CAPÍTULO 5

La capa de red: el plano de control

En este capítulo, completaremos nuestro viaje a través de la capa de red hablando del segundo de sus constituyentes, el **plano de control**: la lógica de la red que controla no solo cómo se reenvía un datagrama de un router a otro, a lo largo de una ruta extremo a extremo desde el host de origen al host de destino, sino también cómo se configuran y gestionan los servicios y componentes de la capa de red. En la Sección 5.2 hablaremos de los algoritmos tradicionales de enrutamiento utilizados para calcular las rutas de coste mínimo en un grafo; estos algoritmos son la base de dos protocolos de enrutamiento ampliamente implantados en Internet: OSPF y BGP, de los que hablaremos en las secciones 5.3 y 5.4, respectivamente. Como veremos, OSPF es un protocolo de enrutamiento que opera dentro de la red de un único ISP. BGP es un protocolo de enrutamiento que sirve para interconectar todas las redes de Internet; por ello se suele decir que BGP es el "pegamento" que mantiene unida Internet. Tradicionalmente, los protocolos de enrutamiento del plano de control se han implementado junto con las funciones de reenvío del plano de datos monolíticamente, dentro de un router. Como vimos en la introducción al Capítulo 4, las redes definidas por software (SDN) separan claramente los planos de datos y de control, implementando las funciones del plano de control mediante un servicio "controlador" separado que es distinto, y remoto, con respecto a los componentes de reenvío de los routers que controla. Hablaremos de los controladores SDN en la Sección 5.5.

En las secciones 5.6 y 5.7, hablaremos un poco de las interioridades de la gestión de una red IP, presentando los protocolos ICMP (*Internet Control Message Protocol*, protocolo de mensajes de control de Internet) y SNMP (*Simple Network Management Protocol*, protocolo simple de gestión de red).

5.1 Introducción

Establezcamos rápidamente el contexto para nuestro estudio del plano de control de la red, recordando las Figuras 4.2 y 4.3. Allí vimos que la tabla de reenvío (en el caso del reenvío basado en el destino) y la tabla de flujo (en el caso del reenvío generalizado) eran los principales elementos que

enlazaban los planos de datos y de control de la capa de red. Dijimos que estas tablas especifican el comportamiento de reenvío del plano de datos local de un router. Vimos que, en el caso del reenvío generalizado, las acciones tomadas (Sección 4.4.2) podían incluir no solo reenviar un paquete a través de un puerto de salida del router, sino también eliminar un paquete, duplicar un paquete y/o reescribir los campos de cabecera de las capas 2, 3 o 4 del paquete.

En este capítulo estudiaremos cómo se calculan, mantienen e instalan estas tablas de reenvío y de flujo. En nuestra introducción a la capa de red en la Sección 4.1 aprendimos que existen dos posibles enfoques para hacerlo.

- *Control por router*. La Figura 5.1 ilustra el caso en el que se ejecuta un algoritmo de enrutamiento en todos y cada uno de los routers; cada router contiene tanto una función de reenvío como una función de enrutamiento. En cada router hay un componente de enrutamiento que se comunica con los componentes de enrutamiento de otros routers, con el fin de calcular los valores de su tabla de reenvío. Esta técnica del control por router se ha estado utilizando en Internet durante décadas. Los protocolos OSPF y BGP que estudiaremos en las Secciones 5.3 y 5.4 están basados en este enfoque de control por router.
- *Control lógicamente centralizado*. La Figura 5.2 ilustra el caso en que un controlador lógicamente centralizado calcula y distribuye las tablas de reenvío que tienen que usar todos y cada uno de los routers. Como vimos en la Sección 4.4, la abstracción generalizada correspondencia-acción permite al router llevar a cabo tanto el reenvío IP tradicional, como un rico conjunto de otras funciones (compartición de carga, cortafuegos y NAT) que anteriormente se habían venido implementando mediante dispositivos intermediarios separados.

  El controlador interactúa con un agente de control (AC) que reside en cada router, a través de un protocolo bien definido, para configurar y gestionar la tabla de flujo de ese router. Normalmente, el AC tiene una funcionalidad mínima; su trabajo consiste en comunicarse con el controlador y en hacer lo que el controlador le ordene. A diferencia de los algoritmos de enrutamiento de la Figura 5.1, los AC no interactúan directamente entre sí, ni participan activamente en el cálculo de la tabla de reenvío. Esta es una distinción fundamental entre el control por router y el control lógicamente centralizado.

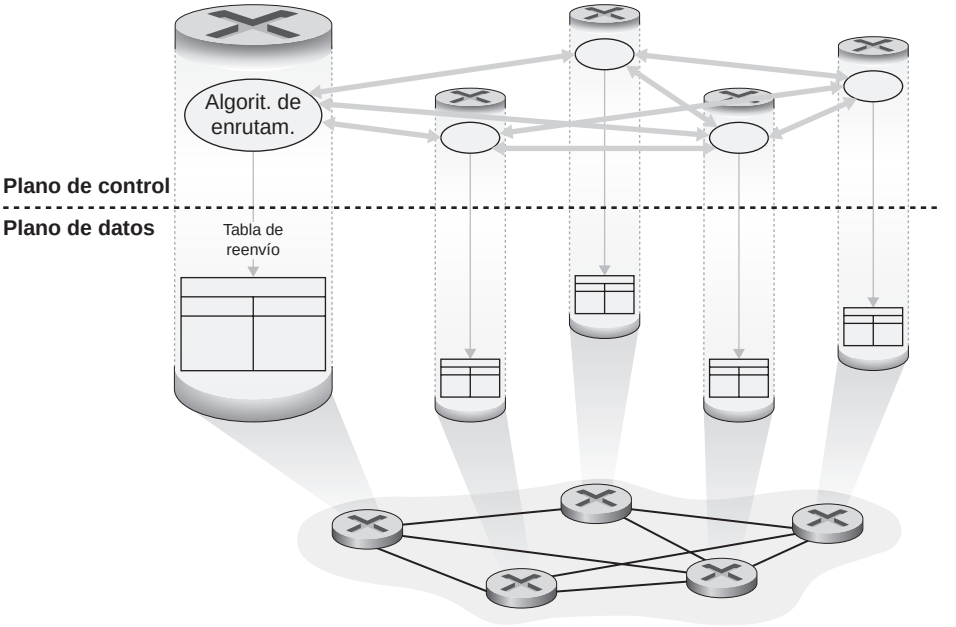


Figura 5.1 ♦ Control por router: los componentes individuales del algoritmo de enrutamiento interactúan en el plano de control.

Por control "lógicamente centralizado" [Levin 2012] queremos decir que al servicio de control de enrutamiento se accede como si fuera un único punto central de servicio, aunque es probable que el servicio se implemente mediante múltiples servidores, por razones de tolerancia a fallos y de escalabilidad del rendimiento. Como veremos en la Sección 5.5, la tecnología SDN adopta esta noción de un controlador lógicamente centralizado, una técnica que se está empleando cada vez más en instalaciones de producción. Google utiliza SDN para controlar los routers de su red de área extensa global B4, que interconecta sus centros de datos [Jain 2013]. SWAN [Hong 2013], de Microsoft Research, utiliza un controlador lógicamente centralizado para gestionar el enrutamiento y el reenvío entre una red de área extensa y una red de centros de datos. China Telecom y China Unicom están empleando SDN tanto dentro de los centros de datos como entre los centros de datos [Li 2015]. AT&T ha señalado [AT&T 2013] que "soporta muchas capacidades SDN, así como mecanismos propietarios, definidos de forma independiente, que encajan en el marco arquitectónico de SDN".

5.2 Algoritmos de enrutamiento

En esta sección estudiaremos los **algoritmos de enrutamiento**, cuyo objetivo es determinar buenas rutas desde los emisores a los receptores, a través de la red de routers. Normalmente, una "buena ruta" es aquella que tiene el coste mínimo. Sin embargo, veremos que, en la práctica, ciertos problemas del mundo real, como las políticas utilizadas (por ejemplo, una regla que establezca que "el router *x*, que pertenece a la organización *Y*, no debería reenviar ningún paquete

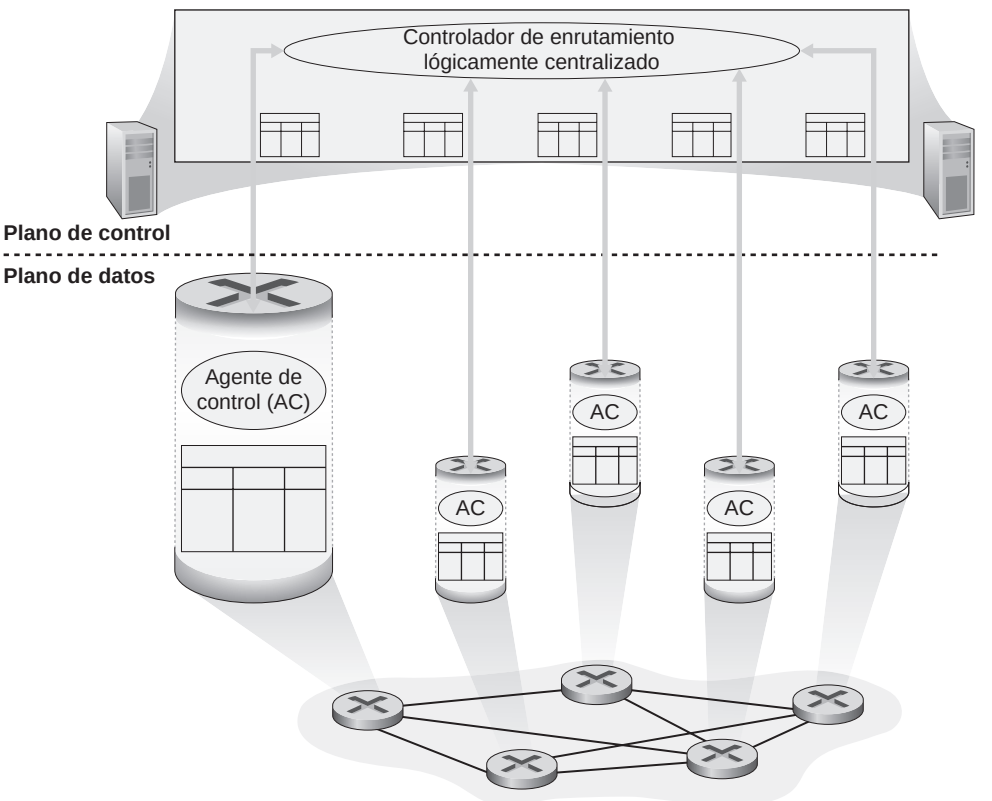


Figura 5.2 ♦ Control lógicamente centralizado: un controlador diferenciado, normalmente remoto, interactúa con agentes de control (AC) locales.

cuyo origen sea la red de la organización *Z*"), también entran en juego. Hay que resaltar que, independientemente de si el plano de control de la red adopta una solución de control por router o de control lógicamente centralizado, siempre debe existir una secuencia de routers bien definida que el paquete atravesará en su viaje desde el host emisor al host receptor. Por ello, los algoritmos de enrutamiento que calculan estas rutas tienen una importancia fundamental, siendo otro de los candidatos para nuestra lista de los 10 conceptos de redes más importantes.

Para formular los problemas de enrutamiento se utilizan grafos. Recuerde que un **grafo** $G = (N, E)$ es un conjunto N de nodos y una colección E de aristas, donde cada arista es una pareja de nodos de N. En el contexto del enrutamiento de la capa de red, los nodos del grafo representan los routers (los puntos en los que se toman las decisiones acerca del reenvío de los paquetes), mientras que las aristas que conectan esos nodos representan los enlaces físicos entre los routers. En la Figura 5.3 se muestra la abstracción en forma de grafo de una red de computadoras. Si desea ver algunos grafos que representan mapas de redes reales, consulte [Dodge 2016, Cheswick 2000]; para ver un estudio de cómo diferentes modelos basados en grafos permiten modelar Internet, consulte [Zegura 1997, Faloutsos 1999, Li 2004].

Como se muestra en la Figura 5.3, una arista también tiene un valor que representa su coste. Normalmente, el coste de una arista puede reflejar la longitud física del enlace correspondiente (por ejemplo, un enlace transoceánico tendría un coste mayor que un enlace terrestre de corto alcance), la velocidad del enlace o el coste monetario asociado con el enlace. Para nuestros propósitos, simplemente utilizaremos el coste del enlace como algo que nos viene dado, sin preocuparnos por cómo se determina. Para cualquier arista (x, y) de E, denominaremos $c(x, y)$ al coste de la arista que conecta los nodos x e y. Si el par (x, y) no pertenece a E, hacemos $c(x, y) = \infty$. Asimismo, solo vamos a considerar en nuestra exposición los grafos no dirigidos (es decir, grafos cuyas aristas no tienen una dirección), de modo que la arista (x, y) es la misma que la arista (y, x) y además $c(x, y) = c(y, x)$; sin embargo, los algoritmos que estudiaremos pueden generalizarse fácilmente para el caso de enlaces dirigidos que tengan un coste diferente en cada dirección. Por último, un nodo y se dice que es un **vecino** del nodo x si (x, y) pertenece a E.

Dado que las distintas aristas del grafo abstracto tienen costes asignados, un objetivo natural de un algoritmo de enrutamiento es identificar las rutas de coste mínimo entre los orígenes y los destinos. Con el fin de definir este problema de manera más precisa, recuerde que una **ruta** en un grafo $G = (N, E)$ es una secuencia de nodos $(x_1, x_2, ..., x_p)$ tal que cada una de las parejas (x_1, x_2), (x_2, x_3),..., (x_{p-1}, x_p) son aristas pertenecientes a E. El coste de una ruta $(x_1, x_2, ..., x_p)$ es simplemente la suma del coste de todas las aristas que componen la ruta; es decir, $c(x_1, x_2) + c(x_2, x_3) + ... + c(x_{p-1}, x_p)$. Dados dos nodos cualesquiera x e y, normalmente existen muchas rutas entre los dos nodos, teniendo cada una de ellas un coste. Una o más de estas rutas será una **ruta de coste mínimo**. Por tanto, el problema del coste mínimo está claro: hallar una ruta entre el origen y el destino que tenga el mínimo coste. Por ejemplo, en la Figura 5.3 la ruta de coste mínimo entre el nodo de origen u y el nodo de destino w es (u, x, y, w), con un coste de ruta igual a 3. Observe que si todas las aristas

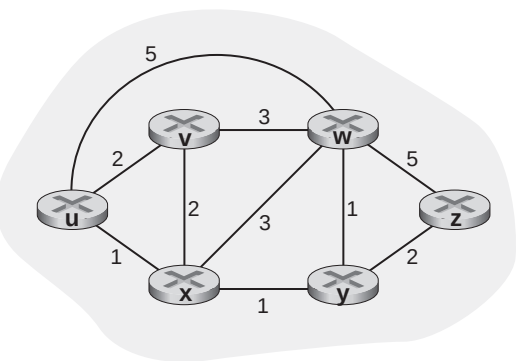


Figura 5.3 ♦ Modelo de grafo abstracto de una red de computadoras.

del grafo tienen el mismo coste, la ruta de coste mínimo es también la **ruta más corta** (es decir, la ruta con el número mínimo de enlaces entre el origen y el destino).

