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).

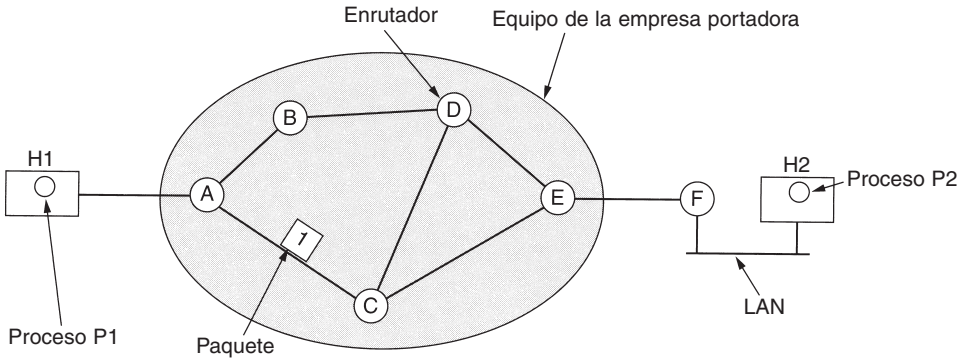


Figura 5-1. El entorno de los protocolos de la capa de red.

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.

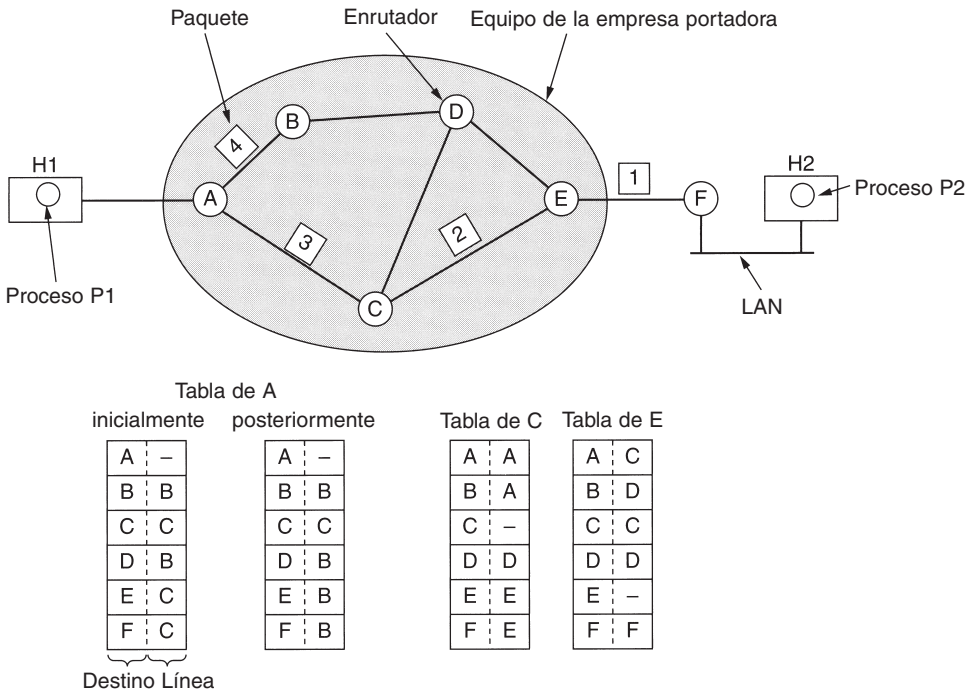


Figura 5-2. Enrutamiento dentro de una subred de datagramas.

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

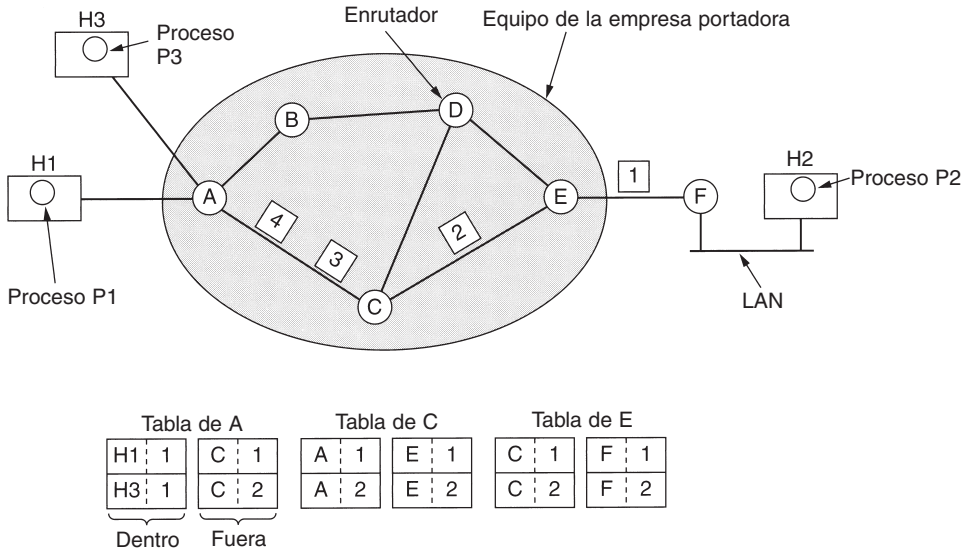


Figura 5-3. Enrutamiento dentro de una subred de circuitos virtuales.

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.

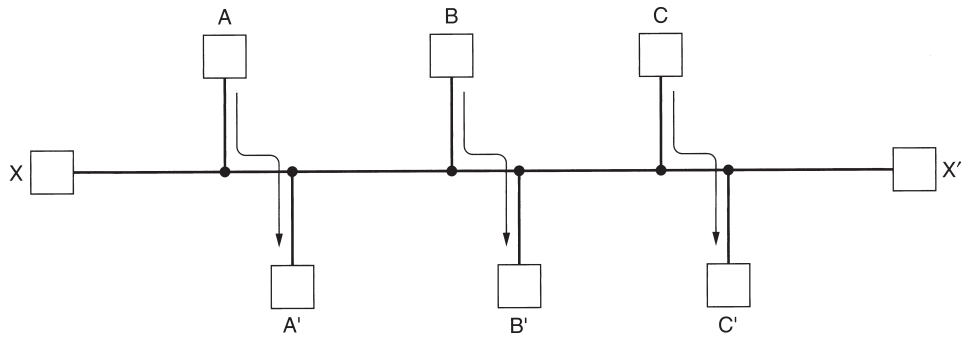


Figura 5-5. El conflicto entre equidad y optimización.

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.

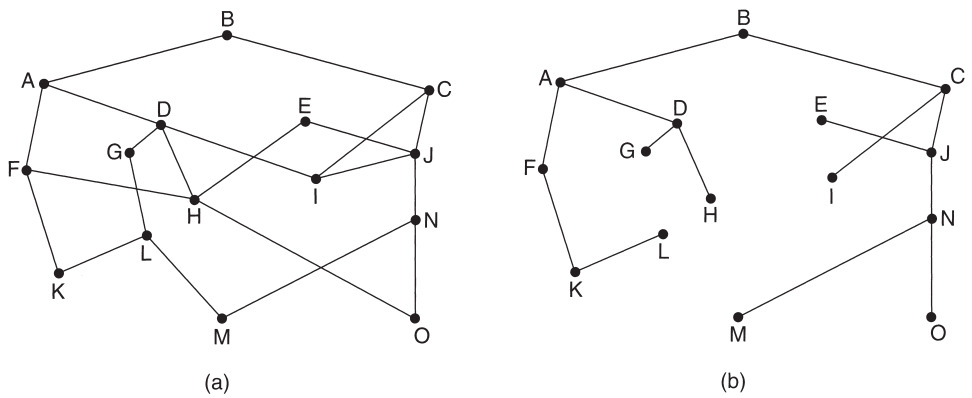


Figura 5-6. (a) Una subred. (b) Árbol sumidero para el enrutador *B*.

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

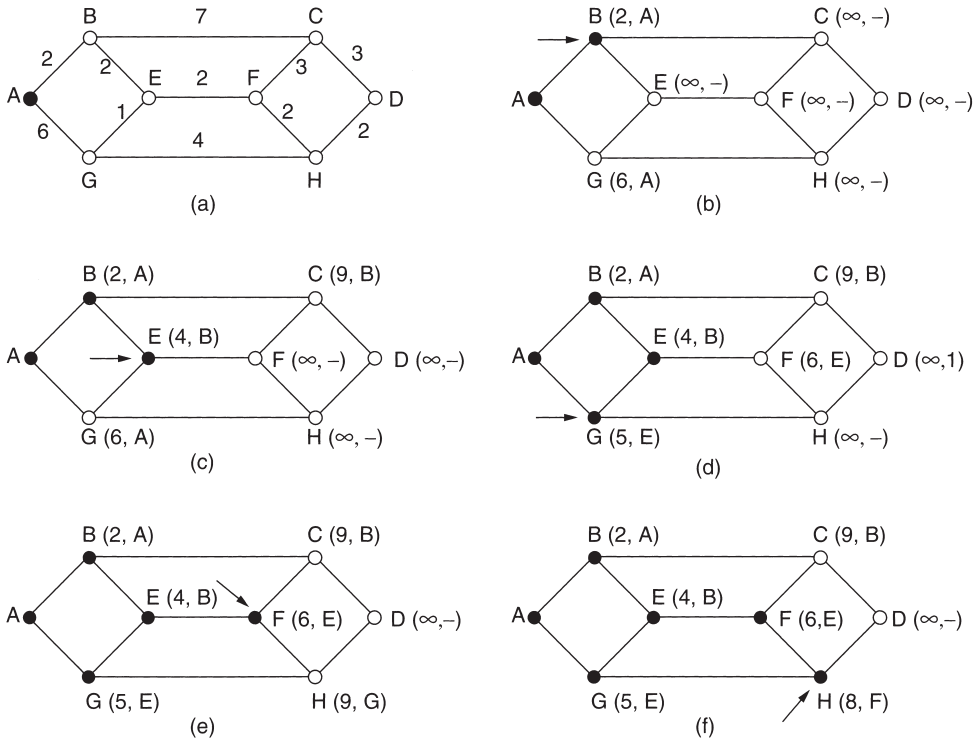


Figura 5-7. Los primeros cinco pasos del cálculo de la ruta más corta de *A* a *D*. Las flechas indican el nodo de trabajo.

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.

