subred (o en una parte de ella), hay una degradación del desempeño. Esta situación se llama **congestión.** En la figura 5-25 se muestra este síntoma. Cuando la cantidad de paquetes descargados en la subred por los *hosts* está dentro de su capacidad de conducción, todos se entregan (excepto unos pocos afligidos por errores de transmisión) y la cantidad entregada es proporcional al número enviado. Sin embargo, a medida que aumenta el tráfico, los enrutadores ya no pueden manejarlo y comienzan a perder paquetes. Esto tiende a empeorar las cosas. Con mucho tráfico, el desempeño se desploma por completo y casi no hay entrega de paquetes.

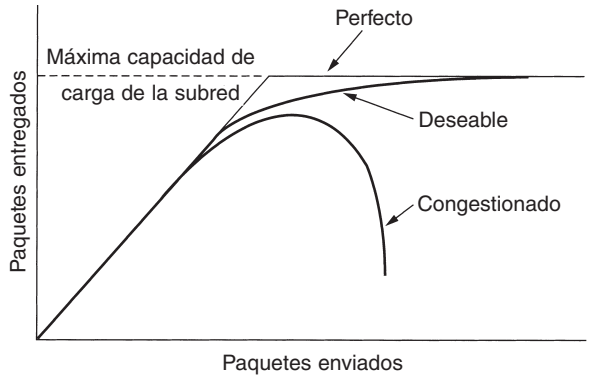


Figura 5-25. Cuando se genera demasiado tráfico, ocurre congestión y se degrada marcadamente el desempeño.

La congestión puede ocurrir por varias razones. Si de manera repentina comienzan a llegar cadenas de paquetes por tres o cuatro líneas de entrada y todas necesitan la misma línea de salida, se generará una cola. Si no hay suficiente memoria para almacenar a todos los paquetes, algunos de ellos se perderán. La adición de memoria puede ayudar hasta cierto punto, pero Nagle (1987) descubrió que si los enrutadores tienen una cantidad infinita de memoria, la congestión empeora en lugar de mejorar, ya que para cuando los paquetes llegan al principio de la cola, su temporizador ha terminado (repetidamente) y se han enviado duplicados. Todos estos paquetes serán debidamente reenviados al siguiente enrutador, aumentando la carga en todo el camino hasta el destino.

Los procesadores lentos también pueden causar congestión. Si las CPUs de los enrutadores son lentas para llevar a cabo las tareas de administración requeridas (búferes de encolamiento, actualización de tablas, etcétera), las colas pueden alargarse, aun cuando haya un exceso de capacidad de línea. De la misma manera, las líneas de poco ancho de banda también pueden causar congestión. La actualización de las líneas sin cambiar los procesadores, o viceversa, por lo general ayuda un poco, pero con frecuencia simplemente desplaza el cuello de botella. Además, actualizar sólo parte de un sistema simplemente mueve el cuello de botella a otra parte. El problema real con frecuencia es un desajuste entre partes del sistema. Este problema persistirá hasta que todos los componentes estén en equilibrio.

Vale la pena indicar de manera explícita la diferencia entre el control de la congestión y el control de flujo, pues la relación es sutil. El control de congestión se ocupa de asegurar que la subred sea capaz de transportar el tráfico ofrecido. Es un asunto global, en el que interviene el comportamiento de todos los *hosts*, todos los enrutadores, el proceso de almacenamiento y reenvío dentro de los enrutadores y todos los demás factores que tienden a disminuir la capacidad de transporte de la subred.

En contraste, el control de flujo se relaciona con el tráfico punto a punto entre un emisor dado y un receptor dado. Su tarea es asegurar que un emisor rápido no pueda transmitir datos de manera continua a una velocidad mayor que la que puede absorber el receptor. El control de flujo casi

siempre implica una retroalimentación directa del receptor al emisor, para indicar al emisor cómo van las cosas en el otro lado.

Para captar la diferencia entre estos dos conceptos, considere una red de fibra óptica con una capacidad de 1000 gigabits/seg en la que una supercomputadora está tratando de transferir un archivo a una computadora personal que opera a 1 Gbps. Aunque no hay congestión (la red misma no está en problemas), se requiere control de flujo para obligar a la supercomputadora a detenerse con frecuencia para darle a la computadora personal un momento de respiro.

En el otro extremo, considere una red de almacenamiento y reenvío con líneas de 1 Mbps y 1000 computadoras grandes, la mitad de las cuales trata de transferir archivos a 100 kbps a la otra mitad. Aquí el problema no es que los emisores rápidos saturen a los receptores lentos, sino simplemente que el tráfico ofrecido total excede lo que la red puede manejar.

La razón por la que se confunden con frecuencia el control de congestión y el control de flujo es que algunos algoritmos de control de congestión operan enviando mensajes de regreso a varios orígenes, indicándoles que reduzcan su velocidad cuando la red se mete en problemas. Por lo tanto, un *host* puede recibir un mensaje de "reducción de velocidad" porque el receptor no puede manejar la carga o porque la red no la puede manejar. Más adelante regresaremos a este punto.

Comenzaremos nuestro estudio del control de congestión examinando un modelo general para manejarlo. Luego veremos métodos generales para prevenirlo. Después de eso, veremos varios algoritmos dinámicos para manejarlo una vez que se ha establecido.

5.3.1 Principios generales del control de congestión

Muchos problemas de los sistemas complejos, como las redes de computadoras, pueden analizarse desde el punto de vista de una teoría de control. Este método conduce a dividir en dos grupos todas las soluciones: de ciclo abierto y de ciclo cerrado. En esencia, las soluciones de ciclo abierto intentan resolver el problema mediante un buen diseño, para asegurarse en primer lugar de que no ocurra. Una vez que el sistema está en funcionamiento, no se hacen correcciones a medio camino.

Las herramientas para llevar a cabo control de ciclo abierto incluyen decidir cuándo aceptar tráfico nuevo, decidir cuándo descartar paquetes, y cuáles, y tomar decisiones de calendarización en varios puntos de la red. Todas tienen en común el hecho de que toman decisiones independientemente del estado actual de la red.

En contraste, las soluciones de ciclo cerrado se basan en el concepto de un ciclo de retroalimentación. Este método tiene tres partes cuando se aplica al control de congestión:

1. Monitorear el sistema para detectar cuándo y dónde ocurren congestiones.
2. Pasar esta información a lugares en los que puedan llevarse a cabo acciones.
3. Ajustar la operación del sistema para corregir el problema.

Es posible utilizar varias métricas para monitorear la subred en busca de congestiones. Las principales son el porcentaje de paquetes descartados debido a falta de espacio de búfer, la longitud

promedio de las colas, la cantidad de paquetes para los cuales termina el temporizador y se transmiten de nueva cuenta, el retardo promedio de los paquetes y la desviación estándar del retardo de paquete. En todos los casos, un aumento en las cifras indica un aumento en la congestión.

El segundo paso del ciclo de retroalimentación es la transferencia de información relativa a la congestión desde el punto en que se detecta hasta el punto en que puede hacerse algo al respecto. La manera más obvia es que el enrutador que detecta la congestión envíe un paquete al origen (u orígenes) del tráfico, anunciando el problema. Por supuesto, estos paquetes adicionales aumentan la carga precisamente en el momento en que no se necesita más carga, es decir, cuando la subred está congestionada.

Por fortuna, existen otras opciones. Por ejemplo, en cada paquete puede reservarse un bit o campo para que los enrutadores lo llenen cuando la congestión rebase algún umbral. Cuando un enrutador detecta este estado congestionado, llena el campo de todos los paquetes de salida, para avisar a los vecinos.

Otra estrategia es hacer que los *hosts* o enrutadores envíen de manera periódica paquetes de sondeo para preguntar explícitamente sobre la congestión. Esta información puede usarse para enrutar el tráfico fuera de áreas con problemas. Algunas estaciones de radio tienen helicópteros que vuelan sobre la ciudad para informar de la congestión en las calles, con la esperanza de que los escuchas enrutarán sus paquetes (autos) fuera de las zonas conflictivas.

En todos los esquemas de retroalimentación, la esperanza es que el conocimiento sobre la congestión hará que los *hosts* emprendan acciones adecuadas con miras a reducir la congestión. Para operar en forma correcta, la escala de tiempo debe ajustarse con cuidado. Si el enrutador grita ALTO cada vez que llegan dos paquetes seguidos, y SIGA, cada vez que está inactivo durante 20 μseg, el sistema oscilará sin control y nunca convergerá. Por otra parte, si un enrutador espera 30 minutos para asegurarse antes de tomar una decisión, el mecanismo de control de congestión reaccionará tan lentamente que no será de utilidad. Para funcionar bien se requiere un justo medio, pero encontrar la constante de tiempo correcta no es un asunto trivial.

Se conocen muchos algoritmos de control de congestión. A fin de organizarlos lógicamente, Yang y Reddy (1995) han desarrollado una taxonomía de los algoritmos de control de congestión. Comienzan por dividir todos los algoritmos en ciclo abierto y ciclo cerrado, como se describió anteriormente. Dividen todavía más los algoritmos de ciclo abierto en algoritmos que actúan en el origen y los que actúan en el destino. Los algoritmos de ciclo cerrado también se dividen en dos subcategorías: retroalimentación explícita e implícita. En los algoritmos de retroalimentación explícita, regresan paquetes desde el punto de congestión para avisar al origen. En los algoritmos implícitos, el origen deduce la existencia de una congestión haciendo observaciones locales, como el tiempo necesario para que regresen las confirmaciones de recepción.

La presencia de congestión significa que la carga es (temporalmente) superior (en una parte del sistema) a la que pueden manejar los recursos. Vienen a la mente dos soluciones: aumentar los recursos o disminuir la carga. Por ejemplo, la subred puede comenzar a utilizar líneas de acceso telefónico para aumentar de manera temporal el ancho de banda entre ciertos puntos. En los sistemas satelitales la potencia de transmisión creciente con frecuencia reditúa un ancho de banda más alto. La división del tráfico entre varias rutas en lugar de usar siempre la mejor también aumenta efectivamente el ancho de banda. Por último, a fin de contar con mayor capacidad, los enrutadores

de repuesto que normalmente sirven sólo como respaldo (para hacer que el sistema sea tolerante a fallas), pueden ponerse en línea cuando aparece una congestión severa.

Sin embargo, a veces no es posible aumentar la capacidad, o ya ha sido aumentada al máximo. Entonces, la única forma de combatir la congestión es disminuir la carga. Existen varias maneras de reducir la carga, como negar el servicio a algunos usuarios, degradar el servicio para algunos o todos los usuarios y obligar a los usuarios a programar sus solicitudes de una manera más predecible.

Algunos de estos métodos, que estudiaremos en breve, se aplican mejor a los circuitos virtuales. En las subredes que usan circuitos virtuales de manera interna, estos métodos pueden usarse en la capa de red. En las subredes de datagramas algunas veces se pueden utilizar en las conexiones de capa de transporte. En este capítulo nos enfocaremos en su uso en la capa de red. En el siguiente veremos lo que puede hacerse en la capa de transporte para manejar la congestión.

5.3.2 Políticas de prevención de congestión

Comencemos nuestro estudio de los métodos de control de congestión estudiando los sistemas de ciclo abierto. Estos sistemas están diseñados para reducir al mínimo la congestión desde el inicio, en lugar de permitir que ocurra y reaccionar después del hecho. Tratan de lograr su objetivo usando políticas adecuadas en varios niveles. En la figura 5-26 vemos diferentes políticas para las capas de enlace de datos, red y transporte que pueden afectar a la congestión (Jain, 1990).

Capa	Políticas
Transporte	• Política de retransmisión • Política de almacenamiento en caché de paquetes fuera de orden • Política de confirmaciones de recepción • Política de control de flujo • Determinación de terminaciones de temporizador
Red	• Circuitos virtuales vs. datagramas en la subred • Política de encolamiento y servicio de paquetes • Política de descarte de paquetes • Algoritmo de enrutamiento • Administración de tiempo de vida del paquete
Enlace de datos	• Política de retransmisiones • Política de almacenamiento en caché de paquetes fuera de orden • Política de confirmación de recepción • Política de control de flujo

Figura 5-26. Políticas relacionadas con la congestión.

Comencemos por la capa de enlace de datos y avancemos hacia arriba. La política de retransmisiones tiene que ver con la rapidez con la que un emisor termina de temporizar y con lo que transmite al ocurrir una terminación de temporizador. Un emisor nervioso que a veces termina de temporizar demasiado pronto y retransmite todos los paquetes pendientes usando el protocolo de

retroceso n impondrá una carga más pesada al sistema que un emisor calmado que usa repetición selectiva. La política de almacenamiento en caché está muy relacionada con esto. Si los receptores descartan de manera rutinaria todos los paquetes que llegan fuera de orden, posteriormente se tendrán que enviar otra vez, lo que creará una carga extra. Con respecto al control de congestión, la repetición selectiva es mejor que el retroceso n.

La política de confirmación de recepción también afecta la congestión. Si la recepción de cada paquete se confirma de inmediato, los paquetes de confirmación de recepción generan tráfico extra. Sin embargo, si se guardan las confirmaciones de recepción para sobreponerlas en el tráfico en sentido inverso, pueden resultar en terminaciones de temporizador y retransmisiones extra. Un esquema de control de flujo estricto (por ejemplo, una ventana pequeña) reduce la tasa de datos y permite, por lo tanto, atacar la congestión.

En la capa de red, la decisión entre circuitos virtuales y datagramas afecta la congestión, ya que muchos algoritmos de control de congestión sólo funcionan con subredes de circuitos virtuales. La política de encolamiento y servicio de paquetes se refiere a que los enrutadores tengan una cola por línea de entrada, y una o varias colas por línea de salida. También se relaciona con el orden en que se procesan los paquetes (por ejemplo, en *round robin* o con base en prioridades). La política de descarte es la regla que indica qué paquete se descarta cuando no hay espacio. Una buena política puede ayudar a aliviar la congestión y una mala puede hacerlo peor.

Un buen algoritmo de enrutamiento puede evitar la congestión si distribuye el tráfico entre todas las líneas, pero uno malo puede enviar demasiado tráfico por líneas ya congestionadas. Por último, la administración del tiempo de vida de los paquetes se encarga del tiempo que puede existir un paquete antes de ser descartado. Si este tiempo es demasiado grande, los paquetes perdidos pueden bloquear la operación durante un buen rato, pero si es demasiado corto, los paquetes pueden expirar antes de llegar a su destino, lo que provoca retransmisiones.

En la capa de transporte surgen los mismos problemas que en la capa de enlace de datos, pero además es más difícil la determinación del intervalo de expiración, porque el tiempo de tránsito a través de la red es menos predecible que el tiempo de tránsito por un cable entre dos enrutadores. Si el intervalo es demasiado corto, se enviarán paquetes extra de manera innecesaria. Si es muy largo, se reducirá la congestión, pero el tiempo de respuesta se verá afectado cada vez que se pierda un paquete.

5.3.3 Control de congestión en subredes de circuitos virtuales

Los métodos de control de congestión antes descritos son básicamente de ciclo abierto: tratan de evitar que ocurran las congestiones, en lugar de manejarlas una vez ocurridas. En esta sección describiremos algunos métodos para el control dinámico de la congestión en las subredes de circuitos virtuales. En las siguientes dos veremos técnicas que pueden usarse en cualquier subred.

Una de las técnicas que se usa ampliamente para evitar que empeoren las congestiones que ya han comenzado es el **control de admisión.** La idea es sencilla: una vez que se ha detectado la congestión, no se establecen circuitos virtuales nuevos hasta que ha desaparecido el problema. Por

lo tanto, fallan los intentos por establecer conexiones nuevas de capa de transporte. Permitir el acceso a más personas simplemente empeoraría las cosas. Aunque este métodos es simple, su implementación es sencilla. En el sistema telefónico, cuando un conmutador se sobrecarga también se pone en práctica el control de admisión, al no darse tonos de marcado.

Un método alterno es permitir el establecimiento de nuevos circuitos virtuales, pero enrutando cuidadosamente los circuitos nuevos por otras rutas que no tengan problemas. Por ejemplo, considere la subred de la figura 5-27(a), en la que dos enrutadores están congestionados.

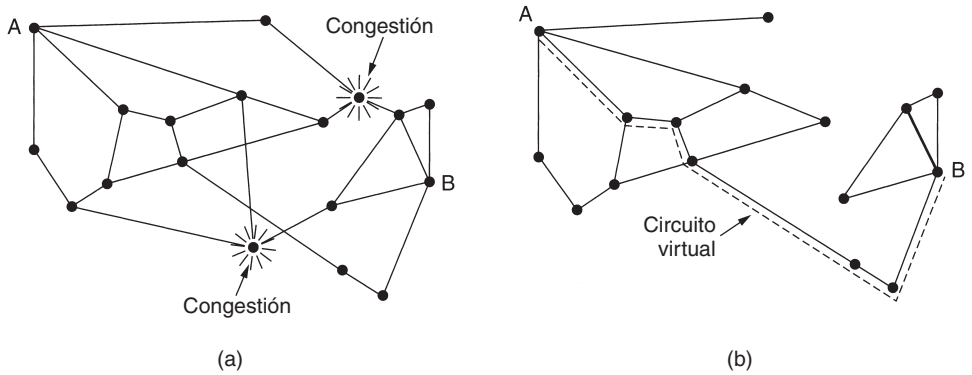


Figura 5-27. (a) Subred congestionada. (b) Subred redibujada que elimina la congestión. También se muestra un circuito virtual de *A* a *B*.

Suponga que un *host* conectado al enrutador *A* quiere establecer una conexión con un *host* conectado al enrutador *B*. Normalmente esta conexión pasaría a través de uno de los enrutadores congestionados. Para evitar esta situación, podemos redibujar la subred como se muestra en la figura 5-27(b), omitiendo los enrutadores congestionados y todas sus líneas. La línea punteada muestra una ruta posible para el circuito virtual que evita los enrutadores congestionados.

Otra estrategia que tiene que ver con los circuitos virtuales es negociar un acuerdo entre el *host* y la subred cuando se establece un circuito virtual. Este arreglo normalmente especifica el volumen y la forma del tráfico, la calidad de servicio requerida y otros parámetros. Para cumplir con su parte del acuerdo, la subred por lo general reservará recursos a lo largo de la ruta cuando se establezca el circuito. Estos recursos pueden incluir espacio en tablas y en búfer en los enrutadores y ancho de banda en las líneas. De esta manera, es poco probable que ocurran congestiones en los circuitos virtuales nuevos, porque está garantizada la disponibilidad de todos los recursos necesarios.

Este tipo de reservación puede llevarse a cabo todo el tiempo como procedimiento operativo estándar, o sólo cuando la subred está congestionada. Una desventaja de hacerlo todo el tiempo es que se tiende a desperdiciar recursos. Si seis circuitos virtuales que podrían usar 1 Mbps pasan por la misma línea física de 6 Mbps, la línea tiene que marcarse como llena, aunque pocas veces ocurra que los seis circuitos virtuales transmitan a toda velocidad al mismo tiempo. En consecuencia, el precio del control de congestión es un ancho de banda sin utilizar (es decir, desperdiciado).

5.3.4 Control de congestión en subredes de datagramas

Veamos ahora un enfoque que puede usarse en subredes de datagramas (y también en subredes de circuitos virtuales). Cada enrutador puede monitorear fácilmente el uso de sus líneas de salida y de otros recursos. Por ejemplo, puede asociar cada línea a una variable real, u, cuyo valor, entre 0.0 y 1.0, refleja el uso reciente de esa línea. Para tener una buena estimación de u, puede tomarse periódicamente una muestra del uso instantáneo de la línea, f (0 o 1), y actualizar u periódicamente de acuerdo con

$$u_{\text{nvo}} = au_{\text{ant}} + (1 - a)f$$

donde la constante a determina la rapidez con que el enrutador olvida la historia reciente.

Siempre que u rebasa el umbral, la línea de salida entra en un estado de "advertencia". Cada paquete nuevo que llega se revisa para ver si su línea de salida está en el estado de advertencia. Si es así, se realiza alguna acción. Ésta puede ser una de varias alternativas, que analizaremos a continuación.

El bit de advertencia

La arquitectura DECNET antigua señalaba el estado de advertencia activando un bit especial en el encabezado del paquete. Frame relay también lo hace. Cuando el paquete llegaba a su destino, la entidad transportadora copiaba el bit en la siguiente confirmación de recepción que se regresaba al origen. A continuación el origen reducía el tráfico.

Mientras el enrutador estuviera en el estado de advertencia, continuaba activando el bit de advertencia, lo que significaba que el origen continuaba obteniendo confirmaciones de recepción con dicho bit activado. El origen monitoreaba la fracción de confirmaciones de recepción con el bit activado y ajustaba su tasa de transmisión de manera acorde. En tanto los bits de advertencia continuaran fluyendo, el origen continuaba disminuyendo su tasa de transmisión. Cuando disminuía lo suficiente, el origen incrementaba su tasa de transmisión. Observe que debido a que cada enrutador a lo largo de la ruta podía activar el bit de advertencia, el tráfico se incrementaba sólo cuando no había enrutadores con problemas.

Paquetes reguladores

El algoritmo de control de congestión anterior es muy sutil. Utiliza medios indirectos para indicar al origen que vaya más despacio. ¿Por qué no indicárselo de manera directa? En este método, el enrutador regresa un **paquete regulador** al *host* de origen, proporcionándole el destino encontrado en el paquete. El paquete original se etiqueta (se activa un bit del encabezado) de manera que no genere más paquetes reguladores más adelante en la ruta y después se reenvía de la manera usual.

Cuando el *host* de origen obtiene el paquete regulador, se le pide que reduzca en un porcentaje X el tráfico enviado al destino especificado. Puesto que otros paquetes dirigidos al mismo destino probablemente ya están en camino y generarán más paquetes reguladores, el *host* debe ignorar

los paquetes reguladores que se refieran a ese destino por un intervalo fijo de tiempo. Una vez que haya expirado ese tiempo, el *host* escucha más paquetes reguladores durante otro intervalo. Si llega alguno, la línea todavía está congestionada, por lo que el *host* reduce el flujo aún más y comienza a ignorar nuevamente los paquetes reguladores. Si no llega ningún paquete de este tipo durante el periodo de escucha, el *host* puede incrementar el flujo otra vez. La retroalimentación implícita de este protocolo puede ayudar a evitar la congestión que aún no estrangula ningún flujo a menos que ocurra un problema.

Los *hosts* pueden reducir el tráfico ajustando los parámetros de sus políticas, por ejemplo, su tamaño de ventana. Por lo general, el primer paquete regulador causa que la tasa de datos se reduzca en 0.50 con respecto a su tasa anterior, el siguiente causa una reducción de 0.25, etcétera. Los incrementos se dan en aumentos más pequeños para evitar que la congestión se vuelva a generar rápidamente.

Se han propuesto algunas variaciones de este algoritmo de control de congestión. En una, los enrutadores pueden mantener varios umbrales. Dependiendo de qué umbral se ha rebasado, el paquete regulador puede contener una advertencia suave, una severa o un ultimátum.

Otra variación es utilizar como señal de activación tamaños de colas o utilización de los búferes en lugar de la utilización de la línea. Es posible utilizar la misma ponderación exponencial con esta métrica como con u, por supuesto.

Paquetes reguladores de salto por salto

A altas velocidades o distancias grandes, el envío de un paquete regulador a los *hosts* de origen no funciona bien porque la reacción es muy lenta. Por ejemplo, considere a un *host* en San Francisco (enrutador *A* de la figura 5-28) que está enviando tráfico a un *host* en Nueva York (enrutador *D* de la figura 5-28) a 155 Mbps. Si al *host* de Nueva York se le comienza a terminar el espacio de búfer, un paquete regulador tardará unos 30 mseg en regresar a San Francisco para indicarle que disminuya su velocidad. La propagación de paquetes reguladores se muestra en el segundo, tercer y cuarto pasos de la figura 5-28(a). En esos 30 mseg se habrán enviado otros 4.6 megabits. Aun si el *host* de San Francisco se desactiva de inmediato, los 4.6 megabits en la línea continuarán fluyendo, y habrá que encargarse de ellos. Sólo hasta el séptimo diagrama de la figura 5-28(a) el enrutador de Nueva York notará un flujo más lento.

Un método alterno es hacer que el paquete regulador ejerza su efecto en cada salto que dé, como se muestra en la secuencia de la figura 5-28(b). Aquí, una vez que el paquete regulador llega a *F*, se obliga a *F* a reducir el flujo a *D*. Hacerlo requerirá que *F* destine más búferes al flujo, ya que el origen aún está transmitiendo a toda velocidad, pero da a *D* un alivio inmediato, como en un mensaje comercial de un remedio contra el dolor de cabeza. En el siguiente paso, el paquete regulador llega a *E*, e indica a éste que reduzca el flujo a *F*. Esta acción impone una mayor carga a los búferes de *E*, pero da un alivio inmediato a *F*. Por último, el paquete regulador llega a *A* y efectivamente se reduce el flujo.

El efecto neto de este esquema de salto por salto es proporcionar un alivio rápido al punto de congestión, a expensas de usar más búferes ascendentes. De esta manera puede cortarse la congestión

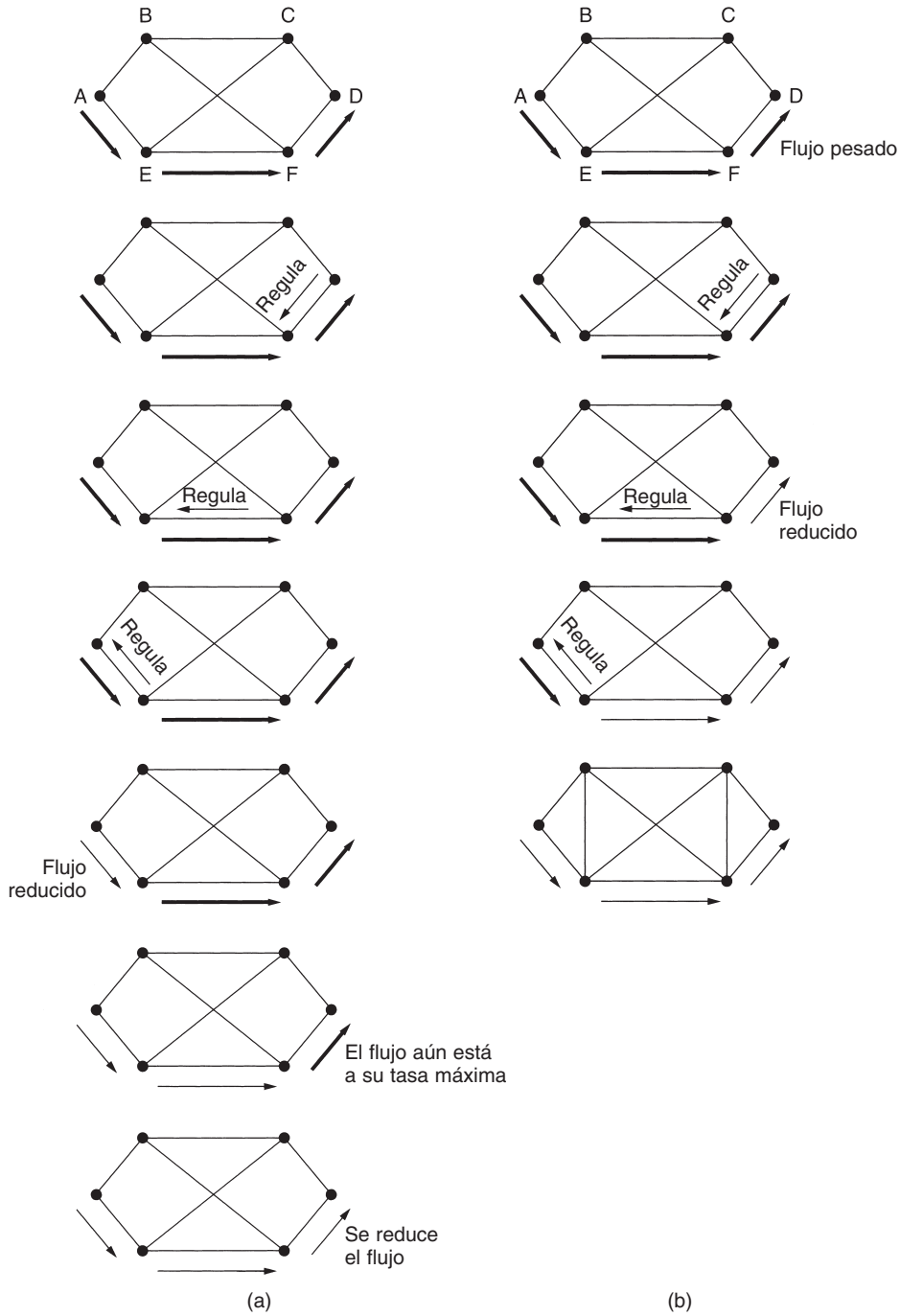


Figura 5-28. (a) Paquete regulador que afecta sólo al origen. (b) Paquete regulador que afecta cada salto por el que pasa.

en la raíz, sin que se pierdan paquetes. La idea se estudia con mayor detalle en Mishra y Kanakia, 1992.

5.3.5 Desprendimiento de carga

Cuando ninguno de los métodos anteriores elimina la congestión, los enrutadores pueden sacar la artillería pesada: el **desprendimiento de carga,** que es una manera rebuscada de decir que, cuando se inunda a los enrutadores con paquetes que no pueden manejar, simplemente los tiran. El término viene del mundo de la generación de energía eléctrica, donde se refiere a la práctica de instalaciones que intencionalmente producen apagones en ciertas áreas para salvar a la red completa de venirse abajo en días calurosos de verano en los que la demanda de energía eléctrica excede por mucho el suministro.

Un enrutador abrumado por paquetes simplemente puede escoger paquetes al azar para desprenderse de ellos, pero normalmente puede hacer algo mejor. El paquete a descartar puede depender de las aplicaciones que se estén ejecutando. En la transferencia de archivos vale más un paquete viejo que uno nuevo, pues el deshacerse del paquete 6 y mantener los paquetes 7 a 10 causará un hueco en el receptor que podría obligar a que se retransmitan los paquetes 6 a 10 (si el receptor descarta de manera rutinaria los paquetes en desorden). En un archivo de 12 paquetes, deshacerse del paquete 6 podría requerir la retransmisión de los paquetes 7 a 12, y deshacerse del 10 podría requerir la retransmisión sólo del 10 al 12. En contraste, en multimedia es más importante un paquete nuevo que uno viejo. La política anterior (más viejo es mejor que más nuevo), con frecuencia se llama **vino,** y la última (más nuevo es mejor que más viejo) con frecuencia se llama **leche.**

Un paso adelante de esto en cuanto a inteligencia requiere la cooperación de los emisores. En muchas aplicaciones, algunos paquetes son más importantes que otros. Por ejemplo, ciertos algoritmos de compresión de vídeo transmiten periódicamente una trama entera y sus tramas subsiguientes como diferencias respecto a la última trama completa. En este caso, es preferible desprenderse de un paquete que es parte de una diferencia que desprenderse de uno que es parte de una trama completa. Como otro ejemplo, considere la transmisión de un documento que contiene texto ASCII e imágenes. La pérdida de una línea de píxeles de una imagen es mucho menos dañina que la pérdida de una línea de texto legible.

Para poner en práctica una política inteligente de descarte, las aplicaciones deben marcar sus paquetes con clases de prioridades para indicar su importancia. Si lo hacen, al tener que descartar paquetes, los enrutadores pueden descartar primero los paquetes de clase más baja, luego los de la siguiente clase más baja, etcétera. Por supuesto, a menos que haya una razón poderosa para marcar los paquetes como MUY IMPORTANTE–NUNCA DESCARTAR, nadie lo hará.

La razón podría ser monetaria, siendo más barato el envío de paquetes de baja prioridad que el de los de alta prioridad. Como alternativa, los emisores podrían tener permitido enviar paquetes de alta prioridad bajo condiciones de carga ligera, pero a medida que aumente la carga, los paquetes podrían descartarse, lo que haría que los usuarios ya no siguieran enviándolos.

Otra opción es permitir que los *hosts* excedan los límites especificados en el acuerdo negociado al establecer el circuito virtual (por ejemplo, usar un ancho de banda mayor que el permitido),

pero sujetos a la condición de que el exceso de tráfico se marque con prioridad baja. Tal estrategia de hecho no es mala idea, porque utiliza con mayor eficiencia los recursos inactivos, permitiendo que los *hosts* los utilicen siempre y cuando nadie más esté interesado, pero sin establecer un derecho sobre ellos cuando los tiempos se vuelven difíciles.

Detección temprana aleatoria

Es bien sabido que tratar con la congestión después de que se detecta por primera vez es más efectivo que dejar que dañe el trabajo y luego tratar de solucionarlo. Esta observación conduce a la idea de descartar paquetes antes de que se ocupe todo el espacio de búfer. Un algoritmo popular para realizar esto se conoce como **RED** (**detección temprana aleatoria**) (Floyd y Jacobson, 1993). En algunos protocolos de transporte (entre ellos TCP), la respuesta a paquetes perdidos es que el origen disminuya su velocidad. El razonamiento detrás de esta lógica es que TCP fue diseñado para redes cableadas, y éstas son muy confiables, por lo tanto, la pérdida de paquetes se debe principalmente a desbordamientos de búfer y no a errores de transmisiones. Este hecho puede aprovecharse para reducir la congestión.

El objetivo de hacer que los enrutadores se deshagan de los paquetes antes de que la situación sea irremediable (de aquí el término "temprana" en el nombre), es que haya tiempo para hacer algo antes de que sea demasiado tarde. Para determinar cuándo comenzar a descartarlos, los enrutadores mantienen un promedio móvil de sus longitudes de cola. Cuando la longitud de cola promedio en algunas líneas sobrepasa un umbral, se dice que la línea está congestionada y se toma alguna medida.

Debido a que tal vez el enrutador no puede saber cuál origen está causando la mayoría de los problemas, probablemente lo mejor que se puede hacer es elegir un paquete al azar de la cola que puso en marcha la acción.

¿Cómo puede el enrutador informar al origen sobre el problema? Una forma es enviarle un paquete regulador, como describimos anteriormente. Sin embargo, con ese método surge un problema ya que coloca todavía más carga en la ya congestionada red. Una estrategia diferente es descartar el paquete seleccionado y no reportarlo. El origen notará en algún momento la falta de confirmación de recepción y tomará medidas. Debido a que sabe que los paquetes perdidos por lo general son causados por la congestión y las eliminaciones, responderá reduciendo la velocidad en lugar de aumentarla. Esta forma implícita de retroalimentación sólo funciona cuando los orígenes responden a la pérdida de paquetes reduciendo su tasa de transmisión. En las redes inalámbricas, en las que la mayoría de las pérdidas se debe al ruido en el enlace de radio, no se puede utilizar este método.

5.3.6 Control de fluctuación

En aplicaciones como la transmisión continua de audio y vídeo no importa gran cosa si los paquetes tardan 20 o 30 mseg en ser entregados, siempre y cuando el tiempo de tránsito (retardo) sea constante. La variación (es decir, la desviación estándar) en el retardo de los paquetes se conoce como **fluctuación.** Una fluctuación alta, por ejemplo cuando unos paquetes tardan en llegar a su

destino 20 mseg y otros 30 mseg, resultará en una calidad desigual del sonido o la imagen. En la figura 5-29 se ilustra la fluctuación. Por tanto, el arreglo de que el 99% de los paquetes se entregue con un retardo que esté entre 24.5 y 25.5 mseg puede ser aceptable.

Por supuesto, el rango escogido debe ser factible. Debe tomar en cuenta el retardo causado por la velocidad de la luz y el retardo mínimo a través de los enrutadores y tal vez dejar un periodo corto para algunos retardos inevitables.

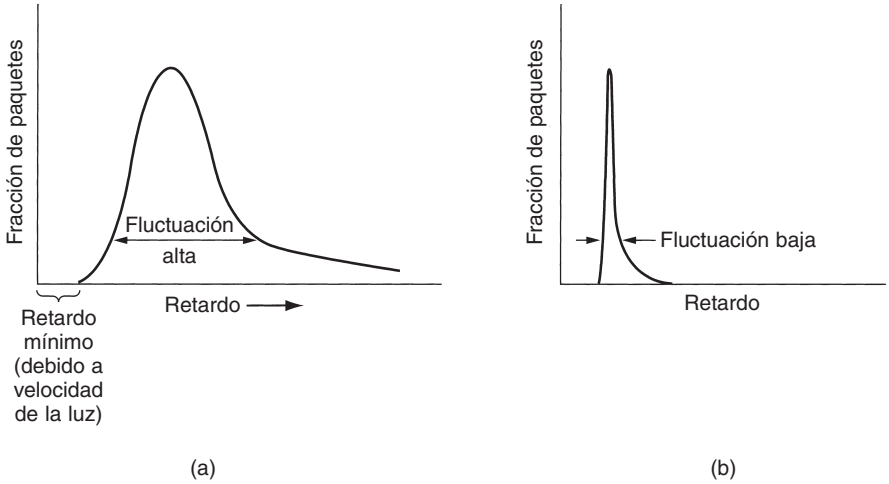


Figura 5-29. (a) Fluctuación alta. (b) Fluctuación baja.

La fluctuación puede limitarse calculando el tiempo de tránsito esperado para cada salto en la ruta. Cuando un paquete llega a un enrutador, éste lo examina para saber qué tan adelantado o retrasado está respecto a lo programado. Esta información se almacena en el paquete y se actualiza en cada salto. Si el paquete está adelantado, se retiene durante el tiempo suficiente para regresarlo a lo programado; si está retrasado, el enrutador trata de sacarlo rápidamente.

De hecho, el algoritmo para determinar cuál de varios paquetes que compiten por una línea de salida debe seguir siempre puede escoger el paquete más retrasado. De esta manera, los paquetes adelantados se frenan y los retrasados se aceleran, reduciendo en ambos casos la cantidad de fluctuación.

En algunas aplicaciones, como el vídeo bajo demanda, la fluctuación puede eliminarse almacenando los datos en el búfer del receptor y después obteniéndolos de dicho búfer en lugar de utilizar la red en tiempo real. Sin embargo, para otras aplicaciones, especialmente aquellas que requieren interacción en tiempo real entre personas como la telefonía y videoconferencia en Internet, el retardo inherente del almacenamiento en el búfer no es aceptable.

El control de congestión es un área activa de investigación. El estado presente se resume en (Gevros y cols., 2001).

5.4 CALIDAD DEL SERVICIO

Las técnicas que observamos en las secciones previas se diseñaron para reducir la congestión y mejorar el rendimiento de la red. Sin embargo, con el crecimiento de las redes de multimedia, con frecuencia estas medidas *ad hoc* no son suficientes. Se necesitan intentos serios para garantizar la calidad del servicio a través del diseño de redes y protocolos. En las siguientes secciones continuaremos nuestro estudio del rendimiento de la red, pero por ahora nos enfocaremos en las formas de proporcionar una calidad de servicio que se ajuste a las necesidades de las aplicaciones. Sin embargo, se debe dejar claro desde el principio que muchas de estas ideas están empezando y sujetas a cambios.

5.4.1 Requerimientos

Un **flujo** es un conjunto de paquetes que van de un origen a un destino. En una red orientada a la conexión, todos los paquetes que pertenezcan a un flujo siguen la misma ruta; en una red sin conexión, pueden seguir diferentes rutas. La necesidad de cada flujo se puede caracterizar por cuatro parámetros principales: confiabilidad, retardo, fluctuación y ancho de banda. Estos parámetros en conjunto determinan la **QoS** (**calidad del servicio**) que el flujo requiere. En la figura 5-30 se listan varias aplicaciones y el nivel de sus requerimientos.

Aplicación	Confiabilidad	Retardo	Fluctuación	Ancho de banda
Correo electrónico	Alta	Bajo	Baja	Bajo
Transferencia de archivos	Alta	Bajo	Baja	Medio
Acceso a Web	Alta	Medio	Baja	Medio
Inicio de sesión remoto	Alta	Medio	Media	Bajo
Audio bajo demanda	Baja	Bajo	Alta	Medio
Vídeo bajo demanda	Baja	Bajo	Alta	Alto
Telefonía	Baja	Alto	Alta	Bajo
Videoconferencia	Baja	Alto	Alta	Alto

Figura 5-30. Qué tan rigurosos son los requerimientos de calidad del servicio.

Las primeras cuatro aplicaciones tienen requerimientos rigurosos en cuanto a confiabilidad. No sería posible enviar bits de manera incorrecta. Este objetivo por lo general se alcanza al realizar una suma de verificación de cada paquete y al verificar dicha suma en el destino. Si se daña un paquete en el tránsito, no se confirma su recepción y se volverá a transmitir posteriormente. Esta estrategia proporciona una alta confiabilidad. Las cuatro aplicaciones finales (audio/vídeo) pueden tolerar errores, por lo que ni se realizan ni comprueban sumas de verificación.

Las aplicaciones de transferencia de archivos, incluyendo correo electrónico y vídeo, no son sensibles al retardo. Si todos los paquetes se retrasan unos segundos de manera uniforme, no hay daño. Las aplicaciones interactivas, como la navegación en Web y el inicio de sesión remoto,

son más sensibles a los retardos. Las aplicaciones en tiempo real, como la telefonía y la videoconferencia, tienen requerimientos estrictos de retardo. Si cada una de las palabras de una llamada telefónica se retrasa exactamente por 2.000 seg, los usuarios hallarán la conexión inaceptable. Por otra parte, la reproducción de archivos de audio o vídeo desde un servidor no requiere un retardo bajo.

Las primeras tres aplicaciones no son sensibles a los paquetes que llegan con intervalos de tiempo irregulares entre ellos. El inicio de sesión remoto es algo sensible a esto, debido a que los caracteres en la pantalla aparecerán en pequeñas ráfagas si una conexión tiene mucha fluctuación. El vídeo y especialmente el audio son en extremo sensibles a la fluctuación. Si un usuario está observando vídeo a través de la red y todos los cuadros se retrasan exactamente 2.000 seg, no hay daño. Pero si el tiempo de transmisión varía de manera aleatoria entre 1 y 2 seg, el resultado sería terrible. En el audio, una fluctuación de incluso unos cuantos milisegundos es claramente audible.

Por último, las aplicaciones difieren en sus anchos de banda; el correo electrónico y el inicio de sesión remoto no necesitan mucho, pero el vídeo en todas sus formas sí lo necesita.

Las redes ATM clasifican los flujos en cuatro categorías amplias con respecto a sus demandas de QoS, como se muestra a continuación:

1. Tasa de bits constante (por ejemplo, telefonía).
2. Tasa de bits variable en tiempo real (por ejemplo, videoconferencia comprimida).
3. Tasa de bits variable no constante (por ejemplo, ver una película a través de Internet).
4. Tasa de bits disponible (por ejemplo, transferencia de archivos).

Estas categorías también son útiles para otros propósitos y otras redes. La tasa de bits constante es un intento por simular un cable al proporcionar un ancho de banda uniforme y un retardo uniforme. La tasa de bits variable ocurre cuando el vídeo está comprimido, algunos cuadros están más comprimidos que otros. Por lo tanto, el envío de un cuadro con muchos detalles podría requerir enviar muchos bits en tanto que el envío de una foto de una pared blanca podría comprimirse muy bien. La tasa de bits disponible es para las aplicaciones, como el correo electrónico, que no son sensibles al retardo o a la fluctuación.

5.4.2 Técnicas para alcanzar buena calidad de servicio

Ahora que sabemos algo sobre los requerimientos de QoS, ¿cómo cumplimos con ellos? Bueno, para empezar, no hay una bola mágica. Ninguna técnica proporciona QoS eficiente y confiable de una manera óptima. En su lugar, se ha desarrollado una variedad de técnicas, con soluciones prácticas que con frecuencia se combinan múltiples técnicas. A continuación examinaremos algunas de las técnicas que los diseñadores de sistemas utilizan para alcanzar la QoS.

Sobreaprovisionamiento

Una solución fácil es proporcionar la suficiente capacidad de enrutador, espacio en búfer y ancho de banda como para que los paquetes fluyan con facilidad. El problema con esta solución es que

es costosa. Conforme pasa el tiempo y los diseñadores tienen una mejor idea de cuánto es suficiente, esta técnica puede ser práctica. En cierta medida, el sistema telefónico tiene un sobreaprovisionamiento. Es raro levantar un auricular telefónico y no obtener un tono de marcado instantáneo. Simplemente hay mucha capacidad disponible ahí que la demanda siempre se puede satisfacer.

Almacenamiento en búfer

Los flujos pueden almacenarse en el búfer en el lado receptor antes de ser entregados. Almacenarlos en el búfer no afecta la confiabilidad o el ancho de banda, e incrementa el retardo, pero atenúa la fluctuación. Para el vídeo o audio bajo demanda, la fluctuación es el problema principal, por lo tanto, esta técnica es muy útil.

En la figura 5-29 vimos la diferencia entre fluctuación alta y fluctuación baja. En la figura 5-31 vemos un flujo de paquetes que se entregan con una fluctuación considerable. El paquete 1 se envía desde el servidor a $t = 0$ seg y llega al cliente a $t = 1$ seg. El paquete 2 tiene un retardo mayor; tarda 2 seg en llegar. Conforme llegan los paquetes, se almacenan en el búfer en la máquina cliente.

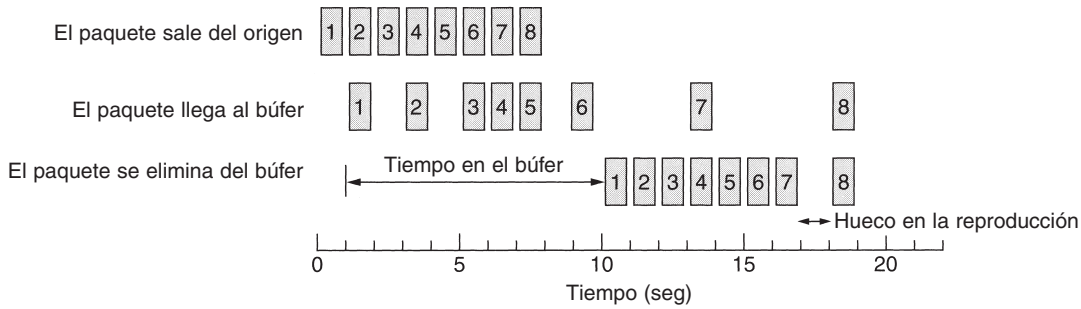


Figura 5-31. Refinamiento del flujo de paquetes almacenándolos en el búfer.

En el seg $t = 10$, la reproducción continúa. En este momento, los paquetes 1 a 6 se han almacenado en el búfer de manera que pueden eliminarse de él en intervalos uniformes para una reproducción suave. Desafortunadamente, el paquete 8 se ha retrasado tanto que no está disponible cuando le toca el turno a su ranura de reproducción, por lo que ésta debe parar hasta que llegue dicho paquete, creando un molesto hueco en la música o película. Este problema se puede atenuar retrasando el tiempo de inicio aún más, aunque hacer eso también requiere un búfer más grande. Los sitios Web comerciales que contienen transmisión continua de vídeo o audio utilizan reproductores que almacenan en el búfer por aproximadamente 10 seg antes de comenzar a reproducir.

Modelado de tráfico

En el ejemplo anterior, el origen envía los paquetes con un espaciado uniforme entre ellos, pero en otros casos, podrían emitirse de manera regular, lo cual puede causar congestión en la red. El envío no uniforme es común si el servidor está manejando muchos flujos al mismo tiempo, y también permite otras acciones, como avance rápido y rebobinado, autenticación de usuario,

etcétera. Además, el enfoque que utilizamos aquí (almacenamiento en el búfer) no siempre es posible, por ejemplo, en la videoconferencia. Sin embargo, si pudiera hacerse algo para hacer que el servidor (y los *hosts* en general) transmita a una tasa uniforme, la calidad del servicio mejoraría. A continuación examinaremos una técnica, el **modelado de tráfico,** que modera el tráfico en el servidor, en lugar de en el cliente.

El modelado de tráfico consiste en regular la *tasa* promedio (y las ráfagas) de la transmisión de los datos. En contraste, los protocolos de ventana corrediza que estudiamos anteriormente limitan la cantidad de datos en tránsito de una vez, no la tasa a la que se envían. Cuando se establece una conexión, el usuario y la subred (es decir, el cliente y la empresa portadora) acuerdan cierto patrón de tráfico (es decir, forma) para ese circuito. Algunas veces esto se llama **acuerdo de nivel de servicio.** En tanto el cliente cumpla con su parte del contrato y sólo envíe los paquetes acordados, la empresa portadora promete entregarlos de manera oportuna. El modelado de tráfico reduce la congestión y, por lo tanto, ayuda a la empresa portadora a cumplir con su promesa. Tales acuerdos no son tan importantes para las transferencias de archivos pero sí para los datos en tiempo real, como conexiones de vídeo y audio, lo cual tiene requerimientos rigurosos de calidad de servicio.

En efecto, con el modelado de tráfico, el cliente le dice a la empresa portadora: Mi patrón de transmisión se parecerá a esto, ¿puedes manejarlo? Si la empresa portadora acepta, surge la cuestión de cómo puede saber ésta si el cliente está cumpliendo con el acuerdo y cómo debe proceder si el cliente no lo está haciendo. La supervisión de un flujo de tráfico se conoce como **supervisión de tráfico** (*traffic policing*). Aceptar una forma de tráfico y supervisarlo más tarde es más fácil en las subredes de circuitos virtuales que en las de datagramas. Sin embargo, incluso en las subredes de datagramas se pueden aplicar las mismas ideas a las conexiones de la capa de transporte.

Algoritmo de cubeta con goteo

Imagínese una cubeta con un pequeño agujero en el fondo, como se ilustra en la figura 5-32(a). Sin importar la rapidez con que entra agua en la cubeta, el flujo de salida tiene una tasa constante, ρ, cuando hay agua en la cubeta, y una tasa de cero cuando la cubeta está vacía. Además, una vez que se llena la cubeta, cualquier agua adicional que entra se derrama por los costados y se pierde (es decir, no aparece en el flujo por debajo del agujero).

Puede aplicarse el mismo concepto a los paquetes, como se muestra en la figura 5-32(b). De manera conceptual, cada *host* está conectado a la red mediante una interfaz que contiene una cubeta con goteo, es decir, una cola interna infinita. Si llega un paquete cuando la cola está llena, éste se descarta. En otras palabras, si uno o más procesos del *host* tratan de enviar paquetes cuando la cola ya tiene la cantidad máxima de paquetes, dicho paquete se descarta sin más. Este arreglo puede incorporarse en la interfaz del hardware, o simularse a través del sistema operativo del *host*. El esquema fue propuesto inicialmente por Turner (1986), y se llama **algoritmo de cubeta con goteo.** De hecho, no es otra cosa que un sistema de encolamiento de un solo servidor con un tiempo de servicio constante.

El *host* puede poner en la red un paquete por pulso de reloj. Nuevamente, esto puede forzarse desde la tarjeta de interfaz o desde el sistema operativo. Este mecanismo convierte un flujo desi-

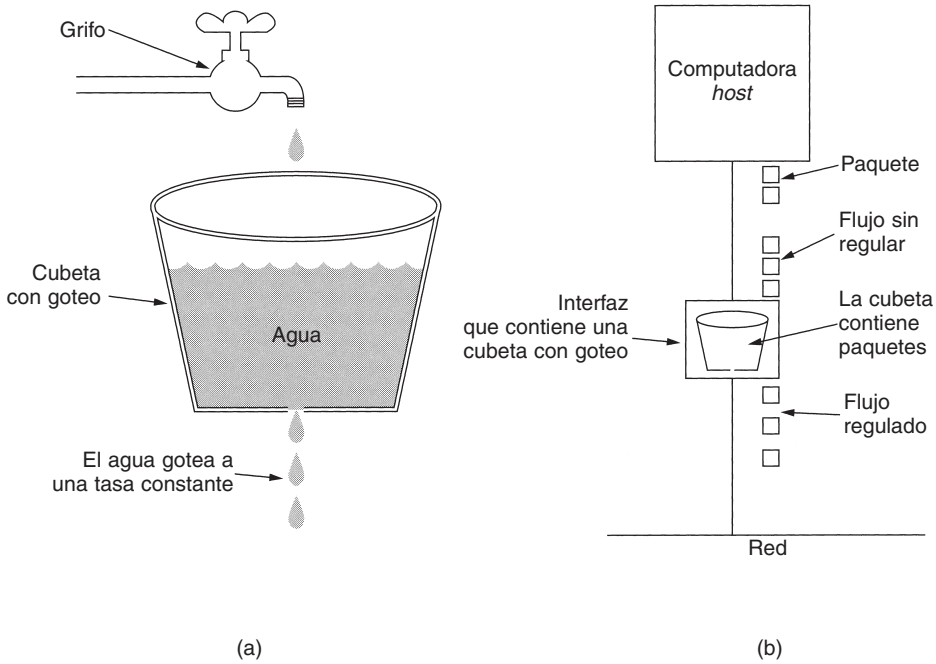


Figura 5-32. (a) Una cubeta con goteo, llena de agua. (b) Cubeta con goteo, llena de paquetes.

gual de paquetes de los procesos de usuario dentro del *host* en un flujo continuo de paquetes hacia la red, moderando las ráfagas y reduciendo en una buena medida las posibilidades de congestión.

Cuando todos los paquetes son del mismo tamaño (por ejemplo, celdas ATM), este algoritmo puede usarse como se describe. Sin embargo, cuando se utilizan paquetes de tamaño variable, con frecuencia es mejor permitir un número fijo de bytes por pulso, en lugar de un solo paquete. Por lo tanto, si la regla es de 1024 bytes por pulso, sólo pueden recibirse por pulso un paquete de 1024 bytes, dos paquetes de 512 bytes, cuatro paquetes de 256 bytes, etcétera. Si el conteo de bytes residuales es demasiado bajo, el siguiente paquete debe esperar hasta el siguiente pulso.

La implementación del algoritmo de cubeta con goteo es fácil. La cubeta con goteo consiste en una cola finita. Si cuando llega un paquete hay espacio en la cola, éste se agrega a ella; de otro modo, se descarta. En cada pulso de reloj se transmite un paquete (a menos que la cola esté vacía).

La cubeta con goteo que usa conteo de bits se implementa casi de la misma manera. En cada pulso un contador se inicializa en n. Si el primer paquete de la cola tiene menos bytes que el valor actual del contador, se transmite y se disminuye el contador en esa cantidad de bytes. Pueden enviarse paquetes adicionales en tanto el contador sea lo suficientemente grande. Cuando el contador está por debajo de la longitud del siguiente paquete de la cola, la transmisión se detiene hasta el siguiente pulso, en cuyo momento se restablece el conteo de bytes residuales y el flujo puede continuar.

Como ejemplo de cubeta con goteo, imagine que una computadora puede producir datos a razón de 25 millones de bytes/seg (200 Mbps) y que la red también opera a esta velocidad. Sin embargo, los enrutadores pueden manejar esta tasa de datos sólo durante intervalos cortos (básicamente, hasta que sus búferes se llenen). Durante intervalos grandes, dichos enrutadores funcionan mejor con tasas que no exceden 2 millones de bytes/seg. Ahora suponga que los datos llegan en ráfagas de un millón de bytes, con una ráfaga de 40 mseg cada segundo. Para reducir la tasa promedio a 2 MB/seg, podemos usar una cubeta con goteo de $\rho = 2$ MB/seg y una capacidad, C, de 1 MB. Esto significa que las ráfagas de hasta 1 MB pueden manejarse sin pérdidas de datos, ya que se distribuyen a través de 500 mseg, sin importar la velocidad a la que lleguen.

En la figura 5-33(a) vemos la entrada de la cubeta con goteo operando a 25 MB/seg durante 40 mseg. En la figura 5-33(b) vemos la salida drenándose a una velocidad uniforme de 2 MB/seg durante 500 mseg.

Algoritmo de cubeta con *tokens*

El algoritmo de cubeta con goteo impone un patrón de salida rígido a la tasa promedio, sin importar la cantidad de ráfagas que tenga el tráfico. En muchas aplicaciones es mejor permitir que la salida se acelere un poco cuando llegan ráfagas grandes, por lo que se necesita un algoritmo más flexible, de preferencia uno que nunca pierda datos. El **algoritmo de cubeta con *tokens*** es uno de tales algoritmos. En este algoritmo, la cubeta con goteo contiene *tokens*, generados por un reloj a razón de un *token* cada ΔT seg. En la figura 5-34(a) se muestra una cubeta que contiene tres *tokens* y cinco paquetes esperando a ser transmitidos. Para que se transmita un paquete, éste debe capturar y destruir un *token*. En la figura 5-34(b) vemos que han pasado tres de los cinco paquetes, pero los otros dos están atorados, esperando la generación de dos o más *tokens*.

El algoritmo de cubeta con *tokens* ofrece una forma diferente de modelado de tráfico que el algoritmo de cubeta con goteo. Este último no permite que los *hosts* inactivos acumulen permisos para enviar posteriormente ráfagas grandes. El algoritmo de cubeta con *tokens* sí permite el ahorro, hasta el tamaño máximo de la cubeta, n. Esta propiedad significa que pueden enviarse a la vez ráfagas de hasta n paquetes, permitiendo cierta irregularidad en el flujo de salida y dando una respuesta más rápida a las ráfagas de entrada repentinas.

Otra diferencia entre los dos algoritmos es que el algoritmo de cubeta con *tokens* descarta los *tokens* (es decir, la capacidad de transmisión) cuando se llena la cubeta, pero nunca descarta los paquetes. En contraste, el algoritmo de cubeta con goteo descarta los paquetes cuando se llena la cubeta.

Aquí también es posible una variación menor, en la que cada *token* representa el derecho de transmitir no un paquete, sino k bytes. Sólo puede transmitirse un paquete si hay suficientes *tokens* disponibles para cubrir su longitud en bytes. Los *tokens* fraccionarios se guardan para uso futuro.

Los algoritmos de cubeta con goteo y cubeta con *tokens* también pueden servir para regular el tráfico entre los enrutadores, así como para regular la salida de un *host*, como en nuestros ejemplos. Sin embargo, una diferencia clara es que una cubeta con *tokens* que regula a un *host* puede hacer que éste detenga el envío cuando las reglas dicen que debe hacerlo. Indicar a un enrutador que detenga la transmisión mientras sigue recibiendo entradas puede dar como resultado una pérdida de datos.

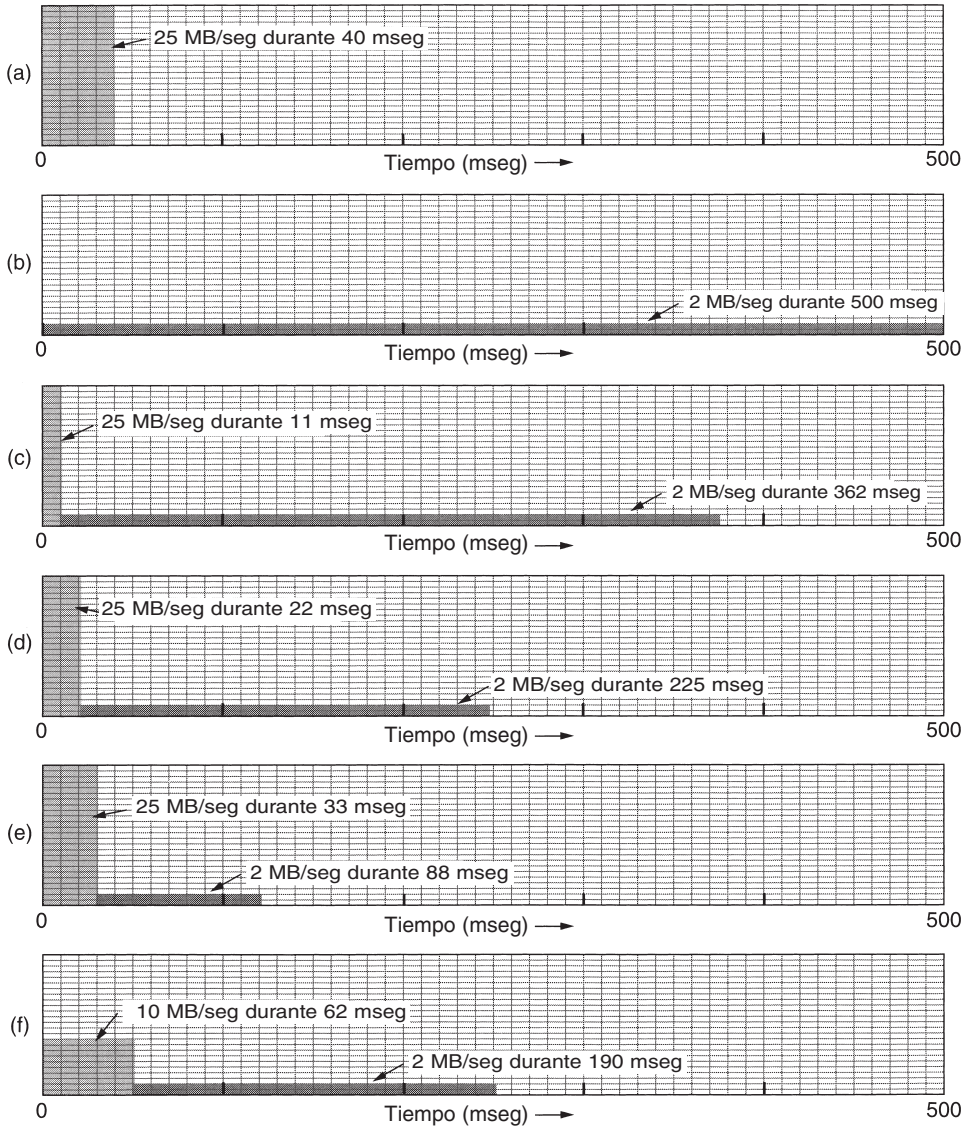


Figura 5-33. (a) Entrada a una cubeta con goteo. (b) Salida de una cubeta con goteo. (c)-(e) Salida de una cubeta con *tokens* con capacidades de 250 KB, 500 KB y 750 KB. (f) Salida de una cubeta con *tokens* de 500 KB que alimenta a una cubeta con goteo de 10 MB/seg.

La implementación del algoritmo básico de cubeta con *tokens* simplemente es sólo una variable que cuenta *tokens*. El contador se incrementa en uno cada ΔT y se decrementa en uno cada vez que se envía un paquete. Cuando el contador llega a cero, ya no pueden enviarse paquetes. En la variante de conteo de bits, el contador se incrementa en k bytes cada ΔT y se decrementa en la longitud de cada paquete enviado.

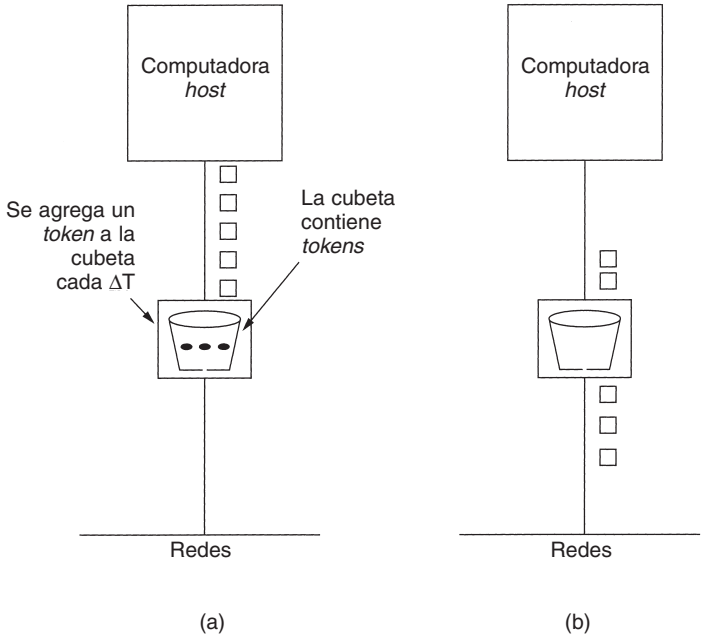


Figura 3-34. Algoritmo de cubeta con *tokens*. (a) Antes. (b) Después.

En esencia, lo que hace la cubeta con *tokens* es permitir ráfagas, pero limitadas a una longitud máxima regulada. Por ejemplo, vea la figura 5-33(c). Ahí tenemos una cubeta de *tokens* con 250 KB de capacidad. Los *tokens* llegan a una tasa que permite un flujo de salida de 2 MB/seg. Suponiendo que la cubeta con *tokens* está llena cuando llega la ráfaga de 1 MB, la cubeta puede drenarse a la velocidad máxima de 25 MB/seg durante unos 11 mseg. Entonces tiene que desacelerarse hasta 2 MB/seg hasta que se ha enviado toda la ráfaga de entrada.

El cálculo de la longitud de ráfaga con tasa máxima es un tanto complicado. No es sólo la división de 1 MB entre 25 MB/seg, ya que, mientras se está enviando la ráfaga, llegan más *tokens*. Si S seg es la longitud de la ráfaga, C bytes es la capacidad de la cubeta con *tokens*, ρ bytes/seg es la tasa de llegada de *tokens* y M bytes/seg es la tasa máxima de salida, podemos ver que una ráfaga de salida contiene un máximo de $C + \rho S$ bytes. También sabemos que la cantidad de bytes en una ráfaga a velocidad máxima con longitud de S segundos es MS. Por lo tanto, tenemos

$$C + \rho S = MS$$

Podemos resolver esta ecuación para obtener $S = C/(M - \rho)$. Para nuestros parámetros de $C = 250$ KB, $M = 25$ MB/seg y $\rho = 2$ MB/seg, tenemos un tiempo de ráfaga de aproximadamente 11 mseg. En la figura 5-33(d) y en la figura 5-33(e) se muestra la cubeta con *tokens* para capacidades de 500 y 750 KB, respectivamente.

Un problema potencial con el algoritmo de cubeta con *tokens* es que permite ráfagas largas, aunque puede regularse el intervalo máximo de ráfaga mediante una selección cuidadosa de ρ y *M*. Con frecuencia es deseable reducir la tasa pico, pero sin regresar al valor mínimo de la cubeta con goteo original.

Una manera de lograr tráfico más uniforme es poner una cubeta con goteo después de la cubeta con *tokens*. La tasa de la cubeta con goteo deberá ser mayor que la ρ de la cubeta con *tokens*, pero menor que la tasa máxima de la red. En la figura 5-33(f) se muestra la salida de una cubeta con *tokens* de 500 KB seguida de una cubeta con goteo de 10 MB/seg.

La supervisión de estos esquemas puede ser un tanto complicada. En esencia, la red tiene que simular el algoritmo y asegurarse de que no se envíen más paquetes o bytes de lo permitido. Sin embargo, estas herramientas proporcionan métodos para modelar el tráfico de la red de formas más manejables para ayudar a cumplir con los requerimientos de calidad del servicio.

Reservación de recursos

El hecho de tener la capacidad de regular la forma del tráfico ofrecido es un buen inicio para garantizar la calidad del servicio. Sin embargo, utilizar efectivamente esta información significa de manera implícita obligar a todos los paquetes de un flujo a que sigan la misma ruta. Su envío a través de enrutadores aleatorios dificulta garantizar algo. Como consecuencia, se debe configurar algo similar a un circuito virtual del origen al destino, y todos los paquetes que pertenecen al flujo deben seguir esta ruta.

Una vez que se tiene una ruta específica para una flujo, es posible reservar recursos a lo largo de esa ruta para asegurar que la capacidad necesaria esté disponible. Se pueden reservar tres tipos de recursos:

1. Ancho de banda.
2. Espacio de búfer.
3. Ciclos de CPU.

El primero, ancho de banda, es el más obvio. Si un flujo requiere 1 Mbps y la línea saliente tiene una capacidad de 2 Mbps, tratar de dirigir tres flujos a través de esa línea no va a funcionar. Por lo tanto, reservar ancho de banda significa no sobrecargar ninguna línea de salida.

Un segundo recurso que por lo general es escaso es el espacio en búfer. Cuando llega un paquete, por lo general el hardware mismo lo deposita en la tarjeta de interfaz de red. A continuación, el software enrutador tiene que copiarlo en un búfer en RAM y colocar en la cola ese búfer para transmitirlo en la línea saliente elegida. Si no hay búfer disponible, el paquete se tiene que descartar debido a que no hay lugar para colocarlo. Para una buena calidad de servicio, es posible reservar algunos búferes para un flujo específico de manera que éste no tenga que competir con otros flujos para obtener espacio en búfer. Siempre que ese flujo necesite un búfer, se le proporcionará uno mientras existan disponibles.

Por último, los ciclos de CPU también son un recurso escaso. Para procesar un paquete se necesita tiempo de CPU del enrutador, por lo que un enrutador sólo puede procesar cierta cantidad de paquetes por segundo. Para asegurar el procesamiento oportuno de cada paquete, es necesario verificar que la CPU no esté sobrecargada.

A primera vista podría parecer que si un enrutador tarda, digamos, 1 μseg, en procesar un paquete, entonces puede procesar 1 millón de paquetes/seg. Esta observación no es verdadera porque siempre habrá periodos inactivos debido a fluctuaciones estadísticas en la carga. Si la CPU necesita cada ciclo para poder realizar su trabajo, la pérdida incluso de algunos ciclos debido a periodos inactivos ocasionales crea un atraso del que nunca se podrá deshacer.

Sin embargo, incluso con una carga que esté ligeramente por debajo de la capacidad teórica, se pueden generar colas y pueden ocurrir retardos. Considere una situación en la que los paquetes llegan de manera aleatoria con una tasa promedio de llegada de λ paquetes/seg. El tiempo de CPU requerido por cada uno también es aleatorio, con una capacidad media de procesamiento de μ paquetes/seg. Bajo el supuesto de que las distribuciones de arribo y de servicio son distribuciones de Poisson, es posible probar, mediante la teoría de encolamiento, que el retardo promedio experimentado por un paquete, *T*, es

$$T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho}$$

donde $\rho = \lambda/\mu$ es el uso de CPU. El primer factor, $1/\mu$, sería el tiempo de servicio si no hubiera competencia. El segundo factor es la reducción de velocidad debido a la competencia con otros flujos. Por ejemplo, si $\lambda = 950{,}000$ paquetes/seg y $\mu = 1{,}000{,}000$ paquetes/seg, entonces $\rho = 0.95$ y el retardo promedio experimentado por cada paquete será de 20 μseg en lugar de 1 μseg. Este tiempo cuenta tanto para el tiempo de encolamiento como para el de servicio, como puede verse cuando la carga es muy baja ($\lambda/\mu \approx 0$). Si hay, digamos, 30 enrutadores a lo largo de la ruta del flujo, el retardo de encolamiento será de alrededor de 600 μseg.

Control de admisión

Ahora nos encontramos en el punto en que el tráfico entrante de algún flujo está bien modelado y puede seguir una sola ruta cuya capacidad puede reservarse por adelantado en los enrutadores a lo largo de la ruta. Cuando un flujo de este tipo se ofrece a un enrutador, éste tiene que decidir, con base en su capacidad y en cuántos compromisos tiene con otros flujos, si lo admite o lo rechaza.

La decisión de aceptar o rechazar un flujo no se trata simplemente de comparar el ancho de banda, los búferes o los ciclos requeridos por el flujo con la capacidad excedida del enrutador en esas tres dimensiones. Es más complicado que eso. Para empezar, aunque algunas aplicaciones podrían saber sobre sus requerimientos de ancho de banda, saben poco acerca de búferes o ciclos de CPU y, por esto, se necesita por lo menos una forma diferente de describir los flujos. Además, algunas aplicaciones son mucho más tolerantes con el incumplimiento ocasional de plazos que otras. Por último, algunas aplicaciones podrían estar dispuestas a negociar los parámetros del flujo y otras no. Por ejemplo, un visor de películas que por lo general se ejecuta a 30 cuadros/seg podría

estar dispuesto a ejecutar a 25 cuadros/seg si no hay suficiente ancho de banda para soportar 30 cuadros/seg. De manera similar, la cantidad de píxeles por cuadro y de ancho de banda de audio, entre otras propiedades, podría ser ajustable.

Debido a que muchas partes pueden estar involucradas en la negociación del flujo (el emisor, el receptor y todos los enrutadores a lo largo de la ruta), los flujos deben describirse de manera precisa en términos de parámetros específicos que se puedan negociar. Un conjunto de tales parámetros se conoce como **especificación de flujo.** Por lo general, el emisor (por ejemplo, el servidor de vídeo) produce una especificación de flujo que propone los parámetros que le gustaría utilizar. Conforme la especificación se propague por la ruta, cada enrutador la examina y modifica los parámetros conforme sea necesario. Las modificaciones sólo pueden reducir el flujo, no incrementarlo (por ejemplo, una tasa más baja de datos, no una más grande). Cuando llega al otro extremo, se pueden establecer los parámetros.

Como ejemplo de lo que puede estar en una especificación de flujo, considere el de la figura 5-35, que se basa en los RFCs 2210 y 2211. Tiene cinco parámetros, el primero de los cuales, la *Tasa de la cubeta con tokens*, es la cantidad de bytes por segundo que se colocan en la cubeta. Ésta es la tasa máxima que el emisor puede transmitir, promediada con respecto a un intervalo de tiempo largo.

Parámetro	Unidad
Tasa de la cubeta con *tokens*	Bytes/seg
Tamaño de la cubeta con *tokens*	Bytes
Tasa pico de datos	Bytes/seg
Tamaño mínimo de paquete	Bytes
Tamaño máximo de paquete	Bytes

Figura 5-35. Ejemplo de especificación de flujo.

El segundo parámetro es el tamaño de la cubeta en bytes. Por ejemplo, si la *Tasa de la cubeta con tokens* es de 1 Mbps y el *Tamaño de la cubeta con tokens* es de 500 KB, la cubeta se puede llenar de manera continua durante 4 seg antes de llenarse por completo (en caso de que no haya transmisiones). Cualesquier *tokens* enviados después de eso, se pierden.

El tercer parámetro, la *Tasa pico de datos*, es la tasa máxima de transmisiones tolerada, incluso durante intervalos de tiempo breves. El emisor nunca debe sobrepasar esta tasa.

Los últimos dos parámetros especifican los tamaños mínimo y máximo de paquetes, incluyendo los encabezados de la capa de red y de transporte (por ejemplo, TCP e IP). El tamaño mínimo es importante porque procesar cada paquete toma un tiempo fijo, aunque sea breve. Un enrutador debe estar preparado para manejar 10,000 paquetes/seg de 1 KB cada uno, pero no para manejar 100,000 paquetes/seg de 50 bytes cada uno, aunque esto representa una tasa de datos menor. El tamaño máximo de paquete es importante debido a las limitaciones internas de la red que no deben sobrepasarse. Por ejemplo, si parte de la ruta es a través de una Ethernet, el tamaño máximo del paquete se restringirá a no más de 1500 bytes, sin importar lo que el resto de la red puede manejar.

Una pregunta interesante es cómo convierte un enrutador una especificación de flujo en un conjunto de reservaciones de recursos específicos. Esa conversión es específica de la implementación y no está estandarizada. Suponga que un enrutador puede procesar 100,000 paquetes/seg. Si se le ofrece un flujo de 1 MB/seg con tamaños de paquete mínimo y máximo de 512 bytes, el enrutador puede calcular que puede transmitir 2048 paquetes/seg de ese flujo. En ese caso, debe reservar 2% de su CPU para ese flujo, o de preferencia más para evitar retardos largos de encolamiento. Si la política de un enrutador es nunca asignar más de 50% de su CPU (lo que implica un retardo con factor de dos, y ya está lleno el 49%, entonces ese flujo debe rechazarse). Se necesitan cálculos similares para los otros recursos.

Entre más rigurosa sea la especificación de flujo, más útil será para los enrutadores. Si una especificación de flujo especifica que necesita una *tasa de cubeta con tokens* de 5 MB/seg pero los paquetes pueden variar de 50 bytes a 1500 bytes, entonces la tasa de paquetes variará aproximadamente de 3500 a 105,000 paquetes/seg. El enrutador podría asustarse por este último número y rechazar el flujo, mientras que con un tamaño mínimo de paquete de 1000 bytes, el flujo de 5 MB/seg podría aceptarse.

Enrutamiento proporcional

La mayoría de los algoritmos de enrutamiento tratan de encontrar la mejor ruta para cada destino y envían a través de ella todo el tráfico a ese destino. Un método diferente que se ha propuesto para proporcionar una calidad de servicio más alta es dividir el tráfico para cada destino a través de diferentes rutas. Puesto que generalmente los enrutadores no tienen un panorama completo del tráfico de toda la red, la única forma factible de dividir el tráfico a través de múltiples rutas es utilizar la información disponible localmente. Un método simple es dividir el tráfico en fracciones iguales o en proporción a la capacidad de los enlaces salientes. Sin embargo, hay disponibles otros algoritmos más refinados (Nelakuditi y Zhang, 2002).

Calendarización de paquetes

Si un enrutador maneja múltiples flujos, existe el peligro de que un flujo acapare mucha de su capacidad y limite a los otros flujos. El procesamiento de paquetes en el orden de arribo significa que un emisor agresivo puede acaparar la mayor parte de la capacidad de los enrutadores por los que pasan sus paquetes, lo que reduce la calidad del servicio para otros. Para hacer fracasar esos intentos, se han diseñado varios algoritmos de programación de paquetes (Bhatti y Crowcroft, 2000).

Uno de los primeros fue el de **encolamiento justo** (*fair queueing*) (Nagle, 1987). La esencia del algoritmo es que los enrutadores tienen colas separadas para cada línea de salida, una por flujo. Cuando una línea se queda inactiva, el enrutador explora las diferentes colas de manera circular, y toma el primer paquete de la siguiente cola. De esta forma, con *n hosts* compitiendo por una línea de salida dada, cada *host* obtiene la oportunidad de enviar uno de *n* paquetes. El envío de más paquetes no mejorará esta fracción.

Aunque al principio este algoritmo tiene un problema: proporciona más ancho de banda a los *hosts* que utilizan paquetes más grandes que a los que utilizan paquetes más pequeños. Demers y cols. (1990) sugirieron una mejora en la que la exploración circular (*round robin*) se realiza de tal

manera que se simule una exploración circular byte por byte, en lugar de paquete por paquete. En efecto, explora las colas de manera repetida, byte por byte, hasta que encuentra el instante en el que finalizará cada paquete. A continuación, los paquetes se ordenan conforme a su tiempo de terminación y se envían en ese orden. En la figura 5-36 se ilustra este algoritmo.

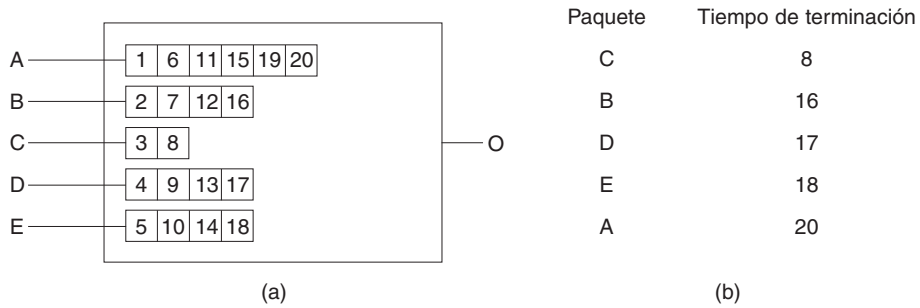


Figura 5-36. (a) Un enrutador con cinco paquetes encolados en la línea *O*. (b) Tiempos de terminación de los cinco paquetes.

En la figura 5-36(a) se muestran paquetes con una longitud de 2 hasta 6 bytes. En el pulso de reloj (virtual) 1, se envía el primer byte del paquete de la línea *A*. Después le toca el turno al primer byte del paquete de la línea *B*, y así sucesivamente. El primer paquete en terminar es *C*, después de ocho pulsos. El orden se muestra en la figura 5-36(b). Debido a que ya no hay más llegadas, los paquete se enviarán en el orden listado, de *C* a *A*.

Un problema con este algoritmo es que da la misma prioridad a todos los *hosts*. En muchas situaciones, es necesario dar a los servidores de vídeo más ancho de banda que a los servidores de archivos regulares, a fin de que puedan proporcionárseles dos o más bytes por pulso. Este algoritmo modificado se conoce como **encolamiento justo ponderado** (*weighted fair queueing*) y se utiliza ampliamente. Algunas veces el peso es igual a la cantidad de flujos provenientes de una máquina, de manera que el proceso obtiene un ancho de banda igual. Una implementación eficiente del algoritmo se analiza en (Shreedhar y Varghese, 1995). El reenvío real de paquetes a través de un enrutador o conmutador se está realizando cada vez más en el hardware (Elhanany y cols., 2001).