Como ejercicio sencillo, intente encontrar la ruta de coste mínimo desde el nodo *u* al *z* en la Figura 5.3 y reflexione sobre cómo ha calculado esa ruta. Si es usted como la mayor parte de la gente, habrá determinado la ruta de *u* a *z* examinando la Figura 5.3, trazando unas pocas rutas de *u* a *z*, y llegando de alguna manera al convencimiento de que la ruta que ha elegido es la de coste mínimo de entre todas las posibles rutas. (¿Ha comprobado las 17 rutas posibles entre *u* y *z*? ¡Probablemente no!). Este cálculo es un ejemplo de algoritmo de enrutamiento centralizado (el algoritmo de enrutamiento se ejecutó en un lugar, su cerebro, que dispone de la información completa de la red). En términos generales, una forma de clasificar los algoritmos de enrutamiento es dependiendo de si son centralizados o descentralizados.

- Un **algoritmo de enrutamiento centralizado** calcula la ruta de coste mínimo entre un origen y un destino utilizando un conocimiento global y completo acerca de la red. Es decir, el algoritmo toma como entradas la conectividad entre todos los nodos y todos los costes de enlace. Esto requiere, por tanto, que el algoritmo obtenga de alguna manera esta información antes de realizar el propio cálculo. El cálculo en sí puede hacerse en un solo sitio (por ejemplo, un controlador lógicamente centralizado, como el de la Figura 5.2), o replicarse en el componente de enrutamiento de todos y cada uno de los routers (por ejemplo, como en la Figura 5.1). La característica distintiva aquí, sin embargo, es que un algoritmo global dispone de toda la información acerca de la conectividad y de los costes de los enlaces. Los algoritmos con información de estado global a menudo se denominan **algoritmos de estado de enlaces** (**LS**, ***Link-State***), ya que el algoritmo tiene que ser consciente del coste de cada enlace de la red. Estudiaremos los algoritmos LS en la Sección 5.2.1.
- En un **algoritmo de enrutamiento descentralizado**, el cálculo de la ruta de coste mínimo se realiza por parte de los routers de manera iterativa y distribuida. Ningún nodo tiene toda la información acerca del coste de todos los enlaces de la red. En lugar de ello, al principio, cada nodo solo conoce los costes de sus propios enlaces directamente conectados. Después, a través de un proceso iterativo de cálculo e intercambio de información con sus nodos vecinos, cada nodo calcula gradualmente la ruta de coste mínimo hacia un destino o conjunto de destinos. El algoritmo de enrutamiento descentralizado que estudiaremos más adelante en la Sección 5.2.2 se denomina **algoritmo de vector de distancias (DV, *Distance-Vector*)**, porque cada nodo mantiene un vector de estimaciones de los costes (distancias) a todos los demás nodos de la red. Dichos algoritmos descentralizados, con intercambio interactivo de mensajes entre routers vecinos, está quizá adaptado de una forma más natural a los planos de control en los que los routers interactúan directamente entre sí, como en la Figura 5.1.

Una segunda forma general de clasificar los algoritmos de enrutamiento es según sean estáticos o dinámicos. En los **algoritmos de enrutamiento estático**, las rutas cambian muy lentamente con el tiempo, a menudo como resultado de una intervención humana (por ejemplo, una persona que edita manualmente la tabla de reenvío de un router). Los **algoritmos de enrutamiento dinámico** modifican los caminos de enrutamiento a medida que la carga de tráfico o la topología de la red cambian. Un algoritmo dinámico puede ejecutarse periódicamente o como respuesta directa a cambios en la topología o en el coste de los enlaces. Aunque los algoritmos dinámicos responden mejor a los cambios de la red, también son más susceptibles a problemas como los bucles de enrutamiento y la oscilación de rutas.

Una tercera forma de clasificar los algoritmos de enrutamiento es según sean sensibles o no a la carga. En un **algoritmo sensible a la carga**, los costes de enlace varían de forma dinámica para reflejar el nivel actual de congestión en el enlace subyacente. Si se asocia un coste alto con un enlace que actualmente esté congestionado, el algoritmo de enrutamiento tenderá a elegir rutas que eviten tal enlace congestionado. Aunque los primeros algoritmos de enrutamiento de ARPAnet eran sensibles a la carga [McQuillan 1980], se encontró una serie de dificultades [Huitema 1998].

Los algoritmos de enrutamiento actuales de Internet (como RIP, OSPF y BGP) son **algoritmos no sensibles a la carga**, ya que el coste de un enlace no refleja explícitamente su nivel actual (o reciente) de congestión.

5.2.1 Algoritmo de enrutamiento de estado de enlaces (LS)

Recuerde que en un algoritmo de estado de enlaces, la topología de la red y el coste de todos los enlaces son conocidos; es decir, están disponibles como entradas para el algoritmo LS. En la práctica, esto se consigue haciendo que cada nodo difunda paquetes del estado de los enlaces a *todos* los demás nodos de la red, conteniendo cada paquete de estado de enlaces las identidades y costes de sus enlaces conectados. En la práctica (por ejemplo, con el protocolo de enrutamiento OSPF de Internet, que estudiaremos en la Sección 5.3), esto suele realizarse mediante un **algoritmo de difusión de estado de los enlaces** [Perlman 1999]. El resultado de difundir la información por parte de los nodos es que todos los nodos tienen una visión completa e idéntica de la red. Cada nodo puede entonces ejecutar el algoritmo LS y calcular el mismo conjunto de rutas de coste mínimo que cualquier otro nodo.

El algoritmo de enrutamiento de estado de enlaces que presentamos a continuación se conoce como *algoritmo de Dijkstra*, en honor a su inventor. Un algoritmo estrechamente relacionado con él es el algoritmo de Prim; consulte [Cormen 2001] para ver una exposición de carácter general sobre los algoritmos de grafos. El algoritmo de Dijkstra calcula la ruta de coste mínimo desde un nodo (el origen, al que denominaremos u) hasta todos los demás nodos de la red. El algoritmo de Dijkstra es iterativo y tiene la propiedad de que después de la k-ésima iteración del algoritmo se conocen las rutas de coste mínimo hacia k nodos de destino, y entre las rutas de coste mínimo a todos los nodos de destino, estas k rutas tendrán los k costes más pequeños. Definimos la siguiente notación:

- $D(v)$: coste de la ruta de coste mínimo desde el nodo de origen al destino v, para esta iteración del algoritmo.
- $p(v)$: nodo anterior (vecino de v) a lo largo de la ruta de coste mínimo desde el origen hasta v.
- N': subconjunto de nodos; v pertenece a N' si la ruta de coste mínimo desde el origen hasta v se conoce de forma definitiva.

El algoritmo de enrutamiento centralizado consta de un paso de inicialización seguido de un bucle. El número de veces que se ejecuta el bucle es igual al número de nodos de la red. Al terminar, el algoritmo habrá calculado las rutas más cortas desde el nodo de origen u hasta cualquier otro nodo de la red.

Algoritmo de estado de enlaces (LS) para el nodo de origen u

```
1  Inicialización:
2   N' = {u}
3   for todo nodo v
4     if v es un vecino de u
5       then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9   encontrar w no perteneciente a N' tal que D(w) sea un mínimo
10  añadir w a N'
11  actualizar D(v) para cada vecino v de w que no pertenezca a N':
12        D(v) = min(D(v), D(w)+ c(w,v) )
13   /* el nuevo coste a v es o bien el antiguo coste a v o el coste
14    de la ruta de coste mínimo a w más el coste desde w a v */
15 until N'= N
```

Como ejemplo, consideremos la red de la Figura 5.3 y calculemos las rutas de coste mínimo desde *u* a todos los destinos posibles. En la Tabla 5.1 se muestra una tabla de resumen de los cálculos del algoritmo, donde cada línea de la tabla proporciona los valores de las variables del algoritmo al final de la iteración. Veamos en detalle los primeros pasos.

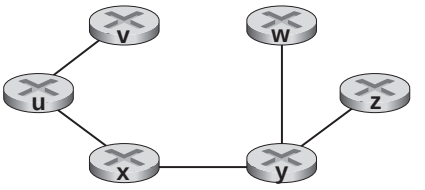
- En el paso de inicialización, las rutas de coste mínimo actualmente conocidas desde *u* a sus vecinos conectados directamente, *v*, *x* y *w*, se inicializan a 2, 1 y 5, respectivamente. En particular, fíjese en que el coste a *w* se define como 5 (aunque pronto veremos que, de hecho, existe una ruta de coste más pequeño), ya que este es el coste del enlace directo (un salto) de *u* a *w*. Los costes a *y* y *z* se hacen igual a infinito porque no están directamente conectados a *u*.
- En la primera iteración, buscamos entre aquellos nodos que todavía no se han añadido al conjunto N' y localizamos el nodo que tiene el coste mínimo después de finalizar la iteración previa. Dicho nodo es *x*, con un coste de 1, y por tanto *x* se añade al conjunto N'. La línea 12 del algoritmo LS se ejecuta entonces para actualizar $D(v)$ para todos los nodos *v*, proporcionando los resultados mostrados en la segunda línea (Paso 1) de la Tabla 5.1. El coste de la ruta a *v* no varía. Se determina que el coste de la ruta a *w* (que era 5 al final de la inicialización) a través del nodo *x* es ahora igual a 4. Por ello, se selecciona esta ruta de coste mínimo y el predecesor de *w* a lo largo de la ruta más corta desde *u* se establece en *x*. De forma similar, se calcula el coste a *y* (a través de *x*) y se obtiene que es 2, actualizándose la tabla de acuerdo con ello.
- En la segunda iteración, se determina que los nodos *v* e *y* tienen las rutas de coste mínimo (2) y se deshace el empate arbitrariamente, añadiendo *y* al conjunto N', de modo que ahora N' contiene a *u*, *x* e *y*. El coste de los restantes nodos que todavía no pertenecen a N', es decir, los nodos *v*, *w* y *z*, se actualiza en la línea 12 del algoritmo LS, dando los resultados mostrados en la tercera fila de la Tabla 5.1.
- Y así sucesivamente. . .

Cuando el algoritmo LS termina, tenemos para cada nodo su predecesor a lo largo de la ruta de coste mínimo desde el nodo de origen. Para cada predecesor, también tenemos *su* predecesor, y así de este modo podemos construir la ruta completa desde el origen a todos los destinos. La tabla de reenvío de un nodo, por ejemplo del nodo *u*, puede entonces construirse a partir de esta información, almacenando para cada destino el nodo del siguiente salto en la ruta de coste mínimo desde *u* al destino. La Figura 5.4 muestra las rutas de coste mínimo resultantes y la tabla de reenvío de *u* para la red de la Figura 5.3.

¿Cuál es la complejidad de cálculo de este algoritmo? Es decir, dados *n* nodos (sin contar el origen) ¿cuántos cálculos hay que realizar, en el caso peor, para determinar las rutas de coste mínimo desde el origen a todos los destinos? En la primera iteración, tenemos que buscar a través de los *n* nodos para determinar el nodo *w* que no pertenece a N' y que tiene el coste mínimo. En la segunda iteración, tenemos que comprobar $n - 1$ nodos para determinar el coste mínimo; en la tercera iteración, hay que comprobar $n - 2$ nodos, y así sucesivamente. En general, el número total

Paso	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Tabla 5.1 ♦ Ejecución del algoritmo de estado de enlaces para la red de la Figura 5.3.

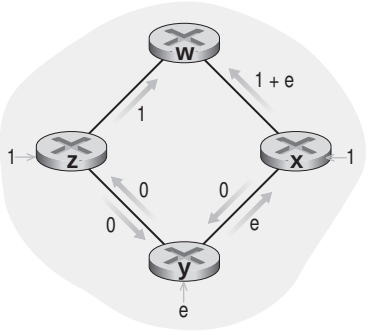


Destino	Enlace
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

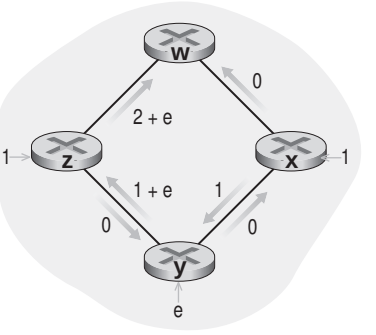
Figura 5.4 ♦ Rutas de coste mínimo y tabla de reenvío para el nodo u.

de nodos a través de los que tenemos que buscar, teniendo en cuenta todas las iteraciones, es igual a $n(n + 1)/2$ y, por tanto, decimos que la anterior implementación del algoritmo LS tiene, en el caso peor, una complejidad de orden n al cuadrado: $O(n^2)$. Una implementación más sofisticada de este algoritmo, que utiliza una estructura de datos conocida como montón (*heap*), puede calcular el mínimo en la línea 9 en tiempo logarítmico en lugar de lineal, reduciendo así la complejidad.

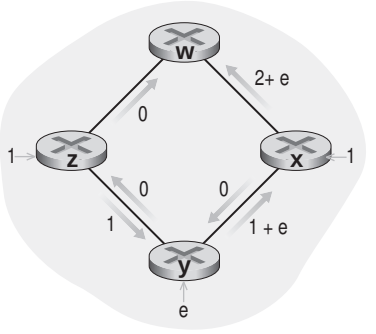
Antes de terminar nuestro estudio del algoritmo LS, consideremos una patología que puede surgir. La Figura 5.5 muestra una topología de red simple donde el coste de cada enlace es igual a la carga transportada por el enlace, reflejando, por ejemplo, el retardo que se experimentaría en ese enlace. En este ejemplo, los costes de los enlaces no son simétricos; es decir, $c(u,v)$ es igual a $c(v,u)$ solo si la carga transportada en ambas direcciones del enlace (u,v) es la misma. En este ejemplo, el nodo z origina una unidad de tráfico destinada a w, el nodo x también origina una unidad de



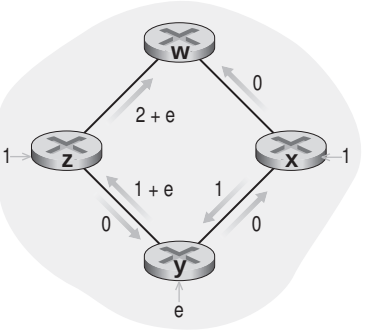
a. Enrutamiento inicial



b. *x*, *y* detectan una ruta mejor hacia *w*, en sentido horario



c. *x*, *y*, *z* detectan una ruta mejor hacia *w*, en sentido antihorario



d. *x*, *y*, *z* detectan una ruta mejor hacia *w*, en sentido horario

Figura 5.5 ♦ Oscilaciones con enrutamiento sensible a la congestión.

tráfico destinada a *w*, y el nodo *y* inyecta una cantidad de tráfico igual a *e*, también destinado a *w*. El enrutamiento inicial se muestra en la Figura 5.5(a), correspondiendo los costes de los enlaces a la cantidad de tráfico transportado.