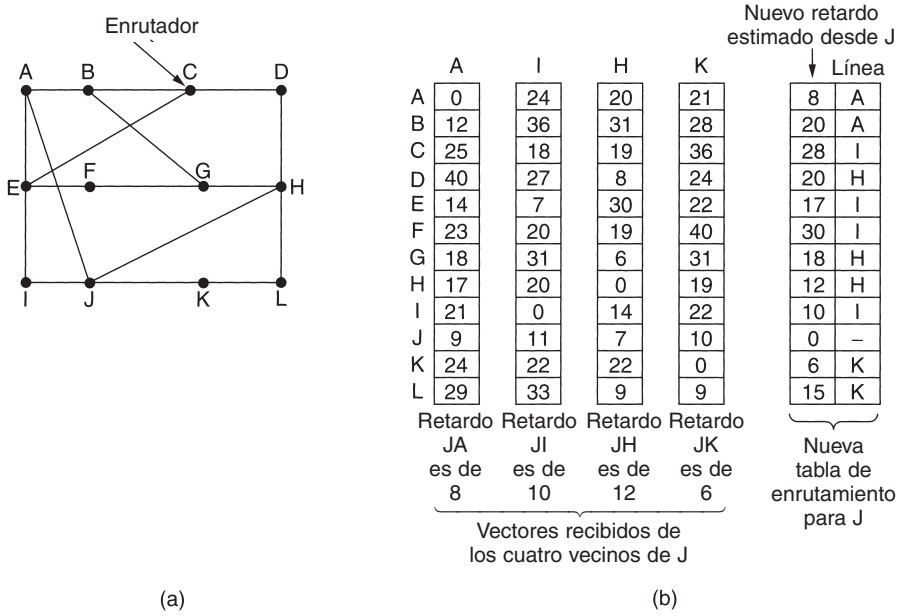


Figura 5-9. (a) Subred. (b) Entrada de A, I, H, K y la nueva tabla de enrutamiento de J.

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*.

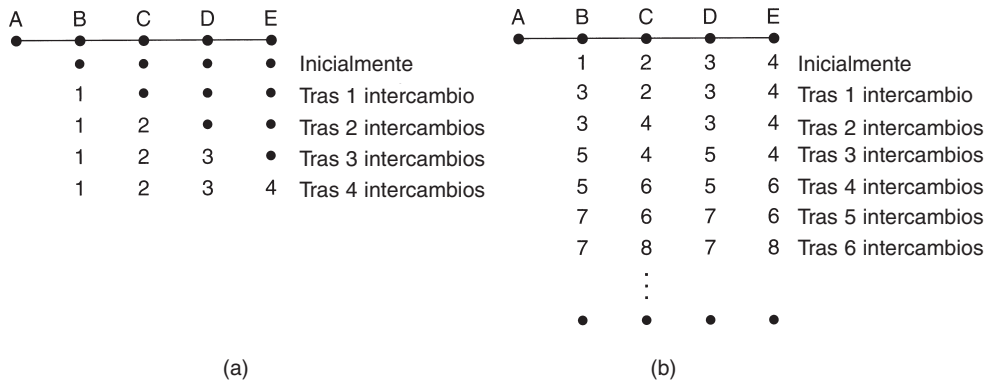


Figura 5-10. El problema de la cuenta hasta infinito.

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.

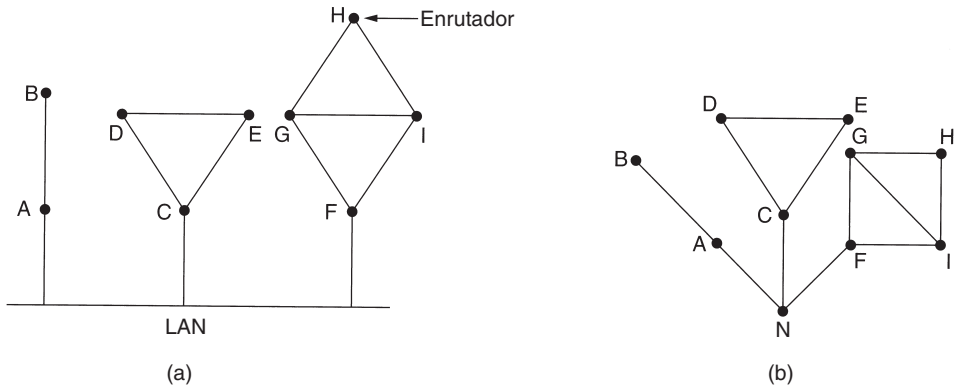


Figura 5-11. (a) Nueve enrutadores y una LAN. (b) Modelo de grafo de (a).

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*.

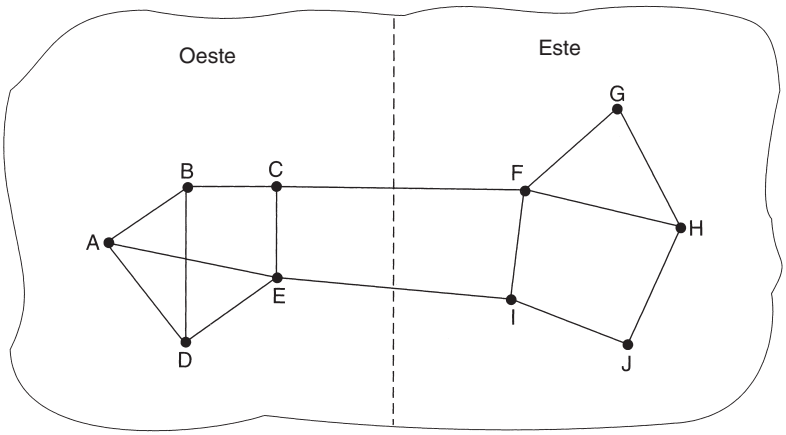
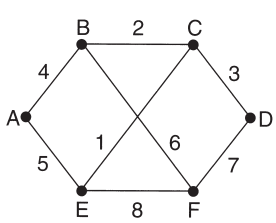


Figura 5-12. Subred en la que las partes este y oeste están conectadas por dos líneas.

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

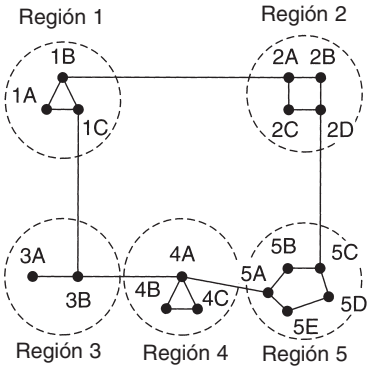
A medida que crece el tamaño de las redes, también lo hacen, de manera proporcional, las tablas de enrutamiento del enrutador. Las tablas que siempre crecen no sólo consumen memoria del enrutador, sino que también se necesita más tiempo de CPU para examinarlas y más ancho de banda para enviar informes 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.

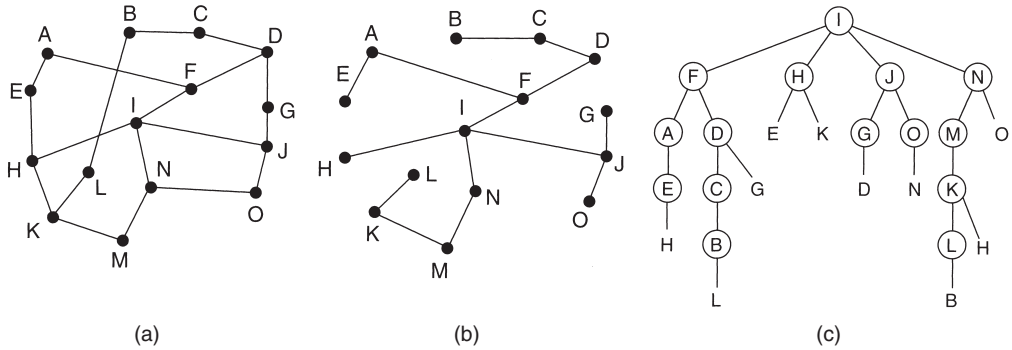


Figura 5-16. Reenvío por ruta invertida. (a) Subred. (b) Árbol sumidero. (c) Árbol construido mediante reenvío por ruta invertida.

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.

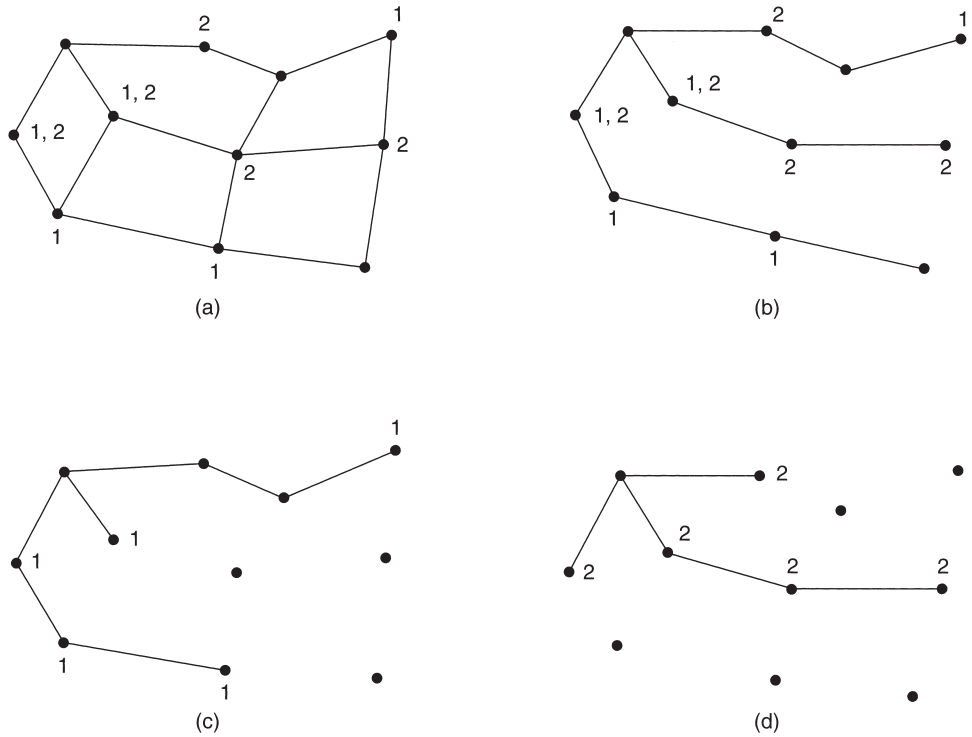


Figura 5-17. (a) Subred. (b) Árbol de expansión del enrutador del extremo izquierdo. (c) Árbol de multidifusión del grupo 1. (d) Árbol de multidifusión para el grupo 2.

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.

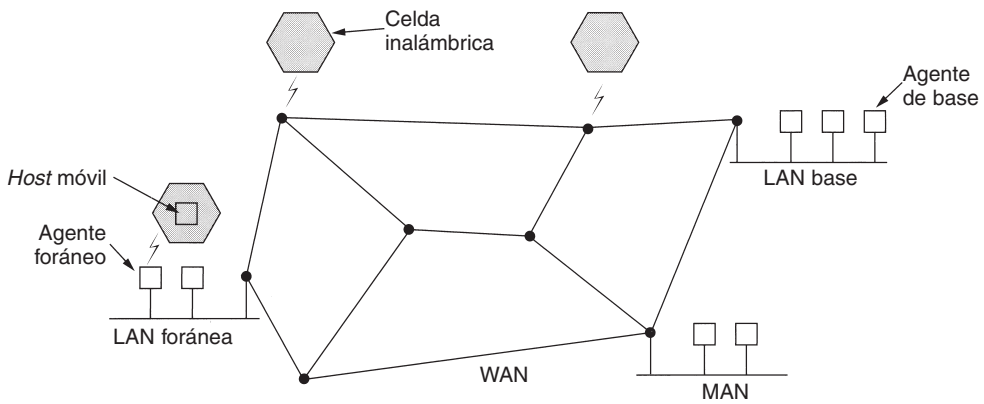


Figura 5-18. WAN a la que están conectadas LANs, MANs y celdas inalámbricas.