5.4.3 Servicios integrados

Entre 1995 y 1997, la IETF se esforzó mucho en diseñar una arquitectura para la multimedia de flujos continuos. Este trabajo resultó en cerca de dos docenas de RFCs, empezando con los RFCs 2205–2210. El nombre genérico para este trabajo es **algoritmos basados en flujo** o **servicios integrados.** Se diseñó tanto para aplicaciones de unidifusión como para multidifusión. Un ejemplo de la primera es un solo usuario difundiendo un clip de vídeo de un sitio de noticias. Un ejemplo del segundo es una colección de estaciones de televisión digital difundiendo sus programas como flujos de paquetes IP a muchos receptores de diferentes ubicaciones. A continuación nos concentraremos en la multidifusión, debido a que la transmisión por unidifusión es un caso especial de multidifusión.

En muchas aplicaciones de multidifusión, los grupos pueden cambiar su membresía de manera dinámica, por ejemplo, conforme las personas entran a una videoconferencia y se aburren y cambian a una telenovela o al canal del juego de croquet. Bajo estas condiciones, el método de hacer que los emisores reserven ancho de banda por adelantado no funciona bien, debido a que requeriría que cada emisor rastreara todas las entradas y salidas de su audiencia. Para un sistema diseñado para transmitir televisión con millones de suscriptores, ni siquiera funcionaría.

RSVP—Protocolo de reservación de recursos

El principal protocolo IETF para la arquitectura de servicios integrados es **RSVP.** Se describe en el RFC 2205 y en otros. Este protocolo se utiliza para marcar las reservas; para el envío de datos se utilizan otros protocolos. RSVP permite que varios emisores transmitan a múltiples grupos de receptores, permite que receptores individuales cambien de canal libremente, optimiza el uso de ancho de banda y elimina la congestión.

En su forma más sencilla, el protocolo usa enrutamiento de multidifusión con árboles de expansión, como se vio antes. A cada grupo se le asigna un grupo de direcciones. Para enviar a un grupo, un emisor pone la dirección del grupo en sus paquetes. El algoritmo estándar de multidifusión construye entonces un árbol de expansión que cubre a todos los miembros del grupo. El algoritmo de enrutamiento no es parte del RSVP. La única diferencia con la multidifusión normal es un poco de información extra multidifundida al grupo periódicamente para indicarle a los enrutadores a lo largo del árbol que mantengan ciertas estructuras de datos en sus memorias.

Como ejemplo, considere la red de la figura 5-37(a). Los *hosts* 1 y 2 son emisores multidifusión, y los *hosts* 3, 4 y 5 son receptores multidifusión. En este ejemplo, los emisores y los receptores son distintos pero, en general, los dos grupos pueden traslaparse. Los árboles de multidifusión de los *hosts* 1 y 2 se muestran en las figuras 5-37(b) y 5-37(c), respectivamente.

Para obtener mejor recepción y eliminar la congestión, cualquiera de los receptores de un grupo puede enviar un mensaje de reservación por el árbol al emisor. El mensaje se propaga usando el algoritmo de reenvío por ruta invertida estudiado antes. En cada salto, el enrutador nota la reservación y aparta el ancho de banda necesario; si no hay suficiente ancho de banda disponible, informa de una falla. En el momento que el mensaje llega de regreso al origen, se ha reservado el ancho de banda desde el emisor hasta el receptor que hace la solicitud de reservación a lo largo del árbol de expansión.

En la figura 5-38(a) se muestra un ejemplo de tales reservaciones. Aquí el *host* 3 ha solicitado un canal al *host* 1. Una vez establecido el canal, los paquetes pueden fluir de 1 a 3 sin congestiones. Ahora considere lo que sucede si el *host* 3 reserva a continuación un canal hacia otro emisor, el *host* 2, para que el usuario pueda ver dos programas de televisión a la vez. Se reserva una segunda ruta, como se muestra en la figura 5-38(b). Observe que se requieren dos canales individuales del *host* 3 al enrutador *E*, porque se están transmitiendo dos flujos independientes.

Por último, en la figura 5-38(c), el *host* 5 decide observar el programa transmitido por el *host* 1 y también hace una reservación. Primero se reserva ancho de banda dedicado hasta el enrutador *H*.

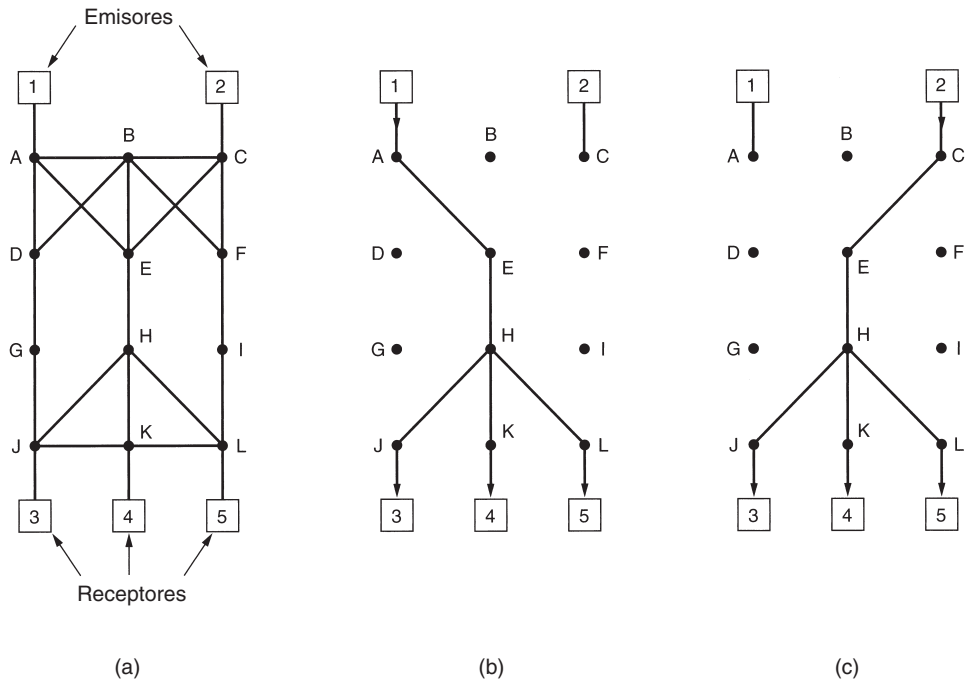


Figura 5-37. (a) Red. (b) Árbol de expansión de multidifusión para el *host* 1. (c) Árbol de expansión de multidifusión para el *host* 2.

Sin embargo, éste ve que ya tiene una alimentación del *host* 1, por lo que, si ya se ha reservado el ancho de banda necesario, no necesita reservar nuevamente. Observe que los *hosts* 3 y 5 podrían haber solicitado diferentes cantidades de ancho de banda (por ejemplo, el 3 tiene una televisión de blanco y negro, por lo que no quiere la información de color), así que la capacidad reservada debe ser lo bastante grande para satisfacer al receptor más voraz.

Al hacer una reservación, un receptor puede especificar (opcionalmente) uno o más orígenes de los que quiere recibir. También puede especificar si estas selecciones quedarán fijas durante toda la reservación, o si el receptor quiere mantener abierta la opción de cambiar los orígenes después. Los enrutadores usan esta información para optimizar la planeación del ancho de banda. En particular, sólo se establece que dos receptores van a compartir una ruta si ambos están de acuerdo en no cambiar los orígenes posteriormente.

La razón de esta estrategia en el caso totalmente dinámico es que el ancho de banda reservado está desacoplado de la selección del origen. Una vez que un receptor ha reservado ancho de banda, puede conmutarse a otro origen y conservar la parte de la ruta existente que es válida para el nuevo origen. Por ejemplo, si el *host* 2 está transmitiendo varios flujos de vídeo, el *host* 3 puede conmutarse entre ellos a voluntad sin cambiar su reservación: a los enrutadores no les importa el programa que está viendo el receptor.

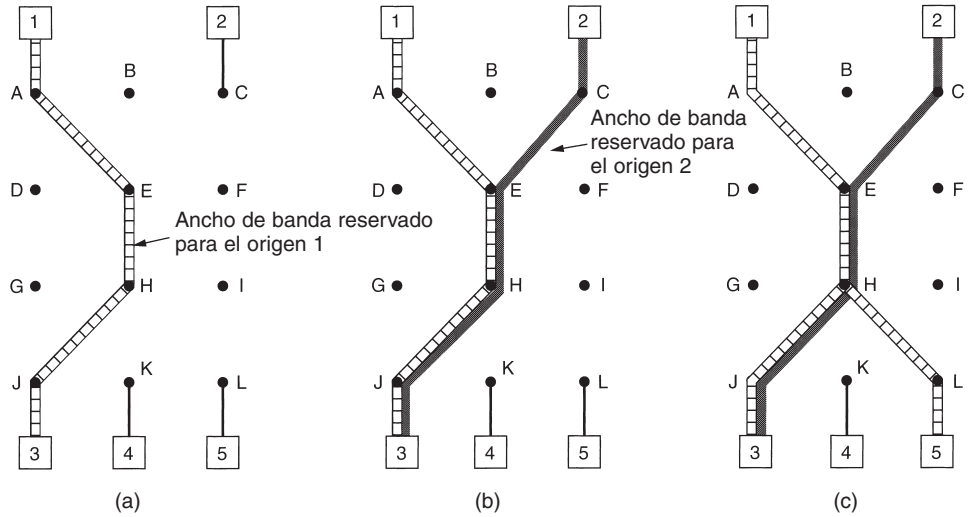


Figura 5-38. (a) El *host* 3 solicita un canal al *host* 1. (b) El *host* 3 solicita entonces un segundo canal al *host* 2. (c) El *host* 5 solicita un canal al *host* 1.

5.4.4 Servicios diferenciados

Los algoritmos basados en flujo tienen el potencial de ofrecer buena calidad de servicio a uno o más flujos debido a que reservan los recursos que son necesarios a lo largo de la ruta. Sin embargo, también tienen una desventaja. Requieren una configuración avanzada para establecer cada flujo, algo que no se escala bien cuando hay miles o millones de flujos. Además, mantienen estado por flujo interno en los enrutadores, haciéndolos vulnerables a las caídas de enrutadores. Por último, los cambios requeridos al código de enrutador son sustanciales e involucran intercambios complejos de enrutador a enrutador para establecer los flujos. Como consecuencia, existen pocas implementaciones de RSVP o algo parecido.

Por estas razones, la IETF también ha diseñado un método más simple para la calidad del servicio, uno que puede implementarse ampliamente de manera local en cada enrutador sin una configuración avanzada y sin que toda la ruta esté involucrada. Este método se conoce como calidad de servicio **basada en clase** (contraria a basada en flujo). La IETF ha estandarizado una arquitectura para él, llamada **servicios diferenciados,** que se describe en los RFCs 2474, 2475, entre otros. A continuación lo describiremos.

Un conjunto de enrutadores que forman un dominio administrativo (por ejemplo, un ISP o una compañía telefónica) puede ofrecer los servicios diferenciados (DS). La administración define un conjunto de clases de servicios con reglas de reenvío correspondientes. Si un cliente firma para un DS, los paquetes del cliente que entran en el dominio podrían contener un campo *Tipo de servicio*, con un mejor servicio proporcionado a algunas clases (por ejemplo, un servicio premium) que a otras. Al tráfico dentro de una clase se le podría requerir que se apegue a algún modelo específico, como a una cubeta con goteo con una tasa especificada de drenado. Un operador con intuición para los negocios

podría cargar una cantidad extra por cada paquete premium transportado o podría permitir hasta *N* paquetes premium por una mensualidad adicional fija. Observe que este esquema no requiere una configuración avanzada, ni reserva de recursos ni negociación extremo a extremo que consuma tiempo para cada flujo, como sucede con los servicios integrados. Esto hace de DS relativamente fácil de implementar.

El servicio basado en clase también ocurre en otras industrias. Por ejemplo, las compañías de envío de paquetes con frecuencia ofrecen servicio de tres días, de dos días, y servicio de un día para otro. Las aerolíneas ofrecen servicio de primera clase, de clase de negocios y de tercera clase. Los trenes que recorren largas distancias con frecuencia tienen múltiples clases de servicios. Incluso el metro de París tiene dos clases de servicios. Para los paquetes, las clases pueden diferir en términos de retardo, fluctuación y probabilidad de ser descartado en caso de congestión, entre otras posibilidades (pero probablemente sin tramas Ethernet más amplias).

Para hacer que la diferencia entre la calidad basada en el servicio y la basada en clase de servicio sea más clara, considere un ejemplo: la telefonía de Internet. Con un esquema basado en flujo, cada llamada telefónica obtiene sus propios recursos y garantías. Con un esquema basado en clase, todas las llamadas telefónicas obtienen los recursos reservados para la telefonía de clase. Estos recursos no pueden ser tomados por paquetes de la clase de transferencia de archivos u otras clases, pero ninguna llamada telefónica obtiene ningún recurso privado reservado sólo para ella.

Reenvío expedito o acelerado

Cada operador debe realizar la selección de clases de servicios, pero debido a que los paquetes con frecuencia se reenvían entre subredes ejecutadas por diferentes operadores, la IETF está trabajando para definir las clases de servicios independientes de la red. La clase más simple es el **reenvío expedito,** por lo tanto, iniciemos con ella. Se describe en el RFC 3246.

La idea detrás del reenvío expedito es muy simple. Dos clases de servicios están disponibles: regular y expedita. Se espera que la mayor parte del tráfico sea regular, pero una pequeña fracción de los paquetes son expeditos. Los paquetes expeditos deben tener la capacidad de transitar la subred como si no hubieran otros paquetes. En la figura 5-39 se muestra una representación simbólica de este sistema de "dos tubos". Observe que todavía hay una línea física. Los dos conductos lógicos que se muestran en la figura representan una forma de reservar ancho de banda, no una segunda línea física.

Una forma de implementar esta estrategia es programar los enrutadores para que tengan dos colas de salida por cada línea de salida, una para los paquetes expeditos y una para los regulares. Cuando llega un paquete, se coloca en la cola de manera acorde. La programación de paquetes debe utilizar algo parecido al encolamiento justo ponderado. Por ejemplo, si 10% del tráfico es expedito y 90% es regular, 20% del ancho de banda podría dedicarse al tráfico expedito y el resto al tráfico regular. Al hacer esto se daría al tráfico expedito dos veces más ancho de banda del que necesita a fin de que dicho tráfico tenga un retardo bajo. Esta asignación se puede alcanzar transmitiendo un paquete expedito por cada cuatro paquetes regulares (suponiendo que el tamaño de la distribución para ambas clases es similar). De esta forma, se espera que los paquetes expeditos vean una red descargada, incluso cuando hay, de hecho, una carga pesada.

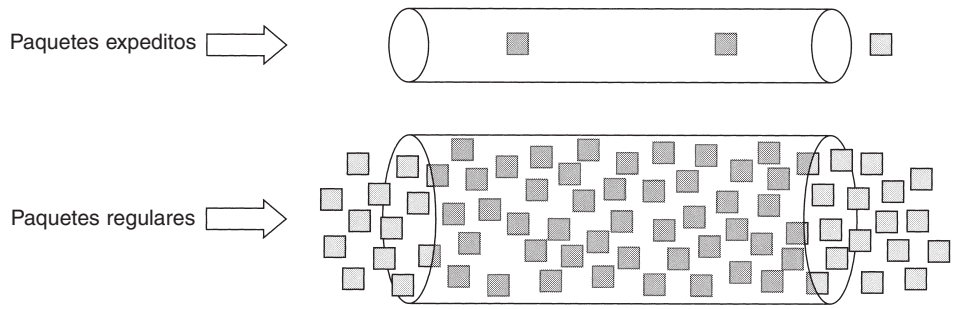


Figura 5-39. Los paquetes expeditos viajan por una red libre de tráfico.

Reenvío asegurado

Un esquema un poco más elaborado para el manejo de las clases de servicios se conoce como **reenvío asegurado.** Se describe en el RFC 2597. Especifica que deberán haber cuatro clases de prioridades, y cada una tendrá sus propios recursos. Además, define tres probabilidades de descarte para paquetes que están en congestión: baja, media y alta. En conjunto, estos dos factores definen 12 clases de servicios.

La figura 5-40 muestra una forma en que los paquetes pueden ser procesados bajo reenvío asegurado. El paso 1 es clasificar los paquetes en una de cuatro clases de prioridades. Este paso podría realizarse en el *host* emisor (como se muestra en la figura) o en el enrutador de ingreso. La ventaja de realizar la clasificación en el *host* emisor es que hay más información disponible acerca de cuáles paquetes pertenecen a qué flujos.

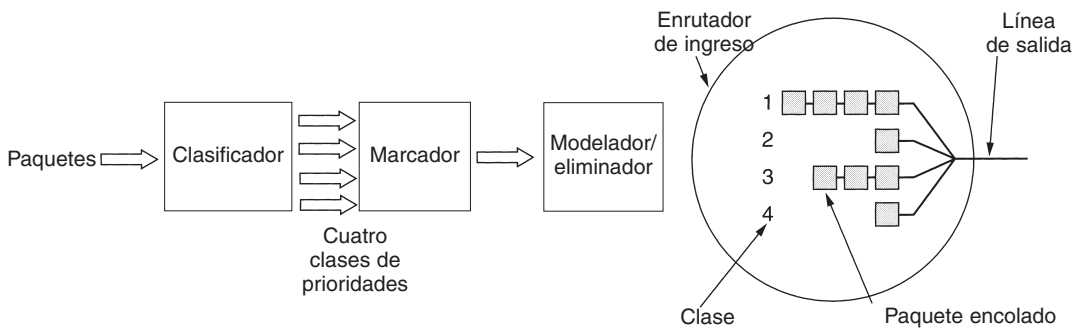


Figura 5-40. Una posible implementación del flujo de datos para el reenvío asegurado.

El paso 2 es marcar los paquetes de acuerdo con su clase. Para este propósito se necesita un campo de encabezado. Por fortuna, en el encabezado IP está disponible un campo *Tipo de servicio* de 8 bits, como veremos un poco más adelante. El RFC 2597 especifica que seis de estos bits se van a utilizar para la clase de servicio, dejando espacio de codificación para clases de servicio históricas y para futuras.

El paso 3 es pasar los paquetes a través de un filtro modelador/eliminador que podría retardar o descartar algunos de ellos para dar una forma aceptable a los cuatro flujos, por ejemplo, mediante cubetas con goteo o con *tokens*. Si hay muchos paquetes, algunos de ellos podrían descartarse aquí, mediante una categoría de eliminación. También son posibles esquemas elaborados que involucren la medición o la retroalimentación.

En este ejemplo, estos tres pasos se realizan en el *host* emisor, por lo que el flujo de salida ahora se introduce en el enrutador de ingreso. Vale la pena mencionar que estos pasos pueden ser realizados por software especial de conectividad de redes o incluso por el sistema operativo, a fin de no tener que cambiar las aplicaciones existentes.

5.4.5 Conmutación de etiquetas y MPLS

Mientras la IETF estaba desarrollando servicios integrados y diferenciados, varios fabricantes de enrutadores estaban desarrollando mejores métodos de reenvío. Este trabajo se enfocó en agregar una etiqueta en frente de cada paquete y realizar el enrutamiento con base en ella y no con base en la dirección de destino. Hacer que la etiqueta sea un índice de una tabla provoca que encontrar la línea correcta de salida sea una simple cuestión de buscar en una tabla. Al utilizar esta técnica, el enrutamiento puede llevarse a cabo de manera muy rápida y cualesquier recursos necesarios pueden reservarse a lo largo de la ruta.

Por supuesto, etiquetar los flujos de esta manera se acerca peligrosamente a los circuitos virtuales. X.25, ATM, frame relay, y otras redes con una subred de circuitos virtuales colocan una etiqueta (es decir, un identificador de circuitos virtuales) en cada paquete, la buscan en una tabla y enrutan con base en la entrada de la tabla. A pesar del hecho de que muchas personas en la comunidad de Internet tienen una aversión intensa por las redes orientadas a la conexión, la idea parece surgir nuevamente, pero esta vez para proporcionar un enrutamiento rápido y calidad de servicio. Sin embargo, hay diferencias esenciales entre la forma en que Internet maneja la construcción de la ruta y la forma en que lo hacen las redes orientadas a la conexión, por lo que esta técnica no utiliza la conmutación de circuitos tradicional.

Esta "nueva" idea de conmutación ha pasado por varios nombres (propietarios), entre ellos **conmutación de etiquetas.** En algún momento, la IETF comenzó a estandarizar la idea bajo el nombre **MPLS** (**conmutación de etiquetas multiprotocolo**). De aquí en adelante lo llamaremos MPLS. Se describe en el RFC 3031, entre muchos otros.

Además, algunas personas hacen una distinción entre *enrutamiento* y *conmutación*. El enrutamiento es el proceso de buscar una dirección de destino en una tabla para saber a dónde enviar los paquetes hacia ese destino. En contraste, la conmutación utiliza una etiqueta que se toma de un paquete como un índice en una tabla de reenvío. Sin embargo, estas definiciones están lejos de ser universales.

El primer problema es en dónde colocar la etiqueta. Debido a que los paquetes IP no fueron diseñados para circuitos virtuales, en el encabezado IP no hay ningún campo disponible para los números de tales circuitos. Por esta razón, se tuvo que agregar un nuevo encabezado MPLS enfrente del encabezado IP. En una línea de enrutador a enrutador que utiliza PPP como protocolo

de tramas, el formato de trama, incluyendo los encabezados PPP, MPLS, IP y TCP, es como se muestra en la figura 5-41. De cierta forma, MPLS es, por lo tanto, la capa 2.5.

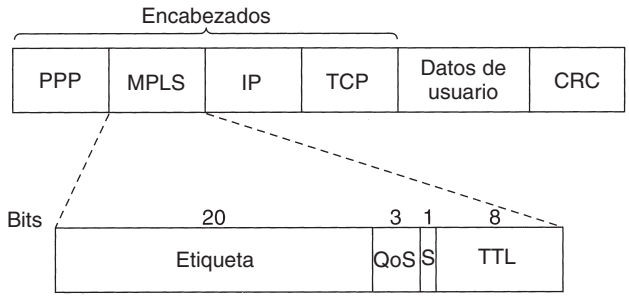


Figura 5-41. Transmisión de un segmento TCP que utiliza IP, MPLS y PPP.

El encabezado MPLS genérico tiene cuatro campos, el más importante de los cuales es el de *Etiqueta*, el cual contiene el índice. El campo *QoS (bits experimentales)* indica la clase de servicio. El campo *S* se relaciona con colocar en una pila múltiples etiquetas en redes jerárquicas (que se analizan más adelante). Si tiene el valor de 1 indica que es la última etiqueta añadida al paquete IP, si es un 0 indica que hay más etiquetas añadidas al paquete. El campo evita el ciclo infinito en caso de que haya inestabilidad en el enrutamiento, ya que se decrementa en cada enrutador y al llegar al valor de 0, el paquete es descartado.

Debido a que los encabezados MPLS no son parte del paquete de la capa de red o de la trama del enlace de datos, MPLS es en gran medida independiente de ambas capas. Entre otras cosas, esta propiedad significa que es posible construir conmutadores MPLS que pueden reenviar tanto paquetes IP como celdas ATM, dependiendo de lo que aparezca. De esta característica proviene la parte "multiprotocolo" del nombre MPLS.

Cuando un paquete mejorado con MPLS (o celda) llega a un enrutador con capacidad MPLS, la etiqueta se utiliza como un índice en una tabla para determinar la línea de salida y la nueva etiqueta a utilizar. Esta conmutación de etiquetas se utiliza en todas las subredes de circuitos virtuales, debido a que las etiquetas sólo tienen importancia local y dos enrutadores diferentes pueden asignar la misma etiqueta a paquetes hacia diferentes destinos, es decir, la etiqueta es reasignada a la salida de cada enrutador, por lo que no se mantiene la misma etiqueta en toda la ruta. En la figura 5-3 vimos en acción este mecanismo. MPLS utiliza la misma técnica.

Una diferencia con respecto a los circuitos virtuales tradicionales es el nivel de agregación. Ciertamente es posible que cada flujo tenga su propio conjunto de etiquetas a través de la subred. Sin embargo, es más común que los enrutadores agrupen múltiples flujos que terminan en un enrutador o una LAN particulares y utilizan una sola etiqueta de ellos. Se dice que los flujos que están agrupados en una sola etiqueta pertenecen a la misma **FEC** (**clase de equivalencia de reenvío**). Esta clase cubre no sólo a dónde van los paquetes, sino también su clase de servicio (en el sentido de los servicios diferenciados), debido a que todos sus paquetes se tratan de la misma forma para propósitos de reenvío.

Con el enrutamiento de circuitos virtuales tradicional no es posible agrupar en el mismo identificador de circuitos virtuales varias rutas diferentes con diferentes puntos finales, debido a que podría no haber forma de distinguirlas en el destino final. Con MPLS, los paquetes aún contienen su dirección de destino final, además de la etiqueta, a fin de que al final de la red de MPLS pueda eliminarse la etiqueta y que el reenvío pueda continuar de la forma normal, utilizando la dirección de destino de la capa de red.

Una diferencia principal entre MPLS y los diseños de circuitos virtuales convencionales es la forma en que está construida la tabla de reenvío. En las redes de circuitos virtuales tradicionales, cuando un usuario desea establecer una conexión, se inicia un paquete de configuración en la red para crear la ruta y crear las entradas de la tabla de reenvío. MPLS no funciona de esa forma porque no hay fase de configuración para cada conexión (pues eso podría romper con la operación de mucho software existente en Internet).

En su lugar, hay dos formas de crear las entradas de la tabla de reenvío. En el método **orientado a datos,** cuando un paquete llega, el primer enrutador que encuentra contacta al siguiente enrutador en el sentido descendente del flujo a donde tiene que ir el paquete y le pide que genere una etiqueta para el flujo. Este método se aplica de manera recursiva. En efecto, ésta es una creación de circuitos virtuales por petición.

Los protocolos que hacen esta propagación son muy cuidadosos para evitar los ciclos cerrados (loops). Por lo general, utilizan una técnica llamada **subprocesos con color** (*colored threads*). La propagación en reversa de una FEC se puede comparar con extraer un subproceso de un color único en la subred. Si un enrutador ve un color que ya tiene, sabe que hay un ciclo y toma una medida para solucionarlo. El método dirigido por datos se utiliza principalmente en redes en las que el transporte subyacente es ATM (como sucede en la mayor parte del sistema telefónico).

La otra forma, que se utiliza en las redes que no se basan en ATM, es el método **dirigido por control.** Tiene algunas variantes. Una de ellas funciona de la siguiente manera. Cuando se inicia un enrutador, verifica para cuáles rutas es el último salto (por ejemplo, qué *hosts* están en su LAN). Después crea una o más FECs para ellas, asigna una etiqueta para cada una y pasa las etiquetas a sus vecinos. Éstos, a su vez, introducen las etiquetas en sus tablas de reenvío y envían nuevas etiquetas a sus vecinos, hasta que todos los enrutadores han adquirido la ruta. También es posible reservar recursos conforme la ruta está construida para garantizar una calidad de servicio apropiada.

MPLS puede operar a múltiples niveles al mismo tiempo. En el nivel más alto, cada empresa portadora puede considerarse como un tipo de metaenrutador, con una ruta a través de los metaenrutadores del origen al destino. Esta ruta puede utilizar MPLS. Sin embargo, MPLS también puede utilizarse dentro de la red de cada empresa portadora, lo que resulta en un segundo nivel de etiquetado. De hecho, un paquete puede llevar consigo una pila entera de etiquetas. El bit *S* de la figura 5-41 permite que un enrutador elimine una etiqueta para saber si quedaron etiquetas adicionales. Se establece a 1 para la etiqueta inferior y 0 para las otras etiquetas. En la práctica, esta característica se utiliza principalmente para implementar redes privadas virtuales y túneles recursivos.

Aunque las ideas básicas detrás de MPLS son directas, los detalles son extremadamente complicados, y tienen muchas variaciones y optimizaciones, por lo que ya no trataremos más ese tema. Para mayor información, vea (Davie y Rekhter, 2000; Lin y cols., 2002; Pepelnjak y Guichard, 2001, y Wang, 2001).

5.5 INTERCONECTIVIDAD

Hasta ahora hemos supuesto de manera implícita que hay una sola red homogénea y que cada máquina usa el mismo protocolo en cada capa. Por desgracia, este supuesto es demasiado optimista. Existen muchas redes diferentes, entre ellas LANs, MANs y WANs. En cada capa hay numerosos protocolos de uso muy difundido. En las siguientes secciones estudiaremos con cuidado los problemas que surgen cuando dos o más redes se juntan, formando una **interred.**

Existe una controversia considerable sobre si la abundancia actual de tipos de red es una condición temporal que desaparecerá tan pronto como todo mundo se dé cuenta de lo maravillosa que es [indique aquí su red favorita], o si es una característica inevitable pero permanente del mundo, que está aquí para quedarse. Tener diferentes redes invariablemente implica tener diferentes protocolos.

Creemos que siempre habrá una variedad de redes (y, por lo tanto, de protocolos) diferentes, por las siguientes razones. Antes que nada, la base instalada de redes diferentes es grande. Casi todas las instalaciones UNIX ejecutan TCP/IP. Muchos negocios grandes aún tienen *mainframes* que ejecutan SNA de IBM. Una cantidad considerable de compañías telefónicas operan redes ATM. Algunas LANs de computadoras personales aún usan Novell NCP/IPX o AppleTalk. Por último, las redes inalámbricas constituyen un área nueva y en desarrollo con una variedad de protocolos. Esta tendencia continuará por años, debido a problemas de herencia, tecnología nueva y al hecho de que no todos los fabricantes se interesan en que sus clientes puedan migrar fácilmente al sistema de otro fabricante.

Segundo, a medida que las computadoras y las redes se vuelven más baratas, el lugar de la toma de decisiones se desplaza hacia abajo. Muchas compañías tienen políticas en el sentido de que las compras de más de un millón de dólares tienen que ser aprobadas por la gerencia general, las compras de más de 100,000 dólares tienen que ser aprobadas por la gerencia media, pero las compras por debajo de 100,000 dólares pueden ser hechas por los jefes de los departamentos sin aprobación de los superiores. Esto puede dar como resultado fácilmente a que el departamento de ingeniería instale estaciones de trabajo de UNIX que ejecuten TCP/IP y a que el departamento de marketing instale Macs con AppleTalk.

Tercero, las distintas redes (por ejemplo, ATM e inalámbricas) tienen tecnologías radicalmente diferentes, por lo que no debe sorprendernos que, a medida que haya avances nuevos en hardware, también se cree software nuevo adaptado al nuevo hardware. Por ejemplo, la casa típica ahora es como la oficina típica de hace 10 años: está llena de computadoras que no se hablan entre ellas. En el futuro, podría ser común que el teléfono, la televisión y otros aparatos estuvieran en red, para controlarlos de manera remota. Esta nueva tecnología sin duda generará nuevos protocolos y redes.

Como ejemplo de cómo se pueden conectar redes diferentes, considere la figura 5-42. Ahí vemos una red corporativa con múltiples ubicaciones enlazadas por una red ATM de área amplia. En una de las ubicaciones, se utiliza una red dorsal óptica FDDI para conectar una Ethernet, una LAN inalámbrica 802.11 y la red de *mainframe* SNA del centro de datos corporativo.

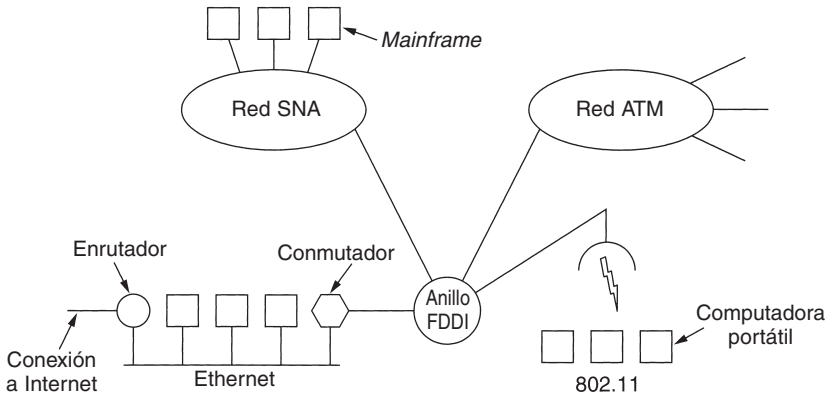


Figura 5-42. Una colección de redes interconectadas.

El propósito de interconectar todas estas redes es permitir que los usuarios de cualquiera de ellas se comuniquen con los usuarios de las demás, así como permitir que los usuarios de cualquiera de ellas accedan los datos de las demás. Lograr este objetivo significa enviar paquetes de una red a otra. Debido a que las redes, por lo general, difieren de formas considerables, obtener paquetes de una red a otra no siempre es tan fácil, como veremos a continuación.

5.5.1 Cómo difieren las redes

Las redes pueden diferir de muchas maneras. Algunas de las diferencias, como técnicas de modulación o formatos de tramas diferentes, se encuentran en las capas de enlace de datos y en la física. No trataremos esas diferencias aquí. En su lugar, en la figura 5-43 listamos algunas diferencias que pueden ocurrir en la capa de red. La conciliación de estas diferencias es lo que hace más difícil la interconexión de redes que la operación con una sola red.

Cuando los paquetes enviados por un origen en una red deben transitar a través de una o más redes foráneas antes de llegar a la red de destino (que también puede ser diferente de la red de origen), pueden ocurrir muchos problemas en las interfaces entre las redes. Para comenzar, cuando los paquetes de una red orientada a la conexión deben transitar a una red sin conexiones, deben reordenarse, algo que el emisor no espera y que el receptor no está preparado para manejar. Con frecuencia se necesitarán conversiones de protocolo, que pueden ser difíciles si la funcionalidad requerida no puede expresarse. También se necesitarán conversiones de direcciones, lo que podría requerir algún tipo de sistema de directorio. El paso de paquetes multidifusión a través de una red que no reconoce la multidifusión requiere la generación de paquetes individuales para cada destino.

Los diferentes tamaños máximos de paquete usados por las diferentes redes son un dolor de cabeza importante. ¿Cómo se pasa un paquete de 8000 bytes a través de una red cuyo tamaño máximo de paquete es de 1500 bytes? Es importante la diferencia en la calidad de servicio cuando

Aspecto	Algunas posibilidades
Servicio ofrecido	Sin conexiones, orientado a conexiones
Protocolos	IP, IPX, SNA, ATM, MPLS, AppleTalk, etc.
Direccionamiento	Plano (802) o jerárquico (IP)
Multidifusión	Presente o ausente (también difusión)
Tamaño de paquete	Cada red tiene su propio máximo
Calidad del servicio	Puede estar presente o ausente; muchos tipos diferentes
Manejo de errores	Entrega confiable, ordenada y desordenada
Control de flujo	Ventana corrediza, control de tasa, otros o ninguno
Control de congestión	Cubeta con goteo, paquetes reguladores, etc.
Seguridad	Reglas de confidencialidad, encriptación, etc.
Parámetros	Diferentes terminaciones de temporizador, especificaciones de flujo, etc.
Contabilidad	Por tiempo de conexión, por paquete, por byte, o sin ella

Figura 5-43. Algunas de las muchas maneras en que pueden diferir las redes.

un paquete que tiene restricciones de tiempo real pasa a través de una red que no ofrece garantías de tiempo real.

El control de errores, de flujo y de congestión suele ser diferente entre las diferentes redes. Si tanto el origen como el destino esperan la entrega de paquetes en secuencia y sin errores, pero una red intermedia simplemente descarta paquetes cuando huele congestión en el horizonte, o los paquetes vagan sin sentido durante un rato y emergen repentinamente para ser entregados, muchas aplicaciones fallarán. Existen diferentes mecanismos de seguridad, ajustes de parámetros y reglas de contabilidad, e incluso leyes de confidencialidad internacionales, que pueden causar problemas.

5.5.2 Conexión de redes

Las redes pueden interconectarse mediante diversos dispositivos, como vimos en el capítulo 4. Revisemos brevemente ese material. En la capa física, las redes se pueden conectar mediante repetidores o concentradores, los cuales mueven los bits de una red a otra idéntica. Éstos son en su mayoría dispositivos analógicos y no comprenden nada sobre protocolos digitales (simplemente regeneran señales).

En la capa de enlace de datos encontramos puentes y conmutadores. Pueden aceptar tramas, examinar las direcciones MAC y reenviar las tramas a una red diferente mientras realizan una traducción menor de protocolos en el proceso, por ejemplo, de Ethernet a FDDI o a 802.11.

En la capa de red hay enrutadores que pueden conectar dos redes. Si éstas tienen capas de red diferentes, el enrutador puede tener la capacidad de traducir entre los formatos de paquetes, aunque la traducción de paquetes ahora es cada vez menos común. Un enrutador que puede manejar múltiples protocolos se conoce como **enrutador multiprotocolo.**

En la capa de transporte se encuentran puertas de enlace de transporte, que pueden interactuar entre dos conexiones de transporte. Por ejemplo, una puerta de enlace de transporte podría permitir que los paquetes fluyeran entre una red TCP y una SNA, las cuales tienen protocolos de transporte diferentes, fijando esencialmente una conexión TCP con una SNA.

Por último, en la capa de aplicación, las puertas de enlace de aplicación traducen semánticas de mensaje. Como ejemplo, las puertas de enlace entre el correo electrónico de Internet (RFC 822) y el correo electrónico X.400 deben analizar los mensajes de correo electrónico y cambiar varios campos de encabezado.

En este capítulo nos enfocaremos en la interconectividad en la capa de red. Para ver cómo difiere esto de la conmutación en la capa de enlace de datos, examine la figura 5-44. En la figura 5-44(a), la máquina de origen, *S*, desea enviar un paquete a la máquina de destino, *D*. Estas máquinas se encuentran en Ethernets diferentes, conectadas mediante un conmutador. *S* encapsula el paquete en una trama y lo envía a su destino. La trama llega al conmutador, el cual ve la dirección MAC de dicha trama y determina que ésta tiene que ir a la LAN 2. El conmutador elimina la trama de la LAN 1 y la coloca en la LAN 2.

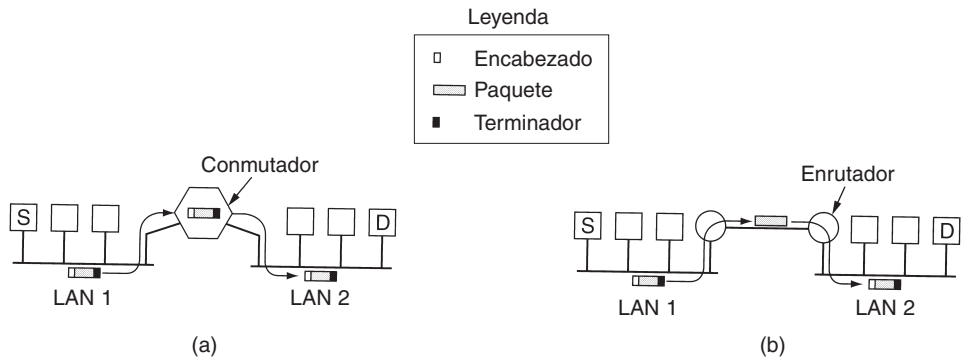


Figura 5-44. (a) Dos Ethernets conectadas mediante un conmutador. (b) Dos Ethernets conectadas mediante enrutadores.

Ahora consideremos la misma situación pero con las dos Ethernets conectadas mediante un par de enrutadores en lugar de un conmutador. Los enrutadores se conectan mediante una línea punto a punto, posiblemente una línea rentada de miles de kilómetros de longitud. Ahora el enrutador recoge la trama y el paquete se elimina del campo de datos de dicha trama. El enrutador examina la dirección del paquete (por ejemplo, una dirección IP) y la busca en su tabla de enrutamiento. Con base en esta dirección, decide enviar el paquete al enrutador remoto, encapsulado en un tipo diferente de trama, dependiendo del protocolo de línea. En el otro extremo el paquete se coloca en el campo de datos de una trama Ethernet y se deposita en la LAN 2.

Lo anterior es la diferencia esencial entre el caso de conmutación (o puenteo) y el caso enrutado. Con un conmutador (o puente), toda la trama se transporta con base en su dirección MAC. Con un enrutador, el paquete se extrae de la trama y la dirección del paquete se utiliza para decidir

a dónde enviarlo. Los conmutadores no tienen que entender el protocolo de capa de red que se está utilizando para conmutar los paquetes. Los enrutadores sí tienen que hacerlo.

5.5.3 Circuitos virtuales concatenados

Dos estilos posibles de interconectividad: la concatenación orientada a la conexión de subredes de circuitos virtuales, y los datagramas estilo Internet. A continuación los examinaremos por separado, pero primero una advertencia. En el pasado, la mayoría de las redes (públicas) eran orientadas a la conexión (frame relay, SNA, 802.16 y ATM aún lo son). Posteriormente, con la aceptación rápida de Internet, los datagramas se pusieron de moda. Sin embargo, sería un error pensar que los datagramas son para siempre. En este negocio, lo único que es para siempre es el cambio. Con la importancia creciente de las redes de multimedia, es probable que la orientación a la conexión regrese de una forma o de otra, puesto que es más fácil garantizar la calidad de servicio con conexiones que sin ellas. Por lo tanto, dedicaremos algún tiempo para estudiar las redes orientadas a la conexión.

En el modelo de circuitos virtuales concatenados, que se muestra en la figura 5-45, se establece una conexión con un *host* de una red distante de un modo parecido a la manera en que se establecen normalmente las conexiones. La subred ve que el destino es remoto y construye un circuito virtual al enrutador más cercano a la red de destino; luego construye un circuito virtual de ese enrutador a una **puerta de enlace** externa (enrutador multiprotocolo). Ésta registra la existencia del circuito virtual en sus tablas y procede a construir otro circuito virtual a un enrutador de la siguiente subred. Este proceso continúa hasta llegar al *host* de destino.

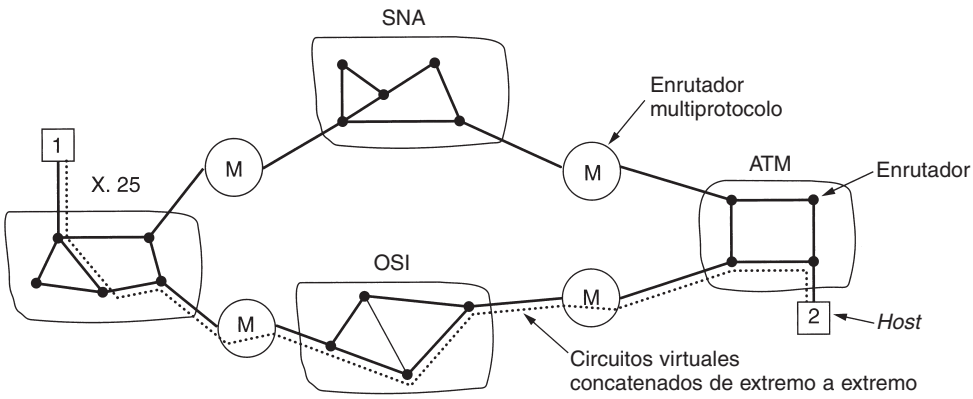


Figura 5-45. Interconectividad mediante circuitos virtuales concatenados.

Una vez que comienzan a fluir paquetes de datos por la ruta, cada puerta de enlace retransmite los paquetes de entrada y hace las conversiones entre los formatos de paquete y los números de circuito virtual, según sea necesario. Obviamente, todos los paquetes de datos deben atravesar la misma secuencia de puertas de enlace. En consecuencia, la red nunca reordena los paquetes de un flujo.

La característica esencial de este enfoque es que se establece una secuencia de circuitos virtuales desde el origen, a través de una o más puertas de enlace, hasta el destino. Cada puerta de enlace mantiene tablas que indican los circuitos virtuales que pasan a través suyo, a dónde se deben enrutar y el nuevo número de circuito virtual.

Este esquema funciona mejor cuando todas las redes tienen aproximadamente las mismas propiedades. Por ejemplo, si todas garantizan la entrega confiable de paquetes de capa de red, entonces, salvo una caída a lo largo de la ruta, el flujo del origen al destino también será confiable. De igual modo, si ninguna de ellas garantiza la entrega confiable, entonces la concatenación de los circuitos virtuales tampoco será confiable. Por otra parte, si la máquina de origen está en una red que garantiza la entrega confiable, pero una de las redes intermedias puede perder paquetes, la concatenación habrá cambiado fundamentalmente la naturaleza del servicio.

Los circuitos virtuales concatenados también son comunes en la capa de transporte. En particular, es posible construir un conducto de bits usando, digamos, SNA, que termine en una puerta de enlace, y luego tener una conexión TCP de esa puerta de enlace a la siguiente. De este modo, puede construirse un circuito virtual de extremo a extremo que abarque diferentes redes y protocolos.

5.5.4 Interconectividad no orientada a la conexión

El modelo alterno de interred es el modelo de datagramas, mostrado en la figura 5-46. En este modelo, el único servicio que ofrece la capa de red a la capa de transporte es la capacidad de inyectar datagramas en la subred y esperar que todo funcione bien. En la capa de red no hay noción en lo absoluto de un circuito virtual, y mucho menos de una concatenación de éstos. Este modelo no requiere que todos los paquetes que pertenecen a una conexión atraviesen la misma secuencia de puertas de enlace. En la figura 5-46 los datagramas del *host* 1 al *host* 2 toman diferentes rutas a través de la interred. Para cada paquete se toma una decisión de enrutamiento independiente, posiblemente dependiendo del tráfico en el momento del envío de dicho paquete. Esta estrategia puede utilizar múltiples rutas y lograr de esta manera un ancho de banda mayor que el modelo de circuitos virtuales concatenados. Por otra parte, no hay garantía de que los paquetes llegarán al destino en orden, suponiendo que lleguen.

El modelo de la figura 5-46 no es tan sencillo como parece. Por una parte, si cada red tiene su propio protocolo de capa de red, no es posible que un paquete de una red transite por otra. Podríamos imaginar a los enrutadores multiprotocolo tratando de traducir de un formato a otro, pero a menos que los dos formatos sean parientes cercanos con los mismos campos de información, tales conversiones siempre serán incompletas, y frecuentemente destinadas al fracaso. Por esta razón, pocas veces se intentan las conversiones.

Un segundo problema, más serio, es el direccionamiento. Imagine un caso sencillo: un *host* de Internet está tratando de enviar un paquete IP a un *host* en una red SNA adyacente. Se podría pensar en una conversión entre direcciones IP y SNA en ambas direcciones. Además, el concepto de lo que es direccionable es diferente. En IP, los *hosts* (en realidad las tarjetas de red) tienen direcciones. En SNA, entidades diferentes a los *hosts* (por ejemplo dispositivos de hardware) pueden también tener direcciones. En el mejor de los casos, alguien tendría que mantener una base de

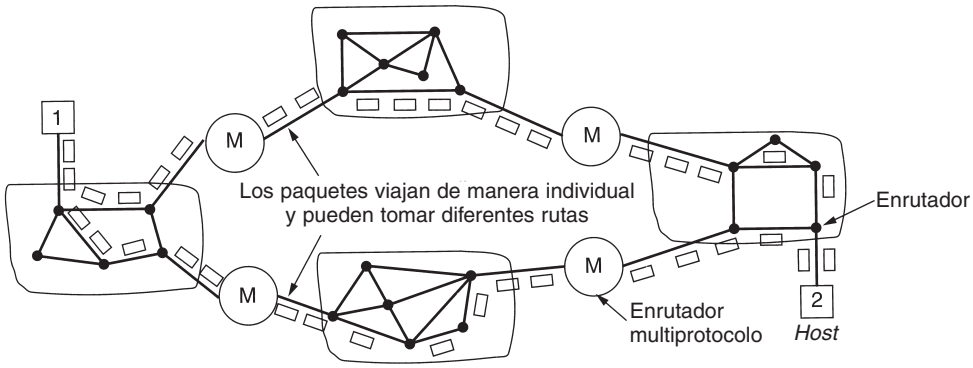


Figura 5-46. Una interred no orientada a la conexión.

datos de las conversiones de todo a todo en la medida de lo posible, pero esto sería constantemente una fuente de problemas.

Otra idea es diseñar un paquete universal de "interred" y hacer que todos los enrutadores lo reconozcan. Este enfoque es, precisamente, el que tiene IP: un paquete diseñado para llevarse por muchas redes. Por supuesto, podría suceder que IPv4 (el protocolo actual de Internet) expulse del mercado a todos los formatos, que Ipv6 (el protocolo futuro de Internet) no se vuelva popular y que no se invente nada más, pero la historia sugiere otra cosa. Hacer que todos se pongan de acuerdo en un solo formato es difícil, más aún cuando las empresas consideran que es una ventaja para ellas contar con un formato patentado bajo su control.

Recapitulemos ahora brevemente las dos maneras en que puede abordarse la interconectividad de redes. El modelo de circuitos virtuales concatenados tiene en esencia las mismas ventajas que el uso de circuitos virtuales en una sola subred: pueden reservarse búferes por adelantado, puede garantizarse la secuencia, pueden usarse encabezados cortos y pueden evitarse los problemas causados por paquetes duplicados retrasados.

El modelo también tiene las mismas desventajas: el espacio de tablas requerido en los enrutadores para cada conexión abierta, la falta de enrutamiento alterno para evitar áreas congestionadas y la vulnerabilidad a fallas de los enrutadores a lo largo de la ruta. También tiene la desventaja de que su implementación es difícil, si no imposible, si una de las redes que intervienen es una red no confiable de datagramas.

Las propiedades del enfoque por datagramas para la interconectividad son las mismas que las de las subredes de datagramas: un mayor potencial de congestión, pero también mayor potencial para adaptarse a él, la robustez ante fallas de los enrutadores y la necesidad de encabezados más grandes. En una interred son posibles varios algoritmos de enrutamiento adaptativo, igual que en una sola red de datagramas.

Una ventaja principal del enfoque por datagramas para la interconectividad es que puede usarse en subredes que no usan circuitos virtuales. Muchas LANs, redes móviles (por ejemplo, flotas

aéreas y navales) e incluso algunas WANs caen en esta categoría. Cuando una interred incluye una de éstas, surgen serios problemas si la estrategia de interredes se basa en circuitos virtuales.

5.5.5 Entunelamiento

El manejo del caso general de lograr la interacción de dos redes diferentes es en extremo difícil. Sin embargo, hay un caso especial común que puede manejarse. Este caso es cuando el *host* de origen y el de destino están en la misma clase de red, pero hay una red diferente en medio. Como ejemplo, piense en un banco internacional con una Ethernet basada en TCP/IP en París, una Ethernet basada en TCP/IP en Londres y una WAN no IP (por ejemplo, ATM) en medio, como se muestra en la figura 5-47.

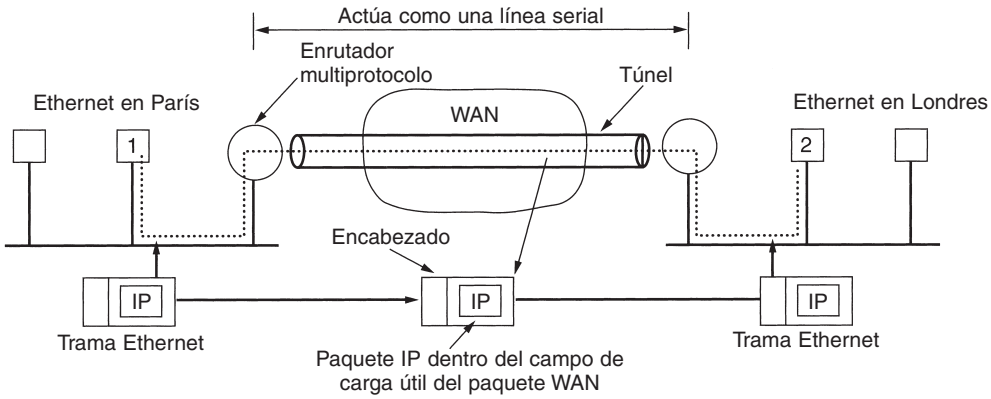


Figura 5-47. Entunelamiento de un paquete de París a Londres.

La solución a este problema es una técnica llamada **entunelamiento.** Para enviar un paquete IP al *host* 2, el *host* 1 construye el paquete que contiene la dirección IP del *host* 2, a continuación lo inserta en una trama Ethernet dirigida al enrutador multiprotocolo de París y, por ultimo, lo pone en la línea Ethernet. Cuando el enrutador multiprotocolo recibe la trama, retira el paquete IP, lo inserta en el campo de carga útil del paquete de capa de red de la WAN y dirige este último a la dirección de la WAN del enrutador multiprotocolo de Londres. Al llegar ahí, el enrutador de Londres retira el paquete IP y lo envía al *host* 2 en una trama Ethernet.

La WAN puede visualizarse como un gran túnel que se extiende de un enrutador multiprotocolo al otro. El paquete IP simplemente viaja de un extremo del túnel al otro, bien acomodado en una caja bonita. No tiene que preocuparse por lidiar con la WAN. Tampoco tienen que hacerlo los *hosts* de cualquiera de las Ethernets. Sólo el enrutador multiprotocolo tiene que entender los paquetes IP y WAN. De hecho, la distancia completa entre la mitad de un enrutador multiprotocolo y la mitad del otro actúa como una línea serial.

El entunelamiento puede aclararse mediante una analogía. Considere una persona que maneja su auto de París a Londres. En Francia, el auto se mueve con su propia energía, pero al llegar al Canal de la Mancha, se carga en un tren de alta velocidad y se transporta a Inglaterra a través del Chunnel (los autos no pueden conducirse a través del Chunnel). En efecto, el auto se transporta como carga, como se muestra en la figura 5-48. En el otro extremo, se libera el auto en las carreteras inglesas y nuevamente continúa moviéndose con sus propios medios. El entunelamiento de paquetes a través de una red foránea funciona de la misma manera.

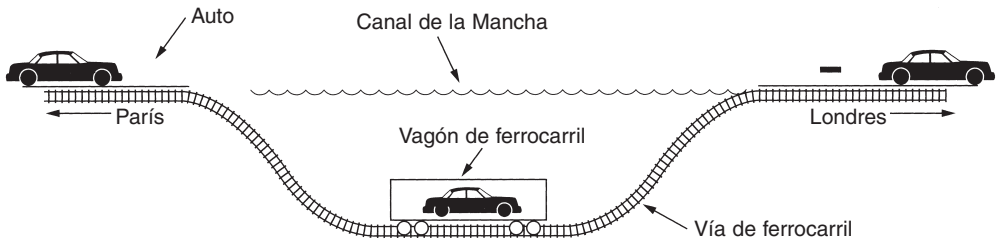


Figura 5-48. Paso de un auto de Francia a Inglaterra a través de un túnel.

5.5.6 Enrutamiento entre redes

El enrutamiento a través de una interred es parecido al enrutamiento en una sola subred, pero con algunas complicaciones adicionales. Por ejemplo, considere la interred de la figura 5-49(a) en la que cinco redes están conectadas mediante seis enrutadores (posiblemente multiprotocolo). Realizar un modelo de grafo de esta situación es complicado por el hecho de que cada enrutador multiprotocolo puede acceder (es decir, enviar paquetes) de manera directa a todos los demás enrutadores conectados a cualquier red a la que esté conectado. Por ejemplo, *B* en la figura 5-49(a) puede acceder directamente a *A* y a *C* a través de la red 2, y también a *D* a través de la red 3. Esto conduce al grafo de la figura 5-49(b).

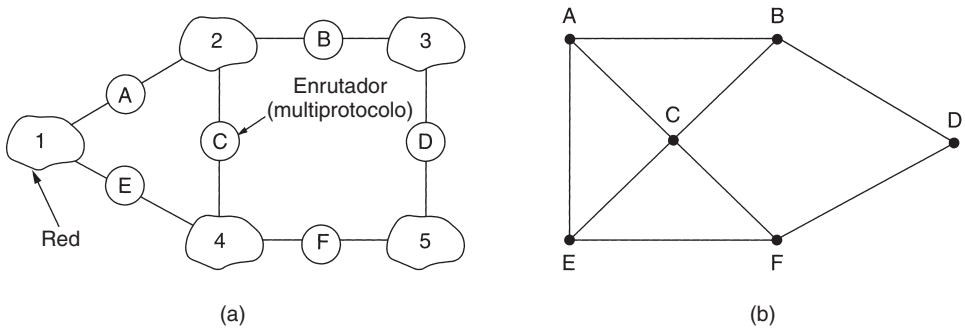


Figura 5-49. (a) Una interred. (b) Grafo de la interred.

Una vez construido el grafo, pueden aplicarse algoritmos de enrutamiento conocidos, como el algoritmo de vector de distancia y el de estado del enlace, al grupo de enrutadores multiprotocolo. Esto da un algoritmo de enrutamiento de dos niveles: en cada red se utiliza un **protocolo de puerta de enlace interior** (**IGP**)**,** pero entre ellas se usa un **protocolo de puerta de enlace exterior** (**EGP**) ("puerta de enlace" es un término antiguo para "enrutador"). De hecho, debido a que estas redes son independientes, cada una puede utilizar un algoritmo diferente del de la otra. Puesto que cada red de una interred es independiente de las demás, con frecuencia se le llama **sistema autónomo** (**AS).**

Un paquete de interred típico parte de su LAN hacia el enrutador multiprotocolo local (en el encabezado de la capa de MAC). Al llegar ahí, el código de la capa de red decide por cuál enrutador multiprotocolo reenviará el paquete, usando sus propias tablas de enrutamiento. Si ese enrutador puede alcanzarse usando el protocolo de la red nativa del paquete, éste se reenvía directamente ahí. De otra manera, se envía por túnel, encapsulado en el protocolo requerido por la red que interviene. Este proceso se repite hasta que el paquete llega a la red de destino.

Una de las diferencias del enrutamiento entre las redes y el enrutamiento dentro de las redes es que el primero con frecuencia requiere el cruce de fronteras internacionales. De pronto, entran en escena varias leyes, como las estrictas leyes suecas de confidencialidad sobre la exportación de datos personales de ciudadanos suecos. Otro ejemplo es la ley canadiense que indica que el tráfico de datos que se origina en Canadá y llega a un destino en Canadá no puede dejar el país. Esto significa que el tráfico de Windsor, Ontario a Vancouver no puede enrutarse a través de Detroit, Estado Unidos, incluso si esta ruta es más rápida y barata.

Otra diferencia entre el enrutamiento interior y el exterior es el costo. Dentro de una sola red, normalmente se aplica un solo algoritmo de cargo. Sin embargo, redes diferentes pueden estar bajo administraciones diferentes, un una ruta puede ser menos cara que otra. Del mismo modo, la calidad de servicio ofrecida por diferentes redes puede ser distinta, y ésta puede ser una razón para escoger una ruta y no otra.

5.5.7 Fragmentación

Cada red impone un tamaño máximo a sus paquetes. Estos límites tienen varias razones, entre ellas:

1. El hardware (por ejemplo, el tamaño de una trama Ethernet).
2. El sistema operativo (por ejemplo, todos los búferes son de 512 bytes).
3. Los protocolos (por ejemplo, la cantidad de bits en el campo de longitud de paquete).
4. El cumplimiento de algún estándar (inter)nacional.
5. El deseo de reducir hasta cierto nivel las retransmisiones inducidas por errores.
6. El deseo de evitar que un paquete ocupe el canal demasiado tiempo.

El resultado de estos factores es que los diseñadores de redes no están en libertad de escoger cualquier tamaño máximo de paquetes que deseen. Las cargas útiles máximas van desde 48 bytes (celdas ATM) hasta 65,515 bytes (paquetes IP), aunque el tamaño de la carga útil en las capas superiores con frecuencia es más grande.

Surge un problema obvio cuando un paquete grande quiere viajar a través de una red cuyo tamaño máximo de paquete es demasiado pequeño. Una solución es asegurar que no ocurra el problema. En otras palabras, la interred debe usar un algoritmo de enrutamiento que evite el envío de paquetes a través de redes que no pueden manejarlos. Sin embargo, esta solución en realidad no es una solución. ¿Qué ocurre si el paquete original es demasiado grande para ser manejado por la red de destino? El algoritmo de enrutamiento no puede pasar por alto el destino.

Básicamente, la única solución al problema es permitir que las puertas de enlace dividan los paquetes en **fragmentos,** enviando cada paquete como paquete de interred individual. Sin embargo, como lo sabe cualquier padre de un niño pequeño, la conversión de un objeto grande en fragmentos pequeños es significativamente más fácil que el proceso inverso. (Los físicos incluso le han dado un nombre a este efecto: segunda ley de la termodinámica.) Las redes de conmutación de paquetes también tienen problemas al unir nuevamente los fragmentos.

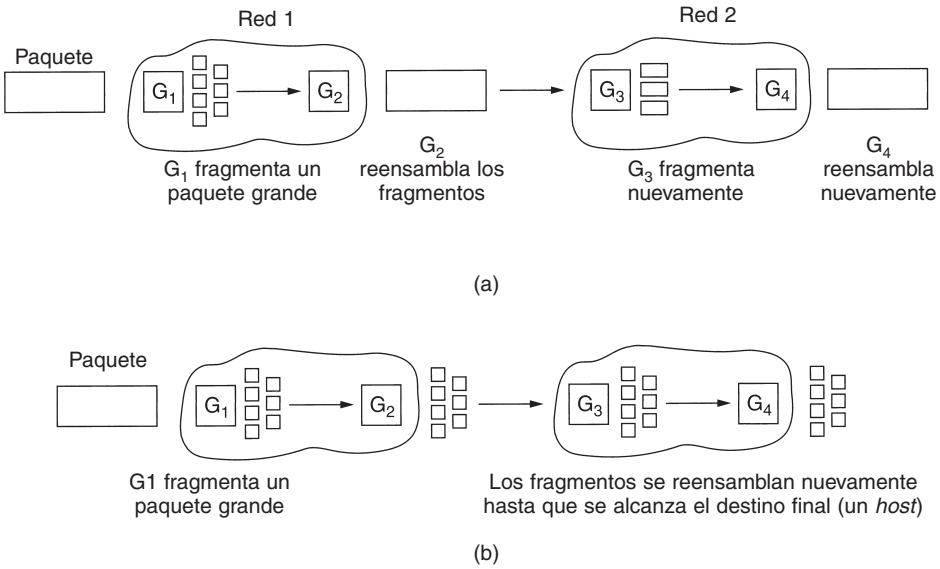


Figura 5-50. (a) Fragmentación transparente. (b) Fragmentación no transparente.

Existen dos estrategias opuestas para recombinar los fragmentos y recuperar el paquete original. La primera es hacer transparente la fragmentación causada por una red de "paquete pequeño" a las demás redes subsiguientes por las que debe pasar el paquete para llegar a su destino final. Esta opción se muestra en la figura 5-50(a). Con este método, la red de paquete pequeño tiene puertas de enlace (lo más probable es que sean enrutadores especializados) que interactúan con

otras redes. Cuando un paquete de tamaño excesivo llega a una puerta de enlace, ésta lo divide en fragmentos. Todos los fragmentos se dirigen a la misma puerta de enlace de salida, donde se recombinan las piezas. De esta manera se ha hecho transparente el paso a través de la red de paquete pequeño. Las redes subsiguientes ni siquiera se enteran de que ha ocurrido una fragmentación. Las redes ATM, por ejemplo, tienen *hardware* especial para proporcionar fragmentación transparente de paquetes en celdas y luego reensamblar las celdas en paquetes. En el mundo ATM, a la fragmentación se le llama segmentación; el concepto es el mismo, pero algunos de los detalles son diferentes.

La fragmentación transparente es sencilla, pero tiene algunos problemas. Por una parte, la puerta de enlace de salida debe saber cuándo ha recibido todas las piezas, por lo que debe incluirse un campo de conteo o un bit de "fin de paquete" en cada paquete. Por otra parte, todos los paquetes deben salir por la misma puerta de enlace. Al no permitir que algunos fragmentos sigan una ruta al destino final, y otros fragmentos una ruta distinta, puede bajar un poco el desempeño. Un último problema es la sobrecarga requerida para reensamblar y volver a fragmentar repetidamente un paquete grande que pasa a través de una serie de redes de paquete pequeño. ATM requiere fragmentación transparente.

La otra estrategia de fragmentación es abstenerse de recombinar los fragmentos en las puertas de enlace intermedias. Una vez que se ha fragmentado un paquete, cada fragmento se trata como si fuera un paquete original. Todos los fragmentos pasan a través de la puerta de enlace (o puertas de enlace) de salida, como se muestra en la figura 5-50(b). La recombinación ocurre sólo en el *host* de destino. IP funciona de esta manera.

La fragmentación no transparente también tiene algunos problemas. Por ejemplo, requiere que "*todos*" los *hosts* sean capaces de hacer el reensamble. Otro problema es que, al fragmentarse un paquete grande, aumenta la sobrecarga total, pues cada fragmento debe tener un encabezado. En tanto que en el primer método la sobrecarga desaparece en cuanto se sale de la red de paquete pequeño, en este método la sobrecarga permanece durante el resto de la travesía. Sin embargo, una ventaja de este método es que ahora pueden usarse varias puertas de enlace de salida, lográndose un mejor desempeño. Por supuesto, si se está usando el modelo de circuito virtual concatenado, esta ventaja no es de ninguna utilidad.

Cuando se divide un paquete, los fragmentos deben numerarse de tal manera que el flujo de datos original pueda reconstruirse. Una manera de numerar los fragmentos es usar un árbol. Si el paquete 0 debe dividirse, se llama a las partes 0.0, 0.1, 0.2, etcétera. Si estos mismos fragmentos deben fragmentarse después, las piezas se numeran como 0.0.0, 0.0.1, 0.0.2,...,0.1.0, 0.1.1, 0.1.2, etcétera. Si se reservan suficientes campos en el encabezado para el peor caso y no se generan duplicaciones en ningún lado, este esquema es suficiente para asegurar que todas las partes puedan reensamblarse correctamente en el destino, sin importar en qué orden lleguen.

Sin embargo, si cualquier red pierde o descarta paquetes, hay la necesidad de retransmisiones de extremo a extremo, con efectos poco afortunados para el sistema de numeración. Suponga que un paquete de 1024 bits se fragmenta inicialmente en cuatro fragmentos del mismo tamaño, 0.0, 0.1, 0.2 y 0.3. Se pierde el fragmento 0.1, pero las otras partes llegan al destino. En algún momento termina el temporizador del origen y se vuelve a transmitir el paquete original, pero esta vez la ruta tomada pasa a través de una red con un límite de 512 bits, por lo que se generan dos fragmentos.

Cuando el nuevo fragmento 0.1 llegue al destino, el receptor pensará que ya se han recibido las cuatro partes y reconstruirá incorrectamente el paquete.

Un sistema de numeración completamente diferente, y mejor, es que el protocolo de interred defina un tamaño de fragmento elemental lo bastante pequeño como para que el fragmento elemental pueda pasar a través de todas las redes. Al fragmentarse un paquete, todas las partes son iguales al tamaño de fragmento elemental, excepto la última, que puede ser más corta. Un paquete de interred puede contener varios fragmentos, por razones de eficiencia. El encabezado de interred debe proporcionar el número de paquete original y el número del (primer) fragmento elemental contenido en el paquete. Como siempre, también debe haber un bit que indique que el último fragmento elemental contenido en el paquete de interred es el último del paquete original.

Este método requiere dos campos de secuencia en el encabezado de interred: el número original de paquete y el número de fragmento. Evidentemente hay una concesión entre el tamaño del fragmento elemental y la cantidad de bits en el número de fragmento. Dado que se supone que el tamaño del fragmento elemental es aceptable para todas las redes, la fragmentación subsiguiente de un paquete de interred que contiene varios fragmentos no produce problemas. El límite último aquí es hacer que el fragmento elemental sea un solo bit o byte y, por lo tanto, el número de fragmento es el desplazamiento del bit o byte en el paquete original, como se muestra en la figura 5-51.

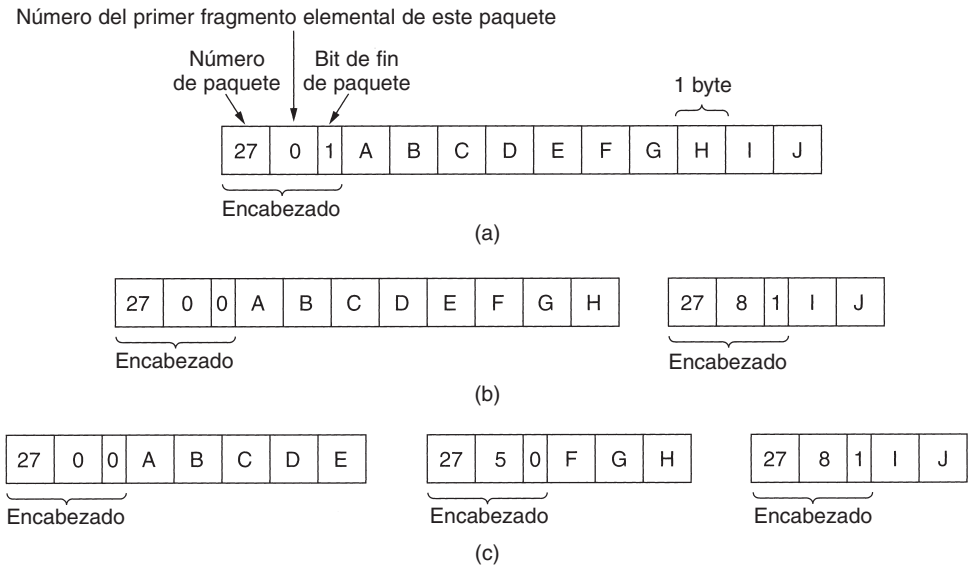


Figura 5-51. Fragmentación cuando el tamaño de datos elemental es de 1 byte. (a) Paquete original que contiene 10 bytes de datos. (b) Fragmentos tras pasar a través de una red con un tamaño máximo de paquete de 8 bytes de carga útil más encabezado. (c) Fragmentos tras pasar a través de una puerta de enlace de tamaño 5.

Algunos protocolos de interred llevan este método aún más lejos y consideran a toda la transmisión a través de un circuito virtual como un paquete gigantesco, por lo que cada fragmento contiene el número absoluto de byte del primer byte del fragmento.

5.6 LA CAPA DE RED DE INTERNET

Antes de entrar a los detalles específicos de la capa de red de Internet, vale la pena dar un vistazo a los principios que guiaron su diseño en el pasado y que hicieron posible el éxito que tiene hoy en día. En la actualidad, con frecuencia parece que la gente los ha olvidado. Estos principios se enumeran y analizan en el RFC 1958, el cual vale la pena leer (y debería ser obligatorio para todos los diseñadores de protocolos, así como un examen al final de la lectura). Este RFC utiliza mucho las ideas encontradas en (Clark, 1988, y Saltzer y cols., 1984). A continuación presentamos los que consideramos como los 10 mejores principios (del más al menos importante).

1. **Asegúrese de que funciona.** No termine el diseño o estándar hasta que múltiples prototipos se hayan comunicado entre sí de manera exitosa. Con demasiada frecuencia, los diseñadores primero escriben un estándar de 1000 páginas, lo aprueban y después descubren que tiene demasiadas fallas y no funciona. Después escriben la versión 1.1 de ese mismo estándar. Ésta no es la manera correcta de proceder.
2. **Mantenga la simplicidad.** Cuando tenga duda, utilice la solución más simple. William de Occam formuló este principio (la navaja de Occam) en el siglo XIV. Dicho en otras palabras: combata las características. Si una característica no es absolutamente esencial, descártela, especialmente si el mismo efecto se puede alcanzar mediante la combinación de otras características.
3. **Elija opciones claras.** Si hay varias maneras para realizar la misma tarea, elija sólo una. Tener dos o más formas de hacer lo mismo es buscarse problemas. Con frecuencia, los estándares tienen múltiples opciones o modos o parámetros debido a que personas poderosas insisten en que su método es el mejor. Los diseñadores deben resistir con fuerza esta tendencia. Simplemente diga no.
4. **Explote la modularidad.** Este principio lleva directamente a la idea de tener pilas de protocolos, cuyas capas son independientes entre sí. De esta forma, si las circunstancias requieren que un módulo o capa cambie, los otros no se verán afectados.
5. **Prevea la heterogeneidad.** En cualquier red grande habrán diferentes tipos de hardware, facilidades de transmisión y aplicaciones. Para manejarlos, el diseño de la red debe ser simple, general y flexible.
6. **Evite las opciones y parámetros estáticos.** Si los parámetros son inevitables (por ejemplo, el tamaño máximo del paquete), es mejor hacer que el emisor y el receptor negocien un valor que definir opciones fijas.

7. **Busque un buen diseño; no es necesario que sea perfecto.** Con frecuencia, los diseñadores tienen un buen diseño pero éste no puede manejar algún caso especial. En lugar de desbaratar el diseño, los diseñadores deberían dejar que esa parte la resuelvan las personas con el caso especial.
8. **Sea estricto cuando envíe y tolerante cuando reciba.** En otras palabras, sólo envíe paquetes que cumplan rigurosamente con los estándares, pero espere paquetes que tal vez no cumplan del todo y trate de lidiar con ellos.
9. **Piense en la capacidad de crecimiento.** Si el sistema va a manejar de manera efectiva millones de *hosts* y miles de millones de usuarios, las bases de datos no centralizadas de cualquier tipo son tolerables y la carga debe dispersarse lo más equitativamente posible en los recursos disponibles.
10. **Considere el desempeño y el costo.** Si una red tiene un desempeño pobre o un costo exagerado, nadie la utilizará.

Dejemos a un lado los principios generales y comencemos a ver los detalles de la capa de red de Internet. En la capa de red, la Internet puede verse como un conjunto de subredes, o **sistemas autónomos** interconectados. No hay una estructura real, pero existen varias redes dorsales principales. Éstas se construyen a partir de líneas de alto ancho de banda y enrutadores rápidos. Conectadas a las redes dorsales hay redes regionales (de nivel medio), y conectadas a estas redes regionales están las LANs de muchas universidades, compañías y proveedores de servicios de Internet. En la figura 5-52 se presenta un dibujo de esta organización cuasijerárquica.

El pegamento que mantiene unida a Internet es el protocolo de capa de red, **IP** (**Protocolo de Internet**). A diferencia de la mayoría de los protocolos de capa de red anteriores, éste se diseñó desde el principio con la interconexión de redes en mente. Una buena manera de visualizar la capa de red es la siguiente. Su trabajo es proporcionar un medio de mejor esfuerzo (es decir, sin garantía) para el transporte de datagramas del origen al destino, sin importar si estas máquinas están en la misma red, o si hay otras redes entre ellas.

La comunicación en Internet funciona como sigue. La capa de transporte toma flujos de datos y los divide en datagramas. En teoría, los datagramas pueden ser de hasta 64 Kbytes cada uno, pero en la práctica por lo general son de unos 1500 bytes (por lo tanto, se ajustan en una trama de Ethernet). Cada datagrama se transmite a través de Internet, posiblemente fragmentándose en unidades más pequeñas en el camino. Cuando todas las piezas llegan finalmente a la máquina de destino, son reensambladas por la capa de red, dejando el datagrama original. A continuación este datagrama se entrega a la capa de transporte, que lo introduce en el flujo de entrada del proceso receptor. Como se muestra en la figura 5-52, un paquete que se origina en el *host* 1 tiene que atravesar seis redes para llegar al *host* 2. En la práctica, por lo general son más de seis.

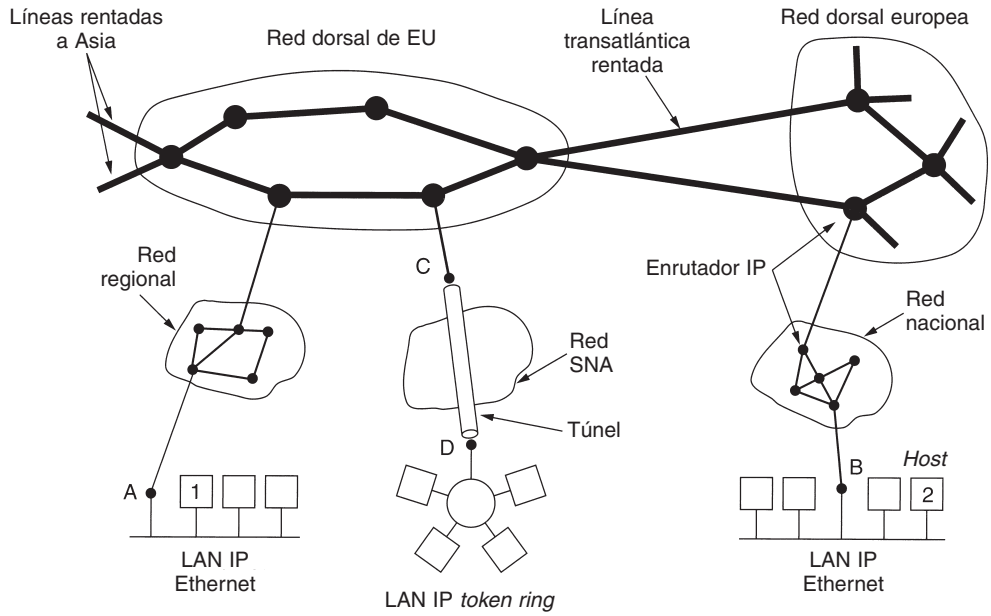


Figura 5-52. Internet es un conjunto interconectado de muchas redes.

5.6.1 El protocolo IP

Un lugar adecuado para comenzar nuestro estudio de la capa de red de Internet es el formato de los datagramas de IP mismos. Un datagrama IP consiste en una parte de encabezado y una parte de texto. El encabezado tiene una parte fija de 20 bytes y una parte opcional de longitud variable. El formato del encabezado se muestra en la figura 5-53. Se transmite en orden de *big endian*: de izquierda a derecha, comenzando por el bit de orden mayor del campo de *Versión*. (SPARC es *big endian*; Pentium es *little endian*.) En las máquinas *little endian*, se requiere conversión por software tanto para la transmisión como para la recepción.

El campo de *Versión* lleva el registro de la versión del protocolo al que pertenece el datagrama. Al incluir la versión en cada datagrama, es posible hacer que la transición entre versiones se lleve meses, o incluso años, ejecutando algunas máquinas la versión vieja y otras la versión nueva. En la actualidad, se está trabajando en una transición entre IPv4 e IPv6, la cual ha tomado años, y no está cerca de terminarse (Durand, 2001; Wiljakka, 2002, y Waddington y Chang, 2002). Algunas personas incluso piensan que nunca se terminará (Weiser, 2001). Como una adición a la numeración, IPv5 fue un protocolo de flujo experimental en tiempo real que no se utilizó ampliamente.

Dado que la longitud del encabezado no es constante, se incluye un campo en el encabezado, *IHL*, para indicar la longitud en palabras de 32 bits. El valor mínimo es de 5, cifra que se aplica cuando no hay opciones. El valor máximo de este campo de 4 bits es 15, lo que limita el encabezado

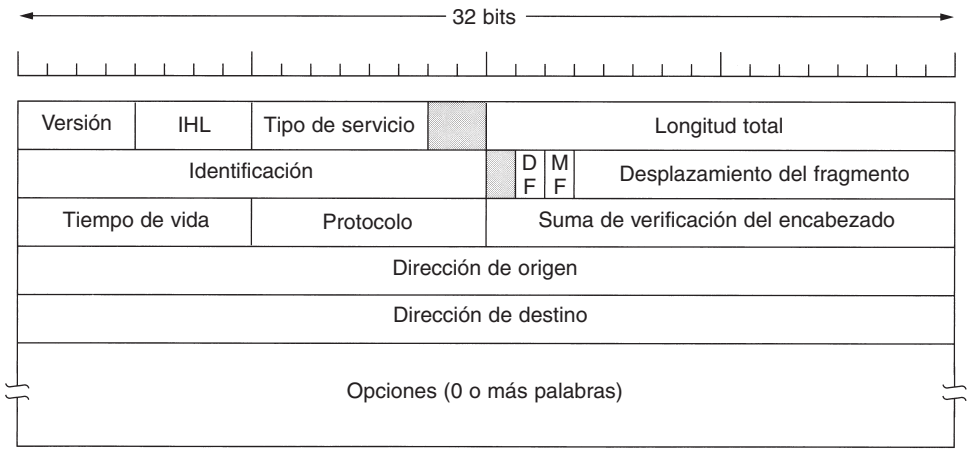


Figura 5-53. El encabezado de IPv4 (Protocolo Internet).

a 60 bytes y, por lo tanto, el campo de *Opciones* a 40 bytes. Para algunas opciones, por ejemplo para una que registre la ruta que ha seguido un paquete, 40 bytes es muy poco, lo que hace inútil esta opción.

El campo de *Tipo de servicio* es uno de los pocos campos que ha cambiado su significado (levemente) durante años. Su propósito aún es distinguir entre las diferentes clases de servicios. Son posibles varias combinaciones de confiabilidad y velocidad. Para voz digitalizada, la entrega rápida le gana a la entrega precisa. Para la transferencia de archivos, es más importante la transmisión libre de errores que la rápida.