Cuando se vuelve a ejecutar el algoritmo LS, el nodo *y* determina, basándose en los costes de enlace mostrados en la Figura 5.5(a), que la ruta en sentido horario a *w* tiene un coste de 1, mientras que la ruta en sentido antihorario a *w* (que era la que había estado utilizando) tiene un coste de $1 + e$. Por tanto, la ruta de coste mínimo de *y* a *w* ahora va en sentido horario. De forma similar, *x* determina que ahora su nueva ruta de coste mínimo a *w* va en sentido horario, dando como resultado los costes mostrados en la Figura 5.5(b). Cuando el algoritmo LS se ejecuta otra vez, los nodos *x*, *y* y *z* detectan una ruta de coste cero a *w* en el sentido antihorario y todos ellos envían su tráfico a las rutas en sentido antihorario. La siguiente vez que se ejecuta el algoritmo LS, *x*, *y* y *z* enrutan su tráfico a las rutas en sentido horario.

¿Qué podemos hacer para evitar tales oscilaciones (que pueden producirse en cualquier algoritmo, no solo el LS, que utilice una métrica de enlace basada en la congestión o el retardo)? Una solución sería obligar a que los costes de enlace no dependan de la cantidad de tráfico transportado (una solución inaceptable, ya que uno de los objetivos del enrutamiento es evitar los enlaces altamente congestionados; por ejemplo, los enlaces con un alto retardo). Otra solución consiste en garantizar que no todos los routers ejecuten el algoritmo LS al mismo tiempo. Esta parece una solución más razonable, ya que podríamos esperar que, aunque los routers ejecuten el algoritmo LS con la misma periodicidad, la instancia de ejecución del algoritmo no sería la misma en cada uno de los nodos. Es interesante observar que los investigadores han comprobado que los routers de Internet pueden auto-sincronizarse entre ellos [Floyd Synchronization 1994]. Es decir, incluso aunque inicialmente ejecuten el algoritmo con el mismo periodo pero en distintos instantes de tiempo, la instancia de ejecución del algoritmo puede llegar a sincronizarse en los routers y permanecer sincronizada. Una forma de evitar tal auto-sincronización es que cada router elija aleatoriamente el instante en el que enviar un anuncio de enlace.

Ahora que ya hemos estudiado el algoritmo LS, vamos a abordar el otro algoritmo de enrutamiento importante que se utiliza actualmente en la práctica: el algoritmo de enrutamiento por vector de distancias.

5.2.2 Algoritmo de enrutamiento por vector de distancias (DV)

Mientras que el algoritmo LS es un algoritmo que emplea información global, el **algoritmo por vector de distancias (DV)** es iterativo, asíncrono y distribuido. Es *distribuido* en el sentido de que cada nodo recibe información de uno o más de sus vecinos *directamente conectados*, realiza un cálculo y luego distribuye los resultados de su cálculo de vuelta a sus vecinos. Es *iterativo* porque este proceso continúa hasta que ya no se intercambia más información entre los vecinos. (Resulta interesante observar que el algoritmo también finaliza por sí mismo, es decir, no existe ninguna señal que indique que los cálculos deberían detenerse; simplemente se detienen.) El algoritmo es *asíncrono* en el sentido de que no requiere que todos los nodos operen sincronizados entre sí. Como tendremos oportunidad de ver, ¡un algoritmo asíncrono, iterativo, distribuido y que finaliza por sí mismo es mucho más interesante y divertido que un algoritmo centralizado!

Antes de presentar el algoritmo de vector de distancias, es conveniente que veamos una relación importante que existe entre los costes de las rutas de coste mínimo. Sea $d_x(y)$ el coste de la ruta de coste mínimo desde el nodo *x* al nodo *y*. Entonces, los costes mínimos están relacionados mediante la conocida ecuación de Bellman-Ford,

$$d_x(y) = \min_v\{c(x,v) + d_v(y)\} \tag{5.1}$$

donde min_v se calcula para todos los vecinos de *x*. La ecuación de Bellman-Ford es bastante intuitiva. De hecho, después de viajar de *x* a *v*, si tomamos la ruta de coste mínimo de *v* a *y*, el coste de la ruta será $c(x,v) + d_v(y)$. Puesto que hay que comenzar viajando a algún vecino *v*, el coste mínimo de *x* a *y* será el mínimo de $c(x,v) + d_v(y)$, calculado para todos los vecinos *v*.

Pero para aquellos que sean escépticos en cuanto a la validez de la ecuación, vamos a comprobarla para el nodo de origen *u* y el nodo de destino *z* de la Figura 5.3. El nodo de origen *u* tiene tres vecinos: los nodos *v*, *x* y *w*. Recorriendo las distintas rutas del grafo, es fácil ver que $d_v(z) = 5$, $d_x(z) = 3$ y $d_w(z) = 3$. Introduciendo estos valores en la Ecuación 5.1, junto con los costes $c(u,v) = 2$, $c(u,x) = 1$ y $c(u,w) = 5$, se obtiene $d_u(z) = \text{mín}\{2 + 5, 5 + 3, 1 + 3\} = 4$, que obviamente es cierto y es exactamente lo que nos proporcionó el algoritmo de Dijskstra para la misma red. Esta rápida verificación debería ayudarle a disipar cualquier duda que pueda tener.

La ecuación de Bellman-Ford no es únicamente una curiosidad intelectual; realmente tiene una importancia práctica significativa. En concreto, la solución de la ecuación de Bellman-Ford proporciona las entradas de la tabla de reenvío del nodo *x*. Para ver esto, sea *v** cualquier nodo vecino que alcanza el mínimo dado por la Ecuación 5.1. Entonces, si el nodo *x* desea enviar un paquete al nodo *y* a lo largo de la ruta de coste mínimo, debería en primer lugar reenviar el paquete al nodo *v**. Por tanto, la tabla de reenvío del nodo *x* especificaría el nodo *v** como el router del siguiente salto para el destino final *y*. Otra importante contribución práctica de la ecuación de Bellman-Ford es que sugiere la forma en que tendrá lugar la comunicación vecino a vecino en el algoritmo de vector de distancias.

La idea básica es la siguiente: cada nodo *x* comienza con $D_x(y)$, una estimación del coste de la ruta de coste mínimo desde sí mismo al nodo *y*, para todos los nodos *y* de *N*. Sea $\mathbf{D}_x = [D_x(y)$: *y* perteneciente a *N*] el vector de distancias del nodo *x*, que es el vector de coste estimado desde *x* a todos los demás nodos *y* pertenecientes a *N*. Con el algoritmo de vector de distancias, cada nodo *x* mantiene la siguiente información de enrutamiento:

- Para cada vecino*v,* el coste *c(x,v)* desde *x* al vecino directamente conectado *v.*
- El vector de distancias del nodo *x*, es decir, $\mathbf{D}_x = [D_x(y)$: *y* perteneciente a *N*], que contiene la estimación que *x* hace de su coste hacia todos los destinos *y* de *N*.
- Los vectores de distancias de cada uno de sus vecinos, es decir, $\mathbf{D}_v = [D_v(y)$: *y* perteneciente a *N*] para cada vecino *v* de *x*.

En el algoritmo asíncrono distribuido, cada nodo envía de vez en cuando una copia de su vector de distancias a cada uno de sus vecinos. Cuando un nodo *x* recibe un nuevo vector de distancias procedente de cualquiera de sus vecinos *w*, guarda dicho vector de *w* y luego utiliza la ecuación de Bellman-Ford para actualizar su propio vector de distancias como sigue:

$$D_x(y) = \min_v\{c(x, v) + D_v(y)\} \quad \text{para cada nodo } y \text{ perteneciente a } N$$

Si el vector de distancias del nodo *x* ha cambiado como resultado de este paso de actualización, entonces el nodo *x* enviará su vector de distancias actualizado a cada uno de sus vecinos, lo que puede a su vez actualizar sus propios vectores de distancias. De forma bastante milagrosa, siempre y cuando todos los nodos continúen intercambiando sus vectores de distancia de forma asíncrona, ¡cada coste estimado $D_x(y)$ converge a $d_x(y)$, el coste real de la ruta de coste mínimo del nodo *x* al nodo *y* [Bertsekas 1991]!

Algoritmo por vector de distancias (DV)

En cada nodo *x*:

```
1  Inicialización:
2    for todos los destinos y pertenecientes a N:
3       Dx(y)= c(x,y)/* si y no es un vecino, entonces c(x,y)= ∞  */
4    for cada vecino w
5       Dw(y) = ? for todos los destinos y pertenecientes a N
6    for cada vecino w
7       enviar vector de distancias Dx = [Dx(y): y perteneciente a N] a w
```

```
8
9  loop
10    wait (hasta ver una variación en el coste de enlace a un vecino w
11          o hasta recibir un vector de distancias de algún vecino w)
12
13    for cada y perteneciente a N:
14        Dx(y) = minv{c(x,v) + Dv(y)}
15
16 if Dx(y) ha variado para cualquier destino y
17    enviar vector de distancia Dx = [Dx(y): y perteneciente a N] a todos los vecinos
18
19 forever
```

En el algoritmo de vector de distancias, un nodo *x* actualiza la estimación de su vector de distancias si se produce un cambio en el coste de uno de sus enlaces directamente conectados o si recibe una actualización de un vector de distancias de algún vecino. Pero para actualizar su propia tabla de reenvío para un determinado destino *y*, lo que realmente necesita saber el nodo *x* no es la distancia de la ruta más corta a *y*, sino el nodo vecino $v^*(y)$, que es el router del siguiente salto a lo largo de la ruta más corta a *y*. Como es lógico, el router del siguiente salto $v^*(y)$ es el vecino *v* que alcanza el mínimo en la línea 14 del algoritmo DV. (Si existen varios vecinos *v* que alcanzan el mínimo, entonces $v^*(y)$ puede ser cualquiera de esos vecinos.) Por tanto, en las líneas 13–14, para cada destino *y* el nodo *x* también determina $v^*(y)$ y actualiza su tabla de reenvío para el destino *y*.

Recuerde que al algoritmo LS es un algoritmo centralizado, en el sentido de que requiere que cada nodo obtenga en primer lugar un mapa completo de la red antes de ejecutar el algoritmo de Dijkstra. El algoritmo DV es un algoritmo *descentralizado* y no utiliza dicha información global. De hecho, la única información que tendrá un nodo es el coste de los enlaces a los vecinos a los que está directamente conectado y la información que recibe de esos vecinos. Cada nodo espera a recibir una actualización de cualquier vecino (líneas 10–11), calcula su nuevo vector de distancias cuando recibe una actualización (línea 14) y distribuye su nuevo vector de distancias a sus vecinos (líneas 16–17). En la práctica, los algoritmos del tipo vector de distancias se utilizan en muchos protocolos de enrutamiento, entre los que se incluyen los protocolos RIP y BGP de Internet, ISO IDRP, Novell IPX y el ARPAnet original.

La Figura 5.6 ilustra el funcionamiento del algoritmo DV para el caso de la red simple de tres nodos mostrada en la parte superior de la figura. El funcionamiento del algoritmo se ilustra de manera síncrona, en la que todos los nodos reciben simultáneamente vectores de distancias de sus vecinos, calculan sus nuevos vectores de distancias e informan a sus vecinos si sus vectores de distancias han cambiado. Después de estudiar este ejemplo, puede comprobar usted mismo que el algoritmo también funciona correctamente en el modo asíncrono, es decir, efectuándose cálculos en los nodos y generándose y recibiéndose actualizaciones en cualquier instante.

La columna de más a la izquierda de la figura muestra tres **tablas de enrutamiento** iniciales para cada uno de los tres nodos. Por ejemplo, la tabla de la esquina superior izquierda corresponde a la tabla de enrutamiento inicial del nodo *x*. Dentro de una tabla de enrutamiento concreta, cada fila es un vector de distancias (específicamente, la tabla de enrutamiento de cada nodo incluye su propio vector de distancias y el de cada uno de sus vecinos). Por tanto, la primera fila de la tabla de enrutamiento inicial del nodo *x* es $\boldsymbol{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. La segunda y tercera filas de esta tabla son los vectores de distancias recibidos más recientemente de los nodos *y* y *z*, respectivamente. Puesto que durante la inicialización el nodo *x* no ha recibido nada aún del nodo *y* ni del *z*, las entradas de la segunda y de la tercera filas se inicializan con infinito.

Después de la inicialización, cada nodo envía su vector de distancias a cada uno de sus dos vecinos. Esto se indica en la Figura 5.6 mediante las flechas que salen de la primera columna de las tablas y van hasta la segunda columna. Por ejemplo, el nodo *x* envía su vector de distancias

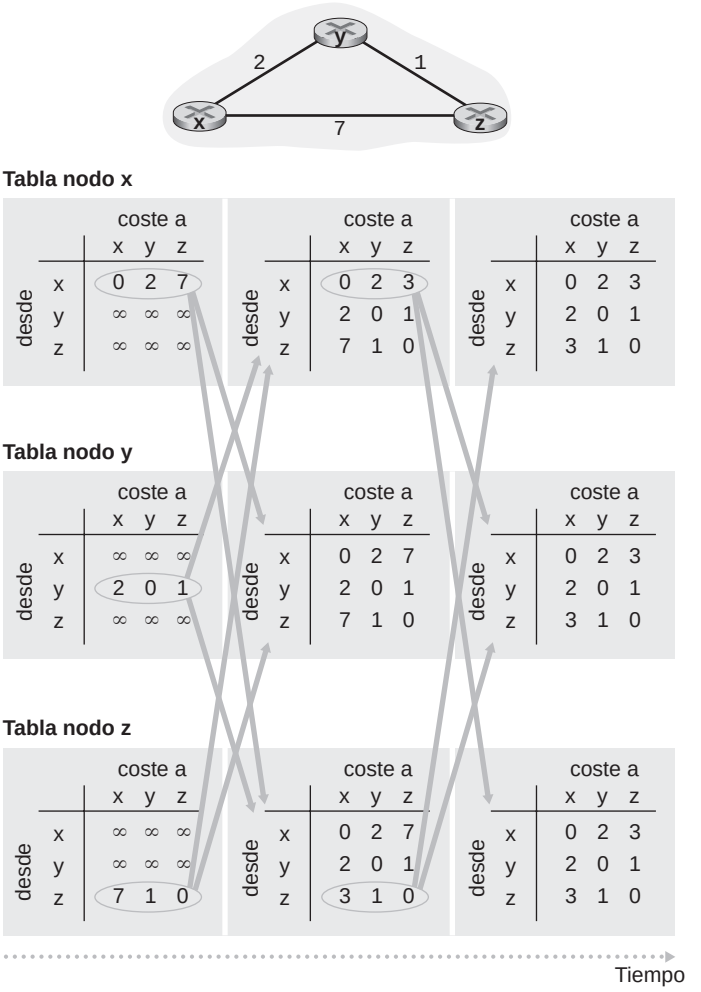


Figura 5.6 ♦ Algoritmo de vector de distancias (DV).

$\boldsymbol{D}_x = [0, 2, 7]$ a los nodos *y* y *z*. Después de recibir las actualizaciones, cada nodo recalcula su propio vector de distancias. Por ejemplo, el nodo *x* calcula

$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x, y) + D_y(y), c(x, z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x, y) + D_y(z), c(x, z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

Por tanto, la segunda columna muestra, para cada nodo, el nuevo vector de distancias del nodo, junto con los vectores de distancias que acaba de recibir de sus vecinos. Observe, por ejemplo, que la estimación del nodo *x* para el coste mínimo al nodo *z*, $D_x(z)$, ha cambiado de 7 a 3. Fíjese también en que para el nodo *x*, el nodo vecino *y* alcanza el mínimo en la línea 14 del algoritmo de vector de distancias; luego en esta etapa del algoritmo tenemos que, en el nodo *x*, $v^*(y) = y$ y $v^*(z) = y$.