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.

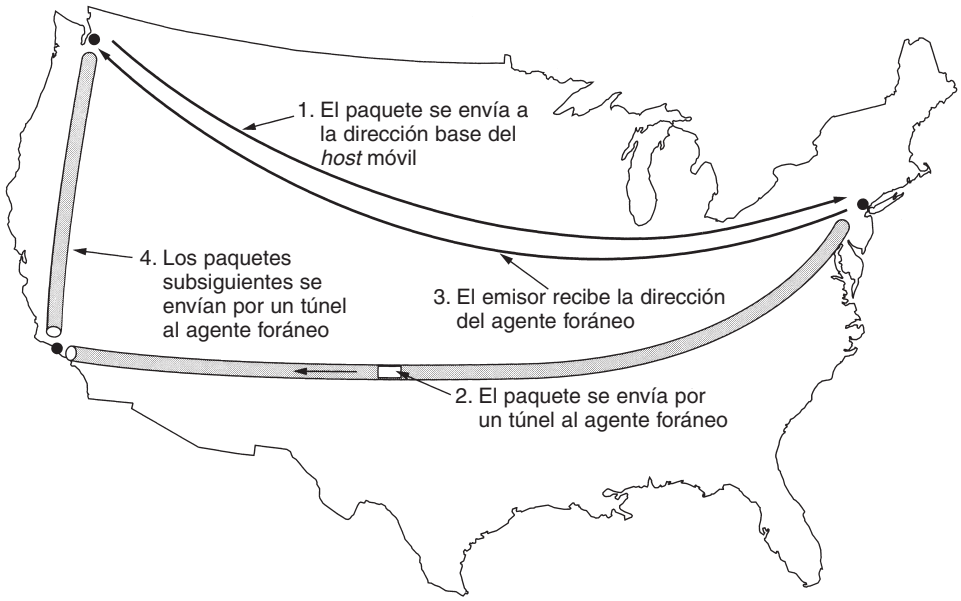


Figura 5-19. Enrutamiento de paquetes para *hosts* móviles.

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".

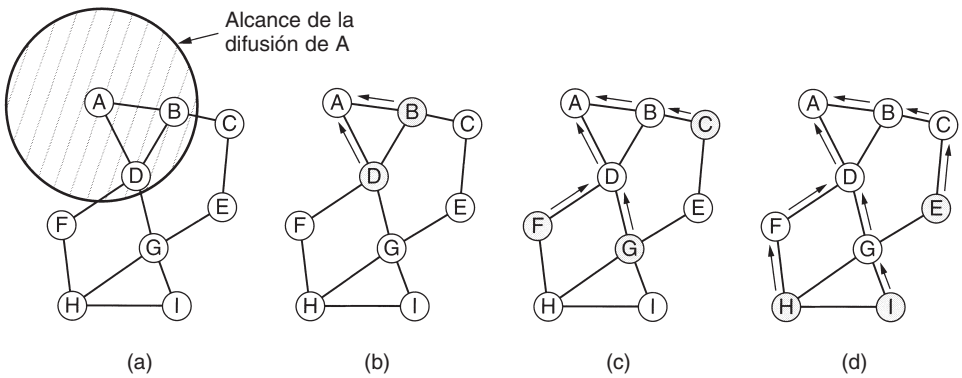


Figura 5-20. (a) Alcance de la difusión de *A*. (b) Después de que *B* y *D* reciben la difusión de *A*. (c) Después de que *C*, *F* y *G* reciben la difusión de *A*. (d) Después de que *E*, *H* e *I* reciben la difusión de *A*. Los nodos sombreados son nuevos receptores. Las flechas muestran las rutas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

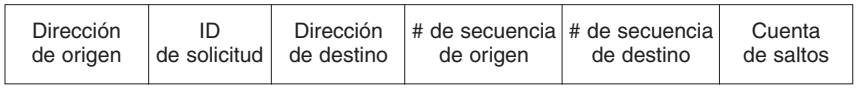


Figura 5-21. Formato de un paquete ROUTE REQUEST.

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.

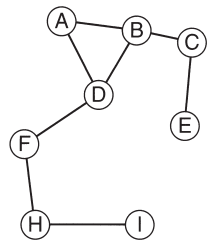
Mantenimiento de rutas

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

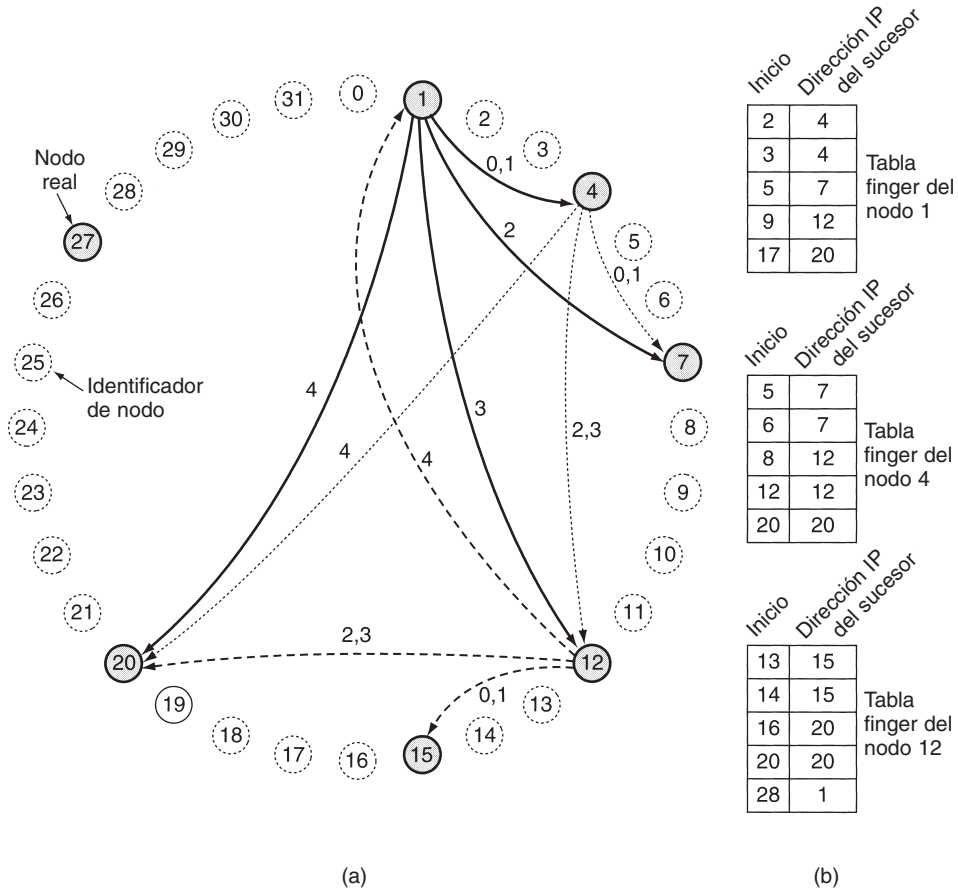


Figura 5-24. (a) Conjunto de 32 identificadores de nodos ordenados en círculo. Los sombreados corresponden a máquinas reales. Los arcos muestran los fingers de los nodos 1, 4 y 12. Las etiquetas de los arcos son los índices de las tablas. (b) Ejemplos de las tablas finger.

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.

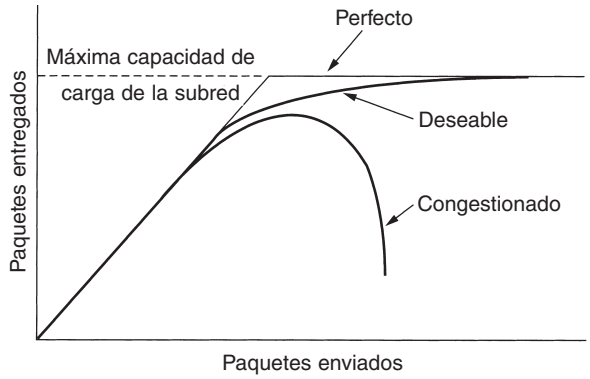


Figura 5-25. Cuando se genera demasiado tráfico, ocurre congestión y se degrada marcadamente el desempeño.

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.

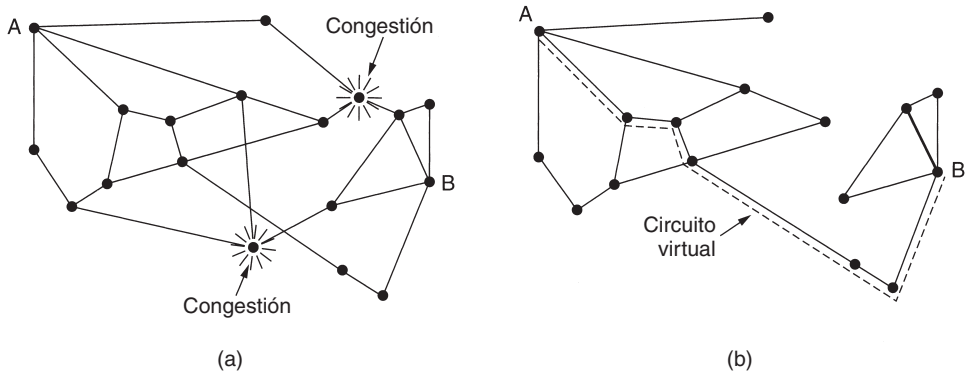


Figura 5-27. (a) Subred congestionada. (b) Subred redibujada que elimina la congestión. También se muestra un circuito virtual de *A* a *B*.