Originalmente, el campo de 6 bits contenía (de izquierda a derecha) un campo de *Precedencia* de tres bits y tres banderas, *D*, *T* y *R*. El campo de *Precedencia* es una prioridad, de 0 (normal) a 7 (paquete de control de red). Los tres bits de bandera permiten al *host* especificar lo que le interesa más del grupo {retardo (*delay*), velocidad real de transporte (*throughput*), confiabilidad (*reliability*)}. En teoría, estos campos permiten a los enrutadores tomar decisiones entre, por ejemplo, un enlace satelital de alto rendimiento y alto retardo o una línea arrendada con bajo rendimiento y poco retardo. En la práctica, los enrutadores actuales ignoran por completo el campo de *Tipo de servicio,* a menos que se les indique lo contrario.

En algún momento, la IETF tiró la toalla y cambió el campo ligeramente para acomodar los servicios diferenciados. Seis de los bits se utilizan para indicar a cuáles de las clases de servicios analizadas anteriormente pertenece cada paquete. Estas clases incluyen las cuatro propiedades de encolamiento, tres posibilidades de eliminación y las clases históricas.

La *Longitud total* incluye todo el datagrama: tanto el encabezado como los datos. La longitud máxima es de 65,535 bytes. Actualmente este límite es tolerable, pero con las redes futuras de gigabits se requerirán datagramas más grandes.

El campo de *Identificación* es necesario para que el *host* de destino determine a qué datagrama pertenece un fragmento recién llegado. Todos los fragmentos de un datagrama contienen el mismo valor de *Identificación*.

A continuación viene un bit sin uso y luego dos campos de 1 bit. *DF* significa no fragmentar (*Don't Fragment*); es una orden para los enrutadores de que no fragmenten el datagrama, porque el destino es incapaz de juntar las piezas de nuevo. Por ejemplo, al arrancar una computadora, su ROM podría pedir el envío de una imagen de memoria a ella como un solo datagrama. Al marcar el datagrama con el bit *DF*, el transmisor sabe que llegará en una pieza, aún si significa que el datagrama debe evitar una red de paquete pequeño en la mejor ruta y tomar una ruta subóptima. Se requiere que todas las máquinas acepten fragmentos de 576 bytes o menos.

MF significa más fragmentos. Todos los fragmentos excepto el último tienen establecido este bit, que es necesario para saber cuándo han llegado todos los fragmentos de un datagrama.

El *Desplazamiento del fragmento* indica en qué parte del datagrama actual va este fragmento. Todos los fragmentos excepto el último del datagrama deben tener un múltiplo de 8 bytes, que es la unidad de fragmentos elemental. Dado que se proporcionan 13 bits, puede haber un máximo de 8192 fragmentos por datagrama, dando una longitud máxima de datagrama de 65,536 bytes, uno más que el campo de *Longitud total*.

El campo de *Tiempo de vida* es un contador que sirve para limitar la vida de un paquete. Se supone que este contador cuenta el tiempo en segundos, permitiendo una vida máxima de 255 seg; debe disminuirse en cada salto y se supone que disminuye muchas veces al encolarse durante un tiempo grande en un enrutador. En la práctica, simplemente cuenta los saltos. Cuando el contador llega a cero, el paquete se descarta y se envía de regreso un paquete de aviso al *host* de origen. Esta característica evita que los datagramas vaguen eternamente, algo que de otra manera podría ocurrir si se llegan a corromper las tablas de enrutamiento.

Una vez que la capa de red ha ensamblado un datagrama completo, necesita saber qué hacer con él. El campo de *Protocolo* indica el protocolo de las capas superiores al que debe entregarse el paquete. TCP es una posibilidad, pero también está UDP y algunos más. La numeración de los protocolos es global en toda Internet, y se define en el RFC 1700, pero en la actualidad dichos protocolos están contenidos en una base de datos en línea localizada en *www.iana.org*.

La *Suma de verificación del encabezado* verifica solamente el encabezado. Tal suma de verificación es útil para la detección de errores generados por palabras de memoria erróneas en un enrutador. El algoritmo es sumar todas las medias palabras de 16 bits a medida que llegan, usando aritmética de complemento a uno, y luego obtener el complemento a uno del resultado. Para los fines de este algoritmo, se supone que la *suma de verificación del encabezado* es cero cuando llega el paquete al destino. Este algoritmo es más robusto que una suma normal. Observe que la *suma de verificación del encabezado* debe recalcularse en cada salto, pues cuando menos uno de los campos siempre cambia (el campo de *Tiempo de vida*), pero pueden usarse trucos para acelerar el cálculo.

La *Dirección de origen* y la *Dirección de destino* indican el número de red y el número de *host*. Estudiaremos las direcciones de Internet en la siguiente sección. El campo de *Opciones* se diseñó para proporcionar un recurso que permitiera que las versiones subsiguientes del protocolo incluyeran información no presente en el diseño original, para permitir que los experimentadores prueben ideas nuevas y para evitar la asignación de bits de encabezado a información pocas veces necesaria. Las opciones son de longitud variable. Cada una empieza con un código de 1 byte que identifica la opción. Algunas opciones van seguidas de un campo de longitud de la opción de 1 byte, y luego de uno o más bytes de datos. El campo de *Opciones* se rellena para completar múltiplos de cuatro bytes. Originalmente se definieron cinco opciones, como se lista en la

figura 5-54, pero se han agregado otras más. La lista completa ahora se mantiene en línea en *www.iana.org/assignments/ip-parameters*

Opción	Descripción
Seguridad	Especifica qué tan secreto es el datagrama
Enrutamiento estricto desde el origen	Indica la ruta completa a seguir
Enrutamiento libre desde el origen	Da una lista de los enrutadores que no deben evitarse
Registrar ruta	Hace que cada enrutador agregue su dirección IP
Marca de tiempo	Hace que cada enrutador agregue su dirección y su marca de tiempo

Figura 5-54. Algunas de las opciones del IP.

La opción de *seguridad* indica qué tan secreta es la información. En teoría, un enrutador militar puede usar este campo para especificar que no se enrute a través de ciertos países que los militares consideren "malos". En la práctica, todos los enrutadores lo ignoran, por lo que su única función real es la de ayudar a los espías a encontrar la información importante con mayor facilidad.

La opción de *enrutamiento estricto desde el origen* da la ruta completa desde el origen hasta el destino como secuencia de direcciones IP. Se requiere que el datagrama siga esa ruta exacta. Esta opción se usa sobre todo cuando los administradores de sistemas envían paquetes de emergencia porque las tablas de enrutamiento se han dañado, o para hacer mediciones de tiempo.

La opción de *enrutamiento libre desde el origen* requiere que el paquete pase por los enrutadores indicados en la lista, y en el orden especificado, pero se le permite pasar a través de otros enrutadores en el camino. Normalmente, esta opción sólo indicará algunos enrutadores, para obligar a una ruta en particular. Por ejemplo, si se desea obligar a un paquete de Londres a Sydney a ir hacia el oeste en lugar de hacia el este, esta opción podría especificar enrutadores en Nueva York, Los Ángeles y Honolulú. Esta opción es de mucha utilidad cuando las consideraciones políticas o económicas dictan pasar a través de, o evitar, ciertos países.

La opción de *registrar ruta* indica a los enrutadores a lo largo de la ruta que agreguen su dirección IP al campo de opción. Esto permite a los administradores del sistema buscar fallas en los algoritmos de enrutamiento ("¿por qué todos los paquetes de Houston a Dallas pasan por Tokio primero?"). Al establecer inicialmente ARPANET, ningún paquete pasaba nunca por más de nueve enrutadores, por lo que 40 bytes de opciones eran más que suficientes. Como se mencionó antes, ahora esto es demasiado poco.

Por último, la opción de *marca de tiempo* es como la opción de *registrar ruta,* excepto que además de registrar su dirección IP de 32 bits, cada enrutador también registra una marca de tiempo de 32 bits. Esta opción también es principalmente para búsquedas de fallas en los algoritmos de enrutamiento.

5.6.2 Direcciones IP

Cada *host* y enrutador de Internet tiene una dirección IP, que codifica su número de red y su número de *host*. La combinación es única: no hay dos máquinas que tengan la misma dirección IP.

Todas las direcciones IP son de 32 bits de longitud y se usan en los campos de *Dirección de origen* y de *Dirección de destino* de los paquetes IP. En importante mencionar que una dirección IP realmente no se refiere a un *host*. En realidad se refiere a una interfaz de red, por lo que si un *host* está en dos redes, debe tener dos direcciones IP. Sin embargo, en la práctica, la mayoría de los *hosts* se encuentran en una red y, por lo tanto, tienen una dirección IP.

Por varias décadas, las direcciones IP se dividieron en cinco categorías, las cuales se listan en la figura 5-55. Esta asignación se ha llamado **direccionamiento con clase** (*classful addressing*). Ya no se utiliza, pero en la literatura aún es común encontrar referencias. Más adelante analizaremos el reemplazo del direccionamiento con clase.

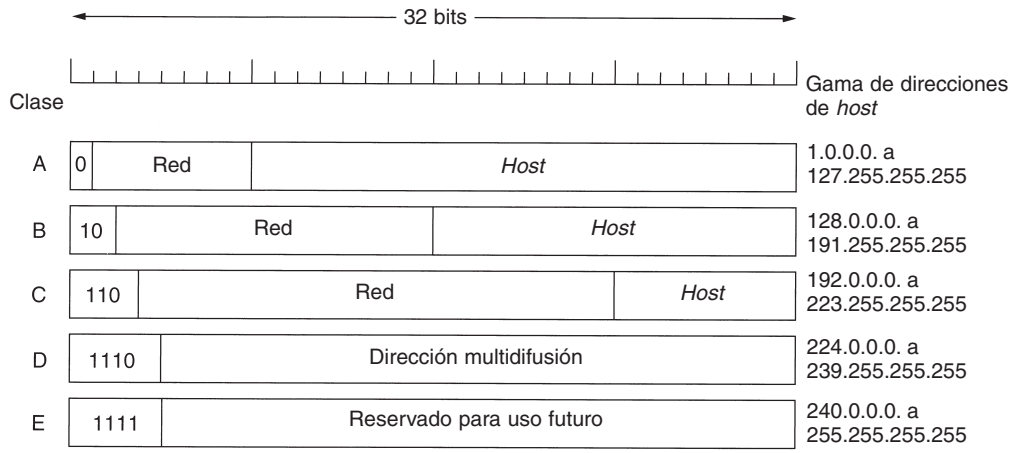


Figura 5-55. Formatos de dirección IP.

Los formatos de clase A, B, C y D permiten hasta 128 redes con 16 millones de *hosts* cada una, 16,382 redes de hasta 64K *hosts*, 2 millones de redes (por ejemplo, LANs) de hasta 256 *hosts* cada una (aunque algunas son especiales). También soportan la multidifusión, en la cual un datagrama es dirigido a múltiples *hosts*. Las direcciones que comienzan con 1111 se reservan para uso futuro. Hay cerca de 500,000 redes conectadas a Internet, y la cifra se duplica cada año. Los números de redes son manejados por una corporación no lucrativa llamada **ICANN** (**Corporación de Internet para la Asignación de Nombres y Números**) para evitar conflictos. A su vez, ICANN ha delegado partes del espacio de direcciones a varias autoridades regionales, las cuales han repartido direcciones IP a los ISPs y a otras compañías.

Las direcciones de red, que son números de 32 bits, generalmente se escriben en **notación decimal con puntos.** En este formato, cada uno de los 4 bytes se escribe en decimal, de 0 a 255. Por ejemplo, la dirección hexadecimal C0290614 se escribe como 192.41.6.20. La dirección IP menor es 0.0.0.0 y la mayor 255.255.255.255.

Los valores 0 y −1(todos 1s) tienen significado especial, como se muestra en la figura 5-56. El valor 0 significa esta red o este *host.* El valor −1 se usa como dirección de difusión para indicar todos los *hosts* de la red indicada.

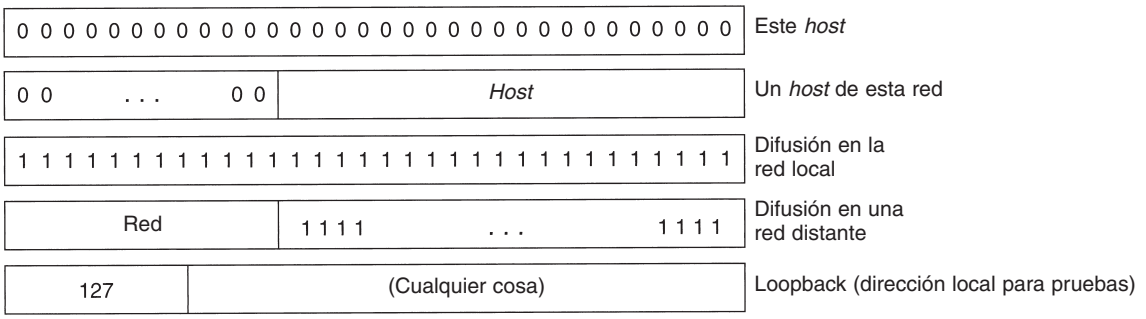


Figura 5-56. Direcciones IP especiales.

La dirección IP 0.0.0.0 es usada por los *hosts* cuando están siendo arrancados, pero no se usa después. Las direcciones IP con 0 como número de red se refieren a la red actual. Estas direcciones permiten que las máquinas se refieran a su propia red sin saber su número (pero tiene que saber su clase para saber cuántos 0s hay que incluir). La dirección que consiste solamente en 1s permite la difusión en la red local, por lo común una LAN. Las direcciones con un número de red propio y solamente unos en el campo de *host* permiten que las máquinas envíen paquetes de difusión a LANs distantes desde cualquier parte de Internet. Por último, todas las direcciones de la forma 127.*xx.yy.zz* se reservan para direcciones locales de prueba (*loopbacks*). Los paquetes enviados a esa dirección no se colocan en el cable; se procesan localmente y se tratan como paquetes de entrada. Esto permite que los paquetes se envíen a la red local sin que el transmisor conozca su número.

Subredes

Como hemos visto, todos los *hosts* de una red deben tener el mismo número de red. Esta propiedad del direccionamiento IP puede causar problemas a medida que crezcan las redes. Por ejemplo, considere una universidad que inició con una red de clase B utilizada por el Depto. de Ciencias de la Computación para las computadoras de su Ethernet. Un año más tarde, el Depto. de Ingeniería Eléctrica deseó conectarse a Internet, por lo que adquirió un repetidor para ampliar la Ethernet CS hasta su edificio. Conforme pasó el tiempo, muchos otros departamentos adquirieron computadoras y rápidamente se alcanzó el límite de cuatro repetidores por Ethernet. Se requirió una organización diferente.

Obtener una segunda dirección de red sería difícil debido a que las direcciones de red son escasas y la universidad ya tiene suficientes direcciones para aproximadamente 60,000 *hosts*. El problema es la regla de que una sola dirección de clase A, B o C haga referencia a una red, no a una colección de LANs. Conforme más y más organizaciones se encontraron en esta situación, se hizo un pequeño cambio al sistema de direccionamiento para manejar tal situación.

La solución a este problema es permitir la división de una red en varias partes para uso interno, pero aún actuar como una sola red ante el mundo exterior. En la actualidad, una red típica de un campus podría lucir como la que se muestra en la figura 5-57, con un enrutador principal conectado a un ISP o a una red regional, y numerosas Ethernets dispersas en diferentes departamentos

del campus. Cada una de las Ethernets tiene su propio enrutador conectado al enrutador principal (posiblemente mediante una LAN de red dorsal, pero la naturaleza de la conexión entre enrutadores no tiene relevancia aquí).

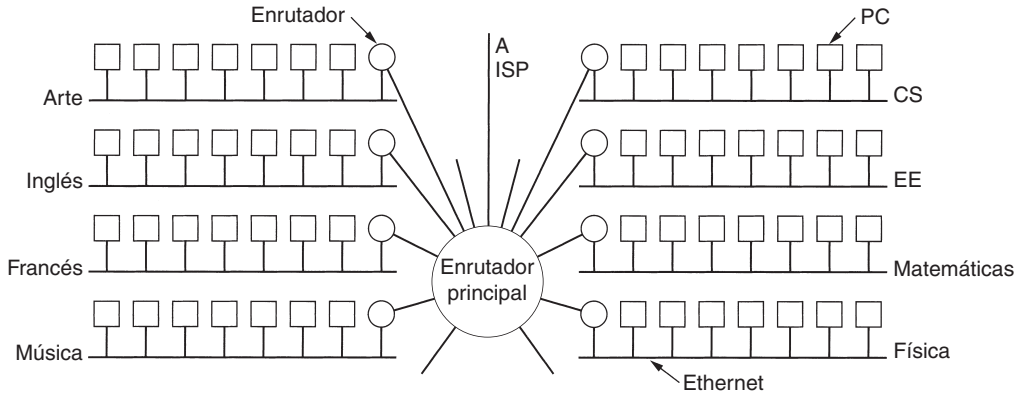


Figura 5-57. Una red de un campus que consiste de LANs para varios departamentos.

En la literatura sobre Internet, a estas partes de la red (en este caso Ethernets) se les llama **subredes.** Como mencionamos en el capítulo 1, este uso entra en conflicto con la "subred" cuyo significado es el grupo de todos los enrutadores y líneas de comunicación de una red. Esperamos que el contexto deje en claro el significado de que se trata. En esta y en la siguiente sección, la nueva definición será la que utilizaremos de manera exclusiva.

Cuando un paquete entra en el enrutador principal, ¿cómo sabe a cuál subred pasarlo (Ethernet)? Una forma sería tener una tabla con 65,536 entradas en el enrutador principal que indiquen cuál enrutador utilizar para cada *host* en el campus. Esta idea funcionaría, pero requeriría una tabla muy grande en el enrutador principal y mucho mantenimiento manual conforme se agregaran, movieran o eliminaran *hosts*.

En su lugar, se inventó un esquema diferente. Básicamente, en lugar de tener una sola dirección de clase B con 14 bits para el número de red y 16 bits para el número de *host*, algunos bits se eliminan del número de *host* para crear un número de subred. Por ejemplo, si la universidad tiene 35 departamentos, podría utilizar un número de subred de 6 bits y un número de *host* de 10 bits, lo que permitiría hasta 64 Ethernets, cada una con un máximo de 1022 *hosts* (0 y −1 no están disponibles, como se mencionó anteriormente). Esta división podría cambiarse posteriormente en caso de que no fuera correcta.

Para implementar subredes, el enrutador principal necesita una **máscara de subred** que indique la división entre el número de red + el número de subred y el *host*, como se muestra en la figura 5-58. Las máscaras de subred también se pueden escribir en notación decimal con puntos, o agregando a la dirección IP una diagonal seguida del número de bits usados para los números de red y subred. Para el ejemplo de la figura 5-58, la máscara de subred puede escribirse como 255.255.252.0. Una notación alternativa es /22 para indicar que la máscara de subred tiene una longitud de 22 bits.

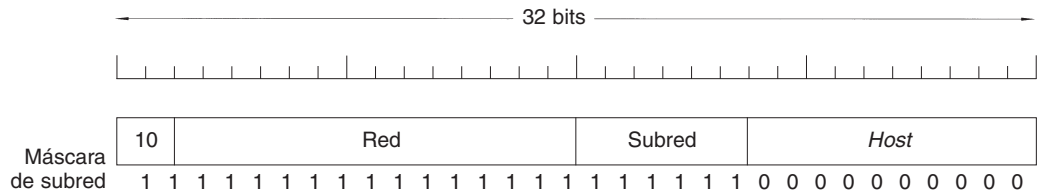


Figura 5-58. Una red de clase B dividida en 64 subredes.

Fuera de la red, la subred no es visible, por lo que la asignación de una subred nueva no requiere comunicación con el ICANN ni la modificación de bases de datos externas. En este ejemplo, la primera subred podría usar direcciones IP a partir de 130.50.4.1, la segunda podría empezar en 130.50.8.1, la tercera podría empezar en 130.50.12.1, etcétera. Para ver por qué las subredes se cuentan en grupos de cuatro, observe que las direcciones binarias correspondientes son como se muestra a continuación:

```
Subred 1: 10000010  00110010  000001|00  00000001
Subred 2: 10000010  00110010  000010|00  00000001
Subred 3: 10000010  00110010  000011|00  00000001
```

La barra vertical (|) muestra el límite entre el número de subred y el número de *host*. A su izquierda se encuentra el número de subred de 6 bits; a su derecha, el número de *host* de 10 bits.

Para ver el funcionamiento de las subredes, es necesario explicar la manera en que se procesan los paquetes IP en un enrutador. Cada enrutador tiene una tabla en la que se lista cierto número de direcciones IP (red, 0) y cierto número de direcciones IP (esta red, *host*). El primer tipo indica cómo llegar a redes distantes. El segundo tipo indica cómo llegar a redes locales. **La interfaz de red a utilizar para alcanzar el destino, así como otra información, está asociada a cada tabla.**

Cuando llega un paquete IP, se busca su dirección de destino en la tabla de enrutamiento. Si el paquete es para una red distante, se reenvía al siguiente enrutador de la interfaz dada en la tabla; si es para un *host* local (por ejemplo, en la LAN del enrutador), se envía directamente al destino. Si la red no está en la tabla, el paquete se reenvía a un enrutador predeterminado con tablas más extensas. Este algoritmo significa que cada enrutador sólo tiene que llevar el registro de otras redes y *hosts* locales (no de pares red-*host*), reduciendo en gran medida el tamaño de la tabla de enrutamiento.

Al introducirse subredes, se cambian las tablas de enrutamiento, agregando entradas con forma de (esta red, subred, 0) y (esta red, esta subred, *host*). Por lo tanto, un enrutador de la subred *k* sabe cómo llegar a todas las demás subredes y a todos los *hosts* de la subred *k*; no tiene que saber los detalles sobre los *hosts* de otras subredes. De hecho, todo lo que se necesita es hacer que cada enrutador haga un AND booleano con la máscara de subred de la red para deshacerse del número de *host* y buscar la dirección resultante en sus tablas (tras determinar de qué clase de red se trata).

Por ejemplo, a un paquete dirigido a 130.50.15.6 que llega a un enrutador de la subred 5 se le aplica un AND con la máscara de subred 255.255.252.0/22 para dar la dirección 130.50.12.0. Esta dirección se busca en las tablas de enrutamiento para averiguar la manera de llegar a los *hosts* de la subred 3. Por lo tanto, la división de redes reduce espacio en la tabla de enrutamiento creando una jerarquía de tres niveles, que consiste en red, subred y *host*.

CIDR—Enrutamiento interdominios sin clases

El IP ya ha estado en uso intensivo por más de una década; ha funcionado extremadamente bien, como lo demuestra el crecimiento exponencial de la Internet. Desgraciadamente, el IP se está convirtiendo con rapidez en víctima de su propia popularidad: se le están acabando las direcciones. Este desastre inminente ha propiciado una gran cantidad de controversias y debates en la comunidad de Internet sobre lo que debe hacerse al respecto. En esta sección describiremos tanto el problema como varias soluciones propuestas.

En 1987 unos pocos visionarios predijeron que algún día Internet podría crecer hasta 100,000 redes. La mayoría de los expertos minimizaron el problema aduciendo que ocurriría en muchas décadas, si es que llegaba a ocurrir. En 1996 se conectó la red 100,000. El problema, en pocas palabras, es que Internet se está quedando rápidamente sin direcciones de IP. En teoría, existen cerca de dos mil millones de direcciones, pero la práctica de organizar el espacio de direcciones por clases (véase la figura 5-55) desperdicia millones de ellas. En particular, el verdadero villano es la red clase B. Para la mayoría de las organizaciones, una red clase A, con 16 millones de direcciones, es demasiado grande, y una red clase C, de 256 direcciones, es demasiado pequeña. Una red clase B, con 65,536, es la adecuada. En el folclor de Internet, esta situación se conoce como el **problema de los tres osos** (del cuento *Ricitos de Oro y los tres osos*).

En realidad, una dirección clase B es demasiado grande para la mayoría de las organizaciones. Hay estudios que demuestran que más de la mitad de todas las redes clase B tienen menos de 50 *hosts*. Una red clase C habría bastado, pero sin duda todas las organizaciones que solicitaron una dirección clase B pensaron que un día el campo de *hosts* de 8 bits les quedaría pequeño. En retrospectiva, podría haber sido mejor que las redes clase C usaran 10 bits en lugar de 8 para el número de *host,* permitiendo 1022 *hosts* por red. De haber sido éste el caso, la mayoría de las organizaciones probablemente se habría conformado con una red clase C, y habría habido medio millón de ellas (en vez de sólo 16,384 redes clase B).

Es duro culpar a los diseñadores de Internet por no haber proporcionado más (y más pequeñas) direcciones de clase B. En el momento en que se tomó la decisión de crear las tres clases, Internet era una red de investigación que conectaba las principales universidades de investigación en Estados Unidos (más un número muy pequeño de compañías y sitios del ejército que hacían investigación de redes). En esa época nadie percibía que Internet llegaría a ser un sistema de comunicación de mercado masivo que rivalizaría con la red telefónica. También en esa época alguien dijo sin dudar: “Estados Unidos tiene alrededor de 2000 universidades. Aun cuando todas se

conectaran a Internet y muchas universidades en otros países también, nunca llegaremos a 16,000 puesto que no hay tantas universidades en todo el mundo. Además, como el número de *host* es un número integral de bytes acelera el procesamiento del paquete".

Sin embargo, si la división hubiera asignado 20 bits al número de red de clase B, habría surgido más rápidamente otro problema: la explosión de tablas de enrutamiento. Desde el punto de vista de los enrutadores, el espacio de direcciones IP es una jerarquía de dos niveles, con números de red y números de *host*. Los enrutadores no tienen que saber de todos los *hosts*, pero sí tienen que saber de todas las redes. Si se usaran medio millón de redes de clase C, todos los enrutadores de Internet necesitarían una tabla con medio millón de entradas, una por red, indicando la línea a usar para llegar a esa red, así como otra información.

Actualmente, el almacenamiento de medio millón de entradas tal vez es posible, aunque caro en los enrutadores críticos que guardan las tablas en la RAM estática de las tarjetas de E/S. Un problema más serio es que la complejidad de diferentes algoritmos relacionados con el mantenimiento de tablas crece con una tasa mayor que la lineal. Pero aún, mucho del *software* y *firmware* existente de los enrutadores se diseñó en un momento en que Internet tenía 1000 redes conectadas, y tener 10,000 redes parecía estar a décadas de distancia. Las decisiones de diseño tomadas entonces ya no tienen nada de óptimas.

Además, diversos algoritmos de enrutamiento requieren que cada enrutador transmita sus tablas periódicamente (por ejemplo, protocolos de vector de distancia). Cuanto más grandes sean las tablas, más probabilidad habrá de que algunas partes se pierdan en el camino, dando pie a datos incompletos en el otro lado y posiblemente a inestabilidades de enrutamiento.

El problema de las tablas de enrutamiento podría haberse resuelto usando una jerarquía más. Por ejemplo, podría haber servido hacer que cada dirección IP tuviera un campo de país, estado, ciudad, red y *host*. Entonces cada enrutador sólo necesitaría saber cómo llegar a los países, a los estados o provincias de su propio país, a las ciudades de su estado o provincia y a las redes de su ciudad. Por desgracia, esta solución requeriría bastante más de 32 bits para las direcciones de IP y usaría ineficientemente las direcciones (Liechtenstein tendría tantos bits como Estados Unidos).

En resumen, la mayoría de las soluciones resuelven un problema pero crean uno nuevo. La solución que se implementó y que dio a Internet un respiro es el **CIDR** (**Enrutamiento Interdominios sin Clases**)**.** El concepto básico del CIDR, que se describe en el RFC 1519, es asignar las direcciones IP restantes en bloques de tamaño variable, independientemente de las clases. Si un sitio necesita, digamos, 2000 direcciones, se le da un bloque de 2048 direcciones con un límite de 2048 bytes.

Eliminar las clases hace más complicado el reenvío. En el sistema antiguo con clases el reenvío se hacía de la siguiente manera: cuando un paquete llegaba a un enrutador, una copia de la dirección IP se desplazaba 28 bits a la derecha para obtener un número de clase de 4 bits. Entonces una rama de 16 vías ordenaba los paquetes en A, B, C y D (si lo soportaba), con ocho de los casos para la clase A, cuatro de los casos para la clase B, dos de los casos para la clase C, uno para D y otro para E. A continuación, el código para cada clase enmascaraba los números de red de 8, 16 o 24 bits y los alineaba a la derecha en una palabra de 32 bits. Entonces se buscaba el número de la red en la tabla A, B o C, por lo común mediante indexación en las redes A y B y apli-

cando *hash* en las redes C. Una vez que se encontrara la entrada, se podría buscar la línea de salida y remitir el paquete.

Con CIDR, este algoritmo sencillo ya no funciona. En cambio, cada entrada de tabla de enrutamiento se extiende para darle una máscara de 32 bits. De esta manera, ahora hay una sola tabla de enrutamiento para todas las redes que consten de un arreglo de tres variables (dirección IP, máscara de subred, línea saliente). Cuando llega un paquete, primero se extrae su dirección de destino IP. Luego (conceptualmente) se analiza la tabla de enrutamiento entrada por entrada, enmascarando la dirección de destino y comparándola con la entrada de la tabla buscando una correspondencia. Es posible que coincidan entradas múltiples (con diferentes longitudes de máscara de subred), en cuyo caso se usa la máscara más larga. De esta manera, si hay una coincidencia para una máscara /20 y una máscara /24, se usa la entrada /24.

Se han ideado algoritmos complejos para acelerar el proceso de coincidencia de dirección (Ruiz-Sánchez y cols., 2001). Los enrutadores comerciales usan chips VLSI programados con estos algoritmos.

Para hacer más claro este proceso de comparación, consideremos un ejemplo en el cual se dispone de millones de direcciones, empezando en 194.24.0.0. Suponga que la Universidad de Cambridge necesita 2048 direcciones y se le asignan las direcciones 194.24.0.0 a 194.24.7.255, junto con la máscara 255.255.248.0. Enseguida, la Universidad de Oxford solicita 4096 direcciones. Puesto que un bloque de 4096 direcciones debe caer en un límite de 4096 bytes, no pueden asignarse las direcciones que comienzan en 194.24.8.0. En cambio, Oxford recibe 194.24.16.0 a 194.24.31.255, junto con la máscara de subred 255.255.240.0. Ahora la Universidad de Edimburgo solicita 1024 direcciones, y se le asignan las direcciones 194.24.8.0 a 194.24.11.255 y la máscara 255.255.252.0. Estas asignaciones se resumen en la figura 5.59.

Universidad	Primera dirección	Segunda dirección	Cantidad	Escrito como
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edimburgo	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Disponible)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

Figura 5-59. Un conjunto de asignaciones de direcciones IP.

Las tablas de enrutamiento de todo el mundo se actualizan con tres entradas, que contienen una dirección base y una máscara de subred. Estas entradas (en binario) son:

Dirección	Máscara
C: 11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000
E: 11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000
O: 11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000

Ahora considere lo que ocurre cuando llega un paquete dirigido a 194.24.17.4, que en binario se representa con la siguiente cadena de 32 bits:

11000010 00011000 00010001 00000100

Primero se le hace un AND booleano con la máscara de Cambridge para obtener

11000010 00011000 00010000 00000000

Este valor no es igual a la dirección base de Cambridge, por lo que vuelve a hacerse un AND con la máscara de Edimburgo para obtener

11000010 00011000 00010000 00000000

Este valor no coincide con la dirección base de Edimburgo, por lo que se intenta con Oxford, obteniendo

11000010 00011000 00010000 00000000

Este valor coincide con la base de Oxford. Si no se encuentran más correspondencias recorriendo la tabla, se usa la entrada Oxford y se envía el paquete junto con la línea denominada en él.

Ahora veamos estas tres universidades desde el punto de vista de un enrutador en Omaha, Nebraska que tiene sólo cuatro líneas de salida: Minneapolis, Nueva York, Dallas y Denver. Cuando el software del enrutador tiene las tres nuevas entradas ahí, observa que puede combinar las tres entradas en una sola **entrada agregada** 194.24.0.0/19 con una dirección binaria y una submáscara como sigue:

11000010 0000000 00000000 00000000 11111111 11111111 11100000 00000000

Esta entrada envía todos los paquetes destinados para cualquiera de las tres universidades de Nueva York. Agregando las tres entradas, el enrutador de Omaha ha reducido en dos entradas el tamaño de su tabla.

Si Nueva York tiene una sola línea a Londres para todo el tráfico del Reino Unido, también puede usar una entrada agregada. Sin embargo, si tiene líneas separadas para Londres y Edimburgo, entonces debe tener tres entradas separadas. La agregación se usa en gran medida en Internet para reducir el tamaño de las tablas de enrutador.

Como una nota final a este ejemplo, la entrada de ruta agregada en Omaha envía también a Nueva York paquetes para las direcciones no asignadas. Si en verdad las direcciones no están asignadas esto no afecta, porque no se supone que ocurran. Sin embargo, si después se asignan a una compañía en California, se necesitará una entrada adicional, 194.24.12.0/22, para estas direcciones.

NAT—Traducción de Dirección de Red

Las direcciones IP son escasas. Un ISP podría tener una dirección de /16 (anteriormente de clase B), dándole 65,534 números de *host*. Si tiene más clientes que esos, tiene un problema. Para

clientes propios con las conexiones de línea conmutada, una manera de resolver el problema es asignar dinámicamente una dirección IP a una computadora cuando ésta llama e inicia la sesión y tomar de vuelta la dirección IP cuando se termina la sesión. De esta manera, una sola dirección de /16 puede manejar hasta 65,534 usuarios activos lo que es probablemente bastante bueno para un ISP con varios cientos de miles de clientes. Cuando termina la sesión, la dirección IP se reasigna a otra visita. Mientras esta estrategia trabaja bien para un ISP con un número moderado de usuarios propios, falla para ISPs que sirven sobre todo a clientes comerciales.

El problema es que los clientes comerciales esperan estar en línea continuamente durante las horas hábiles. Tanto los negocios pequeños —por ejemplo, agencias de viaje de tres personas— como las corporaciones grandes tienen varias computadoras conectadas por una LAN. Algunas computadoras son PCs de empleados; otras pueden ser servidores Web. Generalmente hay un enrutador en la LAN que se conecta al ISP por una línea rentada para proporcionar conectividad continua. Este arreglo significa que cada computadora debe tener todo el día mucho tiempo su propia dirección IP. De hecho, el número total de computadoras poseído por todos sus clientes comerciales combinados no puede exceder el número de direcciones IP que el ISP tiene. Para una dirección de /16, esto limita el número total de computadoras a 65,534. Para un ISP con decenas de miles de clientes comerciales, este límite se excederá rápidamente.

Para empeorar las cosas, más y más usuarios caseros se suscriben a ADSL o Internet por cable. Dos de los rasgos de estos servicios son: (1) el usuario consigue una dirección IP permanente y (2) no hay ningún cargo por la conexión (sólo un pago fijo mensual), por lo que los usuarios de ADSL y de cable se quedan registrados de manera permanente. Este desarrollo se agrega a la escasez de direcciones IP. Asignar direcciones IP conforme se solicitan (dinámicamente) como se hace con los usuarios de conexión por línea conmutada es inútil porque en cualquier momento el número de direcciones IP en uso puede ser muchas veces el número que el ISP posee.

Y sólo para hacerlo un poco más complicado, muchos ADSL y usuarios de cable tienen dos o más computadoras en casa, a menudo una para cada integrante de la familia y todos quieren estar en línea todo el tiempo usando la única dirección IP que su ISP les ha dado. La solución aquí es conectar todas las PCs a través de una LAN y poner un enrutador. Desde el punto de vista del ISP, ahora la familia se parece a un negocio comercial pequeño con un puñado de computadoras. Bienvenido a Pérez, Inc.

El problema de quedarse sin direcciones IP no es un problema teórico que podría ocurrir en algún punto en el futuro distante. Está sucediendo aquí y ahora mismo. La solución a largo plazo es que todo Internet emigre a IPv6, que tiene direcciones de 128 bits. Esta transición se está dando despacio, pero todo el proceso estará completo en cuestión de años. Como consecuencia, algunas personas sentían que se necesitaba un arreglo rápido a corto plazo. Este arreglo surgió en la forma de la **Traducción de Dirección de Red** (**NAT**) que se describe en el RFC 3022 y que resumiremos más adelante. Para información adicional, vea (Dutcher, 2001).

La idea básica de NAT es asignar una sola dirección IP a cada compañía (o a lo sumo, un número pequeño) para el tráfico de Internet. *Dentro* de la compañía, cada computadora tiene una dirección IP única que se usa para enrutar el tráfico interno. Sin embargo, cuando un paquete sale de la compañía y va al ISP, se presenta una traducción de dirección. Para hacer posible este esquema

los tres rangos de direcciones IP se han declarado como privados. Las compañías pueden usarlos internamente cuando lo deseen. La única regla es que ningún paquete que contiene estas direcciones puede aparecer en la propia Internet. Los tres rangos reservados son:

```
10.0.0.0      – 10.255.255.255/8      (16,777,216 hosts)
172.16.0.0    – 172.31.255.255/12     (1,048,576 hosts)
192.168.0.0 – 192.168.255.255/16      (65,536 hosts)
```

El primer rango proporciona 16,777,216 direcciones (excepto 0 y –1, como de costumbre) y es la opción usual de la mayoría de las compañías, incluso si no necesitan tantas direcciones.

El funcionamiento de NAT se muestra en la figura 5-60. Dentro de las instalaciones de la compañía, cada máquina tiene una dirección única de la forma 10.*x.y.z.* Sin embargo, en este ejemplo cuando un paquete sale de las instalaciones de la compañía, pasa a través de una **caja NAT** que convierte la dirección interna de origen de IP, 10.0.0.1 en la figura, a la verdadera dirección IP de la compañía, 198.60.42.12. A menudo, la caja NAT se combina en un solo dispositivo con un *firewall* (servidor de seguridad) que proporciona seguridad controlando cuidadosamente lo que entra y sale de la compañía. En el capítulo 8 estudiaremos los servidores de seguridad. También es posible integrar la caja NAT en el enrutador de la compañía.

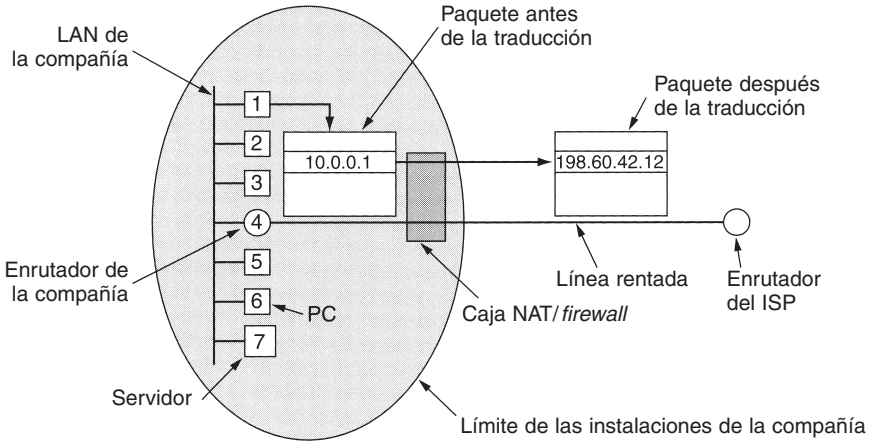


Figura 5-60. Colocación y funcionamiento de una caja NAT.

Hasta ahora hemos ignorado un pequeño detalle: cuando la respuesta vuelve (por ejemplo, un servidor Web), se dirige naturalmente a 198.60.42.12, por lo que, ¿cómo sabe ahora la caja NAT con qué dirección se reemplaza? Aquí está el problema con NAT. Si hubiera un campo de repuesto en el encabezado IP, ese campo podría usarse para guardar el registro del emisor real, pero sólo queda 1 bit sin usar. En principio, podría crearse una nueva opción para mantener la verdadera dirección del origen, pero haciéndolo así se requeriría cambiar el código IP en todas las máquinas de todo Internet para manejar la nueva opción. Ésta no es una alternativa prometedora para un arreglo rápido.

Lo que realmente pasó es lo siguiente. Los diseñadores de NAT observaron que la mayoría de paquetes IP lleva cargas útiles de TCP o UDP. Cuando estudiemos TCP y UDP en el capítulo 6, veremos que los dos tienen encabezados que contienen un puerto de origen y un puerto de destino. Más adelante explicaremos los puertos TCP, pero esa misma explicación es válida para los puertos UDP. Los puertos son enteros de 16 bits que indican dónde empieza y dónde acaba la conexión TCP. Estos puertos proporcionan el campo requerido para hacer que NAT funcione.

Cuando un proceso quiere establecer una conexión TCP con un proceso remoto, se conecta a un puerto TCP sin usar en su propia máquina. Éste se conoce como **puerto de origen** y le indica al código TCP codifican dónde enviar paquetes entrantes que pertenecen a esta conexión. El proceso también proporciona un **puerto de destino** para decir a quién dar los paquetes en el lado remoto. Los puertos 0-1023 se reservan para servicios bien conocidos. Por ejemplo, 80 es el puerto usado por los servidores Web, para que los clientes remotos puedan localizarlos. Cada mensaje TCP saliente contiene un puerto de origen y un puerto de destino. Juntos, estos puertos sirven para identificar los procesos que usan la conexión en ambos extremos.

Una analogía aclara el uso de los puertos. Imagine una compañía con un solo número de teléfono principal. Cuando las personas llaman al número principal, se encuentran con un operador que pregunta qué extensión quieren y entonces los ponen en esa extensión. El número principal es análogo a la dirección IP de la compañía y las extensiones en ambos extremos son análogas a los puertos. Los puertos son de 16 bits extra de dirección que identifican qué proceso obtiene cuál paquete entrante.

Usando el campo *Puerto de origen*, podemos resolver nuestro problema de conversión. Siempre que un paquete saliente entra en la caja NAT, la dirección de origen 10.*x.y.z* se reemplaza por la verdadera dirección IP de la compañía. Además, el campo *Puerto de origen* TCP se reemplaza por un índice en la tabla de traducción de la entrada 65,536 de la caja NAT. Esta entrada de la tabla contiene el puerto de origen y la dirección IP originales. Finalmente, las sumas de verificación de los encabezados IP y TCP se recalculan e insertan en el paquete. Es necesario reemplazar el *Puerto de origen* porque podría ocurrir que ambas conexiones de las máquinas 10.0.0.1 y 10.0.0.2 usaran el puerto 5000, por ejemplo, así que el *Puerto de origen* no basta para identificar el proceso de envío.

Cuando un paquete llega a la caja NAT desde el ISP, el *Puerto de origen* en el encabezado TCP se extrae y utiliza como un índice en la tabla de traducción de la caja NAT. Desde la entrada localizada, la dirección IP interna y el *Puerto de origen* TCP se extraen e insertan en el paquete. Luego, las sumas de verificación de IP y TCP se recalculan e insertan en el paquete. Entonces el paquete se pasa al enrutador de la compañía para su entrega normal utilizando la dirección 10.*x.y.z*.

También puede usarse NAT para aliviar la escasez de IP para usuarios de cable y ADSL. Cuando el ISP le asigna una dirección a cada usuario, usa direcciones 10.*x.y.z*. Cuando los paquetes de máquinas de usuario salen del ISP y entran en la Internet principal, atraviesan una caja NAT que los traduce a la verdadera dirección de Internet del ISP. En el camino de regreso, los paquetes sufren la conversión inversa. En este caso, para el resto de Internet, el ISP y sus usuarios caseros de cable y ADSL son como una compañía grande.

Aunque esta clase de esquema resuelve el problema, muchas personas en la comunidad de IP lo consideran a primera vista como una abominación. Brevemente resumidas, aquí están algunas de las objeciones. Primero, NAT viola el modelo arquitectónico de IP que establece que cada dirección IP identifica una sola máquina globalmente. Toda la estructura del software de Internet se basa en este hecho. Con NAT, las miles de máquinas pueden usar (y lo hacen) la dirección 10.0.0.1.

Segundo, NAT cambia a Internet de una red sin conexión a un tipo de red orientada a la conexión. El problema es que la caja NAT debe mantener la información (la conversión) para cada conexión que la atraviesa. Mantener el estado de la conexión es una propiedad de las redes orientadas a la conexión, no de las no orientadas. Si la caja NAT se cae y se pierde su tabla de traducción, todas sus conexiones TCP se destruyen. En ausencia de NAT, la destrucción de los enrutadores no afecta al TCP. El proceso de envío apenas queda fuera unos segundos y retransmite todos los paquetes cuya recepción no se haya confirmado. Con NAT, Internet es tan vulnerable como una red de circuitos conmutados.

Tercero, NAT viola la regla más fundamental de los protocolos de capas: la capa *k* no puede hacer ninguna suposición de en qué capa k + 1 ha puesto el campo de carga útil. Este principio básico está ahí para mantener independientes las capas. Si TCP se actualiza después a TCP-2, con un diseño de encabezado diferente (por ejemplo, puertos de 32 bits), NAT fallará. Toda la idea de los protocolos de capas es asegurar que los cambios en una capa no requieran cambios en otras capas. NAT destruye esta independencia.

Cuarto, en Internet no se exige que los procesos utilicen TCP o UDP. Si un usuario en la máquina *A* decide usar algún nuevo protocolo de transporte para hablar con un usuario en la máquina *B* (por ejemplo, para una aplicación multimedia), la introducción de una caja NAT hará que la aplicación falle porque la caja NAT no podrá localizar el *Puerto de origen* TCP correctamente.

Quinto, algunas aplicaciones insertan direcciones IP en el cuerpo del texto. El receptor extrae estas direcciones y las usa. Puesto que NAT no sabe nada sobre estas direcciones, no las puede reemplazar, de modo que fallará cualquier intento de usarlas en el extremo remoto. El **FTP** (**Protocolo de Transferencia de Archivos**) estándar funciona de esta manera y puede fallar en presencia del NAT a menos que se tomen precauciones especiales. Del mismo modo, el protocolo de telefonía H.323 de Internet (que estudiaremos en el capítulo 7) tiene esta propiedad y puede fallar en presencia del NAT. Puede ser posible arreglar NAT para que trabaje con H.323, pero no es una buena idea tener que arreglar el código en la caja NAT cada vez que surge una nueva aplicación.

Sexto, debido a que el campo *Puerto de origen* de TCP es de 16 bits, a lo sumo se pueden asignar 65,536 máquinas hacia una dirección IP. En realidad, el número es ligeramente menor porque los primeros 4096 puertos se reservan para usos especiales. Sin embargo, si hay varias direcciones IP disponibles, cada una puede manejar 61,440 máquinas.

En el RFC 2993 se explican éstos y otros problemas con NAT. En general, los antagonistas de NAT dicen que arreglando el problema de las direcciones IP insuficientes de esta manera temporal y poco elegante, la presión de implementar la solución real, es decir la transición a IPv6, se reduce, y el problema persiste.

5.6.3 Protocolos de Control en Internet

Además del IP que se usa para transferencia de datos, Internet tiene algunos protocolos de control que se usan en la capa de redes, como ICMP, ARP, RARP, BOOTP y DHCP. En esta sección veremos cada uno a su vez.

Protocolo de Mensajes de Control en Internet

Los enrutadores supervisan estrechamente el funcionamiento de Internet. Cuando ocurre algo inesperado, el **Protocolo de Mensajes de Control en Internet** (**ICMP**) informa del evento, que también se utiliza para probar Internet. Hay definidos alrededor de una docena de tipos de mensajes ICMP. Los más importantes se listan en la figura 5-61. Cada tipo de mensaje ICMP se encapsula en un paquete IP.

Tipo de mensaje	Descripción
Destination unreachable	El paquete no se pudo entregar
Time exceeded	Campo de tiempo de vida = 0
Parameter problem	Campo de encabezado no válido
Source quench	Paquete regulador
Redirect	Enseña a un enrutador sobre geografía
Echo	Pregunta a una máquina si está viva
Echo reply	Sí, estoy viva
Timestamp request	Misma que solicitud de eco, pero con marca de tiempo
Timestamp reply	Misma que respuesta de eco, pero con marca de tiempo

Figura 5-61. Los principales tipos de mensaje ICMP.

El mensaje DESTINATION UNREACHABLE se usa cuando la subred o un enrutador no pueden localizar el destino o cuando un paquete con el bit *DF* no puede entregarse porque una red de "paquete pequeño" se posiciona en la ruta.

El mensaje TIME EXCEEDED se envía cuando un paquete se cae porque su contador ha llegado a cero. Este evento es un síntoma de que los paquetes se están repitiendo, que hay una congestión enorme, o que los valores del cronómetro se han fijado demasiado bajos.

El mensaje PARAMETER PROBLEM indica que se ha descubierto un valor ilegal en un campo de encabezado. Este problema indica un error en el software de IP del *host* que envía o posiblemente en el software de un enrutador de tránsito.

El mensaje SOURCE QUENCH se utilizaba anteriormente para regular a los *hosts* que estaban enviando demasiados paquetes. Se esperaba que cuando un *host* recibiera este mensaje redujera la velocidad. Se usa cada vez menos porque cuando ocurre la congestión, estos paquetes tienden a agravar más la situación. Ahora el control de congestión en Internet se hace sobre todo en la capa de transporte; lo estudiaremos en detalle en el capítulo 6.

El mensaje REDIRECT se usa cuando un enrutador se percata de que un paquete parece estar mal enrutado. Lo utiliza el enrutador para avisar al *host* emisor sobre el probable error.

Los mensajes ECHO y ECHO REPLY se utilizan para ver si un destino dado es alcanzable y está vivo. Se espera que el destino envíe de vuelta un mensaje ECHO REPLY luego de recibir el mensaje ECHO. Los mensajes TIMESTAMP REQUEST y TIMESTAMP REPLY son similares, sólo que el tiempo de la llegada del mensaje y la salida de la contestación se graban en la contestación. Esta característica se usa para medir el rendimiento de la red.

Además de estos mensajes, se han definido otros. La lista en línea se conserva ahora en *www.iana.org/assignments/icmp-parameters.*

ARP—Protocolo de Resolución de Direcciones

Aunque en Internet cada máquina tiene una (o más) direcciones IP, en realidad éstas no pueden usarse para enviar paquetes porque el hardware de capa de enlace de datos no entiende las direcciones de Internet. Hoy día, la mayoría de los *hosts* en las compañías y universidades se une a una LAN por una tarjeta de red que sólo entiende direcciones LAN. Por ejemplo, cada tarjeta Ethernet viene provista de fábrica con una dirección Ethernet de 48 bits. Los fabricantes de tarjetas Ethernet solicitan un bloque de direcciones a una autoridad central para asegurar que dos tarjetas no tengan la misma dirección (para evitar los conflictos de si las dos tarjetas deban aparecer en la misma LAN). Las tarjetas envían y reciben tramas basadas en direcciones Ethernet de 48 bits. No saben nada de direcciones IP de 32 bits.

La pregunta ahora es: ¿cómo se convierten direcciones IP en direcciones de capa de enlace de datos, como Ethernet? Para explicar cómo funciona esto, veamos el ejemplo de la figura 5-62 en que se ilustra una universidad pequeña con algunas redes de clase C (ahora llamadas /24). Aquí tenemos dos Ethernets, una en el Depto. de Informática, con dirección IP 192.31.65.0 y otra en Ingeniería Eléctrica, con dirección IP 192.31.63.0. Éstas están conectadas por un anillo de red dorsal del campus (por ejemplo, FDDI) con la dirección IP 192.31.60.0. Cada máquina en una Ethernet tiene una dirección única de Ethernet, etiquetadas de *E1* a *E6*, y cada máquina en el anillo de FDDI tiene una dirección de FDDI, etiquetada de *F1* a *F3*.

Empezaremos viendo cómo un usuario en el *host* 1 envía un paquete a un usuario en el *host* 2. Supongamos que el emisor sabe el nombre del receptor pretendido, posiblemente algo como *mary@eagle.cs.uni.edu.* El primer paso es encontrar la dirección IP para el *host* 2, conocido como *eagle.cs.uni.edu.* Esta consulta la realiza el Sistema de Nombres de Dominio que estudiaremos en el capítulo 7. De momento, asumiremos que DNS devuelve la dirección IP al *host* 2 (192.31.65.5).

El software de la capa superior en el *host* 1 elabora ahora un paquete con 192.31.65.5 en el campo *Dirección de destino* y lo da a transmitir al software IP. Éste puede buscar la dirección y ver que el destino esté en su propia red, pero necesita alguna manera de encontrar la dirección Ethernet de destino. Una solución es tener un archivo de configuración en alguna parte del sistema

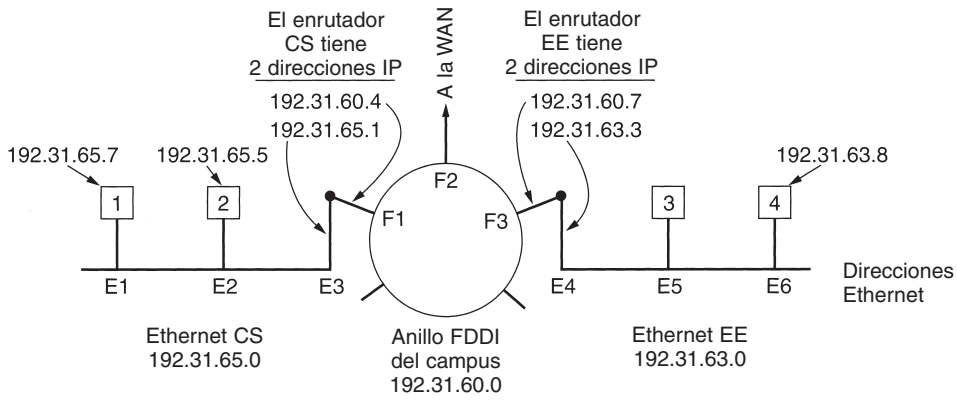


Figura 5-62. Tres redes /24 interconectadas: dos Ethernets y un anillo FDDI.

que relacione direcciones IP con direcciones Ethernet. Aun cuando esta solución es ciertamente posible, para las organizaciones con miles de máquinas, conservar todos estos archivos actualizados propicia errores y consume mucho tiempo.

Una mejor solución es que el *host* 1 dé salida a un paquete de difusión hacia Ethernet preguntando: ¿quién posee la dirección IP 192.31.65.5? La difusión llegará a cada máquina en Ethernet 192.31.65.0, y cada una verificará su dirección IP. Al *host* 2 le bastará responder con su dirección de Ethernet (*E2*). De esta manera, el *host* 1 aprende que esa dirección IP 192.31.65.5 está en el *host* con la dirección Ethernet *E2*. El protocolo utilizado para hacer esta pregunta y obtener la respuesta se llama **Protocolo de Resolución de Direcciones (ARP).** Casi cada máquina en Internet lo ejecuta. La definición de ARP está en el RFC 826.

La ventaja de usar ARP en lugar de archivos de configuración es la sencillez. El gerente de sistemas sólo tiene que asignar a cada máquina una dirección IP y decidir respecto de las máscaras de subred. ARP hace el resto.

A estas alturas, el software IP en el *host* 1 crea una trama Ethernet dirigida a *E2,* pone el paquete IP (dirigido a 192.31.65.5) en el campo de carga útil, y lo descarga hacia la Ethernet. La tarjeta Ethernet del *host* 2 detecta esta trama, la reconoce como una trama para sí mismo, lo recoge, y provoca una interrupción. El controlador de Ethernet extrae el paquete IP de la carga útil y lo pasa al software IP, que ve que esté direccionado correctamente y lo procesa.

Se pueden hacer varias optimizaciones para que ARP trabaje con más eficiencia. Para empezar, una vez que una máquina ha ejecutado ARP, guarda el resultado en caso de tener que ponerse en poco tiempo de nuevo en contacto con la misma máquina. La siguiente vez encontrará la correspondencia en su propio caché, eliminando así la necesidad de una segunda difusión. En muchos casos el *host* 2 necesitará devolver una respuesta, forzando, también, a que se ejecute el ARP para determinar la dirección Ethernet del emisor. Esta difusión de ARP puede evitarse teniendo el *host* 1

que incluir su correspondencia IP a Ethernet en el paquete ARP. Cuando la difusión de ARP llega al *host* 2, se introduce (192.31.65.7, E1) en el caché ARP del *host* 2 para uso futuro. De hecho, todas las máquinas en Ethernet pueden introducir esta correspondencia en su caché ARP.

Otra optimización más es que cada máquina difunda su correspondencia cuando arranca. Por lo general, esta difusión se hace en forma de un ARP que busca su propia dirección IP. No debe haber una respuesta, pero un efecto lateral de la difusión es hacer una entrada en el caché ARP de todas las máquinas. Si llega (inesperadamente) una respuesta, es que la misma dirección IP se ha asignado a dos máquinas. La más reciente debe informar al gerente de sistemas y no arrancar.

Para permitir que las correspondencias cambien, por ejemplo, cuando una tarjeta Ethernet falla y se reemplaza con una nueva (y, por lo tanto, una nueva dirección Ethernet), las entradas en el caché ARP deben expirar en unos cuantos minutos.

Ahora veamos de nuevo la figura 5-62. Esta vez el *host* 1 quiere enviar un paquete al *host* 4 (192.31.63.8). Si utiliza ARP fallará porque el *host* 4 no verá la difusión (los enrutadores no envían difusiones a nivel Ethernet). Hay dos soluciones. Primera, podría configurarse el enrutador CS para que responda a las solicitudes de ARP de la red 192.31.63.0 (y posiblemente del otras redes locales). En este caso, el *host* 1 introducirá (192.31.63.8, E3) en el caché ARP y enviará felizmente todo el tráfico del *host* 4 al enrutador local. Esta solución se llama **proxy ARP.** La segunda solución es hacer que el *host* 1 vea de inmediato que el destino está en una red remota y simplemente envíe todo ese tráfico a una dirección Ethernet predefinida que se ocupe de todo el tráfico remoto, en este caso *E3.* Esta solución no requiere que el enrutador CS sepa a qué redes remotas está sirviendo.

De cualquier modo, lo que sucede es que el *host* 1 empaca el paquete IP en el campo de carga útil de una trama Ethernet dirigida a *E3*. Cuando el enrutador CS obtiene la trama Ethernet, retira el paquete IP del campo de carga útil y busca la dirección IP en sus tablas de enrutamiento. Descubre que se supone que los paquetes para la red 192.31.63.0 van al enrutador 192.31.60.7. Si aún no conoce la dirección FDDI de 192.31.60.7, transmite un paquete ARP al anillo y aprende que su dirección del anillo es *F3.* Inserta entonces el paquete en el campo de carga útil de una trama FDDI dirigido a *F3* y la coloca en el anillo.

En el enrutador EE, el controlador de FDDI retira el paquete del campo de carga útil y lo da al software IP, que ve que necesita enviar el paquete a 192.31.63.8. Si esta dirección IP no está en su caché ARP, transmite una solicitud de ARP en la Ethernet EE y aprende que la dirección de destino es *E6,* por lo que construye una trama Ethernet dirigida a *E6*, pone el paquete en el campo de carga útil y lo envía a través de Ethernet. Cuando la trama Ethernet llega al *host* 4, se extrae el paquete de la trama y se pasa al software IP para su procesamiento.

Ir del *host* 1 a una red distante a través de una WAN funciona esencialmente de la misma manera, sólo que esta vez las tablas del enrutador CS le dicen que utilice el enrutador de WAN cuya dirección FDDI es *F2.*

RARP, BOOTP y DHCP

ARP resuelve el problema de encontrar qué dirección Ethernet corresponde a una dirección IP dada. A veces se tiene que resolver el problema inverso: dada una dirección Ethernet, ¿cuál es la dirección IP correspondiente? En particular, este problema ocurre cuando se inicializa una estación de trabajo sin disco. Dicha máquina recibirá normalmente la imagen binaria de su sistema operativo desde un servidor de archivos remoto. ¿Pero cómo aprende su dirección IP?

La primera solución inventada fue usar el **RARP** (**Protocolo de Resolución de Dirección de Retorno**) (su definición está en el RFC 903). Este protocolo permite que una estación de trabajo recientemente inicializada transmita su dirección Ethernet y diga: "Mi dirección Ethernet de 48 bits es 14.04.05.18.01.25. ¿Alguien allá afuera conoce mi dirección IP?" El servidor RARP ve esta solicitud, busca la dirección Ethernet en sus archivos de configuración y devuelve la dirección IP correspondiente.

Usar RARP es mejor que incrustar una dirección IP en la imagen de memoria porque esto permite usar la misma imagen en todas las máquinas. Si la dirección IP se incrustara en la imagen, cada estación de trabajo necesitaría su propia imagen.

Una desventaja de RARP es que usa una dirección de destino de todos los 1s (de difusión limitada) para llegar al servidor RARP. Sin embargo, dichas difusiones no las envían los enrutadores, por lo que se necesita un servidor RARP en cada red. Para resolver este problema, se inventó un protocolo de arranque alternativo llamado **BOOTP.** A diferencia de RARP, el BOOTP usa mensajes UDP que se envían a través de los enrutadores. También proporciona información adicional a una estación de trabajo sin disco, incluso la dirección IP del servidor de archivos que contiene la imagen de memoria, la dirección IP del enrutador predeterminado y la máscara de subred que debe usar. BOOTP se describe en los RFCs 951, 1048 y 1084.

Un problema serio con BOOTP es que requiere configuración manual de tablas para relacionar una dirección IP con una dirección Ethernet. Cuando se agrega un nuevo *host* a una LAN, no se puede usar el BOOTP hasta que un administrador le haya asignado una dirección IP e introducido manualmente sus direcciones IP y Ethernet en las tablas de configuración de BOOTP. Para eliminar este paso conducente a errores, el BOOTP se extendió y se le dio un nuevo nombre: **DHCP** (**Protocolo de Configuración de *Host* Dinámico**)**.** DHCP permite asignación de dirección IP manual y automática. Se describe en los RFCs 2131 y 2132. En la mayoría de los sistemas, ha reemplazado a RARP y BOOTP.

Como RARP y BOOTP, DHCP se basa en la idea de un servidor especial que asigna direcciones IP a *hosts* que las requieren. Este servidor no necesita estar en la misma LAN que el *host* solicitante. Puesto que el servidor DHCP no se puede alcanzar por difusión, se necesita un **agente de retransmisión DHCP** en cada LAN, como se muestra en la figura 5-63.

Para encontrar su dirección IP, una máquina inicializada recientemente difunde un paquete DHCP DISCOVER. El agente de retransmisión DHCP de su LAN intercepta todas las difusiones DHCP. Cuando encuentra un paquete DHCP DISCOVER, envía el paquete mediante unidifusión al servidor

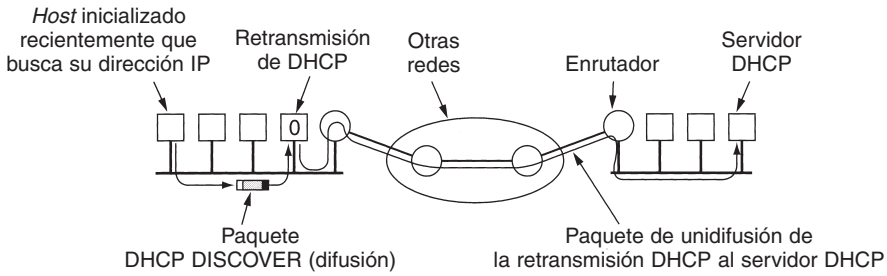


Figura 5-63. Funcionamiento de DHCP.

DHCP, posiblemente en una red distante. La única pieza de información que el agente de retransmisión necesita es la dirección IP del servidor DHCP.

Un problema que surge con la asignación automática de direcciones IP de un rango de direcciones, es por cuánto tiempo debe asignarse una dirección IP. Si un *host* deja la red y no devuelve su dirección IP al servidor DHCP, esa dirección se perderá permanentemente. Después de un periodo, pueden perderse muchas direcciones. Para impedir que eso pase, la asignación de dirección IP puede ser por un periodo fijo, una técnica llamada **arrendamiento.** Simplemente, antes de que expire el arriendo, el *host* debe pedirle una renovación al DHCP. Si no hace una solicitud o ésta se le niega, el *host* ya no puede usar la dirección IP que se le dio antes.

5.6.4 OSPF—Protocolos de enrutamiento de puerta de enlace interior

Hemos terminado nuestro estudio de protocolos de control de Internet. Es tiempo para pasar al tema siguiente: el enrutamiento en Internet. Como lo mencionamos antes, Internet se compone de una gran cantidad de sistemas autónomos. Cada uno de ellos es manejado por una organización diferente y puede usar su propio algoritmo interno de enrutamiento. Por ejemplo, las redes internas de las compañías *X, Y* y *Z* que normalmente se ven como tres sistemas autónomos si los tres están en Internet. Los tres pueden usar internamente algoritmos de enrutamiento diferentes. No obstante, siguiendo los estándares incluso para enrutamiento interno, simplifica la implementación de los límites entre los sistemas autónomos y permite reutilizar el código. En esta sección estudiaremos el enrutamiento dentro de un sistema autónomo. En la siguiente veremos el enrutamiento entre sistemas autónomos. Un algoritmo de enrutamiento dentro de un sistema autónomo se llama **protocolo de puerta de enlace interior** (**IGP**)**;** un algoritmo para enrutamiento entre sistemas autónomos se llama **protocolo de puerta de enlace exterior** (**EGP**)**.**

El protocolo de puerta de enlace interior original de Internet era un protocolo de vector de distancia (RIP) basado en el algoritmo de Bellman-Ford heredado de ARPANET. Funcionó bien en sistemas pequeños, pero no así conforme los sistemas autónomos fueron más grandes. También padeció el problema de la cuenta hasta el infinito y la convergencia generalmente lenta, por lo que se reemplazó en mayo de 1979 por un protocolo de estado del enlace. En 1988, la Fuerza de Tarea de Ingeniería de Internet empezó el trabajo en un sucesor. Ese sucesor, llamado **OSPF** (**Abrir**

Primero la Ruta más Corta), se volvió una norma en 1990. Ahora la mayoría de vendedores de enrutadores lo apoyan, y se ha convertido en el protocolo de puerta de enlace interior principal. A continuación daremos un bosquejo de cómo funciona OSPF. Para la historia completa, vea el RFC 2328.

Dada la larga experiencia con otros protocolos de enrutamiento, el grupo que diseñó el nuevo protocolo tenía una larga lista de requisitos que cumplir. Primero, el algoritmo se tenía que publicar en la literatura abierta, de ahí la "O" inicial de OSPF. Una solución patentada poseída por una compañía no lo haría. Segundo, el nuevo protocolo tenía que apoyar una variedad de métrica de distancia, como la distancia física, retardo, etcétera. Tercero, tenía que ser un algoritmo dinámico, uno que se adaptara automática y rápidamente a los cambios de topología.

Cuarto, y esto era nuevo para OSPF, tenía que apoyar el enrutamiento con base en el tipo de servicio. El nuevo protocolo tenía que poder dirigir el tráfico en tiempo real de una manera y el resto del tráfico de una manera diferente. El protocolo IP tiene un campo *Tipo de servicio,* pero ningún protocolo de enrutamiento existente lo usó. Este campo estaba incluido en OSPF pero tampoco se usó, y finalmente lo eliminaron.

Quinto, y relacionado con el anterior, el nuevo protocolo tenía que balancear la carga, dividiéndola en líneas múltiples. La mayoría de los protocolos anteriores enviaba todos los paquetes por la mejor ruta. La mejor segunda ruta no se usó en absoluto. En muchos casos, dividir la carga en líneas múltiples ofrece un mejor desempeño.

Sexto, se necesitó apoyo para los sistemas jerárquicos. En 1988 Internet había crecido tanto que no se podía esperar que ningún enrutador conociera toda la topología. Se tuvo que diseñar el nuevo protocolo de enrutamiento para que ningún enrutador tuviera que conocerla.

Séptimo, se requirió una pizca de seguridad para impedir que los estudiantes bromistas engañaran a los enrutadores enviándoles falsa información de enrutamiento. Finalmente, se necesitó una previsión para tratar con enrutadores que se conectaban a Internet mediante un túnel. Los protocolos anteriores no manejaron bien este aspecto.

OSPF soporta tres tipos de conexiones y redes:

1. Las líneas punto a punto exactamente entre dos enrutadores.
2. Redes de multiacceso con difusión (por ejemplo, la mayoría de las LANs).
3. Redes de multiacceso sin difusión (por ejemplo, la mayoría de las WANs de paquetes conmutados).

Una red de **multiacceso** es la que puede tener múltiples enrutadores, cada uno de los cuales se puede comunicar directamente con todos los demás. Todas las LANs y WANs tienen esta propiedad. La figura 5-64(a) muestra un sistema autónomo que contiene los tres tipos de redes. Observe que en general los *hosts* no tienen un papel en OSPF.

OSPF funciona resumiendo la colección de redes reales, enrutadores y líneas en un grafo dirigido en el que a cada arco se asigna un costo (distancia, retardo, etcétera). Entonces calcula la ruta más corta con base en los pesos de los arcos. Una conexión en serie entre dos enrutadores se representa por un par de arcos, uno en cada dirección. Sus pesos pueden ser diferentes. Una red de

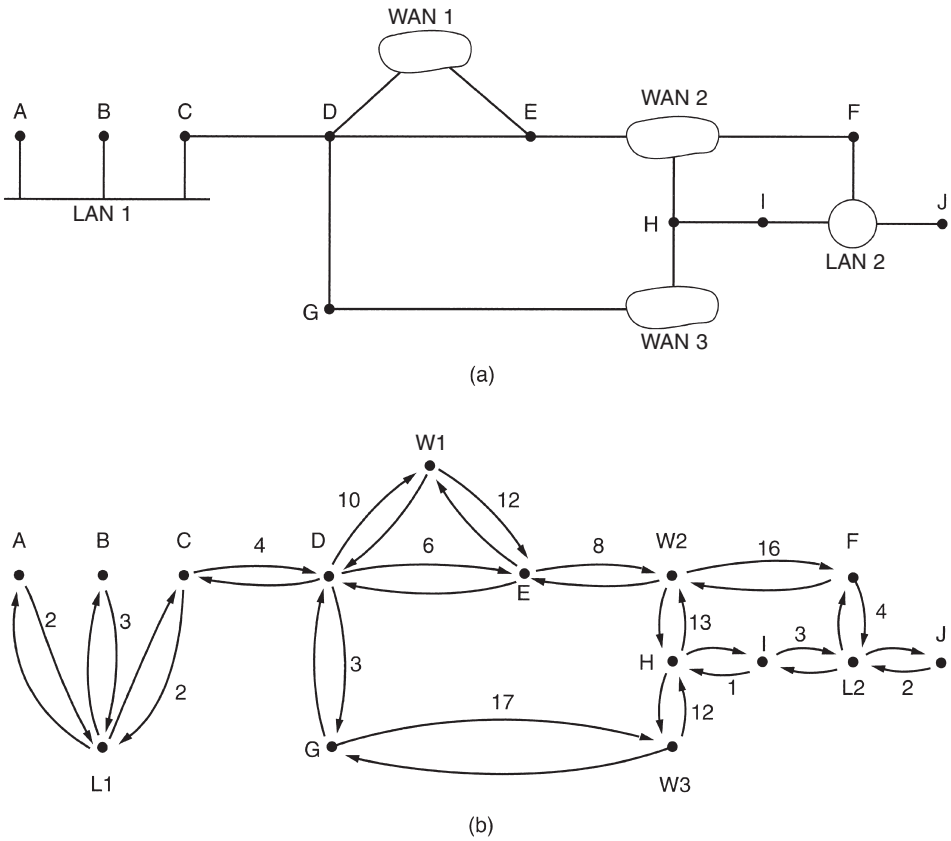


Figura 5-64. (a) Un sistema autónomo. (b) Representación gráfica de (a).

multiacceso se representa con un nodo para la red en sí más un nodo para cada enrutador. Los arcos del nodo de la red a los enrutadores tienen peso 0 y se omiten del grafo.

La figura 5-64(b) muestra la representación gráfica de la red de la figura 5-64(a). Los pesos son simétricos, a menos que se marcaran de otra manera. Lo que OSPF hace fundamentalmente es representar la red real como un grafo y entonces calcular el camino más corto de uno a otro enrutador.

Muchos de los sistemas autónomos en Internet son grandes por sí mismos y nada sencillos de administrar. OSPF les permite dividirlos en **áreas** numeradas donde un área es una red o un conjunto de redes inmediatas. Las áreas no se traslapan ni la necesidad es exhaustiva, es decir, algunos enrutadores no pueden pertenecer a área alguna. Un área es una generalización de una subred. Fuera de un área, su topología y detalles no son visibles.

Cada sistema autónomo tiene un área de **red dorsal,** llamada 0. Todas las áreas se conectan a la red dorsal, posiblemente por túneles, de modo que es posible entrar desde cualquier área en el sistema autónomo a cualquier otra área en el sistema autónomo mediante la red dorsal. En el grafo un túnel se representa como un arco y tiene un costo. Cada enrutador que se conecta a dos o

más áreas es parte de la red dorsal. Como con otras áreas, la topología de la red dorsal no es visible fuera de ésta.

Dentro de un área, cada enrutador tiene la misma base de datos del estado del enlace y ejecuta el mismo algoritmo de la ruta más corta. Su trabajo principal es calcular el camino más corto desde sí mismo a cualquier otro enrutador en el área, incluso el enrutador que se conecta a la red dorsal, de la que debe haber una por lo menos. Un enrutador que conecta dos áreas necesita las bases de datos para las dos áreas y debe ejecutar el algoritmo de la ruta más corta por separado.

Durante la operación normal, pueden necesitarse tres tipos de rutas: dentro del área, entre áreas y entre sistemas autónomos. Las rutas dentro del área son las más fáciles, puesto que el enrutador de origen ya conoce el camino más corto al enrutador de destino. El enrutamiento entre áreas siempre procede en tres pasos: va del origen a la red dorsal; va a través de la red dorsal al área de destino; va al destino. Este algoritmo fuerza una configuración de estrella en OSPF con la red dorsal actuando como concentrador y las otras áreas como rayos. Los paquetes se enrutan del origen al destino "como están". No se encapsulan ni se entunelan, a menos que vayan a un área cuya única conexión a la red dorsal sea un túnel. La figura 5-65 muestra parte de Internet con sistemas autónomos y áreas.

OSPF distingue cuatro clases de enrutadores:

1. Enrutadores internos que están totalmente dentro de un área.
2. Enrutadores de límite de área que conectan dos o más áreas.
3. Enrutadores de la red dorsal que están en la red dorsal.
4. Enrutadores fronterizos de sistemas autónomos que se comunican con los enrutadores de otros sistemas autónomos.

Estas clases se pueden traslapar. Por ejemplo, todos los enrutadores de límite de área forman parte de la red dorsal automáticamente. Además, un enrutador que está en la red dorsal pero no forma parte de cualquier otra área también es un enrutador interno. En la figura 5-65 se ilustran ejemplos de las cuatro clases de enrutadores.

Cuando un enrutador se inicia, envía mensajes HELLO en todas sus líneas punto a punto y los multidifunde en las LANs al grupo que contiene los enrutadores restantes. En las WANs, necesita alguna información de configuración para saber a quién contactar. A partir de las respuestas, cada enrutador aprende quiénes son sus vecinos. Todos los enrutadores en la misma LAN son vecinos.

OSPF trabaja intercambiando información entre enrutadores adyacentes, que no es lo mismo que entre enrutadores vecinos. En particular, es ineficaz tener cualquier enrutador en la LAN que se comunica con cualquier otro enrutador en la LAN. Para evitar esta situación, se elige un enrutador como **enrutador designado.** Se dice que es **adyacente** a todos los demás enrutadores en su LAN, e intercambia información con ellos. Los enrutadores vecinos que no son adyacentes no intercambian información entre sí. Un enrutador designado como respaldo siempre se guarda actualizado, para facilitar la transición en caso de que el primer enrutador designado se cayera y necesitara ser reemplazado de manera inmediata.

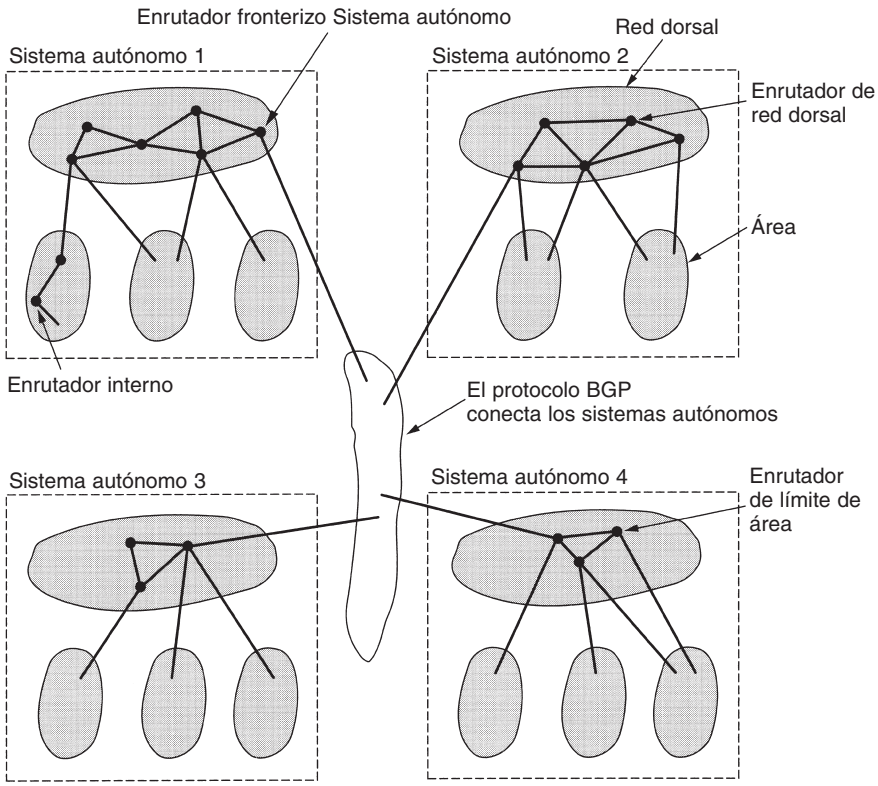


Figura 5-65. Relación entre sistemas autónomos, redes dorsales y áreas en OSPF.

Durante la operación normal, cada enrutador inunda periódicamente con mensajes LINK STATE UPDATE a cada uno de sus enrutadores adyacentes. Este mensaje da su estado y proporciona los costos usados en la base de datos topológica. Para hacerlos confiables, se confirma la recepción de los mensajes de inundación. Cada mensaje tiene un número de secuencia para que un enrutador pueda ver si un LINK STATE UPDATE entrante es más viejo o más nuevo que el que tiene actualmente. Los enrutadores también envían estos mensajes cuando una línea sube o baja o su costo cambia.

Los mensajes DATABASE DESCRIPTION dan los números de secuencia de todas las entradas de estado del enlace poseídas por el emisor actualmente. Comparando sus propios valores con los del emisor, el receptor puede determinar quién tiene los valores más recientes. Estos mensajes se usan cuando se activa una línea.

Cualquier socio puede pedir información del estado del enlace al otro usando los mensajes LINK STATE REQUEST. El resultado de este algoritmo es que cada par de enrutadores adyacentes hace una verificación para ver quién tiene los datos más recientes, y de esta manera se difunde la nueva información a lo largo del área. Todos estos mensajes se envían como paquetes IP. En la figura 5-66 se resumen los cinco tipos de mensajes.

Tipo de mensaje	Descripción
Hello	Descubre quiénes son los vecinos
Link state update	Proporciona los costos del emisor a sus vecinos
Link state ack	Confirma la recepción de la actualización del estado del enlace
Database description	Anuncia qué actualizaciones tiene el emisor
Link state request	Solicita información del socio

Figura 5-66. Los cinco tipos de mensajes de OSPF.

Finalmente podemos reunir todas las piezas. Utilizando la inundación de mensajes, cada enrutador informa a todos los demás enrutadores en su área sobre sus vecinos y costos. Esta información permite a cada enrutador construir el grafo para su(s) área(s) y calcular la ruta más corta. El área de la red dorsal también hace esto. Además, los enrutadores de la red dorsal aceptan la información de los enrutadores del límite de área para calcular la mejor ruta de cada enrutador de la red dorsal a cada enrutador restante. Esta información se difunde a los enrutadores de límite de área que la anuncian dentro de sus áreas. Usando esta información, un enrutador que está a punto de enviar un paquete dentro del área puede seleccionar el enrutador de mejor salida a la red dorsal.