Una vez que los nodos recalculan sus vectores de distancias, envían de nuevo los valores actualizados a sus vecinos (si se ha producido un cambio). Esto se indica en la Figura 5.6 mediante las flechas que van desde la segunda columna a la tercera columna de las tablas. Observe que únicamente los nodos *x* y *z* envían actualizaciones: el vector de distancias del nodo *y* no ha cambiado, por lo que no envía ninguna actualización. Después de recibir las actualizaciones, los nodos recalculan sus vectores de distancias y actualizan sus tablas de enrutamiento, lo que se muestra en la tercera columna.

El proceso de recibir vectores de distancias actualizados de los vecinos, recalcular las entradas de la tabla de enrutamiento e informar a los vecinos de los costes modificados de la ruta de coste mínimo hacia un destino continúa hasta que ya no se envían mensajes de actualización. En esta situación, puesto que no se envían mensajes de actualización, no se realizarán más cálculos de la tabla de enrutamiento y el algoritmo entrará en un estado de reposo; es decir, todos los nodos se encontrarán realizando la espera indicada por las líneas 10–11 del algoritmo del vector de distancias. El algoritmo permanece en el estado de reposo hasta que el coste de un enlace cambia, como se explica a continuación.

Algoritmo de vector de distancias: cambios en el coste de los enlaces y fallo de los enlaces

Cuando un nodo que ejecuta el algoritmo DV detecta un cambio en el coste del enlace desde sí mismo a un vecino (líneas 10–11), actualiza su vector de distancias (líneas 13–14) y, si existe un cambio en el coste de la ruta de coste mínimo, informa a sus vecinos (líneas 16–17) de su nuevo vector de distancias. La Figura 5.7(a) ilustra un escenario en el que el coste del enlace de *y* a *x* cambia de 4 a 1. Aquí vamos a centrarnos únicamente en las entradas de la tabla de distancias de *y* y *z* al destino *x*. El algoritmo de vector de distancias da lugar a la siguiente secuencia de sucesos:

- En el instante t_0, *y* detecta el cambio en el coste del enlace (el coste ha cambiado de 4 a 1), actualiza su vector de distancias e informa a sus vecinos de este cambio, puesto que su vector de distancias ha cambiado.
- En el instante t_1, *z* recibe la actualización de *y* y actualiza su tabla. Calcula el nuevo coste mínimo a *x* (ha disminuido de 5 a 2) y envía su nuevo vector de distancias a sus vecinos.
- En el instante t_2, *y* recibe la actualización de *z* y actualiza su tabla de distancias. El coste mínimo de *y* no cambia y, por tanto, *y* no envía ningún mensaje a *z*. El algoritmo entra en el estado de reposo.

Así, solo se han necesitado dos iteraciones para que el algoritmo DV alcance un estado de reposo. Las buenas noticias acerca de la disminución del coste entre *x* e *y* se han propagado rápidamente a través de la red.

Consideremos ahora lo que ocurre cuando el coste de un enlace *aumenta*. Suponga que el coste del enlace entre *x* e *y* aumenta de 4 a 60, como se muestra en la Figura 5.7(b).

1. Antes de que el coste del enlace varíe, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$ y $D_z(x) = 5$. En el instante t_0 *y* detecta el cambio en el coste del enlace (el coste ha variado de 4 a 60). El nodo *y* calcula su nueva ruta de coste mínimo a *x*, obteniendo un coste de:

$$D_y(x) = \text{mín}\{c(y, x) + D_x(x), c(y, z) + D_z(x)\} = \text{mín}\{60 + 0, 1 + 5\} = 6$$

   Por supuesto, con nuestra visión global de la red, podemos ver que este nuevo coste a través de *z* es *erróneo*. Pero la única información que tiene el nodo *y* es que su coste directo a *x* es 60 y que *z* le ha dicho a *y* que *z* podría alcanzar *x* con un coste de 5. Por tanto, para llegar a *x*, *y* ahora

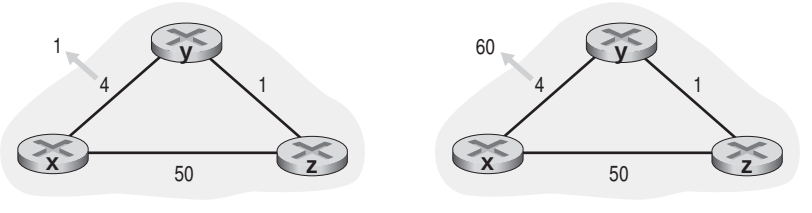


Figura 5.7 ♦ Cambios en el coste del enlace.

enrutaría a través de *z*, confiando plenamente en que *z* será capaz de llegar hasta *x* con un coste de 5. En t_1 tenemos por tanto un **bucle de enrutamiento** (para llegar a *x*, *y* enruta a través de *z*, y *z* enruta a través de *y*). Un bucle de enrutamiento es como un agujero negro (un paquete destinado a *x* que llega a *y* o *z* en t_1 rebotará entre estos dos nodos permanentemente, o hasta que las tablas de reenvío cambien).

2. Dado que el nodo *y* ha calculado un nuevo coste mínimo a *x*, informa a *z* de este nuevo vector de distancias en el instante t_1.
3. En algún momento posterior a t_1, *z* recibe un nuevo vector de distancias de *y*, que indica que el coste mínimo de *y* a *x* es 6. *z* sabe que puede llegar a y con un coste de 1 y, por tanto, calcula un nuevo coste mínimo a *x* que será igual a $D_z(x) = \text{mín}\{50 + 0, 1 + 6\} = 7$. Puesto que el coste mínimo de *z* a *x* ha aumentado, a continuación informa a *y* de su nuevo vector de distancias en t_2.
4. De forma similar, después de recibir el nuevo vector de distancias de *z*, el nodo *y* determina que $D_y(x) = 8$ y envía a *z* su vector de distancias. El nodo *z* determina entonces que $D_z(x) = 9$ y envía a *y* su vector de distancias, y así sucesivamente.

¿Durante cuánto tiempo continuará el proceso? Puede comprobar por sí mismo que el bucle se mantendrá durante 44 iteraciones (intercambios de mensajes entre *y* y *z*), hasta que *z* finalmente calcule que el coste de su ruta a través de *y* es mayor que 50. En ese punto, *z* (¡finalmente!) determinará que su ruta de coste mínimo a *x* es a través de su conexión directa con *x*. El nodo *y* entonces enrutará hacia *x* a través de *z*. ¡Como puede ver, las malas noticias acerca del aumento del coste del enlace han tardado mucho en propagarse! ¿Qué habría ocurrido si el coste del enlace $c(y, x)$ hubiera cambiado de 4 a 10.000 y el coste $c(z, x)$ hubiera sido 9.999? A causa de los escenarios de este tipo, en ocasiones se designa a este problema con el nombre de problema de la cuenta hasta infinito.

Algoritmo de vector de distancias: adición de la inversa envenenada

El escenario concreto de bucle que acabamos de describir puede evitarse utilizando una técnica conocida como *inversa envenenada* (*poisoned reverse*). La idea es simple: si *z* enruta a través de *y* para llegar al destino *x*, entonces *z* anunciará a *y* que $D_z(x) = \infty$ (incluso aunque *z* sepa que, en realidad, $D_z(x) = 5$). El nodo *z* continuará contando esta pequeña mentira a *y* mientras continúe enrutando hacia *x* a través de *y*. Dado que *y* cree que *z* no dispone de una ruta hacia *x*, el nodo *y* nunca intentará enrutar hacia *x* a través de *z*, mientras *z* continúe enrutando hacia *x* a través de *y* (y mintiendo acerca de ello).

Veamos ahora cómo la inversa envenenada resuelve el problema concreto del bucle encontrado antes, en la Figura 5.5(b). Como resultado de la inversa envenenada, la tabla de distancias de *y* indica que $D_z(x) = \infty$. Cuando el coste del enlace (x, y) cambia de 4 a 60 en el instante t_0, el nodo *y* actualiza su tabla y continúa enrutando directamente a *x*, aunque a un coste mayor, de 60, e informa a *z* de su nuevo coste a *x*, es decir, $D_y(x) = 60$. Después de recibir la actualización en t_1, *z* cambia inmediatamente su ruta a *x* para que sea a través del enlace directo (z, x) con un coste de 50. Puesto que esta es la nueva ruta de coste mínimo a *x*, y dado que la ruta ya no pasa a través de *y*, ahora *z* informa en t_2 a *y* de que $D_z(x) = 50$. Después de recibir la actualización de *z*, el nodo *y* actualiza su tabla de distancias con $D_y(x) = 51$. Además, dado que ahora *z* está en la ruta de coste mínimo de *y* a *x*, el nodo *y* envenena la ruta inversa de *z* a *x*, informando a *z* en el instante t_3 de que $D_y(x) = \infty$ (aunque *y* sepa que, en realidad, $D_y(x) = 51$).

¿Resuelve la inversa envenenada el problema general de la cuenta hasta infinito? No. El lector puede comprobar por sí mismo que los bucles que implican a tres o más nodos (en lugar de simplemente a dos nodos vecinos) no serán detectados por la técnica de la inversa envenenada.

Comparación de los algoritmos de enrutamiento LS y DV

Los algoritmos de vector de distancias y de estado de enlaces utilizan métodos complementarios para el cálculo de las rutas. Con el algoritmo de vector de distancias, cada nodo *solo* se comunica

con sus vecinos directamente conectados, pero les proporciona sus estimaciones de coste mínimo desde sí mismo a *todos* los demás nodos (conocidos) de la red. El algoritmo de estado de enlaces requiere información global. En consecuencia, cuando se lo implemente en todos y cada uno de los routers, como por ejemplo en las Figuras 4.2 y 5.1, cada nodo tendrá que comunicarse con todos los restantes nodos (vía difusión), pero solo les informará de los costes de sus enlaces directamente conectados. Vamos a terminar este estudio sobre los algoritmos de estado de enlaces y de vector de distancias haciendo una rápida comparación de algunos de sus atributos. Recuerde que N es el conjunto de nodos (routers) y E es el conjunto de aristas (enlaces).

- *Complejidad del mensaje*. Hemos visto que el algoritmo LS requiere que cada nodo conozca el coste de cada enlace de la red. Esto requiere el envío de $O(|N|\ |E|)$ mensajes. Además, cuando el coste de un enlace cambia, el nuevo coste tiene que enviarse a todos los nodos. El algoritmo de vector de distancias requiere intercambios de mensajes entre los vecinos directamente conectados en cada iteración. Hemos visto que el tiempo necesario para que el algoritmo converja puede depender de muchos factores. Cuando los costes de los enlaces cambian, el algoritmo de vector de distancias propagará los resultados del coste del enlace que ha cambiado solo si el nuevo coste de enlace da lugar a una ruta de coste mínimo distinta para uno de los nodos conectados a dicho enlace.
- *Velocidad de convergencia.* Hemos visto que nuestra implementación del algoritmo de estado de enlaces es un algoritmo $O(|N|^2)$ que requiere enviar $O(|N|\ |E|))$ mensajes. El algoritmo de vector de distancias puede converger lentamente y pueden aparecer bucles de enrutamiento mientras está convergiendo. Este algoritmo también sufre el problema de la cuenta hasta infinito.
- *Robustez.* ¿Qué puede ocurrir si un router falla, funciona mal o es saboteado? Con el algoritmo de estado de enlaces, un router podría difundir un coste incorrecto para uno de sus enlaces conectados (pero no para los otros). Un nodo también podría corromper o eliminar cualquier paquete recibido como parte de un mensaje de difusión LS. Pero, con el algoritmo LS, un nodo solo calcula su propia tabla de reenvío, mientras que otros nodos realizan cálculos similares por sí mismos. Esto significa que los cálculos de rutas son hasta cierto punto independientes en LS, proporcionando un mayor grado de robustez. Con el algoritmo de vector de distancias, un nodo puede anunciar rutas de coste mínimo incorrectas a uno o a todos los destinos. (De hecho, en 1997, un router que funcionaba mal en un pequeño ISP proporcionó a los routers troncales nacionales información de enrutamiento errónea. Esto hizo que otros routers inundaran con una gran cantidad de tráfico al router que funcionaba mal y provocó que amplias partes de Internet estuvieran desconectadas durante varias horas [Neumann 1997].) En un sentido más general, observamos que, en cada iteración, los cálculos de un nodo con el algoritmo de vector de distancias se pasan a sus vecinos y luego, indirectamente, al vecino del vecino en la siguiente iteración. En este sentido, con el algoritmo de vector de distancias, un cálculo de nodo incorrecto puede difundirse a través de toda la red.

En resumen, ningún algoritmo es el ganador evidente; de hecho, ambos algoritmos se utilizan en Internet.

5.3 Enrutamiento dentro de un sistema autónomo en Internet: OSPF

En nuestro estudio de los algoritmos de estado de enlaces y de vector de distancias, hemos visto la red simplemente como una colección de routers interconectados. Un router era indistinguible de otro, en el sentido de que los routers ejecutaban el mismo algoritmo de enrutamiento para calcular las rutas a través de la red completa. En la práctica, este modelo y la imagen de un conjunto homogéneo de routers que ejecutan todos ellos el mismo algoritmo de enrutamiento es un poco simplista, por al menos dos razones importantes:

- *Escala.* Cuando el número de routers comienza a hacerse grande, la sobrecarga implicada en la comunicación, cálculo y almacenamiento de la información de enrutamiento se hace prohibitiva. Actualmente, Internet consta de cientos de millones de routers. Almacenar la información de enrutamiento para todos los posibles destinos en cada uno de estos routers evidentemente requeriría enormes cantidades de memoria. ¡La sobrecarga requerida para difundir las actualizaciones de conectividad y de coste de los enlaces entre todos los routers sería inmensa! Seguramente, un algoritmo de vector de distancias que iterara entre tal enorme cantidad de routers nunca llegaría a converger. Evidentemente, es preciso hacer algo para reducir la complejidad del cálculo de rutas en una red tan grande como Internet.
- *Autonomía administrativa.* Como se describe en la Sección 1.3, Internet es una red de ISPs, estando compuesto cada ISP por su propia red de routers. Un ISP generalmente desea operar su red a su antojo (por ejemplo, emplear cualquier algoritmo de enrutamiento que elija dentro de su red) u ocultar al mundo exterior ciertos aspectos de la organización interna de su red. Idealmente, una organización debería poder operar y administrar su red como desee, sin por ello dejar de conectar su red a otras redes externas.

Estos dos problemas pueden resolverse organizando los routers en **sistemas autónomos (AS, *Autonomous System*)**, estando cada AS formado por un grupo de routers que se encuentran bajo el mismo control administrativo. A menudo, los routers de un ISP, y los enlaces que los interconectan, constituyen un único AS. Otros ISP, sin embargo, dividen su red en múltiples AS. En particular, algunos ISP de nivel 1 utilizan un solo AS gigante para toda su red, mientras que otros ISP dividen su red en decenas de AS interconectados. Cada sistema autónomo se identifica mediante su número de sistema autónomo (ASN, *Autonomous System Number*), que es único globalmente [RFC 1930]. Los números de AS, como las direcciones IP, son asignados por los registros regionales de ICANN [ICANN 2016].