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

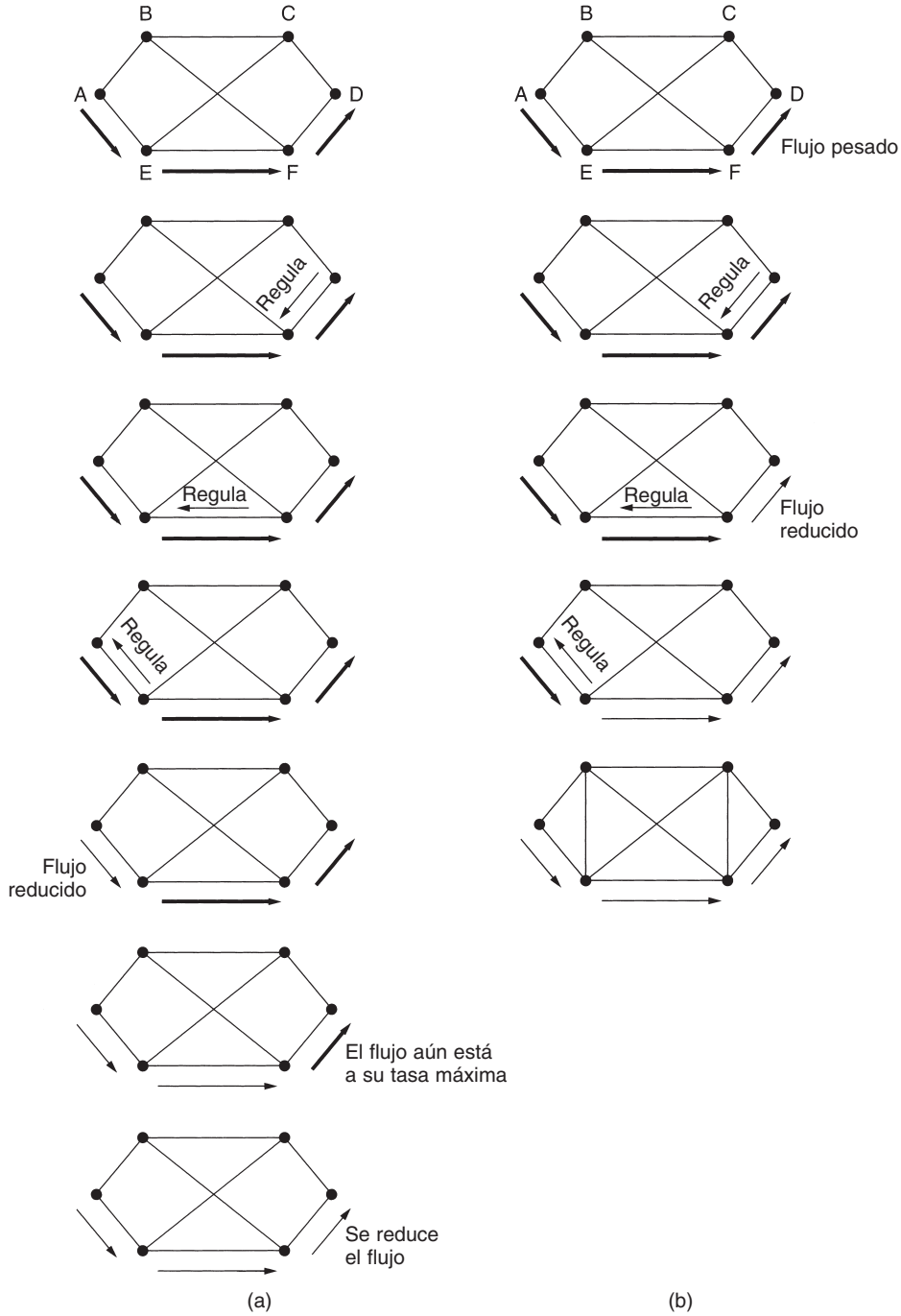


Figura 5-28. (a) Paquete regulador que afecta sólo al origen. (b) Paquete regulador que afecta cada salto por el que pasa.

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.

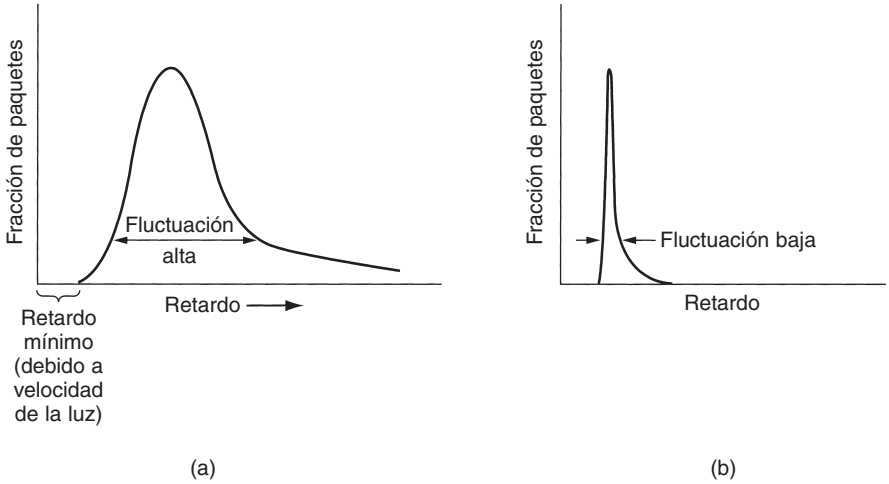


Figura 5-29. (a) Fluctuación alta. (b) Fluctuación baja.

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.

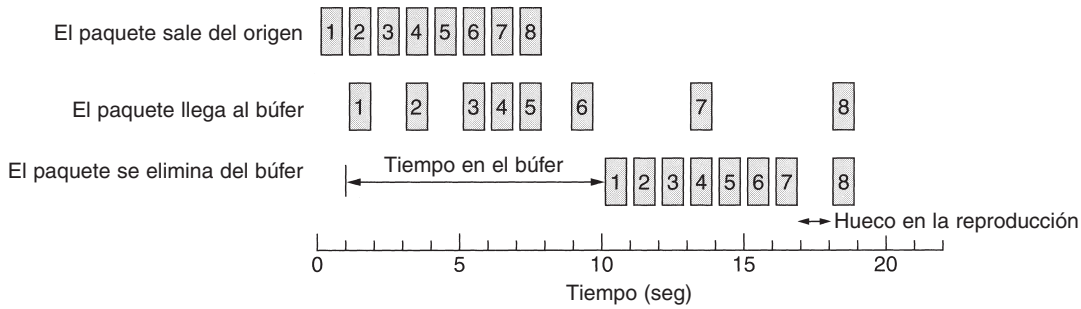


Figura 5-31. Refinamiento del flujo de paquetes almacenándolos en el búfer.

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-

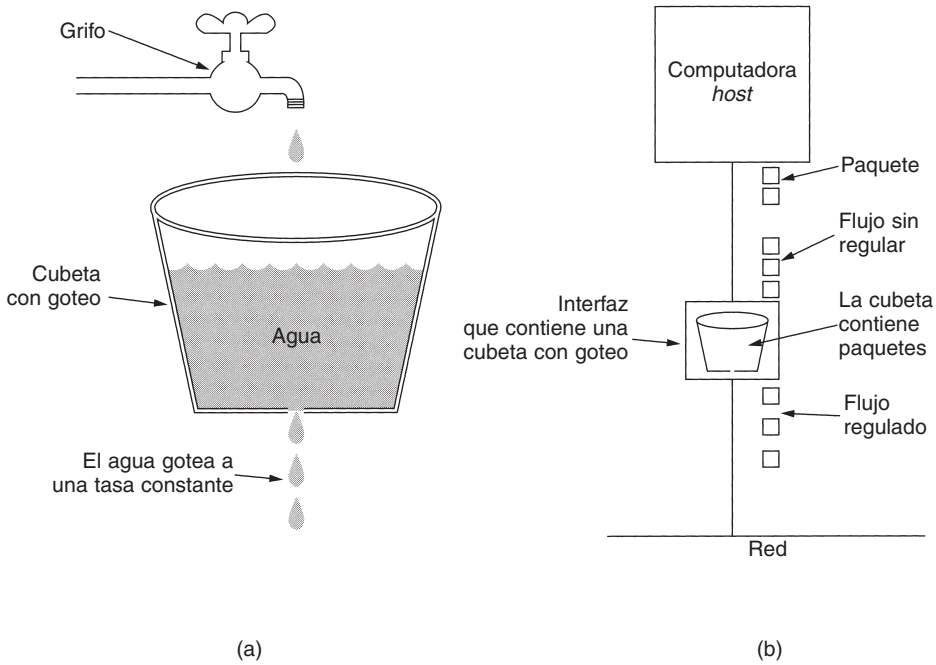


Figura 5-32. (a) Una cubeta con goteo, llena de agua. (b) Cubeta con goteo, llena de paquetes.

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.

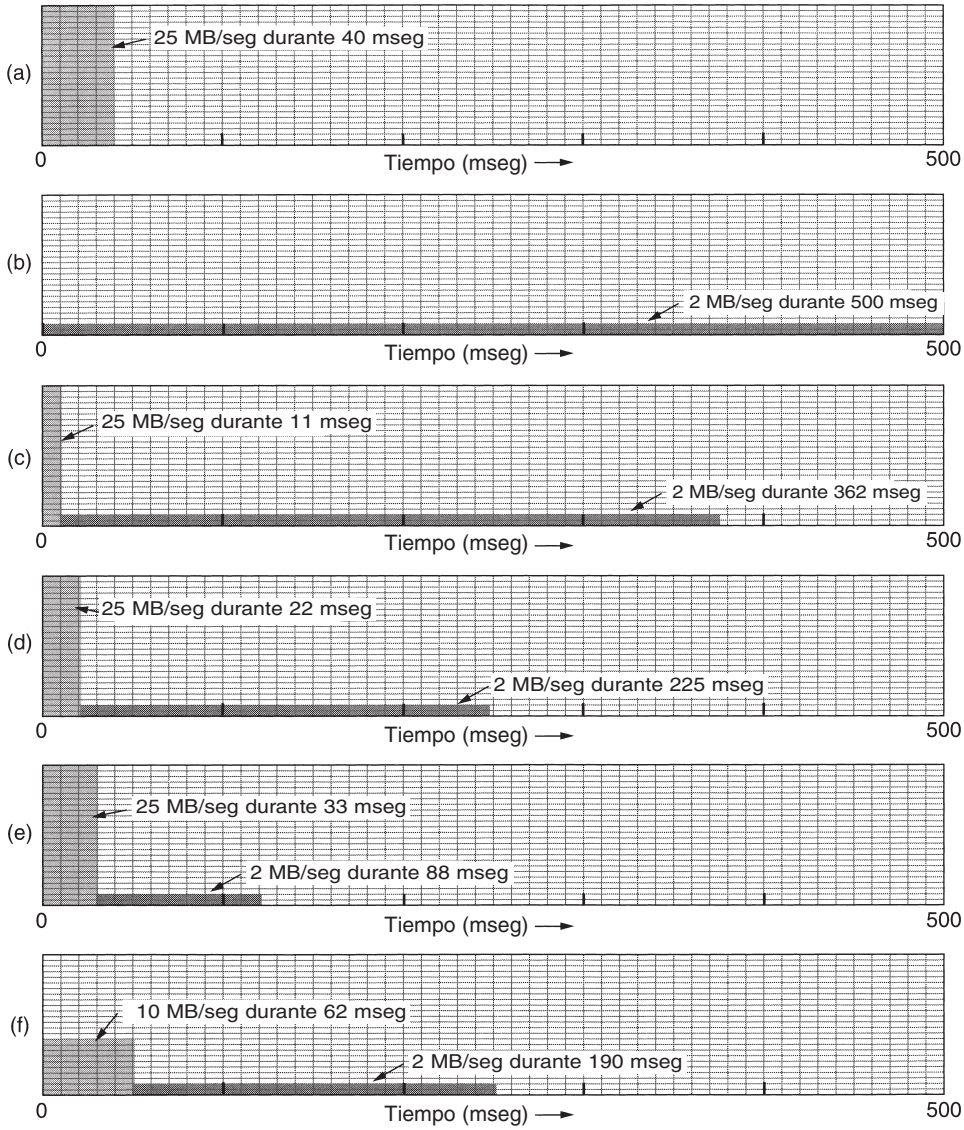


Figura 5-33. (a) Entrada a una cubeta con goteo. (b) Salida de una cubeta con goteo. (c)-(e) Salida de una cubeta con *tokens* con capacidades de 250 KB, 500 KB y 750 KB. (f) Salida de una cubeta con *tokens* de 500 KB que alimenta a una cubeta con goteo de 10 MB/seg.