5.6.5 BGP—Protocolo de Puerta de Enlace de Frontera

Dentro de un solo sistema autónomo, el protocolo de enrutamiento recomendado es OSPF (aunque no es ciertamente el único en uso). Entre los sistemas autónomos se utiliza un protocolo diferente, el **Protocolo de Puerta de Enlace de Frontera** (**BGP**)**.** Se necesita un protocolo diferente entre sistemas autónomos porque los objetivos de un protocolo de puerta de enlace interior y un protocolo de puerta de enlace exterior no son los mismos. Todo lo que tiene que hacer un protocolo de puerta de enlace interior es mover lo más eficazmente posible los paquetes del origen al destino. No tiene que preocuparse por las políticas.

Los enrutadores del protocolo de puerta de enlace exterior tienen que preocuparse en gran manera por la política (Metz, 2001). Por ejemplo, un sistema autónomo corporativo podría desear la habilidad para enviar paquetes a cualquier sitio de Internet y recibir los paquetes de cualquier sitio de Internet. Sin embargo, podría no estar dispuesto a llevar paquetes de tránsito que se originan en un sistema autónomo foráneo con destino a un sistema autónomo foráneo diferente, aun cuando su propio sistema autónomo estaba en la ruta más corta entre los dos sistemas autónomos foráneos (“Ése es su problema, no el nuestro”). Por otro lado, podría estar dispuesto a llevar el tráfico del tránsito para sus vecinos o incluso para otros sistemas autónomos específicos que pagaron por este servicio. Por ejemplo, las compañías de teléfonos podrían estar contentas de actuar como empresas portadoras para sus clientes, pero no para otros. En general, los protocolos de puerta de enlace exterior, y BGP en particular, se han diseñado para permitir que se implementen muchos tipos de políticas de enrutamiento en el tráfico entre sistemas autónomos.

Las políticas típicas implican consideraciones políticas, de seguridad, o económicas. Algunos ejemplos de limitaciones de enrutamiento son:

1. Ningún tránsito a través de ciertos sistemas autónomos.
2. Nunca ponga Irak en una ruta que inicie en el Pentágono.
3. No pasar por Estados Unidos para llegar de la Columbia Británica a Ontario.
4. Transite por Albania sólo si no hay otra alternativa al destino.
5. El tráfico que empieza o termina en IBM no debe transitar por Microsoft.

Las políticas en cada enrutador de BGP se configuran manualmente (o usando algún tipo de escritura). No son parte del protocolo.

Desde el punto de vista de un enrutador de BGP, el mundo consiste en sistemas autónomos y las líneas que los conectan. Dos sistemas autónomos se consideran conectados si hay una línea entre un enrutador fronterizo en cada uno. Dado el especial interés de BGP en el transporte de tráfico, las redes se agrupan en una de tres categorías. La primera son las **redes *stub,*** que tienen sólo una conexión con el grafo de BGP. Éstas no se pueden usar para transportar tráfico porque no hay nadie en el otro lado. Luego vienen las **redes multiconectadas.** Éstas podrían usarse para el transporte de tráfico excepto que lo rechacen. Finalmente, están las **redes de tránsito,** como redes dorsales, que están dispuestas a ocuparse de paquetes de terceros, posiblemente con algunas restricciones, y normalmente por pago.

Los pares de enrutadores de BGP se comunican entre sí estableciendo conexiones TCP. Operando de esta manera proporcionan comunicación confiable y ocultan todo detalle de red que pase a través de ellos.

Básicamente, BGP es muy parecido a un protocolo de vector de distancia, pero muy diferente de la mayoría de otros como RIP. En lugar de mantener el costo para cada destino, cada enrutador de BGP guarda el registro de la ruta utilizada, por lo que se conoce como un protocolo de vector de ruta. Del mismo modo, en lugar de darle a cada vecino el costo de cada posible destino estimado periódicamente, cada enrutador de BGP les dice el camino exacto que está usando.

Por ejemplo, considere los enrutadores de BGP mostrados en figura 5-67(a). En particular, considere la tabla de enrutamiento de *F*. Suponga que utiliza la ruta *FGCD* para llegar a *D*. Cuando los vecinos le dan información de la ruta, le proporcionan sus rutas completas, como se muestra en la figura 5-67(b) (para simplificar, sólo se muestra aquí el destino *D*).

Luego que han llegado todas las rutas de los vecinos, *F* las examina para ver cuál es la mejor. Desecha pronto las de *I* y *E*, porque pasan a través de la propia *F*. La opción está entonces entre usar *B* y *G*. Cada enrutador de BGP contiene un módulo que examina las rutas a un destino dado y las califica, devolviendo un número para la "distancia" a ese destino por cada ruta. Cualquier ruta que viole una restricción de la política recibe automáticamente una calificación al infinito. Entonces el enrutador adopta la ruta de la distancia más corta. La función de calificar no es parte del protocolo de BGP y puede ser cualquier función que el gerente de sistemas desee.

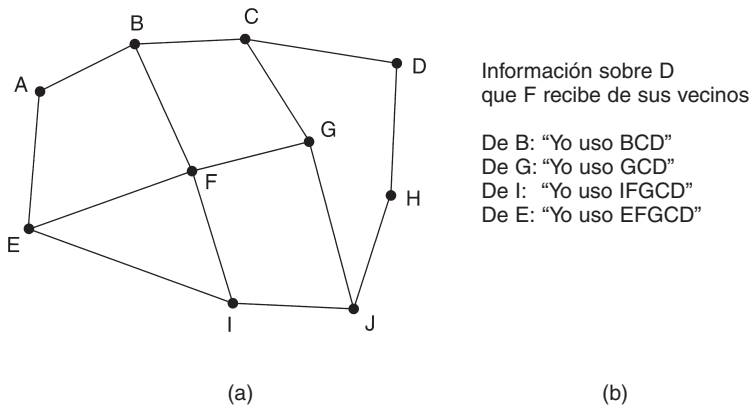


Figura 5-67. (a) Conjunto de enrutadores de BGP. (b) Información enviada a F.

BGP resuelve fácilmente el problema de la cuenta hasta el infinito que plaga otros algoritmos de vector de distancia. Por ejemplo, suponga que *G* se congela o que la línea *FG* se cae. Entonces *F* recibe las rutas de sus tres vecinos restantes. Estas rutas son *BCD, IFGCD* y *EFGCD*. Puede ver inmediatamente que las dos últimas rutas son vanas, ya que atraviesan *F*, por lo que escoge *FBCD* como su nueva ruta. A menudo otros algoritmos de vector de distancia hacen una mala elección porque no saben quiénes de sus vecinos tienen rutas independientes al destino y quiénes no. La definición de BGP está en los RFCs 1771 a 1774.

5.6.6 Multidifusión de Internet

La comunicación normal de IP está entre un emisor y un receptor. Sin embargo, para algunas aplicaciones es útil que un proceso pueda enviar simultáneamente a una gran cantidad de receptores, por ejemplo, actualización duplicada, bases de datos distribuidas, transmisión de cotizaciones de acciones a corredores múltiples, y manejo de conferencia digital en llamadas telefónicas (es decir, entre muchas partes).

IP apoya la multidifusión, usando direcciones clase D. Cada dirección clase D identifica un grupo de *hosts*. Hay 28 bits disponibles para identificar los grupos, de modo que pueden existir al mismo tiempo más de 250 millones de grupos. Cuando un proceso envía un paquete a una dirección clase D, se hace el mejor esfuerzo para entregarlo a todos los miembros del grupo direccionado, pero no se da garantía alguna. Quizá algunos miembros no reciban el paquete.

Se soportan dos tipos de direcciones de grupo: las permanentes y las temporales. Un grupo permanente siempre está allí y no se tiene que preparar. Cada grupo permanente tiene una dirección de grupo permanente. Algunos ejemplos de direcciones de grupo permanentes son:

224.0.0.1 Todos los sistemas en una LAN
224.0.0.2 Todos los enrutadores en una LAN
224.0.0.5 Todos los enrutadores de OSPF en una LAN
224.0.0.6 Todos los enrutadores designados de OSPF en una LAN

Los grupos temporales se deben crear antes de que se puedan usar. Un proceso puede pedir a su *host* que se una a un grupo específico. También puede pedirle que deje el grupo. Cuando el último proceso en un *host* deja un grupo, ese grupo ya no está presente en el *host*. Cada *host* conserva el registro de qué grupos pertenecen actualmente a sus procesos.

La multidifusión se implementa mediante enrutadores de multidifusión especiales que pueden o no colocarse con los enrutadores normales. Alrededor de una vez por minuto, cada uno de estos enrutadores envía una multidifusión de hardware (es decir, una multidifusión de la capa de enlace de datos) a los *hosts* en su LAN (dirección 224.0.0.1) pidiéndoles que devuelvan información de los grupos a que pertenecen actualmente sus procesos. Cada *host* devuelve las respuestas a todas las direcciones clase D interesadas.

Estos paquetes de preguntas y respuestas utilizan un protocolo llamado **IGMP** (**Protocolo de Administración de Grupo de Internet**) que es vagamente análogo al ICMP. Tiene sólo dos tipos de paquetes: pregunta y respuesta, cada uno con un formato simple, fijo, que contiene alguna información de control en la primera palabra del campo de carga útil y una dirección clase D en la segunda palabra. Se describe en el RFC 1112.

El enrutamiento de multidifusión se crea utilizando árboles de difusión. Cada enrutador de multidifusión intercambia información con sus vecinos, usando un protocolo de vector de distancia modificado para que cada uno construya un árbol de expansión por grupo que cubra a todos los miembros del grupo. Se usan varias optimizaciones para recortar el árbol y eliminar los enrutadores y redes que no se interesan en grupos particulares. El protocolo hace un gran uso de túneles para no molestar a los nodos que no pertenecen al árbol de expansión.

5.6.7 IP móvil

Muchos usuarios de Internet tienen computadoras portátiles y desean permanecer conectados a Internet cuando visitan un sitio de Internet distante e incluso durante el camino. Desgraciadamente, el sistema de dirección IP facilita el trabajo lejos de casa más de dicho que de hecho. En esta sección examinaremos el problema y la solución. Una descripción más detallada se da en (Perkins, 1998a).

El villano real es el propio esquema de direccionamiento. Cada dirección IP contiene un número de red y un número de *host*. Por ejemplo, considere la máquina con dirección IP 160.80.40.20/16. El número de red es 160.80 (8272 en sistema decimal); 40.20 es el número de *host* (10260 en el decimal). Los enrutadores en todo el mundo tienen tablas de enrutamiento que indican qué línea usar para conseguir conectarse a la red 160.80. Siempre que un paquete llegue con un destino de dirección IP de la forma 160.80.xxx.yyy, saldrá de esa línea.

Si de repente la máquina con esa dirección se lleva fuera a algún sitio distante, los paquetes se le seguirán enrutando a su LAN principal (o enrutador). El dueño ya no podrá recibir más correo

electrónico, etcétera. Dar a la máquina una nueva dirección IP que corresponda a su nueva situación es poco atractivo porque se tendría que informar del cambio a una gran cantidad de personas, programas y bases de datos.

Otro método es que los enrutadores tengan que usar direcciones IP completas para enrutamiento, en lugar de sólo la red. Sin embargo, esta estrategia requeriría que cada enrutador tuviera millones de entradas de tablas, a un costo astronómico para Internet.

Cuando las personas empezaron a exigir la capacidad de conectar sus PCs portátiles a Internet dondequiera que estuvieran, la IETF preparó un grupo de trabajo para encontrar una solución. El grupo de trabajo formuló rápidamente varios objetivos considerados deseables para cualquier solución. Los principales fueron:

1. Cada *host* móvil debe poder usar su dirección IP principal en cualquier parte.
2. No se permiten cambios de software a los *hosts* fijos.
3. No se permiten cambios al software ni a las tablas del enrutador.
4. La mayoría de paquetes para *host* móviles no debe hacer desvíos en la ruta.
5. No se debe incurrir en sobrecarga cuando un *host* móvil está en casa.

La solución escogida fue la que se describe en la sección 5.2.8. Como un repaso breve, cada sitio que permita vagar a sus usuarios tiene que crear un agente de base. Cada sitio que permita visitantes tiene que crear un agente foráneo. Cuando un *host* móvil se presenta a un sitio foráneo, contacta al *host* foráneo y se registra. El *host* foráneo entonces contacta al agente de base del usuario y le da una **dirección temporal,** (*care-* of *address*) normalmente la propia dirección IP del agente foráneo.

Cuando un paquete llega a la LAN principal del usuario, entra a algún enrutador adjunto a la LAN. Por lo general, el enrutador trata de localizar al *host*, difundiendo un paquete ARP preguntando, por ejemplo: ¿cuál es la dirección Ethernet de 160.80.40.20? El agente de base responde a esta pregunta dando su propia dirección de Ethernet. El enrutador envía entonces los paquetes para 160.80.40.20 al agente principal. A su vez, este último los canaliza a la dirección temporal encapsulándolos en el campo de carga útil de un paquete IP dirigido al agente foráneo. El agente foráneo entonces lo desencapsula y lo entrega a la dirección del enlace de datos del *host* móvil. Además, el agente de base da la dirección temporal al emisor, para que los paquetes futuros se puedan canalizar directamente al agente foráneo. Esta solución reúne todos los requisitos declarados anteriormente.

Tal vez valga la pena mencionar un pequeño detalle. Cuando el *host* se mueve, el enrutador probablemente tenga su dirección Ethernet en el caché (próxima a quedar invalidada). Reemplazar esa dirección Ethernet con la del agente de base se hace por un truco llamado **ARP gratuito.** Éste es un mensaje especial, no solicitado al enrutador que hace reemplazar una entrada específica del caché, en este caso la del *host* móvil que está a punto de salir. Cuando el *host* móvil vuelve después, se usa el mismo truco para actualizar de nuevo el caché del enrutador.

Nada en el diseño le impide a un *host* móvil ser su propio agente foráneo, pero ese método sólo funciona si el *host* móvil (en su capacidad de agente foráneo) está conectado lógicamente a Internet

en su sitio actual. Incluso, el *host* móvil debe tener la capacidad de adquirir una dirección IP (temporal). Esa dirección IP debe pertenecer a la LAN a que está adjunto actualmente.

La solución de la IETF para los *hosts* móviles resuelve varios otros problemas no mencionados hasta ahora. Por ejemplo, ¿cómo se localizan agentes? La solución es que cada agente transmita periódicamente su dirección y el tipo de servicios que está dispuesto a proporcionar (por ejemplo, de base, foráneo, o ambos). Cuando un *host* móvil llega a alguna parte, simplemente puede escuchar estas difusiones, llamadas **anuncios.** Como alternativa, puede difundir un paquete que anuncie su llegada y esperar que el agente foráneo local responda a él.

Otro problema que se tuvo que resolver es qué hacer con los *host* móviles mal educados que se van sin decir adiós. La solución es hacer válido el registro sólo para un intervalo de tiempo fijo. Si no se actualiza periódicamente, queda fuera para que el *host* foráneo pueda limpiar sus tablas.

Otro problema más es la seguridad. Cuando un agente de base recibe un mensaje que le pide que por favor envíe todos los paquetes de Roberta a alguna dirección IP, lo mejor es no hacerlo a menos que se convenza que Roberta es el origen de esta solicitud, y no alguien que intenta personificarla. Para este propósito se usan los protocolos de autenticación criptográficos. En el capítulo 8 estudiaremos esos protocolos.

Un punto final abordado por el grupo de trabajo se relaciona con los niveles de movilidad. Imagine un avión con una Ethernet a bordo utilizada por la navegación y computadoras de la aviación. En esta Ethernet hay un enrutador normal que se comunica con la Internet conectada en tierra a través de un enlace de radio. Un buen día, algún hábil ejecutivo de marketing tiene la idea de instalar conectores de Ethernet en todos los descansabrazos para que los pasajeros con computadoras móviles también se puedan conectar.

Ahora tenemos dos niveles de movilidad: las propias computadoras del avión que son estacionarias con respecto a Ethernet y las de los pasajeros, que son móviles con respecto al avión. Además, el enrutador a bordo es móvil con respecto a los enrutadores en tierra. Al ser móvil con respecto a un sistema que de suyo es móvil, se puede manejar utilizando el entunelamiento recursivo.

5.6.8 IPv6

Si bien el CIDR y el NAT pueden durar unos pocos años más, todo mundo se da cuenta que los días del IP en su forma actual (Ipv4) están contados. Además de estos problemas técnicos, hay otro asunto acechando. Hasta hace poco, la Internet ha sido utilizada en gran medida por universidades, industrias de alta tecnología y el gobierno (especialmente el Departamento de Defensa de Estados Unidos). Con la explosión del interés por la Internet que comenzó a mediados de la década de 1990, es muy posible que en el próximo milenio será usada por un grupo mucho más grande de gente, especialmente gente con necesidades diferentes. Por una parte, millones de personas con computadoras portátiles inalámbricas podrían usarla para mantenerse en contacto con sus bases. Por otra, con la inminente convergencia de las industrias de la computación, las comunicaciones y el entretenimiento, tal vez no pase mucho tiempo ante de que todos los teléfonos y los televisores del

mundo estén en un nodo Internet, produciendo mil millones de máquinas utilizadas para recibir audio y vídeo bajo demanda. En estas circunstancias, se hizo evidente que el IP tenía que evolucionar y volverse más flexible.

Al ver en el horizonte estos problemas, la IETF comenzó a trabajar en 1990 en una versión nueva del IP, una que nunca se quedaría sin direcciones, resolvería varios otros problemas y sería más flexible y eficiente también. Sus metas principales eran:

1. Manejar miles de millones de *hosts*, aún con asignación de espacio de direcciones ineficiente.
2. Reducir el tamaño de las tablas de enrutamiento.
3. Simplificar el protocolo, para permitir a los enrutadores el procesamiento más rápido de los paquetes.
4. Proporcionar mayor seguridad (verificación de autenticidad y confidencialidad) que el IP actual.
5. Prestar mayor atención al tipo de servicio, especialmente con datos en tiempo real.
6. Ayudar a la multidifusión permitiendo la especificación de alcances.
7. Posibilitar que un *host* sea móvil sin cambiar su dirección.
8. Permitir que el protocolo evolucione.
9. Permitir que el protocolo viejo y el nuevo coexistan por años.

Para encontrar un protocolo que cumpliera con todos estos requisitos, la IETF hizo una convocatoria solicitando propuestas y estudios en el RFC 1550. Se recibieron 21 respuestas, no todas propuestas completas. En diciembre de 1992 había siete propuestas serias en la mesa; iban desde hacer cambios menores al IP hasta desecharlo y reemplazarlo por un protocolo completamente diferente.

Una propuesta fue ejecutar TCP sobre CLNP, que, con sus direcciones de 160 bits habría proporcionado suficiente espacio de direcciones para siempre y habría unificado dos protocolos de capa de red principales. Sin embargo, muchos pensaron que esto habría sido una aceptación de que algunas cosas en el mundo OSI en realidad estaban bien hechas, algo considerado políticamente incorrecto en los círculos de Internet. El CLNP se creó tomando como modelo al IP, por lo que los dos no son realmente tan diferentes. De hecho, el protocolo escogido es más diferente del IP que CLNP. Otra cosa en contra de CLNP fue su pobre manejo de tipos de servicio, algo requerido para difundir multimedia eficientemente.

Tres de las mejores propuestas se publicaron en *IEEE Network* (Deering, 1993; Francis, 1993, y Katz y Ford, 1993). Tras muchos análisis, revisiones e intrigas, se seleccionó una versión modificada de la combinación de las propuestas de Deering y Francis, llamada ahora **SIPP** (**Protocolo Simple de Internet Mejorado**)**,** y se le dio la designación **Ipv6.**

El IPv6 cumple los objetivos bastante bien: mantiene las buenas características del IP, descarta y reduce las malas, y agrega nuevas donde se necesitan. En general, IPv6 no es compatible con

IPv4, pero es compatible con todos los demás protocolos Internet, incluidos TCP, UDP, ICMP, IGMP, OSPF, BGP y DNS, a veces con algunas pequeñas modificaciones (principalmente para manejar direcciones más grandes). Las características principales del IPv6 se analizan a continuación. Puede encontrarse mayor información en los RFCs 2460 a 2466.

Por principio, y lo más importante, el IPv6 tiene direcciones más grandes que el IPv4; son de 16 bytes de longitud, lo que resuelve el problema que se buscaba resolver: proporcionar una cantidad prácticamente ilimitada de direcciones Internet. Hablaremos más sobre las direcciones un poco más adelante.

La segunda mejora principal del IPv6 es la simplificación del encabezado, que contiene sólo 7 campos (contra 13 en el IPv4). Este cambio permite a los enrutadores procesar con mayor rapidez los paquetes y mejorar, por tanto, la velocidad real de transporte. También estudiaremos pronto el encabezado.

La tercera mejora importante fue el mejor apoyo de las opciones. Este cambio fue esencial con el nuevo encabezado, pues campos que antes eran obligatorios ahora son opcionales. Además, es diferente la manera de representar las opciones, haciendo más sencillo que los enrutadores hagan caso omiso de opciones no dirigidas a ellos. Esta característica mejora el tiempo de procesamiento de los paquetes.

Una cuarta área en la que el IPv6 representa un avance importante es la seguridad. La IETF tenía infinidad de historias sobre preadolescentes precoces que usaban sus computadoras personales para meterse en bancos e instalaciones militares por todas partes de Internet. Se tenía la fuerte sensación de que había que hacerse algo para mejorar la seguridad. La autenticación y la privacidad son características clave del IP nuevo. Estas características fueron incluídas posteriormente en el IPv4, así que las diferencias no son tan marcadas en el área de la seguridad.

Por último, se ha puesto mayor atención en la calidad del servicio. En el pasado se realizaron varios esfuerzos débiles, pero con el crecimiento actual de la multimedia en Internet, se requiere un mayor esfuerzo.

El encabezado principal del IPv6

El encabezado del IPv6 se muestra en la figura 5-68. El campo de *Versión* siempre es 6 para el IPv6 (y de 4 para el IPv4). Durante el periodo de transición del IPv4 al IPv6, que probablemente se llevará una década, los enrutadores podrán examinar este campo para saber el tipo de paquete que tienen. Como nota al margen, esta prueba ocupa algunas instrucciones en la ruta crítica, por lo que muchas implementaciones probablemente la evitarán usando algún campo del encabezado de enlace de datos para distinguir los paquetes IPv4 de los IPv6. De esta manera, los paquetes pueden pasarse directamente al manejador correcto de la capa de red. Sin embargo, hacer que la capa de enlace de datos esté consciente de los tipos de los paquetes de red viola por completo el principio de diseño de que ninguna capa debe estar enterada del significado de los bits entregados por la capa superior a ella. Los debates entre los bandos de “hacerlo bien” y “hacerlo rápido” sin duda serán largos y acalorados.

El campo *Clase de tráfico* se usa para distinguir entre los paquetes con requisitos diferentes de entrega en tiempo real. Un campo diseñado para este propósito ha estado en el IP desde el principio,

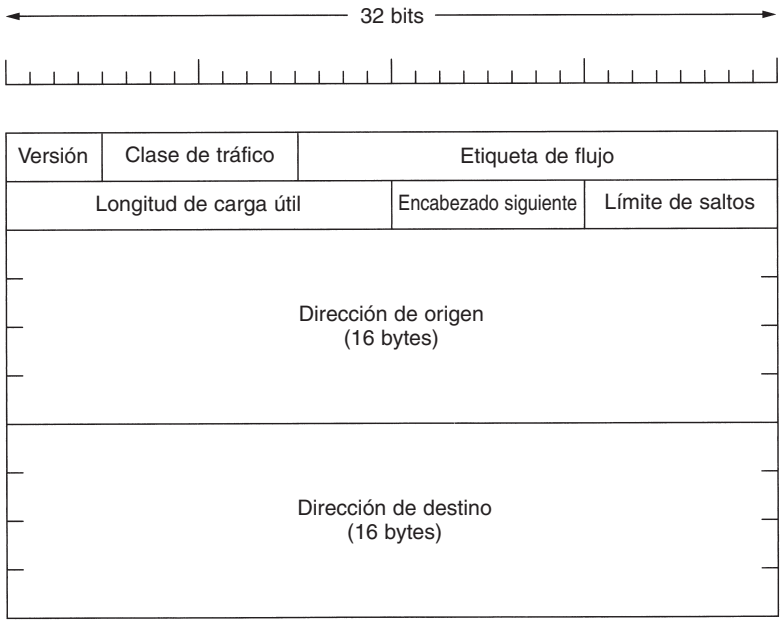


Figura 5-68. Encabezado fijo del IPv6 (obligatorio).

pero los enrutadores lo han implementado sólo esporádicamente. Ahora están en camino los experimentos para determinar qué tan bueno puede ser para usarse en la entrega de multimedia.

El campo de *Etiqueta de flujo* aún es experimental, pero se usará para permitir a un origen y a un destino establecer una pseudoconexión con propiedades y requisitos particulares. Por ejemplo, una cadena de paquetes de un proceso en cierto *host* de origen dirigido a cierto proceso en cierto *host* de destino puede tener requisitos de retardo muy estrictos y, por tanto, necesitar un ancho de banda reservado. El flujo puede establecerse por adelantado, dándole un identificador. Cuando aparece un paquete con una *Etiqueta de flujo* diferente de cero, todos los enrutadores pueden buscarla en sus tablas internas para ver el tipo de tratamiento especial que requiere. En efecto, los flujos son un intento de tener lo mejor de ambos mundos: la flexibilidad de una subred de datagramas y las garantías de una subred de circuitos virtuales.

Cada flujo está designado por la dirección de origen, la dirección de destino y el número de flujo, por lo que pueden estar activos muchos flujos al mismo tiempo entre un par dado de direcciones IP. También, de esta manera, aun si dos flujos provenientes de *hosts* diferentes pero con el mismo número de flujo pasan por el mismo enrutador, el enrutador será capaz de distinguirlos usando las direcciones de origen y destino. Se espera que se escojan los números de flujo al azar, en lugar de asignarlos secuencialmente comenzando por el 1, para simplificar el proceso de dispersión en los enrutadores.

El campo de *Longitud de carga útil* indica cuántos bytes siguen al encabezado de 40 bytes de la figura 5-68. El nombre de campo se cambió de *Longitud total* en el IPv4 porque el significado

cambió ligeramente: los 40 bytes del encabezado ya no se cuentan como parte de la longitud, como antes.

El campo *Encabezado siguiente* revela el secreto. La razón por la que pudo simplificarse el encabezado es que puede haber encabezados adicionales (opcionales) de extensión. Este campo indica cuál de los seis encabezados de extensión (actualmente), de haberlos, sigue a éste. Si este encabezado es el último encabezado de IP, el campo de *Encabezado siguiente* indica el manejador de protocolo de transporte (por ejemplo, TCP, UDP) al que se entregará el paquete.

El campo de *Límite de saltos* se usa para evitar que los paquetes vivan eternamente. En la práctica es igual al campo de *Tiempo de vida* del IPv4, es decir, un campo que se disminuye en cada salto. En teoría, en el IPv4 era un tiempo en segundos, pero ningún enrutador lo usaba de esa manera, por lo que se cambió el nombre para reflejar la manera en que se usa realmente.

Luego vienen los campos de *Dirección de origen* y *Dirección de destino.* La propuesta original de Deering, el SIP, usaba direcciones de 8, pero durante el periodo de revisión muchas personas sintieron que, con direcciones de 8 bytes, en algunas décadas el IPv6 se quedaría sin direcciones, y que con direcciones de 16 bytes nunca se acabarían. Otros argumentaban que 16 bytes era demasiado, mientras que unos más estaban a favor de usar direcciones de 20 bytes para hacerlas compatibles con el protocolo de datagramas de OSI. Otro grupo quería direcciones de tamaño variable. Tras mucho debate, se decidió que la mejor media sería una dirección de 16 bytes de longitud fija.

Se ha desarrollado una nueva notación para escribir direcciones de 16 bytes: se escriben como ocho grupos de cuatro dígitos hexadecimales, separados los grupos por dos puntos, como sigue:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Ya que muchas direcciones tendrán muchos ceros en ellas, se han autorizado tres optimizaciones. Primero, los ceros a la izquierda de un grupo pueden omitirse, por lo que 0123 puede escribirse como 123. Segundo, pueden reemplazarse uno o más grupos de 16 ceros por un par de signos de dos puntos. Por tanto, la dirección anterior se vuelve ahora

8000::0123:4567:89AB:CDEF

Por último, las direcciones IPv4 pueden escribirse como un par de signos de dos puntos y un número decimal anterior separado por puntos, como por ejemplo

::192.31.20.46

Tal vez no sea necesario ser tan explícitos al respecto, pero hay muchas direcciones de 16 bytes. Específicamente, hay 2^{128} de ellas, lo que aproximadamente es 3×10^{38}. Si la Tierra completa, incluidos los océanos, estuviera cubierta de computadoras, el IPv6 permitiría 7×10^{23} direcciones IP por metro cuadrado. Los estudiantes de química notarán que este número es mayor que el número de Avogadro. Aunque no fue la intención darle a cada molécula de la superficie terrestre su propia dirección IP, no estamos lejos de ello.

En la práctica, el espacio de direcciones no se usará eficientemente, igual que el espacio de números telefónicos (el código de área de Manhattan, 212, está prácticamente lleno, pero el

de Wyoming, 307, está casi vacío). En el RFC 3194, Durand y Huitema calcularon que, usando la asignación de números telefónicos como guía, hasta en la situación más pesimista habrá más de 1000 direcciones IP por metro cuadrado de la superficie terrestre (tierra y agua). En cualquier situación probable, habrá billones de ellas por metro cuadrado. En pocas palabras, parece poco probable que se acabarán en el futuro previsible.

Es instructivo comparar el encabezado de IPv4 (figura 5-53) con el de IPv6 (figura 5-68) para ver lo que se ha dejado fuera del IPv6. El campo *IHL* se fue porque el encabezado de IPv6 tiene una longitud fija. El campo de *Protocolo* se retiró porque el campo *Encabezado siguiente* indica lo que sigue al último encabezado de IP (por ejemplo, un segmento UDP o TCP).

Se retiraron todos los campos relacionados con la fragmentación, puesto que el IPv6 tiene un enfoque distinto hacia la fragmentación. Para empezar, todos los *hosts* que se ajustan al IPv6 deben determinar dinámicamente el tamaño de datagrama. Asimismo, el mínimo se incrementó de 576 a 1280 para permitir 1024 bytes de datos y varios encabezados. Esta regla hace que sea menos posible que ocurra la fragmentación. Además, cuando un *host* envía un paquete de IPv6 demasiado grande, en lugar de fragmentarlo, el enrutador que es incapaz de reenviarlo devuelve un mensaje de error. Este mensaje indica al *host* que divida todos los paquetes futuros a ese destino. Es mucho más eficiente hacer que el *host* envíe paquetes del tamaño correcto desde el principio que hacer que los enrutadores los fragmenten sobre la marcha.

Por último, el campo de *Suma de verificación* desaparece, porque su cálculo reduce en gran medida el desempeño. Con las redes confiables de hoy, además del hecho de que la capa de enlace de datos y las capas de transporte normalmente tienen sus propias sumas de verificación, el provecho de otra suma de verificación no valía el costo de desempeño que generaba. Al removerse estas características ha quedado un protocolo de capa de red compacto y sólido. Por tanto, la meta del IPv6 (un protocolo rápido y flexible con bastante espacio de direcciones) se ha cumplido con este diseño.

Encabezados de extensión

Con todo, algunos de los campos faltantes del IPv4 en ocasiones son necesarios, por lo que el IPv6 introdujo el concepto de **encabezado de extensión** (opcional). Estos encabezados pueden usarse para proporcionar información extra, pero codificada de una manera eficiente. Hay seis tipos de encabezados de extensión definidos actualmente, que se listan en la figura 5-69. Todos son opcionales, pero si hay más de uno, deben aparecer justo después del encabezado fijo, y de preferencia en el orden listado.

Algunos de los encabezados tienen un formato fijo; otros contienen un número variable de campos de longitud variable. En éstos, cada elemento está codificado como una tupla (tipo, longitud, valor). El *Tipo* es un campo de 1 byte que indica la opción de la que se trata. Los valores de *Tipo* se han escogido de modo que los dos primeros bits indican a los enrutadores que no saben cómo procesar la opción lo que tienen que hacer. Las posibilidades son: saltar la opción, descartar el paquete, descartar el paquete y enviar de regreso un paquete ICMP, y lo mismo que lo anterior, pero no enviar paquetes ICMP a direcciones de multidifusión (para evitar que un paquete de multidifusión malo genere millones de informes ICMP).

Encabezado de extensión	Descripción
Opciones salto por salto	Información diversa para los enrutadores
Opciones de destino	Información adicional para el destino
Enrutamiento	Ruta total o parcial a seguir
Fragmentación	Manejo de fragmentos de datagramas
Autenticación	Verificación de la identidad del emisor
Carga útil de seguridad encriptada	Información sobre el contenido encriptado

Figura 5-69. Encabezados de extensión del IPv6.

La *Longitud* también es un campo de 1 byte, e indica la longitud del valor (0 a 255 bytes). El *Valor* es cualquier información requerida, de hasta 255 bytes.

El encabezado de salto por salto se usa para información que deben examinar todos los enrutadores a lo largo de la ruta. Hasta ahora, se ha definido una opción: manejo de datagramas de más de 64K. El formato de este encabezado se muestra en la figura 5-70. Cuando se utiliza, el campo de *Longitud* de carga útil del siguiente encabezado se establece a cero.

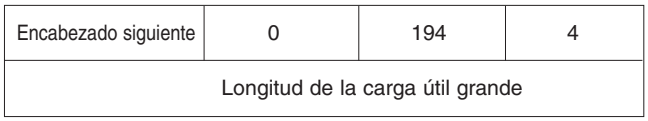


Figura 5-70. Encabezado de extensión salto por salto para datagramas grandes (jumbogramas).

Como todos los encabezados de extensión, éste comienza con 1 byte que indica el tipo de encabezado que sigue. A este byte le sigue uno que indica la longitud del encabezado salto por salto en bytes, excluyendo los primeros 8 bytes, que son obligatorios. Todas las extensiones empiezan de esta manera.

Los 2 bytes siguientes indican que esta opción define el tamaño del datagrama (código 194) como número de 4 bytes. Los últimos 4 bytes indican el tamaño del datagrama. No se permiten los tamaños menores que 65,536, que darán como resultado que el primer enrutador descarte el paquete y devuelva un mensaje ICMP de error. Los datagramas que usan este encabezado de extensión se llaman **jumbogramas.** El uso de jumbogramas es importante para las aplicaciones de supercomputadoras que deben transferir con eficiencia gigabytes de datos a través de Internet.

El encabezado de opciones de destino está proyectado para campos que sólo necesitan ser interpretados en el *host* de destino. En la versión inicial de IPv6, las únicas opciones definidas son las opciones nulas con respecto a inflar este encabezado a un múltiplo de 8 bytes, por lo que inicialmente no se usará. Se incluyó para asegurarse que ese nuevo enrutamiento y el software del *host* pudieran manejarlo, en caso de que algún día alguien pensara en una opción de destino.

El encabezado de enrutamiento lista uno o más enrutadores que deben visitarse en el camino al destino. Es muy similar al enrutamiento libre del IPv4 en el sentido de que todas las direcciones listadas se deben visitar en orden, pero también se podrían visitar otros enrutadores no listados que se encuentren en medio de la ruta. El formato del encabezado de enrutamiento se muestra en la figura 5-71.

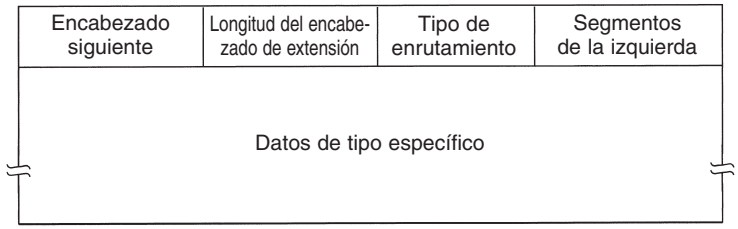


Figura 5-71. Encabezado de extensión para enrutamiento.

Los primeros 4 bytes del encabezado de extensión de enrutamiento contienen cuatro enteros de 1 byte. Anteriormente describimos los campos *Encabezado siguiente* y *Longitud del encabezado de extensión*. El campo *Tipo de enrutamiento* da el formato del resto del encabezado. Teclear 0 significa que una palabra reservada de 32 bits sigue a la primera palabra, seguida por algún número de direcciones de IPv6. Pueden inventarse otros tipos en el futuro según se necesite. Finalmente, el campo *Segmentos restantes* registra cuántas direcciones de la lista no se han visitado todavía. Se reduce cada vez que se visita una. Cuando llega a 0, el paquete está solo sin más guía sobre qué ruta seguir. En general, a estas alturas está tan cerca del destino que la mejor ruta es obvia.

El encabezado de fragmento maneja la fragmentación de una manera parecida a la del IPv4. El encabezado contiene el identificador del datagrama, el número de fragmento y un bit que indica si seguirán más fragmentos. En el IPv6, a diferencia del IPv4, sólo el *host* de origen puede fragmentar un paquete. Los enrutadores a lo largo del camino no pueden hacerlo. Aunque este cambio es un rompimiento filosófico con el pasado, simplifica el trabajo del enrutador y acelera el enrutamiento. Como se mencionó antes, si un enrutador confronta un paquete demasiado grande, lo descarta y devuelve un paquete ICMP al origen. Esta información permite que el *host* de origen fragmente el paquete en pedazos más pequeños usando este encabezado y lo intente de nuevo.

El encabezado de autenticación proporciona un mecanismo mediante el cual el receptor de un paquete puede estar seguro de quién lo envió. La carga útil de seguridad encriptada posibilita encriptar el contenido de un paquete de modo que sólo el receptor pretendido pueda leerlo. Estos encabezados usan técnicas criptográficas para lograr su cometido.

Controversias

Dados el proceso de diseño abierto y las fuertes opiniones de muchas de las personas participantes, no debería sorprender que muchas decisiones tomadas para el IPv6 fueran tema de fuertes controversias. Resumiremos a continuación algunas de ellas. Para los detalles, vea los RFCs.

Ya hemos mencionado el argumento sobre la longitud de las direcciones. El resultado fue una solución intermedia: las direcciones de 16 bytes de longitud fija.

Surgió otra pelea sobre la longitud del campo de *Límite de saltos.* Una parte sentía que la limitación de la cantidad máxima de saltos a 255 (implícita al usar un campo de 8 bits) fue un error grave. A fin de cuentas, hoy son comunes las rutas de 32 saltos, y en 10 años podrán ser comunes rutas mucho más grandes. Esta gente argumentaba que el uso de una dirección enorme era ir demasiado lejos, pero que el uso de una cuenta de saltos pequeña era tener una visión miope. Desde su punto de vista, el peor pecado que puede cometer un informático es proporcionar insuficientes bits en algún lugar.

La respuesta fue que se pueden hacer argumentos para aumentar todos los campos, lo que llevaría a un encabezado inflamado. También, la función del campo de *Límite de saltos* es evitar que los paquetes vaguen durante demasiado tiempo, y 65,535 saltos son demasiados. Por último, a medida que crezca Internet, se construirán más y más enlaces de larga distancia, posibilitando la ida de un país a otro en media docena de saltos, cuando mucho. Si se requieren más de 125 saltos para llegar del origen y el destino a sus puertas de enlace internacionales, algo está mal con las redes dorsales nacionales. Los de 8 bits ganaron esta partida.

Otra papa caliente fue el tamaño máximo de paquete. La comunidad de las supercomputadoras quería paquetes de más de 64 KB. Cuando una supercomputadora comienza a transferir, el asunto realmente va en serio, y no quiere que se le interrumpa cada 64 KB. El argumento en contra de los paquetes grandes es que, si un paquete de 1 MB llega a una línea T1 de 1.5 Mbps, el paquete bloqueará la línea durante más de 5 segundos, produciendo un retardo muy notorio para los usuarios interactivos que comparten la línea. Se llegó a un punto medio: los paquetes normales se limitan a 64 KB, pero puede usarse el encabezado de extensión de salto por salto para permitir los jumbogramas.

Un tercer tema candente fue la desaparición de la suma de verificación del IPv4. Para algunas personas esto representa algo parecido a quitarle los frenos a un automóvil. Hacerlo ciertamente aligera al automóvil y, por tanto, puede ir más rápido pero, de ocurrir un evento inesperado, tendremos problemas.

El argumento en contra de las sumas de verificación fue que cualquier aplicación a la que de verdad le importa la integridad de sus datos de todos modos tiene que tener una suma de verificación en la capa de transporte, por lo que tener otra en el IP (además de la suma de verificación de la capa de enlace de datos) es un exceso. Además, la experiencia mostraba que el cálculo de una suma de verificación en el IP era un gasto importante en el IPv4. El bando en contra de la suma de verificación ganó ésta, y el IPv6 no tiene una suma de verificación.

Los *hosts* móviles también fueron tema de contienda. Si una computadora portátil vuela al otro lado del mundo, ¿puede continuar operando en el destino con la misma dirección IPv6, o tiene que usar un esquema con agentes foráneos y agentes de base? Los *hosts* móviles también generan asimetrías en el sistema de enrutamiento. Es posible el caso en que una computadora móvil pequeña pueda escuchar fácilmente la señal emitida por un enrutador estacionario grande, pero que el enrutador estacionario no pueda escuchar la débil señal emitida por el *host* móvil. En consecuencia, algunas personas querían incluir soporte explícito de *hosts* móviles en el IPv6. Este esfuerzo falló cuando no se pudo generar consenso para ninguna propuesta específica.

Probablemente la batalla principal fue sobre la seguridad. Todos estaban de acuerdo en que se necesitaba. La guerra fue sobre dónde y cuándo. Primero dónde. El argumento a favor de ponerla en la capa de red es que entonces se vuelve un servicio estándar que todas las aplicaciones pueden usar sin ninguna planeación adelantada. El argumento en contra es que las aplicaciones realmente seguras por lo general no quieren nada menos que la encriptación de terminal a terminal, donde la aplicación de origen hace la encriptación y la aplicación de destino la deshace. Con cualquier otra cosa menos, el usuario está a merced de implementaciones de capa de red con fallas potenciales, sobre las que no tiene control. La respuesta a este argumento es que tales aplicaciones simplemente pueden abstenerse de usar las características de seguridad del IP y encargarse ellas mismas del asunto. La réplica a esto es que la gente que no confía en que la red lo haga bien no quiere pagar el precio de implementaciones de IP lentas y estorbosas que tengan esta capacidad, aun si está inhabilitada.

Otro aspecto sobre dónde poner la seguridad se relaciona con que muchos países (pero no todos) tienen leyes de exportación estrictas en lo referente a criptografía. Algunos, particularmente Francia e Irak, también restringen mucho su uso doméstico, de manera que la gente no pueda ocultar secretos de la policía. Como resultado, cualquier implementación de IP que use un sistema criptográfico lo bastante robusto como para tener algún valor no podría exportarse de los Estados Unidos (y de muchos otros países) a clientes mundiales. La mayoría de los proveedores de computadoras se oponen enérgicamente a tener que mantener dos juegos de *software*, uno para uso doméstico y otro para exportación.

Un punto donde no hubo controversia es que nadie espera que la Internet IPv4 se apague un domingo por la mañana y reinicie como Internet IPv6 la mañana del lunes. En cambio, se convertirían "islas" aisladas a IPv6, comunicándose inicialmente a través de túneles. A medida que crezcan las islas IPv6, se integrarán a islas más grandes. Tarde o temprano todas las islas se integrarán, y la Internet habrá sido convertida por completo. Dada la cuantiosa inversión en los enrutadores IPv4 actualmente instalados, el proceso de conversión probablemente tardará una década. Por esta razón, se ha puesto un enorme esfuerzo en asegurar que esta transición sea lo menos dolorosa posible. Para mayor información sobre IPv6, vea (Loshin, 1999).

5.7 RESUMEN

La capa de red proporciona servicios a la capa de transporte; puede basarse tanto en circuitos virtuales como en datagramas. En ambos casos, la tarea principal de esta capa es enrutar paquetes del origen al destino. En las subredes de circuitos virtuales se toma una decisión de enrutamiento al establecerse el circuito virtual; en las subredes de datagramas, se hace en cada paquete.

Se usan muchos algoritmos de enrutamiento en las redes de computadoras. Los algoritmos estáticos incluyen el enrutamiento por ruta más corta y la inundación. Los algoritmos dinámicos incluyen el enrutamiento por vector de distancia y el enrutamiento por estado del enlace. La mayoría de las redes usan algunos de éstos. Otros temas importantes relativos al enrutamiento son el enrutamiento jerárquico, el enrutamiento para *hosts* móviles, el enrutamiento por difusión, el enrutamiento por multidifusión y el enrutamiento en redes de igual a igual.

Las subredes pueden congestionarse, aumentando el retardo y reduciendo la velocidad real de transporte de los paquetes. Los diseñadores de redes intentan evitar la congestión mediante un diseño adecuado. Las técnicas incluyen política de retransmisión, almacenamiento en caché, control de flujo, entre otras. Si ocurre congestión, habrá que encargarse de ella. Pueden enviarse paquetes reguladores de regreso, desecharse parte de la carga, y aplicarse otros métodos.

El siguiente paso que va más allá de sólo tratar con la congestión es tratar realmente de alcanzar una calidad prometida de servicio. Los métodos que se pueden utilizar para esto incluyen el almacenamiento en el búfer en el cliente, el modelado de tráfico, la reservación de recursos y el control de acceso. Entre los métodos que se han diseñado para una buena calidad del servicio se encuentran los servicios integrados (incluyendo RSVP), los servicios diferenciados y MPLS.

Las redes difieren de varias maneras, por lo que cuando se conectan múltiples redes pueden ocurrir problemas. A veces los problemas pueden superarse enviando los paquetes mediante túneles a través de una red distinta, pero si las redes de origen y destino son diferentes, este método falla. Cuando las diferentes redes tienen diferentes tamaños máximos de paquete, puede requerirse una fragmentación.

La Internet posee una copiosa variedad de protocolos relacionados con la capa de red. Éstos incluyen protocolo de capa de transporte de datos, IP, pero también los protocolos de control ICMP, ARP y RARP, y los protocolos de enrutamiento OSPF y BGP. Internet se está quedando sin direcciones IP, por lo que se ha desarrollado una versión nueva de IP, el IPv6.

PROBLEMAS

1. Indique dos aplicaciones de ejemplo para las cuales es adecuado un servicio orientado a conexiones. Luego dé dos ejemplos en los que el servicio sin conexiones es lo mejor.
2. ¿Hay circunstancias en las cuales un servicio de circuito virtual entregará (o cuando menos debería entregar) paquetes en desorden? Explique.
3. Las subredes de datagramas enrutan cada paquete como unidad separada, independiente de las demás. Las subredes de circuitos virtuales no tienen que hacer esto, ya que cada paquete de datos sigue una ruta predeterminada. ¿Significa esto que las subredes de circuitos virtuales no necesitan la capacidad de enrutar paquetes aislados de un origen arbitrario a un destino arbitrario? Explique su respuesta.
4. Dé tres ejemplos de parámetros de protocolo que podrían negociarse al establecer una conexión.
5. Considere el siguiente problema de diseño que concierne a la implementación del servicio de circuitos virtuales. Si los circuitos virtuales son internos a la subred, cada paquete de datos debe tener un encabezado de 3 bytes, y cada enrutador debe destinar hasta 8 bytes de almacenamiento para la identificación de circuitos. Si se usan datagramas de manera interna, se requieren encabezados de 15 bytes, pero no se requiere espacio de tabla en los enrutadores. La capacidad de transmisión cuesta 1 centavo para cada 10^6 bytes, por salto. Puede comprarse memoria de enrutamiento por 1 centavo por byte y se de-

precia en dos años (tomando en cuenta que la semana laboral es de 40 horas). Estadísticamente, la sesión promedio dura 1000 seg, tiempo durante el cual se transmiten 200 paquetes. El paquete medio requiere cuatro saltos. ¿Qué implementación es más económica, y por cuánto?

6. Suponiendo que todos los enrutadores y *hosts* están trabajando de manera adecuada y que el software de ambos está libre de errores, ¿hay alguna posibilidad, por pequeña que sea, de que un paquete sea entregado al destino equivocado?

7. Considere la red de la figura 5-7, pero ignore los pesos de las líneas. Suponga que dicha red utiliza la inundación como algoritmo de enrutamiento. Si un paquete enviado mediante *A* a *D* tiene una cuenta máxima de salto de 3, liste todas las rutas que éste tomará. También mencione cuántos saltos merecedores de ancho de banda realiza.

8. Dé una heurística sencilla para encontrar dos rutas a través de una red de origen dado a un destino dado que pueda sobrevivir a la pérdida de cualquier línea de comunicación (suponiendo que existen dos de tales rutas). Los enrutadores se consideran lo bastante confiables, por lo que no es necesario preocuparse por la posibilidad de caída de los enrutadores.

9. Considere la subred de la figura 5-13(a). Se usa enrutamiento por vector de distancia y acaban de llegar los siguientes vectores al enrutador *C*: de *B:* (5, 0, 8, 12, 6, 2); de *D:* (16, 12, 6, 0, 9, 10), y de *E:* (7, 6, 3, 9, 0, 4). Los retardos medios a *B*, *D* y *E* son 6, 3 y 5, respectivamente. ¿Cuál es la nueva tabla de enrutamiento de *C*? Indique tanto la línea de salida a usar como el retardo esperado.

10. Si en una red de 50 enrutadores los retardos se registran como números de 8 bits y se intercambian vectores de retardo dos veces por segundo, ¿qué ancho de banda por línea dúplex total es consumido por el algoritmo de enrutamiento distribuido? Suponga que cada enrutador tiene tres líneas a los demás enrutadores.

11. En la figura 5-14 el OR booleano de los dos grupos de bits ACF es de 111 en cada fila. ¿Es éste un mero accidente, o es cierto para todas las subredes en todas las circunstancias?

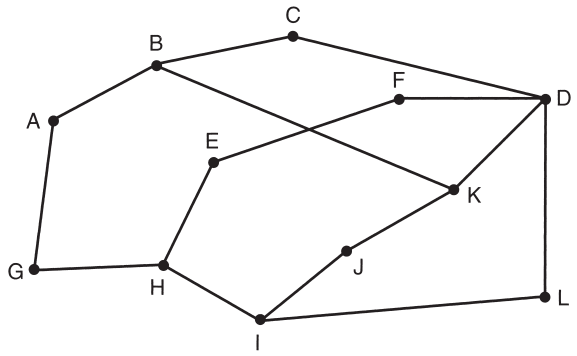
12. Para un enrutamiento jerárquico con 4800 enrutadores, ¿cuál región y tamaños de clúster deberían elegirse para minimizar el tamaño de la tabla de enrutamiento para una jerarquía de tres capas? Un buen lugar de inicio es la hipótesis de que una solución *k* clústeres de *k* regiones de *k* enrutadores está cerca de ser óptima, lo que significa que *k* es aproximadamente la raíz cúbica de 4800 (cerca de 16). Utilice la prueba y el error para verificar las combinaciones en las que los tres parámetros están en el límite de 16.

13. En el texto se indicó que, cuando un *host* móvil no está en casa, los paquetes enviados a su LAN base son interceptados por su agente de base en esa LAN. En una red IP de una LAN 802.3, ¿cómo logra esta intercepción el agente de base?

14. Viendo la subred de la figura 5-6, ¿cuántos paquetes se generan por una difusión de *B*, usando
(a) reenvío por ruta invertida?
(b) árbol sumidero?

15. Considere la red de la figura 5-16(a). Imagine que entre *F* y *G* se agrega una línea nueva pero el árbol sumidero de la figura 5-16(b) permanece sin cambios. ¿Qué cambios ocurren en la figura 5-16(c)?

16. Calcule un árbol de expansión multidifusión para el enrutador *C* de la siguiente subred para un grupo con miembros en los enrutadores *A*, *B*, *C*, *D*, *E*, *F*, *I* y *K*.



17. En la figura 5-20, ¿los nodos *H* o *I* difunden alguna vez en la búsqueda mostrada iniciada en *A*?

18. Suponga que el nodo *B* de la figura 5-20 ha reiniciado y no tiene información de enrutamiento en sus tablas. De repente necesita una ruta a *H*. Envía difusiones con *TTL* establecido a 1, 2, 3, etcétera. ¿Cuántas rondas da para encontrar la ruta?

19. En la versión más simple del algoritmo Chord para búsqueda de igual a igual, las búsquedas no utilizan la tabla *finger*. En su lugar, se ejecutan en forma lineal alrededor del círculo, en cualquier dirección. ¿Puede un nodo predecir de manera precisa en qué dirección debe buscar? Explique su respuesta.

20. Considere el círculo Chord de la figura 5-24. Suponga que el nodo 10 de repente se activa. ¿Esto afecta la tabla *finger* del nodo 1; de ser así, cómo?

21. Como posible mecanismo de control de congestión en una subred que usa circuitos virtuales internamente, un enrutador podría abstenerse de confirmar la recepción de un paquete hasta que (1) sabe que su última transmisión por el circuito virtual se recibió con éxito y que (2) tiene un búfer libre. Por sencillez, suponga que los enrutadores usan un protocolo de parada y espera y que cada circuito virtual tiene un búfer dedicado a él para cada destino del tráfico. Si se quieren *T* seg para transmitir un paquete (de datos o de confirmación de recepción) y hay *n* enrutadores en la ruta, ¿cuál es la velocidad con que se entregan paquetes al *host* de destino? Suponga que los errores de transmisión son poco frecuentes y que la conexión *host*-enrutador es infinitamente rápida.

22. Una subred de datagramas permite que los enrutadores puedan deshacerse de paquetes cuando lo necesiten. La probabilidad de que un enrutador descarte un paquete es de *p*. Considere el caso de un *host* de origen conectado al enrutador de origen, que está conectado al enrutador de destino, y por él al *host* de destino. Si cualquiera de los enrutadores descarta un paquete, el *host* de origen tarde o temprano termina la temporización e intenta de nuevo. Si tanto las líneas *host*-enrutador como enrutador-enrutador se cuentan como saltos, ¿cuál es la media de

(a) saltos que da un paquete por transmisión?
(b) transmisiones que hace un paquete?
(c) saltos requeridos por paquete recibido?

23. Describa dos diferencias principales entre el método de bit de advertencia y el método RED.

24. Cite una razón por la que el algoritmo de cubeta con goteo debe tener sólo un paquete por intervalo, independientemente del tamaño del paquete.

25. La variante de conteo de bytes del algoritmo de cubeta con goteo se usa en cierto sistema. La regla es que pueden enviarse por intervalo un paquete de 1024 bytes, dos paquetes de 512 bytes, etcétera. Indique una restricción seria de este sistema que no se menciona en el texto.

26. Una red ATM usa un esquema de cubeta con *tokens* para la conformación de tráfico. Se pone un *token* nuevo en la cubeta cada 5 μseg. Cada *token* cabe en una celda, que contiene 48 bytes de datos ¿Cuál es la tasa de datos máxima sustentable?

27. Una computadora de una red de 6 Mbps se regula mediante una cubeta con *tokens*. La cubeta con *tokens* se llena a razón de 1 Mbps. Inicialmente está llena a su capacidad máxima de 8 megabits. ¿Durante cuánto tiempo puede la computadora transmitir a 6 Mbps?

28. Imagine una especificación de flujo que tiene un tamaño máximo de paquete de 1000 bytes, una tasa de cubeta con *tokens* de 10 millones de bytes/seg, un tamaño de cubeta con *tokens* de 1 millón de bytes y una tasa máxima de transmisión de 50 millones de bytes/seg. ¿Cuánto tiempo puede durar una ráfaga a la velocidad máxima?

29. La red de la figura 5-37 utiliza RSVP con árboles de multidifusión para los *hosts* 1 y 2. Suponga que el *host* 3 solicita un canal de ancho de banda de 2 MB/seg para un flujo del *host* 1 y otro canal de ancho de banda de 1 MB/seg para un flujo del *host* 2. Al mismo tiempo, el *host* 4 solicita un canal de ancho de banda de 2 MB/seg para un flujo del *host* 1 y el *host* 5 solicita un canal de ancho de banda de 1 MB/seg para un flujo del *host* 2. ¿Cuánto ancho de banda se reservará para estas solicitudes en los enrutadores *A*, *B*, *C*, *E*, *H*, *J*, *K* y *L*?

30. La CPU de un enrutador puede procesar 2 millones de paquetes/seg. La carga que se le ofrece es 1.5 millones de paquetes/seg. Si una ruta del origen al destino contiene 10 enrutadores, ¿cuánto tiempo tardan las CPUs en encolar y dar servicio?

31. Considere el usuario de los servicios diferenciados con reenvío expedito. ¿Hay alguna garantía de que los paquetes expeditos experimenten un retardo más pequeño que los paquetes regulares? ¿Por qué sí o por qué no?

32. ¿Es necesaria la fragmentación en interredes de circuitos virtuales concatenados o sólo en los sistemas de datagramas?

33. El entunelamiento a través de una subred de circuitos virtuales concatenada es directo: el enrutador multiprotocolo en un extremo sólo establece un circuito virtual al otro extremo y pasa los paquetes a través de él. ¿El entunelamiento también puede utilizarse en las subredes de datagramas? ¿De ser así, cómo?

34. Suponga que el *host A* está conectado a un enrutador *R* 1, *R* 1 está conectado a otro enrutador, *R* 2, y *R* 2 está conectado al *host B*. Suponga que un mensaje TCP que contiene 900 bytes de datos y 20 bytes de encabezados TCP se pasa al código IP en el *host A* para entregarlo a *B*. Muestre los campos *Longitud total, Identificación*, *DF*, *MF* y *Desplazamiento del fragmento* del encabezado IP en cada paquete transmitido a través de los tres enlaces. Suponga que el enlace *A-R1* puede soportar un tamaño máximo de trama de 1024 bytes, así como un encabezado de trama de 14 bytes; el enlace *R1-R2* puede soportar un tamaño máximo de trama de 512 bytes, así como un encabezado de trama de 8 bytes, y el enlace *R2-B* puede soportar un tamaño máximo de trama de 512 bytes, incluyendo un encabezado de trama de 12 bytes.

35. Un enrutador está eliminando paquetes IP cuya longitud máxima (datos más encabezado) es de 1024 bytes. Suponiendo que los paquetes vivan por 10 seg, ¿cuál es la velocidad máxima a la que el enrutador puede operar sin el peligro de desbordar el espacio de números ID del datagrama IP?

36. Un datagrama IP que utiliza la opción *Enrutamiento de origen estricto* tiene que fragmentarse. ¿Cree que la opción se copia en cada fragmento, o con colocarlo en el primer fragmento es suficiente? Explique su respuesta.

37. Suponga que en lugar de usar 16 bits para la parte de red de una dirección clase B, se hubieran usado 20 bits. ¿Cuántas redes clase B habría?

38. Convierta la dirección de IP cuya representación hexadecimal es C22F1582 a notación decimal con puntos.

39. Una red en Internet tiene una máscara de subred de 255.255.240.0. ¿Cuál es la cantidad máxima de *hosts* que puede manejar?

40. Hay una gran cantidad de direcciones IP consecutivas, comenzando en 198.16.0.0. Suponga que cuatro organizaciones, *A*, *B*, *C* y *D*, solicitan 4000, 2000, 4000, y 8000 direcciones, respectivamente, y en ese orden. Dé la primera dirección asignada, la última dirección IP asignada y la máscara en la notación *w.x.y.z/s* para cada una de ellas.

41. Un enrutador acaba de recibir las siguientes nuevas direcciones IP: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21 y 57.6.120.0/21. Si todas éstas utilizan la misma línea de salida, ¿se pueden agregar? De ser así, ¿a qué? Si no, ¿por qué?

42. El conjunto de direcciones IP de 29.18.0.0 a 19.18.128.255 se ha agregado a 29.18.0.0/17. Sin embargo, hay un hueco de 1024 direcciones sin asignar de 29.18.60.0 a 29.18.63.255 que de repente se asignan a un *host* que utiliza una línea de salida diferente. Ahora es necesario dividir la dirección agregada en sus bloques constituyentes, agregar el nuevo bloque a la tabla y, después, ver si es posible alguna reagregación? Si no lo es, ¿qué se puede hacer en lugar de eso?

43. Un enrutador tiene las siguientes entradas (CIDR) en su tabla de enrutamiento:

Dirección/máscara	**Siguiente salto**
135.46.56.0/22	Interfaz 0
135.46.60.0/22	Interfaz 1
192.53.40.0/23	Enrutador 1
predeterminada	Enrutador 2

Para cada una de las siguientes direcciones IP, ¿qué hace el enrutador si llega un paquete con esa dirección?

(a) 135.46.63.10
(b) 135.46.57.14
(c) 135.46.52.2
(d) 192.53.40.7
(e) 192.53.56.7

44. Muchas compañías tienen la política de contar con dos (o más) enrutadores que conecten a la compañía a Internet para proporcionar alguna redundancia en caso de que una de ellas falle. ¿Esta política aún es posible con NAT? Explique su respuesta.

45. Usted explica el protocolo ARP a un amigo. Cuando usted termina su explicación, él dice: "Ya entiendo. ARP proporciona un servicio a la capa de red, por lo que es parte de la capa de enlace de datos". ¿Qué le diría a su amigo?

46. ARP y RARP asignan direcciones de un espacio a otro. En este sentido, son similares. Sin embargo, sus implementaciones son esencialmente diferentes. ¿En qué aspecto fundamental son diferentes?

47. Describa una forma de reensamblar fragmentos IP en el destino.

48. La mayoría de los algoritmos de reensamble de datagramas IP tienen un temporizador para evitar que un fragmento perdido enlace búferes de reensamble por siempre. Suponga que un datagrama se divide en cuatro fragmentos. Los primeros tres fragmentos llegan y el cuarto se retrasa. En algún momento, el temporizador termina, por lo que se descartan los tres fragmentos de la memoria del receptor. Un poco más tarde, llega el último fragmento. ¿Qué se debería hacer con él?

49. Tanto en IP como en ATM, la suma de verificación cubre sólo el encabezado y no los datos. ¿Por qué supone que se eligió este diseño?

50. Una persona que vive en Boston viaja a Minneápolis, y lleva su computadora portátil. Para su sorpresa, la LAN de su destino en Minneápolis es una LAN IP inalámbrica, por lo que no tiene que conectarse. ¿Para que el correo electrónico y otro tipo de tráfico llegue de manera correcta aún es necesario todo el proceso con los agentes de base y foráneos?

51. IPv6 utiliza direcciones de 16 bytes. Si un bloque de 1 millón de direcciones se asigna cada picosegundo, ¿cuánto tardará la dirección?

52. El campo *Protocolo* utilizado en el encabezado IPv4 no está presente en el encabezado IPv6 fijo. ¿Por qué?

53. Cuando se introduce el protocolo IPv6, ¿tiene que cambiarse el protocolo ARP? De ser así, ¿los cambios son conceptuales o técnicos?

54. Escriba un programa para simular enrutamiento que utilice inundación. Cada paquete debe contener un contador que se decrementa en cada salto. Cuando el contador llega a cero, el paquete se descarta. El tiempo es discreto y cada línea maneja un paquete por intervalo de tiempo. Cree tres versiones del programa: todas las líneas están inundadas, todas las líneas, excepto la de entrada, están inundadas, y sólo las k mejores líneas (elegidas de manera estática) están inundadas. Compare la inundación con el enrutamiento determinista ($k = 1$) con base en el retardo y el ancho de banda utilizado.

55. Escriba un programa que simule una red de computadoras usando tiempo discreto. El primer paquete de cada cola de enrutador da un salto por intervalo de tiempo. Cada enrutador sólo tiene un número finito de búferes. Si un paquete llega y no hay espacio para él, se descarta y no se retransmite. En su lugar, hay un protocolo de extremo a extremo, lleno de terminaciones de temporización y paquetes de confirmación de recepción, que en algún momento regenera dicho paquete del enrutador de origen. Grafique la velocidad real de transporte de la red como una función del intervalo de terminación de temporizador de extremo a extremo, con parámetros de tasa de error.

56. Escriba una función para realizar el reenvío en un enrutador IP. El procedimiento tiene un parámetro, una dirección IP. También tiene acceso a una tabla global que consiste de un arreglo de tres variables. Cada arreglo contiene tres enteros: una dirección IP, una máscara de subred y la línea de salida a utilizar. La función usa CIDR para buscar la dirección IP en la tabla y regresa la línea a utilizar como su valor.

57. Utilice los programas *traceroute* (UNIX) o *tracert* (Windows) para trazar la ruta de su computadora a varias universidades de otros continentes. Haga una lista de los enlaces transoceánicos que ha descubierto. Algunos sitios para probar son:

www.berkeley.edu (California)
www.mit.edu (Massachusetts)
www.vu.nl (Amsterdam)
www.ucl.ac.uk (Londres)
www.usyd.edu.au (Sydney)
www.u-tokyo.ac.jp (Tokyo)
www.uct.ac.za (Cape Town)

6

LA CAPA DE TRANSPORTE

La capa de transporte no es una capa más. Es el corazón de toda la jerarquía de protocolos. La tarea de esta capa es proporcionar un transporte de datos confiable y económico de la máquina de origen a la máquina de destino, independientemente de la red o redes físicas en uso. Sin la capa de transporte, el concepto total de los protocolos en capas tendría poco sentido. En este capítulo estudiaremos en detalle la capa de transporte, incluidos sus servicios, diseño, protocolos y desempeño.

6.1 EL SERVICIO DE TRANSPORTE

En las siguientes secciones daremos una introducción al servicio de transporte. Veremos el tipo de servicio proporcionado a la capa de aplicación. Veremos el tipo de servicio que se proporciona a la capa de aplicación. Para que el tema del servicio de transporte quede claro, analizaremos dos conjuntos de primitivas de la capa de transporte. Primero analizaremos uno muy sencillo (e hipotético) para mostrar las ideas básicas. Después veremos la interfaz que se utiliza comúnmente en Internet.

6.1.1 Servicios proporcionados a las capas superiores

La meta fundamental de la capa de transporte es proporcionar un servicio eficiente, confiable y económico a sus usuarios, que normalmente son procesos de la capa de aplicación. Para lograr este objetivo, la capa de transporte utiliza los servicios proporcionados por la capa de red. El hardware

o software de la capa de transporte que se encarga del trabajo se llama **entidad de transporte,** la cual puede estar en el *kernel* (núcleo) del sistema operativo, en un proceso de usuario independiente, en un paquete de biblioteca que forma parte de las aplicaciones de red o en la tarjeta de red. En la figura 6-1 se ilustra la relación (lógica) entre las capas de red, transporte y aplicación.

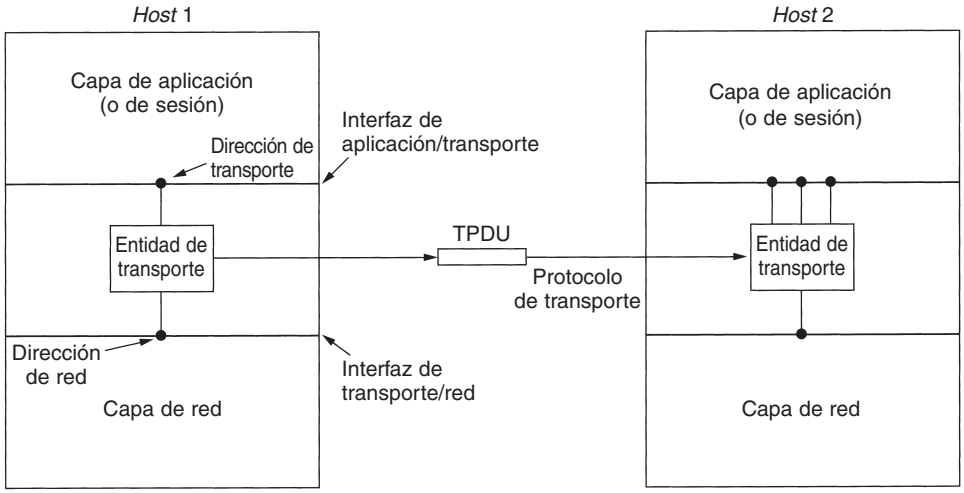


Figura 6-1. Las capas de red, transporte y aplicación.

Así como hay dos tipos de servicio de red, orientado y no orientado a la conexión, hay dos tipos de servicio de transporte. El servicio de transporte orientado a la conexión es parecido en muchos sentidos al servicio de red orientado a la conexión. En ambos casos, las conexiones tienen tres fases: establecimiento, transferencia de datos y liberación (o terminación). El direccionamiento y el control de flujo también son semejantes en ambas capas. Además, el servicio de transporte no orientado a la conexión es muy parecido al servicio de red no orientado a la conexión.

La pregunta obvia es: ¿si el servicio de la capa de transporte es tan parecido al de la capa de red, por qué hay dos capas diferentes? ¿Por qué no es suficiente una sola capa? La respuesta es sutil, pero crucial, y nos remite a la figura 1-9. El código de transporte se ejecuta por completo en las máquinas de los usuarios, pero la capa de red, por lo general, se ejecuta en los enrutadores, los cuales son operados por la empresa portadora (por lo menos en el caso de una red de área amplia). ¿Qué sucede si la capa de red ofrece un servicio poco confiable? ¿Qué tal si esa capa pierde paquetes con frecuencia? ¿Qué ocurre si los enrutadores se caen de cuando en cuando?

Problemas, eso es lo que ocurre. Los usuarios no tienen control sobre la capa de red, por lo que no pueden resolver los problemas de un mal servicio usando mejores enrutadores o incrementando el manejo de errores en la capa de enlace de datos. La única posibilidad es poner encima de la capa de red otra capa que mejore la calidad del servicio. Si, en una subred orientada a la conexión, a la mitad de una transmisión larga se informa a una entidad de transporte que su conexión de red ha sido terminada de manera abrupta, sin indicación de lo sucedido a los datos actualmente en tránsito, la entidad puede establecer una nueva conexión de red con la entidad de transporte

remota. Usando esta nueva conexión de red, la entidad puede enviar una solicitud a su igual preguntando cuáles datos llegaron y cuáles no, y reiniciar a partir de donde se originó la interrupción.

Esencialmente, la existencia de la capa de transporte hace posible que el servicio de transporte sea más confiable que el servicio de red subyacente. La capa de transporte puede detectar y compensar paquetes perdidos y datos alterados. Más aún, las primitivas del servicio de transporte se pueden implementar como llamadas a procedimientos de biblioteca con el propósito de que sean independientes de las primitivas del servicio de red, las cuales podrían variar considerablemente entre las redes (por ejemplo, el servicio LAN no orientado a la conexión puede ser bastante diferente del servicio WAN orientado a la conexión). Al ocultar el servicio de red detrás de un conjunto de primitivas de servicio de transporte, el cambio del servicio de red simplemente requiere reemplazar un conjunto de procedimientos de biblioteca por otro que haga lo mismo con un servicio subyacente distinto.

Gracias a la capa de transporte, es posible escribir programas de aplicación usando un conjunto estándar de primitivas, y que estos programas funcionen en una amplia variedad de redes sin necesidad de preocuparse por lidiar con diferentes interfaces de subred y transmisiones no confiables. Si ninguna red real tuviera fallas, y si todas tuvieran las mismas primitivas de servicio y se pudiera garantizar que nunca jamás cambiaran, tal vez la capa de transporte sería innecesaria. Sin embargo, en el mundo real esta capa cumple la función clave de aislar a las capas superiores de la tecnología, el diseño y las imperfecciones de la subred.

Por esta razón, mucha gente establece una distinción entre las capas 1 a 4, por una parte, y la(s) capa(s) por encima de la 4, por la otra. Las cuatro capas inferiores pueden verse como el **proveedor del servicio de transporte,** y la(s) capa(s) superiores son el **usuario del servicio de transporte.** Esta distinción entre proveedor y usuario tiene un impacto considerable en el diseño de las capas y pone a la capa de transporte en una posición clave, ya que constituye el límite principal entre el proveedor y el usuario del servicio confiable de transmisión de datos.

6.1.2 Primitivas del servicio de transporte

Para permitir que los usuarios accedan al servicio de transporte, la capa de transporte debe proporcionar algunas operaciones a los programas de aplicación, es decir, una interfaz del servicio de transporte. Cada servicio de transporte tiene su propia interfaz. Con el propósito de ver los aspectos básicos, en esta sección examinaremos primero un servicio de transporte sencillo (hipotético) y su interfaz. En la siguiente sección veremos un ejemplo real.

El servicio de transporte es parecido al servicio de red, pero hay algunas diferencias importantes. La principal es que el propósito del servicio de red es modelar el servicio ofrecido por las redes reales, con todos sus problemas. Las redes reales pueden perder paquetes, por lo que el servicio de red generalmente no es confiable.

En cambio, el servicio de transporte (orientado a la conexión) sí es confiable. Claro que las redes reales no están libres de errores, pero ése es precisamente el propósito de la capa de transporte: ofrecer un servicio confiable en una red no confiable.

Como ejemplo, considere dos procesos conectados mediante canalizaciones en UNIX. Ambos consideran que la conexión entre ellos es perfecta. No quieren saber de confirmaciones de recepción, paquetes perdidos, congestión ni nada por el estilo. Lo que quieren es una conexión 100 por ciento confiable. El proceso *A* pone datos en un extremo de la canalización y el proceso *B* los saca por el otro extremo. Ésta es la esencia del servicio de transporte orientado a la conexión: ocultar las imperfecciones del servicio de red para que los procesos usuarios puedan dar por hecho simplemente la existencia de un flujo de bits libre de errores.

Como nota al margen, la capa de transporte también puede proporcionar un servicio no confiable (de datagramas), pero hay muy poco que decir al respecto, por lo que en este capítulo nos concentraremos principalmente en el servicio de transporte orientado a la conexión. Sin embargo, hay algunas aplicaciones que se benefician del transporte no orientado a la conexión, como la computación cliente-servidor y la multimedia de flujo continuo, por lo que veremos algo sobre ellas más adelante.

Una segunda diferencia entre los servicios de red y de transporte es a quién están dirigidos. El servicio de red lo usan únicamente las entidades de transporte. Pocos usuarios escriben sus propias entidades de transporte y, por lo tanto, pocos usuarios o programas llegan a ver los aspectos internos del servicio de red. En contraste, muchos programas (y, por lo tanto, programadores) ven las primitivas de transporte. En consecuencia, el servicio de transporte debe ser adecuado y fácil de usar.

Para tener una idea de lo que podría ser el servicio de transporte, considere las cinco primitivas listadas en la figura 6-2. Esta interfaz de transporte ciertamente es sencilla, pero muestra la esencia de lo que debe hacer una interfaz de transporte orientada a la conexión: permite que los programas de aplicación establezcan, usen y liberen conexiones, lo cual es suficiente para muchas aplicaciones.

Primitiva	Paquete enviado	Significado
LISTEN	(ninguno)	Se bloquea hasta que algún proceso intenta la conexión
CONNECT	CONNECTION REQ.	Intenta activamente establecer una conexión
SEND	DATA	Envía información
RECEIVE	(ninguno)	Se bloquea hasta que llega un paquete DATA
DISCONNECT	DISCONNECTION REQ.	Este lado quiere liberar la conexión

Figura 6-2. Primitivas de un servicio de transporte sencillo.

Para ver cómo podrían usarse estas primitivas, considere una aplicación con un servidor y cierta cantidad de clientes remotos. Para comenzar, el servicio ejecuta una primitiva LISTEN, normalmente llamando a un procedimiento de biblioteca que hace una llamada de sistema para bloquear al servidor hasta la aparición de un cliente. Cuando un cliente desea comunicarse con el servidor, ejecuta una primitiva CONNECT. La entidad de transporte ejecuta esta primitiva bloqueando al

invocador y enviando un paquete al servidor. En la carga útil de este paquete se encuentra un mensaje de capa de transporte encapsulado, dirigido a la entidad de transporte del servidor.

Aquí es pertinente una nota rápida sobre la terminología. A falta de un mejor término, usaremos las siglas poco elegantes de **TPDU** (**Unidad de Datos del Protocolo de Transporte**) para referirnos a los mensajes enviados de una entidad de transporte a otra. Por lo tanto, las TPDUs (intercambiadas por la capa de transporte) están contenidas en paquetes (intercambiados por la capa de red). A su vez, los paquetes están contenidos en tramas (intercambiados por la capa de enlace de datos). Cuando llega una trama, la capa de enlace de datos procesa el encabezado de la trama y pasa el contenido del campo de carga útil de la trama a la entidad de red. Esta última procesa el encabezado del paquete y pasa el contenido de la carga útil del paquete a la entidad de transporte. Este anidamiento se ilustra en la figura 6-3.

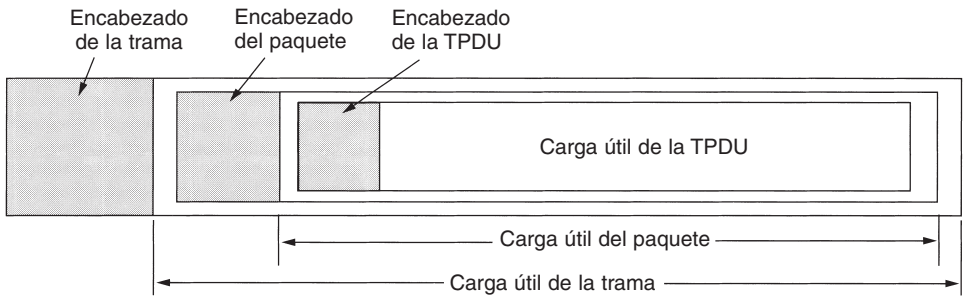


Figura 6-3. Anidamiento de las TPDUs, los paquetes y las tramas.

Regresando a nuestro ejemplo de cliente-servidor, la llamada CONNECT del cliente ocasiona el envío de una TPDU CONNECTION REQUEST (solicitud de conexión) al servidor. Al llegar ésta, la entidad de transporte verifica que el servidor esté bloqueado en LISTEN (es decir, esté interesado en manejar solicitudes). A continuación desbloquea el servidor y envía una TPDU CONNECTION ACCEPTED (conexión aceptada) de regreso al cliente. Al llegar esta TPDU, el cliente se desbloquea y se establece la conexión.

Ahora pueden intercambiarse datos usando las primitivas SEND y RECEIVE. En la forma más simple, cualquiera de las dos partes puede emitir una RECEIVE (bloqueadora) para esperar que la otra parte emita una SEND. Al llegar la TPDU, el receptor se desbloquea y puede procesar la TPDU y enviar una respuesta. Mientras ambos lados puedan llevar el control de quién tiene el turno para transmitir, este esquema funciona bien.

Observe que en la capa de transporte, incluso un intercambio de datos unidireccional es más complicado que en la capa de red. También se confirmará (tarde o temprano) la recepción de cada paquete de datos enviado. Asimismo, la recepción de los paquetes que llevan TPDUs de control se confirmará de manera implícita o explícita. Estas confirmaciones son manejadas por las entidades de transporte usando el protocolo de capa de red, y son transparentes para los usuarios de transporte. De la misma forma, las entidades de transporte necesitarán preocuparse por los temporizadores

y las retransmisiones. Los usuarios de transporte no se enteran de ningún aspecto de esta mecánica. Para ellos, una conexión es un conducto de bits confiable: un usuario mete bits en él y mágicamente aparecen en el otro lado. Esta capacidad de ocultar la complejidad es la razón por la cual los protocolos en capas son herramientas tan poderosas.

Cuando ya no se necesita una conexión, debe liberarse para desocupar espacio en las tablas de las dos entidades de transporte. La desconexión tiene dos variantes: asimétrica y simétrica. En la variante asimétrica, cualquiera de los dos usuarios de transporte puede emitir una primitiva DISCONNECT, que resulta en el envío de una TPDU DISCONNECT a la entidad de transporte remota. A su llegada, se libera la conexión.

En la variante simétrica, cada parte se cierra por separado, independientemente de la otra. Cuando una de las partes emite una DISCONNECT, quiere decir que ya no tiene más datos por enviar, pero aún está dispuesta a recibir datos de la otra parte. En este modelo, una conexión se libera cuando ambas partes han emitido una primitiva DISCONNECT.

En la figura 6-4 se presenta un diagrama de estado del establecimiento y liberación de una conexión con estas primitivas sencillas. Cada transición es activada por algún evento, ya sea una primitiva ejecutada por el usuario de transporte local o la llegada de un paquete. Por sencillez, aquí suponemos que la confirmación de recepción de cada TPDU se realiza por separado. También suponemos que se usa un modelo de desconexión simétrica, y que el cliente la realiza primero. Cabe señalar que este modelo es muy poco refinado. Más tarde veremos un modelo más realista.