Los routers de un mismo AS ejecutan todos ellos el mismo algoritmo de enrutamiento y disponen de información unos de otros. El algoritmo de enrutamiento que se ejecuta dentro de un sistema autónomo se denomina **protocolo de enrutamiento interno del sistema autónomo**.

OSPF

El enrutamiento OSPF (*Open Shortest Path First*, protocolo abierto de preferencia para la ruta más corta) y su pariente próximo, IS-IS, se utilizan ampliamente para el enrutamiento interno a los sistemas autónomos en Internet. El 'Open' de OSPF indica que la especificación del protocolo de enrutamiento está disponible públicamente (a diferencia, por ejemplo, del protocolo EIGRP de Cisco, que solo recientemente se ha convertido en abierto [Savage 2015], después de ser durante unos 20 años un protocolo propiedad de Cisco). La versión más reciente de OSPF, la versión 2, está definida en [RFC 2328], un documento público.

OSPF es un protocolo de estado de enlaces que utiliza la técnica de inundación de información de estado de los enlaces y un algoritmo de la ruta de coste mínimo de Dijkstra. Con OSPF, un router construye un mapa topológico completo (es decir, un grafo) del sistema autónomo entero. A continuación, cada router ejecuta localmente el algoritmo de la ruta más corta de Dijkstra para determinar un árbol de rutas más cortas a todas las *subredes*, con él mismo como nodo raíz. El administrador de la red configura los costes de los enlaces individuales (consulte el recuadro En la práctica: configuración de los pesos de los enlaces en OSPF). El administrador puede decidir hacer igual a 1 el coste de todos los enlaces, proporcionando así un enrutamiento con un número mínimo de saltos, o puede definir los pesos de los enlaces para que sean inversamente proporcionales a la capacidad de los mismos, con el fin de disuadir al tráfico de utilizar los enlaces con poco ancho de banda. OSPF no establece una política para definir el peso de los enlaces (esta tarea le corresponde al administrador de la red), sino que proporciona el mecanismo (el protocolo) para determinar el enrutamiento de coste mínimo para el conjunto dado de pesos de los enlaces.

EN LA PRÁCTICA

CONFIGURACIÓN DE LOS PESOS DE LOS ENLACES EN OSPF

En nuestra exposición sobre el enrutamiento de estado de enlaces hemos supuesto implícitamente que los pesos de los enlaces están definidos, que se está ejecutando un algoritmo de enrutamiento como OSPF y que el tráfico fluye de acuerdo con las tablas de enrutamiento calculadas por el algoritmo LS. En términos de causa y efecto, los pesos de los enlaces nos vienen dados (es decir, se conocen de antemano) y dan como resultado (mediante el algoritmo de Dijkstra) las rutas que minimizan el coste global. Desde este punto de vista, los pesos de los enlaces reflejan el coste de utilizar un enlace (por ejemplo, si los pesos son inversamente proporcionales a la capacidad, entonces los enlaces de alta capacidad tendrían asociados pesos más pequeños y, por tanto, serían más atractivos desde el punto de vista del enrutamiento) y el algoritmo de Disjkstra sirve para minimizar el coste global.

En la práctica, la relación causa-efecto entre el peso de los enlaces y las rutas puede invertirse, cuando los operadores de red configuran los pesos de los enlaces de manera que se obtengan rutas que permitan alcanzar determinados objetivos de ingeniería de tráfico [Fortz 2000, Fortz 2002]. Por ejemplo, suponga que un operador de red tiene una estimación del flujo de tráfico que entra en la red por cada punto de entrada y que está destinado a cada punto de salida. El operador podría querer entonces implementar un enrutamiento específico para los flujos de entrada-a-salida que minimice la tasa máxima de utilización de los enlaces de la red. Pero con un algoritmo de enrutamiento como OSPF, la principal herramienta de la que dispone el operador para alterar el enrutamiento de los flujos a través de la red son los pesos de los enlaces. Por tanto, para alcanzar el objetivo de minimizar la tasa máxima de utilización de los enlaces, el operador tiene que encontrar el conjunto de pesos de los enlaces que permita alcanzar este objetivo. Esto constituye una inversión de la relación causa-efecto: el enrutamiento deseado de los flujos es conocido y tienen que determinarse los pesos de los enlaces OSPF de manera que el algoritmo de enrutamiento OSPF dé como resultado el enrutamiento de flujos deseado.

Con OSPF, un router difunde la información de enrutamiento a *todos* los demás routers del sistema autónomo, no solo a sus routers vecinos. Un router difunde la información de estado de los enlaces cuando se produce un cambio en el estado de un enlace (por ejemplo, un cambio en el coste o en su estado activo/inactivo). También difunde periódicamente el estado de un enlace (al menos una vez cada 30 minutos), incluso aunque el estado del mismo no haya cambiado. El documento RFC 2328 destaca que "esta actualización periódica de los anuncios del estado de los enlaces añade robustez al algoritmo LS". Los anuncios OSPF están contenidos en mensajes OSPF que son transportados directamente por IP, siendo el número del protocolo de la capa superior para OSPF igual a 89. Por tanto, el protocolo OSPF tiene que implementar por sí mismo funcionalidades tales como la de transferencia fiable de mensajes y la de envío de mensajes de difusión acerca del estado de los enlaces. El protocolo OSPF también comprueba que los enlaces estén operativos (mediante un mensaje HELLO que se envía a un vecino conectado) y permite al router OSPF obtener de un vecino la base de datos de estado de los enlaces de toda la red.

Algunas de las funcionalidades avanzadas incluidas en OSPF son las siguientes:

- *Seguridad.* Los intercambios entre routers OSPF (por ejemplo, actualizaciones de estado de los enlaces) pueden ser autenticados. Con la autenticación, solo pueden participar en el protocolo OSPF los routers de confianza del sistema autónomo, impidiendo así que intrusos maliciosos (o estudiantes de redes que apliquen sus conocimientos recién adquiridos sin permiso) inyecten información incorrecta en las tablas de los routers. De manera predeterminada, los paquetes OSPF entre routers no son autenticados y podrían ser alterados. Pueden configurarse dos tipos de autenticación: simple y MD5 (consulte el Capítulo 8 para obtener información sobre MD5 y la autenticación en general). Con la autenticación simple, se configura la misma contraseña en todos los routers. Cuando un router envía un paquete OSPF, incluye la contraseña en texto plano

("legible"). Evidentemente, la autenticación simple no es muy segura. La autenticación MD5 está basada en claves secretas compartidas que están configuradas en todos los routers. Para cada paquete OSPF que se envía, el router calcula el hash MD5 del contenido del paquete OSPF, al que se añade la clave secreta (consulte el Capítulo 8 para ver una explicación acerca de los códigos de autenticación de mensajes). A continuación, el router incluye el valor hash resultante en el paquete OSPF. El router receptor, utilizando la clave secreta preconfigurada, calculará un hash MD5 del paquete y lo comparará con el valor hash que transporta el paquete, verificando de este modo la autenticidad del mismo. En la autenticación MD5 también se utilizan números de secuencia para protegerse frente a ataques por repetición.

- *Varias rutas de igual coste.* Cuando varias rutas a un destino tienen el mismo coste, OSPF permite utilizar varias rutas (es decir, no es necesario elegir una misma ruta para transportar todo el tráfico cuando existen varias rutas con igual coste).
- *Soporte integrado para enrutamiento por unidifusión y por multidifusión.* OSPF multidifusión (MOSPF, *Multicast OSPF*) [RFC 1584] añade extensiones simples a OSPF para permitir el enrutamiento por multidifusión. MOSPF utiliza la base de datos de enlaces OSPF existente y añade un nuevo tipo de anuncio de estado de enlaces al mecanismo de difusión de estado de los enlaces de OSPF.
- *Soporte para definir una jerarquía dentro de un mismo AS.* Un sistema autónomo OSPF puede configurarse jerárquicamente en áreas. Cada área ejecuta su propio algoritmo OSPF de enrutamiento por estado de enlaces, encargándose cada router de un área de difundir su información de estado de enlaces a todos los restantes routers del área. Dentro de un área, uno o más routers de frontera de área son responsables de enrutar los paquetes hacia fuera del área. Por último, una y solo una de las áreas OSPF del AS se configura como área troncal. El papel principal del área troncal es enrutar el tráfico entre las restantes áreas del AS. El área troncal es la que contiene siempre todos los routers de frontera de área del AS y puede contener también routers que no sean de frontera. El enrutamiento entre áreas dentro del AS requiere que el paquete se enrute primero hacia router de frontera de área (enrutamiento interno al área), que luego se enrute a través del área troncal hasta el router de frontera perteneciente al área de destino y que después se enrute hasta su destino final.

OSPF es un protocolo relativamente complejo, por lo que aquí lo hemos cubierto de forma necesariamente breve; en [Huitema 1998; Moy 1998; RFC 2328] se proporcionan detalles adicionales.

5.4 Enrutamiento entre los ISP: BGP

Acabamos de ver que OSPF es un ejemplo de protocolo de enrutamiento dentro de un sistema autónomo. Cuando se enruta un paquete entre un origen y un destino que se encuentran en un mismo sistema autónomo, la ruta que el paquete sigue está completamente determinada por el protocolo de enrutamiento dentro del sistema autónomo. Sin embargo, para enrutar un paquete entre múltiples sistemas autónomos (por ejemplo, desde un teléfono inteligente en Tombuctú hasta un servidor situado en un centro de datos de Silicon Valley, California), necesitamos un **protocolo de enrutamiento entre sistemas autónomos**. Puesto que un protocolo de enrutamiento entre sistemas autónomos requiere la coordinación de múltiples AS, los sistemas autónomos que se estén comunicando deberán ejecutar el mismo protocolo de enrutamiento entre sistemas autónomos. De hecho, en Internet, todos los AS utilizan el mismo protocolo de enrutamiento entre sistemas autónomos, denominado *Border Gateway Protocol*, más conocido como **BGP** [RFC 4271; Stewart 1999].

Se podría sostener que BGP es el más importante de todos los protocolos de Internet (el único otro competidor sería el protocolo IP que hemos estudiado en la Sección 4.3), ya que es el que une a los miles de ISP existentes en Internet. Como pronto veremos, BGP es un protocolo descentralizado

y asíncrono, en la línea del enrutamiento por vector de distancias descrito en la Sección 5.2.2. Aunque BGP es un protocolo complejo y difícil, si queremos comprender en profundidad Internet necesitamos familiarizarnos con sus principios fundamentales y su funcionamiento. El tiempo que dediquemos a aprender BGP estará bien invertido.

5.4.1 El papel de BGP

Para comprender las responsabilidades de BGP, pensemos en un sistema autónomo y en un router cualquiera dentro de ese sistema autónomo. Recuerde que todo router dispone de una tabla de reenvío, que juega el papel central en el proceso de reenviar los paquetes entrantes hacia los enlaces de salida del router. Como hemos visto, para los destinos que se encuentran dentro del mismo AS, las entradas de la tabla de reenvío del router están determinadas por el protocolo de enrutamiento dentro del sistema autónomo. ¿Pero qué ocurre con los destinos que se encuentran fuera del AS? Es precisamente aquí donde BGP viene al rescate.

En BGP, los paquetes no se enrutan hacia una dirección de destino específica, sino hacia prefijos CIDR, representando cada prefijo a una subred o a una colección de subredes. En el mundo de BGP, un destino puede tener la forma 138.16.68/22, que en este ejemplo incluiría 1.024 direcciones IP. Por tanto, la tabla de reenvío de un router dispondrá de entradas de la forma (*x*, *I*), donde *x* es un prefijo (como por ejemplo 138.16.68/22) e *I* es el número de interfaz de una de las interfaces del router.

Como protocolo de enrutamiento entre sistemas autónomos, BGP proporciona a cada router mecanismos para:

1. *Obtener de los sistemas autónomos vecinos información acerca de la alcanzabilidad de los prefijos.* En particular, BGP permite a cada subred anunciar su existencia al resto de Internet. Una subred vocifera "Existo y estoy aquí", y BGP garantiza que todos los routers de Internet sepan sobre ella. Si no fuera por BGP, las subredes estarían aisladas, resultando desconocidas e inalcanzables para el resto de Internet.
2. *Determinar las "mejores" rutas hacia los distintos prefijos.* Un router puede llegar a conocer dos o más rutas hacia un prefijo específico. Para determinar la mejor ruta, el router ejecutará localmente un procedimiento de selección de rutas de BGP (utilizando la información de alcanzabilidad de prefijos que ha obtenido de los routers vecinos). La mejor ruta se determinará basándose tanto en las políticas existentes, como en la información de alcanzabilidad.

Veamos ahora cómo lleva a cabo BGP estas dos tareas.

5.4.2 Anuncio de la información de rutas BGP

Considere la red mostrada en la Figura 5.8. Como se puede ver, esta sencilla red tiene tres sistemas autónomos: AS1, AS2 y AS3. En la figura se muestra que AS3 incluye una subred con prefijo x. En un sistema autónomo determinado, cada router puede ser un **router de pasarela** o un **router interno**. Un router de pasarela es aquel que está situado en la frontera de un sistema autónomo y se conecta directamente a uno o más routers de otros sistemas autónomos. Un router interno solo está conectado a hosts y routers pertenecientes a su propio sistema autónomo. En AS1, por ejemplo, el router 1c es un router de pasarela; los routers 1a, 1b y 1d son routers internos.

Pensemos en la tarea de anunciar la información de alcanzabilidad del prefijo x a todos los routers mostrados en la Figura 5.8. A alto nivel, esta tarea resulta sencilla. En primer lugar, AS3 envía un mensaje BGP a AS2, informándole de que x existe y de que se encuentra en AS3; llamemos a este mensaje "AS3 x". A continuación, AS2 envía un mensaje BGP a AS1, informándole de que x existe y de que se puede llegar a x pasando primero por AS2 y luego yendo a AS3; llamemos a este mensaje "AS2 AS3 x". De esta forma, los distintos sistemas autónomos no solo aprenderán que x existe, sino también una ruta de sistemas autónomos que conduce hasta x.

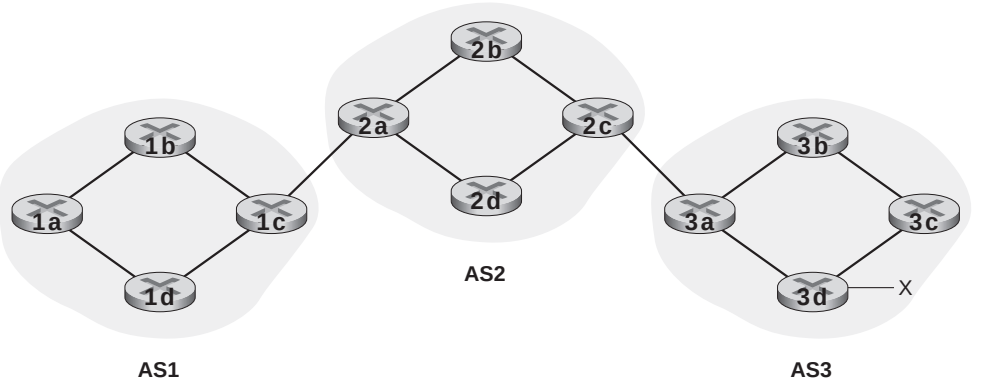


Figura 5.8 ♦ Red con tres sistemas autónomos. AS3 incluye una subred con prefijo x.