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.

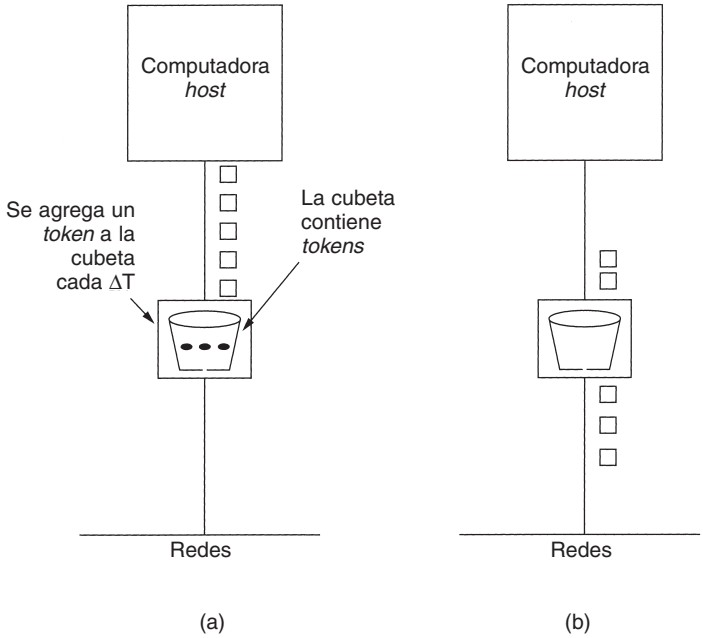


Figura 3-34. Algoritmo de cubeta con *tokens*. (a) Antes. (b) Después.

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.

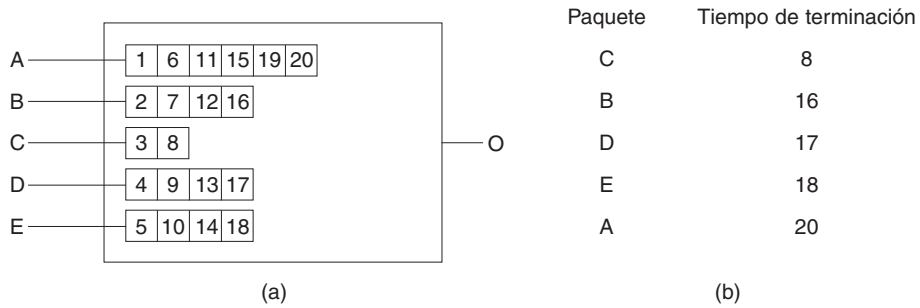


Figura 5-36. (a) Un enrutador con cinco paquetes encolados en la línea *O*. (b) Tiempos de terminación de los cinco paquetes.

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*.

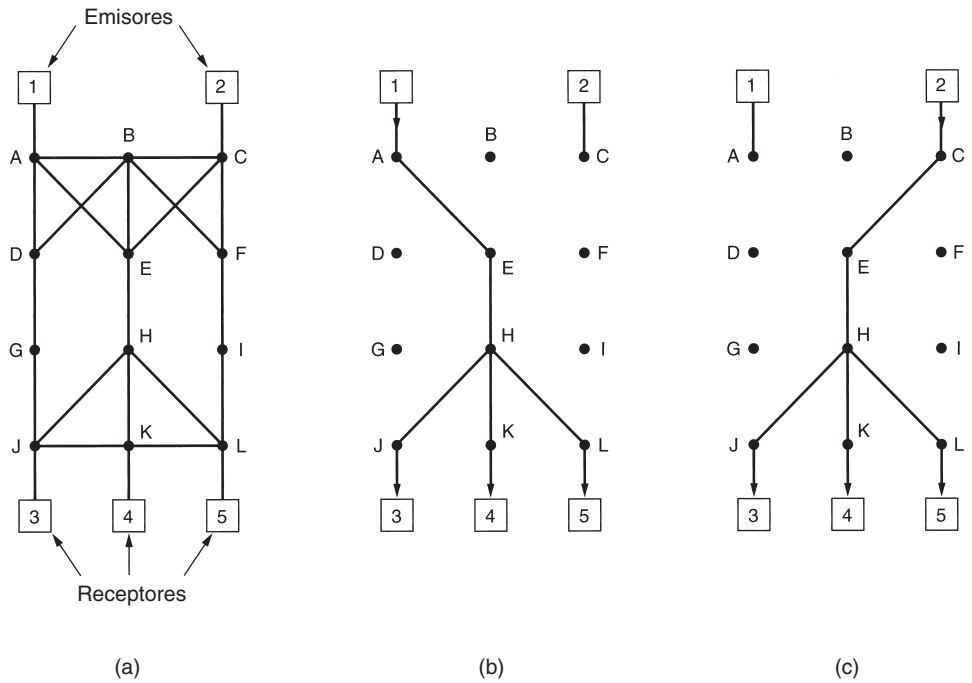


Figura 5-37. (a) Red. (b) Árbol de expansión de multidifusión para el *host* 1. (c) Árbol de expansión de multidifusión para el *host* 2.

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.

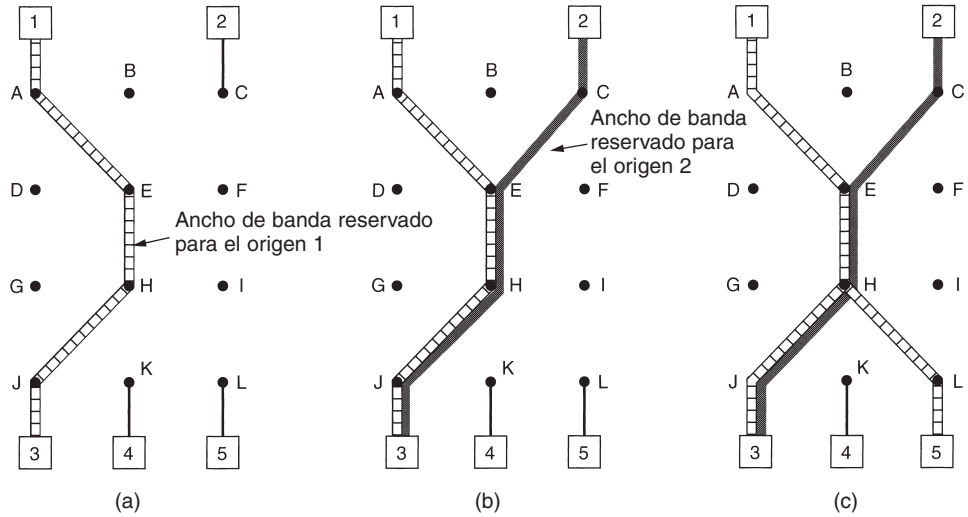


Figura 5-38. (a) El *host* 3 solicita un canal al *host* 1. (b) El *host* 3 solicita entonces un segundo canal al *host* 2. (c) El *host* 5 solicita un canal al *host* 1.

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.

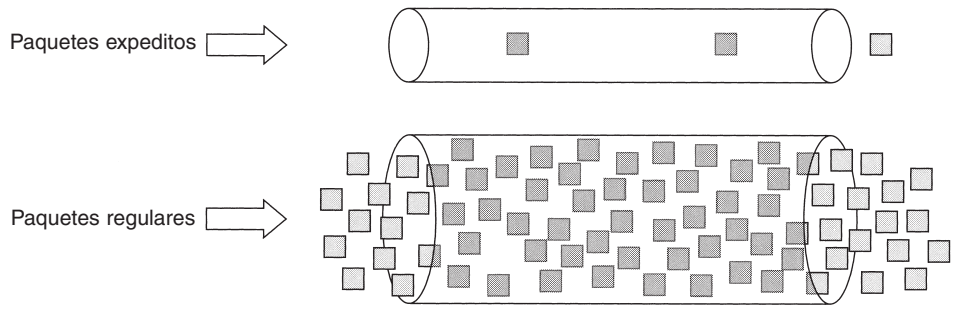


Figura 5-39. Los paquetes expeditos viajan por una red libre de tráfico.

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.

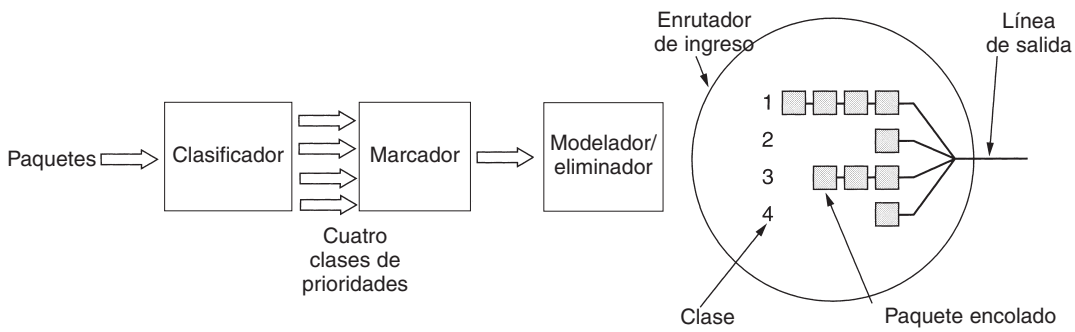


Figura 5-40. Una posible implementación del flujo de datos para el reenvío asegurado.

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.

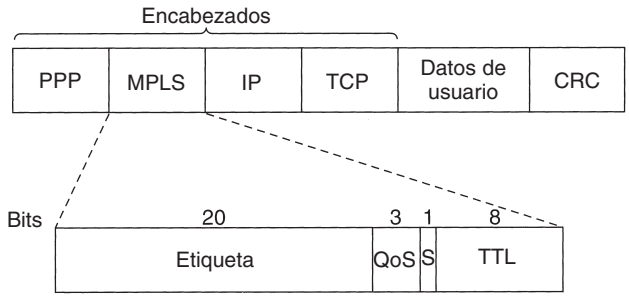


Figura 5-41. Transmisión de un segmento TCP que utiliza IP, MPLS y PPP.

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.

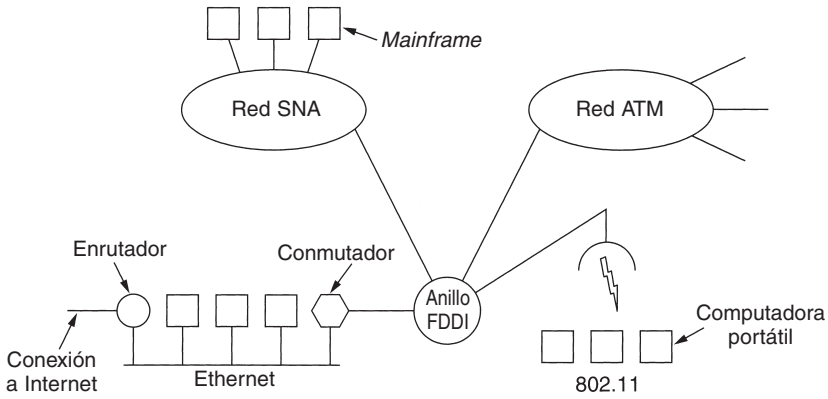


Figura 5-42. Una colección de redes interconectadas.