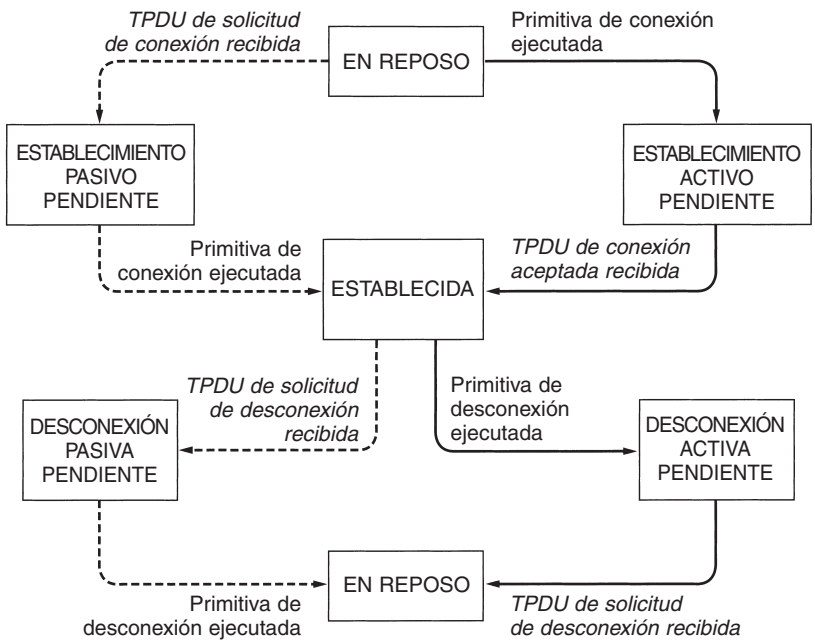


Figura 6-4. Diagrama de estado de un esquema sencillo de manejo de conexiones. Las transiciones escritas en cursivas son causadas por llegadas de paquetes. Las líneas continuas muestran la secuencia de estados del cliente. Las líneas punteadas muestran la secuencia de estados del servidor.

6.1.3 *Sockets* de Berkeley

Inspeccionemos brevemente otro grupo de primitivas de transporte, las primitivas de *socket* usadas en el UNIX de Berkeley para el TCP. Éstas se listan en la figura 6-5. En términos generales, las primitivas siguen el modelo de nuestro primer ejemplo, pero ofrecen más características y flexibilidad. No veremos aquí las TPDUs correspondientes. Ese análisis tendrá que esperar hasta que estudiemos el TCP posteriormente en este capítulo.

Primitiva	Significado
SOCKET	Crea un nuevo punto terminal de comunicación
BIND	Adjunta una dirección local a un *socket*
LISTEN	Anuncia la disposición a aceptar conexiones; indica el tamaño de cola
ACCEPT	Bloquea al invocador hasta la llegada de un intento de conexión
CONNECT	Intenta establecer activamente una conexión
SEND	Envía datos a través de la conexión
RECEIVE	Recibe datos de la conexión
CLOSE	Libera la conexión

Figura 6-5. Primitivas de *socket* para TCP.

Las primeras cuatro primitivas de la lista son ejecutadas en ese orden por los servidores. La primitiva SOCKET crea un nuevo punto de comunicación y le asigna espacio en las tablas de la entidad de transporte. Los parámetros de la llamada especifican el formato de direccionamiento que se utilizará, el tipo de servicio deseado (por ejemplo, flujo confiable de bytes) y el protocolo. Una llamada SOCKET con éxito devuelve un descriptor de archivo ordinario que se utiliza con las siguientes llamadas, de la misma manera que lo hace una llamada OPEN.

Los *sockets* recién creados no tienen direcciones de red. Éstas se asignan mediante la primitiva BIND. Una vez que un servidor ha destinado una dirección a un *socket*, los clientes remotos pueden conectarse a él. La razón para que la llamada SOCKET no cree directamente una dirección es que algunos procesos se encargan de sus direcciones (por ejemplo, han estado usando su misma dirección durante años y todos la conocen), mientras que otros no lo hacen.

A continuación viene la llamada LISTEN, que asigna espacio para poner en cola las llamadas entrantes por si varios clientes intentan conectarse al mismo tiempo. A diferencia de la llamada LISTEN de nuestro primer ejemplo, en el modelo de *sockets* LISTEN no es una llamada bloqueadora.

Para bloquearse en espera de una conexión entrante, el servidor ejecuta una primitiva ACCEPT. Cuando llega una TPDU solicitando una conexión, la entidad de transporte crea un *socket* nuevo con las mismas propiedades que el original y devuelve un descriptor de archivo para él. A continuación, el servidor puede ramificar un proceso o subproceso para manejar la conexión en el *socket* nuevo y regresar a esperar la siguiente conexión en el *socket* original. ACCEPT regresa un descriptor de archivo normal, que puede utilizarse para leer y escribir de la forma estándar, al igual que con los archivos.

Ahora veamos el cliente. Aquí también debe crearse un *socket* primero usando la primitiva SOCKET, pero no se requiere BIND, puesto que la dirección usada no le importa al servidor. La primitiva CONNECT bloquea al invocador y comienza activamente el proceso de conexión. Al completarse éste (es decir, cuando se recibe la TPDU adecuada del servidor) el proceso cliente se desbloquea y se establece la conexión. Ambos lados pueden usar ahora SEND y RECEIVE para transmitir y recibir datos a través de la conexión dúplex total. Las llamadas de sistema READ y WRITE de UNIX también se pueden utilizar si no son necesarias las opciones especiales de SEND y RECEIVE.

La liberación de las conexiones a los *sockets* es simétrica. La conexión se libera cuando ambos lados han ejecutado una primitiva CLOSE.

6.1.4 Un ejemplo de programación de *sockets*: un servidor de archivos de Internet

Veamos el código cliente-servidor de la figura 6-6 como ejemplo del uso de las llamadas de *sockets*. Ahí se muestra un servidor de archivos muy antiguo junto con un cliente de ejemplo que lo utiliza. El código tiene muchas limitaciones (que se analizan más adelante), pero en principio el código del servidor puede compilarse y ejecutarse en cualquier sistema UNIX conectado a Internet. A continuación, el código del cliente puede compilarse y ejecutarse en cualquier otra máquina UNIX conectada a Internet, en cualquier parte del mundo. El código del cliente puede ejecutarse con los parámetros apropiados para obtener cualquier archivo al que el servidor tenga acceso. El archivo se escribe a la salida estándar, la cual, por supuesto, puede redirigirse a un archivo o a un canal.

Veamos primero el código del servidor. Comienza con algunos encabezados estándar, los últimos tres de los cuales contienen las principales definiciones y estructuras de datos relacionadas con Internet. A continuación se encuentra una definición de *SERVER_PORT* como 12345. Este número se eligió de manera arbitraria. Cualquier número que se encuentre entre 1024 y 65535 funcionará siempre y cuando otro proceso no lo esté utilizando. Por supuesto, el cliente y el servidor tienen que utilizar el mismo puerto. Si este servidor llegara a convertirse en un éxito mundial (lo cual no es probable, debido a lo antiguo que es), se le asignaría un puerto permanente debajo de 1024 y aparecería en *www.iana.org*.

Las siguientes dos líneas del servidor definen dos constantes necesarias. La primera determina el tamaño de bloque utilizado para la transferencia de archivos. La segunda determina cuántas conexiones pendientes pueden almacenarse antes de empezar a descartar las excedentes que lleguen.

Después de las declaraciones de variables locales, comienza el código del servidor. Comienza inicializando una estructura de datos que contendrá la dirección IP del servidor. Esta estructura de datos pronto se anexará al *socket* del servidor. La llamada a *memset* establece en 0s toda la estructura de datos. Las siguientes tres asignaciones llenarán tres de sus campos. El último de éstos contiene el puerto del servidor. Las funciones *htonl* y *htons* están relacionadas con la conversión de valores a un formato estándar a fin de que el código se ejecute correctamente tanto en las máquinas *big-endian* (por ejemplo, la SPARC) como en las *little-endian* (por ejemplo, las Pentium). Su semántica exacta no es importante aquí.

A continuación el servidor crea un *socket* y verifica si hay errores (lo cual se indica mediante $s < 0$). En una versión de producción del código, el mensaje de error puede ser mucho más explicativo. La llamada a *setsockopt* es necesaria para permitir que el puerto sea reutilizado a fin de que el servidor se pueda ejecutar de manera indefinida, llenando los campos solicitud tras solicitud. Ahora la dirección IP se enlaza con el *socket* y se realiza una verificación para ver si la llamada a *bind* tuvo éxito. El último paso en la inicialización es la llamada a *listen* para anunciar que el servidor está dispuesto a aceptar llamadas entrantes e indicar al sistema que almacene la cantidad de ellas especificada en *QUEUE_SIZE* en caso de que lleguen más mientras el servidor aún esté procesando la actual. Si la cola está llena y llegan solicitudes adicionales, se descartan irremediablemente.

En este punto el servidor entra a su ciclo principal, al cual nunca abandona. La única forma de detenerlo es desde afuera. La llamada a *accept* bloquea el servidor hasta que algún cliente trata de establecer una conexión con él. Si la llamada a *accept* tiene éxito, se regresa un descriptor de archivo que puede utilizarse para leer y escribir, de la misma forma en la que los descriptores de archivo pueden utilizarse para leer y escribir desde las canalizaciones. Sin embargo, a diferencia de las canalizaciones, que son unidireccionales, los *sockets* son bidireccionales, por lo que *sa* (dirección de *socket*) puede utilizarse para leer de la conexión y también para escribir en ella.

Una vez que se establece la conexión, el servidor lee en ella el nombre del archivo. Si el nombre aún no está disponible, el servidor se bloquea y lo espera. Una vez que obtiene el nombre, el servidor abre el archivo y luego entra en un ciclo que lee de manera alterna bloques del archivo y los escribe en el *socket* hasta que el archivo se haya copiado por completo. A continuación el servidor cierra el archivo y la conexión y espera hasta que aparezca la siguiente conexión. Repite este ciclo de manera indefinida.

Ahora veamos el código del cliente. Para entender su funcionamiento, es necesario comprender cómo se invoca. Suponiendo que se llama *cliente*, una llamada típica es:

```
cliente flits.cs.vu.nl /usr/tom/nombredearchivo >f
```

Esta llamada sólo funciona si el servidor ya se está ejecutando en *flits.cs.vu.nl,* si existe el archivo */usr/tom/nombredearchivo* y si el servidor tiene acceso de lectura a él. Si la llamada es exitosa, el archivo se transfiere a través de Internet y se escribe en *f*, después de lo cual finaliza el programa cliente. Puesto que el servidor continúa después de una transferencia, el cliente puede iniciarse una y otra vez para obtener otros archivos.

El código del cliente inicia con algunas inclusiones y declaraciones. La ejecución comienza verificando si el código ha sido llamado con el número correcto de argumentos (*argc* = 3 significa el nombre del programa más dos argumentos). Observe que *argv* [1] contiene el nombre del servidor (por ejemplo, *flits.cs.vu.nl*) y que *gethostbyname* lo convierte en una dirección IP. Esta función utiliza DNS para buscar el nombre. En el capítulo 7 analizaremos DNS.

A continuación se crea e inicializa un *socket*. Después de esto, el cliente intenta establecer una conexión TCP con el servidor, mediante *connect*. Si el servidor está activo y ejecutándose en la máquina especificada y enlazado a *SERVER_PORT* y si está inactivo o tiene espacio es su cola *listen*, la conexión se establecerá (en algún momento). El cliente utiliza esta conexión para enviar el nombre del archivo escribiendo en el *socket*. El número de bytes enviados es un byte mayor que el propio nombre, puesto que se envía al servidor el byte 0 para indicarle en dónde termina dicho nombre.

```
/* Esta página contiene un programa cliente que puede solicitar un archivo
 * desde el programa servidor de la siguiente página. El servidor responde
 * enviando el archivo completo.
 */

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

#define SERVER_PORT 12345               /* arbitrario, pero el cliente y el
                                        /* servidor deben coincidir */
#define BUF_SIZE 4096                   /* tamaño de bloque para transferencia */

int main(int argc, char **argv)
{
 int c, s, bytes;
 char buf[BUF_SIZE];                    /* búfer para el archivo entrante */
 struct hostent *h;                     /* información sobre el servidor */
 struct sockaddr_in channel;            /* contiene la dirección IP */

 if (argc != 3) fatal("Usage: client server-name file-name");
 h = gethostbyname(argv[1]);            /* busca la dirección IP del host */
 if (!h) fatal("gethostbyname failed");

 s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
 if (s < 0) fatal("socket");
 memset(&channel, 0, sizeof(channel));
 channel.sin_family= AF_INET;
 memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
 channel.sin_port= htons(SERVER_PORT);

 c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
 if (c < 0) fatal("connect failed");

 /* Se ha establecido la conexión. Se envía el nombre del archivo incluyendo
 /* el byte 0 al final. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Obtiene el archivo y lo escribe en la salida estándar. */
  while (1) {
     bytes = read(s, buf, BUF_SIZE);    /* lee del socket */
     if (bytes <= 0) exit(0);           /* verifica el final del archivo */
     write(1, buf, bytes);              /* escribe en la salida estándar */
 }
}

fatal(char *string)
{
 printf("%s\n", string);
 exit(1);
}
```

Figura 6-6. Código del cliente que utiliza *sockets*. El código del servidor se encuentra en la siguiente página.

```
#include <sys/types.h>                  /* Éste es el código del servidor */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345               /* arbitrario, pero el cliente y el
                                        /* servidor deben coincidir */
#define BUF_SIZE 4096                   /* tamaño de bloque para la
                                        /* transferencia */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
 int s, b, l, fd, sa, bytes, on = 1;
 char buf[BUF_SIZE];                    /* búfer para el archivo saliente */
 struct sockaddr_in channel;            /* contiene la dirección IP */

 /* Construye la estructura de la dirección para enlazar el socket. */
 memset(&channel, 0, sizeof(channel));  /* canal cero */
 channel.sin_family = AF_INET;
 channel.sin_addr.s_addr = htonl(INADDR_ANY);
 channel.sin_port = htons(SERVER_PORT);

 /* Apertura pasiva. Espera una conexión. */
 s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* crea el socket */
 if (s < 0) fatal("socket failed");
 setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

 b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
 if (b < 0) fatal("bind failed");

 l = listen(s, QUEUE_SIZE);             /* especifica el tamaño de la cola */
 if (l < 0) fatal("listen failed");

 /* El socket ahora está configurado y enlazado. Espera una conexión y la
 /* procesa. */
 while (1) {
    sa = accept(s, 0, 0);               /* se bloquea para la solicitud de
                                        /* conexión */
    if (sa < 0) fatal("accept failed");

    read(sa, buf, BUF_SIZE);            /* lee el nombre del archivo desde el
                                        /* socket */

    /* Obtiene y regresa el archivo.*/
    fd = open(buf, O_RDONLY);           /* abre el archivo para regresarlo */
    if (fd < 0) fatal("open failed");

    while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* lee del archivo */
          if (bytes <= 0) break;        /* verifica el final del archivo */
          write(sa, buf, bytes);        /* escribe bytes en el socket */
    }
    close(fd);                          /* cierra el archivo */
    close(sa);                          /* cierra la conexión */
 }
}
```

Ahora el cliente entra en un ciclo, lee el archivo bloque por bloque desde el *socket* y lo copia a la salida estándar. Cuando termina, simplemente abandona la conexión.

El procedimiento *fatal* imprime un mensaje de error y termina. El servidor necesita el mismo procedimiento, pero se omitió debido a la falta de espacio en la página. Puesto que el cliente y el servidor se compilan de manera separada y por lo general se ejecutan en computadoras diferentes, no pueden compartir el código de *fatal*.

Estos dos programas (así como otro material relacionado con este libro) se pueden obtener del sitio Web del libro

```
http://www.prenhall.com/tanenbaum
```

haciendo clic en el vínculo Companion Web Site que se encuentra junto a la fotografía de la portada. Dichos programas pueden bajarse y compilarse en cualquier sistema UNIX (por ejemplo, Solaris, BSD, Linux) mediante:

```
cc –o client client.c –lsocket -lnsl
cc –o server server.c –lsocket –lnsl
```

El servidor se inicia con sólo teclear

```
server
```

El cliente necesita dos argumentos, como se mencionó anteriormente. En el sitio Web también hay disponible una versión para Windows.

Sólo como aclaración, este servidor no tiene nada de refinado. Su verificación de errores es insuficiente y su reporte de errores es ordinario. Claramente, nunca ha escuchado sobre la seguridad, y utilizar sólo llamadas de sistema UNIX no es lo más recomendable respecto a independencia de la plataforma. También da por sentados algunos detalles que son técnicamente ilegales, como asumir que el nombre del archivo se ajusta en el búfer y que se transmite de manera inmediata y sin divisiones. Debido a que maneja todas las solicitudes en forma estrictamente secuencial (puesto que sólo tiene un solo subproceso), su desempeño es pobre. A pesar de estas fallas, es un servidor de archivos de Internet completo y funcional. En los ejercicios, se invita al lector a mejorarlo. Para mayor información sobre la programación con *sockets*, vea (Stevens, 1997).

6.2 ELEMENTOS DE LOS PROTOCOLOS DE TRANSPORTE

El servicio de transporte se implementa mediante un **protocolo de transporte** entre las dos entidades de transporte. En ciertos aspectos, los protocolos de transporte se parecen a los protocolos de enlace de datos que estudiamos detalladamente en el capítulo 3. Ambos se encargan del control de errores, la secuenciación y el control de flujo, entre otros aspectos.

Sin embargo, existen diferencias significativas entre los dos, las cuales se deben a diferencias importantes entre los entornos en que operan ambos protocolos, como se muestra en la figura 6-7. En la capa de enlace de datos, dos enrutadores se comunican directamente mediante un canal físico mientras que, en la capa de transporte, ese canal físico es reemplazado por la subred completa. Esta diferencia tiene muchas implicaciones importantes para los protocolos, como veremos en este capítulo.

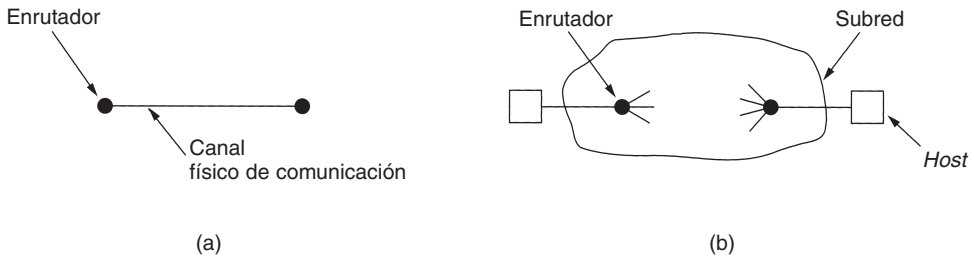


Figura 6-7. (a) Entorno de la capa de enlace de datos. (b) Entorno de la capa de transporte.

Por una parte, en la capa de enlace de datos no es necesario que un enrutador especifique el enrutador con el que quiere comunicarse; cada línea de salida especifica de manera única un enrutador en particular. En la capa de transporte se requiere el direccionamiento explícito de los destinos.

Por otro lado, el proceso de establecimiento de una conexión a través del cable de la figura 6-7(a) es sencillo: el otro extremo siempre está ahí (a menos que se caiga, en cuyo caso no estará ahí). Sea como sea, no hay demasiado que hacer. En la capa de transporte, el establecimiento inicial de la conexión es más complicado, como veremos.

Otra diferencia, muy irritante, entre la capa de enlace de datos y la capa de transporte es la existencia potencial de capacidad de almacenamiento en la subred. Al enviar un enrutador una trama, ésta puede llegar o perderse, pero no puede andar de un lado a otro durante un rato, esconderse en un rincón alejado del mundo y aparecer repentinamente en algún momento inoportuno 30 segundos después. Si la subred usa datagramas y enrutamiento adaptativo internamente, hay una probabilidad nada despreciable de que un paquete pueda almacenarse durante varios segundos y entregarse después. Las consecuencias de esta capacidad de almacenamiento de paquetes en la subred pueden ser desastrosas y requerir el uso de protocolos especiales.

Una última diferencia entre las capas de enlace de datos y de transporte es de cantidad, más que de tipo. Se requieren búferes y control de flujo en ambas capas, pero la presencia de una cantidad de conexiones grande y dinámicamente variable en la capa de transporte puede requerir un enfoque distinto del que se usa en la capa de enlace de datos. En el capítulo 3 vimos que algunos de los protocolos asignan una cantidad fija de búferes a cada línea de modo que, al llegar una trama, siempre hay un búfer disponible. En la capa de transporte, la gran cantidad de conexiones que deben manejarse hace menos atractiva la idea de dedicar muchos búferes a cada una. En las siguientes secciones examinaremos todos estos importantes temas, además de otros.

6.2.1 Direccionamiento

Cuando un proceso (por ejemplo, un usuario) de aplicación desea establecer una conexión con un proceso de aplicación remoto, debe especificar a cuál se conectará. (El transporte no orientado a la conexión tiene el mismo problema: ¿a quién debe enviarse cada mensaje?) El método que normalmente se emplea es definir direcciones de transporte en las que los procesos pueden estar a la

escucha de solicitudes de conexión. En Internet, estos puntos terminales se denominan **puertos.** En las redes ATM se llaman **AAL-SAPs.** Usaremos el término genérico **TSAP** (**Punto de Acceso al Servicio de Transporte**). Los puntos terminales análogos de la capa de red (es decir, direcciones de capa de red) se llaman **NSAP** (**Punto de Acceso al Servicio de Red**). Las direcciones IP son ejemplos de NSAPs.

En la figura 6-8 se ilustra la relación entre el NSAP, el TSAP y la conexión de transporte. Los procesos de aplicación, tanto clientes como servidores, se pueden enlazar por sí mismos a un TSAP para establecer una conexión a un TSAP remoto.Estas conexiones se realizan a través de NSAPs en cada *host,* como se muestra. Como en algunas redes cada computadora tiene un solo NSAP, los TSAPs sirven para distinguir los múltiples puntos terminales de transporte que comparten un NSAP.

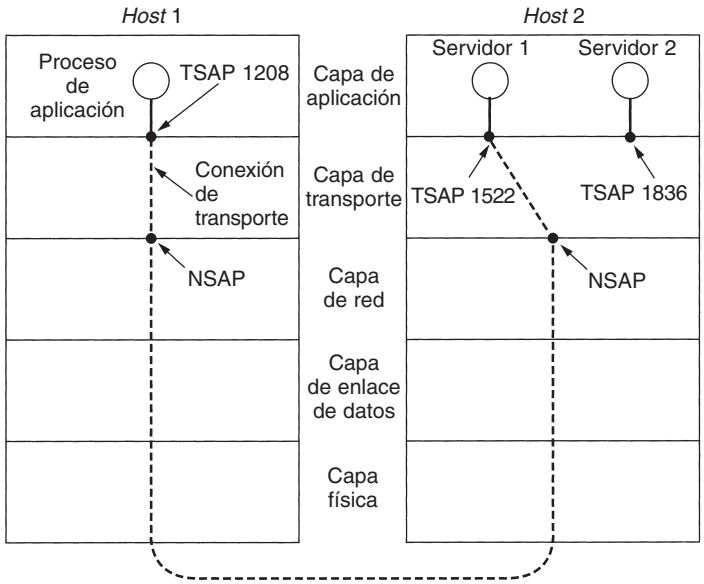


Figura 6-8. TSAPs, NSAPs y conexiones de transporte.

El siguiente es un posible escenario para una conexión de transporte.

1. Un proceso servidor de hora del día del *host* 2 se enlaza con el TSAP 1522 para esperar una llamada entrante. La manera en que un proceso se enlaza con un TSAP está fuera del modelo de red y depende por entero del sistema operativo local. Podría, por ejemplo, usarse una llamada como nuestra LISTEN.

2. Un proceso de aplicación del *host* 1 quiere averiguar la hora del día, por lo que emite una solicitud CONNECT especificando el TSAP 1208 como el origen y el TSAP 1522 como destino. Esta acción al final da como resultado una conexión de transporte que se establece entre el proceso de aplicación del *host* 1 y el servidor 1 del *host* 2.

3. A continuación el proceso de aplicación envía una solicitud de hora.
4. El proceso de servidor de hora responde con la hora actual.
5. Después se libera la conexión de transporte.

Observe que en el *host* 2 podría haber otros servidores enlazados a otros TSAPs en espera de conexiones entrantes sobre el mismo NSAP.

El panorama que hemos bosquejado está muy bien, excepto que hemos ocultado un pequeño problema: ¿cómo sabe el proceso de usuario del *host* 1 que el servidor de hora del día está conectado al TSAP 1522? Una posibilidad es que el servidor de hora del día se ha estado conectando al TSAP 1522 durante años, y gradualmente todos los usuarios de la red han aprendido esto. En este modelo, los servicios tienen direcciones TSAP estables que se listan en archivos en lugares bien conocidos, como el archivo */etc/services* de los sistemas UNIX, que lista cuáles servidores están enlazados de manera permanente a cuáles puertos.

Aunque las direcciones TSAP estables podrían funcionar bien con una cantidad pequeña de servicios clave que nunca cambian (por ejemplo, el servidor Web), en general, los procesos de usuario frecuentemente desean comunicarse con otros procesos de usuario que sólo existen durante un tiempo corto y no tienen una dirección TSAP conocida por adelantado. Es más, si puede haber muchos procesos de servidor, la mayoría de los cuales se usan pocas veces, es un desperdicio tenerlos activados a todos, escuchando en una dirección TSAP estable todo el día. En pocas palabras, se requiere un mejor esquema.

En la figura 6-9 se muestra un esquema simplificado. Se conoce como **protocolo inicial de conexión.** En lugar de que cada servidor concebible escuche en un TSAP bien conocido, cada máquina que desea ofrecer servicio a usuarios remotos tiene un **servidor de procesos** especial que actúa como *proxy* de los servidores de menor uso. Este servidor escucha en un grupo de puertos al mismo tiempo, esperando una solicitud de conexión. Los usuarios potenciales de un servicio comienzan por emitir una solicitud CONNECT, especificando la dirección TSAP del servicio que desean. Si no hay ningún servidor esperándolos, consiguen una conexión al servidor de procesos, como se muestra en la figura 6-9(a).

Tras obtener la solicitud entrante, el servidor de procesos genera el servidor solicitado, permitiéndole heredar la conexión con el usuario existente. El nuevo servidor entonces hace el trabajo requerido, mientras que el servidor de procesos retorna a escuchar solicitudes nuevas, como se muestra en la figura 6-9(b).

Aunque el protocolo de conexión inicial funciona bien para aquellos servidores que pueden crearse conforme son necesarios, hay muchas situaciones en las que los servicios existen independientemente del servidor de procesos. Por ejemplo, un servidor de archivos necesita operar en un *hardware* especial (una máquina con un disco) y no puede simplemente crearse sobre la marcha cuando alguien quiere comunicarse con él.

Para manejar esta situación, se usa con frecuencia un esquema alterno. En este modelo, existe un proceso especial llamado **servidor de nombres,** o a veces **servidor de directorio.** Para encontrar la dirección TSAP correspondiente a un nombre de servicio dado, como “hora del día”, el usuario establece una conexión con el servidor de nombres (que escucha en un TSAP bien

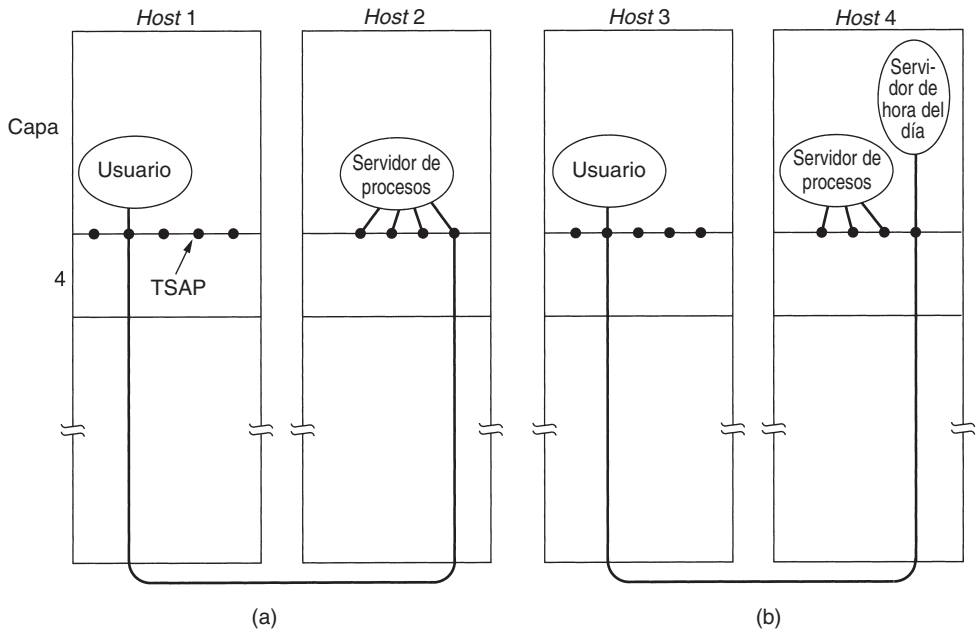


Figura 6-9. Manera en que un proceso de usuario del *host* 1 establece una conexión con un servidor de hora del día del *host* 2.

conocido). Entonces el usuario envía un mensaje especificando el nombre del servicio, y el servidor de nombres devuelve la dirección TSAP. Luego el usuario libera la conexión con el servidor de nombres y establece una nueva con el servicio deseado.

En este modelo, al crearse un servicio nuevo, debe registrarse en el servidor de nombres, dando tanto su nombre de servicio (generalmente, una cadena ASCII) como la dirección de su TSAP. El servidor de nombres registra esta información en su base de datos interna por lo que, cuando llegan solicitudes posteriores, sabe las respuestas.

La función del servidor de nombres es análoga al operador de asistencia de directorio del sistema telefónico: proporciona el número correspondiente a un nombre determinado. Al igual que en el sistema telefónico, es esencial que la dirección bien conocida del TSAP usado por el servidor de nombres (o el servidor de procesos del protocolo de conexión inicial) en realidad sea bien conocida. Si usted no conoce el número del operador de información, no puede llamar al operador de información para averiguarlo. Si usted cree que el número que se marca para obtener información es obvio, inténtelo alguna vez en otro país.

6.2.2 Establecimiento de una conexión

El establecimiento de una conexión suena fácil, pero en realidad es sorprendentemente complicado. A primera vista, parecería suficiente con que una entidad de transporte enviara una TPDU

CONNECTION REQUEST al destino y esperar una respuesta CONNECTION ACCEPTED. El problema ocurre cuando la red puede perder, almacenar o duplicar paquetes. Este comportamiento causa complicaciones serias.

Imagine una subred que está tan congestionada que las confirmaciones de recepción casi nunca regresan a tiempo, y cada paquete expira y se retransmite dos o tres veces. Suponga que la subred usa datagramas internamente, y que cada paquete sigue una ruta diferente. Algunos de los paquetes podrían atorarse en un congestionamiento de tráfico dentro de la subred y tardar mucho tiempo en llegar, es decir, se almacenarían en la subred y reaparecerían mucho después.

La peor pesadilla posible es la que sigue. Un usuario establece una conexión con un banco, envía mensajes indicando al banco que transfiera una gran cantidad de dinero a la cuenta de una persona no del todo confiable y a continuación libera la conexión. Por mala fortuna, cada paquete de la transacción se duplica y almacena en la subred. Tras liberar la conexión, todos los paquetes salen de la subred y llegan al destino en orden, solicitando al banco que establezca una conexión nueva, transfiera el dinero (nuevamente) y libere la conexión. El banco no tiene manera de saber que son duplicados; debe suponer que ésta es una segunda transacción independiente, y transfiere nuevamente el dinero. Durante el resto de esta sección estudiaremos el problema de los duplicados retardados, haciendo hincapié en los algoritmos para establecer conexiones de una manera confiable, de modo que pesadillas como la anterior no puedan ocurrir.

El meollo del problema es la existencia de duplicados retrasados. Esto puede atacarse de varias maneras, ninguna de las cuales es muy satisfactoria. Una es usar direcciones de transporte desechables. En este enfoque, cada vez que se requiere una dirección de transporte, se genera una nueva. Al liberarse la conexión, se descarta la dirección y no se vuelve a utilizar. Esta estrategia imposibilita el modelo de servidor de procesos de la figura 6-9.

Otra posibilidad es dar a cada conexión un identificador de conexión (es decir, un número de secuencia que se incrementa con cada conexión establecida), seleccionado por la parte iniciadora, y ponerlo en cada TPDU, incluida la que solicita la conexión. Tras la liberación de una conexión, cada entidad de transporte podría actualizar una tabla que liste conexiones obsoletas como pares (entidad de transporte igual, identificador de conexión). Cada vez que entrara una solicitud de conexión, podría cotejarse con la tabla para saber si pertenece a una conexión previamente liberada.

Por desgracia, este esquema tiene una falla básica: requiere que cada entidad de transporte mantenga una cierta cantidad de información histórica durante un tiempo indefinido. Si se cae una máquina y pierde su memoria, ya no sabrá qué identificadores de conexión usó.

Más bien, necesitamos un enfoque diferente. En lugar de permitir que los paquetes vivan eternamente en la subred, debemos diseñar un mecanismo para eliminar a los paquetes viejos que aún andan vagando por ahí. Si podemos asegurar que ningún paquete viva más allá de cierto tiempo conocido, el problema se vuelve algo más manejable.

El tiempo de vida de un paquete puede restringirse a un máximo conocido usando una de las siguientes técnicas:

1. Un diseño de subred restringido.
2. Colocar un contador de saltos en cada paquete.
3. Marcar el tiempo en cada paquete.

El primer método incluye cualquier método que evite que los paquetes hagan ciclos, combinado con una manera de limitar el retardo por congestionamientos a través de la trayectoria más larga posible (ahora conocida). El segundo método consiste en inicializar el conteo de saltos con un valor apropiado y decrementarlo cada vez que se reenvíe el paquete. El protocolo de red simplemente descarta cualquier paquete cuyo contador de saltos llega a cero. El tercer método requiere que cada paquete lleve la hora en la que fue creado, y que los enrutadores se pongan de acuerdo en descartar cualquier paquete que haya rebasado cierto tiempo predeterminado. Este último método requiere que los relojes de los enrutadores estén sincronizados, lo que no es una tarea fácil a menos que se logre la sincronización externamente a la red, por ejemplo, utilizando GPS o alguna estación de radio que difunda la hora exacta periódicamente.

En la práctica, necesitaremos garantizar no sólo que el paquete está eliminado, sino que también lo están todas sus confirmaciones de recepción, por lo que ahora introduciremos *T*, que es un múltiplo pequeño del tiempo de vida de paquete máximo verdadero. El múltiplo depende del protocolo, y simplemente tiene el efecto de hacer más grande a *T*. Si esperamos un tiempo *T* tras el envío de un paquete, podemos estar seguros de que todos los rastros suyos ya han desaparecido, y que ni él ni sus confirmaciones de recepción aparecerán repentinamente de la nada para complicar el asunto.

Al limitar los tiempos de vida de los paquetes, es posible proponer una manera a prueba de errores de establecer conexiones seguras. El método descrito a continuación se debe a Tomlinson (1975); resuelve el problema pero presenta algunas peculiaridades propias. El método fue refinado por Sunshine y Dalal (1978). En la práctica se usan ampliamente variantes suyas, incluso en TCP.

Para resolver el problema de una máquina que pierde toda la memoria acerca de su estado tras una caída, Tomlinson propuso equipar cada *host* con un reloj de hora del día. Los relojes de los diferentes *hosts* no necesitan estar sincronizados. Se supone que cada reloj tiene la forma de un contador binario que se incrementa a sí mismo a intervalos uniformes. Además, la cantidad de bits del contador debe ser igual o mayor que la cantidad de bits en los números de secuencia. Por último, y lo más importante, se supone que el reloj continúa operando aun ante la caída del *host*.

La idea básica es asegurar que nunca estén pendientes al mismo tiempo dos TPDUs de número idéntico. Cuando se establece una conexión, los *k* bits de orden menor del reloj se usan como número inicial de secuencia (también *k* bits). Por tanto, y a diferencia de los protocolos del capítulo 3, cada conexión comienza a numerar sus TPDUs con un número de secuencia inicial diferente. El espacio de secuencia también debe ser lo bastante grande para que, al regresar al principio de los números de secuencia, las TPDUs viejas con el mismo número de secuencia hayan desaparecido hace mucho tiempo. En la figura 6-10 se muestra esta relación lineal entre tiempo y números secuenciales iniciales.

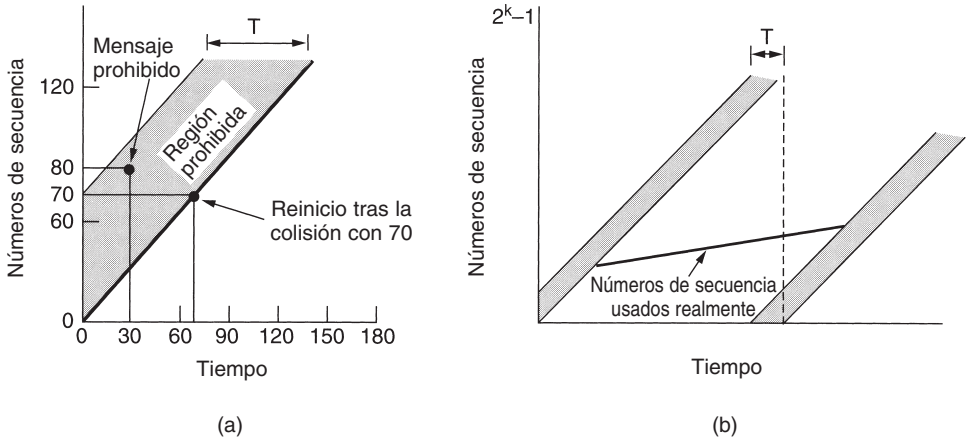


Figura 6-10. (a) Las TPDUs no pueden entrar en la zona prohibida. (b) El problema de la resincronización.

Una vez que ambas entidades de transporte han acordado el número de secuencia inicial, puede usarse cualquier protocolo de ventana corrediza para el control de flujo de datos. En realidad, la curva inicial de números de secuencia (indicada por la línea gruesa) no es realmente lineal, sino una escalera, ya que el reloj avanza en pasos discretos. Por sencillez, ignoraremos este detalle.

Cuando un *host* se cae ocurre un problema. Al reactivarse, sus entidades de transporte no saben dónde estaban en el espacio de secuencia. Una solución es requerir que las entidades de transporte estén inactivas durante T segundos tras una recuperación para permitir que todas las TPDUs viejas expiren. Sin embargo, en una interred compleja, T puede ser bastante grande, por lo que no es atractiva esta estrategia.

Para evitar requerir T seg de tiempo muerto tras una caída, es necesario introducir una nueva restricción en el uso de números de secuencia. Podemos ver claramente la necesidad de esta restricción mediante un ejemplo. Sea T, el tiempo máximo de vida de un paquete, 60 seg, y que el pulso del reloj sea de uno por segundo. Como lo indica la línea gruesa en la figura 6-10(a), el número de secuencia inicial de una conexión abierta en el momento x será x. Imagine que, en $t = 30$ seg, una TPDU de datos ordinaria enviada a través de la conexión 5 (previamente abierta), recibe el número de secuencia 80. Llamemos X a esta TPDU. De inmediato tras el envío de la TPDU X, el *host* se cae y reinicia pronto. En $t = 60$, el *host* comienza a reabrir las conexiones 0 a 4. En $t = 70$, reabre la conexión 5, usando el número de secuencia inicial 70, como se requiere. Durante los siguientes 15 segundos envía las TPDUs de datos 70 a 80. Por tanto, en $t = 85$ se ha inyectado una TPDU nueva con número de secuencia 80 y conexión 5 en la subred. Por desgracia, la TPDU X aún existe. Si ésta llegara al receptor antes de la nueva TPDU 80, la TPDU X sería aceptada y la TPDU 80 correcta sería rechazada como duplicado.

Para evitar tales problemas, debemos evitar la asignación de números de secuencia nuevos (es decir, asignados a TPDUs nuevas) durante un tiempo T antes de su uso potencial como números iniciales de secuencia. Las combinaciones ilegales de tiempo y número de secuencia se muestran

como la **región prohibida** en la figura 6-10(a). Antes de enviar cualquier TPDU por alguna conexión, la entidad de transporte debe leer el reloj y comprobar que no está en la región prohibida.

El protocolo puede meterse en problemas de dos maneras. Si un *host* envía demasiados datos con demasiada rapidez a través de una conexión recién abierta, la curva de número de secuencia real contra tiempo puede subir con mayor rapidez que la curva de número de secuencia inicial contra tiempo. Esto significa que la tasa de datos máxima en cualquier conexión es de una TPDU por pulso de reloj, y también significa que la entidad de transporte debe esperar hasta que el reloj pulse antes de abrir una nueva conexión tras un reinicio por cada caída, no sea que se use dos veces el mismo número de secuencia. Ambos puntos son argumentos a favor de un pulso de reloj corto (unos cuantos milisegundos).

Desgraciadamente, el ingreso en la región prohibida desde abajo al enviar con demasiada rapidez no es la única manera de meterse en problemas. Por la figura 6-10(b) debe quedar claro que con cualquier tasa de datos menor que la tasa del reloj, la curva de números de secuencia reales usados contra tiempo tarde o temprano entrará en la región prohibida por la izquierda. Cuanto mayor sea la pendiente de la curva de números de secuencia reales, mayor será el retardo de este evento. Como ya indicamos, justo antes de enviar cada TPDU, la entidad de transporte debe comprobar que no esté a punto de entrar en la región prohibida; de ser así, debe retardar la TPDU durante T seg o resincronizar los números de secuencia.

El método basado en reloj resuelve el problema del duplicado retrasado de las TPDUs de datos, pero para que este método resulte de utilidad, debe establecerse primero una conexión. Dado que las TPDUs de control también pueden retrasarse, hay el problema potencial de lograr que ambos lados acuerden el número de secuencia inicial. Supongamos, por ejemplo, que se establecen conexiones haciendo que el *host* 1 envíe una TPDU CONNECTION REQUEST con el número de secuencia inicial y el número de puerto de destino a un igual remoto, el *host* 2. El receptor, el *host* 2, confirma entonces la recepción de esta solicitud enviando de regreso una TPDU CONNECTION ACCEPTED. Si la TPDU CONNECTION REQUEST se pierde, pero aparece con retardo una CONNECTION REQUEST duplicada en el *host* 2, se establecerá incorrectamente la conexión.

Para resolver este problema, Tomlinson (1975) desarrolló el **acuerdo de tres vías** (*three-way handshake*). Este protocolo de establecimiento no requiere que ambos lados comiencen a transmitir con el mismo número de secuencia, por lo que puede usarse con otros métodos de sincronización distintos del método de reloj global. El procedimiento normal de establecimiento al iniciar el *host* 1 se muestra en la figura 6-11(a). El *host* 1 escoge un número de secuencia, x, y envía al *host* 2 una TPDU CONNECTION REQUEST que lo contiene. El *host* 2 responde con una TPDU CONNECTION ACCEPTED confirmando la recepción de x y anunciando su propio número de secuencia inicial, y. Por último, el *host* 1 confirma la recepción de la selección de un número de secuencia inicial del *host* 2 en la primera TPDU de datos que envía.

Ahora veamos la manera en que funciona el acuerdo de tres vías en presencia de TPDUs de control duplicadas con retraso. En la figura 6-11(b), la primera TPDU es una CONNECTION REQUEST duplicada con retraso de una conexión vieja. Esta TPDU llega al *host* 2 sin el conocimiento del *host* 1. El *host* 2 reacciona a esta TPDU enviando al *host* 1 una TPDU ACK, solicitando de hecho la comprobación de que el *host* 1 en verdad trató de establecer una nueva conexión. Al rechazar el *host* 1 el intento de establecimiento de conexión del *host* 2, éste se da cuenta de que fue

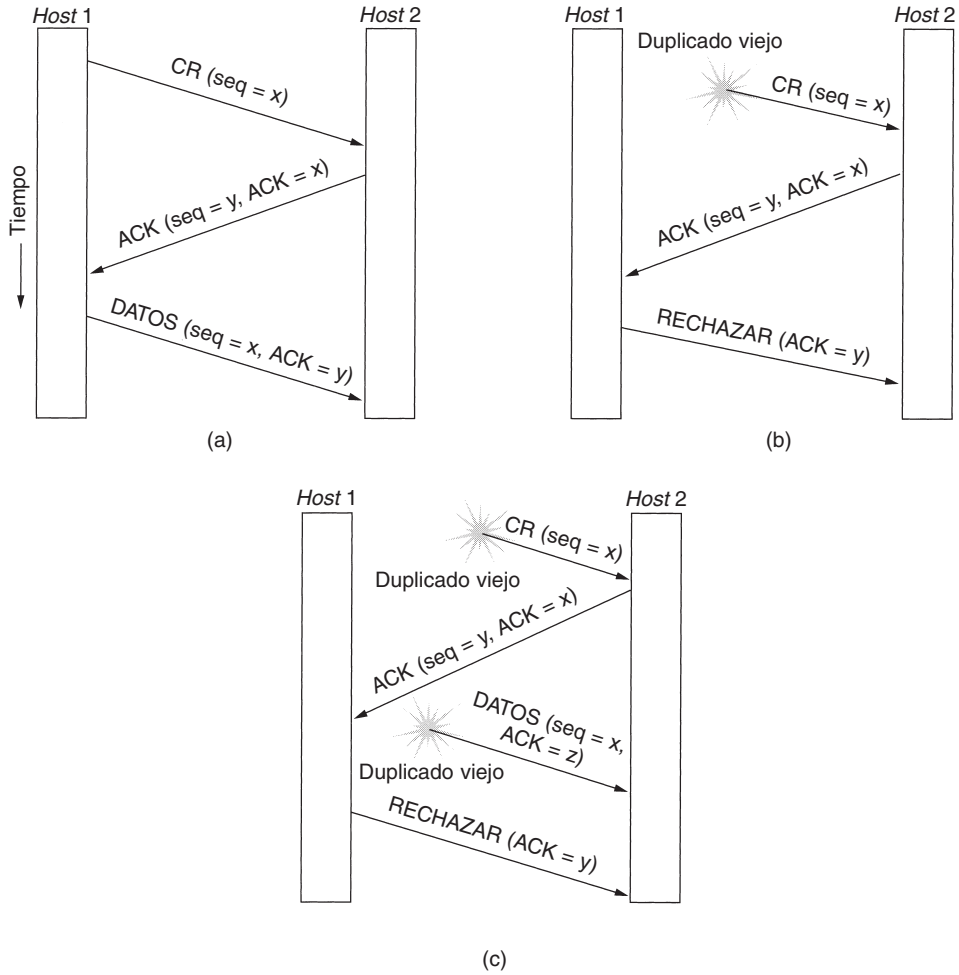


Figura 6-11. Tres escenarios para establecer una conexión usando un acuerdo de tres vías. CR significa CONNECTION REQUEST. (a) Operación normal. (b) CONNECTION REQUEST duplicada vieja que aparece de la nada. (c) CONNECTION REQUEST duplicada y ACK duplicada.

engañado por un duplicado con retardo y abandona la conexión. De esta manera, un duplicado con retardo no causa daño.

El peor caso ocurre cuando en la subred deambulan tanto una CONNECTION REQUEST retardada como una ACK. Este caso se muestra en la figura 6-11(c). Como en el ejemplo previo, el *host* 2 recibe una CONNECTION REQUEST retrasada y la contesta. En este momento es crucial notar que el *host* 2 ha propuesto usar *y* como número de secuencia inicial para el tráfico del *host* 2 al *host* 1, sabiendo bien que no existen todavía TPDUs que contengan el número de secuencia *y* ni confirmaciones de recepción de *y*. Cuando llega la segunda TPDU retrasada al *host* 2, el hecho

de que se confirmó la recepción de *z* en lugar de *y* indica al *host* 2 que éste también es un duplicado viejo. Lo importante que se debe tomar en cuenta aquí es que no haya combinación de viejas TPDUs que puedan causar la falla del protocolo y permitan el establecimiento de una conexión accidentalmente cuando nadie la quiere.

6.2.3 Liberación de una conexión

La liberación de una conexión es más fácil que su establecimiento. No obstante, hay más escollos de los que uno podría imaginar. Como mencionamos antes, hay dos estilos de terminación de una conexión: liberación asimétrica y liberación simétrica. La liberación asimétrica es la manera en que funciona el sistema telefónico: cuando una parte cuelga, se interrumpe la conexión. La liberación simétrica trata la conexión como dos conexiones unidireccionales distintas, y requiere que cada una se libere por separado.

La liberación asimétrica es abrupta y puede resultar en la pérdida de datos. Considere el escenario de la figura 6-12. Tras establecerse la conexión, el *host* 1 envía una TPDU que llega adecuadamente al *host* 2. Entonces el *host* 1 envía otra TPDU. Desgraciadamente, el *host* 2 emite una DISCONNECT antes de llegar la segunda TPDU. El resultado es que se libera la conexión y se pierden datos.

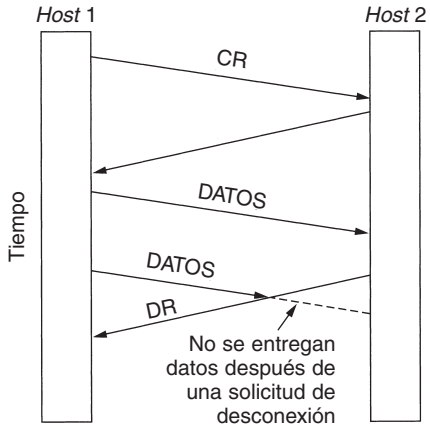


Figura 6-12. Desconexión abrupta con pérdida de datos.

Es obvio que se requiere un protocolo de liberación más refinado para evitar la pérdida de datos. Una posibilidad es usar la liberación simétrica, en la que cada dirección se libera independientemente de la otra. Aquí, un *host* puede continuar recibiendo datos aun tras haber enviado una TPDU DISCONNECT.

La liberación simétrica es ideal cuando cada proceso tiene una cantidad fija de datos por enviar y sabe con certidumbre cuándo los ha enviado. En otras situaciones, la determinación de si

se ha efectuado o no todo el trabajo y si debe terminarse o no la conexión no es tan obvia. Podríamos pensar en un protocolo en el que el *host* 1 diga: "Ya terminé. ¿Terminaste también?" Si el *host* 2 responde: "Ya terminé también. Adiós", la conexión puede liberarse con seguridad.

Por desgracia, este protocolo no siempre funciona. Hay un problema famoso que tiene que ver con ese asunto; se llama **problema de los dos ejércitos.** Imagine que un ejército blanco está acampado en un valle, como se muestra en la figura 6-13. En los dos cerros que rodean al valle hay ejércitos azules. El ejército blanco es más grande que cualquiera de los dos ejércitos azules por separado, pero juntos éstos son más grandes que el ejército blanco. Si cualquiera de los dos ejércitos azules ataca por su cuenta, será derrotado, pero si los dos atacan simultáneamente obtendrán la victoria.

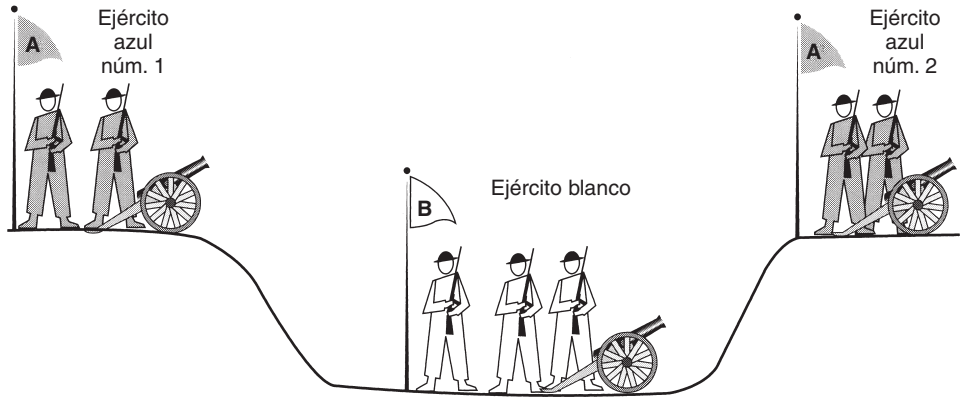


Figura 6-13. Problema de los dos ejércitos.

Los ejércitos azules quieren sincronizar sus ataques. Sin embargo, su único medio de comunicación es el envío de mensajeros a pie a través del valle, donde podrían ser capturados, perdiéndose el mensaje (es decir, tienen que usar un canal de comunicación no confiable). La pregunta es: ¿Existe un protocolo que permita que los ejércitos azules ganen?

Supongamos que el comandante del ejército azul núm. 1 envía un mensaje que dice: "Propongo que ataquemos al amanecer del 29 marzo. ¿Qué les parece?" Ahora supongamos que llega el mensaje y que el comandante del ejército azul núm. 2 está de acuerdo, y que su respuesta llega con seguridad al ejército azul núm. 1. ¿Ocurrirá el ataque? Probablemente no, porque el comandante núm. 2 no sabe si su respuesta llegó. Si no llegó, el ejército azul núm. 1 no atacará, y sería tonto de su parte emprender el ataque.

Mejoremos ahora el protocolo haciéndolo un acuerdo de tres vías. El iniciador de la propuesta original debe confirmar la recepción de la respuesta. Suponiendo que no se pierden mensajes, el ejército azul núm. 2 recibirá la confirmación de recepción, pero ahora el que dudará será el comandante del ejército azul núm. 1. A fin de cuentas, no sabe si ha llegado su confirmación de

recepción y, si no llegó, sabe que el ejército núm. 2 no atacará. Podríamos probar ahora un protocolo de acuerdo de cuatro vías, pero tampoco ayudaría.

De hecho, puede demostrarse que no existe un protocolo que funcione. Supongamos que existiera algún protocolo. O el último mensaje del protocolo es esencial, o no lo es. Si no lo es, hay que eliminarlo (así como los demás mensajes no esenciales) hasta que quede un protocolo en el que todos los mensajes sean esenciales. ¿Qué ocurre si el mensaje final no pasa? Acabamos de decir que es esencial, por lo que, si se pierde, el ataque no ocurrirá. Dado que el emisor del mensaje final nunca puede estar seguro de su llegada, no se arriesgará a atacar. Peor aún, el otro ejército azul sabe esto, por lo que no atacará tampoco.

Para ver la aplicación del problema de los dos ejércitos a la liberación de conexiones, simplemente sustituya "atacar" por "desconectar". Si ninguna de las partes está preparada para desconectarse hasta estar convencida de que la otra está preparada para desconectarse también, nunca ocurrirá la desconexión.

En la práctica, generalmente estamos más dispuestos a correr riesgos al liberar conexiones que al atacar ejércitos blancos, por lo que la situación no es del todo desesperada. En la figura 6-14 se ilustran cuatro escenarios de liberación usando un acuerdo de tres vías. Aunque este protocolo no es infalible, generalmente es adecuado.

En la figura 6-14(a) vemos el caso normal en el que uno de los usuarios envía una TPDU DR (DISCONNECTION REQUEST, solicitud de desconexión) a fin de iniciar la liberación de una conexión. Al llegar, el receptor devuelve también una TPDU DR e inicia un temporizador, para el caso de que se pierda su DR. Al llegar esta DR, el emisor original envía de regreso una TPDU ACK y libera la conexión. Finalmente, cuando la TPDU ACK llega, el receptor también libera la conexión. La liberación de una conexión significa que la entidad de transporte remueve la información sobre la conexión de su tabla de conexiones abiertas y avisa de alguna manera al dueño de la conexión (el usuario de transporte). Esta acción es diferente a aquélla en la que el usuario de transporte emite una primitiva DISCONNECT.

Si se pierde la última TPDU ACK, como se muestra en la figura 6-14(b), el temporizador salva la situación. Al expirar el temporizador, la conexión se libera de todos modos.

Ahora consideremos el caso de la pérdida de la segunda DR. El usuario que inicia la desconexión no recibirá la respuesta esperada, su temporizador expirará y todo comenzará de nuevo. En la figura 6-14(c) vemos la manera en que funciona esto, suponiendo que la segunda vez no se pierden TPDUs y que todas se entregan correctamente y a tiempo.

Nuestro último escenario, la figura 6-14(d), es el misma que en la figura 6-14(c), excepto que ahora suponemos que todos los intentos repetidos de retransmitir la DR también fallan debido a la pérdida de TPDUs. Tras *N* reintentos, el emisor simplemente se da por vencido y libera la conexión. Mientras tanto, expira el temporizador del receptor y también se sale.

Aunque este protocolo generalmente es suficiente, en teoría puede fallar si se pierden la DR inicial y *N* retransmisiones. El emisor se dará por vencido y liberará la conexión, pero el otro lado no sabrá nada sobre los intentos de desconexión y seguirá plenamente activo. Esta situación origina una conexión semiabierta.

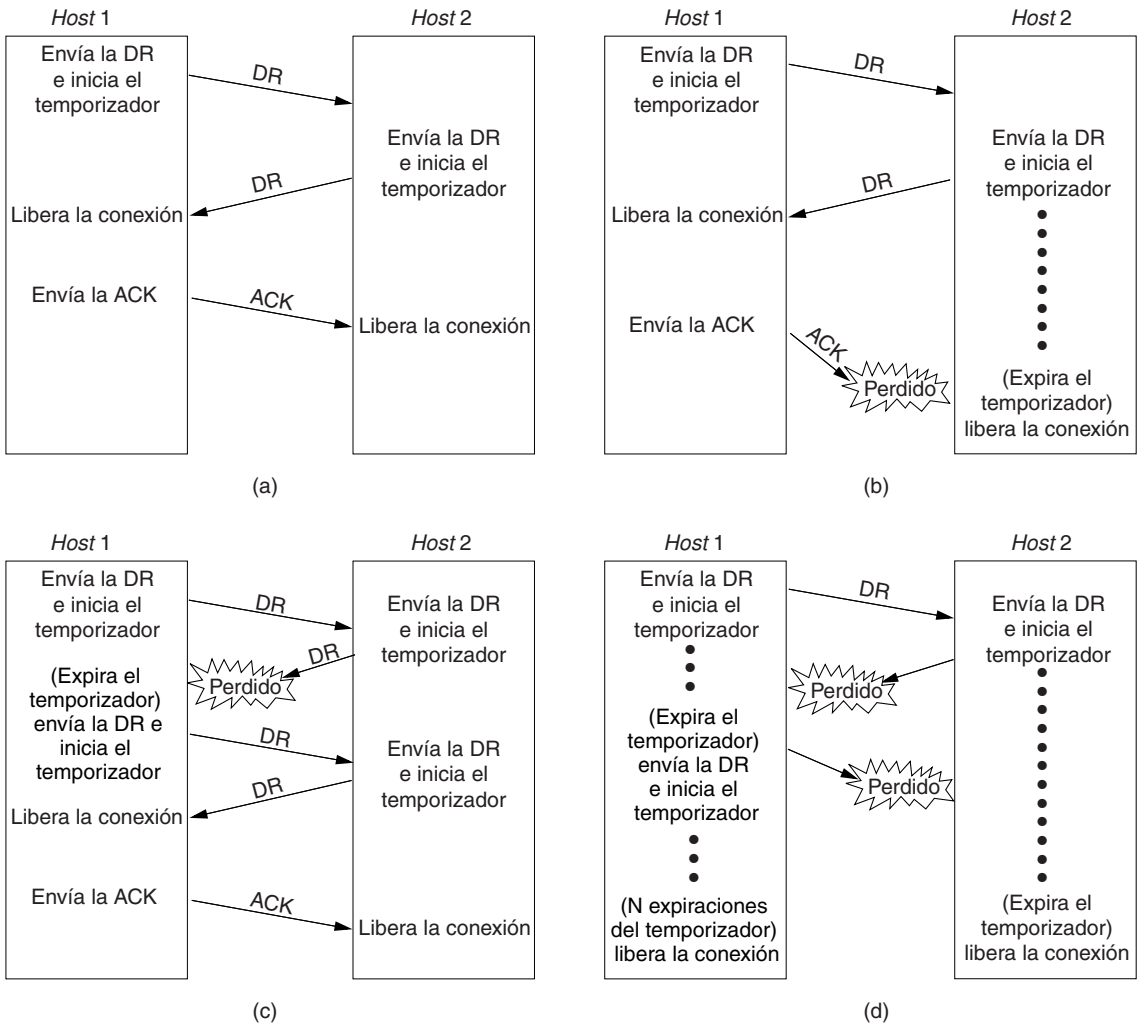


Figura 6-14. Cuatro escenarios de un protocolo para liberar una conexión. (a) Caso normal del acuerdo de tres vías. (b) Pérdida de la última ACK. (c) Respuesta perdida. (d) Respuesta perdida y pérdida de las DRs subsecuentes.

Pudimos haber evitado este problema no permitiendo que el emisor se diera por vencido tras *N* reintentos, sino obligándolo a seguir insistiendo hasta recibir una respuesta. Sin embargo, si se permite que expire el temporizador en el otro lado, entonces el emisor continuará eternamente, pues nunca aparecerá una respuesta. Si no permitimos que expire el temporizador en el lado receptor, entonces el protocolo se atora en la figura 6-14(b).

Otra manera de eliminar las conexiones semiabiertas es tener una regla que diga que, si no ha llegado ninguna TPDU durante cierta cantidad de segundos, se libera automáticamente la conexión. De esta manera, si un lado llega a desconectarse, el otro lado detectará la falta de actividad

y también se desconectará. Por supuesto que si se pone en práctica esta regla, es necesario que cada entidad de transporte tenga un temporizador que se detenga y se reinicie con cada envío de una TPDU. Si expira este temporizador, se transmite una TPDU ficticia, simplemente para evitar que el otro lado se desconecte. Por otra parte, si se usa la regla de desconexión automática y se pierden demasiadas TPDU ficticias una tras otra en una conexión que de otro modo estaría en reposo, primero un lado y luego el otro se desconectarán automáticamente.

No insistiremos más en este punto, pero ya debe quedar claro que la liberación de una conexión no es ni remotamente tan sencilla como parece inicialmente.

6.2.4 Control de flujo y almacenamiento en búfer

Habiendo examinado el establecimiento y la liberación de conexiones con algún detalle, veamos ahora la manera en que se manejan las conexiones mientras están en uso. Ya conocemos uno de los aspectos clave: el control de flujo. En algunos sentidos el problema del control de flujo en la capa de transporte es igual que en la capa de enlace de datos, pero en otros es diferente. La similitud básica es que en ambas capas se requiere una ventana corrediza u otro esquema en cada conexión para evitar que un emisor rápido desborde a un receptor lento. La diferencia principal es que un enrutador por lo regular tiene relativamente pocas líneas, y que un *host* puede tener numerosas conexiones. Esta diferencia hace que sea impráctico implementar en la capa de transporte la estrategia de almacenamiento en búfer de la capa de enlace de datos.

En los protocolos de enlace de datos del capítulo 3, las tramas se almacenaban en búferes tanto en el enrutador emisor como en el receptor. Por ejemplo, en el protocolo 6 se requiere que tanto el emisor como el receptor dediquen *MAX_SEQ* + 1 búferes a cada línea, la mitad para entradas y la mitad para salidas. En un *host* con un máximo de, digamos, 64 conexiones y un número de secuencia de 4 bits, este protocolo requerirá 1024 búferes.

En la capa de enlace de datos, el lado emisor debe almacenar en búfer las tramas de salida porque cabe la posibilidad de que tengan que retransmitirse. Si la subred provee un servicio de datagramas, la entidad de transporte emisora también debe manejar búferes, por la misma razón. Si el receptor sabe que el emisor almacena en búfer todas las TPDUs hasta que se confirma su recepción, el receptor podría o no dedicar búferes específicos a conexiones específicas, según considere necesario. Por ejemplo, el receptor podría mantener un solo grupo de búferes compartido por todas las conexiones. Cuando entra una TPDU, se hace un intento por adquirir dinámicamente un búfer nuevo. Si hay uno disponible, se acepta la TPDU; de otro modo, se descarta. Dado que el emisor está preparado para retransmitir las TPDUs perdidas por la subred, no hay nada de malo en hacer que el receptor se deshaga de las TPDUs, aunque se desperdicien algunos recursos. El emisor simplemente sigue intentando hasta que recibe una confirmación de recepción.

En resumen, si el servicio de red no es confiable, el emisor debe almacenar en búfer todas las TPDUs enviadas, igual que en la capa de enlace de datos. Sin embargo, con un servicio de red confiable son posibles otros arreglos. En particular, si el emisor sabe que el receptor siempre tiene espacio de búfer, no necesita retener copias de las TPDUs que envía. Sin embargo, si el receptor no puede garantizar que se aceptará cada TPDU que llegue, el emisor tendrá que usar búferes de

todas maneras. En el último caso, el emisor no puede confiar en la confirmación de recepción de la capa de red porque esto sólo significa que ha llegado la TPDU, no que ha sido aceptada. Regresaremos después a este importante punto.

Aun si el receptor está de acuerdo en utilizar búferes, todavía queda la cuestión del tamaño de éstos. Si la mayoría de las TPDUs tiene aproximadamente el mismo tamaño, es natural organizar los búferes como un grupo de búferes de tamaño idéntico, con una TPDU por búfer, como en la figura 6-15(a). Sin embargo, si hay una variación grande en el tamaño de las TPDUs, de unos cuantos caracteres ingresados en una terminal hasta miles de caracteres provenientes de transferencias de archivos, el grupo de búferes de tamaño fijo presenta problemas. Si el tamaño de búfer se escoge igual a la TPDU más grande, se desperdiciará espacio cada vez que llegue una TPDU corta. Si el tamaño se escoge menor que el tamaño máximo de TPDU, se requerirán varios búferes para las TPDUs grandes, con la complejidad inherente.

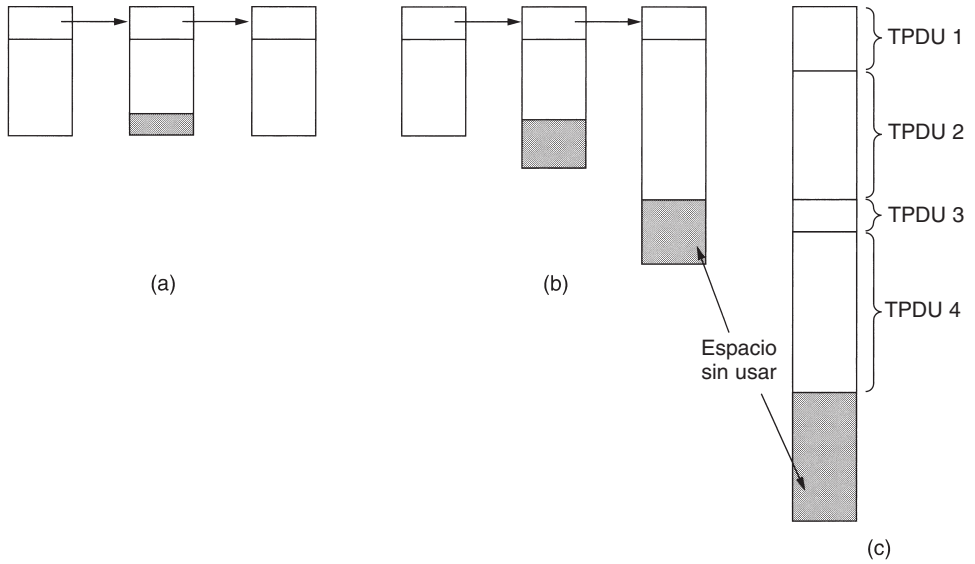


Figura 6-15. (a) Búferes encadenados de tamaño fijo. (b) Búferes encadenados de tamaño variable. (c) Un gran búfer circular por conexión.

Otra forma de enfrentar el problema del tamaño de los búferes es el uso de búferes de tamaño variable, como en la figura 6-15(b). La ventaja aquí es un mejor uso de la memoria, al costo de una administración de búferes más complicada. Una tercera posibilidad es dedicar un solo búfer circular grande por conexión, como en la figura 6-15(c). Este sistema también hace buen uso de la memoria cuando todas las conexiones tienen una carga alta, pero es deficiente si algunas conexiones cuentan con poca carga.

El equilibrio óptimo entre el almacenamiento en búfer en el origen y en el destino depende del tipo de tráfico transportado por la conexión. Para un tráfico de bajo ancho de banda con ráfagas, como el producido por una terminal interactiva, es mejor no dedicarle búferes, sino adquirirlos de

manera dinámica en ambos extremos. Dado que el emisor no puede estar seguro de que el receptor será capaz de adquirir un búfer, el emisor debe retener una copia de la TPDU hasta recibir su confirmación de recepción. Por otra parte, para la transferencia de archivos y otro tráfico de alto ancho de banda, es mejor si el receptor dedica una ventana completa de búferes, para permitir el flujo de datos a máxima velocidad. Por tanto, para un tráfico en ráfagas de bajo ancho de banda, es mejor mantener búferes en el emisor; para tráfico continuo de alto ancho de banda, es mejor hacerlo en el receptor.

A medida que se abren y cierran conexiones, y a medida que cambia el patrón del tráfico, el emisor y el receptor necesitan ajustar dinámicamente sus asignaciones de búferes. En consecuencia, el protocolo de transporte debe permitir que un *host* emisor solicite espacio en búfer en el otro extremo. Los búferes podrían repartirse por conexión o, en conjunto, para todas las conexiones en operación entre los dos *hosts*. Como alternativa, el receptor, sabiendo su capacidad de manejo de búferes (pero sin saber el tráfico generado) podría indicar al emisor "te he reservado *X* búferes". Si aumentara la cantidad de conexiones abiertas, podría ser necesario reducir una asignación, por lo que el protocolo debe contemplar esta posibilidad.

Una manera razonablemente general de manejar la asignación dinámica de búferes es desacoplarlos de las confirmaciones de recepción, en contraste con los protocolos de ventana corrediza del capítulo 3. La administración dinámica de búferes implica, de hecho, una ventana de tamaño variable. Inicialmente, el emisor solicita una cierta cantidad de búferes, con base en sus necesidades percibidas. El receptor entonces otorga tantos búferes como puede. Cada vez que el emisor envía una TPDU, debe disminuir su asignación, deteniéndose por completo al llegar la asignación a cero. El receptor entonces incorpora tanto las confirmaciones de recepción como las asignaciones de búfer al tráfico de regreso.

En la figura 6-16 se muestra un ejemplo de la manera en que podría trabajar la administración dinámica de ventanas en una subred de datagramas con números de secuencia de 4 bits. Supongamos que la información de asignación de búferes viaja en TPDUs distintas, como se muestra, y no se incorpora en el tráfico de regreso. Inicialmente, *A* quiere ocho búferes, pero se le otorgan solamente cuatro. Entonces envía tres TPDUs, de las cuales se pierde la tercera. La TPDU 6 confirma la recepción de todas las TPDUs hasta el número de secuencia 1, inclusive, permitiendo por tanto que *A* libere esos búferes, y además informa a *A* que tiene permiso de enviar tres TPDUs más comenzando después de 1 (es decir, las TPDUs 2, 3 y 4). *A* sabe que ya ha enviado el número 2, por lo que piensa que debe enviar las TPDUs 3 y 4, lo que procede a hacer. En este punto se bloquea y debe esperar una nueva asignación de búfer. Por otro lado, las retransmisiones inducidas por expiraciones del temporizador (línea 9) sí pueden ocurrir durante el bloqueo, pues usan búferes ya asignados. En la línea 10, *B* confirma la recepción de todas las TPDUs hasta la 4, inclusive, pero se niega a permitir que *A* continúe. Tal situación es imposible con los protocolos de ventana fija del capítulo 3. La siguiente TPDU de *B* a *A* asigna otro búfer y permite a *A* continuar.

Pueden surgir problemas potenciales con los esquemas de asignación de búferes de este tipo en las redes de datagramas si hay pérdidas de TPDU. Observe la línea 16. *B* ha asignado ahora más búferes a *A*, pero la TPDU de asignación se perdió. Dado que las TPDUs de control no están en secuencia, *A* se encuentra estancado. Para evitar esta situación, cada *host* debe enviar periódica-

	A	Mensaje	B	Comentarios
1	→	<solicito 8 búferes>	→	A quiere 8 búferes
2	←	<ack = 15, buf = 4>	←	B sólo otorga los mensajes 0 a 3
3	→	<seq = 0, data = m0>	→	A tiene 3 búferes libres
4	→	<seq = 1, data = m1>	→	A tiene 2 búferes libres
5	→	<seq = 2, data = m2>	• • •	Mensaje perdido, pero A piensa que le queda 1
6	←	<ack = 1, buf = 3>	←	B confirma la recepción de 0 y 1, permite 2-4
7	→	<seq = 3, data = m3>	→	A tiene un búfer libre
8	→	<seq = 4, data = m4>	→	A tiene 0 búferes libres y debe detenerse
9	→	<seq = 2, data = m2>	→	El temporizador de A expira y retransmite
10	←	<ack = 4, buf = 0>	←	Todo confirmado, pero A bloqueado aún
11	←	<ack = 4, buf = 1>	←	A puede enviar 5 ahora
12	←	<ack = 4, buf = 2>	←	B encontró un búfer nuevo en algún lado
13	→	<seq = 5, data = m5>	→	A tiene 1 búfer libre
14	→	<seq = 6, data = m6>	→	A está bloqueado nuevamente
15	←	<ack = 6, buf = 0>	←	A aún está bloqueado
16	• • •	<ack = 6, buf = 4>	←	Bloqueo irreversible potencial

Figura 6-16. Asignación dinámica de búferes. Las flechas muestran la dirección de la transmisión. Los puntos suspensivos (...) indican una TPDU perdida.

mente una TPDU de control con la confirmación de recepción y estado de búferes de cada conexión. De esta manera, el estancamiento se romperá tarde o temprano.

Hasta ahora hemos supuesto tácitamente que el único límite impuesto a la tasa de datos del emisor es la cantidad de espacio de búfer disponible en el receptor. A medida que siga cayendo significativamente el precio de la memoria, podría llegar a ser factible equipar a los *hosts* con tanta memoria que la falta de búferes dejaría de ser un problema.

Si el espacio de búfer ya no limita el flujo máximo, aparecerá otro cuello de botella: la capacidad de carga de la subred. Si enrutadores adyacentes pueden intercambiar cuando mucho x paquetes/seg y hay k trayectorias sin traslape entre un par de *hosts*, no hay manera de que esos *hosts* puedan intercambiar más de kx TPDUs/seg, sin importar la cantidad de espacio de búfer disponible en cada terminal. Si el emisor presiona demasiado (es decir, envía más de kx TPDUs/seg), la subred se congestionará, pues será incapaz de entregar las TPDUs a la velocidad con que llegan.

Lo que se necesita es un mecanismo basado en la capacidad de carga de la subred en lugar de la capacidad de almacenamiento en búfer del receptor. Es claro que el mecanismo de control de flujo debe aplicarse al emisor para evitar que estén pendientes demasiadas TPDUs sin confirmación de recepción al mismo tiempo. Belsnes (1975) propuso el uso de un esquema de control de flujo de ventana corrediza en el que el emisor ajusta dinámicamente el tamaño de la ventana para igualarla a la capacidad de carga de la red. Si la red puede manejar c TPDUs/seg y el tiempo de ciclo (incluidos transmisión, propagación, encolamiento, procesamiento en el receptor y devolución de la confirmación de recepción) es de r, entonces la ventana del emisor debe ser cr. Con una ventana de este tamaño, el emisor normalmente opera con el canal a su máxima capacidad. Cualquier pequeña disminución en el desempeño de la red causará que se bloquee.

A fin de ajustar periódicamente el tamaño de la ventana, el emisor podría vigilar ambos parámetros y calcular después el tamaño de ventana deseado. La capacidad de carga puede determinarse con sólo contar la cantidad de TPDUs confirmadas durante algún periodo y dividirla entre el periodo. Durante la medición, el emisor debe enviar a la mayor velocidad posible, para asegurarse que sea la capacidad de carga de la red, y no la baja tasa de entrada, el factor limitante de la tasa de confirmaciones de recepción. El tiempo requerido para la confirmación de recepción de una TPDU transmitida puede medirse exactamente y mantenerse una media de operación. Dado que la capacidad de la red depende de la cantidad de tráfico en ella, debe ajustarse el tamaño de la ventana con frecuencia para responder a los cambios en la capacidad de carga. Como veremos después, Internet usa un esquema parecido.

6.2.5 Multiplexión

La multiplexión de varias conversaciones en conexiones, circuitos virtuales y enlaces físicos desempeña un papel importante en diferentes capas de la arquitectura de la red. En la capa de transporte puede surgir la necesidad de multiplexión por varias razones. Por ejemplo, si en un *host* sólo se dispone de una dirección de red, todas las conexiones de transporte de esa máquina tendrán que utilizarla. Cuando llega una TDPU, se necesita algún mecanismo para saber a cuál proceso asignarla. Esta situación, conocida como **multiplexión hacia arriba,** se muestra en la figura 6-17(a). En esta figura, cuatro distintas conexiones de transporte utilizan la misma conexión de red (por ejemplo, dirección IP) al *host* remoto.

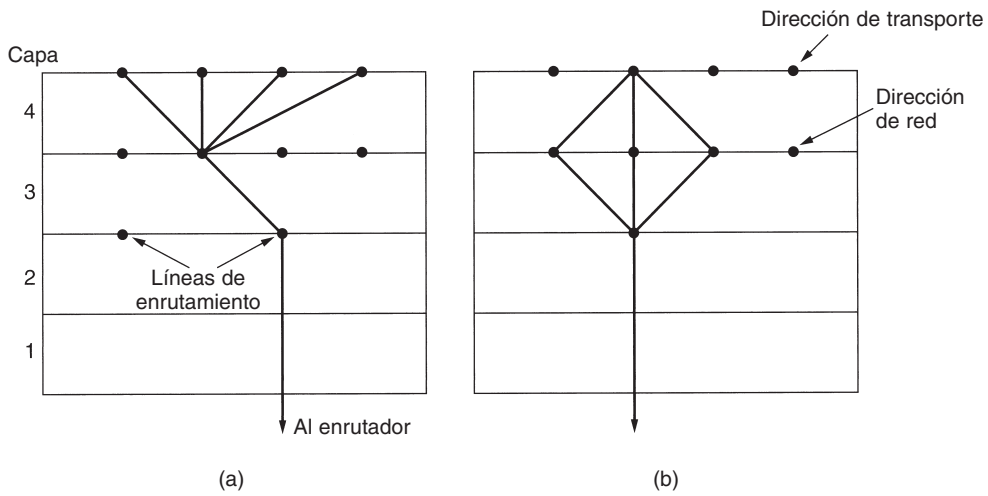


Figura 6-17. (a) Multiplexión hacia arriba. (b) Multiplexión hacia abajo.

La multiplexión también puede ser útil en la capa de transporte por otra razón. Por ejemplo, supongamos que una subred utiliza circuitos virtuales de manera interna y que impone una tasa máxima de datos a cada uno. Si un usuario necesita más ancho de banda del que le puede proporcionar un circuito virtual, una alternativa es abrir múltiples conexiones de red y distribuir el tráfico entre ellas en *round robin* (de manera circular), como se muestra en la figura 6-17(b). Esto se denomina **multiplexión hacia abajo.** Con *k* conexiones de red abiertas, el ancho de banda efectivo se incrementa por un factor de *k.* Un ejemplo común de multiplexión hacia abajo se da en los usuarios caseros que poseen una línea ISDN. Esta línea proporciona dos conexiones separadas de 64 kbps cada una. Al usar ambas para llamar a un proveedor de Internet y dividir el tráfico en las dos líneas se consigue un ancho de banda efectivo de 128 kbps.

6.2.6 Recuperación de caídas

Si los *hosts* y los enrutadores están sujetos a caídas, la recuperación de éstas se vuelve un tema importante. Si la entidad de transporte está por entero dentro de los *hosts*, la recuperación de caídas de la red y de enrutadores es sencilla. Si la capa de red proporciona servicio de datagramas, las entidades de transporte esperan la pérdida de algunas TPDUs todo el tiempo, y saben cómo manejarla. Si la capa de red proporciona servicio orientado a la conexión, entonces la pérdida de un circuito virtual se maneja estableciendo uno nuevo y sondeando la entidad de transporte remota para saber cuáles TPDUs ha recibido y cuáles no. Estas últimas pueden retransmitirse.

Un problema más complicado es la manera de recuperarse de caídas del *host*. En particular, podría ser deseable que los clientes sean capaces de continuar trabajando cuando los servidores caen y se reinician rápidamente. Para ilustrar la dificultad, supongamos que un *host*, el cliente, envía un archivo grande a otro *host*, el servidor de archivos, usando un protocolo de parada y espera sencillo. La capa de transporte del servidor simplemente pasa las TPDUs entrantes al usuario de transporte, una por una. A medio camino de la transmisión se cae el servidor. Al reactivarse, sus tablas se reinicializan, por lo que ya no sabe con precisión dónde estaba.