Aunque la explicación del párrafo anterior sobre el anuncio de información de alcanzabilidad BGP transmite correctamente la idea general, no es una explicación precisa, en el sentido de que no son los sistemas autónomos los que se intercambian mensajes, sino que en realidad quien se los intercambia son los routers. Para entender este punto, volvamos a examinar el ejemplo de la Figura 5.8. En BGP, las parejas de routers intercambian información de enrutamiento a través de conexiones TCP semipermanentes, utilizando el puerto 179. Cada una de esas conexiones TCP, junto con todos los mensajes BGP enviados a través de la conexión, se denomina **conexión BGP**. Además, una conexión BGP que abarca dos sistemas autónomos se llama conexión **BGP externa (eBGP)**, mientras que una sesión BGP entre routers de un mismo sistema autónomo se denomina conexión **BGP interna (iBGP)**. En la Figura 5.9 se muestran ejemplos de conexiones BGP para la red de la Figura 5.8. Normalmente habrá una conexión eBGP por cada enlace que conecte directamente routers de pasarela situados en diferentes sistemas autónomos; así, en la Figura 5.9 hay una conexión eBGP entre los routers de pasarela 1c y 2a, y otra conexión eBGP entre los routers de pasarela 2c y 3a.

Existen también conexiones iBGP entre los routers que forman cada uno de los sistemas autónomos. En particular, la Figura 5.9 muestra una configuración común, en la que hay una conexión BGP por cada pareja de routers interna a un sistema autónomo, creando una malla de conexiones

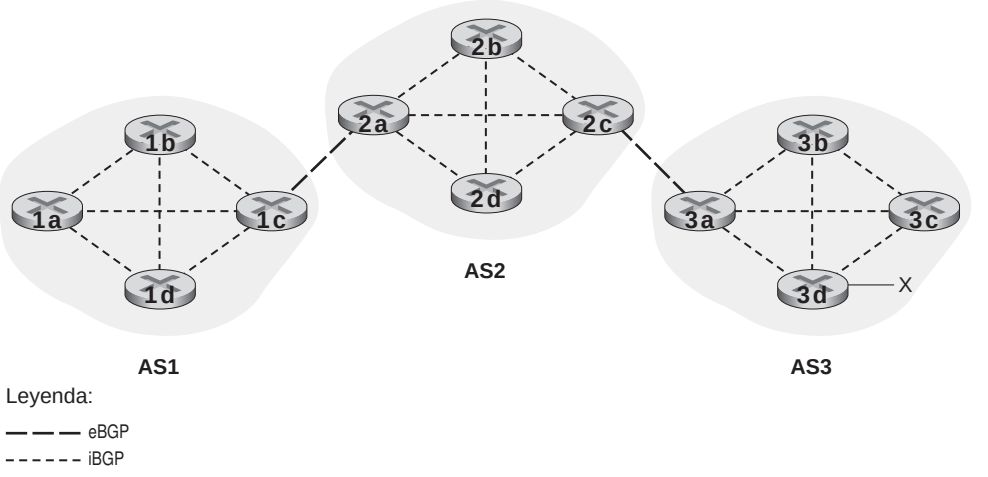


Figura 5.9 ♦ Conexiones eBGP e iBGP.

TCP dentro de cada AS. En la Figura 5.9, las conexiones eBGP se muestran con punteado más largo; las conexiones iBGP se ilustran mediante punteado más corto. Observe que las conexiones iBGP no siempre se corresponden con enlaces físicos.

Para propagar la información de alcanzabilidad se utilizan sesiones tanto iBGP como eBGP. Consideremos de nuevo la tarea de anunciar la información de alcanzabilidad para el prefijo x a todos los routers de AS1 y AS2. En este proceso, el router de pasarela 3a envía primero un mensaje eBGP "AS3 x" al router de pasarela 2c. A continuación, el router de pasarela 2c envía el mensaje iBGP "AS3 x" a todos los restantes routers de AS2, incluyendo el router de pasarela 2a. Después, el router 2a envía el mensaje eBGP "AS2 AS3 x" al router de pasarela 1c. Finalmente, el router de pasarela 1c utiliza iBGP para enviar el mensaje "AS2 AS3 x" a todos los routers de AS1. Tras completar este proceso, todos los routers de AS1 y AS2 conocen la existencia de x y conocen también una ruta de sistemas autónomos que conduce hasta x.

Por supuesto, en una red real puede haber muchas rutas diferentes desde un cierto router hasta un destino determinado, cada una de ellas a través de una secuencia distinta de sistemas autónomos. Considere, por ejemplo, la red de la Figura 5.10, que es la red original de la Figura 5.8, pero con un enlace físico adicional que une el router 1d con el router 3d. En este caso, hay dos rutas desde AS1 hasta x: la ruta "AS2 AS3 x" a través del router 1c, y la nueva ruta "AS3 x" a través del router 1d.

5.4.3 Determinación de las mejores rutas

Como acabamos de ver, puede haber muchas rutas desde un cierto router hasta una subred de destino. En Internet, de hecho, los routers reciben a menudo información de alcanzabilidad acerca de docenas de diferentes rutas posibles. ¿Cómo elige un router entre estas rutas (y luego configura correspondientemente su tabla de reenvío)?

Antes de entrar en esta cuestión crítica, necesitamos presentar algo más de terminología BGP. Cuando un router anuncia un prefijo a través de una conexión BGP, incluye con el prefijo varios **atributos BGP**. En la jerga de BGP, un prefijo junto con sus atributos se denomina **ruta**. Dos de los atributos más importantes son AS-PATH y NEXT-HOP. El atributo AS-PATH contiene la lista de sistemas autónomos a través de los que ha pasado el anuncio, como hemos visto en los ejemplos anteriores. Para generar el valor AS-PATH, cuando se pasa un prefijo a un sistema autónomo, el sistema añade su ASN a la lista contenida en AS-PATH. Por ejemplo, en la Figura 5.10 hay dos rutas desde AS1 hasta la subred x: una que utiliza el valor AS-PATH "AS2 AS3" y otra que emplea el valor AS-PATH "AS3". Los routers BGP también utilizan el atributo AS-PATH para detectar e impedir los bucles de anuncio; en concreto, si un router ve que su propio sistema autónomo está en la lista de la ruta, rechazará el anuncio.

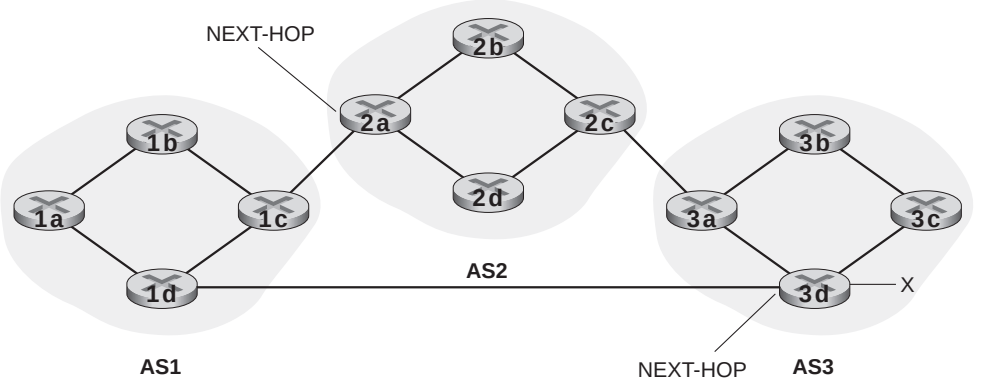


Figura 5.10 ♦ Red ampliada con un enlace directo entre AS1 y AS3.

El atributo NEXT-HOP proporciona el enlace crítico entre el protocolo de enrutamiento interno del sistema autónomo y el protocolo de enrutamiento entre sistemas autónomos, y tiene un sutil aunque importante uso. El siguiente salto (NEXT-HOP) es la *dirección IP de la interfaz de router que inicia la secuencia de sistemas autónomos* (AS-PATH). Para comprender el uso de este atributo, vamos a hacer referencia de nuevo a la Figura 5.10. Como se indica en ella, el atributo NEXT-HOP para la ruta "AS2 AS3 x", que va de AS1 hasta x y pasa por AS2, es la dirección IP de la interfaz izquierda del router 2a. El atributo NEXT-HOP para la ruta "AS3 x", que va de AS1 hasta x y que evita pasar por AS2, es la dirección IP de la interfaz izquierda del router 3d. En resumen, en este sencillo ejemplo, cada router de AS1 aprende que existen dos rutas BGP hacia el prefijo x:

Dirección IP de la interfaz izquierda del router 2a; AS2 AS3; x
Dirección IP de la interfaz izquierda del router 3d; AS3; x

Aquí, hemos escrito cada ruta BGP en forma de lista con tres componentes: NEXT-HOP; AS-PATH; prefijo de destino. En la práctica, una ruta BGP incluye atributos adicionales, que por el momento vamos a ignorar. Observe que el atributo NEXT-HOP es una dirección IP de un router que *no* pertenece a AS1; sin embargo, la subred que contiene esta dirección IP está directamente conectada a AS1.

Enrutamiento de la patata caliente

Ya estamos *finalmente* en posición de hablar acerca de los algoritmos de enrutamiento BGP de una forma precisa. Comenzaremos con uno de los algoritmos más simples, el denominado **enrutamiento de la patata caliente**.

Fíjese en el router 1b de la Figura 5.10. Como acabamos de describir, este router aprenderá que existen dos rutas BGP posibles hasta el prefijo x. En el enrutamiento de la patata caliente, la ruta elegida (de entre todas las posibles) será aquella que tenga el mínimo coste hasta el router NEXT-HOP que da comienzo a la ruta. En este ejemplo, el router 1b consultará su información de enrutamiento interna al sistema autónomo para encontrar la ruta interna de mínimo coste hasta el router NEXT-HOP 2a y la ruta interna de mínimo coste hasta el router NEXT-HOP 3d, y luego seleccionará la que tenga un menor coste de las dos. Por ejemplo, suponga que definimos el coste como el número de enlaces atravesados. Entonces el mínimo coste entre el router 1b y el router 2a será 2; el mínimo coste entre el router 1b y el router 3d será 3 y, por tanto, se seleccionaría el router 2a. A continuación, el router 1b consultaría su tabla de reenvío (configurada por su algoritmo de enrutamiento interno al sistema autónomo) y localizaría la interfaz *I* que se encuentra en la ruta de mínimo coste hacia el router 2a. Después, añadirá (*x*, *I*) a su tabla de reenvío.

Los pasos para añadir a la tabla de reenvío de un router un prefijo externo al sistema autónomo (en el caso del enrutamiento de la patata caliente) se resumen en la Figura 5.11. Es importante observar que, a la hora de añadir a una tabla de reenvío un prefijo externo al sistema autónomo, se utilizan tanto el protocolo de enrutamiento entre sistemas autónomos (BGP) como el protocolo de enrutamiento interno al sistema autónomo (por ejemplo, OSPF).

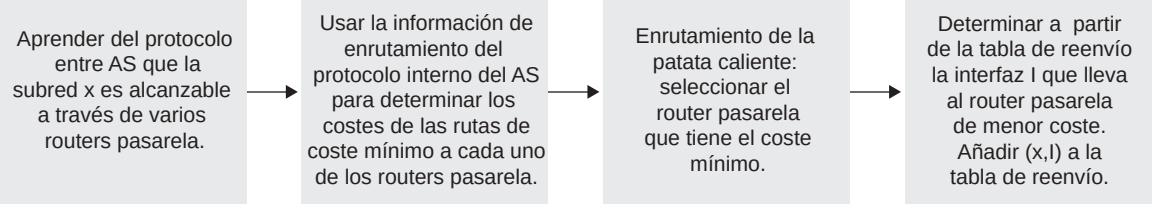


Figura 5.11 ♦ Pasos para añadir a la tabla de reenvío de un router un destino externo al sistema autónomo.

La idea que subyace al enrutamiento de la patata caliente es que el router 1b debe preocuparse de conseguir que los paquetes salgan de su propio sistema autónomo lo más rápidamente posible (o, para ser más específicos, con el menor coste posible), sin preocuparse del coste que tengan las partes restantes de la ruta hasta el destino, situadas fuera de su propio sistema autónomo. En el nombre "enrutamiento de la patata caliente" subyace la idea de que un paquete es análogo a una patata caliente, que nos está quemando las manos. Puesto que quema, queremos pasársela a otra persona (a otro sistema autónomo) lo más rápidamente posible. El enrutamiento de la patata caliente es, por tanto, un algoritmo egoísta: trata de reducir el coste dentro de su propio sistema autónomo, ignorando los restantes componentes de los costes extremo a extremo que están fuera de su sistema autónomo. Fíjese en que, con el enrutamiento de la patata caliente, dos routers pertenecientes a un mismo sistema autónomo podrían elegir dos rutas de sistemas autónomos diferentes para un mismo prefijo. Por ejemplo, acabamos de ver que el router 1b enviaría los paquetes a través de AS2 para alcanzar x. Sin embargo, el router 1d evitaría AS2 y enviaría los paquetes directamente a AS3 para alcanzar x.

Algoritmo de selección de ruta

En la práctica, BGP usa un algoritmo que es más complicado que el enrutamiento de la patata caliente, pero de todos modos incorpora también dicho enrutamiento. Para cualquier prefijo de destino especificado, la entrada al algoritmo de selección de ruta de BGP es el conjunto de todas las rutas hasta ese prefijo que han sido aprendidas y aceptadas por el router. Si solo existe una ruta, obviamente BGP seleccionará esa ruta. Si existen dos o más rutas hacia el mismo prefijo, entonces BGP invoca secuencialmente las siguientes reglas de eliminación hasta quedarse con una ruta:

1. Se asigna un valor de **preferencia local** a las rutas, como uno de sus atributos (además de los atributos AS-PATH y NEXT-HOP). La preferencia local de una ruta podría haber sido definida por el router o podría haber sido aprendida de otro router perteneciente al mismo sistema autónomo. El valor del atributo de preferencia local es una decisión política que se deja completamente en manos del administrador de red del sistema autónomo (en breve veremos en detalle las políticas de BGP). Se eligen las rutas con los valores de preferencia local más altos.
2. De las rutas que quedan (todas ellas con el mismo valor de preferencia local), se selecciona la ruta con la secuencia de sistemas autónomos (AS-PATH) más corta. Si esta regla fuera la única para seleccionar la ruta, entonces BGP estaría aplicando un algoritmo de vector de distancias para determinar la ruta, siendo la métrica de distancia utilizada el número de saltos entre sistemas autónomos, en lugar del número de saltos entre routers.
3. Para las restantes rutas (todas con el mismo valor de preferencia local y la misma longitud de AS-PATH), se utiliza el enrutamiento de la patata caliente, es decir, se selecciona la ruta con el router del siguiente salto (NEXT-HOP) más próximo.
4. Si siguiera quedando más de una ruta, el router utilizaría identificadores BGP para seleccionar la ruta; véase [Stewart 1999].