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.

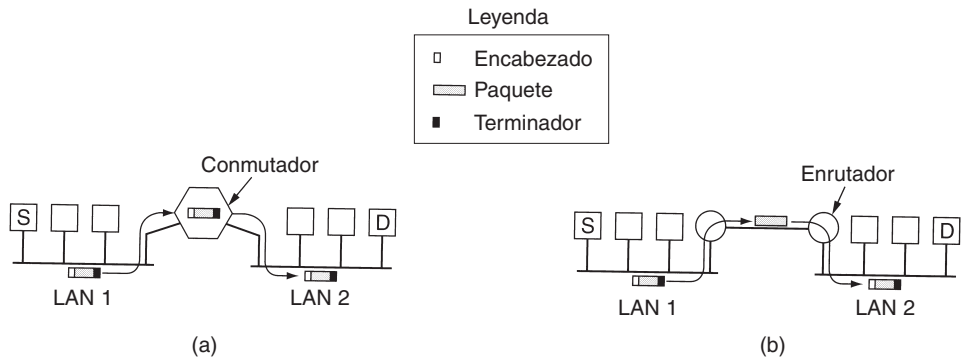


Figura 5-44. (a) Dos Ethernets conectadas mediante un conmutador. (b) Dos Ethernets conectadas mediante enrutadores.

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.

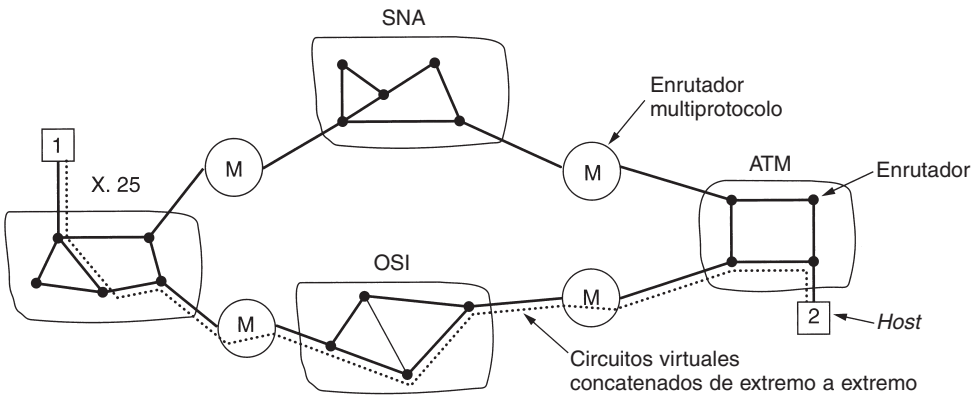


Figura 5-45. Interconectividad mediante circuitos virtuales concatenados.

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

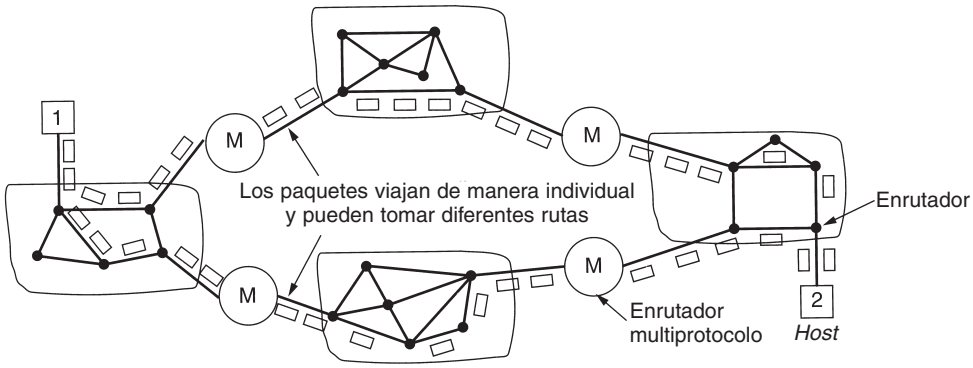


Figura 5-46. Una interred no orientada a la conexión.

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.

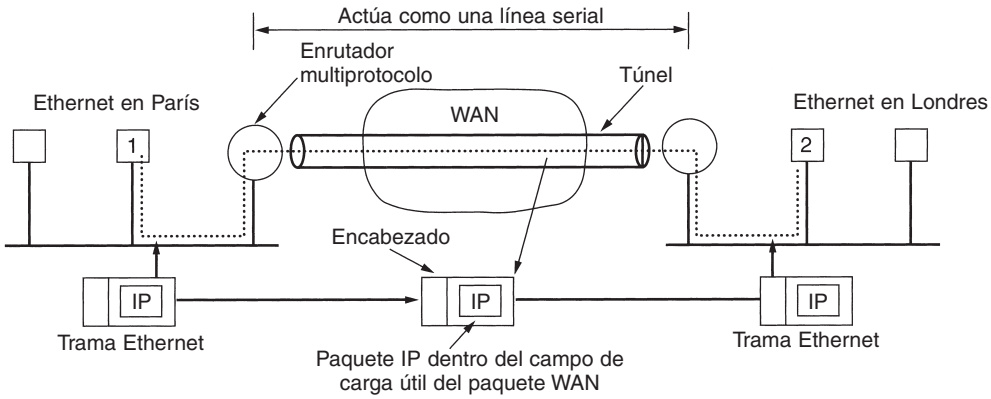


Figura 5-47. Entunelamiento de un paquete de París a Londres.

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.

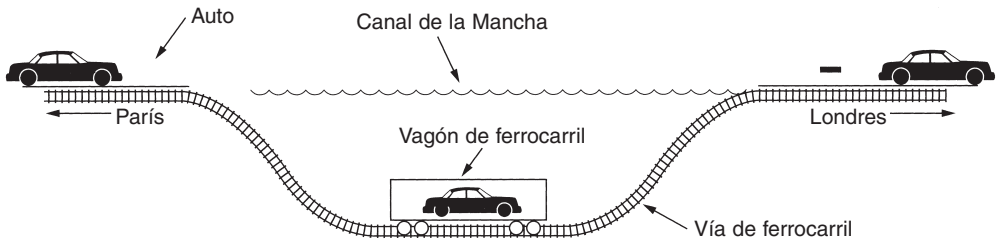


Figura 5-48. Paso de un auto de Francia a Inglaterra a través de un túnel.

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).

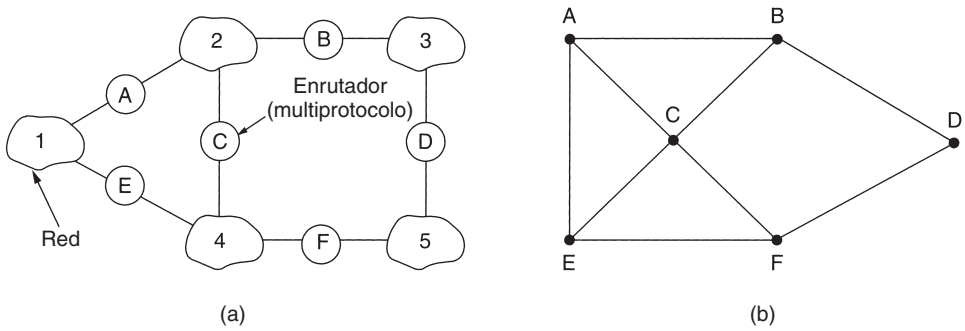


Figura 5-49. (a) Una interred. (b) Grafo de la interred.

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.

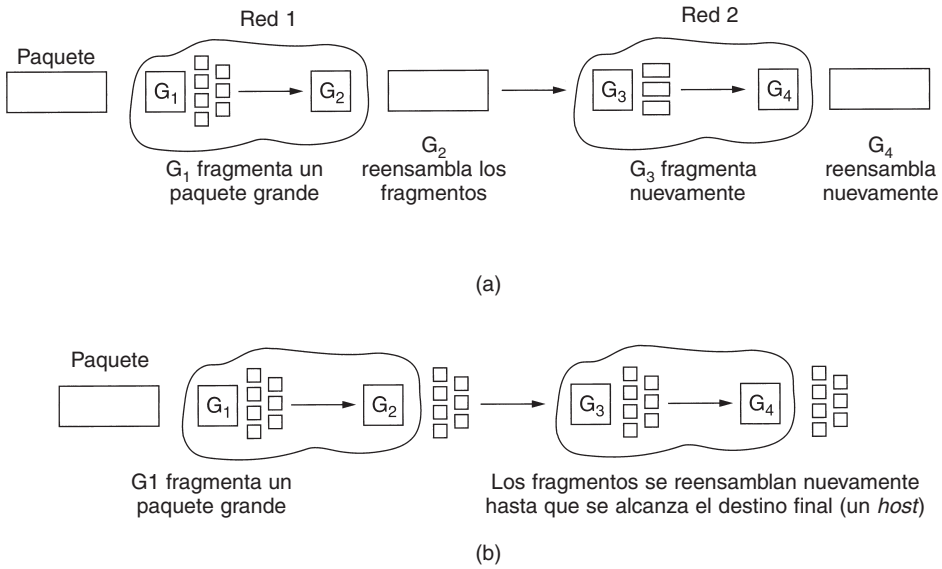


Figura 5-50. (a) Fragmentación transparente. (b) Fragmentación no transparente.

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.

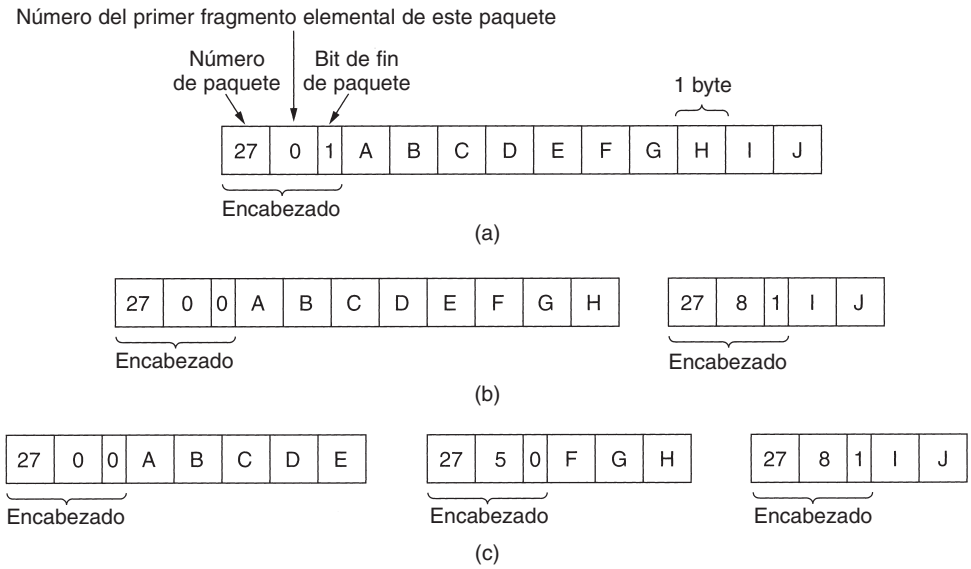


Figura 5-51. Fragmentación cuando el tamaño de datos elemental es de 1 byte. (a) Paquete original que contiene 10 bytes de datos. (b) Fragmentos tras pasar a través de una red con un tamaño máximo de paquete de 8 bytes de carga útil más encabezado. (c) Fragmentos tras pasar a través de una puerta de 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.