En un intento por recuperar su estado previo, el servidor podría enviar una TPDU de difusión a todos los demás *hosts*, anunciando que acaba de caerse y solicitando a sus clientes que le informen el estado de todas las conexiones abiertas. Cada cliente puede estar en uno de dos estados: una TPDU pendiente, *S1*, o ninguna TPDU pendiente, *S0*. Con base en esta información de estado, el cliente debe decidir si retransmitirá o no la TPDU más reciente.

A primera vista parecería obvio: el cliente debe retransmitir sólo si tiene una TPDU pendiente sin confirmación de recepción (es decir, si está en el estado *S1*) cuando se entera de la caída. Sin embargo, una inspección más cercana revela dificultades con este enfoque ingenuo. Considere, por ejemplo, la situación en la que la entidad de transporte del servidor envía primero una confirmación de recepción y luego escribe en el proceso de aplicación. La escritura de una TPDU en este flujo de salida y el envío de una confirmación de recepción son dos eventos diferentes que no pueden hacerse simultáneamente. Si ocurre una caída tras el envío de la confirmación de recepción

pero antes de hacer la escritura, el cliente recibirá la confirmación de recepción y estará, por tanto, en el estado *S0* al llegar el anuncio de recuperación de la caída. Entonces el cliente no retransmitirá, pensando (incorrectamente) que la TPDU llegó. Esta decisión del cliente conduce a una TPDU faltante.

En este punto el lector podría pensar: "Ese problema se resuelve fácilmente. Todo lo que hay que hacer es reprogramar la entidad de transporte para que primero haga la escritura y luego envíe la confirmación de recepción". Piénselo mejor. Imagine que se ha hecho la escritura pero que la caída ocurre antes de poder enviar la confirmación de recepción. El cliente estará en el estado *S1*, y por tanto retransmitirá, conduciendo a una TPDU duplicada no detectada en el flujo de salida al proceso de aplicación del servidor.

Sin importar cómo se programen el emisor y el receptor, siempre habrá situaciones en las que el protocolo no podrá recuperarse correctamente. El servidor puede programarse de una de dos maneras: mandar confirmación de recepción primero o escribir primero. El cliente puede programarse de cuatro maneras: siempre retransmitir la última TPDU, nunca retransmitir la última TPDU, retransmitir sólo en el estado *S0* o retransmitir sólo en el estado *S1*. Esto da ocho combinaciones pero, como veremos, para cada combinación existe algún grupo de eventos que hacen fallar al protocolo.

Son posibles tres eventos en el servidor: el envío de una confirmación de recepción (*A*), la escritura al proceso de salida (*W*) y una caída (*C*). Los tres eventos pueden ocurrir en seis órdenes diferentes: *AC*(*W*), *AWC*, *C*(*AW*), *C*(*WA*), *WAC* y *WC*(*A*), donde los paréntesis se usan para indicar que ni *A* ni *W* pueden seguir a *C* (es decir, una vez que el servidor se cae, se cayó). En la figura 6-18 se muestran las ocho combinaciones de la estrategia cliente-servidor y las secuencias de eventos válidas para cada una. Observe que para cada estrategia hay alguna secuencia de eventos que causa la falla del protocolo. Por ejemplo, si el cliente siempre retransmite, el evento *AWC* generará un duplicado no detectado, aun si los otros dos eventos funcionan adecuadamente.

Hacer más elaborado el protocolo no sirve de nada. Aunque el cliente y el servidor intercambien varias TPDUs antes de que el servidor intente escribir, para que el cliente sepa exactamente lo que está a punto de ocurrir, el cliente no tiene manera de saber si ha ocurrido una caída justo antes o justo después de la escritura. La conclusión es inevitable: según nuestra regla básica de que no debe haber eventos simultáneos, la caída de un *host* y su recuperación no pueden hacerse transparentes a las capas superiores.

En términos más generales, este resultado puede replantearse como que la recuperación de una caída de capa N sólo puede hacerla la capa $N + 1$, y sólo si la capa superior retiene suficiente información del estado. Como se mencionó antes, la capa de transporte puede recuperarse de fallas de la capa de red, siempre y cuando cada extremo de una conexión lleve el registro de dónde está.

Este problema nos lleva al asunto de qué significa en realidad una confirmación de recepción de extremo a extremo. En principio, el protocolo de transporte es de extremo a extremo, y no está encadenado como las capas inferiores. Ahora considere el caso de un usuario que introduce solicitudes de transacciones a una base de datos remota. Suponga que la entidad de transporte remota está programada para pasar primero las TPDUs a la siguiente capa superior y luego emitir las confirmaciones de recepción. Aun en este caso, la recepción de una confirmación de

Estrategia usada por el *host* emisor	Estrategia usada por el *host* receptor: Primero ACK, luego escritura			Primero escritura, luego ACK		
	AC(W)	AWC	C(AW)	C(WA)	WAC	WC(A)
Siempre retransmitir	BIEN	DUP.	BIEN	BIEN	DUP.	DUP.
Nunca retransmitir	PERD.	BIEN	PERD.	PERD.	BIEN	BIEN
Retransmitir en S0	BIEN	DUP.	PERD.	PERD.	DUP.	BIEN
Retransmitir en S1	PERD.	BIEN	BIEN	BIEN	BIEN	DUP.

BIEN = El protocolo funciona correctamente
DUP. = El protocolo genera un mensaje duplicado
PERD. = El protocolo pierde un mensaje

Figura 6-18. Diferentes combinaciones de la estrategia cliente-servidor.

recepción en la máquina del usuario no significa necesariamente que el *host* remoto se quedó encendido suficiente tiempo para actualizar la base de datos. Probablemente es imposible lograr una verdadera confirmación de recepción de extremo a extremo, cuya recepción significa que el trabajo se ha hecho, y cuya falta significa que no. Este punto lo analizan con mayor detalle Saltzer y cols. (1984.)

6.3 UN PROTOCOLO DE TRANSPORTE SENCILLO

Para hacer más concretas las ideas estudiadas hasta ahora, en esta sección estudiaremos detalladamente una capa de transporte de ejemplo. Las primitivas de servicio abstractas que usaremos son las primitivas orientadas a la conexión de la figura 6-2. La elección de estas primitivas hace que el ejemplo sea similar (pero más sencillo) al popular protocolo TCP.

6.3.1 Las primitivas de servicio de ejemplo

Nuestro primer problema es cómo expresar de manera concreta estas primitivas de transporte. CONNECT es fácil: sencillamente tenemos un procedimiento de biblioteca *connect* que puede llamarse con los parámetros adecuados necesarios para establecer una conexión. Los parámetros son los TSAPs locales y remotos. Durante la llamada, el invocador se bloquea (es decir, se suspende) mientras la entidad de transporte intenta establecer la conexión. Si ésta tiene éxito, se desbloquea el invocador y puede comenzar la transmisión de datos.

Cuando un proceso requiere la capacidad de aceptar llamadas entrantes, llama a *listen*, especificando un TSAP en particular en el que quiere escuchar. El proceso entonces se bloquea hasta que algún proceso remoto intenta establecer una conexión con su TSAP.

Observe que este modelo es altamente asimétrico. Un lado es pasivo, pues ejecuta un *listen* y espera hasta que ocurre algo. El otro lado es activo e inicia la conexión. Una pregunta interesante es qué debe hacerse si el lado activo comienza primero. Una estrategia es hacer que falle el intento de conexión si no hay un escuchador en el TSAP remoto. Otra estrategia es hacer que el iniciador se bloquee (posiblemente para siempre) hasta que aparezca un escuchador.

Una alternativa, usada en nuestro ejemplo, es mantener la solicitud de conexión del lado receptor durante cierto intervalo de tiempo. Si un proceso de ese *host* llama a *listen* antes de que expire el temporizador, se establece la conexión; de otro modo se rechaza y se desbloquea al invocador, devolviéndole un error.

Para liberar una conexión, usaremos un procedimiento *disconnect*. Cuando ambos lados se han desconectado, se libera la conexión. En otras palabras, estamos usando un modelo de desconexión simétrico.

La transmisión de datos tiene precisamente el mismo problema que el establecimiento de una conexión: el envío es activo, pero la recepción es pasiva. Usaremos la misma solución para la transmisión de datos que para el establecimiento de la conexión: una llamada activa *send* que transmite datos, y una llamada pasiva *receive* que bloquea hasta que llega una TPDU.

Nuestra definición concreta del servicio consiste, por tanto, en cinco primitivas: CONNECT, LISTEN, DISCONNECT, SEND y RECEIVE. Cada primitiva corresponde exactamente a un procedimiento de biblioteca que ejecuta la primitiva. Los parámetros de las primitivas de servicio y los procedimientos de biblioteca son los siguientes:

```
connum = LISTEN(local)
connum = CONNECT(local, remote)
status  = SEND(connum, buffer, bytes)
status  = RECEIVE(connum, buffer, bytes)
status  = DISCONNECT(connum)
```

La primitiva LISTEN anuncia la disposición del invocador a aceptar solicitudes de conexión dirigidas al TSAP indicado. El usuario de la primitiva se bloquea hasta que se hace un intento de conexión a él. No hay expiración del temporizador.

La primitiva CONNECT toma dos parámetros, un TSAP local (es decir, dirección de transporte), *local*, y un TSAP remoto, *remote,* e intenta establecer una conexión de transporte entre los dos. Si tiene éxito, devuelve en *connum* un número no negativo que sirve para identificar la conexión en llamadas subsiguientes. Si falla, la razón del fracaso se pone en *connum* como número negativo. En nuestro modelo sencillo, cada TSAP puede participar sólo en una conexión de transporte, por lo que una razón posible de falla es que una de las direcciones de transporte ya esté en uso. Algunas otras razones son: *host* remoto caído, dirección local ilegal y dirección remota ilegal.

La primitiva SEND transmite el contenido del búfer como mensaje por la conexión de transporte indicada, posiblemente en varias unidades si es demasiado grande. Los errores posibles, devueltos en *status*, son falta de conexión, dirección de búfer ilegal y cuenta negativa.

La primitiva RECEIVE indica el deseo del invocador de aceptar datos. El tamaño del mensaje de entrada se pone en *bytes*. Si el proceso remoto ha liberado la conexión o la dirección de búfer es ilegal (por ejemplo, fuera del programa del usuario), *status* se establece a un código de error que indica la naturaleza del problema.

La primitiva DISCONNECT termina una conexión de transporte; el parámetro *connum* indica cuál. Los errores posibles son que *connum* pertenezca a otro proceso, o que *connum* no sea un identificador de conexión válido. El código de error, o 0 si todo funcionó bien, se devuelve en *status*.

6.3.2 La entidad de transporte de ejemplo

Antes de ver el código de la entidad de transporte de ejemplo, tenga bien presente que este ejemplo es análogo a los primeros ejemplos presentados en el capítulo 3: su propósito es pedagógico, no una propuesta seria. En aras de la sencillez, se han omitido aquí muchos de los detalles técnicos (como comprobación extensa de errores) necesarios para un sistema de producción.

La capa de transporte hace uso de las primitivas de servicio de red para enviar y recibir las TPDUs. Para este ejemplo, necesitamos escoger las primitivas de servicio de red a usar. Una selección podría haber sido el servicio de datagramas no confiable. No lo hemos seleccionado para mantener sencillo el ejemplo. Con el servicio de datagramas no confiable, el código de transporte habría sido grande y complejo, encargándose principalmente de paquetes perdidos y retrasados. Es más, muchas de estas ideas ya se analizaron con detalle en el capítulo 3.

En cambio, hemos optado por usar un servicio de red confiable orientado a la conexión. De esta manera, podremos enfocarnos a los asuntos de transporte que no ocurren en las capas inferiores. Entre éstos tenemos el establecimiento de conexiones, la liberación de conexiones y la administración de crédito. Un servicio sencillo de transporte construido encima de una red ATM podría verse así.

En general, la entidad de transporte puede ser parte del sistema operativo del *host* o puede ser un paquete de rutinas de biblioteca operando en el espacio de direcciones del usuario. Por sencillez, nuestro ejemplo se ha programado como si fuera un paquete de biblioteca, pero los cambios necesarios para hacerlo parte del sistema operativo son mínimos (principalmente la manera de acceder a los búferes de usuario).

Sin embargo, vale la pena señalar que en este ejemplo la "entidad de transporte" no es en realidad una entidad independiente, sino parte del proceso de usuario. En particular, cuando el usuario ejecuta una primitiva que se bloquea, como LISTEN, toda la entidad de transporte se bloquea también. En tanto que este diseño es adecuado para un *host* con un solo proceso de usuario, en un *host* con varios usuarios sería más natural hacer que la entidad de transporte sea un proceso aparte, diferente de todos los procesos de usuario.

La interfaz con la capa de red se establece mediante los procedimientos *to_net* y *from_net* (no se muestran). Cada uno tiene seis parámetros. Primero viene el identificador de conexión, que se relaciona uno a uno con los circuitos virtuales de la red. A continuación vienen los bits *Q* y *M* que, al estar establecidos en 1, indican "mensaje de control" y "continúan más datos de este mensaje

en el siguiente paquete", respectivamente. Tras esto tenemos el tipo de paquete, seleccionado entre los seis tipos de paquete listados en la figura 6-19. Por último, tenemos un apuntador a los datos mismos y un entero que indica el número de bytes de datos.

Paquete de red	Significado
CALL REQUEST	Se envía para establecer una conexión
CALL ACCEPTED	Respuesta a CALL REQUEST
CLEAR REQUEST	Se envía para liberar una conexión
CLEAR CONFIRMATION	Respuesta a CLEAR REQUEST
DATA	Sirve para transportar datos
CREDIT	Paquete de control para manejar la ventana

Figura 6-19. Paquetes de capa de red usados en nuestro ejemplo.

En las llamadas a *to_net*, la entidad de transporte llena todos los parámetros para que los lea la capa de red; en las llamadas a *from_net*, la capa de red divide un paquete de entrada antes de pasarlo a la entidad de transporte. Al pasar información como parámetros de procedimiento en lugar de pasar el paquete de salida o entrada mismo, la capa de transporte queda protegida de los detalles del protocolo de capa de red. Si la entidad de transporte intenta enviar un paquete cuando la ventana corrediza del circuito virtual subyacente esté llena, se suspende en *to_net* hasta que haya espacio en la ventana. Este mecanismo es transparente para la entidad de transporte y lo controla la capa de red usando comandos como *enable_transport_layer* y *disable_transport_layer* análogos a los descritos en los protocolos del capítulo 3. La capa de red también realiza la administración de la ventana de la capa de paquetes.

Además de este mecanismo de suspensión transparente, hay procedimientos *sleep* y *wakeup* explícitos (que no se muestran) invocados por la entidad de transporte. El procedimiento *sleep* se invoca cuando la entidad de transporte está bloqueada lógicamente en espera de un evento externo, generalmente la llegada de un paquete. Tras haber llamado a *sleep*, la entidad de transporte (y, por supuesto, el proceso de usuario) deja de ejecutarse.

El código de la entidad de transporte se muestra en la figura 6-20. Cada conexión está siempre en uno de siete estados, a saber:

1. IDLE—No se ha establecido todavía una conexión.
2. WAITING—Se ha ejecutado CONNECT y enviado CALL REQUEST.
3. QUEUED—Ha llegado una CALL REQUEST; aún no hay LISTEN.
4. ESTABLISHED—Se ha establecido la conexión.
5. SENDING—El usuario está esperando permiso de enviar un paquete.
6. RECEIVING—Se ha hecho un RECEIVE.
7. DISCONNECTING—Se ha hecho localmente un DISCONNECT.

Las transiciones entre estados pueden ocurrir al suceder cualquiera de los siguientes eventos: se ejecuta una primitiva, llega un paquete o expira el temporizador.

Los procedimientos mostrados en la figura 6-20 son de dos tipos. Casi todos pueden invocarse directamente desde los programas de usuario. Sin embargo, *packet_arrival* y *clock* son diferentes. Eventos externos los activan espontáneamente: la llegada de un paquete y los pulsos del reloj, respectivamente. De hecho, son rutinas de interrupción. Supondremos que no se invocan mientras está ejecutándose un procedimiento de identidad de transporte; sólo pueden invocarse cuando el proceso de usuario está dormido o ejecutándose fuera de la entidad de transporte. Esta propiedad es crucial para el funcionamiento correcto del código.

La existencia del bit Q (calificador) en el encabezado del paquete nos permite evitar la sobrecarga de un encabezado de protocolo de transporte. Los mensajes de datos ordinarios se envían como paquetes de datos con $Q = 0$. Los mensajes de control de protocolo de transporte, de los cuales sólo hay uno (CREDIT) en nuestro ejemplo, se envían como paquetes de datos con $Q = 1$. Estos mensajes de control son detectados y procesados por la entidad de transporte receptora.

La principal estructura de datos usada por la entidad de transporte es el arreglo *conn,* que tiene un registro por cada conexión potencial. El registro mantiene el estado de la conexión, incluidas las direcciones de transporte en ambos extremos, la cantidad de mensajes enviados y recibidos por la conexión, el estado actual, el apuntador al búfer de usuario, la cantidad de bytes de los mensajes actuales enviados o recibidos hasta el momento, un bit que indica si el usuario remoto ha emitido o no un DISCONNECT, un temporizador y un contador de permisos que sirve para habilitar el envío de mensajes. No todos estos campos se usan en nuestro ejemplo sencillo, pero una entidad de transporte completa los necesitaría todos, y tal vez más. Se supone que cada entrada de *conn* está inicialmente en el estado inactivo (*IDLE*).

Cuando el usuario llama a CONNECT, la capa de red recibe instrucciones para enviar un paquete CALL REQUEST a la máquina remota, y el usuario se pone a dormir. Cuando llega el paquete CALL REQUEST al otro lado, la entidad de transporte se interrumpe para ejecutar *packet_arrival* a fin de comprobar que el usuario local está escuchando en la dirección especificada. De ser así, se devuelve un paquete CALL ACCEPTED y se despierta al usuario remoto; de no ser así, la CALL REQUEST se pone en cola durante *TIMEOUT* pulsos de reloj. Si se hace un LISTEN en este periodo, se establece la conexión; de otro modo, expira el temporizador y se rechaza la solicitud con un paquete CLEAR REQUEST para evitar que se bloquee de manera indefinida.

Aunque hemos eliminado el encabezado del protocolo de transporte, aún necesitamos una manera de llevar el control de cuál paquete pertenece a cuál conexión de transporte, ya que pueden existir varias conexiones al mismo tiempo. El enfoque más sencillo es usar el número de circuito virtual de la capa de red también como número de conexión de transporte. Es más, puede usarse también el número de circuito virtual como índice del arreglo *conn.* Al llegar un paquete por el circuito virtual k de la capa de red, pertenecerá a la conexión de transporte k, cuyo estado está en el registro *conn*[k]. En las conexiones iniciadas en un *host*, la entidad de transporte originadora selecciona el número de conexión. En las llamadas entrantes, la capa de red hace la selección, escogiendo cualquier número de circuito virtual no usado.

Para evitar el hecho de tener que proporcionar y administrar *búferes* en la entidad de transporte, se usa aquí un mecanismo de control de flujo diferente de la ventana corrediza tradicional.

```
#define MAX_CONN 32                   /* número máximo de conexiones simultáneas */
#define MAX_MSG_SIZE 8192             /* mensaje más grande en bytes */
#define MAX_PKT_SIZE 512              /* paquete más grande en bytes */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT} pkt_type;
typedef enum {IDLE,WAITING,QUEUED,ESTABLISHED,SENDING,RECEIVING,DISCONN} cstate;

/* Variables globales. */
transport_address listen_address;     /* dirección local en la que se está escuchando */
int listen_conn;                      /* identificador de conexión para escuchar */
unsigned char data[MAX_PKT_SIZE];     /* área de trabajo para datos de paquetes */

struct conn {
 transport_address local_address, remote_address;
 cstate state;                        /* estado de esta conexión */
 unsigned char *user_buf_addr;        /* apuntador al búfer de recepción */
 int byte_count;                      /* conteo de envío/recepción */
 int clr_req_received;                /* establecido al recibir el paquete CLEAR_REQ */
 int timer;                           /* se utiliza para establecer el temporizador
                                         de paquetes CALL_REQ */
 int credits;                         /* cantidad de mensajes que se puede enviar */
}conn[MAX_CONN + 1];                  /* la ranura 0 no se utiliza */

void sleep(void);                     /* prototipos */
void wakeup(void);
void to_net(int cid, int q, int m, pkt_type pt, unsigned char *p, int bytes);
void from_net(int *cid, int *q, int *m, pkt_type *pt, unsigned char *p, int *bytes);

int listen(transport_address t)
{/* El usuario desea escuchar por conexión. Ve si CALL_REQ ya llegó. */
 int i, found = 0;

 for (i = 1; i <= MAX_CONN; i++)      /* busca CALL_REQ en la tabla */
     if (conn[i].state == QUEUED && conn[i].local_address == t) {
          found = i;
          break;
     }

  if (found == 0) {
       /* No hay CALL_REQ en espera. Se duerme hasta que llega o expira el
          temporizador. */
       listen_address = t;  sleep();  i = listen_conn ;
  }
  conn[i].state = ESTABLISHED;        /* se ESTABLECE la conexión */
  conn[i].timer = 0;                  /* no se usa el temporizador */
```

```
  listen_conn = 0;                    /* se supone que 0 es una dirección inválida */
  to_net(i, 0, 0, CALL_ACC, data, 0); /* indica a la red que acepte una conexión */
  return(i);                          /* devuelve el identificador de conexión */
 }

 int connect(transport_address l, transport_address r)
 { /* El usuario desea conectarse a un proceso remoto; envía paquete CALL_REQ. */
  int i;
  struct conn *cptr;

  data[0] = r;   data[1] = l;         /* el paquete CALL_REQ los necesita */
  i = MAX_CONN;                       /* busca hacia atrás en la tabla */
  while (conn[i].state != IDLE && i > 1) i = i - 1;
  if (conn[i].state == IDLE) {
       /* Crea una entrada en la tabla indicando que CALL_REQ se ha enviado. */
       cptr = &conn[i];
       cptr->local_address = l; cptr->remote_address = r;
       cptr->state = WAITING; cptr->clr_req_received = 0;
       cptr->credits = 0; cptr->timer = 0;
       to_net(i, 0, 0, CALL_REQ, data, 2);
       sleep();                       /* espera CALL_ACC o CLEAR_REQ */
       if (cptr->state == ESTABLISHED) return(i);
       if (cptr->clr_req_received) {
            /* El otro lado rechazó la llamada. */
            cptr->state = IDLE;       /* regresa al estado de inactividad */
            to_net(i, 0, 0, CLEAR_CONF, data, 0);
            return(ERR_REJECT);
       }
  } else return(ERR_FULL);            /* rechaza la conexión: no hay espacio en la
                                         tabla */
}

int send(int cid, unsigned char bufptr[], int bytes)
{ /* El usuario desea enviar un mensaje. */
  int i, count, m;
  struct conn *cptr = &conn[cid];

  /* Pasa al estado de transmisión. */
  cptr->state = SENDING;
  cptr->byte_count = 0;               /* # de bytes enviados hasta ahora de este
                                         mensaje */
  if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
  if (cptr->clr_req_received == 0) {
       /* Crédito disponible; divide el mensaje en paquetes si es necesario. */
       do {
            if (bytes - cptr->byte_count > MAX_PKT_SIZE) { /* mensaje con múltiples
                                                              paquetes */
                 count = MAX_PKT_SIZE;  m = 1;    /* más paquetes posteriormente */
             } else {                 /* mensaje de un solo paquete */
                 count = bytes - cptr->byte_count;  m = 0; /* último paquete de este
                                                              mensaje*/
             }
             for (i = 0; i < count; i++) data[i] = bufptr[cptr->byte_count + i];
             to_net(cid, 0, m, DATA_PKT, data, count);     /* envía un paquete */
             cptr->byte_count = cptr->byte_count + count;  /* incrementa los bytes
                                                              enviados hasta ahora */
       } while (cptr->byte_count < bytes);  /* repite el ciclo hasta enviar todo el
                                               mensaje */
```

```
      cptr->credits--;                /* cada mensaje utiliza un crédito */
      cptr->state = ESTABLISHED;
      return(OK);
 } else {
      cptr->state = ESTABLISHED;
      return(ERR_CLOSED);             /* transmisión fallida: el igual desea
                                         desconectarse */
 }
}

int receive(int cid, unsigned char bufptr[], int *bytes)
{ /* El usuario está preparado para recibir un mensaje. */
  struct conn *cptr = &conn[cid];

  if (cptr->clr_req_received == 0) {
       /* La conexión aún permanece; trata de recibir. */
       cptr->state = RECEIVING;
       cptr->user_buf_addr = bufptr;
       cptr->byte_count = 0;
       data[0] = CRED;
       data[1] = 1;
       to_net(cid, 1, 0, CREDIT, data, 2); /* envía crédito */
       sleep();                       /* se bloquea en espera de datos */
       *bytes = cptr->byte_count;
 }
 cptr->state = ESTABLISHED;
 return(cptr->clr_req_received ? ERR_CLOSED : OK);
}

int disconnect(int cid)
{ /* El usuario desea liberar una conexión. */
  struct conn *cptr = &conn[cid];

  if (cptr->clr_req_received) {       /* el otro lado inició la terminación */
       cptr->state = IDLE;            /* se libera la conexión */
       to_net(cid, 0, 0, CLEAR_CONF, data, 0);
  } else {                            /* nosotros iniciamos la terminación */
      cptr->state = DISCONN;          /* no liberada hasta que el otro lado esté
                                         de acuerdo */
      to_net(cid, 0, 0, CLEAR_REQ, data, 0);
  }
  return(OK);
}

void packet_arrival(void)
{ /* Ha llegado un paquete, se obtiene y procesa. */
  int cid;                            /* conexión en la cual arribó el paquete */
  int count, i, q, m;
  pkt_type ptype;     /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
  unsigned char data[MAX_PKT_SIZE];   /* porción de datos del paquete que llegó */
  struct conn *cptr;

  from_net(&cid, &q, &m, &ptype, data, &count);     /* se obtiene*/
  cptr = &conn[cid];
```

```
  switch (ptype) {
   case CALL_REQ:                    /* el usuario remoto desea establecer conexión */
    cptr->local_address = data[0];  cptr->remote_address = data[1];
    if (cptr->local_address == listen_address) {
          listen_conn = cid;  cptr->state = ESTABLISHED;  wakeup();
    } else {
         cptr->state = QUEUED;  cptr->timer = TIMEOUT;
    }
    cptr->clr_req_received = 0;   cptr->credits = 0;
    break;

   case CALL_ACC:                    /* el usuario remoto ha aceptado nuestra CALL_
                                        REQ */
    cptr->state = ESTABLISHED;
    wakeup();
    break;

   case CLEAR_REQ:                   /* usuario remoto desea desconectarse o rechazar
                                        la llamada */
    cptr->clr_req_received = 1;
    if (cptr->state == DISCONN) cptr->state = IDLE; /* limpia la colisión */
    if (cptr->state == WAITING || cptr->state == RECEIVING || cptr->state == SENDING)
       wakeup();
    break;

   case CLEAR_CONF:                  /* el usuario remoto está de acuerdo en la
                                        desconexión */
    cptr->state = IDLE;
    break;

   case CREDIT:                      /* el usuario remoto espera datos */
    cptr->credits += data[1];
    if (cptr->state == SENDING) wakeup();
    break;

   case DATA_PKT:                    /* el usuario remoto ha enviado datos */
    for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
    cptr->byte_count += count;
    if (m == 0 ) wakeup();
  }
}
void clock(void)
{ /* El reloj pulsó, verificar si expiró el temporizador de solicitudes de conexión
     en cola. */
  int i;
  struct conn *cptr;
  for (i = 1; i <= MAX_CONN; i++) {
      cptr = &conn[i];
      if (cptr->timer > 0) {         /* el temporizador estaba en funcionamiento */
            cptr->timer--;
            if (cptr->timer == 0) { /* el temporizador expiró */
                  cptr->state = IDLE;
                  to_net(i, 0, 0, CLEAR_REQ, data, 0);
            }
      }
  }
}
```

Figura 6-20. Entidad de transporte de ejemplo.

Cuando un usuario llama a RECEIVE, se envía un **mensaje de crédito** especial a la entidad de transporte de la máquina emisora y se registra en el arreglo *conn*. Al llamar a SEND, la entidad de transporte revisa si ha llegado un crédito en la conexión especificada. De ser así, el mensaje se envía (en múltiples paquetes de ser necesario) y se disminuye el crédito; en caso contrario, la entidad de transporte se pone a dormir hasta la llegada de un crédito. Este mecanismo garantiza que no se enviará nunca un mensaje a menos que el otro lado ya haya hecho un RECEIVE. Como resultado, siempre que llegue un mensaje habrá un búfer disponible en el cual puede ponerse. El esquema puede generalizarse fácilmente para permitir que los receptores proporcionen múltiples búferes y soliciten múltiples mensajes.

Es importante no olvidar la sencillez de la figura 6-20. Una entidad de transporte para uso real normalmente comprobaría la validez de todos los parámetros proporcionados por el usuario, manejaría la recuperación de caídas de la capa de red, se encargaría de colisiones de llamadas y soportaría un servicio de transporte más general, incluidas funciones como interrupciones, datagramas y versiones no bloqueadoras de las primitivas SEND y RECEIVE.

6.3.3 El ejemplo como máquina de estados finitos

Escribir una entidad de transporte es un trabajo difícil y exigente, especialmente para los protocolos más realistas. A fin de reducir la posibilidad de cometer errores, el estado del protocolo se puede representar como máquina de estados finitos.

Ya hemos visto que nuestro protocolo de ejemplo tiene siete estados por conexión. También es posible aislar 12 eventos que pueden ocurrir para pasar una conexión de un estado a otro. Cinco de estos eventos son las cinco primitivas de servicio. Otros seis son las llegadas de los seis tipos de paquete legales. El último es la expiración del temporizador. En la figura 6-21 se muestran las acciones principales del protocolo en forma de matriz. Las columnas son los estados y las filas son los 12 eventos.

Cada entrada de la matriz (es decir, la máquina de estados finitos) de la figura 6-21 tiene hasta tres campos: un predicado, una acción y un estado nuevo. El predicado indica en qué condiciones se emprende la acción. Por ejemplo, en la entrada de la esquina superior izquierda, si se ejecuta una LISTEN y no hay más espacio de tabla (predicado *P1*) falla la LISTEN y no cambia el estado. Por otra parte, si ya ha llegado un paquete CALL REQUEST para la dirección de transporte en la que se está escuchando (predicado *P2*), se establece de inmediato la conexión. Otra posibilidad es que *P2* sea falso, es decir, que no ha entrado ningún CALL REQUEST, en cuyo caso la conexión permanece en el estado *IDLE*, esperando un paquete CALL REQUEST.

Es importante indicar que la selección de estados a usar en la matriz no es fijada completamente por el protocolo mismo. En este ejemplo, no hay ningún estado *LISTENING*, lo que sería algo razonable después de una LISTEN. No hay estado *LISTENING* porque un estado se asocia a un registro de conexión, y LISTEN no crea un registro de conexión. ¿Por qué no? Porque hemos decidido usar los números de circuito virtual de la capa de red como identificadores de la conexión, y el número de circuito virtual para una LISTEN es escogido por la capa de red al llegar el paquete CALL REQUEST.

		Estado						
		Inactivo	En espera	En cola	Establecido	Trans-mitiendo	Recibiendo	Des-conexión
Primitivas	LISTEN	P1: ~/Inactivo P2: A1/Estab. P2: A2/Inactivo		~/Estab.				
	CONNECT	P1: ~/Inactivo P1: A3/En esp.						
	DISCONNECT				P4: A5/Inactivo P4: A6/Desc.			
	SEND				P5: A7/Estab. P5: A8/Trans.			
	RECEIVE				A9/Recib.			
Paquetes de entrada	Call_req	P3: A1/Estab. P3: A4/En cola						
	Call_acc		~/Estab.					
	Clear_req		~/Inactivo		A10/Estab.	A10/Estab.	A10/Estab.	~/Inactivo
	Clear_conf							~/Inactivo
	DataPkt						A12/Estab.	
	Credit				A11/Estab.	A7/Estab.		
Reloj	Timeout			~/Inactivo				

Predicados	Acciones	
P1: Tabla de conexiones llena	A1: Envía Call_acc	A7: Envía mensaje
P2: Call_req pendiente	A2: Espera Call_req	A8: Espera crédito
P3: LISTEN pendiente	A3: Envía Call_req	A9: Envía crédito
P4: Clear_req pendiente	A4: Inicia temporizador	A10: Establece indicador Clr_req_received
P5: Crédito disponible	A5: Envía Clear_conf	A11: Registra crédito
	A6: Envía Clear_req	A12: Acepta mensaje

Figura 6-21. El protocolo de ejemplo como máquina de estados finitos. Cada entrada tiene un predicado opcional, una acción opcional y el estado nuevo. La tilde indica que no se realizó ninguna acción importante. Una barra sobre un predicado indica la negación del predicado. Las entradas en blanco corresponden a eventos imposibles o inválidos.

Las acciones *A1* a *A12* son las acciones principales, como el envío de paquetes y el inicio de temporizadores. No se listan todas las acciones menores, como la inicialización de los campos de un registro de conexión. Si una acción comprende despertar un proceso dormido, las acciones que

siguen a su despertar también cuentan. Por ejemplo, si entra un paquete CALL REQUEST y un proceso estaba dormido esperándolo, la transmisión del paquete CALL ACCEPT que sigue al despertar cuenta como parte de la acción de CALL REQUEST. Tras la ejecución de cada acción, la conexión puede pasar a un estado nuevo, como se muestra en la figura 6-21.

La ventaja de representar el protocolo como una matriz es triple. Primero, de esta manera es mucho más fácil que el programador revise sistemáticamente cada combinación de estado y evento para ver si se requiere una acción. En las implementaciones de producción se usarían algunas de las combinaciones para manejo de errores. En la figura 6-21 no se hace distinción entre situaciones imposibles e ilegales. Por ejemplo, si una conexión está en el estado *waiting*, el evento DISCONNECT es imposible porque el usuario está bloqueado y no puede ejecutar primitivas. Por otra parte, en el estado *sending* no se esperan paquetes de datos porque no se ha emitido crédito. La llegada de un paquete de datos es un error del protocolo.

La segunda ventaja de la representación en matriz del protocolo tiene que ver con su implementación. Podemos imaginar un arreglo de dos dimensiones en el que el elemento $a[i][j]$ es un apuntador o índice al procedimiento que maneja la ocurrencia del evento i cuando se está en el estado j. Una posible implementación es escribir la entidad de transporte como ciclo corto, esperando un evento en la parte superior del ciclo. Al ocurrir un evento, se localiza la conexión pertinente y se extrae su estado. Sabiendo ahora el evento y el estado, la entidad de transporte simplemente indexa en el arreglo a e invoca el procedimiento adecuado. Este enfoque produce un diseño mucho más regular y sistemático que nuestra entidad de transporte.

La tercera ventaja del enfoque de máquina de estados finitos se aprecia en la descripción del protocolo. En algunos documentos estándares, los protocolos se dan como máquinas de estados finitos similares a la de la figura 6-21. Pasar de este tipo de descripción a una entidad de transporte operante es mucho más fácil si la entidad de transporte también es impulsada por una máquina de estados finitos basada en el estándar.

La desventaja principal del enfoque de máquina de estados finitos es que puede ser más difícil de entender que el ejemplo de programación que usamos inicialmente. Sin embargo, este problema puede resolverse parcialmente dibujando la máquina de estados finitos en forma de grafo, como se hace en la figura 6-22.

6.4 LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: UDP

Internet tiene dos protocolos principales en la capa de transporte, uno orientado y otro no orientado a la conexión. En las siguientes secciones analizaremos los dos. El protocolo no orientado a la conexión es UDP. El protocolo orientado a la conexión es TCP. Empezaremos con UDP porque es básicamente IP con un encabezado corto. También veremos dos aplicaciones de UDP.

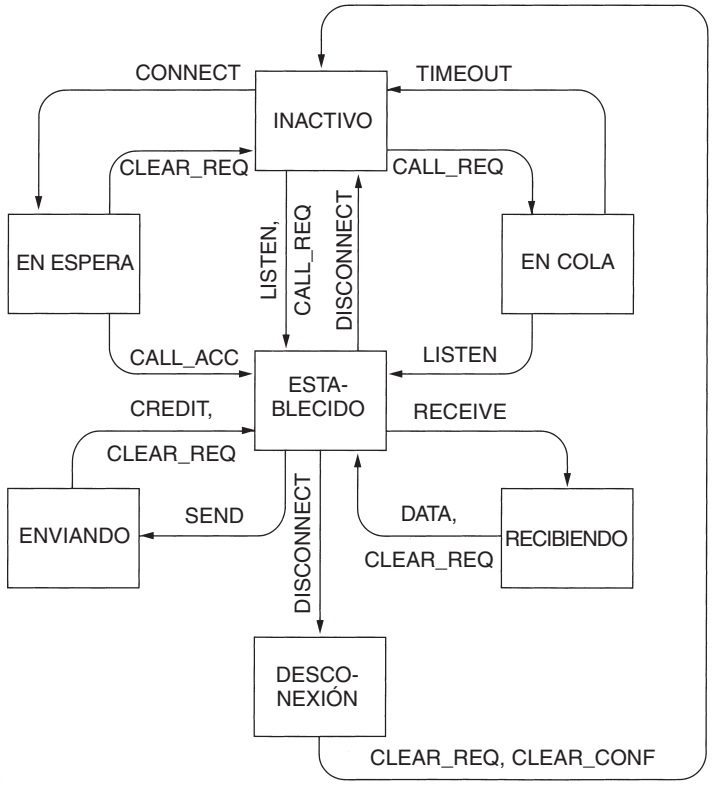


Figura 6-22. El protocolo de ejemplo en forma gráfica. Por sencillez, se han omitido las transiciones que dejan sin cambio el estado de las conexiones.

6.4.1 Introducción a UDP

El conjunto de protocolos de Internet soporta un protocolo de transporte no orientado a la conexión, **UDP** (**Protocolo de Datagramas de Usuario).** Este protocolo proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener que establecer una conexión. UDP se describe en el RFC 768.

UDP transmite **segmentos** que consisten en un encabezado de 8 bytes seguido por la carga útil. En la figura 6-23 se muestra tal encabezado. Los dos puertos sirven para identificar los puntos terminales dentro de las máquinas de origen y destino. Cuando llega un paquete UDP, su carga útil se entrega al proceso que está enlazado al puerto de destino. Este enlace ocurre cuando se utiliza la primitiva BIND o algo similar, como vimos en la figura 6-6 para TCP (el proceso de enlace es el mismo para UDP). De hecho, el valor principal de contar con UDP en lugar de simplemente utilizar el IP puro es la adición de los puertos de origen y destino. Sin los campos de puerto, la capa de transporte no sabría qué hacer con el paquete. Con ellos, entrega los segmentos de manera correcta.

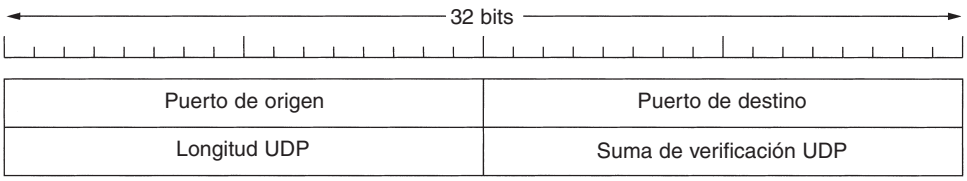


Figura 6-23. El encabezado UDP.

El puerto de origen se necesita principalmente cuando debe enviarse una respuesta al origen. Al copiar el campo *Puerto de origen* del segmento que llega en el campo *Puerto de destino* del segmento que sale, el proceso que envía la respuesta puede especificar cuál proceso de la máquina emisora va a obtenerlo.

El campo *Longitud UDP* incluye el encabezado de 8 bytes y los datos. El campo *Suma de verificación UDP* es opcional y se almacena como 0 si no se calcula (un 0 calculado se almacena como 1s). Su desactivación no tiene sentido a menos que la calidad del servicio de los datos no importe (por ejemplo, en la voz digitalizada).

Probablemente valga la pena mencionar de manera explícita algunas de las cosas que UDP *no* realiza. No realiza control de flujo, control de errores o retransmisión cuando se recibe un segmento erróneo. Todo lo anterior le corresponde a los procesos de usuario. Lo que sí realiza es proporcionar una interfaz al protocolo IP con la característica agregada de desmultiplexar varios procesos utilizando los puertos. Esto es todo lo que hace. Para aplicaciones que necesitan tener control preciso sobre el flujo de paquetes, control de errores o temporización, UDP es lo ideal.

Un área en la que UDP es especialmente útil es en las situaciones cliente-servidor. Con frecuencia, el cliente envía una solicitud corta al servidor y espera una respuesta corta. Si se pierde la solicitud o la respuesta, el cliente simplemente puede terminar y probar nuevamente. El código no sólo es simple, sino que se necesitan muy pocos mensajes (uno en cada dirección) en comparación con un protocolo que requiere una configuración inicial.

Una aplicación que utiliza de esta manera a UDP es DNS (el Sistema de Nombres de Dominio), el cual analizaremos en el capítulo 7. En resumen, un programa que necesita buscar la dirección IP de algún *host*, por ejemplo, *www.cs.berkeley.edu*, puede enviar al servidor DNS un paquete UDP que contenga el nombre de dicho *host*. El servidor responde con un paquete UDP que contiene la dirección IP del *host*. No se necesita configuración por adelantado ni tampoco liberación posterior. Sólo dos mensajes viajan a través de la red.

6.4.2 Llamada a procedimiento remoto

En cierto sentido, enviar un mensaje a un *host* remoto y obtener una respuesta es muy parecido a hacer una llamada a función en un lenguaje de programación. En ambos casos, usted inicia con uno o más parámetros y obtiene un resultado. Esta observación ha llevado a la gente a tratar de que las interacciones de solicitud-respuesta en redes se asignen en forma de llamadas a procedimiento. Esto hace que las aplicaciones de red sean mucho más fáciles de programar y de manejar.

Por ejemplo, imagine un procedimiento llamado *obt_direccion_IP* (*nombre_de_host*) que envía un paquete UDP a un servidor DNS y espera una respuesta, y en caso de que no llegue ninguna con rapidez, termina y lo intenta nuevamente. De esta forma, todos los detalles de la conectividad pueden ocultarse al programador.

El trabajo clave en esta área fue realizado por Birrell y Nelson (1984), quienes sugirieron que se permitiera que los programas invocaran procedimientos localizados en *hosts* remotos. Cuando un proceso en la máquina 1 llama a uno en la máquina 2, el proceso invocador de la primera se suspende y la ejecución del procedimiento se lleva a cabo en la 2. La información se puede transportar desde el invocador al proceso invocado en los parámetros, y se puede regresar en el resultado del procedimiento. El paso de mensajes es transparente para el programador. Esta técnica se conoce como **RPC** (**Llamada a Procedimiento Remoto**) y se ha vuelto la base de muchas aplicaciones de redes. Tradicionalmente, el procedimiento invocador se conoce como cliente y el proceso invocado, como servidor, por lo que aquí utilizaremos esos nombres.

El propósito esencial de RPC es hacer que una llamada a procedimiento remoto sea lo más parecida posible a una local. En la forma más simple, para llamar a un procedimiento remoto, el programa cliente debe enlazarse con un pequeño procedimiento de biblioteca, llamado ***stub* del cliente,** que representa al procedimiento servidor en el espacio de direcciones del cliente. De forma similar, el servidor se enlaza con una llamada a procedimiento denominada ***stub* del servidor.** Estos procedimientos ocultan el hecho de que la llamada a procedimiento desde el cliente al servidor no es local.

En la figura 6-24 se muestran los pasos reales para realizar una RPC. El paso 1 consiste en que el cliente llame al *stub* del cliente. Ésta es una llamada a procedimiento local, y los parámetros se colocan en la pila de la forma tradicional. El paso 2 consiste en que el *stub* del cliente empaca los parámetros en un mensaje y realiza una llamada de sistema para enviar dicho mensaje. El empaquetamiento de los parámetros se conoce como **marshaling.** El paso 3 consiste en que el *kernel* envía el mensaje desde la máquina cliente a la máquina servidor. El paso 4 consiste en que el *kernel* pasa el paquete entrante al *stub* del servidor. Por último, el paso 5 consiste en que el *stub* del servidor llame al procedimiento servidor con parámetros sin *marshaling*. La respuesta sigue la misma ruta en la dirección opuesta.

El elemento clave a notar aquí es que el procedimiento cliente, escrito por el usuario, simplemente realiza una llamada a procedimiento normal (es decir, local) al *stub* del cliente, lo cual tiene el mismo nombre que el procedimiento servidor. Puesto que el procedimiento cliente y el *stub* del cliente están en el mismo espacio de direcciones, los parámetros se pasan de la forma usual. De manera similar, el procedimiento servidor es llamado por un procedimiento en su espacio de direcciones con los parámetros que esperaba. Para el procedimiento servidor, nada es inusual. De esta forma, en lugar de que la E/S se realice en *sockets*, la comunicación de red se realiza simulando una llamada a procedimiento normal.

A pesar de la elegancia conceptual de RPC, hay algunas desventajas ocultas. La más grande es el uso de parámetros de apuntador. Por lo general, pasar un apuntador a un procedimiento no es un problema. El procedimiento invocado puede utilizar el apuntador de la misma manera que el invocador, porque ambos procedimientos habitan el mismo espacio de direcciones virtual. Con

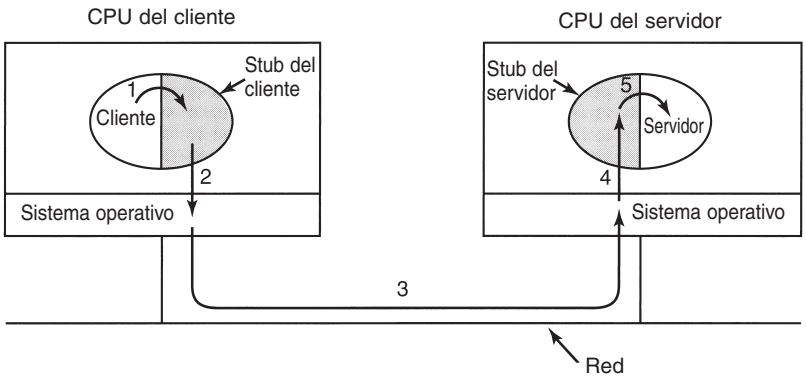


Figura 6-24. Pasos para realizar una llamada a procedimiento remoto. Los stubs están sombreados.

RPC, el paso de apuntadores es imposible porque el cliente y el servidor están en diferentes espacios de direcciones.

En algunos casos, se pueden utilizar trucos para hacer que el paso de apuntadores sea posible. Suponga que el primer parámetro es un apuntador a un entero, *k*. El *stub* del cliente puede clasificar *k* y enviarlo a lo largo del servidor. El *stub* del servidor a continuación crea un apuntador a *k* y lo pasa al procedimiento servidor, justamente como lo espera. Cuando el procedimiento servidor regresa el control al *stub* del servidor, este último regresa *k* al cliente, donde *k* se copia en el anterior, por si el servidor cambió. En efecto, la secuencia de llamada estándar de llamada por referencia se ha reemplazado con copiar-restaurar. Desgraciadamente, este truco no siempre funciona, por ejemplo, si el apuntador apunta hacia un grafo o a otra estructura de datos compleja. Por esta razón, se deben colocar algunas restricciones para los procedimientos llamados de manera remota.

Un segundo problema es que en los lenguajes de tipos flexibles, como C, es perfectamente legal escribir un procedimiento que calcule el producto interno de dos vectores (arreglos), sin especificar la longitud de cada uno. Éstos pueden terminarse mediante un valor especial conocido sólo por los procedimientos invocador e invocado. Bajo estas circunstancias, es esencialmente imposible que el *stub* del cliente aplique *marshaling* a los parámetros debido a que no tiene forma de determinar su longitud.

Un tercer problema es que no siempre es posible deducir los tipos de parámetros, ni siquiera de una especificación formal o el código mismo. Un ejemplo de esto es *printf*, el cual puede tener cualquier número de parámetros (por lo menos uno), y éstos pueden ser una mezcla arbitraria de tipos enteros, cortos, largos, caracteres, cadenas, así como de números de punto flotante de varias longitudes, entre otros. Tratar de llamar a *printf* como un procedimiento remoto sería prácticamente imposible debido a que C es demasiado permisivo. Sin embargo, una regla que especifique el uso de RPC siempre y cuando no se utilice C (o C++) para programar tal vez no sea muy popular.

Un cuarto problema se relaciona con el uso de las variables globales. Por lo general, los procedimientos invocador e invocado pueden comunicarse utilizando variables globales y parámetros. Si el procedimiento invocado se mueve a una máquina remota, el código fallará porque las variables globales ya no se pueden compartir.

Estos problemas no significan que RPC no tiene mucho futuro. De hecho, se utiliza ampliamente, pero se necesitan algunas restricciones para hacerlo funcionar bien en la práctica.

Por supuesto, RPC no necesita utilizar paquetes UDP, pero RPC y UDP son una buena combinación y UDP se utiliza comúnmente con RPC. No obstante, cuando los parámetros o resultados son más grandes que el tamaño máximo del paquete UDP o cuando la operación solicitada no tiene la misma potencia (es decir, no se puede repetir de forma segura, como cuando se incrementa un contador), tal vez sea necesario establecer una conexión TCP y enviar una solicitud a través de ella en lugar de utilizar UDP.

6.4.3 El protocolo de transporte en tiempo real

El RPC cliente-servidor es un área en la que se utiliza ampliamente UDP. Otra son las aplicaciones multimedia en tiempo real. En particular, conforme el radio en Internet, la telefonía en Internet, la música bajo demanda, la videoconferencia, el vídeo bajo demanda y otras aplicaciones multimedia se volvían más comunes, las personas que descubrieron cada una de esas aplicaciones estaba reinventando más o menos el mismo protocolo de transporte de tiempo-real. Gradualmente se volvió más claro que tener un protocolo de transporte genérico para múltiples aplicaciones sería una excelente idea. Y fue por esto que nació el **RTP** (**Protocolo de Transporte en Tiempo Real**). Se describe en el RFC 1889 y ahora se utiliza ampliamente.

La posición del RTP en la pila de protocolos es algo extraña. Se decidió colocarlo en el espacio de usuario y ejecutarlo (por lo general) sobre UDP. Opera como se muestra a continuación. La aplicación multimedia consiste en múltiples flujos de audio, vídeo, texto, entre otros. Éstos se colocan en la biblioteca RTP, la cual está en el espacio de usuario junto con la aplicación. Esta biblioteca multiplexa los flujos y los codifica en paquetes RTP, que después coloca en un *socket*. En el otro extremo del *socket* (en el *kernel* del sistema operativo), los paquetes UDP se generan e incrustan en paquetes IP. Si la computadora está en una Ethernet, los paquetes IP se colocan a continuación en tramas Ethernet para su transmisión. En la figura 6-25(a) se muestra la pila de protocolos para esta situación, y en la 6-25(b) se muestra el anidamiento de paquetes.

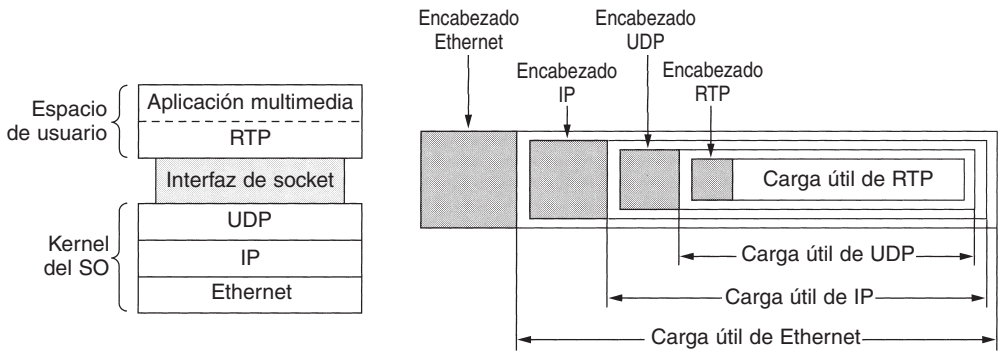


Figura 6-25. (a) La posición del RTP en la pila de protocolos. (b) Anidamiento de paquetes.

Como consecuencia de este diseño, es un poco difícil decir en cuál capa está RTP. Debido a que se ejecuta en el espacio de usuario y a que está enlazado al programa de aplicación, ciertamente luce como un protocolo de aplicación. Por otro lado, es un protocolo genérico, independiente de la aplicación que simplemente proporciona características de transporte, por lo que se parece a un protocolo de transporte. Probablemente la mejor descripción es que es un protocolo de transporte que está implementado en la capa de aplicación.

La función básica de RTP es multiplexar varios flujos de datos en tiempo real en un solo flujo de paquetes UDP. El flujo UDP se puede enviar a un solo destino (unidifusión) o a múltiples destinos (multidifusión). Debido a que RTP sólo utiliza UDP normal, sus paquetes no son tratados de manera especial por los enrutadores, a menos que se habiliten algunas características de calidad de servicio IP normales. En particular, no hay garantías especiales acerca de la entrega, fluctuación, etcétera.

A cada paquete enviado en un flujo RTP se le da un número más grande que a su predecesor. Esta numeración permite al destino determinar si falta algún paquete. Si falta alguno, la mejor acción que el destino puede realizar es aproximar el valor faltante mediante la interpolación. La retransmisión no es una opción práctica debido a que el paquete retransmitido probablemente llegará muy tarde como para ser útil. En consecuencia, RTP no tiene control de flujo, control de errores, confirmaciones de recepción ni ningún mecanismo para solicitar retransmisiones.

Cada carga útil RTP podría contener múltiples muestras, y éstas podrían codificarse de la forma en la que la aplicación desee. Para permitir la interconectividad, RTP define diversos perfiles (por ejemplo, un solo flujo de audio), y para cada perfil se podrían permitir múltiples formatos de codificación. Por ejemplo, un solo flujo de audio podría codificarse como muestras de PCM de 8 bits a 8 kHz, codificación delta, codificación predictiva, codificación GSM, MP3, etcétera. RTP proporciona un campo de encabezado en el que el origen puede especificar la codificación, pero por otro lado no tiene nada que ver en la forma en que se realiza la codificación.

Otra característica que muchas de las aplicaciones en tiempo real necesitan es la marcación del tiempo (*timestamping*). La idea aquí es permitir que el origen asocie una marca de tiempo con la primera muestra de cada paquete. Las marcas de tiempo son relativas al inicio del flujo, por lo que sólo son importantes las diferencias entre dichas marcas de tiempo. Los valores absolutos no tienen significado. Este mecanismo permite que el destino realice una pequeña cantidad de almacenamiento en búfer y reproduzca cada muestra el número exacto de milisegundos después del inicio del flujo, independientemente de cuándo llegó el paquete que contiene la muestra. La marcación del tiempo no sólo reduce los efectos de la fluctuación, sino que también permite que múltiples flujos estén sincronizados entre sí. Por ejemplo, un programa de televisión digital podría tener un flujo de vídeo y dos flujos de audio. Estos flujos de audio podrían ser para difusiones de estéreo o para manejar películas con la banda sonora del idioma original y con una doblada al idioma local, dándole una opción al espectador. Cada flujo proviene de un dispositivo físico diferente, pero si tienen marcas de tiempo de un solo contador, pueden reproducirse de manera síncrona, incluso si los flujos se transmiten de manera irregular.

En la figura 6-26 se ilustra el encabezado RTP. Consiste de tres palabras de 32 bits y de algunas extensiones. La primera palabra contiene el campo *Versión*, que es la 2. Esperemos que esta versión esté muy cerca de la final debido a que sólo quedó pendiente un punto del código

(aunque se puede definir como 3 dando a entender que la versión real estaba en una palabra de extensión).

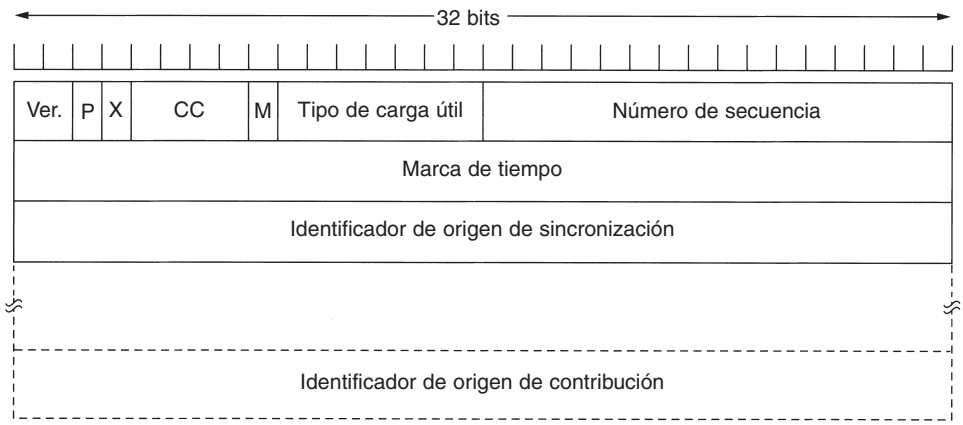


Figura 6-26. El encabezado RTP.

El bit *P* indica que el paquete se ha rellenado a un múltiplo de 4 bytes. El último byte de relleno indica cuántos bytes se agregaron. El bit *X* indica que hay un encabezado de extensión. El formato y el significado de este encabezado no se definen. Lo único que se define es que la primera palabra de la extensión proporciona la longitud. Ésta es una puerta de escape para cualquier requerimiento imprevisto.

El campo *CC* indica cuántos orígenes de contribución están presentes, de 0 a 15 (vea más adelante). El bit *M* es un bit marcador específico de la aplicación. Puede utilizarse para marcar el inicio de un cuadro de vídeo, el inicio de una palabra en un canal de audio, o algo más que la aplicación entienda. El campo *Tipo de carga útil* indica cuál algoritmo de codificación se ha utilizado (por ejemplo, audio de 8 bits sin compresión, MP3, etcétera). Puesto que cada paquete lleva este campo, la codificación puede cambiar durante la transmisión. El *Número de secuencia* es simplemente un contador que se incrementa en cada paquete RTP enviado. Se utiliza para detectar paquetes perdidos.

El origen del flujo produce la marca de tiempo para indicar cuándo se creó la primera muestra en el paquete. Este valor puede ayudar a reducir la fluctuación en el receptor al desacoplar la reproducción del tiempo de llegada del paquete. El *Identificador de origen de sincronización* indica a cuál flujo pertenece el paquete. Es el método utilizado para multiplexar y desmultiplexar varios flujos de datos en un solo flujo de paquetes UDP. Por último, los *Identificadores de origen de contribución*, en caso de que haya, se utilizan cuando los mezcladores están presentes en el estudio. En ese caso, el mezclador es el origen de sincronización, y los flujos que se mezclan se listan aquí.

RTP tiene un hermano pequeño llamado **RTCP** (**Protocolo de Control de Transporte en Tiempo Real**). Maneja la retroalimentación, sincronización y la interfaz de usuario, pero no transporta ningún tipo de datos. La primera función se puede utilizar para proporcionar a los orígenes

retroalimentación en caso de retardo, fluctuación, ancho de banda, congestión y otras propiedades de red. El proceso de codificación puede utilizar esta información para incrementar la tasa de datos (y para proporcionar mejor calidad) cuando la red está funcionando bien y para disminuir la tasa de datos cuando hay problemas en la red. Al proporcionar retroalimentación continua, los algoritmos de codificación se pueden adaptar continuamente para proporcionar la mejor calidad posible bajo las circunstancias actuales. Por ejemplo, si el ancho de banda crece o decrece durante la transmisión, la codificación puede cambiar de MP3 a PCM de 8 bits a codificación delta conforme se requiera. El campo *Tipo de carga útil* se utiliza para indicar al destino cuál algoritmo de codificación se emplea en el paquete actual, lo que hace posible modificarlo a solicitud.

RTCP también maneja sincronización entre flujos. El problema es que flujos diferentes pueden utilizar relojes diferentes, con distintas granularidades y distintas tasas de derivación. RTCP puede utilizarse para mantenerlos sincronizados.

Por último, RTCP proporciona una forma para nombrar los diversos orígenes (por ejemplo, en texto ASCII). Esta información puede desplegarse en la pantalla del receptor para indicar quién está hablando en ese momento.

Podrá encontrar más información en (Perkins, 2002).

6.5 LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP

UDP es un protocolo simple y tiene algunos usos específicos, como interacciones cliente-servidor y multimedia, pero para la mayoría de las aplicaciones de Internet se necesita una entrega en secuencia confiable. UDP no puede proporcionar esto, por lo que se necesita otro protocolo. Se llama TCP y es el más utilizado en Internet. A continuación lo estudiaremos con detalle.

6.5.1 Introducción a TCP

TCP (**Protocolo de Control de Transmisión**) se diseño específicamente para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable. Una interred difiere de una sola red debido a que diversas partes podrían tener diferentes topologías, anchos de banda, retardos, tamaños de paquete y otros parámetros. TCP tiene un diseño que se adapta de manera dinámica a las propiedades de la interred y que se sobrepone a muchos tipos de fallas.

TCP se definió formalmente en el RFC 793. Conforme el tiempo pasó, se detectaron varios errores e inconsistencias, y los requerimientos de algunas áreas cambiaron. En el RFC 1122 se detallan estas especificaciones y algunos arreglos de errores. En el RFC 1323 se dan algunas extensiones.

Cada máquina que soporta TCP tiene una entidad de transporte TCP, ya sea un procedimiento de biblioteca, un proceso de usuario o parte del *kernel*. En todos los casos, maneja flujos TCP e interactúa con la capa IP. Una entidad TCP acepta flujos de datos de usuario de procesos locales, los divide en fragmentos que no excedan los 64 KB (en la práctica, por lo general, 1460 bytes de datos que se ajustan en una sola trama Ethernet con los encabezados IP y TCP), y envía cada fragmento como un datagrama IP independiente. Cuando los datagramas que contienen datos TCP

llegan a una máquina, se pasan a la entidad TCP, la cual reconstruye los flujos de bytes originales. Por simplicidad, algunas veces utilizaremos "TCP" para referirnos a la entidad de transporte TCP (una pieza de software) o al protocolo TCP (un conjunto de reglas). El contexto dejará claro a que nos referimos. Por ejemplo, en la frase "El usuario proporciona los datos a TCP", es claro que nos referimos a la entidad de transporte TCP.

La capa IP no proporciona ninguna garantía de que los datagramas se entregarán de manera apropiada, por lo que corresponde a TCP terminar los temporizadores y retransmitir los datagramas conforme sea necesario. Los datagramas que llegan podrían hacerlo en el orden incorrecto; también corresponde a TCP reensamblarlos en mensajes en la secuencia apropiada. En resumen, TCP debe proporcionar la confiabilidad que la mayoría de los usuarios desean y que IP no proporciona.

6.5.2 El modelo del servicio TCP

El servicio TCP se obtiene al hacer que tanto el servidor como el cliente creen puntos terminales, llamados *sockets*, como se mencionó en la sección 6.1.3. Cada *socket* tiene un número (dirección), que consiste en la dirección IP del *host,* y un número de 16 bits, que es local a ese *host*, llamado **puerto.** Un puerto es el nombre TCP para un TSAP. Para obtener el servicio TCP, se debe establecer de manera explícita una conexión entre un *socket* en la máquina emisora y uno en la máquina receptora. Las llamadas de *socket* se listan en la figura 6-5.

Un *socket* puede utilizarse para múltiples conexiones al mismo tiempo. En otras palabras, dos o más conexiones pueden terminar en el mismo *socket*. Las conexiones se identifican mediante los identificadores de *socket* de los dos extremos, es decir (*socket1*, *socket2*). No se utiliza ningún otro número de circuitos virtuales ni identificador.

Los números de puerto menores que 1024 se llaman **puertos bien conocidos** y se reservan para servicios estándar. Por ejemplo, cualquier proceso que desee establecer una conexión a un *host* para transferir un archivo utilizando FTP puede conectarse con el puerto 21 del *host* de destino para conectar su demonio (*daemon*) FTP. La lista de puertos bien conocidos se proporciona en *www.iana.org*. Se han asignado aproximadamente 300. En la figura 6-27 se listan algunos de los más conocidos.

Ciertamente podría ser posible que el demonio FTP se conecte a sí mismo al puerto 21 en tiempo de arranque, que el demonio telnet se conecte a sí mismo al puerto 23 en tiempo de arranque, y así sucesivamente. Sin embargo, hacer lo anterior podría llenar la memoria con demonios que están inactivos la mayor parte del tiempo. En su lugar, lo que se hace generalmente es que un solo demonio, llamado **inetd** (**demonio de Internet**) en UNIX, se conecte a sí mismo a múltiples puertos y esperar la primera conexión entrante. Cuando eso ocurre, inetd bifurca un nuevo proceso y ejecuta el demonio apropiado en él, dejando que ese demonio maneje la solicitud. De esta forma, los demonios distintos a inetd sólo están activos cuando hay trabajo para ellos. Inetd consulta un archivo de configuración para saber cuál puerto utilizar. En consecuencia, el administrador del sistema puede configurar el sistema para tener demonios permanentes en los puertos más ocupados (por ejemplo, el puerto 80) e inetd en los demás.

Puerto	Protocolo	Uso
21	FTP	Transferencia de archivos
23	Telnet	Inicio remoto de sesión
25	SMTP	Correo electrónico
69	TFTP	Protocolo de transferencia de archivos trivial
79	Finger	Búsqueda de información sobre un usuario
80	HTTP	World Wide Web
110	POP-3	Acceso remoto al correo electrónico
119	NNTP	Noticias USENET

Figura 6-27. Algunos puertos asignados.

Todas las conexiones TCP son de dúplex total y de punto a punto. Dúplex total significa que el tráfico puede ir en ambas direcciones al mismo tiempo. Punto a punto significa que cada conexión tiene exactamente dos puntos finales. TCP no soporta la multidifusión ni la difusión.

Una conexión TCP es un flujo de bytes, no uno de mensajes. Los límites de los mensajes no se preservan de extremo a extremo. Por ejemplo, si el proceso emisor realiza cuatro escrituras de 512 bytes en un flujo TCP, tal vez estos datos se entreguen al proceso receptor como cuatro fragmentos de 512 bytes, dos fragmentos de 1024 bytes, uno de 2048 bytes (vea la figura 6-28), o de alguna otra forma. No hay manera de que el receptor detecte la(s) unidad(es) en la(s) que se escribieron los datos.

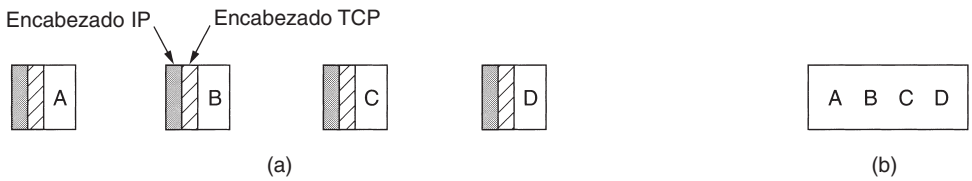


Figura 6-28. (a) Cuatro segmentos de 512 bytes enviados como datagramas IP independientes. (b) Los 2048 bytes de datos que se entregan a la aplicación en una sola llamada a READ.

Los archivos de UNIX también tienen esta propiedad. El lector de un archivo no puede indicar si éste escribió un bloque a la vez, un byte a la vez o todos al mismo tiempo. Al igual que con un archivo de UNIX, el software TCP no tiene idea de lo que significan los bytes y no le interesa averiguarlo. Un byte es sólo un byte.

Cuando una aplicación pasa datos a TCP, éste decide si los envía inmediatamente o los almacena en el búfer (a fin de recolectar una gran cantidad y, así, enviarlos al mismo tiempo). Sin embargo, algunas veces, la aplicación realmente necesita que los datos se envíen de inmediato. Por ejemplo, suponga que un usuario inicia una sesión en una máquina remota. Una vez que se termina una línea de comandos y que se introduce un retorno de carro, es esencial que la línea se envíe

a la máquina remota inmediatamente y que no se almacene en el búfer hasta que llegue la siguiente línea. Para obtener los datos, las aplicaciones pueden utilizar el indicador PUSH, que es una señal para TCP de que no debe retrasar la transmisión.

Algunas de las primeras aplicaciones utilizaban el indicador PUSH como un tipo de marcador para delinear los límites de los mensajes. Si bien este truco funcionaba algunas veces, otras fallaba debido a que no todas las implementaciones de TCP pasan el indicador PUSH a la aplicación del receptor. Además, si llegan indicadores PUSH antes de que el primero se haya transmitido (por ejemplo, debido a que la línea de salida está ocupada), TCP es libre de recolectar todos los datos con indicadores PUSH en un solo datagrama IP, sin ninguna separación entre las diversas piezas.

Una última característica del servicio TCP que vale la pena mencionar son los **datos urgentes.** Cuando un usuario interactivo oprime las teclas Supr o Ctrl+C para interrumpir una operación remota que ha iniciado, la aplicación emisora coloca información de control en el flujo de datos y se la da a TCP junto con el indicador URGENT. Este evento ocasiona que TCP interrumpa el encolamiento de datos y transmita inmediatamente todo lo que tenga para esa conexión.

Cuando el destino recibe los datos urgentes, se interrumpe la aplicación receptora (por ejemplo, se le da una señal en términos de UNIX), a fin de que pueda detener lo que esté haciendo y que lea el flujo de datos para encontrar los datos urgentes. El final de los datos urgentes se marca para que la aplicación sepa cuándo terminan. El inicio de éstos no se marca; la aplicación tiene que averiguarlo. Este esquema proporciona básicamente un mecanismo de señalización simple y deja todo lo demás a la aplicación.

6.5.3 El protocolo TCP

En esta sección daremos un repaso general del protocolo TCP; en la siguiente veremos el encabezado del protocolo, campo por campo.

Una característica clave de TCP, y una que domina el diseño del protocolo, es que cada byte de una conexión TCP tiene su propio número de secuencia de 32 bits. Cuando Internet comenzó, las líneas entre los enrutadores eran principalmente líneas alquiladas de 56 kbps, por lo que un *host* que mandaba datos a toda velocidad tardaba una semana en recorrer los números de secuencia. A las velocidades de las redes modernas, los números de secuencia pueden consumirse con una rapidez alarmante, como veremos más adelante. Los números de secuencia separados de 32 bits se utilizan para confirmaciones de recepción y para el mecanismo de ventana, como se analiza a continuación.

La entidad TCP emisora y la receptora intercambian datos en forma de segmentos. Un **segmento** consiste en un encabezado TCP fijo de 20 bytes (más una parte opcional) seguido de cero o más bytes de datos. El *software* de TCP decide el tamaño de los segmentos; puede acumular datos de varias escrituras para formar un segmento, o dividir los datos de una escritura en varios segmentos. Hay dos límites que restringen el tamaño de segmento. Primero, cada segmento, incluido el encabezado TCP, debe caber en la carga útil de 65,515 bytes del IP. Segundo, cada red tiene una **unidad máxima de transferencia** (**MTU**) y cada segmento debe caber en la MTU. En la práctica, la MTU es, generalmente, de 1500 bytes (el tamaño de la carga útil en Ethernet) y, por tanto, define el límite superior del tamaño de segmento.

El protocolo básico usado por las entidades TCP es el protocolo de ventana corrediza. Cuando un transmisor envía un segmento, también inicia un temporizador. Cuando llega el segmento al destino, la entidad TCP receptora devuelve un segmento (con datos, si existen, de otro modo sin ellos) que contiene un número de confirmación de recepción igual al siguiente número de secuencia que espera recibir. Si el temporizador del emisor expira antes de la recepción de la confirmación, el emisor envía de nuevo el segmento.

Aunque este protocolo suena sencillo, tiene muchos vericuetos que explicaremos a continuación. Por ejemplo, pueden llegar segmentos fuera de orden, por lo que los bytes 3072–4095 podrían llegar pero no enviarse confirmación de recepción porque los bytes 2048–3071 no han aparecido aún. También pueden retardarse segmentos en tránsito durante tanto tiempo que el temporizador del emisor expira y los segmentos se retransmiten. Las retransmisiones podrían incluir rangos de bytes diferentes a los de la transmisión original, lo cual requiere una administración cuidadosa para llevar el control de los bytes que se han recibido correctamente en un momento determinado. Sin embargo, esto es factible ya que cada byte del flujo tiene su propio desplazamiento único.

El TCP debe estar preparado para manejar y resolver estos problemas de una manera eficiente. Se ha invertido una cantidad considerable de esfuerzo en la optimización del desempeño de los flujos TCP, incluso ante problemas de red. A continuación se estudiarán varios de los algoritmos usados por muchas implementaciones de TCP.

6.5.4 El encabezado del segmento TCP

En la figura 6-29 se muestra la distribución de un segmento TCP. Cada segmento comienza con un encabezado de formato fijo de 20 bytes. El encabezado fijo puede ir seguido de opciones de encabezado. Tras las opciones, si las hay, pueden continuar hasta 65,535 − 20 − 20 = 65,495 bytes de datos, donde los primeros 20 se refieren al encabezado IP y los segundos al encabezado TCP. Los segmentos sin datos son legales y se usan por lo común para confirmaciones de recepción y mensajes de control.

Realicemos la disección del encabezado TCP campo por campo. Los campos de *Puerto de origen* y *Puerto de destino* identifican los puntos terminales locales de la conexión. Los puertos bien conocidos se especifican en *www.iana.org* pero cada *host* puede asignar los demás según sea necesario. La dirección de un puerto más la dirección IP de su *host* forman un punto terminal único de 48 bits. Los puntos terminales de origen y de destino en conjunto identifican la conexión.

Los campos de *Número de secuencia* y *Número de confirmación de recepción* desempeñan sus funciones normales. Nótese que el segundo especifica el siguiente byte esperado, no el último byte correctamente recibido. Ambos tienen 32 bits de longitud porque cada byte de datos está numerado en un flujo TCP.

La *Longitud del encabezado TCP* indica la cantidad de palabras de 32 bits contenidas en el encabezado TCP. Esta información es necesaria porque el campo de *Opciones* es de longitud variable, por lo que el encabezado también. Técnicamente, este campo en realidad indica el comienzo

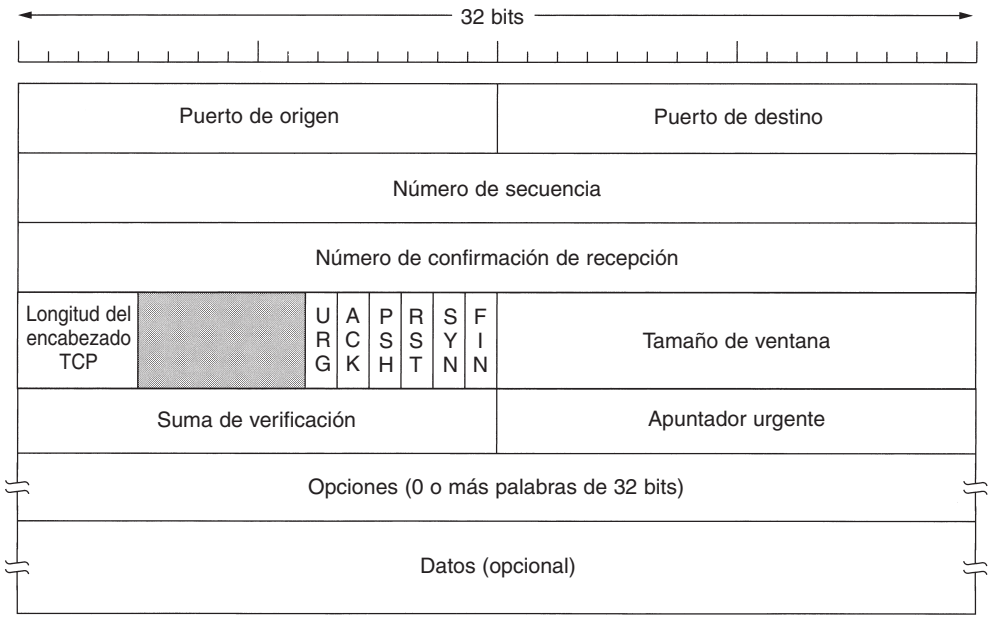


Figura 6-29. Encabezado TCP.

de los datos en el segmento, medido en palabras de 32 bits, pero ese número es simplemente la longitud del encabezado en palabras, por lo que el efecto es el mismo.

A continuación viene un campo de 6 bits que no se usa. El que este campo haya sobrevivido intacto durante más de una década es testimonio de lo bien pensado que está el TCP. Protocolos inferiores lo habrían necesitado para corregir errores del diseño original.

Ahora vienen seis indicadores de 1 bit. *URG* se establece en 1 se está en uso el *apuntador urgente.* El *apuntador urgente* sirve para indicar un desplazamiento en bytes a partir del número actual de secuencia en el que se encuentran datos urgentes. Este recurso sustituye los mensajes de interrupción. Como se mencionó antes, este recurso es un mecanismo rudimentario para permitir que el emisor envíe una señal al receptor sin implicar al TCP en la razón de la interrupción.

El bit *ACK* se establece en 1 para indicar que el *Número de confirmación de recepción* es válido. Si el *ACK* es 0, el segmento no contiene una confirmación de recepción, por lo que se ignora el campo de *Número de confirmación de recepción*.

El bit *PSH* indica datos que se deben transmitir de inmediato. Por este medio se solicita atentamente al receptor que entregue los datos a la aplicación a su llegada y no los almacene en búfer hasta la recepción de un búfer completo (lo que podría hacer en otras circunstancias por razones de eficiencia).

El bit *RST* se usa para restablecer una conexión que se ha confundido debido a una caída de *host* u otra razón; también sirve para rechazar un segmento no válido o un intento de abrir una conexión. Por lo general, si usted recibe un segmento con el bit *RST* encendido, tiene un problema entre manos.

El bit *SYN* se usa para establecer conexiones. La solicitud de conexión tiene *SYN* = 1 y *ACK* = 0 para indicar que el campo de confirmación de recepción incorporado no está en uso. La respuesta de conexión sí lleva una confirmación de recepción, por lo que tiene *SYN* = 1 y *ACK* = 1. En esencia, el bit *SYN* se usa para denotar CONNECTION REQUEST y CONNECTION ACCEPTED, y el bit *ACK* sirve para distinguir entre ambas posibilidades.

El bit *FIN* se usa para liberar una conexión; especifica que el emisor no tiene más datos que transmitir. Sin embargo, tras cerrar una conexión, un proceso puede continuar recibiendo datos indefinidamente. Ambos segmentos, *SYN* y *FIN*, tienen números de secuencia y, por tanto, tienen la garantía de procesarse en el orden correcto.

El control de flujo en el TCP se maneja usando una ventana corrediza de tamaño variable. El campo *Tamaño de ventana* indica la cantidad de bytes que pueden enviarse comenzando por el byte cuya recepción se ha confirmado. Es válido un campo de *Tamaño de ventana* de 0, e indica que se han recibido los bytes hasta *Número de confirmación de recepción* − 1, inclusive, pero que el receptor actualmente necesita un descanso y quisiera no recibir más datos por el momento, gracias. El permiso para enviar puede otorgarse después enviando un segmento con el mismo *Número de confirmación de recepción* y un campo *Tamaño de ventana* distinto de cero.

En los protocolos del capítulo 3, las confirmaciones de recepción de las tramas recibidas y los permisos para enviar nuevas tramas estaban enlazados. Ésta fue una consecuencia de un tamaño de ventana fijo para cada protocolo. En TCP, las confirmaciones de recepción y los permisos para enviar datos adicionales son completamente independientes. En efecto, un receptor puede decir: "He recibido bytes hasta *k*, pero por ahora no deseo más". Esta independencia (de hecho, una ventana de tamaño variable) da flexibilidad adicional. Más adelante lo estudiaremos con más detalle.

También se proporciona una *Suma de verificación* para agregar confiabilidad. Es una suma de verificación del encabezado, los datos y el pseudoencabezado conceptual mostrado en la figura 6-30. Al realizar este cálculo, se establece el campo de *Suma de verificación* del TCP en cero, y se rellena el campo de datos con un byte cero adicional si la longitud es un número impar. El algoritmo de suma de verificación simplemente suma todas las palabras de 16 bits en complemento a 1 y luego obtiene el complemento a 1 de la suma. Como consecuencia, al realizar el cálculo el receptor con el segmento completo, incluido el campo de *Suma de verificación,* el resultado debe ser 0.