Como ejemplo, consideremos de nuevo el router 1b de la Figura 5.10. Recuerde que hay exactamente dos rutas BGP hacia el prefijo x, una que pasa a través de AS2 y otra que evita pasar a través de AS2. Recuerde también que si solo usáramos el enrutamiento de la patata caliente, entonces BGP enrutaría los paquetes hacia x a través de AS2. Pero en el anterior algoritmo de selección de ruta, la regla 2 se aplica antes de la regla 3, haciendo que BGP seleccione la ruta que evita pasar por AS2, ya que esa ruta tiene un AS-PATH más corto. Así que, como puede ver, con el anterior algoritmo de selección de ruta BGP ya no es un algoritmo egoísta: elige preferentemente las rutas con secuencias sistemas autónomos cortas (reduciendo así el retardo extremo a extremo).

Como hemos mencionado anteriormente, BGP es el estándar *de facto* para el enrutamiento entre sistemas autónomos de Internet. Para ver el contenido de varias tablas de enrutamiento BGP (¡muy grandes!) extraídas de routers de los ISP de nivel 1, consulte http://www.routeviews.org. Frecuentemente, las tablas de enrutamiento BGP contienen más de medio millón de rutas (es

decir, prefijos y sus atributos correspondientes). Puede ver estadísticas acerca del tamaño y las características de las tablas de enrutamiento BGP en [Potaroo 2016].

5.4.4 IP-Anycast

Además de ser el protocolo de enrutamiento entre sistemas autónomos de Internet, BGP se usa a menudo para implementar el servicio IP-anycast [RFC 1546, RFC 7094], utilizado comúnmente en DNS. Para entender la motivación de IP-anycast, piense que, en muchas aplicaciones, nos interesa (1) duplicar el mismo contenido en diferentes servidores, situados en muchas ubicaciones distintas, geográficamente dispersas; y (2) hacer que cada usuario acceda al contenido usando el servidor que tenga más próximo. Por ejemplo, una red CDN podría replicar los vídeos y otros objetos en servidores situados en distintos países. De forma similar, el sistema DNS puede replicar los registros DNS en servidores DNS dispersos por todo el mundo. Cuando un usuario quiera acceder a este contenido replicado, resulta conveniente dirigir al usuario al servidor "más próximo" que disponga de ese contenido. El algoritmo de selección de rutas de BGP proporciona un mecanismo sencillo y natural para hacer esto.

Para concretar nuestra exposición, veamos de qué modo podría una red CDN utilizar IP-anycast. Como se muestra en la Figura 5.12, durante la etapa de configuración de IP-anycast la empresa CDN asigna la *misma* dirección IP a cada uno de sus servidores, y utiliza BGP estándar para anunciar esa dirección IP de cada uno de los servidores. Cuando un router BGP recibe múltiples anuncios de ruta para esta dirección IP, trata esos anuncios como si proporcionaran diferentes rutas hacia una misma ubicación física (cuando, de hecho, los anuncios son para diferentes rutas hacia distintas ubicaciones físicas). Al configurar su tabla de enrutamiento, cada router utilizará localmente el algoritmo de selección de rutas de BGP para elegir la "mejor" ruta hacia esa dirección IP (por ejemplo, la ruta más cercana, midiendo según el número de saltos de

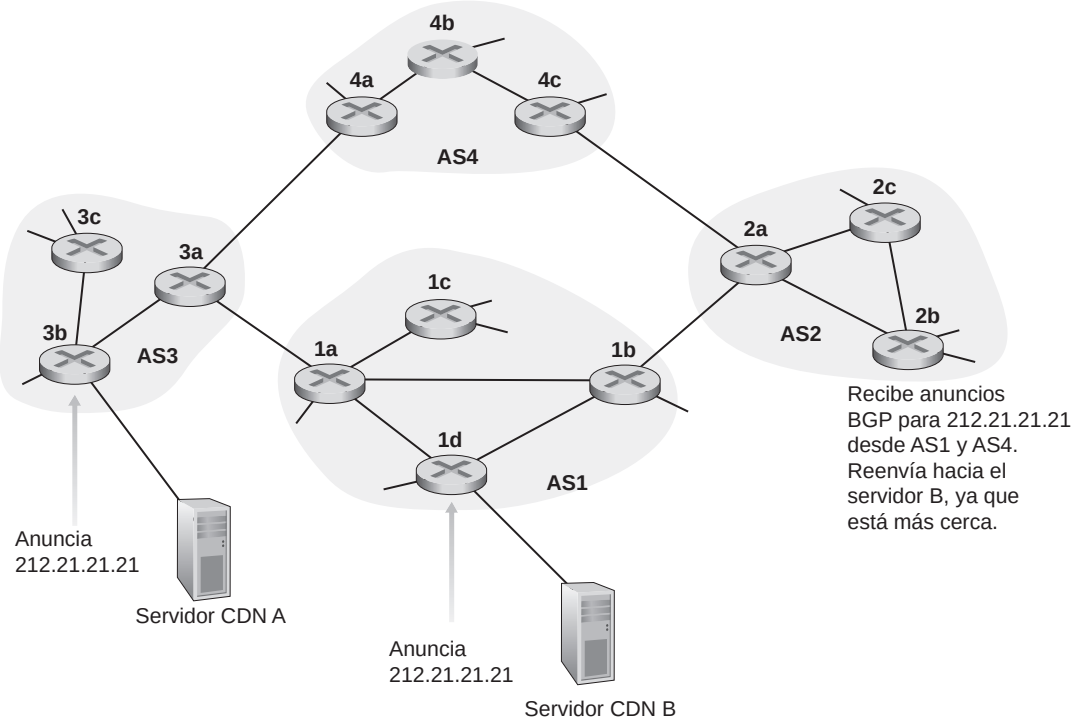


Figura 5.12 ♦ Utilización de IP-anycast para dirigir a los usuarios al servidor CDN más próximo.

sistema autónomo). Por ejemplo, si una ruta BGP (correspondiente a una ubicación) está a una distancia de un solo salto de sistema autónomo con respecto al router y todas las restantes rutas BGP (correspondientes a otras ubicaciones) están a una distancia de dos o más saltos de sistema autónomo, entonces el router BGP decidirá enrutar los paquetes hacia la ubicación que está a un solo salto de distancia. Después de esta fase inicial BGP de anuncio de direcciones, la red CDN puede dedicarse a su tarea principal: distribuir contenido. Cuando un cliente solicita el vídeo, la CDN devuelve al cliente la dirección IP común utilizada por los servidores geográficamente dispersos, con independencia de dónde esté situado el cliente. Cuando el cliente envía una solicitud a esa dirección IP, los routers de Internet reenvían ese paquete de solicitud al servidor "más próximo", según determine el algoritmo de selección de rutas de BGP.

Aunque el ejemplo anterior de CDN ilustra adecuadamente cómo puede usarse IP-anycast, en la práctica las redes CDN no suelen utilizar IP-anycast, porque los cambios en el enrutamiento BGP pueden hacer que diferentes paquetes de una misma conexión TCP lleguen a diferentes instancias del servidor web. Pero IP-anycast sí que es ampliamente utilizado por el sistema DNS para dirigir las consultas DNS al servidor DNS raíz más cercano. Recuerde, de la Sección 2.4, que actualmente existen 13 direcciones IP para los servidores DNS raíz. Pero para cada una de estas direcciones hay múltiples servidores DNS raíz, teniendo algunas de estas direcciones más de 100 servidores DNS raíz dispersos por todo el mundo. Cuando se envía una consulta DNS a una de estas 13 direcciones IP, se utiliza IP-anycast para enrutar la consulta hacia el servidor DNS raíz más cercano que sea responsable de esa dirección.

5.4.5 Política de enrutamiento

Cuando un router selecciona una ruta hacia un destino, la política de enrutamiento del sistema autónomo puede prevalecer sobre todas las restantes consideraciones, como la secuencia más corta de sistemas autónomos o el enrutamiento de la patata caliente. De hecho, en el algoritmo de selección de ruta, las rutas se seleccionan primero de acuerdo con el atributo de preferencia local, cuyo valor está fijado por la política del sistema autónomo local.

Vamos a ilustrar algunos de los conceptos básicos de la política de enrutamiento de BGP con un ejemplo sencillo. La Figura 5.13 muestra seis sistemas autónomos interconectados: A, B, C, W, X e Y. Es importante observar que A, B, C, W, X e Y son sistemas autónomos, no routers. Supongamos que los sistemas autónomos W, X e Y son ISP de acceso, y que A, B y C son redes troncales de empresas operadoras. Supongamos también que A, B y C se intercambian tráfico directamente y que proporcionan información BGP completa a sus redes clientes. Todo el tráfico que entra en la red de un ISP de acceso tiene que estar destinado a esa red, y todo el tráfico que sale de la red de un ISP de acceso tiene que haber sido originado en esa red. W e Y son claramente proveedores ISP de acceso. X es un **ISP de acceso multidomiciliado**, ya que está conectado al resto de la red a través de dos proveedores diferentes (un escenario cada vez más común en la práctica). Sin embargo, al igual que sucede con W e Y, X tiene que ser el origen/destino de todo el tráfico que sale/entra en X. Pero, ¿cómo puede implementarse este comportamiento de la red de acceso? ¿Cómo impedir que X reenvíe tráfico entre B y C? Esto se puede conseguir fácilmente controlando la forma en que

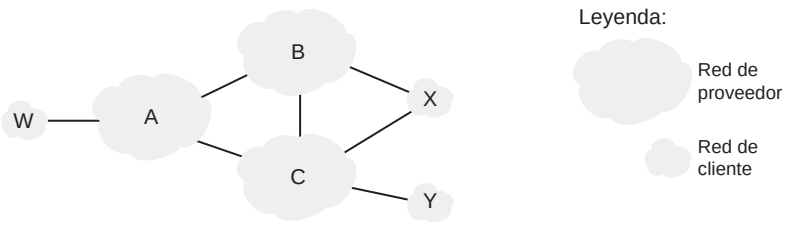


Figura 5.13 ♦ Un escenario simple de política BGP.

EN LA PRÁCTICA

¿POR QUÉ LOS SISTEMAS AUTÓNOMOS DISPONEN DE DIFERENTES PROTOCOLOS DE ENRUTAMIENTO PARA USO INTERNO Y PARA COMUNICARSE ENTRE ELLOS?

Una vez estudiados los detalles de una serie de protocolos específicos de enrutamiento interno a los sistemas autónomos y de enrutamiento entre sistemas autónomos, implantados actualmente en Internet, vamos a concluir considerando quizá la cuestión más importante que podríamos plantearnos acerca de estos protocolos (¡esperamos que le haya surgido esta duda a lo largo del capítulo y que los árboles no le hayan impedido ver el bosque!): ¿por qué se utilizan protocolos de enrutamiento distintos dentro de los sistemas autónomos y para comunicarse entre unos sistemas autónomos y otros?

La respuesta a esta pregunta se encuentra en las diferencias de objetivos entre el enrutamiento dentro de un sistema autónomo y el enrutamiento entre sistemas autónomos:

- *Política.* Entre sistemas autónomos, prevalecen las políticas. Puede ser importante que el tráfico originado en un determinado sistema autónomo no pueda atravesar otro sistema autónomo específico. De forma similar, un sistema autónomo dado puede desear controlar el tráfico de tránsito que transporta entre otros sistemas autónomos. Hemos visto que BGP transporta atributos de ruta y permite la distribución controlada de información de enrutamiento, de manera que pueden tomarse ese tipo decisiones de enrutamiento basadas en políticas. Dentro de un sistema autónomo, todo está en teoría bajo el mismo control administrativo y, por tanto, las políticas desempeñan un papel mucho menos importante en la elección de rutas dentro del sistema autónomo.
- *Escala.* La capacidad de ampliación de un algoritmo de enrutamiento y de sus estructuras de datos, con el fin de gestionar el enrutamiento hacia/entre una gran cantidad de redes, es un problema crítico en el enrutamiento entre sistemas autónomos. Dentro de un sistema autónomo, la escalabilidad es un problema menor, aunque sólo sea porque si un ISP se hace demasiado grande, siempre es posible dividirlo en dos sistemas autónomos y realizar el enrutamiento entre los dos nuevos sistemas. (Recuerde que OSPF permite crear una jerarquía de ese estilo, dividiendo un sistema autónomo en áreas.)
- *Rendimiento.* Puesto que el enrutamiento entre sistemas autónomos está muy orientado a las políticas, la calidad (por ejemplo, el rendimiento) de las rutas utilizadas suele ser una cuestión secundaria (es decir, una ruta más larga o más costosa que satisfaga determinados criterios políticos puede perfectamente tener preferencia sobre una ruta más corta, pero que no cumpla dichos criterios). De hecho, hemos visto que entre sistemas autónomos no existe ni siquiera el concepto de coste asociado con las rutas (salvo por el recuento de saltos entre sistemas autónomos). Sin embargo, dentro de un sistema autónomo tales cuestiones políticas tienen una importancia menor, lo que permite al enrutamiento centrarse más en el rendimiento que se puede alcanzar en una ruta.

son anunciadas las rutas BGP. En concreto, X operará como la red de un ISP de acceso si anuncia (a sus vecinos B y C) que no tiene ninguna ruta a ningún otro destino excepto a ella misma. Es decir, incluso aunque X pueda conocer una determinada ruta, por ejemplo XCY, para llegar a la red Y, no anunciará este camino a B. Puesto que B no es consciente de que X dispone de un camino hasta Y, B nunca reenviará tráfico destinado a Y (o a C) a través de X. Este sencillo ejemplo ilustra cómo se puede utilizar una política selectiva de anuncio de rutas para implementar las relaciones de enrutamiento cliente/proveedor.

A continuación, vamos a centrarnos en la red de un proveedor, por ejemplo, en el sistema autónomo B. Suponga que B ha aprendido (de A) que A dispone de la ruta AW hacia W. B puede, por tanto, incluir la ruta AW en su base de información de enrutamiento. Evidentemente, B también quiere anunciar la ruta BAW a su cliente, X, para que X sepa que puede llegar a W a través de B. Pero, ¿debería B anunciar la ruta BAW a C? Si lo hace, entonces C podría enviar tráfico a W por la ruta BAW. Si A, B y C son proveedores troncales, entonces B podría pensar, con razón, que no

tendría que asumir la carga (¡y el coste!) de transportar el tráfico en tránsito entre A y C. B podría pensar, con razón, que es el trabajo (¡y el coste!) de A y C asegurarse de que C puede enrutar hacia/desde los clientes de A a través de una conexión directa entre A y C. Actualmente, no existe ningún estándar oficial que gobierne cómo los ISP troncales deben llevar a cabo el enrutamiento entre ellos. Sin embargo, una regla heurística seguida por los ISP comerciales es que cualquier tráfico que fluya a través de la red troncal de un ISP tiene que tener su origen o su destino (o ambos) en una red que sea un cliente de dicho ISP; si fuera de otro modo, ese tráfico estaría obteniendo un viaje gratis a través de la red del ISP. Los acuerdos bilaterales individuales (que regulan cuestiones como las mencionadas más arriba) normalmente son negociados entre parejas de proveedores ISP y suelen ser confidenciales; [Huston 1999a] proporciona una interesante explicación acerca de esos acuerdos bilaterales. Si desea ver una descripción detallada de cómo las políticas de enrutamiento reflejan las relaciones comerciales entre los ISP, consulte [Gao 2001; Dmitiropoulos 2007]. Para ver una descripción de las políticas de enrutamiento BGP desde el punto de vista de un ISP, consulte [Caesar 2005b].