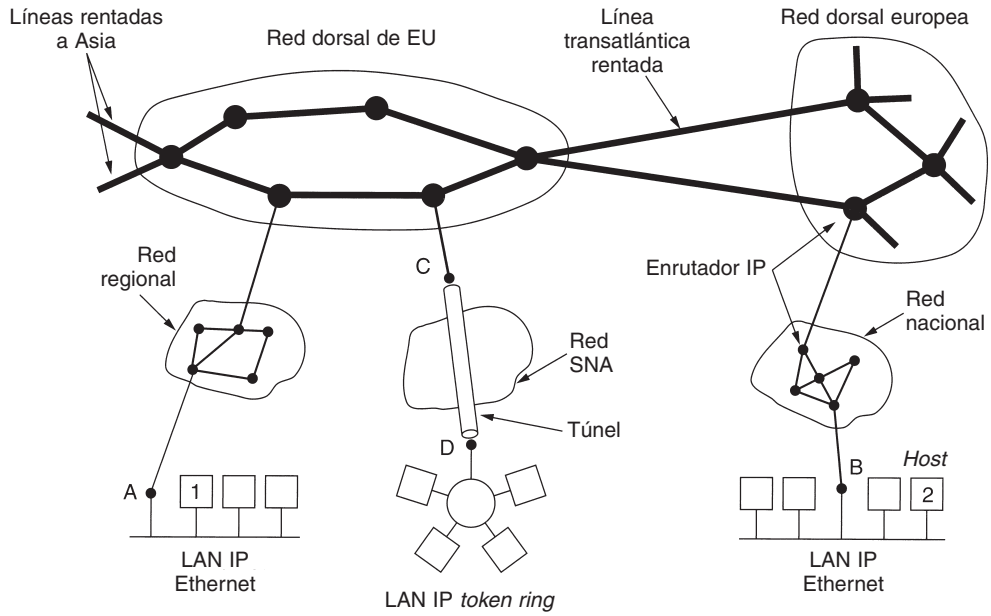


Figura 5-52. Internet es un conjunto interconectado de muchas redes.

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

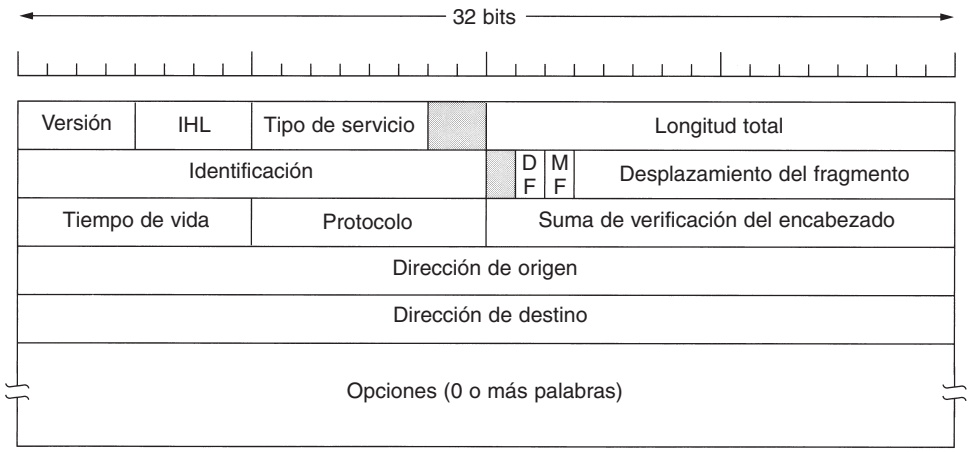


Figura 5-53. El encabezado de IPv4 (Protocolo Internet).

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.

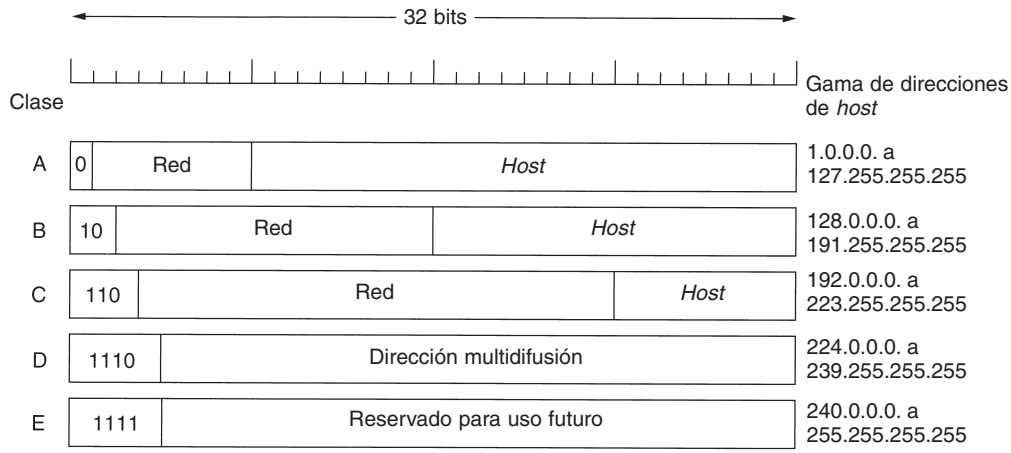


Figura 5-55. Formatos de dirección IP.

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.

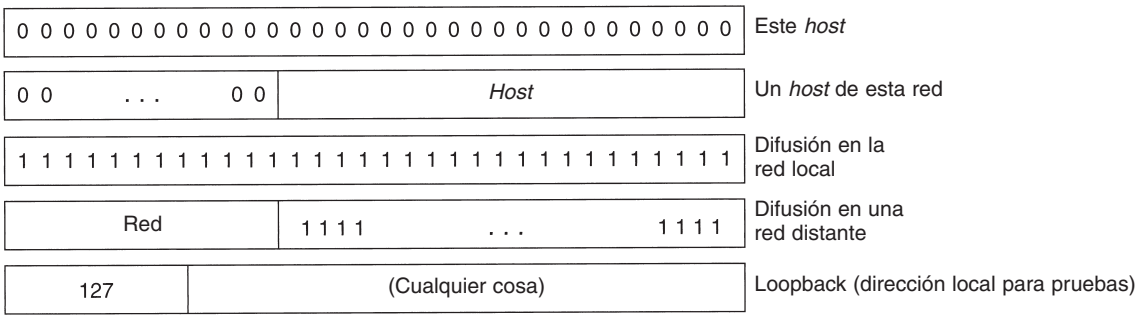


Figura 5-56. Direcciones IP especiales.

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).

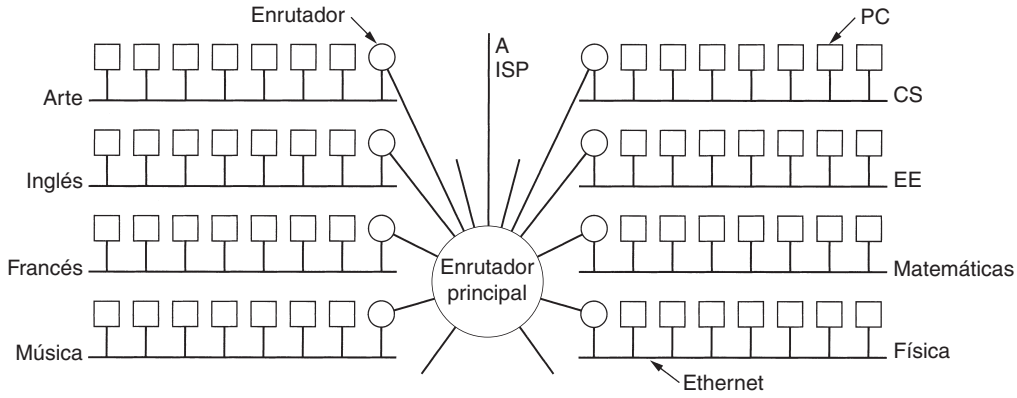


Figura 5-57. Una red de un campus que consiste de LANs para varios departamentos.

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.

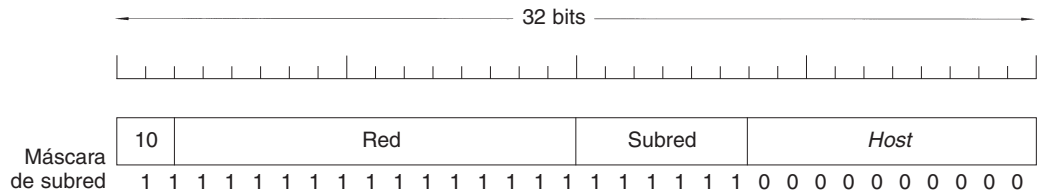


Figura 5-58. Una red de clase B dividida en 64 subredes.

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.

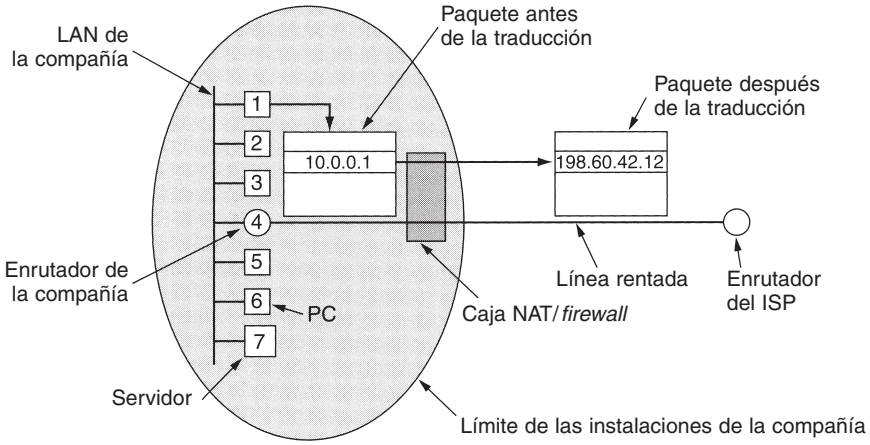


Figura 5-60. Colocación y funcionamiento de una caja NAT.