El pseudoencabezado contiene las direcciones IP de 32 bits de las máquinas de origen y de destino, el número de protocolo de TCP (6), y la cuenta de bytes del segmento TCP (incluido el encabezado). La inclusión del pseudoencabezado en el cálculo de la suma de verificación TCP ayuda a detectar paquetes mal entregados, pero hacerlo viola la jerarquía de protocolos puesto que las direcciones de IP que contiene pertenecen a la capa IP, no a la capa TCP. UDP utiliza el mismo pseudoencabezado para su suma de verificación.

El campo *Opciones* ofrece una forma de agregar características extra no cubiertas por el encabezado normal. La opción más importante es la que permite que cada *host* especifique la carga útil TCP máxima que está dispuesto a aceptar. El uso de segmentos grandes es más eficiente que el de segmentos pequeños, puesto que el encabezado de 20 bytes puede entonces amortizarse entre más datos, pero los *hosts* pequeños tal vez no puedan manejar segmentos muy grandes. Durante el

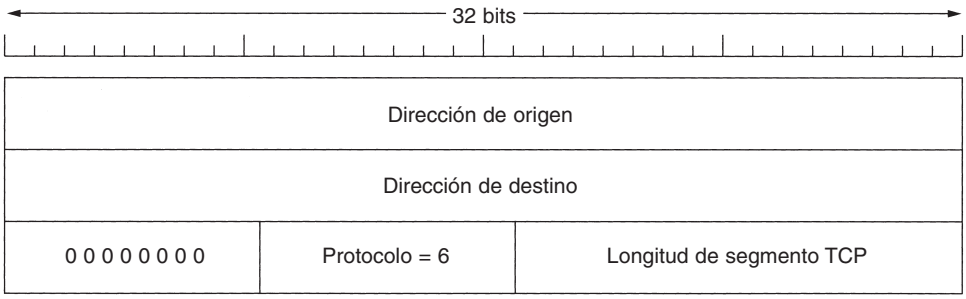


Figura 6-30. Pseudoencabezado incluido en la suma de verificación del TCP.

establecimiento de la conexión, cada lado puede anunciar su máximo y ver el de su compañero. Si un *host* no usa esta opción, tiene una carga útil predeterminada de 536 bytes. Se requiere que todos los *hosts* de Internet acepten segmentos TCP de 536 + 20 = 556 bytes. No es necesario que el tamaño máximo de segmento en ambas direcciones sea el mismo.

En las líneas con alto ancho de banda, alto retardo o ambas cosas, la ventana de 64 KB con frecuencia es un problema. En una línea T3 (44.736 Mbps) se requieren sólo 12 mseg para enviar una ventana completa de 64 KB. Si el retardo de propagación de ida y vuelta es de 50 mseg (típico de una fibra transcontinental), el emisor estará inactivo 3/4 del tiempo en espera de confirmaciones de recepción. En una conexión satelital la situación es peor aún. Un tamaño de ventana más grande permitirá al emisor continuar enviando datos, pero como el campo de *Tamaño de ventana* es de 16 bits, es imposible expresar tal tamaño. En el RFC 1323 se propuso una opción de *escala de ventana* que permite al emisor y al receptor negociar un factor de escala de ventana. Este número da la posibilidad de que ambos lados desplacen el campo de *Tamaño de ventana* hasta 14 bits a la izquierda, permitiendo por tanto ventanas de hasta 2^{30} bytes. La mayoría de las implementaciones actuales de TCP manejan esta opción.

Otra opción propuesta en el RFC 1106 y ahora de uso difundido es el empleo de la repetición selectiva en lugar del protocolo de retroceso *n*. Si el receptor recibe un segmento malo y luego una gran cantidad de segmentos buenos, el temporizador del protocolo TCP normal expirará en algún momento y se retransmitirán todos los segmentos sin confirmación de recepción, incluidos los que se recibieron correctamente. El RFC 1106 introdujo los NAKs, para permitir que el receptor solicite un segmento (o segmentos) específico. Tras recibirlo, puede enviar una confirmación de recepción de todos los datos que tiene en búfer, reduciendo de esta manera la cantidad de datos retransmitidos.

6.5.5 Establecimiento de una conexión TCP

En el TCP las conexiones se establecen usando el acuerdo de tres vías estudiado en la sección 6.2.2. Para establecer una conexión, un lado, digamos el servidor, espera pasivamente una conexión entrante ejecutando las primitivas LISTEN y ACCEPT y especificando cierto origen o bien nadie en particular.

El otro lado, digamos el cliente, ejecuta una primitiva CONNECT especificando la dirección y el puerto IP con el que se desea conectar, el tamaño máximo de segmento TCP que está dispuesto a aceptar y opcionalmente algunos datos de usuario (por ejemplo, una contraseña). La primitiva CONNECT envía un segmento TCP con el bit *SYN* encendido y el bit *ACK* apagado, y espera una respuesta.

Al llegar el segmento al destino, la entidad TCP ahí revisa si hay un proceso que haya ejecutado un LISTEN en el puerto indicado en el campo de *Puerto de destino*. Si no lo hay, envía una respuesta con el bit *RST* encendido para rechazar la conexión.

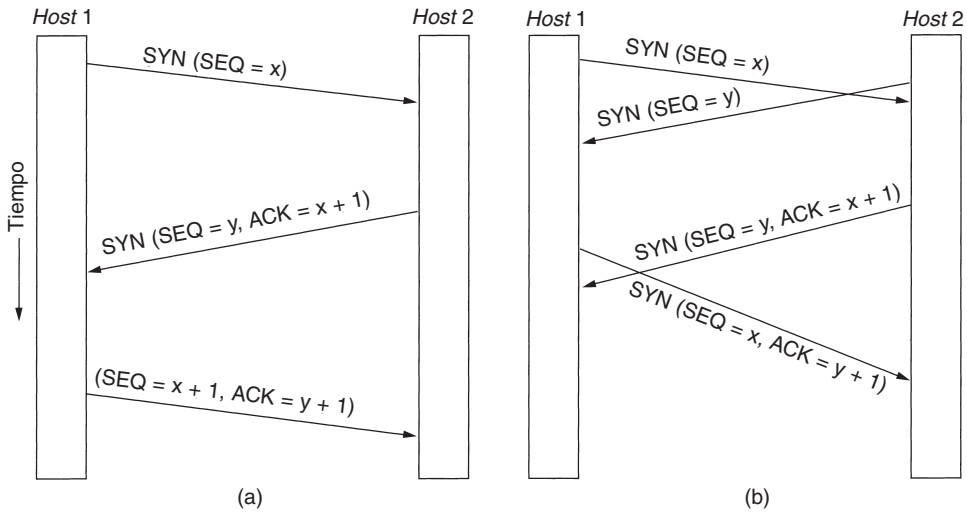


Figura 6-31. (a) Establecimiento de una conexión TCP en el caso normal. (b) Colisión de llamadas.

Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión; si la acepta, se devuelve un segmento de confirmación de recepción. La secuencia de segmentos TCP enviados en el caso normal se muestra en la figura 6-31(a). Nótese que un segmento *SYN* consume 1 byte de espacio de secuencia, por lo que puede reconocerse sin ambigüedades.

En el caso en que dos *hosts* intentan simultáneamente establecer una conexión entre los mismos dos *sockets*, la secuencia de eventos es la que se ilustra en la figura 6-31(b). El resultado de estos eventos es que sólo se establece una conexión, no dos, pues las conexiones se identifican por sus puntos terminales. Si el primer establecimiento resulta en una conexión identificada por (*x, y*), al igual que en el segundo, sólo se hace una entrada de tabla, es decir, de (*x, y*).

El número de secuencia inicial de una conexión no es 0 por las razones que señalamos antes. Se usa un esquema basado en reloj, con un pulso de reloj cada 4 μseg. Por seguridad adicional, al caerse un *host,* no podrá reiniciarse durante el tiempo máximo de paquete (120 seg) para asegurar que no haya paquetes de conexiones previas vagando por Internet.

6.5.6 Liberación de una conexión TCP

Aunque las conexiones TCP son dúplex total, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones símplex. Cada conexión símplex se libera independientemente de su igual. Para liberar una conexión, cualquiera de las partes puede enviar un segmento TCP con el bit *FIN* establecido, lo que significa que no tiene más datos por transmitir. Al confirmarse la recepción del *FIN*, ese sentido se apaga. Sin embargo, puede continuar un flujo de datos indefinido en el otro sentido. Cuando ambos sentidos se han apagado, se libera la conexión. Normalmente se requieren cuatro segmentos TCP para liberar una conexión, un *FIN* y un *ACK* para cada sentido. Sin embargo, es posible que el primer *ACK* y el segundo *FIN* estén contenidos en el mismo segmento, reduciendo la cuenta total a tres.

Al igual que con las llamadas telefónicas en las que ambas partes dicen adiós y cuelgan el teléfono simultáneamente, ambos extremos de una conexión TCP pueden enviar segmentos *FIN* al mismo tiempo. La recepción de ambos se confirma de la manera normal, y se apaga la conexión. De hecho, en esencia no hay diferencia entre la liberación secuencial o simultánea por parte de los *hosts.*

Para evitar el problema de los dos ejércitos, se usan temporizadores. Si no llega una respuesta a un *FIN* en un máximo de dos tiempos de vida de paquete, el emisor del *FIN* libera la conexión. Tarde o temprano el otro lado notará que, al parecer, ya nadie lo está escuchando, y también expirará su temporizador. Aunque esta solución no es perfecta, dado el hecho de que teóricamente es imposible una solución perfecta tendremos que conformarnos con ella. En la práctica, pocas veces ocurren problemas.

6.5.7 Modelado de administración de conexiones TCP

Los pasos requeridos para establecer y liberar conexiones pueden representarse en una máquina de estados finitos con los 11 estados listados en la figura 6-32. En cada estado son legales ciertos eventos. Al ocurrir un evento legal, debe emprenderse alguna acción. Si ocurren otros eventos, se informa un error.

Cada conexión comienza en el estado *CLOSED* (cerrado) y deja ese estado cuando hace una apertura pasiva (LISTEN), o una apertura activa (CONNECT). Si el otro lado realiza la acción opuesta, se establece una conexión y el estado se vuelve *ESTABLISHED*. La liberación de la conexión puede iniciarse desde cualquiera de los dos lados. Al completarse, el estado regresa a *CLOSED.*

La máquina de estados finitos se muestra en la figura 6-28. El caso común de un cliente que se conecta activamente a un servidor pasivo se indica con líneas gruesas (continuas para el cliente, punteadas para el servidor). Las líneas delgadas son secuencia de eventos poco comunes. Cada línea de la figura 6-33 se marca mediante un par *evento/acción*. El evento puede ser una llamada de sistema iniciada por el usuario (CONNECT, LISTEN, SEND o CLOSE), la llegada de un segmento (*SYN, FIN, ACK* o *RST*) o, en un caso, una expiración de temporizador del doble del tiempo de vida máximo del paquete. La acción es el envío de un segmento de control (*SYN, FIN* o *RST*), o nada, indicado por —. Los comentarios aparecen entre paréntesis.

Estado	Descripción
CLOSED	No hay conexión activa ni pendiente
LISTEN	El servidor espera una llamada
SYN RCVD	Llegó solicitud de conexión; espera ACK
SYN SENT	La aplicación comenzó a abrir una conexión
ESTABLISHED	Estado normal de transferencia de datos
FIN WAIT 1	La aplicación dijo que ya terminó
FIN WAIT 2	El otro lado acordó liberar
TIMED WAIT	Espera que todos los paquetes mueran
CLOSING	Ambos lados intentaron cerrar simultáneamente
CLOSE WAIT	El otro lado inició una liberación
LAST ACK	Espera que todos los paquetes mueran

Figura 6-32. Estados usados en la máquina de estados finitos de administración de conexiones TCP.

El diagrama puede entenderse mejor siguiendo primero la trayectoria de un cliente (la línea continua gruesa) y luego la de un servidor (línea punteada gruesa). Al emitir una solicitud CONNECT una aplicación de la máquina cliente, la entidad TCP local crea un registro de conexión, lo marca para indicar que está en el estado *SYN SENT,* y envía un segmento *SYN*. Observe que muchas conexiones pueden estar abiertas (o en proceso de apertura) al mismo tiempo como parte de varias aplicaciones, por lo que el estado es por conexión y se asienta en el registro de conexiones. Al llegar el *SYN+ACK,* el TCP envía al *ACK* final del acuerdo de tres vías y se conmuta al estado *ESTABLISHED*. Ahora pueden enviarse y recibirse datos.

Al terminar una aplicación, ejecuta una primitiva CLOSE, que causa que la entidad TCP local envíe un segmento *FIN* y espere el *ACK* correspondiente (recuadro punteado rotulado "cierre activo"). Al llegar el *ACK*, se hace una transición al estado *FIN WAIT* 2, y ya está cerrado un sentido de la conexión. Cuando también cierra el otro lado, llega un *FIN*, para el cual se envía una confirmación de recepción. Ahora ambos lados están cerrados, pero el TCP espera un tiempo igual al tiempo de vida máximo del paquete para garantizar que todos los paquetes de la conexión han sido eliminados, como protección en caso de la pérdida de una confirmación de recepción. Al expirar el temporizador, el TCP borra el registro de la conexión.

Examinemos ahora la administración de la conexión desde el punto de vista del servidor. El servidor hace un LISTEN y se detiene a esperar la aparición de alguien. Al llegar un *SYN*, se envía una confirmación de recepción y el servidor pasa al estado *SYN RCVD*. Cuando llega la confirmación de recepción del *SYN* del servidor, el acuerdo de tres vías se ha completado y el servidor regresa al estado *ESTABLISHED.* Ahora puede ocurrir la transferencia de datos.

Cuando el cliente ha tenido suficiente, hace un CLOSE, que causa la llegada de un *FIN* al servidor (recuadro punteado rotulado "cierre pasivo"). Entonces se envía una señal al servidor. Cuando éste también hace un CLOSE, se envía un *FIN* al cliente. Al llegar la confirmación de recepción del cliente, el servidor libera la conexión y elimina el registro de conexión.

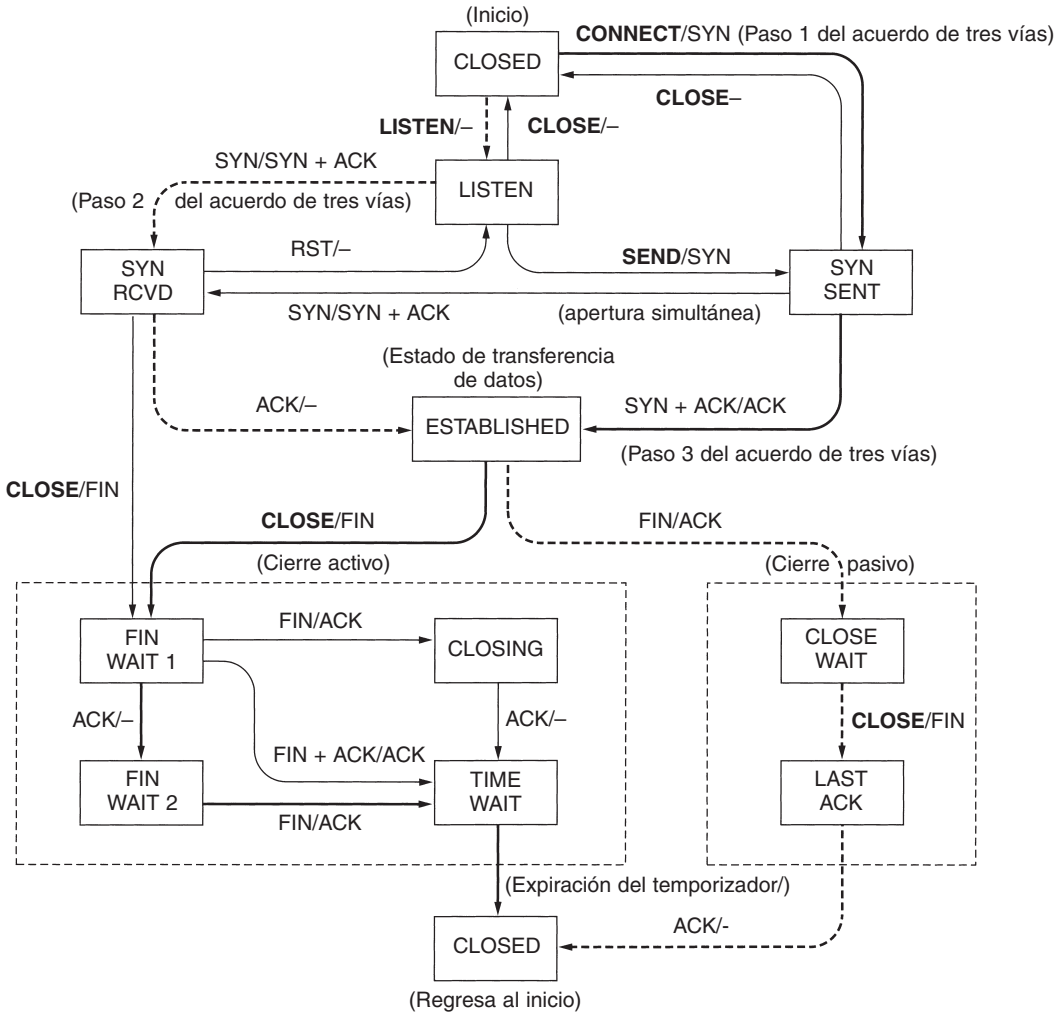


Figura 6-33. Máquina de estados finitos de administración de conexiones TCP. La línea continua gruesa es la trayectoria normal de un cliente. La línea punteada gruesa es la trayectoria normal de un servidor. Las líneas delgadas son eventos poco comunes. Cada transición está indicada por el evento que la ocasiona y la acción resultante, separada por una diagonal.

6.5.8 Política de transmisión del TCP

Como ya vimos, la administración de ventanas en el TCP no está vinculada directamente a las confirmaciones de recepción como en la mayoría de los protocolos de enlace de datos. Por ejemplo, suponga que el receptor tiene un búfer de 4096 bytes, como se muestra en la figura 6-34. Si el emisor envía un segmento de 2048 bytes que se recibe correctamente, el receptor enviará la confirmación de recepción del segmento. Sin embargo, dado que ahora sólo tiene 2048 bytes de

espacio de búfer (hasta que la aplicación retire algunos datos de éste), anunciará una ventana de 2048 comenzando con el siguiente byte esperado.

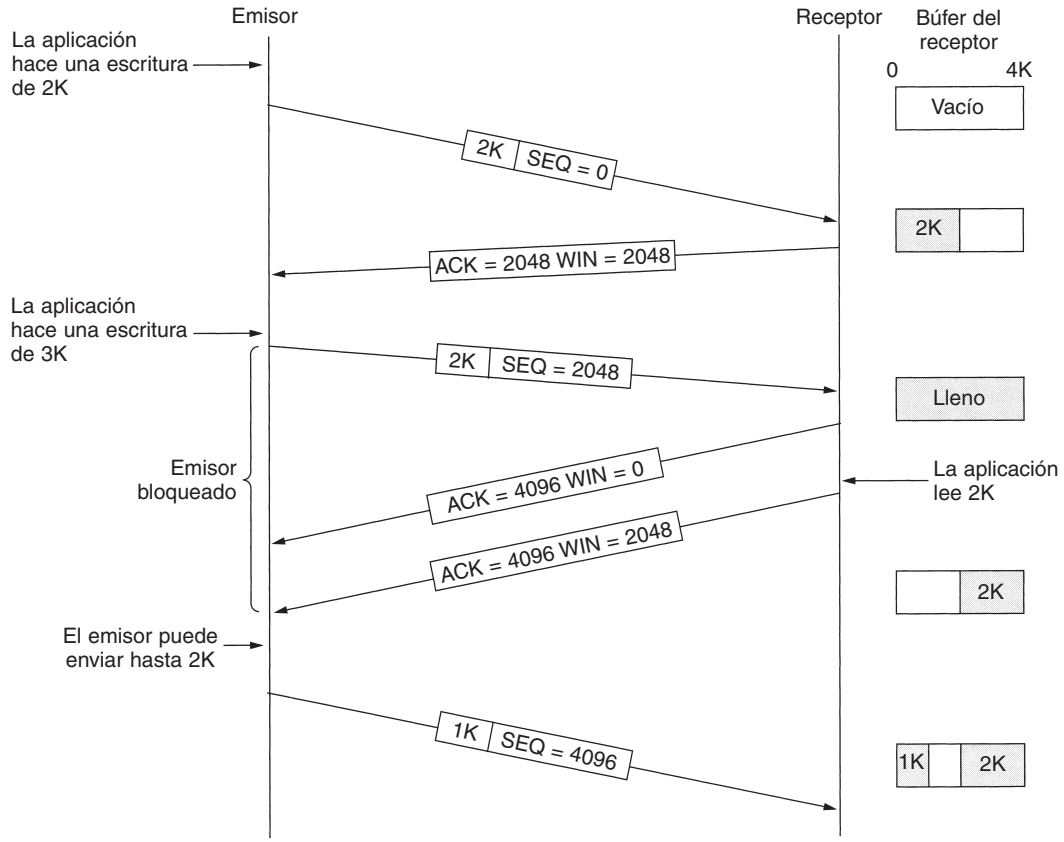


Figura 6-34. Administración de ventanas en TCP.

Ahora el emisor envía otros 2048 bytes, para los cuales el receptor envía la confirmación de recepción, pero la ventana anunciada es de 0. El emisor debe detenerse hasta que el proceso de aplicación del *host* receptor retire algunos datos del búfer, en cuyo momento el TCP puede anunciar una ventana más grande.

Cuando la ventana es de 0, el emisor normalmente no puede enviar segmentos, salvo en dos situaciones. Primera, pueden enviarse datos urgentes (por ejemplo, para permitir que el usuario elimine el proceso en ejecución en la máquina remota). Segunda, el emisor puede enviar un segmento de 1 byte para hacer que el receptor reanuncie el siguiente byte esperado y el tamaño de la ventana. El estándar TCP proporciona explícitamente esta opción para evitar un bloqueo irreversible si llega a perderse un anuncio de ventana.

No se requiere que los emisores envíen datos tan pronto como llegan de la aplicación. Tampoco se requiere que los receptores envíen confirmaciones de recepción tan pronto como sea posible. Por ejemplo, en la figura 6-34, cuando llegaron los primeros 2 KB de datos, el TCP, sabiendo que tenía disponible una ventana de 4 KB, habría actuado perfectamente bien si simplemente almacena en búfer los datos hasta la llegada de otros 2 KB, para poder transmitir un segmento con una carga útil de 4 KB. Esta libertad puede explotarse para mejorar el desempeño.

Considere una conexión telnet a un editor interactivo que reacciona con cada pulso de tecla. En el peor caso, al llegar un carácter a la entidad TCP emisora, el TCP crea un segmento TCP de 21 bytes que entrega al IP para su envío como datagrama IP de 41 bytes. Del lado receptor, el TCP envía de inmediato una confirmación de recepción de 40 bytes (20 bytes de encabezado TCP y 20 bytes de encabezado IP). Después, cuando el editor ha leído el byte, el TCP envía una actualización de ventana, recorriendo la ventana 1 byte hacia la derecha. Este paquete también es de 40 bytes. Por último, cuando el editor ha procesado el carácter, lo retransmite como paquete de 41 bytes. En conjunto, se usan 162 bytes de ancho de banda y se envían cuatro segmentos por cada carácter pulsado. Cuando es escaso el ancho de banda, no es deseable este método de operación.

Un enfoque que usan muchas implementaciones del TCP para mejorar esta situación es el retardo de las confirmaciones de recepción y de las actualizaciones de ventana durante 500 mseg con la esperanza de que lleguen algunos datos con los cuales viajar gratuitamente. Suponiendo que el editor hace eco en un lapso de 500 mseg, sólo se necesita enviar ahora un paquete de 41 bytes de regreso al usuario remoto, recortando a la mitad la cuenta de paquetes y el uso de ancho de banda.

Aunque esta regla reduce la carga impuesta a la red por el receptor, éste aún opera de manera ineficiente al enviar paquetes de 41 bytes que contienen 1 byte de datos. Una manera de reducir este uso es empleando el **algoritmo de Nagle** (Nagle, 1984). Lo que sugirió Nagle es sencillo: al llegar datos al emisor un byte a la vez, simplemente se envía el primer byte y se almacena en búfer los demás hasta la confirmación de recepción del byte pendiente. Luego se transmiten todos los caracteres del búfer en un segmento TCP y nuevamente se comienzan a almacenar en búfer los datos hasta que se haya confirmado la recepción de todos. Si el usuario escribe con rapidez y la red es lenta, puede entrar una cantidad importante de caracteres en cada segmento, reduciendo en gran medida el ancho de banda usado. Además, el algoritmo permite el envío de un nuevo paquete si han entrado suficientes datos para llenar la mitad de la ventana o la totalidad de un segmento.

El algoritmo de Nagle se usa ampliamente en las implementaciones de TCP, pero hay veces en que es mejor inhabilitarlo. En particular, al operar una aplicación X-Windows a través de Internet, los movimientos del ratón tienen que enviarse a la computadora remota. (X-Windows es el sistema de ventanas que se utiliza en la mayoría de los sistemas UNIX.) Su acumulación para enviarlos en ráfagas hace que el movimiento del cursor sea errático, lo que no complace mucho a los usuarios.

Otro problema que puede arruinar el desempeño del TCP es el **síndrome de ventana tonta** (Clark, 1982). Este problema ocurre cuando se pasan datos a la entidad emisora en bloques grandes, pero una aplicación interactiva del lado receptor lee datos a razón de 1 byte a la vez. Para ver el problema, observe la figura 6-35. Inicialmente, el búfer TCP del lado receptor está lleno y el

emisor lo sabe (es decir, tiene un tamaño de ventana de 0). Entonces la aplicación interactiva lee un carácter del flujo TCP. Esta acción hace feliz al TCP receptor, por lo que envía una actualización de ventana al emisor indicando que está bien que envíe 1 byte. El emisor accede y envía 1 byte. El búfer ahora está lleno, por lo que el receptor confirma la recepción del segmento de 1 byte pero establece la ventana en 0. Este comportamiento puede continuar indefinidamente.

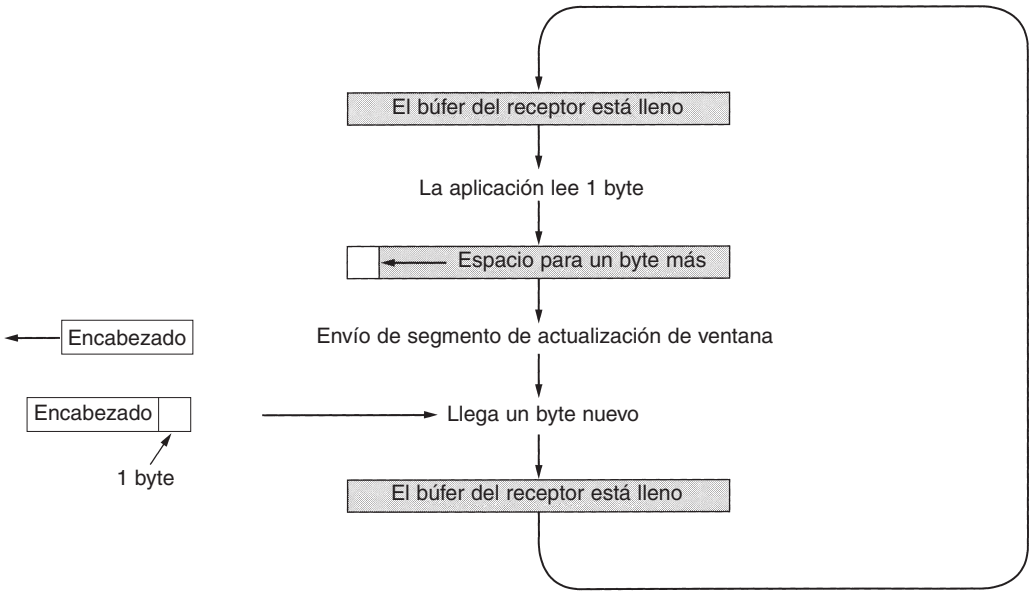


Figura 6-35. Síndrome de ventana tonta.

La solución de Clark es evitar que el receptor envíe una actualización de ventana para 1 byte. En cambio, se le obliga a esperar hasta tener disponible una cantidad de espacio, y luego lo anuncia. Específicamente, el receptor no debe enviar una actualización de ventana hasta que pueda manejar el tamaño máximo de segmento que anunció al establecerse la conexión, o que su búfer quede a la mitad de capacidad, lo que sea más pequeño.

Además, el emisor también puede ayudar al no enviar segmentos muy pequeños. En cambio, debe intentar esperar hasta haber acumulado suficiente espacio en la ventana para enviar un segmento completo, o cuando menos uno que contenga la mitad del tamaño de búfer del receptor (que debe estimar a partir del patrón de las actualizaciones de ventana que ha recibido anteriormente).

El algoritmo de Nagle y la solución de Clark al síndrome de ventana tonta son complementarios. Nagle trataba de resolver el problema causado por la entrega de datos al TCP desde la aplicación emisora un byte a la vez. Clark trataba de resolver el problema de que la aplicación receptora toma los datos del TCP un byte a la vez. Ambas soluciones son válidas y pueden operar juntas. La meta es que el emisor no envíe segmentos pequeños y que el receptor no los pida.

El TCP receptor también puede hacer más para mejorar el desempeño que simplemente actualizar ventanas en unidades grandes. Al igual que el TCP emisor, tiene la capacidad de almacenar datos en el búfer, por lo que puede bloquear una solicitud READ de la aplicación hasta poder proporcionar un bloque grande de datos. Hacer esto reduce la cantidad de llamadas al TCP y, por tanto, la sobrecarga. Por supuesto, también aumenta el tiempo de respuesta, pero en las aplicaciones no interactivas, como la transferencia de archivos, la eficiencia puede tener mayor peso que el tiempo de respuesta a las solicitudes individuales.

Otro problema del receptor es qué debe hacer con los segmentos fuera de orden. Pueden conservarse o descartarse, al albedrío del receptor. Por supuesto, las confirmaciones de recepción pueden enviarse sólo después de haber recibido todos los datos hasta el byte confirmado. Si el receptor recibe los segmentos 0, 1, 2, 4, 5, 6 y 7, puede enviar una confirmación de recepción de todos los bytes hasta el último byte del segmento 2, inclusive. Al expirar el temporizador del emisor, retransmitirá el segmento 3. Si el receptor tienen en búfer los segmentos 4 a 7, al recibir el segmento 3 puede enviar una confirmación de recepción de todos los bytes hasta el final del segmento 7.

6.5.9 Control de congestión en TCP

Cuando la carga ofrecida a cualquier red es mayor que la que puede manejar, se genera una congestión. Internet no es ninguna excepción. En esta sección estudiaremos los algoritmos que se han desarrollado durante la última década para manejar la congestión. Aunque la capa de red también intenta manejarlos, gran parte del trabajo pesado recae sobre el TCP porque la solución real a la congestión es la disminución de la tasa de datos.

En teoría, puede manejarse la congestión aprovechando un principio de física: la ley de conservación de los paquetes. La idea es no inyectar un paquete nuevo en la red hasta que salga uno viejo (es decir, se entregue). El TCP intenta lograr esta meta manipulando dinámicamente el tamaño de la ventana.

El primer paso del manejo de la congestión es su detección. Antaño la detección de congestionamientos era muy difícil. La expiración de un temporizador causada por un paquete perdido podía deberse a (1) ruido en la línea de transmisión o (2) el descarte de paquetes en el enrutador congestionado. Saber cuál era la diferencia era difícil.

Hoy día, la pérdida de paquetes por errores de transmisión es relativamente rara debido a que las troncales de larga distancia son de fibra (aunque las redes inalámbricas son otra historia). En consecuencia, la mayoría de las expiraciones de tiempo en Internet se deben a congestión. Todos los algoritmos TCP de Internet suponen que las expiraciones de tiempo son causadas por congestión y las revisan en busca de problemas, de la misma manera que los mineros observan a sus canarios.

Antes de analizar la manera en que el TCP reacciona a la congestión, describiremos primero lo que hace para evitar que ocurra. Al establecerse una conexión, se tiene que seleccionar un tamaño de ventana adecuado. El receptor puede especificar una ventana con base en su tamaño de búfer. Si el emisor se ajusta a su tamaño de ventana, no ocurrirán problemas por desbordamiento de búferes en la terminal receptora, pero aún pueden ocurrir debido a congestión interna en la red.

En la figura 6-36 ilustramos este problema hidráulicamente. En la figura 6-36(a) vemos un tubo grueso que conduce a un receptor de poca capacidad. Mientras el emisor no envíe más agua de la que puede contener la cubeta, no se perderá agua. En la figura 6-36(b), el factor limitante no es la capacidad de la cubeta, sino la capacidad de conducción interna de la red. Si entra demasiada agua a alta velocidad, ésta retrocederá, perdiéndose algo (en este caso, por el desbordamiento del embudo).

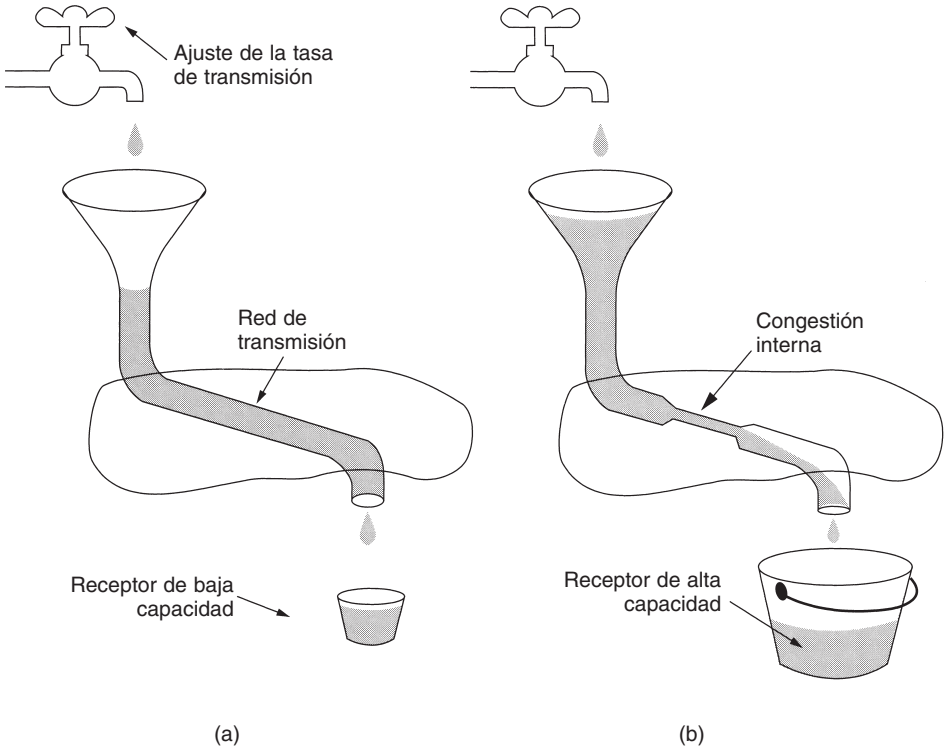


Figura 6-36. (a) Red rápida alimentando un receptor de baja capacidad. (b) Red lenta alimentando un receptor de alta capacidad.

La solución de Internet es aceptar que existen dos problemas potenciales (capacidad de la red y capacidad del receptor) y manejarlos por separado. Para ello, cada emisor mantiene dos ventanas: la ventana que ha otorgado el receptor y una segunda ventana, la **ventana de congestión.** Cada una refleja la cantidad de bytes que puede enviar el emisor. La cantidad de bytes que pueden enviarse es la cifra menor de las dos ventanas. Por tanto, la ventana efectiva es el mínimo de lo que el emisor piensa que es correcto y lo que el receptor piensa que está bien. Si el receptor dice "envía 8 KB" pero el emisor sabe que las ráfagas de más de 4 KB saturan la red, envía 4 KB. Por otra parte, si el receptor dice "envía 8 KB" y el emisor sabe que las ráfagas de hasta 32 KB pueden llegar sin problemas, envía los 8 KB solicitados.

Al establecer una conexión, el emisor asigna a la ventana de congestión el tamaño de segmento máximo usado por la conexión; entonces envía un segmento máximo. Si se recibe la confirmación de recepción de este segmento antes de que expire el temporizador, el emisor agrega el equivalente en bytes de un segmento a la ventana de congestión para hacerla de dos segmentos de tamaño máximo, y envía dos segmentos. A medida que se confirma cada uno de estos segmentos, se aumenta el tamaño de la ventana de congestión en un segmento máximo. Cuando la ventana de congestión es de *n* segmentos, si de todos los *n* se reciben confirmaciones de recepción a tiempo, se aumenta el tamaño de la ventana de congestión en la cuenta de bytes correspondiente a *n* segmentos. De hecho, cada ráfaga confirmada duplica la ventana de congestionamiento.

La ventana de congestión sigue creciendo exponencialmente hasta ocurrir una expiración del temporizador o alcanzar el tamaño de la ventana receptora. La idea es que, si las ráfagas de 1024, 2048 y 4096 bytes funcionan bien, pero una ráfaga de 8192 produce una expiración del temporizador, la ventana de congestión debe establecerse en 4096 para evitar la congestión. Mientras el tamaño de la ventana de congestión permanezca en 4096, no se enviará una ráfaga de mayor longitud, sin importar la cantidad de espacio de ventana otorgada por el receptor. Este algoritmo se llama **arranque lento,** pero no es lento en lo absoluto (Jacobson, 1988); es exponencial, y se requiere que todas las implementaciones de TCP lo manejen.

Veamos ahora el algoritmo de control de congestión de Internet, el cual usa un tercer parámetro, el **umbral,** inicialmente de 64 KB, además de las ventanas de recepción y congestión. Al ocurrir una expiración del temporizador, se establece el umbral en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo. Luego se usa el arranque lento para determinar lo que puede manejar la red, excepto que el crecimiento exponencial termina al alcanzar el umbral. A partir de este punto, las transmisiones exitosas aumentan linealmente la ventana de congestión (en un segmento máximo por ráfaga) en lugar de uno por segmento. En efecto, este algoritmo está suponiendo que probablemente es aceptable recortar la ventana de congestión a la mitad, y luego aumentarla gradualmente a partir de ahí.

Como ilustración de la operación del algoritmo de congestión, véase la figura 6-37. El tamaño máximo de segmento aquí es de 1024 bytes. Inicialmente, la ventana de congestión era de 64 KB, pero ocurre una expiración del temporizador, así que se establece el umbral en 32KB y la ventana de congestión en 1KB, para la transmisión 0. La ventana de congestión entonces crece exponencialmente hasta alcanzar el umbral (32 KB). A partir de entonces, crece linealmente.

La transmisión 13 tiene mala suerte (debería saberlo) y ocurre una expiración del temporizador. Se establece el umbral en la mitad de la ventana actual (ahora de 40 KB, por lo que la mitad es de 20 KB), e inicia de nuevo el arranque lento. Al llegar las confirmaciones de recepción de la transmisión 14, los primeros cuatro incrementan la ventana de congestión en un segmento máximo, pero después de eso el crecimiento se vuelve lineal nuevamente.

Si no ocurren más expiraciones del temporizador, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor. En ese punto, dejará de crecer y permanecerá constante mientras no ocurran más expiraciones del temporizador y la ventana del receptor no cambie de tamaño. Como nota al margen, si llega un paquete SOURCE QUENCH de ICMP y pasa al TCP, este evento será tratado de la misma manera que una expiración del temporizador. Un enfoque alternativo (y más reciente) se describe en el RFC 3168.

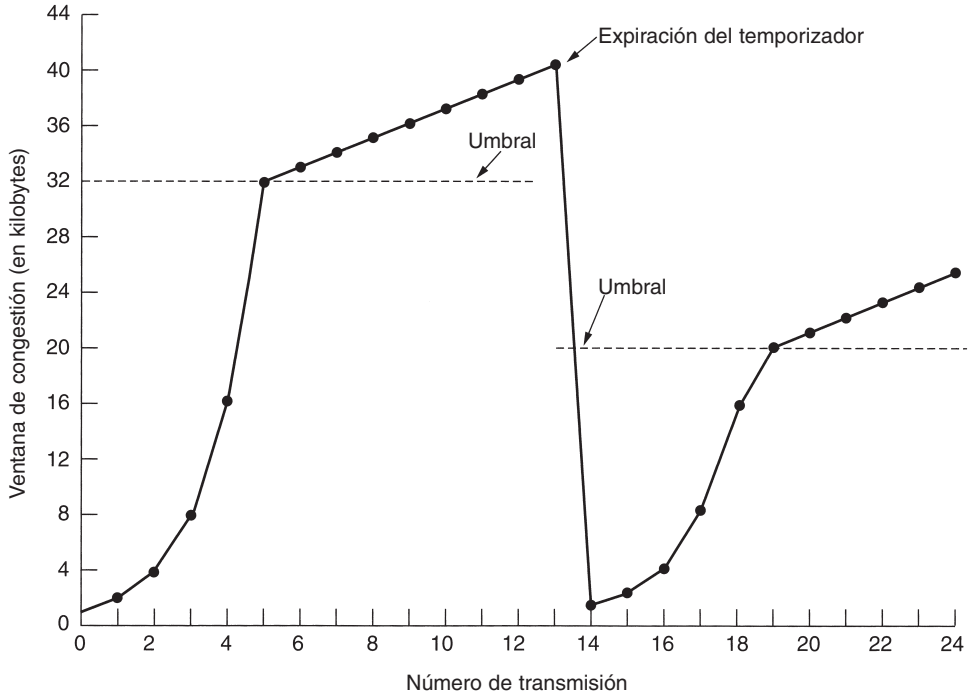


Figura 6-37. Ejemplo del algoritmo de congestión de Internet.

6.5.10 Administración de temporizadores del TCP

El TCP usa varios temporizadores (al menos conceptualmente) para hacer su trabajo. El más importante de éstos es el **temporizador de retransmisión.** Al enviarse un segmento, se inicia un temporizador de retransmisiones. Si la confirmación de recepción del segmento llega antes de expirar el temporizador, éste se detiene. Si, por otra parte, el temporizador termina antes de llegar la confirmación de recepción, se retransmite el segmento (y se inicia nuevamente el temporizador). Surge entonces la pregunta: ¿qué tan grande debe ser el intervalo de expiración del temporizador?

Este problema es mucho más difícil en la capa de transporte de Internet que en los protocolos de enlace de datos genéricos del capítulo 3. En este último caso, el retardo esperado es altamente predecible (es decir, tiene una variación baja), por lo que el temporizador puede ejecutarse para expirar justo después del momento en que se espera la confirmación de recepción, como se muestra en la figura 6-38(a). Dado que las confirmaciones de recepción pocas veces se retardan en la capa de enlace de datos (debido a la falta de congestión), la ausencia de una confirmación de recepción en el momento esperado generalmente significa que la trama o la confirmación de recepción se han perdido.

El TCP enfrenta un entorno radicalmente distinto. La función de densidad de probabilidad del tiempo que tarda en regresar una confirmación de recepción TCP se parece más a la figura 6-38(b) que a la figura 6-38(a). Es complicada la determinación del tiempo de ida y vuelta al destino. Aun

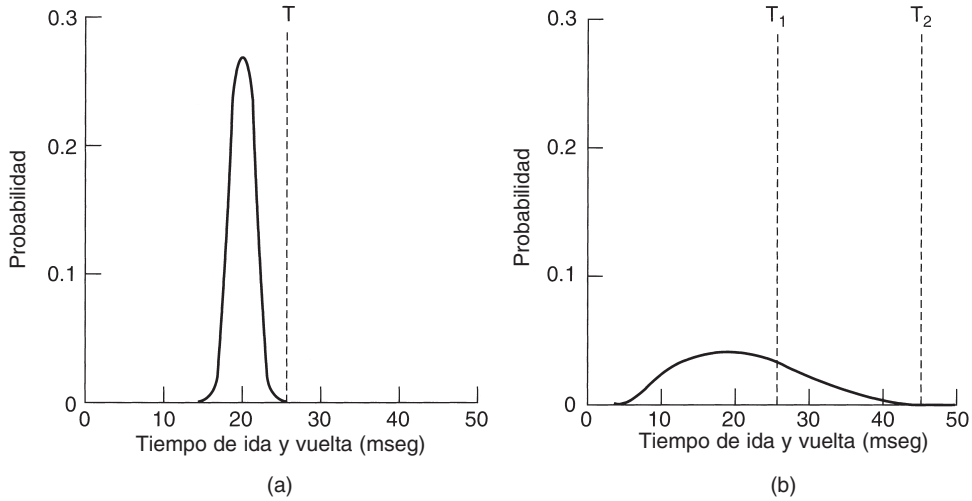


Figura 6-38. (a) Densidad de probabilidad de los tiempos de llegada de confirmaciones de recepción en la capa de enlace de datos. (b) Densidad de probabilidad de los tiempos de llegada de confirmaciones de recepción para el TCP.

cuando se conoce, la selección del intervalo de expiración del temporizador también es difícil. Si se hace demasiado corto, digamos T_1 en la figura 6-38(b), ocurrirán retransmisiones innecesarias, cargando la Internet con paquetes inútiles. Si se hace demasiado largo (por ejemplo, T_2), el desempeño sufrirá debido al gran retardo de retransmisión de cada paquete perdido. Es más, la varianza y la media de la distribución de llegadas de confirmaciones de recepción pueden variar con rapidez en unos cuantos segundos, a medida que se generan y se resuelven congestionamientos.

La solución es usar un algoritmo muy dinámico que ajuste constantemente el intervalo de expiración del temporizador, con base en mediciones continuas del desempeño de la red. El algoritmo que generalmente usa el TCP lo debemos a Jacobson (1988) y funciona como sigue. Por cada conexión, el TCP mantiene una variable, *RTT* (*round-trip time*), que es la mejor estimación actual del tiempo de ida y vuelta al destino en cuestión. Al enviarse un segmento, se inicia un temporizador, tanto para ver el tiempo que tarda la confirmación de recepción como para habilitar una retransmisión si se tarda demasiado. Si llega la confirmación de recepción antes de expirar el temporizador, el TCP mide el tiempo que tardó la confirmación de recepción, digamos *M*. Entonces actualiza *RTT* de acuerdo con la fórmula

$$RTT = \alpha RTT + (1 - \alpha)M$$

donde α es un factor de amortiguamiento que determina el peso que se le da al valor anterior. Por lo común, $\alpha = 7/8$.

Aun dado un buen valor de *RTT*, la selección de una expiración adecuada del temporizador de retransmisión no es un asunto sencillo. Normalmente el TCP usa βRTT, pero el truco es seleccionar β. En las implementaciones iniciales, β siempre era 2, pero la experiencia demostró que un valor constante era inflexible puesto que no respondía cuando subía la variación.

En 1988, Jacobson propuso hacer que β fuera aproximadamente proporcional a la desviación estándar de la función de densidad de probabilidad del tiempo de llegada de las confirmaciones de recepción, por lo que una variación grande significa una β grande, y viceversa. En particular, sugirió el uso de la *desviación media* como una forma rápida de estimar la *desviación estándar.* Su algoritmo requiere mantener otra variable amortiguada, *D,* la desviación. Al llegar una confirmación de recepción, se calcula la diferencia entre el valor esperado y el observado, $|RTT - M|$. Un valor amortiguado de esta cifra se mantiene en *D* mediante la fórmula

$$D = \alpha D + (1 - \alpha)\,|RTT - M|$$

donde α puede ser o no el mismo valor usado para amortiguar *RTT*. Si bien *D* no es exactamente igual a la desviación estándar, es bastante buena, y Jacobson demostró la manera de calcularla usando sólo sumas, restas y desplazamientos de enteros, lo que es una gran ventaja. La mayoría de las implementaciones TCP usan ahora este algoritmo y establecen el intervalo de expiración del temporizador en

$$\text{Expiración del temporizador} = RTT + 4 \times D$$

La selección del factor 4 es un tanto arbitraria, pero tiene dos ventajas. Primera, puede hacerse la multiplicación por 4 con un solo desplazamiento. Segunda, reduce al mínimo las expiraciones de temporizador y las retransmisiones innecesarias porque menos del 1% de todos los paquetes llegan más de cuatro desviaciones estándar tarde. (En realidad, Jacobson sugirió inicialmente que se usaran 2, pero el trabajo posterior ha demostrado que 4 da un mejor desempeño.)

Un problema que ocurre con la estimación dinámica de *RTT* es qué se debe hacer cuando expira el temporizador de un segmento y se envía de nuevo. Cuando llega la confirmación de recepción, no es claro si éste se refiere a la primera transmisión o a una posterior. Si se adivina mal se puede contaminar seriamente la estimación de *RTT*. Phil Karn descubrió este problema de la manera difícil. Él es un radioaficionado interesado en la transmisión de paquetes TCP/IP a través de la radio amateur, un medio notoriamente poco confiable (en un buen día, pasarán la mitad de los paquetes). Karn hizo una propuesta sencilla: no actualizar el *RTT* con ninguno de los segmentos retransmitidos. En cambio, se duplica la expiración del temporizador con cada falla hasta que los segmentos pasan a la primera. Este sistema se llama **algoritmo de Karn** y lo usan la mayoría de las implementaciones TCP.

El temporizador de retransmisiones no es el único usado por el TCP. El segundo temporizador es el **temporizador de persistencia,** diseñado para evitar el siguiente bloqueo irreversible. El receptor envía una confirmación de recepción con un tamaño de ventana de 0, indicando al emisor que espere. Después, el receptor actualiza la ventana, pero se pierde al paquete con la actualización. Ahora, tanto el emisor como el receptor están esperando que el otro haga algo. Cuando termina el temporizador de persistencia, el emisor envía un sondeo al receptor. La respuesta al sondeo da el tamaño de la ventana. Si aún es de cero, se inicia el temporizador de persistencia nuevamente y se repite el ciclo. Si es diferente de cero, pueden enviarse datos.

Un tercer temporizador usado en algunas implementaciones es el **temporizador de seguir con vida** (*keepalive timer*). Cuando una conexión ha estado inactiva durante demasiado tiempo, el

temporizador de seguir con vida puede expirar, haciendo que un lado compruebe que el otro aún está ahí. Si no se recibe respuesta, se termina la conexión. Esta característica es motivo de controversias porque agrega sobrecarga y puede terminar una conexión saludable debido a una partición temporal de la red.

El último temporizador usado en cada conexión TCP es el que se usa en el estado *TIMED WAIT* durante el cierre; opera durante el doble del tiempo máximo de vida de paquete para asegurar que, al cerrarse una conexión, todos los paquetes creados por ella hayan desaparecido.

6.5.11 TCP y UDP inalámbricos

En teoría, los protocolos de transporte deben ser independientes de la tecnología de la capa de red subyacente. En particular, el TCP no debería preocuparse si el IP está operando por fibra o por radio. En la práctica sí importa, puesto que la mayoría de las implementaciones de TCP han sido optimizadas cuidadosamente con base en supuestos que se cumplen en las redes alámbricas, pero no en las inalámbricas. Ignorar las propiedades de la transmisión inalámbrica puede conducir a implementaciones del TCP correctas desde el punto de vista lógico pero con un desempeño horrendo.

El problema principal es el algoritmo de control de congestionamientos. Hoy día, casi todas las implementaciones de TCP suponen que las expiraciones del temporizador ocurren por congestionamientos, no por paquetes perdidos. En consecuencia, al expirar un temporizador, el TCP disminuye su velocidad y envía con menor ímpetu (por ejemplo, el algoritmo de arranque lento de Jacobson). Lo que se pretende con este enfoque es reducir la carga de la red y aliviar así la congestión.

Desgraciadamente, los enlaces de transmisión inalámbrica son muy poco confiables. Pierden paquetes todo el tiempo. El enfoque adecuado para el manejo de paquetes perdidos es enviarlos nuevamente, tan pronto como sea posible. La reducción de la velocidad simplemente empeora las cosas. Si, digamos, se pierde el 20% de todos los paquetes, entonces cuando el emisor envía 100 paquetes/seg, la velocidad real de transporte es de 80 paquetes/seg. Si el emisor reduce su velocidad a 50 paquetes/seg, la velocidad real de transporte cae a 40 paquetes/seg.

En efecto, al perderse un paquete en una red alámbrica, el emisor debe reducir la velocidad. Cuando se pierde uno en una red inalámbrica, el emisor debe acelerar. Cuando el emisor no sabe de qué clase de red se trata, es difícil tomar la decisión correcta.

Con frecuencia, la trayectoria del emisor al receptor no es homogénea. Los primeros 1000 km podrían ser a través de una red alámbrica, pero el último km podría ser inalámbrico. Ahora es más difícil la decisión correcta en el caso de una expiración del temporizador, ya que es importante saber dónde ocurrió el problema. Una solución propuesta por Bakne y Badrinath (1995), el **TCP indirecto,** es la división de la conexión TCP en dos conexiones distintas, como se muestra en la figura 6-39. La primera va del emisor a la estación base. La segunda va de la estación base al receptor. La estación base simplemente copia paquetes entre las conexiones en ambas direcciones.

La ventaja de este esquema es que ahora ambas conexiones son homogéneas. Las expiraciones del temporizador en la primera conexión pueden reducir la velocidad del emisor, y las expiraciones del temporizador en la segunda pueden acelerarla. También es posible ajustar otros parámetros

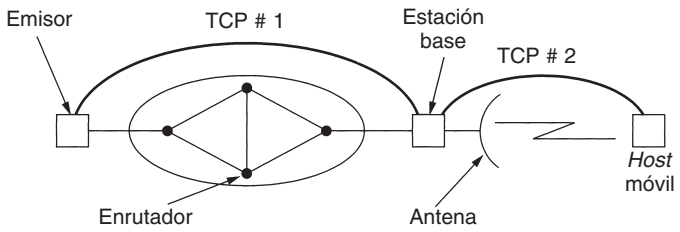


Figura 6-39. División de una conexión TCP en dos conexiones.

por separado para cada conexión. La desventaja es que se viola por completo la semántica del TCP. Dado que cada parte de la conexión es una conexión TCP completa, la estación base confirma la recepción de cada segmento TCP de la manera normal, sólo que ahora la recepción de una confirmación en el emisor no significa que el receptor recibió el segmento, sino que la estación base lo recibió.

Una solución diferente, debido a Balakrishnan y cols. (1995), no quebranta la semántica del TCP. Funciona haciendo varias modificaciones pequeñas al código de la capa de red de la estación base. Uno de los cambios es la adición de un agente espía que observa y almacena en caché los segmentos TCP que van al *host* móvil y las confirmaciones de recepción que regresan de él. Cuando el agente espía ve un segmento TCP que sale al *host* móvil, pero no ve el regreso de una confirmación de recepción antes de que su temporizador (relativamente corto) expire, simplemente retransmite ese segmento, sin indicar al origen que lo está haciendo. El agente también genera una retransmisión cuando detecta confirmaciones de recepción duplicadas del *host* móvil, lo que invariablemente significa que algo le ha fallado a este *host*. Las confirmaciones de recepción duplicadas se descartan en seguida, para evitar que el origen las malinterprete como una señal de congestionamiento.

Sin embargo, una desventaja de esta transparencia es que, si el enlace inalámbrico tiene muchas pérdidas, el temporizador del transmisor podría expirar esperando una confirmación de recepción e invocar el algoritmo de control de congestión. Con en TCP indirecto, el algoritmo de control de congestión nunca iniciará hasta que realmente haya congestión en la parte alámbrica de la red.

El documento de Balakrishnan y cols., también sugiere una solución al problema de segmentos perdidos que se originan en el *host* móvil. Al notar una estación base un hueco en los números de secuencia de entrada, genera una solicitud de repetición selectiva de los bytes faltantes usando una opción TCP.

Gracias a estos dos mecanismos, el enlace inalámbrico se hace más confiable en ambas direcciones, sin que el origen lo sepa y sin cambiar la semántica del TCP.

Si bien el UDP no tiene los mismos problemas que el TCP, la comunicación inalámbrica también le produce dificultades. El problema principal es que los programas usan el UDP pensando que es altamente confiable. Saben que no hay garantías, pero aun así esperan que sea casi perfecto. En un entorno inalámbrico, UDP estará muy lejos de serlo. En aquellos programas capaces de recuperarse de la pérdida de mensajes UDP, pasar repentinamente de un entorno en el que pueden

perderse mensajes, pero rara vez ocurre, a uno en el que se pierden constantemente, puede dar pie a un desempeño desastroso.

La comunicación inalámbrica también afecta otras áreas, no sólo el desempeño. Por ejemplo, ¿cómo encuentra un *host* móvil una impresora local a la cual conectarse, en lugar de usar su impresora base? Algo parecido a esto es cómo acceder a la página WWW de la celda local, aun si no se conoce su nombre. También, los diseñadores de páginas WWW tienden a suponer que hay mucho ancho de banda disponible. Un logotipo grande en cada página se vuelve contraproducente si su transmisión tarda 10 seg en un enlace inalámbrico lento cada vez que se hace referencia a la página, irritando sobremanera a los usuarios.

Conforme las redes inalámbricas se vuelvan más comunes, los problemas de ejecutar TCP sobre ellas se volverán más serios. En (Barakat y cols., 2000; Ghani y Dixit, 1999; Huston, 2001, y Xylomenos y cols., 2001), encontrará información adicional sobre esta área.

6.5.12 TCP para Transacciones

Al inicio de este capítulo vimos las llamadas a procedimiento remoto como una forma de implementar sistemas cliente-servidor. Si tanto la solicitud como la respuesta son suficientemente pequeñas para caber en paquetes sencillos y la operación tiene la misma potencia, simplemente se puede utilizar UDP. Sin embargo, si estas condiciones no se cumplen, el uso de UDP no es tan conveniente. Por ejemplo, si la respuesta puede ser más grande, las piezas deben seguir una secuencia y se debe diseñar un mecanismo para retransmitir las piezas perdidas. En efecto, la aplicación tiene que remodelar el TCP.

Es obvio que esto no resulta tan conveniente, pero tampoco lo es utilizar el TCP mismo. El problema es la eficiencia. En la figura 6-40(a) se muestra la secuencia normal de paquetes para realizar una RPC en TCP. En el mejor de los casos se necesitan los siguientes nueve paquetes.

1. El cliente envía un paquete *SYN* para establecer una conexión.
2. El servidor envía un paquete *ACK* para confirmar la recepción del paquete *SYN*.
3. El cliente completa el acuerdo de tres vías.
4. El cliente envía la solicitud real.
5. El cliente envía un paquete *FIN* para indicar que ha terminado el envío.
6. El servidor confirma la recepción de la solicitud y el paquete *FIN*.
7. El servidor envía la respuesta al cliente.
8. El servidor envía un paquete *FIN* para indicar que también ha terminado.
9. El cliente confirma la recepción del paquete *FIN* del servidor.

Observe que éste es el mejor de los casos. En el peor, la confirmación de recepción de la solicitud del cliente y del paquete *FIN* se realiza por separado, al igual que la respuesta y el paquete *FIN* del servidor.

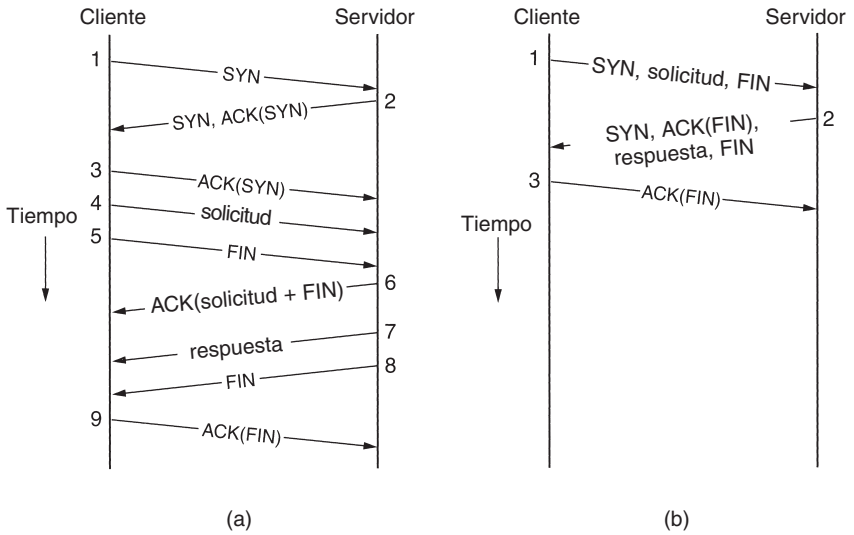


Figura 6-40. (a) RPC mediante el TCP normal. (b) RPC mediante el T/TCP.

Con esto surge rápidamente la pregunta de si hay alguna forma para combinar la eficiencia de RPC utilizando UDP (sólo dos mensajes) con la confiabilidad de TCP. La respuesta es: Casi. Puede hacerse con una variante TCP experimental llamada **T/TCP** (**TCP para Transacciones**), que se describe en los RFCs 1379 y 1644.

La idea central es modificar ligeramente la secuencia estándar de configuración de conexión para permitir la transferencia de datos durante la configuración. En la figura 6-40(b) se ilustra el protocolo T/TCP. El primer paquete del cliente contiene el bit *SYN*, la solicitud misma y el paquete *FIN*. Lo que dice es: Deseo establecer una conexión, aquí están los datos, y con esto termino.

Cuando el servidor obtiene la solicitud, busca o calcula la respuesta y elige cómo responder. Si la respuesta se ajusta en un paquete, da la respuesta de la figura 6-40(b), que dice: Confirmo la recepción de tu paquete *FIN*, aquí está la respuesta, y con esto termino. A continuación el cliente confirma la recepción del paquete *FIN* del servidor, y el protocolo termina en tres mensajes.

Sin embargo, si el resultado es de más de un paquete, el servidor también tiene la opción de no encender el bit *FIN*, en cuyo caso puede enviar múltiples paquetes antes de cerrar su dirección.

Probablemente valga la pena mencionar que T/TCP no es la única mejora propuesta a TCP. Otra propuesta es **SCTP** (**Protocolo de Transmisión de Control de Flujo**). Sus características incluyen preservación de los límites de mensajes, modos de entrega múltiples (por ejemplo, entrega en desorden), multihoming (destinos de respaldo) y confirmaciones de recepción selectivas (Stewart y Metz, 2001). Sin embargo, siempre que alguien propone cambiar algo que ha trabajado bien por algún tiempo considerable, hay una gran batalla entre las siguientes posturas: “Los usuarios están demandando más características” y “Si no está roto, no lo arregles”.

6.6 ASPECTOS DEL DESEMPEÑO

Los asuntos relacionados con el desempeño son muy importantes en las redes de cómputo. Cuando hay cientos o miles de computadoras conectadas entre sí, son comunes las interacciones complejas, con consecuencias imprevisibles. Frecuentemente esta complejidad conduce a un desempeño pobre, sin que nadie sepa por qué. En las siguientes secciones examinaremos muchos temas relacionados con el desempeño de las redes para ver los tipos de problemas que existen y lo que se puede hacer para resolverlos.

Desgraciadamente, el entendimiento del desempeño de las redes es más un arte que una ciencia. Muy poca de la teoría tiene en realidad alguna utilidad en la práctica. Lo mejor que podemos hacer es dar reglas empíricas derivadas de los tropiezos y ejemplos actuales tomados del mundo real. Intencionalmente hemos postergado este análisis hasta después de estudiar la capa de transporte en las redes TCP y ATM, a fin de poder señalar los lugares en los que se han hecho bien o mal las cosas.

La capa de transporte no es el único lugar en el que surgen asuntos relacionados con el desempeño. Vimos algunos de ellos en la capa de red, en el capítulo anterior. No obstante, por lo general la capa de red está bastante ocupada con el enrutamiento y el control de congestionamientos. Los puntos más amplios, orientados al sistema, tienden a relacionarse con el transporte, por lo que este capítulo es un lugar adecuado para examinarlos.

En las siguientes cinco secciones estudiaremos cinco aspectos del desempeño de las redes:

1. Problemas de desempeño.
2. Medición del desempeño de una red.
3. Diseño de sistemas con mejor desempeño.
4. Procesamiento rápido de las TPDUs.
5. Protocolos para redes futuras de alto desempeño.

Como comentario, necesitamos un nombre para las unidades intercambiadas por las entidades de transporte. El término de TCP, segmento, es confuso en el mejor de los casos, y nunca se usa fuera del mundo TCP en este contexto. Los términos CS-PDU, SAR-PDU y CPCS-PDU son específicos de ATM. Los paquetes claramente se refieren a la capa de red y los mensajes pertenecen a la capa de aplicación. A falta de un término estándar, volveremos a llamar TPDUs a las unidades intercambiadas por las entidades de transporte. Cuando deseemos referirnos tanto a TPDUs como a paquetes, usaremos paquete como término colectivo, como en “la CPU debe ser lo bastante rápida como para procesar los paquetes de entrada en tiempo real”. Con esto nos referimos tanto al paquete de capa de red como a la TPDU encapsulada en él.

6.6.1 Problemas de desempeño en las redes de cómputo

Algunos problemas de desempeño, como la congestión, son causados por sobrecargas temporales de los recursos. Si repentinamente llega más tráfico a un enrutador que el que puede manejar, surgirá la congestión y el desempeño bajará. Ya estudiamos la congestión en detalle en el capítulo anterior.

El desempeño también se degrada cuando hay un desequilibrio estructural de los recursos. Por ejemplo, si una línea de comunicación de gigabits está conectada a una PC de bajo rendimiento, la pobre CPU no será capaz de procesar los paquetes de entrada a la velocidad suficiente, y se perderán algunos. Estos paquetes se retransmitirán tarde o temprano, agregando un retardo, desperdiciando ancho de banda y reduciendo en general el desempeño.

Las sobrecargas también pueden generarse sincrónicamente. Por ejemplo, si una TPDU contiene un parámetro erróneo (por ejemplo, el puerto al que está destinada), en muchos casos el receptor cortésmente enviará una notificación de error. Ahora considere lo que podría ocurrir si se difundiera una TPDU errónea a 10,000 máquinas: cada una podría devolver un mensaje de error. La **tormenta de difusión** resultante podría paralizar la red. El UDP adoleció de este problema hasta que se cambió el protocolo para hacer que los *hosts* evitaran responder a errores en las TPDUs de UDP enviadas a direcciones de difusión.

Un segundo ejemplo de sobrecarga síncrona es lo que ocurre tras una falla del suministro eléctrico. Al regresar la energía, todas las máquinas saltan simultáneamente a sus ROMs para reiniciarse. Una secuencia de arranque común podría requerir acudir primero a algún servidor (DHCP) para conocer la identidad verdadera de la máquina, y luego a un servidor de archivos para obtener una copia del sistema operativo. Si cientos de máquinas hacen todo esto al mismo tiempo, el servidor probablemente se vendría abajo por la carga.

Aun en ausencia de sobrecargas síncronas y con suficientes recursos disponibles, puede haber un bajo desempeño debido a la falta de afinación del sistema. Por ejemplo, si una máquina tiene bastante potencia y memoria de CPU, pero no se ha asignado suficiente memoria como espacio de búfer, ocurrirán desbordamientos y se perderán varias TPDUs. De la misma manera, si el algoritmo de calendarización no tiene una prioridad bastante alta como para procesar las TPDUs de entrada, podrán perderse algunas.

Otro asunto relativo a la afinación es el establecimiento correcto de los temporizadores. Cuando se envía una TPDU, normalmente se utiliza un temporizador para protegerse contra pérdidas. Si se asigna un valor muy bajo al temporizador, ocurrirán retransmisiones innecesarias, congestionando los alambres. Si el valor es demasiado alto, ocurrirán retardos innecesarios tras la pérdida de una TPDU. Otros parámetros afinables incluyen el tiempo de espera para incorporar datos a paquetes antes de enviar confirmaciones de recepción por separado, y la cantidad de retransmisiones antes de darse por vencido.

Las redes de gigabits traen consigo nuevos problemas de desempeño. Por ejemplo, considere el envío de una ráfaga de datos de 64 KB de San Diego a Boston para llenar el búfer de 64 KB del receptor. Suponga que el enlace es de 1 Gbps y que el retardo de la luz en un sentido a través de la fibra es de 20 mseg. Inicialmente, en $t = 0$, el canal está vacío, como se muestra en la figura 6-41(a). Apenas 500 μseg después [figura 6-41(b)] todas las TPDUs están en la fibra. La TPDU a la cabeza ahora estará en algún lugar del vecindario de Brawley, todavía al sur de California. Sin embargo, el emisor debe detenerse hasta recibir la actualización de ventana.

Después de 20 mseg, la TPDU puntera llega a Boston, como se muestra en la figura 6-41(c), y se envía una confirmación de recepción. Por último, 40 mseg después de comenzar, llega la primera confirmación de recepción al emisor y puede transmitirse la segunda ráfaga. Dado que la línea de transmisión se usó durante 0.5 mseg de un total de 40, la eficiencia es de aproximadamente 1.25%. Esta situación es típica de la operación de protocolos antiguos sobre líneas de gigabits.

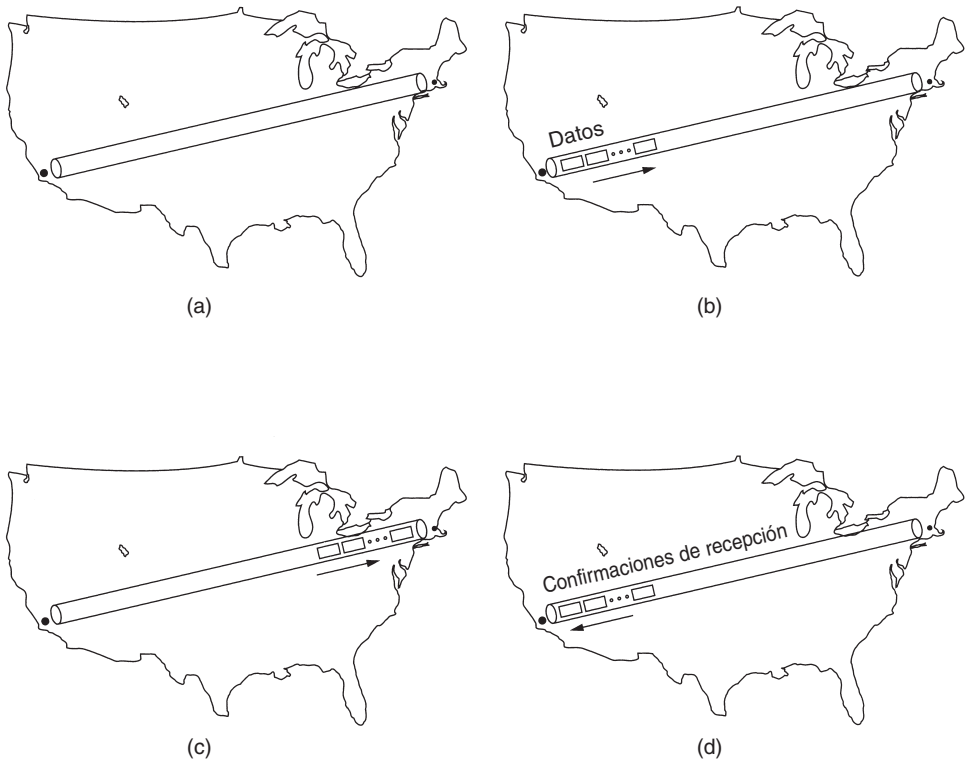


Figura 6-41. Estado de transmisión de un megabit de San Diego a Boston. (a) En $t = 0$. (b) Tras 500 μseg. (c) Tras 20 mseg. (d) Tras 40 mseg.

Una cantidad que conviene recordar durante el análisis del desempeño de redes es el **producto ancho de banda-retardo** que se obtiene al multiplicar el ancho de banda (en bits/seg) por el tiempo de retardo de ida y vuelta (en seg). El producto es la capacidad del canal desde el emisor al receptor y de regreso (en bits).

Para el ejemplo de la figura 6-41, el producto ancho de banda-retardo es de 40 millones de bits. En otras palabras, el emisor tendría que enviar una ráfaga de 40 millones de bits para trabajar a toda la velocidad hasta la llegada de la primera confirmación de recepción. Se requiere esta cantidad de bits para llenar el canal (en ambas direcciones). Ésta es la razón por la que una ráfaga de medio millón de bits sólo logra una eficiencia del 1.25%: es sólo el 1.25% de la capacidad del canal.

La conclusión aquí es que, para lograr un buen desempeño, la ventana del receptor debe tener cuando menos el tamaño del producto ancho de banda-retardo, y de preferencia ser un poco más grande, puesto que el receptor podría no responder instantáneamente. Para una línea transcontinental de gigabits se requieren cuando menos 5 megabytes para cada conexión.

Si la eficiencia es muy baja para el envío de un megabit, imagine lo que será al enviar unos cuantos cientos de bytes de una breve solicitud. A menos que pueda encontrarse otro uso para la

línea mientras el primer cliente espera una respuesta, una línea de gigabits no es mejor que una línea de megabits, sólo más cara.

Otro problema de desempeño que ocurre con las aplicaciones de tiempo crítico como audio y vídeo es la fluctuación. Un tiempo medio de transmisión corto no es suficiente. También se requiere una desviación estándar pequeña. El logro de un tiempo medio de transmisión corto con una desviación estándar pequeña requiere esfuerzos serios de ingeniería.

6.6.2 Medición del desempeño de las redes

Cuando una red tiene un desempeño pobre, sus usuarios frecuentemente se quejan con los operadores, exigiendo mejoras. Para mejorar el desempeño, los operadores deben primero determinar exactamente lo que ocurre. Para saberlo, los operadores deben efectuar mediciones. En esta sección veremos las mediciones de desempeño de las redes. El estudio siguiente se basa en el trabajo de Mogul (1993).

El ciclo usado para mejorar el desempeño de las redes contiene los siguientes pasos:

1. Medir los parámetros pertinentes y el desempeño de la red.
2. Tratar de entender lo que ocurre.
3. Cambiar un parámetro.

Estos pasos se repiten hasta que el desempeño sea lo bastante bueno o que quede claro que se han agotado todas las mejoras posibles.

Las mediciones pueden hacerse de muchas maneras y en muchos lugares (tanto físicos como en la pila de protocolos). El tipo de medición más básico es arrancar un temporizador al iniciar una actividad y medir el tiempo que tarda la actividad. Por ejemplo, saber el tiempo que toma la confirmación de recepción de una TPDU es una medición clave. Otras mediciones se hacen con contadores que registran la frecuencia con que ocurre un evento (por ejemplo, cantidad de TPDUs perdidas). Por último, con frecuencia nos interesa saber la cantidad de algo, como el número de bytes procesados durante cierto intervalo de tiempo.

La medición del desempeño y los parámetros de una red tiene muchos escollos potenciales. A continuación describimos algunos de ellos. Cualquier intento sistemático de medir el desempeño de una red debe tener cuidado de evitarlos.

Asegúrese que el tamaño de la muestra es lo bastante grande

No mida el tiempo de envío de una TPDU, sino repita la medición, digamos, un millón de veces y obtenga el promedio. Una muestra grande reducirá la incertidumbre de la media y la desviación estándar medidas. Esta incertidumbre puede calcularse usando fórmulas estadísticas estándar.

Asegúrese de que las muestras son representativas

Idealmente, la secuencia total de un millón de mediciones debería repetirse a horas del día y de la semana diferentes para ver el efecto de diferentes cargas del sistema sobre la cantidad medida. Por ejemplo, las mediciones de congestión sirven de poco si se toman en un momento en el que no hay congestión. A veces los resultados pueden ser contraintuitivos inicialmente, como la presencia de congestión intensa a las 10, 11, 13 y 14 horas, pero sin congestión al mediodía (cuando todos los usuarios están en el refrigerio).

Tenga cuidado al usar relojes de intervalos grandes

Los relojes de computadora funcionan sumando uno a un contador a intervalos regulares. Por ejemplo, un temporizador de milisegundos suma uno al contador cada 1 mseg. El uso de tal temporizador para medir un evento que tarda menos de 1 mseg es posible, pero requiere cuidado.

Por ejemplo, para medir el tiempo de envío de una TPDU, el reloj del sistema (digamos, en milisegundos) debe leerse al entrar en el código de capa de transporte, y nuevamente al salir. Si el tiempo de envío real de la TPDU es de 300 μseg, la diferencia entre las dos lecturas será 0 o 1, ambas cifras equivocadas. Sin embargo, si la medición se repite un millón de veces y se suman todas las mediciones y se dividen entre un millón, el tiempo medio tendrá una exactitud del orden de menos de 1 μseg.

Asegúrese de que no ocurre nada inesperado durante sus pruebas

Realizar mediciones en un sistema universitario el día en que tiene que entregarse un importante proyecto de laboratorio puede dar resultados diferentes a los que se podrían obtener el día siguiente. Del mismo modo, si un investigador ha decidido difundir una videoconferencia por la red mientras usted hace sus pruebas, sus resultados podrían alterarse. Es mejor ejecutar las pruebas en un sistema inactivo y crear la carga completa usted mismo, pero aun este enfoque tiene algunos escollos. Usted podría pensar que nadie usará la red a las 3 A.M., pero esa podría ser precisamente la hora en la que el programa automático de respaldos comienza a copiar todos los discos a videocinta. Es más, puede haber un tráfico pesado hacia sus fantásticas páginas del World Wide Web desde husos horarios distantes.

El caché puede arruinar las mediciones

Si quiere medir los tiempos de transferencia de archivos, la manera obvia de hacerlo es abrir un archivo grande, leerlo todo, cerrarlo y ver el tiempo que tarda. Luego se repetirá la medición muchas veces más para obtener un buen promedio. El problema es que el sistema puede manejar el archivo en caché, por lo que sólo la primera medición realmente comprende tráfico de red. Las demás son sólo lecturas del caché local. Los resultados de tales mediciones son esencialmente inservibles (a menos que se desee medir el desempeño del caché).

Con frecuencia puede evitarse el almacenamiento en caché simplemente desbordando el caché. Por ejemplo, si el caché es de 10 MB, el ciclo de prueba podría abrir, leer y cerrar dos archivos

de 10 MB en cada vuelta, en un intento por obligar a que la tasa de aciertos del caché sea de 0. Aun así, se recomienda cuidado a menos que esté completamente seguro de que entiende el algoritmo de almacenamiento en caché.

Los búferes pueden tener un efecto similar. Un programa de servicio de desempeño del TCP/IP bastante común ha llegado a informar que el UDP puede lograr un desempeño sustancialmente mayor al permitido por la línea física. ¿Por qué ocurre esto? Una llamada al UDP normalmente devuelve al control tan pronto como el *kernel* ha aceptado el mensaje y lo ha agregado a la cola de transmisión. Si hay suficiente espacio de búfer, cronometrar 1000 llamadas UDP no implica que todos los datos se han enviado. La mayoría de ellos podría estar aún en el *kernel*, pero el programa de servicio de desempeño piensa que se han transmitido todos.

Entienda lo que está midiendo

Al medir el tiempo de lectura de un archivo remoto, las mediciones dependen de la red, los sistemas operativos tanto en el cliente como en el servidor, las tarjetas de interfaz de hardware empleadas, sus controladores y otros factores. Si se tiene cuidado, finalmente se descubrirá el tiempo de transferencia de archivos para la configuración en uso. Si la meta es afinar esa configuración en particular, tales mediciones son adecuadas.

Sin embargo, si se están haciendo mediciones similares en tres sistemas diferentes a fin de seleccionar la tarjeta de interfaz de red a adquirir, sus resultados podrían desviarse por completo por el hecho de que uno de los controladores de la red realmente está mal y solamente está aprovechando el 10% del desempeño de la tarjeta.

Tenga cuidado con la extrapolación de los resultados

Suponga que hace mediciones con cargas de red simuladas que van desde 0 (en reposo) a 0.4 (40% de la capacidad), como lo indican los puntos de datos y la línea continua que los atraviesa en la figura 6-42. Puede ser tentador extrapolar linealmente, como lo indica la línea punteada. Sin embargo, muchos resultados de encolamiento comprenden un factor de $1/(1 - \rho)$, donde ρ es la carga, por lo que los valores verdaderos pueden parecerse más a la línea punteada, que se eleva más rápido que linealmente.

6.6.3 Diseño de sistemas para un mejor desempeño

La medición y los ajustes pueden con frecuencia mejorar considerablemente el desempeño, pero no pueden sustituir un buen diseño original. Una red mal diseñada puede mejorarse sólo hasta un límite. Más allá, tiene que rehacerse desde el principio.