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

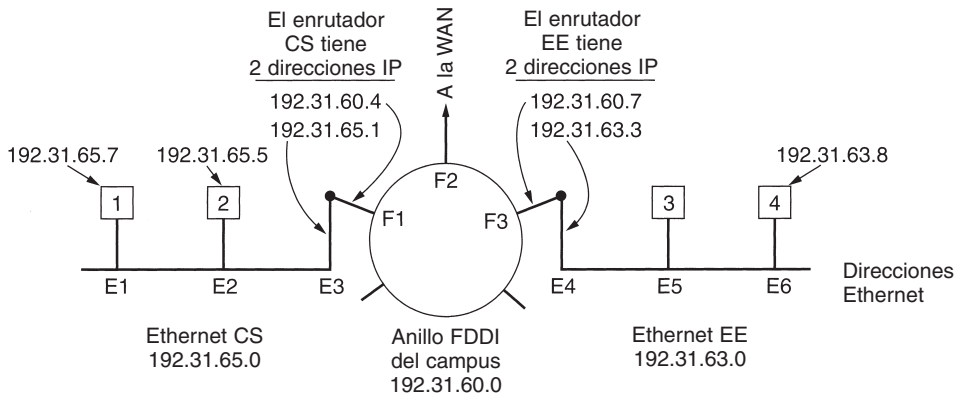


Figura 5-62. Tres redes /24 interconectadas: dos Ethernets y un anillo FDDI.

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

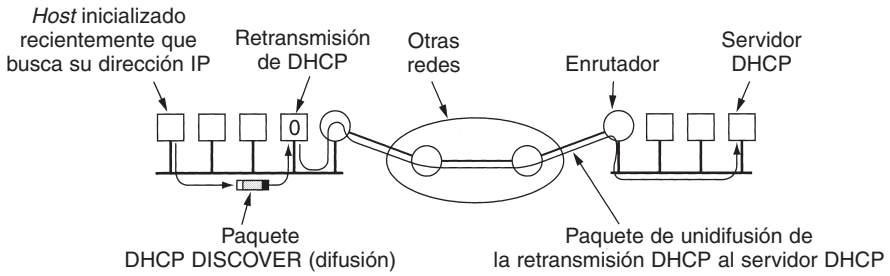


Figura 5-63. Funcionamiento de DHCP.

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

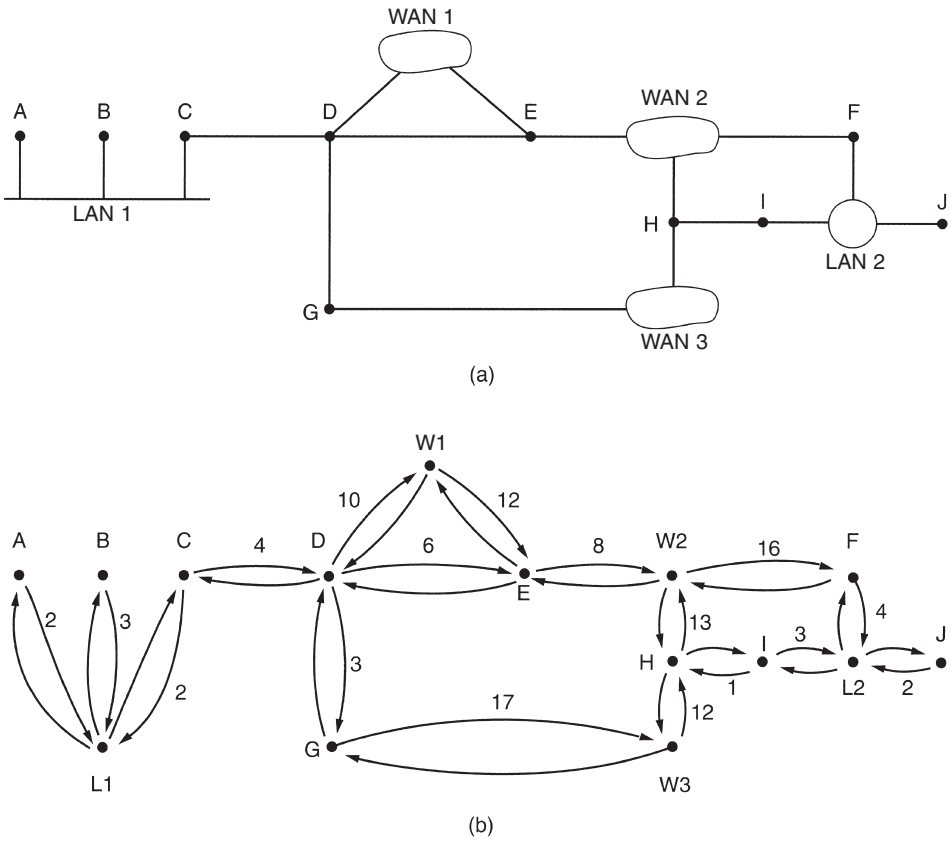


Figura 5-64. (a) Un sistema autónomo. (b) Representación gráfica de (a).

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.

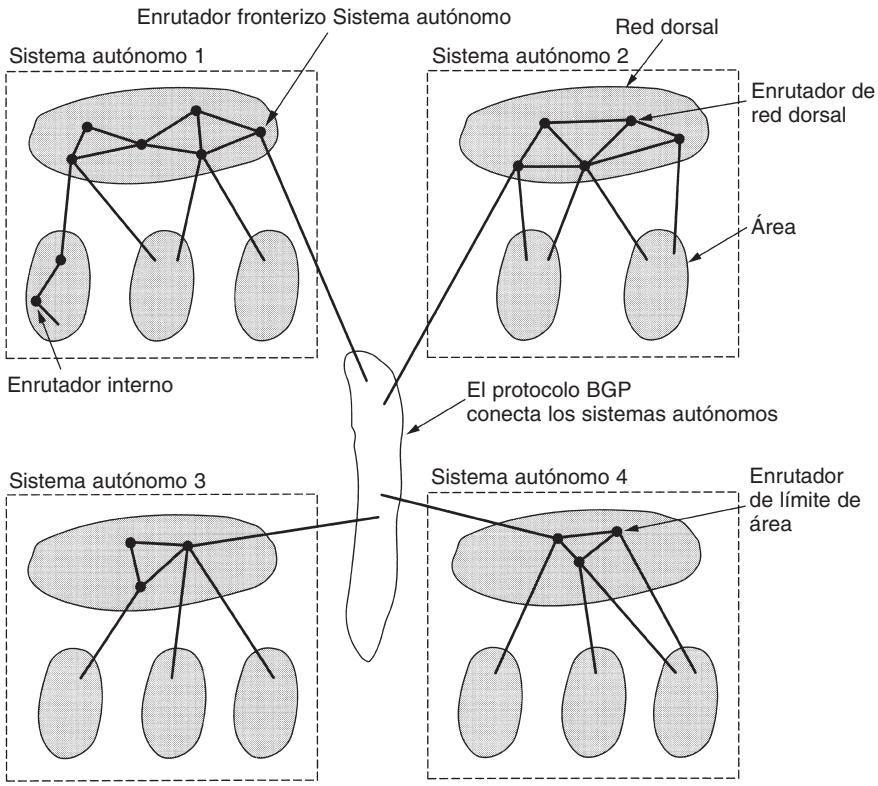


Figura 5-65. Relación entre sistemas autónomos, redes dorsales y áreas en OSPF.

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.

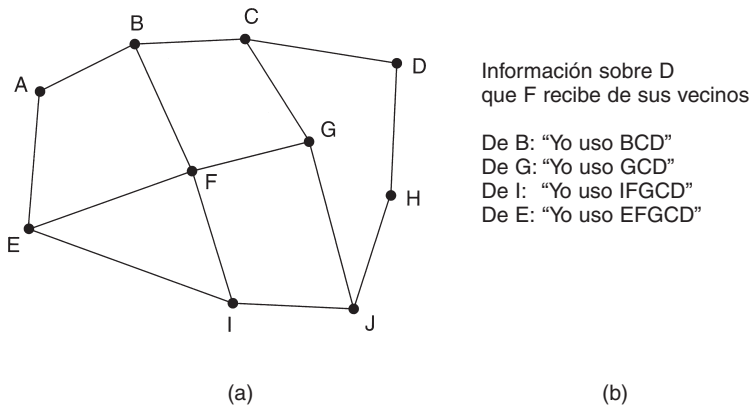


Figura 5-67. (a) Conjunto de enrutadores de BGP. (b) Información enviada a F.

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,

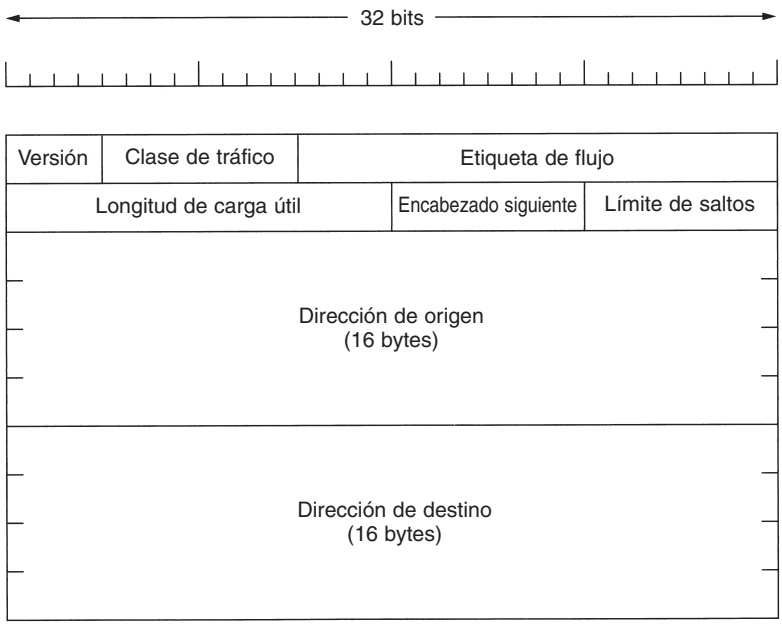


Figura 5-68. Encabezado fijo del IPv6 (obligatorio).

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.

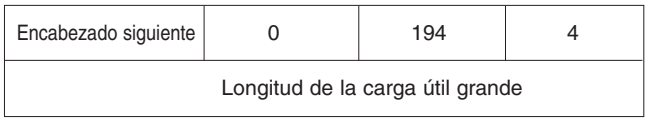


Figura 5-70. Encabezado de extensión salto por salto para datagramas grandes (jumbogramas).

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.

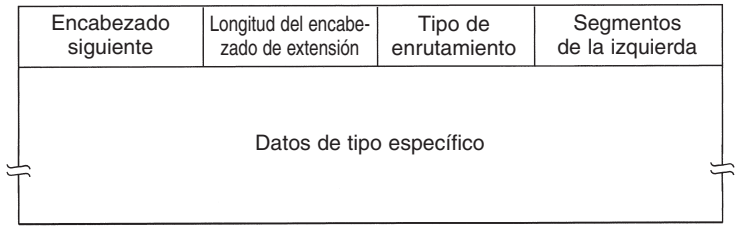


Figura 5-71. Encabezado de extensión para enrutamiento.

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.