En esta sección presentaremos algunas reglas empíricas basadas en la experiencia con muchas redes. Estas reglas se relacionan con el diseño del sistema, no sólo con el diseño de la red, ya que el software y el sistema operativo con frecuencia son más importantes que los enrutadores y las tarjetas de interfaz. La mayoría de estas ideas han sido del conocimiento común de los diseñadores

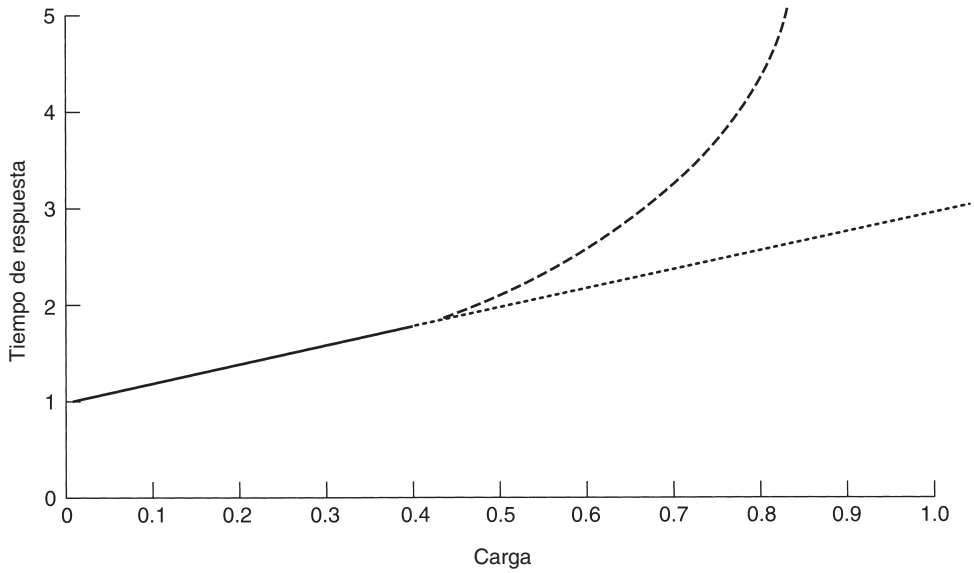


Figura 6-42. Respuesta en función de la carga.

de redes durante años y se han pasado verbalmente de generación en generación. Mogul fue el primero en postularlas explícitamente (1993); nuestro estudio sigue principalmente la secuencia del suyo. Otra fuente apropiada es (Metcalfe, 1993).

Regla #1: La velocidad de la CPU es más importante que la velocidad de la red

Una amplia experiencia ha demostrado que en casi todas las redes la sobrecarga de los sistemas operativos y protocolos domina el tiempo de utilización en el alambre. Por ejemplo, en teoría, el tiempo mínimo de una RPC en una red Ethernet es de 102 μseg, correspondiente a una solicitud mínima (64 bytes) seguida de una respuesta mínima (64 bytes). En la práctica, reducir la sobrecarga de software y conseguir el tiempo de RPC más cercano a 102 μseg es un logro considerable.

De la misma manera, el problema principal al operar a 1 Gbps es llevar los bits desde el búfer del usuario hasta la primera fibra a velocidad suficiente y lograr que la CPU receptora los procese tan rápidamente como entran. En pocas palabras, si se duplica la velocidad de la CPU, con frecuencia casi se puede duplicar la velocidad real de transporte. La duplicación de la capacidad de la red en muchos casos no tiene efecto, ya que el cuello de botella generalmente está en los *hosts*.

Regla #2: Reducir el número de paquetes para reducir la sobrecarga de software

El procesamiento de una TPDU tiene cierta cantidad de sobrecarga por TPDU (por ejemplo, procesamiento de encabezados) y cierta cantidad de procesamiento por byte (por ejemplo, procesar la suma de verificación). Al enviar 1 millón de bytes, la sobrecarga por byte es la misma sin

importar el tamaño de la TPDU. Sin embargo, el uso de TPDUs de 128 bytes implica 32 veces más sobrecarga por TPDU que el uso de TPDUs de 4 KB. Esta sobrecarga crece con rapidez.

Además de la sobrecarga de las TPDUs, hay una sobrecarga en las capas inferiores que se debe considerar. Cada paquete que llega causa una interrupción. En un procesador moderno con canalización, cada interrupción rompe la canalización de la CPU, interfiere con el caché, requiere un cambio en el contexto de administración de la memoria y obliga al almacenamiento de una cantidad de registros de CPU importante. Una reducción de *n* veces en las TPDUs enviadas reduce la sobrecarga de la interrupción y de los paquetes en un factor de *n*.

Esta observación es un argumento a favor de la recolección de una cantidad importante de datos antes de su transmisión, a fin de reducir las interrupciones en el otro lado. El algoritmo de Nagle y la solución de Clark al síndrome de la ventana tonta son intentos por lograr precisamente esto.

Regla #3: Reducir al mínimo las conmutaciones de contexto

Las conmutaciones de contexto (por ejemplo, del modo de *kernel* al modo de usuario) son mortales; pueden tener los mismos inconvenientes que las interrupciones, siendo la peor una larga serie de fallas de caché iniciales. Las conmutaciones de contexto pueden reducirse haciendo que el procedimiento de biblioteca que envía los datos los guarde en un búfer interno hasta tener una buena cantidad de ellos. De la misma manera, en el lado receptor las TPDUs de entrada pequeñas deben juntarse y pasarse al usuario como un bloque completo y no individualmente, para reducir al mínimo las conmutaciones de contexto.

En el mejor caso, un paquete entrante causa una conmutación de contexto del usuario actual al núcleo, y luego una conmutación al proceso receptor para darle los nuevos datos. Desgraciadamente, en muchos sistemas operativos ocurren conmutaciones de contexto adicionales. Por ejemplo, si el administrador de la red ejecuta un proceso especial en el espacio de usuario, un paquete entrante tenderá a causar una conmutación de contexto del usuario actual al *kernel*, luego otra del *kernel* al administrador de red, seguida de otra de regreso al *kernel* y, por último, una de éste al proceso receptor. Esta secuencia se muestra en la figura 6-43. Todas estas conmutaciones de contexto en cada paquete desperdician mucho tiempo de CPU y tienen un efecto devastador sobre el desempeño de la red.

Regla #4: Reducir al mínimo las copias

Peores que las múltiples conmutaciones de contexto son las múltiples copias. No es inusitado que un paquete entrante se copie tres o cuatro veces antes de entregarse la TPDU que contiene. Después de recibirse un paquete en la interfaz de la red en un búfer especial integrado a una tarjeta, es común que se copie en un búfer del *kernel*. De ahí se copia en el búfer de capa de red, luego en el búfer de la capa de transporte y, por último, en el proceso de aplicación receptor.

Un sistema operativo ingenioso copiará una palabra a la vez, pero no es raro que se requieran unas cinco instrucciones por palabra (carga, almacenamiento, incremento de un registro de índice,

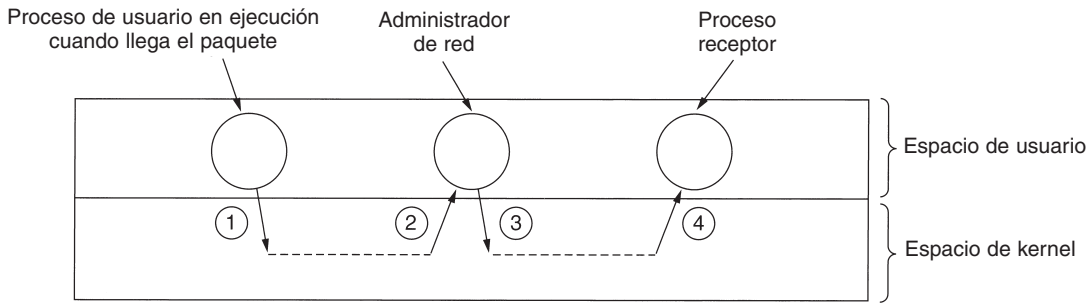


Figura 6-43. Cuatro conmutaciones de contexto para manejar un paquete con un administrador de red de espacio de usuario.

prueba de fin de datos y ramificación condicional). La elaboración de tres copias de cada paquete a cinco instrucciones por palabra de 32 bits copiada requiere 15/4 o cerca de cuatro instrucciones por byte copiado. En una CPU de 500 MIPS, una instrucción toma 2 nseg, de tal manera que cada byte necesita 8 nseg de tiempo de procesamiento o cerca de 1 nseg por bit, lo cual da una tasa máxima de 1 Gbps. Si incluimos la sobrecarga del procesamiento del encabezado, el manejo de interrupciones y las conmutaciones de contexto, podrían lograrse 500 Mbps, sin considerar el procesamiento real de los datos. Queda claro que es imposible manejar a toda velocidad una línea Ethernet de 10 Gbps.

De hecho, es probable que tampoco se pueda manejar a toda velocidad una línea de 500 Mbps. En el cálculo anterior hemos supuesto que una máquina de 500 MIPS puede ejecutar 500 millones de instrucciones por segundo. En realidad, las máquinas sólo pueden operar a tales velocidades si no hacen referencia a la memoria. Las operaciones de memoria con frecuencia son diez veces más lentas que las instrucciones registro a registro (es decir, 20 nseg/instrucción). Si en realidad 20 por ciento de las instrucciones hacen referencia a la memoria (es decir, son fallas de caché), lo cual es probable cuando entran en contacto con los paquetes entrantes, el tiempo de ejecución promedio de las instrucciones es de 5.6 nseg ($0.8 \times 2 + 0.2 \times 20$). Con cuatro instrucciones/byte, necesitamos 22.4 nseg/byte, o 2.8 nseg/bit), lo cual da cerca de 357 Mbps. Al factorizar 50 por ciento de sobrecarga da como resultado 178 Mbps. Observe que la asistencia de hardware no ayuda aquí. El problema es que el sistema operativo está ejecutando demasiadas copias.

Regla #5: Es posible comprar más ancho de banda, pero no un retardo menor

Las tres reglas que siguen tienen que ver con la comunicación, más que con el procesamiento del protocolo. La primera regla indica que, si se desea más ancho de banda, se puede comprar. La instalación de una segunda fibra junto a la primera duplica el ancho de banda, pero no hace nada para reducir el retardo. Para que el retardo sea más corto es necesario mejorar el software del protocolo, el sistema operativo o la interfaz de red. Incluso si se mejoran las tres cosas, el retardo no se reducirá si el cuello de botella es el tiempo de transmisión.

Regla #6: Evitar la congestión es mejor que recuperarse de ella

La vieja máxima de que más vale prevenir que lamentar ciertamente se aplica a la congestión en redes. Al congestionarse una red, se pierden paquetes, se desperdicia ancho de banda, se introducen retardos inútiles, y otras cosas. La recuperación requiere tiempo y paciencia; es mejor no tener que llegar a este punto. Evitar la congestión es como recibir una vacuna: duele un poco en el momento, pero evita algo que sería mucho más doloroso.

Regla #7: Evitar expiraciones del temporizador

Los temporizadores son necesarios en las redes, pero deben usarse con cuidado y deben reducirse al mínimo las expiraciones del temporizador. Al expirar un temporizador, lo común es que se repita una acción. Si realmente es necesario repetir la acción, que así sea, pero su repetición innecesaria es un desperdicio.

La manera de evitar el trabajo extra es tener cuidado de que los intervalos del temporizador sean más bien conservadores. Un temporizador que tarda demasiado en expirar agrega una pequeña cantidad de retardo extra a una conexión en el caso (improbable) de la pérdida de una TPDU. Un temporizador que expira cuando no debería consume tiempo de CPU valioso, desperdicia ancho de banda e impone una sobrecarga tal vez a docenas de enrutadores sin razón alguna.

6.6.4 Procesamiento rápido de las TPDUs

La moraleja de la historia anterior es que el obstáculo principal en las redes rápidas es el software de los protocolos. En esta sección veremos algunas maneras de acelerar este software. Para mayor información, véase (Clark y cols., 1989, y Chase y cols., 2001).

La sobrecarga de procesamiento de las TPDUs tiene dos componentes: la sobrecarga por TPDU y la sobrecarga por byte. Ambas deben combatirse. La clave para el procesamiento rápido de las TPDUs es separar el caso normal (transferencia de datos de un solo sentido) y manejarlo como caso especial. Aunque se necesita una secuencia de TPDUs especiales para entrar en el estado ESTABLISHED, una vez ahí el procesamiento de las TPDUs es directo hasta que un lado cierra la conexión.

Comencemos por examinar el lado emisor en el estado *ESTABLISHED* cuando hay datos por transmitir. Por claridad, supondremos que la entidad de transporte está en el *kernel*, aunque los mismos conceptos se aplican si es un proceso de espacio de usuario o una biblioteca en el proceso emisor. En la figura 6-44, el proceso emisor causa una interrupción en el *kernel* para ejecutar SEND. Lo primero que hace la entidad de transporte es probar si éste es el caso normal: el estado es *ESTABLISHED*, ningún lado está tratando de cerrar la conexión, se está enviando una TPDU normal (es decir, no fuera de banda) completa, y hay suficiente espacio de ventana disponible en el receptor. Si se cumplen todas las condiciones, no se requieren pruebas adicionales y puede seguirse la trayectoria rápida a través de la entidad de transporte emisora. Por lo general, esta ruta se toma la mayoría de las veces.

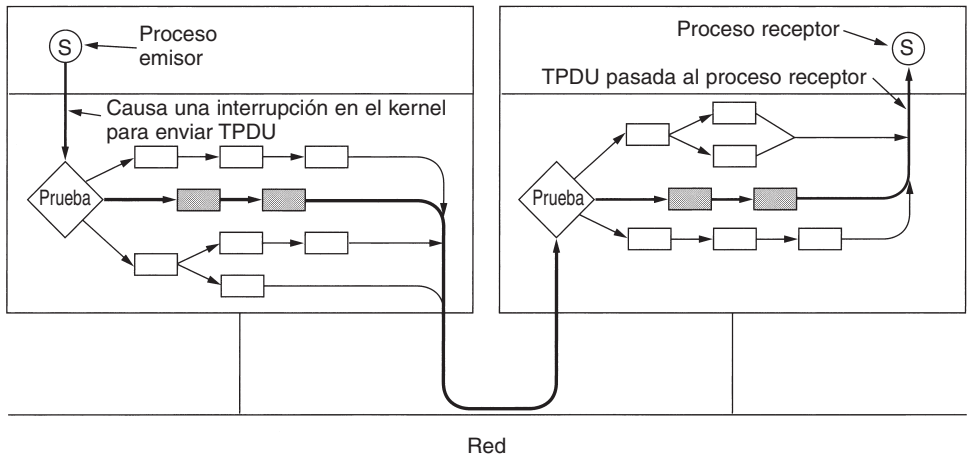


Figura 6-44. La trayectoria rápida del emisor al receptor se indica con una línea gruesa. Los pasos de procesamiento de esta trayectoria se muestran sombreados.

En el caso normal, los encabezados de las TPDUs de datos consecutivas son casi iguales. Para aprovechar este hecho, se almacena un encabezado prototipo en la entidad de transporte. Al principio de la trayectoria rápida, el encabezado se copia lo más rápidamente posible en un búfer de trabajo, palabra por palabra. Los campos que cambian de una TPDU a otra se sobreescriben en el búfer. Con frecuencia, estos campos se deducen fácilmente de las variables de estado, como el siguiente número de secuencia. A continuación se pasan a la capa de red un apuntador al encabezado completo de la TPDU más un apuntador a los datos de usuario. Aquí puede seguirse la misma estrategia (no se muestra en la figura 6-44). Por último, la capa de red entrega el paquete resultante a la capa de enlace de datos para su transmisión.

Como ejemplo del funcionamiento de este principio en la práctica, consideremos el TCP/IP. En la figura 6-45(a) se muestra el encabezado TCP. Los campos que son iguales en varias TPDUs consecutivas durante un flujo en un solo sentido aparecen sombreados. Todo lo que tiene que hacer la entidad de transporte emisora es copiar las cinco palabras del encabezado prototipo en el búfer de salida, actualizar el número de secuencia (copiándolo de una palabra en la memoria), calcular la suma de verificación e incrementar el número de secuencia en la memoria. Entonces puede entregar el encabezado y los datos a un procedimiento IP especial para enviar una TPDU normal máxima. El IP entonces copia su encabezado prototipo de cinco palabras [véase la figura 6-45(b)] en el búfer, llena el campo de *Identificación* y calcula su suma de verificación. El paquete ya está listo para transmitirse.

Veamos ahora el proceso de trayectoria rápida del lado receptor de la figura 6-44. El paso 1 es localizar el registro de conexión de la TPDU entrante. En el TCP, el registro de conexión puede almacenarse en una tabla de *hash* en la que alguna función sencilla de las dos direcciones IP y los dos puertos es la clave. Una vez localizado el registro de conexión, ambas direcciones y ambos puertos deben compararse para comprobar que se ha encontrado el registro correcto.

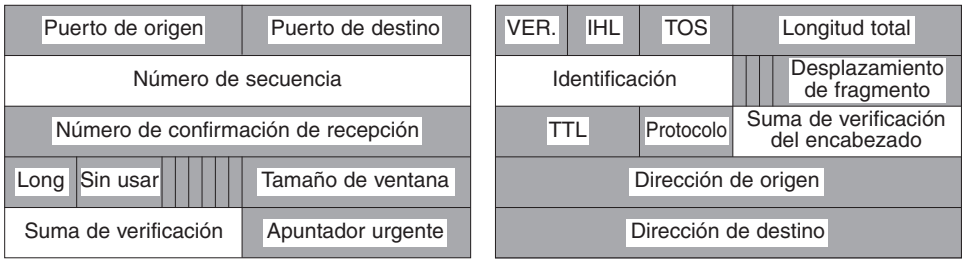


Figura 6-45. (a) Encabezado TCP. (b) Encabezado IP. En ambos casos, los campos sombreados se toman sin cambios del prototipo.

Una optimización que con frecuencia acelera aún más la búsqueda del registro de conexión es sencilla: mantener un apuntador al último registro usado y probar ese primero. Clark y cols. (1989) probó esto y observó una tasa de éxito mayor al 90%. Otras heurísticas de búsqueda se describen en (McKenney y Dove, 1992).

A continuación se revisa la TPDU para ver si es normal: el estado es *ESTABLISHED*, ninguno de los dos lados está tratando de cerrar la conexión, la TPDU es completa, no hay indicadores especiales encendidos y el número de secuencia es el esperado. Estas pruebas se llevan apenas unas cuantas instrucciones. Si se cumplen todas las condiciones, se invoca un procedimiento TCP especial de trayectoria rápida.

La trayectoria rápida actualiza el registro de la conexión y copia los datos en el espacio de usuario. Mientras copia, el procedimiento también calcula la suma de verificación, eliminado una pasada extra por los datos. Si la suma de verificación es correcta, se actualiza el registro de conexión y se devuelve una confirmación de recepción. El esquema general de hacer primero una comprobación rápida para ver si el encabezado es el esperado y tener un procedimiento especial para manejar ese caso se llama **predicción de encabezado.** Muchas implementaciones del TCP lo usan. Cuando esta optimización y todas las demás estudiadas en este capítulo se usan juntas, es posible conseguir que el TCP opere al 90% de la velocidad de una copia local de memoria a memoria, suponiendo que la red misma es lo bastante rápida.

Dos áreas en las que son posibles mejoras sustanciales del desempeño son el manejo de búferes y la administración de los temporizadores. El aspecto importante del manejo de búferes es evitar el copiado innecesario, como se explicó antes. La administración de los temporizadores es importante porque casi ninguno de los temporizadores expira; se ajustan para protegerse contra pérdidas de TPDUs, pero la mayoría de las TPDUs llegan correctamente, y sus confirmaciones de recepción también. Por tanto, es importante optimizar el manejo de los temporizadores para que casi nunca expiren.

Un esquema común consiste en usar una lista enlazada de eventos del temporizador ordenada por hora de expiración. La entrada inicial contiene un contador que indica la cantidad de pulsos de reloj que faltan para la expiración. Cada entrada subsecuente contiene un contador que indica el rezago en pulsos después de la entrada previa. Por tanto, si los temporizadores expiran a 3, 10 y 12 pulsos, respectivamente, los tres contadores son de 3, 7 y 2, respectivamente.

Después de cada pulso de reloj, se decrementa el contador del encabezado inicial. Cuando llega a cero, su evento se procesa y el siguiente elemento de la lista es ahora el inicial; su contador no necesita cambiarse. En este esquema, la inserción y eliminación de temporizadores son operaciones costosas, con tiempos de ejecución proporcionales a la longitud de la lista.

Puede usarse un enfoque más eficiente si el intervalo máximo del temporizador está limitado y se conoce por adelantado. Aquí puede utilizarse un arreglo llamado **rueda de temporización,** como se muestra en la figura 6-46. Cada ranura corresponde a un pulso de reloj. El tiempo actual en la figura es $T = 4$. Los temporizadores se programan para expirar 3, 10 y 12 pulsos más adelante. Si se establece un temporizador nuevo para expirar en siete pulsos, simplemente se crea una entrada en la ranura 11. Del mismo modo, si el temporizador establecido para $T + 10$ tiene que cancelarse, debe examinarse la lista que comienza en la ranura 14 para eliminar la entrada pertinente. Observe que el arreglo de la figura 6-46 no puede manejar temporizadores más allá de $T + 15$.

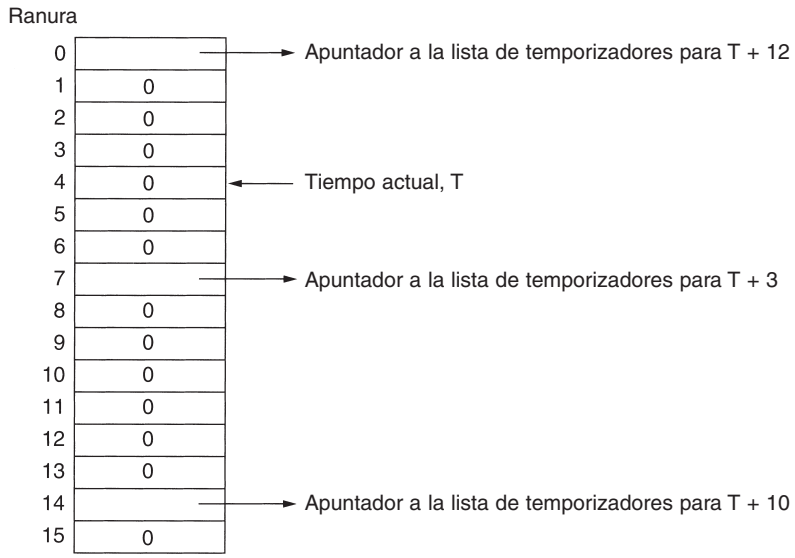


Figura 6-46. Rueda de temporización.

Cuando el reloj pulsa, el apuntador de tiempo actual avanza una ranura (circularmente). Si la entrada a la que ahora se apunta es diferente de cero, todos sus temporizadores se procesan. Se estudian muchas variaciones de la idea básica en (Varghese y Lauck, 1987).

6.6.5 Protocolos para redes de gigabits

Al inicio de la década de 1990 comenzaron a aparecer las redes de gigabits. La primera reacción de la gente fue usar en ellas los viejos protocolos, pero pronto surgieron varios problemas. En esta sección estudiaremos algunos de estos problemas y las medidas que están tomando los protocolos nuevos para resolverlos conforme surgen redes todavía más rápidas.

El primer problema es que muchos protocolos usan números de secuencia de 32 bits. En los principios de Internet, las líneas entre enrutadores fueron en su mayoría líneas rentadas de 56 kbps, así que un *host* transmitiendo a toda velocidad tomaba alrededor de una semana para dar vuelta a los números de secuencia. Para los diseñadores de TCP, 2^{32} era una aproximación muy buena al infinito porque había poco riesgo de que los paquetes viejos deambularan por la red una semana después de su transmisión. Con la Ethernet de 10 Mpbs, el tiempo para dar vuelta a los números de secuencia se redujo a 57 minutos, mucho más corto pero aún manejable. Con una Ethernet de 1 Gbps sirviendo datos en Internet, el tiempo para dar vuelta a los números de secuencia es cercano a 34 segundos, bastante abajo de los 120 seg de tiempo de vida máximo de un paquete en Internet. De buenas a primeras, 2^{32} ya no es una buena aproximación al infinito porque un emisor puede recorrer el espacio de secuencia aunque los paquetes viejos aún existan en la red. No obstante, el RFC 1323 proporciona una ventana de escape.

El origen de este problema es que muchos diseñadores de protocolos simplemente supusieron, tácitamente, que el tiempo requerido para consumir el espacio de secuencia completo excedería por mucho el tiempo de vida máximo de los paquetes. En consecuencia, no había necesidad de preocuparse por el problema de que duplicados viejos sobrevivieran aún al dar vuelta a los números de secuencia. A velocidades de gigabits, falla ese supuesto implícito.

Un segundo problema es que las velocidades de comunicación han mejorado con mucha mayor rapidez que las velocidades de cómputo. (Nota a los ingenieros en computación: ¡salgan a darles una paliza a los ingenieros en comunicaciones! Contamos con ustedes.) En los años 70, ARPANET operaba a 56 kbps y tenía computadoras que operaban a aproximadamente 1 MIPS. Los paquetes eran de 1008 bits, por lo que ARPANET era capaz de entregar unos 56 paquetes/seg. Con casi 18 mseg disponibles por paquete, un *host* podía darse el lujo de gastar 18,000 instrucciones en el procesamiento de un paquete. Claro que hacerlo requería todo el tiempo de CPU, pero podían ejecutarse 9000 instrucciones por paquete y aún así tener la mitad del tiempo de CPU libre para llevar a cabo tareas reales.

Compare estas cantidades con las computadoras modernas de 1000 MIPS que intercambian paquetes de 1500 bytes por una línea de gigabits. Los paquetes pueden entrar a una velocidad de más de 80,000 por segundo, por lo que el procesamiento de los paquetes debe completarse en 6.25 μseg si queremos reservar la mitad de la CPU para las aplicaciones. En 6.25 μseg, una computadora de 1000 MIPS puede ejecutar 6250 instrucciones, apenas 1/3 de lo que tenían disponibles los *hosts* de ARPANET. Es más, las instrucciones RISC modernas hacen menos por instrucción de lo que hacían las viejas instrucciones CISC, por lo que el problema es aún peor de lo que parece. En conclusión, hay menos tiempo disponible para el procesamiento de los protocolos del que había antes, por lo que los protocolos deben volverse más sencillos.

Un tercer problema es que el protocolo de retroceso n se desempeña mal en las líneas con un producto ancho de banda-retardo grande. Por ejemplo, considere una línea de 4000 km que opera a 1 Gbps. El tiempo de transmisión de ida y vuelta es de 40 mseg, durante el cual un emisor puede enviar 5 megabytes. Si se detecta un error, pasarán 40 mseg antes de que el emisor se entere. Si se usa el retroceso n, el emisor tendrá que transmitir no sólo el paquete erróneo, sino también los 5 megabytes de paquetes que llegaron después. Evidentemente, éste es un desperdicio masivo de recursos.

Un cuarto problema es que las líneas de gigabits son fundamentalmente diferentes de las líneas de megabits en el sentido de que las líneas largas de gigabits están limitadas por el retardo en

lugar del ancho de banda. En la figura 6-47 mostramos el tiempo que tarda la transferencia de un archivo de 1 megabit a 4000 km con varias velocidades de transmisión. A velocidades de hasta 1 Mbps, el tiempo de transmisión está dominado por la velocidad con que pueden enviarse los bits. A 1 Gbps, el retardo de ida y vuelta de 40 mseg domina al 1 mseg que se tarda la inserción de los bits en la fibra. Aumentos mayores del ancho de banda apenas tienen algún efecto.

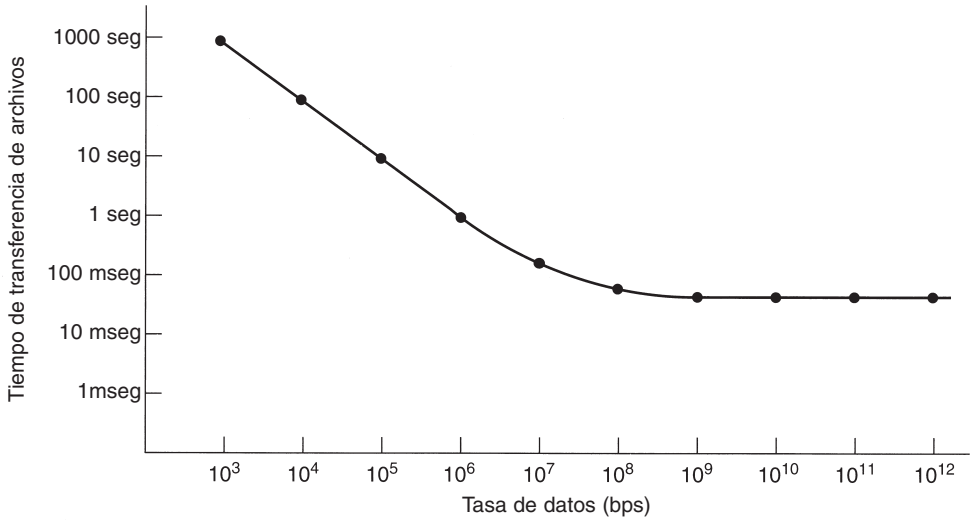


Figura 6-47. Tiempo de transferencia y confirmación de recepción de un archivo de 1 megabit a través de una línea de 4000 km.

La figura 6-47 tiene implicaciones desafortunadas para los protocolos de red; indica que los protocolos de parada y espera, como el RPC, tienen un límite superior de desempeño inherente. Este límite lo establece la velocidad de la luz. Ningún progreso tecnológico en la óptica mejorará la situación (aunque ayudarían nuevas leyes de la física).

Un quinto problema que vale la pena mencionar no es tecnológico ni de protocolo, como los otros, sino resultado de las aplicaciones nuevas. En pocas palabras, en muchas aplicaciones de gigabits, como las de multimedia, la variación en los tiempos de llegada de los paquetes es tan importante como el retardo medio mismo. Una tasa de entrega lenta pero uniforme con frecuencia es preferible a una rápida pero variable.

Pasemos ahora de los problemas a las maneras de resolverlos. Primero haremos algunas observaciones generales, luego veremos los mecanismos de protocolos, la disposición de los paquetes y el software de los protocolos.

El principio básico que deben aprender de memoria todos los diseñadores de redes de gigabits es:

Diseñar pensando en la velocidad, no en la optimización del ancho de banda.

Los protocolos viejos con frecuencia se diseñaban tratando de reducir al mínimo la cantidad de bits en el alambre, comúnmente usando campos pequeños y empacándolos en bytes y palabras.

Hoy día hay más que suficiente ancho de banda. El procesamiento del protocolo es el problema, por lo que los protocolos deberían diseñarse para reducirlo al mínimo. Los diseñadores del IPv6 entendieron claramente este principio.

Una manera tentadora de acelerar el procedimiento es construir interfaces de red rápidas en hardware. Lo malo de esta estrategia es que, a menos que el protocolo sea excesivamente sencillo, "hardware" simplemente significa una tarjeta con una segunda CPU y su propio programa. Para asegurar que el coprocesador de la red sea más económico que la CPU principal, con frecuencia se usa un chip más lento. La consecuencia de este diseño es que una buena parte del tiempo la CPU principal (rápida) queda en espera de que la segunda CPU (lenta) haga el trabajo crítico. Es un mito pensar que la CPU principal tiene otras tareas que hacer mientras espera. Es más, cuando dos CPUs de propósito general se comunican, pueden ocurrir condiciones de competencia, por lo que se requieren protocolos complejos entre los dos procesadores para sincronizarlos correctamente. Por lo general, el mejor enfoque es hacer que los protocolos sean sencillos y dejar que la CPU principal realice el trabajo.

Veamos ahora el asunto de la retroalimentación en los protocolos de alta velocidad. Debido al ciclo de retardo (relativamente) largo, debe evitarse la retroalimentación: la señalización del receptor al emisor tarda demasiado. Un ejemplo de retroalimentación es el control de la tasa de transmisión mediante un protocolo de ventana corrediza. Para evitar los retardos (largos) inherentes en el envío de actualizaciones de ventana del receptor al emisor, es mejor usar un protocolo basado en la tasa. En tal protocolo, el emisor puede enviar todo lo que quiera, siempre y cuando no envíe a mayor velocidad que cierta tasa acordada de antemano entre el emisor y el receptor.

Un segundo ejemplo de retroalimentación es el algoritmo de arranque lento de Jacobson. Este algoritmo efectúa múltiples sondeos para saber qué tanto puede manejar la red. Con las redes de alta velocidad, efectuar media docena de sondeos pequeños para ver la respuesta de la red es un desperdicio de ancho de banda enorme. Un esquema más eficiente es hacer que el emisor, el receptor y la red reserven los recursos necesarios en el momento de establecer la conexión. La reservación de recursos por adelantado también tiene la ventaja de facilitar la reducción de la fluctuación. En pocas palabras, la elevación de las velocidades inevitablemente empuja el diseño hacia la operación orientada a la conexión, o a algo muy parecido. Por supuesto, si el ancho de banda se incrementa en el futuro en forma tal que a nadie le importe desperdiciarlo, las reglas de diseño se tornarán muy diferentes.

La disposición de los paquetes es una consideración importante en las redes de gigabits. El encabezado debería contener la menor cantidad de campos posible, a fin de reducir el tiempo de procesamiento; estos campos deben ser lo bastante grandes como para cumplir su cometido, y estar alineados con los límites de palabra para facilitar su procesamiento. En este contexto, "bastante grandes" significa que no ocurren problemas como números de secuencia que dan vuelta mientras existen aún paquetes viejos, receptores incapaces de anunciar suficiente espacio de ventana porque el campo de ventana es demasiado pequeño, etcétera.

El encabezado y los datos deben tener sumas de verificación aparte, por dos razones. Primera, hacer posible la obtención de la suma de verificación del encabezado, pero no de los datos. Segunda, comprobar que el encabezado esté correcto antes de comenzar a copiar los datos en el espacio de usuario. Es deseable calcular la suma de verificación de los datos al mismo tiempo que

se copian los datos en el espacio de usuario, pero si el contenido del encabezado está equivocado, el copiado podría hacerse a un proceso equivocado. Para evitar un copiado incorrecto pero permitir que se calcule la suma de verificación de los datos durante el copiado, es esencial que las dos sumas de verificación estén separadas.

El tamaño máximo de los datos debe ser lo bastante grande para permitir una operación eficiente inclusive ante retardos largos. También, cuanto mayor es el tamaño del bloque de datos, menor es la fracción del ancho de banda total dedicada a los encabezados. 1500 bytes es demasiado pequeño.

Otra característica valiosa es la capacidad de enviar una cantidad normal de datos junto con la solicitud de conexión. Así, puede ahorrarse el tiempo de un viaje de ida y vuelta.

Por último, es importante decir algo sobre el software de los protocolos. Una idea clave es concentrarse en el caso exitoso. Muchos protocolos viejos tienden a remarcar lo que se debe hacer cuando algo falla (por ejemplo, cuando se pierde un paquete). Para lograr que los protocolos operen rápidamente, el diseñador debería enfocarse en reducir al mínimo el tiempo de procesamiento cuando todo funciona bien. La reducción al mínimo del tiempo de procesamiento cuando ocurren errores es secundaria.

Un segundo asunto relacionado con el software es la minimización del tiempo de copiado. Como vimos antes, el copiado de datos con frecuencia es la fuente principal de sobrecarga. Idealmente, el hardware debe poner en la memoria cada paquete entrante como un bloque contiguo de datos. El software entonces debe copiar este paquete en el búfer del usuario con un solo copiado de bloque. Dependiendo del modo de funcionamiento del caché, inclusive puede ser deseable evitar un ciclo de copiado. En otras palabras, para copiar 1024 palabras, la manera más rápida podría ser la ejecución de 1024 instrucciones MOVE una tras otra (o 1024 pares cargar-almacenar). La rutina de copiado es tan crítica que debe desarrollarse cuidadosamente en código ensamblador, a menos que haya una manera de lograr que el compilador produzca el código óptimo.

6.7 RESUMEN

La capa de transporte es la clave para entender los protocolos en capas. Esta capa proporciona varios servicios, siendo el más importante un flujo de bytes punto a punto, confiable y orientado a al conexión desde el emisor hasta el receptor. Se accede a ella a través de primitivas de servicio que permiten establecer, usar y liberar conexiones. Una interfaz común de la capa de transporte es la que proporcionan los *sockets* de Berkeley.

Los protocolos de transporte deben ser capaces de administrar conexiones a través de redes no confiables. El establecimiento de conexiones se complica por la existencia de paquetes duplicados retardados que pueden reaparecer en momentos inoportunos. Para manejarlos, se requieren acuerdos de tres vías para establecer las conexiones. La liberación de una conexión es más sencilla que su establecimiento, pero aun así no está exento de complejidad debido al problema de los dos ejércitos.

Aun si la capa de red es completamente confiable, la capa de transporte tiene bastante trabajo: debe manejar todas las primitivas de servicio, administrar conexiones y temporizadores y asignar y usar créditos.

Internet tiene dos protocolos de transporte principales: UDP y TCP. UDP es un protocolo no orientado a la conexión que es principalmente una envoltura para los paquetes IP con la característica adicional de multiplexar y desmultiplexar diversos procesos utilizando una sola dirección IP. UDP puede emplearse para interacciones cliente-servidor, por ejemplo, utilizando RPC. También puede emplearse para construir protocolos en tiempo real como RTP.

El protocolo de transporte principal de Internet es TCP. Proporciona un flujo de bytes bidireccional y confiable. Utiliza un encabezado de 20 bytes en todos los segmentos. Los enrutadores de Internet pueden fragmentar los segmentos, por lo que los *hosts* deben estar preparados para reensamblarlos. Se ha invertido mucho trabajo en optimizar el desempeño del TCP, mediante algoritmos de Nagle, Clark, Jacobson, Karn, entre otros. Los enlaces inalámbricos agregan una variedad de complicaciones al TCP. El TCP para transacciones es una extensión del TCP que maneja las interacciones cliente-servidor con un número reducido de paquetes.

El desempeño de las redes generalmente es dominado por la sobrecarga de procesamiento de los protocolos y las TPDUs, y esta situación empeora a mayores velocidades. Los protocolos deberían diseñarse para reducir al mínimo la cantidad de TPDUs, de conmutaciones de contexto y de veces que se copia cada TPDU. En las redes de gigabits se requieren protocolos sencillos.

PROBLEMAS

1. En nuestras primitivas de ejemplo de la figura 6-2, LISTEN es una llamada bloqueadora. ¿Es estrictamente necesario esto? De no serlo, explique cómo debe usarse una primitiva no bloqueadora. ¿Qué ventaja tendría esto respecto al esquema descrito en el texto?

2. En el modelo de la figura 6-4, se supone que la capa de red puede perder paquetes y, por tanto, su recepción se debe confirmar individualmente. Suponga que la capa de red es 100% confiable y que nunca pierde paquetes. ¿Qué cambios, si acaso, se necesitarán en la figura 6-4?

3. En las dos partes de la figura 6-6 hay un comentario de que los valores de *SERVER_PORT* deben ser los mismos en el cliente y en el servidor. ¿Por qué es tan importante?

4. Suponga que el esquema operado por reloj para generar números de secuencia iniciales se usa con un contador de reloj de 15 bits de ancho. El reloj pulsa una vez cada 100 mseg, y el tiempo de vida máximo de un paquete es de 60 seg. ¿Con qué frecuencia ocurre la resincronización
(a) en el peor caso?
(b) cuando los datos consumen 240 números de secuencia/min?

5. ¿Por qué tiene que ser el tiempo de vida máximo de paquete, *T*, lo bastante grande para asegurar que han desaparecido no sólo el paquete, sino también sus confirmaciones de recepción?

6. Imagine que se usa un acuerdo de dos vías en lugar de uno de tres vías para establecer las conexiones. En otras palabras, no se requiere el tercer mensaje. ¿Son posibles ahora los bloqueos irreversibles? Dé un ejemplo o demuestre que no pueden existir.

7. Imagine un problema de *n*-ejércitos generalizado, en el que el acuerdo de dos de los ejércitos azules es suficiente para la victoria. ¿Existe un protocolo que permita ganar a los azules?

8. Considere el problema de la recuperación después de una caída del *host* (es decir, la figura 6-18). Si el intervalo entre la escritura y el envío de una confirmación de recepción, o viceversa, puede hacerse relativamente pequeño, ¿cuáles son las mejores estrategias emisor-receptor para reducir al mínimo la posibilidad de una falla del protocolo?

9. ¿Son posibles los bloqueos irreversibles con la entidad de transporte descrita en el texto (figura 6-20)?

10. Por curiosidad, el implementador de la entidad de transporte de la figura 6-20 ha decidido incluir contadores en el procedimiento *sleep* para obtener estadísticas sobre el arreglo *conn*. Entre éstas están la cantidad de conexiones en cada uno de los siete estados posibles, n_i ($i = 1,..., 7$). Tras escribir un enorme programa en FORTRAN para analizar los datos, nuestro implementador descubrió que la relación $\Sigma n_i = MAX\_CONN$ parece ser verdadera siempre. ¿Hay otras invariantes en las que intervengan sólo estas siete variables?

11. ¿Qué ocurre cuando el usuario de la entidad de transporte dada en la figura 6-20 envía un mensaje de longitud cero? Explique el significado de su respuesta.

12. Por cada evento que puede ocurrir en la entidad de transporte de la figura 6-20, indique si es legal o no cuando el usuario está durmiendo en el estado *sending*.

13. Explique las ventajas y desventajas de los créditos en comparación con los protocolos de ventana corrediza.

14. ¿Por qué existe el UDP? ¿No habría bastado con permitir que los procesos de usuario enviaran paquetes IP en bruto?

15. Considere un protocolo de nivel de aplicación simple construido encima de UDP que permite a un cliente recuperar un archivo desde un servidor remoto que reside en una dirección bien conocida. El cliente primero envía una solicitud con el nombre del archivo, y el servidor responde con una secuencia de paquetes de datos que contienen diferentes partes del archivo solicitado. Para asegurar la confiabilidad y una entrega en secuencia, el cliente y el servidor utilizan un protocolo de parada y espera. Ignorando el aspecto de desempeño obvio, ¿ve usted un problema con este protocolo? Piense cuidadosamente en la posibilidad de la caída de los procesos.

16. Un cliente envía una solicitud de 128 bytes a un servidor localizado a 100 km de distancia a través de una fibra óptica de 1 gigabit. ¿Cuál es la eficiencia de la línea durante la llamada a procedimiento remoto?

17. Considere nuevamente la situación del problema anterior. Calcule el tiempo de respuesta mínimo posible para la línea de 1 Gbps y para una de 1 Mbps. ¿Qué conclusión puede obtener?

18. Tanto UDP como TCP utilizan números de puerto para identificar la entidad de destino cuando entregan un paquete. Dé dos razones por las cuales estos protocolos inventaron un nuevo ID abstracto (números de puerto), en lugar de utilizar IDs de proceso, que ya existían cuando se diseñaron estos protocolos.

19. ¿Cuál es el tamaño total de la MTU mínima de TCP, incluyendo la sobrecarga de TCP e IP pero no la de la capa de enlace de datos?

20. La fragmentación y el reensamble de datagramas son manejados por IP y son transparentes para TCP. ¿Esto significa que TCP no tiene que preocuparse porque los datos lleguen en el orden equivocado?

21. RTP se utiliza para transmitir audio con calidad de CD, el cual crea un par de muestras de 16 bits, 44,100 veces/seg, una muestra por cada uno de los canales de estéreo. ¿Cuántos paquetes por segundo debe transmitir RTP?

22. ¿Sería posible colocar el código RTP en el *kernel* del sistema operativo junto con el código UDP? Explique su respuesta.

23. Un proceso del *host* 1 se ha asignado al puerto *p*, y un proceso del *host* 2 se ha asignado al puerto *q*. ¿Es posible que haya dos o más conexiones TCP entre estos dos puertos al mismo tiempo?

24. En la figura 6-29 vimos que además del campo *Confirmación de recepción* de 32 bits, hay un bit *ACK* en la cuarta palabra. ¿Esto agrega realmente algo? ¿Por qué sí o por qué no?

25. La máxima carga útil de un segmento TCP son 65,495 bytes. ¿Por qué se eligió ese extraño número?

26. Describa dos formas de entrar en el estado *SYN RCVD* de la figura 6-33.

27. Indique una desventaja potencial del uso del algoritmo de Nagle en una red muy congestionada.

28. Considere el efecto de usar arranque lento en una línea con un tiempo de ida y vuelta de 10 mseg sin congestionamientos. La ventana receptora es de 24 KB y el tamaño máximo de segmento es de 2 KB. ¿Cuánto tiempo pasará antes de poder enviar la primera ventana completa?

29. Suponga que la ventana de congestionamiento del TCP está ajustada a 18 KB y que ocurre una expiración del temporizador. ¿Qué tan grande será la ventana si las siguientes cuatro ráfagas de transmisiones tienen éxito? Suponga que el tamaño máximo de segmento es de 1 KB.

30. Si el tiempo de ida y vuelta del TCP, *RTT*, actualmente es de 30 mseg y las siguientes confirmaciones de recepción llegan después de 26, 32 y 24 mseg, respectivamente, ¿cuál es la nueva estimación de *RTT* utilizando el algoritmo de Jacobson? Use $\alpha = 0.9$.

31. Una máquina TCP envía ventanas de 65,535 bytes por un canal de 1 Gbps que tiene un retardo de 10 mseg en un solo sentido. ¿Cuál es la velocidad real de transporte máxima que se puede lograr? ¿Cuál es la eficiencia de la línea?

32. ¿Cuál es la velocidad máxima de línea a la que un *host* puede enviar cargas útiles TCP de 1500 bytes con un tiempo de vida máximo de paquete de 120 seg sin que los números de secuencia den vuelta? Tome en cuenta la sobrecarga TCP, IP y Ethernet. Suponga que las tramas Ethernet se pueden enviar de manera continua.

33. En una red que tiene un tamaño máximo de TPDU de 128 bytes, un tiempo de vida máximo de TPDU de 30 seg, y un número de secuencia de 8 bits, ¿cuál es la tasa máxima de datos por conexión?

34. Suponga que usted está midiendo el tiempo para recibir una TPDU. Cuando ocurre una interrupción, lee el reloj del sistema en milisegundos. Cuando la TPDU se ha procesado por completo, lee nuevamente el reloj. Mide 0 mseg 270,000 veces y 1 mseg 730,000 veces. ¿Cuánto tarda en recibir una TPDU?

35. Una CPU ejecuta instrucciones a la tasa de 1000 MIPS. Los datos pueden copiarse 64 bits a la vez, y cada palabra toma diez instrucciones para copiarse. Si un paquete entrante tiene que copiarse cuatro veces, ¿puede este sistema manejar una línea de 1 Gbps? Por simplicidad, suponga que todas las instrucciones, incluso aquellas que leen o escriben en la memoria, se ejecutan a la tasa total de 1000 MIPS.

36. Para resolver el problema de los números de secuencia repetidos mientras que los paquetes anteriores aún existen, se podrían utilizar números de secuencia de 64 bits. Sin embargo, teóricamente, una fibra óptica puede ejecutarse a 75 Tbps. ¿Cuál es el tiempo de vida máximo de paquete que se requiere para asegurarse de que las futuras redes de 75 Tbps no tengan problemas de números de secuencia que den vuelta incluso con los números de secuencia de 64 bits? Suponga que cada byte tiene su propio número de secuencia, al igual que TCP.

37. Mencione una ventaja de RPC sobre UDP en comparación con el TCP para transacciones. Mencione una ventaja del T/TCP sobre RPC.

38. En la figura 6-40(a) vimos que para completar el RCP se necesitan 9 paquetes. ¿Hay alguna circunstancia en la que se necesiten exactamente 10 paquetes?

39. En la sección 6.6.5 calculamos que una línea de gigabits descarga 80,000 paquetes/seg en el *host*, y éste dispone de sólo 6250 instrucciones para procesarlos, lo que deja la mitad del tiempo de CPU para las aplicaciones. Para este cálculo se tomó un tamaño de paquete de 1500 bytes. Rehaga el cálculo para un paquete de tamaño ARPANET (128 bytes). En ambos casos, suponga que los tamaños de paquete dados incluyen toda la sobrecarga.

40. Para una red de 1 Gbps que opera sobre 4000 km, el retardo es el factor limitante, no el ancho de banda. Considere una MAN con un promedio de 20 km entre el origen y el destino. ¿A qué tasa de datos el retardo de ida y vuelta debido a la velocidad de la luz iguala el retardo de transmisión para un paquete de 1 KB?

41. Calcule el producto de retardo de ancho de banda para las siguientes redes: (1) T1 (1.5 Mbps), (2) Ethernet (10 Mbps), (3) T3 (45 Mbps) y (4) STS-3 (155 Mbps). Suponga un RTT de 100 mseg. Recuerde que un encabezado TCP tiene 16 bits reservados para el tamaño de ventana. ¿Cuáles son sus implicaciones a la luz de sus cálculos?

42. ¿Cuál es el producto de retardo de ancho de banda para un canal de 50 Mbps en un satélite geoestacionario? Si todos los paquetes son de 1500 bytes (incluyendo la sobrecarga), ¿qué tan grande debería ser la ventana en los paquetes?

43. El servidor de archivos de la figura 6-6 está muy lejos de ser perfecto y se le pueden hacer algunas mejoras. Realice las siguientes modificaciones.
(a) Dé al cliente un tercer argumento que especifique un rango de bytes.
(b) Agregue un indicador–w al cliente que permita que el archivo se escriba en el servidor.

44. Modifique el programa de la figura 6-20 para que realice recuperación de errores. Agregue un nuevo tipo de paquetes, *reset*, que pueda llegar después de que los dos lados abran una conexión pero que ninguno la cierre. Este evento, que sucede de manera simultánea en ambos lados de la conexión, significa que cualquier paquete que estuviera en tránsito se ha entregado o destruido, pero de cualquier modo ya no está en la subred.

45. Escriba un programa que simule manejo de búfer en una entidad de transporte, utilizando una ventana corrediza para el control de flujo en lugar del sistema de créditos de la figura 6-20. Deje que los procesos de las capas superiores abran conexiones, envíen datos y cierren las conexiones de manera aleatoria. Por sencillez, haga que los datos viajen sólo de la máquina *A* a la máquina *B*. Experimente con estrategias de asignación de búfer diferentes en *B*, como dedicar búferes a conexiones específicas y establecer un grupo de búferes común, y mida el rendimiento total alcanzado por cada estrategia.

46. Diseñe e implemente un sistema de salones de conversación que permita conversar a múltiples grupos de usuarios. Un coordinador del salón reside en una dirección bien conocida de red, utiliza UDP para comunicarse con los clientes del salón, configura los servidores del salón para cada sesión de conversación y mantiene un directorio de sesiones de conversación. Hay un servidor de conversación por sesión. Un servidor de este tipo utiliza TCP para comunicarse con los clientes. Un cliente de conversación permite que los usuarios inicien, se unan y dejen una sesión de conversación. Diseñe e implemente el código del coordinador, del servidor y del cliente.

7

LA CAPA DE APLICACIÓN

Habiendo concluido todos los preliminares, ahora llegamos a la capa de aplicación, donde pueden encontrarse todas las aplicaciones interesantes. Las capas por debajo de la de aplicación están ahí para proporcionar transporte confiable, pero no hacen ningún trabajo verdadero para los usuarios. En este capítulo estudiaremos algunas aplicaciones de red reales.

Sin embargo, aun en la capa de aplicación se necesitan protocolos de apoyo que permitan el funcionamiento de las aplicaciones reales. De manera acorde, veremos uno de éstos antes de que comencemos con las aplicaciones mismas. El sujeto en cuestión es el DNS, que maneja la asignación de nombres dentro de Internet. Después examinaremos tres aplicaciones reales: correo electrónico, World Wide Web y, finalmente, multimedia.

7.1 DNS—EL SISTEMA DE NOMBRES DE DOMINIO

Aunque en teoría los programas pueden hacer referencia a *hosts*, buzones de correo y otros recursos mediante sus direcciones de red (por ejemplo, IP), a las personas se les dificulta recordar estas direcciones. Además, enviar correo electrónico a *tana@128.111.24.41* significa que si el ISP u organización de Tana mueve el servidor de correo electrónico a una máquina diferente, la cual tiene una dirección IP diferente, la dirección de correo electrónico de Tana tiene que cambiar. Debido a esto, se introdujeron los nombres ASCII, con el fin de separar los nombres de máquina de las direcciones de máquina. De esta manera, la dirección de Tana podría ser algo como *tana@art.ucsb.edu*. Sin embargo, la red misma sólo comprende direcciones numéricas, por lo que se requieren algunos mecanismos para convertir las cadenas ASCII a direcciones de red. En las siguientes secciones analizaremos la forma en que se logra esta correspondencia en Internet.

Hace mucho, en los tiempos de ARPANET, sólo había un archivo, *hosts.txt*, en el que se listaban todos los *hosts* y sus direcciones IP. Cada noche, todos los *hosts* obtenían este archivo del sitio en el que se mantenía. En una red conformada por unas cuantas máquinas grandes de tiempo compartido, este método funcionaba razonablemente bien.

Sin embargo, cuando miles de estaciones de trabajo se conectaron a la red, todos se dieron cuenta de que este método no podría funcionar eternamente. Por una parte, el tamaño del archivo crecería de manera considerable. Un problema aún más importante era que ocurrirían conflictos constantes con los nombres de los *hosts* a menos de que dichos nombres se administraran centralmente, algo impensable en una red internacional enorme. Para resolver estos problemas, se inventó el **DNS (Sistema de Nombres de Dominio).**

La esencia del DNS es la invención de un esquema de nombres jerárquico basado en dominios y un sistema de base de datos distribuido para implementar este esquema de nombres. El DNS se usa principalmente para relacionar los nombres de *host* y destinos de correo electrónico con las direcciones IP, pero también puede usarse con otros fines. El DNS se define en los RFCs 1034 y 1035.

Muy brevemente, la forma en que se utiliza el DNS es la siguiente. Para relacionar un nombre con una dirección IP, un programa de aplicación llama a un procedimiento de biblioteca llamado **resolvedor,** y le pasa el nombre como parámetro. En la figura 6.6 vimos un ejemplo de un resolvedor, *gethostbyname*. El resolvedor envía un paquete UDP a un servidor DNS local, que después busca el nombre y devuelve la dirección IP al resolvedor, que entonces lo devuelve al solicitante. Una vez que tiene la dirección IP, el programa puede establecer una conexión TCP con el destino, o enviarle paquetes UDP.

7.1.1 El espacio de nombres del DNS

La administración de un grupo grande y continuamente cambiante de nombres es un problema nada sencillo. En el sistema postal, la administración de nombres se hace requiriendo letras para especificar (implícita o explícitamente) el país, estado o provincia, ciudad y calle, y dirección del destinatario. Con este tipo de direccionamiento jerárquico, no hay confusión entre el Marvin Anderson de Main St., en White Plains, N.Y. y el Marvin Anderson de Main St., en Austin, Texas. El DNS funciona de la misma manera.

Conceptualmente, Internet se divide en 200 **dominios** de nivel superior, cada uno de los cuales abarca muchos *hosts*. Cada dominio se divide en subdominios, los cuales, a su vez, también se dividen, y así sucesivamente. Todos estos dominios pueden representarse mediante un árbol, como se muestra en la figura 7-1. Las hojas del árbol representan los dominios que no tienen subdominios (pero que, por supuesto, contienen máquinas). Un dominio de hoja puede contener un solo *host*, o puede representar a una compañía y contener miles de *hosts*.

Los dominios de nivel superior se dividen en dos categorías: genéricos y de país. Los dominios genéricos originales son *com* (*comercial*), *edu* (instituciones educativas), *gov* (el gobierno federal de Estados Unidos), *int* (ciertas organizaciones internacionales), *mil* (las fuerzas armadas de Estados Unidos), *net* (proveedores de red) y *org* (organizaciones no lucrativas). Los dominios de país incluyen una entrada para cada país, como se define en la ISO 3166.

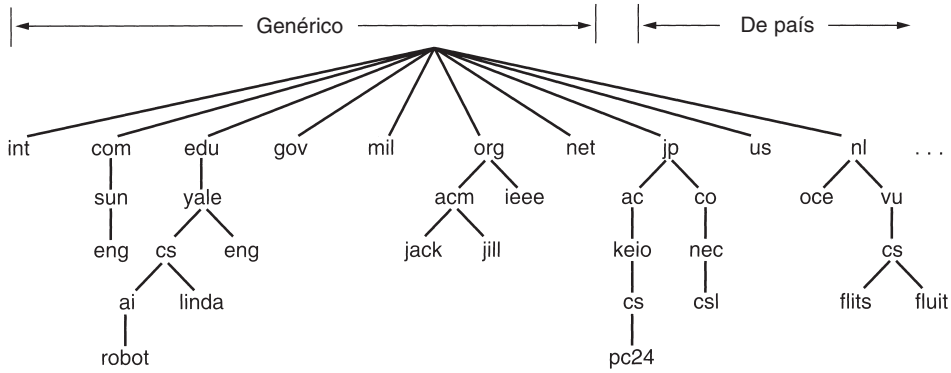


Figura 7-1. Parte del espacio de nombres de dominio de Internet.

En noviembre de 2000, ICANN aprobó cuatro nuevos dominios de nivel superior y propósito general: *biz* (negocios), *info* (información), *name* (nombres de personas) y *pro* (profesiones, como doctores y abogados). Además, se introdujeron otros tres nombres de dominio especializados de nivel superior a solicitud de ciertas industrias. Éstos son: *aero* (industria aeroespacial), *coop* (cooperativas) y *museum* (museos). En el futuro se agregarán otros dominios de nivel superior.

Como nota al margen, conforme Internet se vuelva más comercial, también se volverá más contenciosa. Por ejemplo, considere a *pro*. Estaba destinado para los profesionales certificados, pero, ¿quién es un profesional? y, ¿certificado por quién? Los doctores y abogados claramente son profesionales, pero, ¿qué pasa con los fotógrafos independientes, maestros de piano, magos, plomeros, barberos, exterminadores, artistas de tatuajes, soldados y prostitutas? ¿Estas ocupaciones son profesionales y, por lo tanto, candidatos a los dominios *pro*? De ser así, ¿quién certifica a los practicantes individuales?

En general, obtener un dominio de segundo nivel, como *nombre-de-compañía.*com, es fácil. Simplemente se necesita ir con el registrador del dominio de nivel superior correspondiente (*com*, en este caso) para ver si el nombre deseado está disponible y si no es la marca registrada de alguien más. Si no hay problemas, el solicitante paga una pequeña cuota anual y obtiene el nombre. En la actualidad, casi todas las palabras comunes (en inglés) ya se han tomado en el dominio *com*. Pruebe con artículos del hogar, animales, plantas, partes del cuerpo, etcétera. Hay muy pocos disponibles.

Cada dominio se nombra por la ruta hacia arriba desde él a la raíz (sin nombre). Los componentes se separan con puntos. Por lo tanto, el departamento de ingeniería de Sun Microsystems podría utilizar *eng.sun.com.*, en lugar de un nombre tipo UNIX como /*com*/*sun*/*eng.* Observe que esta asignación jerárquica significa que *eng.sun.com.* no entra en conflicto con un uso potencial de *eng* —por ejemplo, *eng.yale.edu.,* que podría usarse en el departamento de inglés de Yale.

Los nombres de dominio pueden ser absolutos o relativos. Un nombre de dominio absoluto termina con un punto (por ejemplo, *eng.sun.com.*), y uno relativo no. Los nombres relativos tienen que interpretarse en algún contexto para determinar de manera única su significado verdadero. En

ambos casos, un dominio nombrado hace referencia a un nodo específico del árbol y a todos los nodos por debajo de él.

Los nombres de dominio no hacen distinción entre mayúsculas y minúsculas, por lo que *edu*, *Edu* y *EDU* significan lo mismo. Los nombres de componentes pueden ser de hasta 63 caracteres de longitud, y los de ruta completa de hasta 255 caracteres.

En principio, los dominios pueden introducirse en el árbol de dos maneras diferentes. Por ejemplo, *cs.yale.edu* también podría estar listado bajo el dominio de país *us* como *cs.yale.ct.us.* Sin embargo, en la práctica, casi todas las organizaciones de Estados Unidos están bajo un dominio genérico, y casi todas las de fuera de Estados Unidos están bajo el dominio de su país. No hay ninguna regla que impida registrarse bajo dos dominios de nivel superior, pero pocas organizaciones lo hacen, excepto las multinacionales (por ejemplo, *sony.com* y *sony.nl*).

Cada dominio controla cómo se asignan los dominios que están debajo de él. Por ejemplo, Japón tiene los dominios *ac.jp* y *co.jp* que son espejos de *edu* y *com.* Los Países Bajos no hacen esta distinción y ponen a todas las organizaciones directamente bajo *nl.* Por lo tanto, los siguientes tres son departamentos universitarios de informática:

1. *cs.yale.edu* (Universidad de Yale, en Estados Unidos)
2. *cs.vu.nl* (Vrije Universiteit, en los Países Bajos)
3. *cs.keio.ac.jp* (Universidad Keio, en Japón)

Para crear un nuevo dominio, se requiere el permiso del dominio en el que se incluirá. Por ejemplo, si se inicia un grupo VLSI en Yale y quiere que se le conozca como *vlsi.cs.yale.edu,* requiere permiso de quien administra *cs.yale.edu.* De la misma manera, si se crea una universidad nueva, digamos la Universidad del Norte de Dakota del Sur, debe solicitar al administrador del dominio *edu* que le asigne *unsd.edu.* De esta manera se evitan los conflictos de nombres y cada dominio puede llevar el registro de todos sus subdominios. Una vez que se ha creado y registrado un nuevo dominio puede crear subdominios, como *cs.unsd.edu,* sin obtener el permiso de nadie más arriba en el árbol.

Los nombres reflejan los límites organizacionales, no las redes físicas. Por ejemplo, si los departamentos de informática e ingeniería eléctrica se ubican en el mismo edificio y comparten la misma LAN, de todas maneras pueden tener dominios diferentes. De manera similar, si el departamento de informática está dividido entre el edificio Babbage y el edificio Turing, todos los *hosts* de ambos edificios pertenecerán, normalmente, al mismo dominio.

7.1.2 Registros de recursos

Cada dominio, sea un *host* individual o un dominio de nivel superior, puede tener un grupo de **registros de recursos** asociados a él. En un *host* individual, el registro de recursos más común es simplemente su dirección IP, pero también existen muchos otros tipos de registros de recursos. Cuando un resolvedor da un nombre de dominio al DNS, lo que recibe son los registros de

recursos asociados a ese nombre. Por lo tanto, la función real del DNS es relacionar los dominios de nombres con los registros de recursos.

Un registro de recursos tiene cinco tuplas. Aunque éstas se codifican en binario por cuestión de eficiencia, en la mayoría de las presentaciones, los registros de recursos se dan como texto ASCII, una línea por registro de recursos. El formato que usaremos es el siguiente:

```
Nombre_dominio  Tiempo_de_vida  Clase  Tipo  Valor
```

El *Nombre_dominio* indica el dominio al que pertenece este registro. Por lo general, existen muchos registros por dominio y cada copia de la base de datos contiene información de muchos dominios. Por lo tanto, este campo es la clave primaria de búsqueda usada para atender las consultas. El orden de los registros de la base de datos no es significativo.

El campo de *Tiempo_de_vida* es una indicación de la estabilidad del registro. La información altamente estable recibe un valor grande, como 86,400 (la cantidad de segundos en un día). La información altamente volátil recibe un valor pequeño, como 60 (1 minuto). Regresaremos a este punto después de haber estudiado el almacenamiento en caché.

El tercer campo de cada registro de recursos es *Class*. Para la información de Internet, siempre es *IN*. Para información que no es de Internet, se pueden utilizar otros códigos, pero en la práctica, éstos raramente se ven.

El campo *Tipo* indica el tipo de registro de que se trata. En la figura 7-2 se listan los tipos más importantes.

Tipo	Significado	Valor
SOA	Inicio de autoridad	Parámetros para esta zona
A	Dirección IP de un *host*	Entero de 32 bits
MX	Intercambio de correo	Prioridad, dominio dispuesto a aceptar correo electrónico
NS	Servidor de nombres	Nombre de un servidor para este dominio
CNAME	Nombre canónico	Nombre de dominio
PTR	Apuntador	Alias de una dirección IP
HINFO	Descripción del *host*	CPU y SO en ASCII
TXT	Texto	Texto ASCII no interpretado

Figura 7-2. Principales tipos de registro de recursos DNS.

Un registro *SOA* proporciona el nombre de la fuente primaria de información sobre la zona del servidor de nombres (que se describe más adelante), la dirección de correo electrónico de su administrador, un número de serie único y varias banderas y temporizadores.

El tipo de registro más importante es el registro *A* (dirección) que contiene una dirección IP de 32 bits de algún *host*. Cada *host* de Internet debe tener cuando menos una dirección IP, para que otras máquinas puedan comunicarse con él. Algunos *hosts* tienen dos o más conexiones de red, en cuyo caso tendrán un registro de recursos tipo *A* por cada conexión de red (y, por lo tanto, por

cada dirección IP). DNS se puede configurar para iterar a través de éstos, regresando el primer registro en la primera solicitud, el segundo registro en la segunda solicitud, y así sucesivamente.

El siguiente tipo de registro más importante es *MX*, que especifica el nombre del dominio que está preparado para aceptar correo electrónico del dominio especificado. Se utiliza porque no todas las máquinas están preparadas para aceptar correo electrónico. Si alguien desea enviar correo electrónico a, por ejemplo, *bill@microsoft.com*, el *host* emisor necesita encontrar un servidor de correo en *microsoft.com* que esté dispuesto a aceptar correo electrónico. El registro *MX* puede proporcionar esta información.

Los registros *NS* especifican servidores de nombres. Por ejemplo, cada base de datos DNS normalmente tiene un registro *NS* por cada dominio de nivel superior, de modo que el correo electrónico pueda enviarse a partes alejadas del árbol de nombres. Regresaremos a este punto más adelante.

Los registros *CNAME* permiten la creación de alias. Por ejemplo, una persona familiarizada con los nombres de Internet en general que quiere enviar un mensaje a alguien cuyo nombre de inicio de sesión es *paul* y que está en el departamento de informática del M.I.T., podría suponer que *paul@cs.mit.edu* funcionará. De hecho, esta dirección no funcionará, puesto que el dominio del departamento de informática del M.I.T. es *lcs.mit.edu.* Sin embargo, como servicio para la gente que no sabe esto, el M.I.T. podría crear una entrada *CNAME* para encaminar a la gente y a los programas en la dirección correcta. La entrada podría ser como la siguiente:

```
cs.mit.edu   86400   IN   CNAME   lcs.mit.edu
```

Al igual que *CNAME*, *PTR* apunta a otro nombre. Sin embargo, a diferencia de *CNAME*, que en realidad es sólo una definición de macro, *PTR* es un tipo de datos DNS normal, cuya interpretación depende del contexto en el que se encontró. En la práctica, *PTR* casi siempre se usa para asociar un nombre a una dirección IP a fin de permitir búsquedas de la dirección IP y devolver el nombre de la máquina correspondiente. Éstas se llaman **búsquedas invertidas.**

Los registros *HINFO* permiten que la gente conozca el tipo de máquina y sistema operativo al que corresponde un dominio. Por último, los registros *TXT* permiten a los dominios identificarse de modos arbitrarios. Ambos tipos de registro son para el provecho de los usuarios. Ninguno es obligatorio, por lo que los programas no pueden contar con que los recibirán (y probablemente no puedan manejarlos si los obtienen).

Por último, llegamos al campo *Valor.* Este campo puede ser un número, un nombre de dominio o una cadena ASCII. La semántica depende del tipo de registro. En la figura 7-2 se presenta una descripción corta de los campos de *Valor* para cada uno de los tipos principales de registro.

Como ejemplo del tipo de información que podría encontrarse en la base de datos DNS de un dominio, vea la figura 7-3. En ésta se ilustra una parte de una base de datos (semihipotética) correspondiente al dominio *cs.vu.nl* mostrado en la figura 7-1. La base de datos contiene siete tipos de registros de recursos.

La primera línea no de comentario de la figura 7-3 da un poco de información básica sobre el dominio, de lo que ya no nos ocuparemos. Las siguientes dos líneas dan información textual

```
;Datos autorizados correspondientes a cs.vu.nl
cs.vu.nl.       86400  IN  SOA    star boss (9527,7200,7200,241920,86400)
cs.vu.nl.       86400  IN  TXT    "Divisie Wiskunde en Informatica."
cs.vu.nl.       86400  IN  TXT    "Vrije Universiteit Amsterdam."
cs.vu.nl.       86400  IN  MX     1 zephyr.cs.vu.nl.
cs.vu.nl.       86400  IN  MX     2 top.cs.vu.nl.

flits.cs.vu.nl. 86400  IN  HINFO  Sun Unix
flits.cs.vu.nl. 86400  IN  A      130.37.16.112
flits.cs.vu.nl. 86400  IN  A      192.31.231.165
flits.cs.vu.nl. 86400  IN  MX     1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX     2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX     3 top.cs.vu.nl.
www.cs.vu.nl.   86400  IN  CNAME  star.cs.vu.nl
ftp.cs.vu.nl.   86400  IN  CNAME  zephyr.cs.vu.nl

rowboat                IN  A      130.37.56.201
                       IN  MX     1 rowboat
                       IN  MX     2 zephyr
                       IN  HINFO  Sun Unix

little-sister          IN  A      130.37.62.23
                       IN  HINFO  Mac MacOS

laserjet               IN  A      192.31.231.216
                       IN  HINFO  "HP Laserjet IIISi" Proprietary
```

Figura 7-3. Parte de una posible base de datos DNS para *cs.vu.nl.*

sobre la localización del dominio. Luego están dos entradas que dan el primer y segundo lugar a donde se intentará entregar correo electrónico dirigido a *person@cs.vu.nl.* Primero se intentará en la *zephyr* (una máquina específica). Si falla, a continuación se intentará en la *top*.

Después de la línea en blanco, que se agregó para hacer más clara la lectura, siguen líneas que indican que la *flits* es una estación de trabajo Sun que ejecuta UNIX, y dan sus dos direcciones IP. Después se indican tres posibilidades para manejar el correo electrónico enviado a *flits.cs.vu.nl.* La primera opción naturalmente es la *flits* misma, pero si está inactiva, la *zephyr* y la *top* son la segunda y tercera opciones, respectivamente. Luego viene un alias, *www.cs.vu.nl,* para que esta dirección pueda usarse sin designar una máquina específica. La creación de este alias permite a *cs.vu.nl* cambiar su servidor *Web* sin invalidar la dirección que la gente usa para llegar a él. Un argumento similar es válido para *ftp.cs.vu.nl.*

Las siguientes cuatro líneas contienen una entrada típica de una estación de trabajo, en este caso, *rowboat.cs.vu.nl.* La información proporcionada contiene la dirección IP, los destinos de correo primarios y secundarios, así como información sobre la máquina. Luego viene una entrada para un sistema no UNIX incapaz de recibir correo por sí mismo, seguida de una entrada para una impresora láser que está conectada a Internet.

Lo que no se muestra (y no está en este archivo) son las direcciones IP a usar para buscar los dominios de nivel superior. Éstas son necesarias para buscar *hosts* distantes pero, dado que no son parte del dominio *cs.vu.nl,* no están en este archivo. Tales direcciones son suministradas por los servidores raíz, cuyas direcciones IP están presentes en un archivo de configuración del sistema y se cargan en el caché del DNS cuando se arranca el servidor DNS. Hay aproximadamente una docena de servidores raíz esparcidos en todo el mundo, y cada uno conoce las direcciones IP de todos los servidores de dominio de nivel superior. Por lo tanto, si una máquina conoce la dirección IP de por lo menos un servidor raíz, puede buscar cualquier nombre DNS.

7.1.3 Servidores de nombres

Cuando menos en teoría, un solo servidor de nombres podría contener toda la base de datos DNS y responder a todas las consultas dirigidas a ella. En la práctica, este servidor estaría tan sobrecargado que sería inservible. Más aún, si llegara a caerse, la Internet completa se vendría abajo.

Para evitar los problemas asociados a tener una sola fuente de información, el espacio de nombres DNS se divide en **zonas** no traslapantes. En la figura 7-4 se muestra una manera posible de dividir el espacio de nombres de la figura 7-1. Cada zona contiene una parte del árbol y también contiene servidores de nombres que tienen la información de autorización correspondiente a esa zona. Por lo general, una zona tendrá un servidor de nombres primario, que obtiene su información de un archivo en su disco, y uno o más servidores de nombres secundarios, que obtienen su información del primario. Para mejorar la confiabilidad, algunos servidores de cierta zona pueden situarse fuera de la zona.

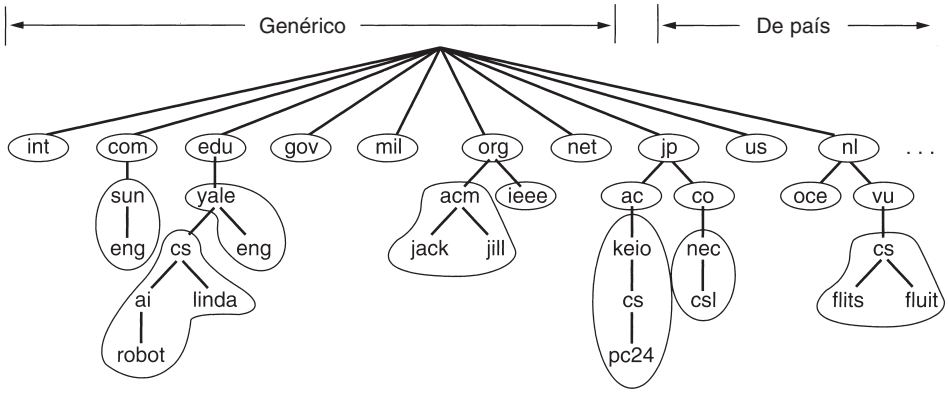


Figura 7-4. Parte del espacio de nombres DNS, donde se muestra la división en zonas.

El lugar donde se colocan los límites de las zonas dentro de una zona es responsabilidad del administrador de esa zona. Esta decisión se toma en gran medida con base en la cantidad de servidores de nombres deseados y su ubicación. Por ejemplo, en la figura 7-4, Yale tiene un servidor

para *yale.edu* que maneja *eng.yale.edu,* pero no *cs.yale.edu,* que es una zona aparte con sus propios servidores de nombres. Tal decisión podría tomarse cuando un departamento, como el de inglés, no desea operar su propio servidor de nombres, pero un departamento como el de informática sí. En consecuencia, *cs.yale.edu* es una zona aparte, pero *eng.yale.edu* no.

Cuando un resolvedor tiene una consulta referente a un nombre de dominio, la pasa a uno de los servidores de nombres locales. Si el dominio que se busca cae bajo la jurisdicción del servidor de nombres, como *ai.cs.yale.edu,* que cae bajo *cs.yale.edu,* devuelve los registros de recursos autorizados. Un **registro autorizado** es uno que proviene de la autoridad que administra el registro y, por lo tanto, siempre es correcto. Los registros autorizados contrastan con los registros en caché, que podrían no estar actualizados.

Por otro lado, si el dominio es remoto y no hay información disponible localmente sobre el dominio solicitado, el servidor de nombres envía un mensaje de consulta al servidor de nombres de nivel superior en el que le solicita dicho dominio. Para hacer más claro este proceso, considere el ejemplo de la figura 7-5. Aquí, un resolvedor de *flits.cs.vu.nl* quiere saber la dirección IP del *host linda.cs.yale.edu.* En el paso 1, envía una consulta al servidor de nombres local, *cs.vu.nl.* Esta consulta contiene el nombre de dominio buscado, el tipo (*A*) y la clase (*IN*).

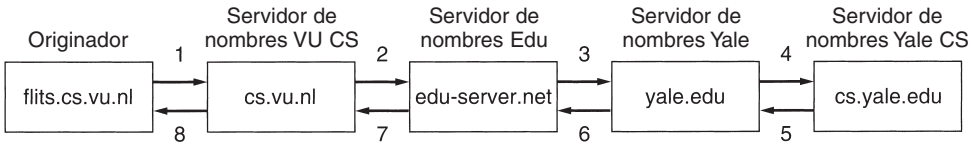


Figura 7-5. Manera en que un resolvedor busca un nombre remoto en ocho pasos.

Supongamos que al servidor de nombres local nunca se le ha solicitado este dominio, por lo que no sabe nada sobre él; puede preguntar a otros servidores de nombres cercanos, pero si ninguno de ellos sabe nada, enviará un paquete UDP al servidor de *edu* indicado en su base de datos (vea la figura 7-5), *edu-server.net.* Desde luego, es improbable que este servidor sepa la dirección de *linda.cs.yale.edu,* y probablemente tampoco sabe la de *cs.yale.edu,* pero debe conocer a todos sus hijos, por lo que reenvía la solicitud al servidor de nombres de *yale.edu* (paso 3). A su vez, éste reenvía la solicitud a *cs.yale.edu* (paso 4), que debe tener los registros de recursos autorizados. Puesto que cada solicitud es de un cliente a un servidor, el registro de recursos solicitado regresa a través de los pasos 5 a 8.

Una vez que estos registros regresan al servidor de nombres *cs.vu.nl,* se almacenan en caché, por si se necesitan posteriormente. Sin embargo, esta información no es autorizada, puesto que los cambios hechos en *cs.yale.edu* no se propagarán a todos los cachés del mundo que puedan saber sobre ella. Por esta razón, las entradas de caché no deben vivir demasiado tiempo. Ésta es la razón por la cual el campo *Tiempo_de_vida* se incluye en cada registro de recursos; indica a los servidores de nombres remotos cuánto tiempo deben mantener en caché los registros. Si cierta máquina ha tenido la misma dirección IP durante años, puede ser seguro poner en caché esa información

durante un día. En el caso de información más volátil, podría ser más seguro purgar los registros tras unos cuantos segundos o un minuto.

Vale la pena mencionar que el método de consultas aquí descrito se conoce como **consulta recursiva,** puesto que cada servidor que no tiene toda la información solicitada la busca en algún otro lado y luego la proporciona. Es posible un procedimiento alternativo. En él, cuando una consulta no puede satisfacerse localmente, ésta falla, pero se devuelve el nombre del siguiente servidor a intentar a lo largo de la línea. Algunos servidores no implementan consultas recursivas y siempre devuelven el nombre del siguiente servidor a intentar.

También vale la pena indicar que cuando un cliente DNS no recibe una respuesta antes de que termine su temporizador, por lo general probará con otro servidor la siguiente vez. La suposición aquí es que el servidor probablemente está inactivo, y no que la solicitud o la respuesta se perdieron.

Si bien DNS es muy importante para el funcionamiento correcto de Internet, todo lo que hace realmente es relacionar nombres simbólicos de máquinas con sus direcciones IP. No ayuda a localizar personas, recursos, servicios u objetos en general. Para localizar este tipo de cosas, se ha definido otro servicio de directorio, llamado **LDAP** (**Protocolo Ligero de Acceso al Directorio**). Éste es una versión simplificada del servicio de directorio OSI X.500 y se describe en el RFC 2251. Organiza información como un árbol y permite búsquedas en componentes diferentes. Se puede considerar como un directorio telefónico de "páginas blancas". No lo analizaremos más en este libro, pero para mayor información, vea (Weltman y Dahbura, 2000).

7.2 CORREO ELECTRÓNICO

El correo electrónico, o ***e-mail*,** como lo conocen muchos aficionados, ha existido por más de dos décadas. Antes de 1990, se utilizaba principalmente en ambientes académicos. En la década de 1990, se dio a conocer al público y creció en forma exponencial al punto que el número de mensajes de correo electrónico enviados por día ahora es mayor que el número de cartas por **correo caracol** (es decir, en papel).

El correo electrónico, como la mayoría de otras formas de comunicación, tiene sus propias convenciones y estilos. En particular, es muy informal y tiene un umbral bajo de uso. Las personas que nunca hubieran soñado con escribir una carta a un personaje importante, no dudarían un instante para enviarle un mensaje de correo electrónico.

El correo electrónico está lleno de abreviaturas, como BTW (*By The Way*, por cierto), ROTFL (*Rolling On The Floor Laughing*, rodando por el suelo muerto de risa) e IMHO (*In My Humble Opinión*, en mi humilde opinión). Muchas personas también utilizan pequeños símbolos ASCII llamados **caritas** o **símbolos de emociones** en sus mensajes de correo electrónico. Algunos de los más interesantes se listan en la figura 7-6. Para la mayoría, rote el libro 90 grados en dirección de las manecillas del reloj y se verán con mayor claridad. Un minilibro que presenta cerca de 650 caritas es (Sanderson y Dougherty, 1993).

Los primeros sistemas de correo electrónico simplemente consistían en protocolos de transferencia de archivos, con la convención de que la primera línea de cada mensaje (es decir, archivo)