Con esto completamos nuestra breve introducción a BGP. Es importante comprender BGP porque juega un papel crucial en Internet. Le animamos a consultar las referencias [Griffin 2012; Stewart 1999; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007] para aprender más acerca de BGP.

5.4.6 Cómo encajan las piezas: obtención de presencia en Internet

Aunque esta subsección no trata acerca de BGP *per se*, engarza muchos de los protocolos y conceptos que hemos visto hasta ahora, incluyendo el direccionamiento IP, DNS y BGP.

Suponga que acaba de crear una pequeña empresa que dispone de una serie de servidores, incluyendo un servidor web público que describe los productos y servicios de su empresa, un servidor de correo desde el que sus empleados obtienen sus mensajes de correo electrónico y un servidor DNS. Naturalmente, lo que querrá es que todo el mundo pueda visitar su sitio web, para conocer sus excitantes productos y servicios. Además, querrá que sus empleados sean capaces de enviar y recibir correos electrónicos de clientes potenciales repartidos por todo el mundo.

Para conseguir estos objetivos, primero necesitará obtener conectividad Internet, lo cual se consigue firmando un contrato con un ISP local y conectándose a él. Su empresa dispondrá de un router de pasarela, que estará conectado a un router de su ISP local. Esta conexión podría tratarse de una conexión DSL a través de la infraestructura telefónica existente, de una línea arrendada que conecte con el router del ISP o de cualquiera de las otras muchas soluciones de acceso descritas en el Capítulo 1. Su ISP local también le proporcionará un rango de direcciones IP, como por ejemplo un rango de direcciones /24 compuesto por 256 direcciones. Una vez que disponga de la conectividad física y del rango de direcciones IP, tendrá que asignar una de las direcciones IP de su rango a su servidor web, otra a su servidor de correo, otra a su servidor DNS, otra a su router de pasarela y otras direcciones IP a otros servidores y dispositivos de la red de su empresa.

Además de firmar un contrato con su ISP, también tendrá que pagar a una entidad de registro de Internet para obtener un nombre de dominio para su empresa, como se describe en el Capítulo 2. Por ejemplo, si el nombre de su empresa es Xanadu Inc., lógicamente intentaría adquirir el nombre de dominio xanadu.com. Su empresa deberá obtener también presencia en el sistema DNS. Específicamente, como los usuarios externos querrán contactar con su servidor DNS para obtener las direcciones IP de sus servidores, también tendrá que proporcionar a su entidad de registro la dirección IP de su servidor DNS. Su entidad de registro añadirá entonces una entrada para su servidor DNS (nombre de dominio y dirección IP correspondiente) en los servidores del dominio de nivel superior .com, como se describe en el Capítulo 2. Una vez completado este paso, cualquier usuario que conozca su nombre de dominio (por ejemplo, xanadu.com), será capaz de obtener la dirección IP de su servidor DNS a través del sistema DNS.

Para que la gente pueda descubrir las direcciones IP de su servidor web, tendrá que incluir entradas en su servidor DNS que establezcan la correspondencia entre el nombre de host de su servidor web (por ejemplo, xanadu.com) y su dirección IP. También deberá tener entradas similares para otros servidores de su empresa que estén públicamente disponibles, incluyendo su servidor de correo. Con todo esto, si Alicia quiere acceder a su servidor web, el sistema DNS contactará con su servidor DNS, averiguará la dirección IP de su servidor web y se la proporcionará a Alicia. Alicia podrá entonces establecer una conexión TCP directamente con su servidor web.

Sin embargo, todavía queda otro paso necesario y crucial para permitir que usuarios de todo el mundo accedan a su servidor web. Piense en lo que sucedería cuando Alicia, que conoce la dirección IP de su servidor web, envíe un datagrama IP (por ejemplo, un segmento SYN TCP) a esa dirección IP. Ese datagrama será enrutado a través de Internet, visitando una serie de routers en muchos sistemas autónomos distintos, y terminará por llegar a su servidor web. Cuando cualquiera de los routers reciba el datagrama, buscará en su tabla de reenvío una entrada que le permita determinar a través de qué puerto de salida debe reenviar el datagrama. Por tanto, cada uno de los routers necesita conocer la existencia del prefijo /24 de su empresa (o de alguna entrada agregada que lo incluya). ¿Cómo puede un router conocer el prefijo de su empresa? ¡Como acabamos de ver, lo consigue gracias a BGP! Específicamente, cuando su empresa firma un contrato con un ISP local y se le asigna un prefijo (es decir, un rango de direcciones), su ISP local usará BGP para anunciar su prefijo a los ISP con los que esté conectado. Esos ISP usarán BGP, a su vez, para propagar el anuncio. Al final, todos los routers de Internet conocerán su prefijo (o algún agregado que lo incluya) y serán así capaces de reenviar apropiadamente los datagramas destinados a su servidor web y a su servidor de correo.

5.5 El plano de control SDN

En esta sección, vamos a sumergirnos en el plano de control SDN: la lógica de la red que controla el reenvío de paquetes entre los dispositivos compatibles con SDN de una red, así como la configuración y gestión de estos dispositivos y sus servicios. Nuestro estudio aquí parte del análisis del reenvío SDN generalizado que hemos esbozado anteriormente en la Sección 4.4, así que quizá convenga que revise esa sección, al igual que la Sección 5.1 de este capítulo, antes de continuar. Como en la Sección 4.4, adoptaremos aquí la terminología usada en la literatura sobre SDN y denominaremos a los dispositivos de reenvío de la red "conmutadores de paquetes" (o simplemente conmutadores, sobreentendiéndose lo de "paquetes"), ya que las decisiones de reenvío pueden tomarse basándose en las direcciones de origen/destino de la capa de red, en las direcciones de origen/destino de la capa de enlace o en muchos otros valores de los campos de las cabeceras de transporte, de red y de enlace del paquete.

Podemos identificar cuatro características fundamentales de una arquitectura SDN [Kreutz 2015]:

- *Reenvío basado en el flujo*. El reenvío de paquetes por parte de conmutadores controlados mediante SDN puede estar basado en los valores de diversos campos de las cabeceras de las capas de transporte, de red o de enlace. Ya vimos en la Sección 4.4 que la abstracción OpenFlow 1.0 permite el reenvío basado en once valores diferentes de campos de cabecera. Esto contrasta fuertemente con el enfoque tradicional de reenvío utilizado por los routers y que estudiamos en la secciones 5.2-5.4, en el que el reenvío de datagramas IP se basaba exclusivamente en la dirección IP de destino del datagrama. Recuerde de la Figura 5.2 que las reglas de reenvío de paquetes están especificadas en la tabla de flujo de un conmutador; es responsabilidad del plano de control SDN calcular, gestionar e instalar las entradas de las tablas de flujo en todos los conmutadores de la red.

- *Separación del plano de datos y el plano de control.* Esta separación se muestra claramente en las Figuras 5.2 y 5.14. El plano de datos está compuesto por los conmutadores de la red: dispositivos relativamente simples (pero rápidos) que ejecutan las reglas de "correspondencia más acción" contenidas en sus tablas de flujo. El plano de control está compuesto por servidores y software que determinan y gestionan las tablas de flujo de los conmutadores.
- *Funciones de control de la red: externas a los conmutadores del plano de datos.* Dado que la "S" de SDN significa "software", quizá no resulte sorprendente que el plano de control SDN esté implementado en software. A diferencia de los routers tradicionales, sin embargo, este software se ejecuta en servidores que son diferentes (y remotos) de los conmutadores de la red. Como se muestra en la Figura 5.14, el propio plano de control consta de dos componentes: un controlador SDN (o sistema operativo de red [Gude 2008]) y un conjunto de aplicaciones de control de red. El controlador mantiene información precisa sobre el estado de la red (por ejemplo, el estado de los hosts, conmutadores y enlaces remotos), proporciona esta información a las aplicaciones de control de red que se ejecutan en el plano de control y proporciona los medios con los cuales estas aplicaciones pueden monitorizar, programar y controlar los dispositivos de la red subyacente. Aunque el controlador de la Figura 5.14 se muestra como un único servidor central, en la práctica el controlador solo está centralizado desde el punto de vista lógico; normalmente se implementa mediante varios servidores, que proporcionan unas prestaciones coordinadas y escalables, así como una alta disponibilidad.
- *Una red programable.* La red se puede programar mediante las aplicaciones de control de red que se ejecutan en el plano de control. Estas aplicaciones representan el "cerebro" del plano de control SDN y utilizan las API proporcionadas por el controlador SDN con el fin de especificar y controlar el plano de datos en los dispositivos de la red. Por ejemplo, una aplicación de control de red para enrutamiento podría determinar las rutas extremo a extremo entre orígenes y destinos (por ejemplo, ejecutando el algoritmo de Dijkstra a partir de la información de estado

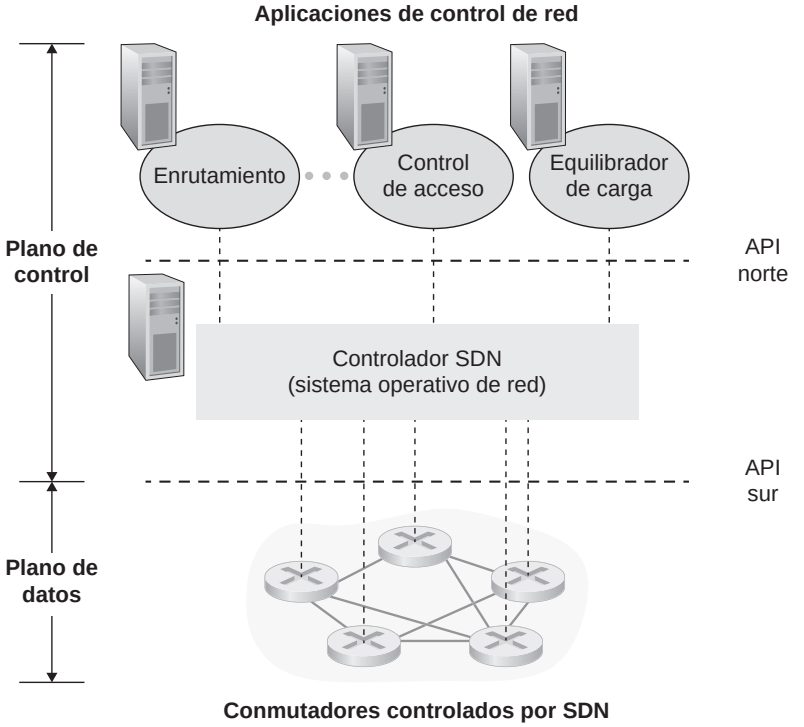


Figura 5.14 ♦ Componentes de la arquitectura SDN: conmutadores controlados por SDN, el controlador SDN y las aplicaciones de control de red.

de los nodos y de los enlaces mantenida por el controlador SDN). Otra aplicación de red podría encargarse del control de acceso, es decir, de determinar qué paquetes hay que bloquear en un conmutador, como en nuestro tercer ejemplo de la Sección 4.4.3. Otra aplicación diferente podría reenviar los paquetes de forma que se consiga un equilibrio de carga entres servidores (el segundo ejemplo que hemos considerado en la Sección 4.4.3).

Teniendo en cuenta todo esto, podemos ver que SDN representa un significativo "desempaquetamiento" de la funcionalidad de red: los conmutadores del plano de datos, los controladores SDN y las aplicaciones de control de red son entidades separadas, que pueden ser suministradas por diferentes proveedores y organizaciones. Esto contrasta con el modelo pre-SDN, en el que un router/switch (junto con su software integrado del plano de control y sus implementaciones de protocolos) era monolítico, estaba integrado verticalmente y era suministrado por un único fabricante. Este desempaquetamiento de la funcionalidad de red en SDN ha sido comparado con la evolución, en su día, desde las computadoras mainframe (en las que el hardware, el software del sistema y las aplicaciones eran suministrados por un único fabricante) a las computadoras personales (con su hardware, sistemas operativos y aplicaciones separados). El desempaquetamiento del hardware de la computadora, del software del sistema y de las aplicaciones ha dado como resultado un ecosistema abierto y rico, guiado por la innovación en esas tres áreas; la esperanza para SDN es que también traiga consigo una rica y similar ola de innovación.

Dada nuestra comprensión de la arquitectura SDN de la Figura 5.14, surgen de manera natural numerosas cuestiones. ¿Cómo y dónde se calculan en la práctica las tablas de flujo? ¿Cómo se actualizan en los dispositivos controlados por SDN esas tablas, en respuesta a sucesos (por ejemplo, un enlace conectado que se active/desactive)? ¿Y cómo se coordinan las entradas de las tablas de flujo de múltiples conmutadores, de tal manera que resulte una funcionalidad de la red orquestada y coherente (por ejemplo, rutas extremo a extremo de reenvío de paquetes desde los orígenes hasta los destinos, o cortafuegos distribuidos coordinados)? Es tarea del plano de control SDN proporcionar estas, y otras muchas, capacidades.

5.5.1 El plano de control SDN: controlador SDN y aplicaciones SDN de control de red

Comencemos nuestra exposición sobre el plano de control SDN de una forma abstracta, analizando las capacidades genéricas que el plano de control debe proporcionar. Como veremos, este enfoque abstracto, sobre los "principios fundamentales", nos llevará a una arquitectura global que refleja el modo en que se han implementado en la práctica los planos de control.

Como hemos dicho antes, el plano de control SDN está dividido, a grandes rasgos, en dos componentes: el controlador SDN y las aplicaciones SDN de control de red. Exploremos en primer lugar el controlador. Se han desarrollado muchos controladores SDN desde que surgiera el primero de ellos [Gude 2008]; en [Kreutz 2015] podrá encontrar un resumen exhaustivo y actualizado. La Figura 5.15 proporciona una vista más detallada de un controlador SDN genérico. La funcionalidad de un controlador puede organizarse, desde un punto de vista general, en tres capas. Veamos cuáles son estas capas en el orden tradicional abajo-arriba:

- *Una capa de comunicaciones: comunicación entre el controlador SDN y los dispositivos de red controlados.* Obviamente, si un controlador SDN tiene que controlar el funcionamiento de un conmutador, host u otro dispositivo remoto compatible con SDN, se necesita un protocolo para transferir información entre el controlador y ese dispositivo. Además, un dispositivo debe ser capaz de comunicar al controlador sucesos observados localmente (por ejemplo, un mensaje que indique que un enlace conectado se ha activado o desactivado o que un dispositivo se acaba de unir a la red, o un latido que indique que un dispositivo está activo y funcionando). Estos sucesos proporcionan al controlador SDN una visión actualizada del estado de la red. Este protocolo

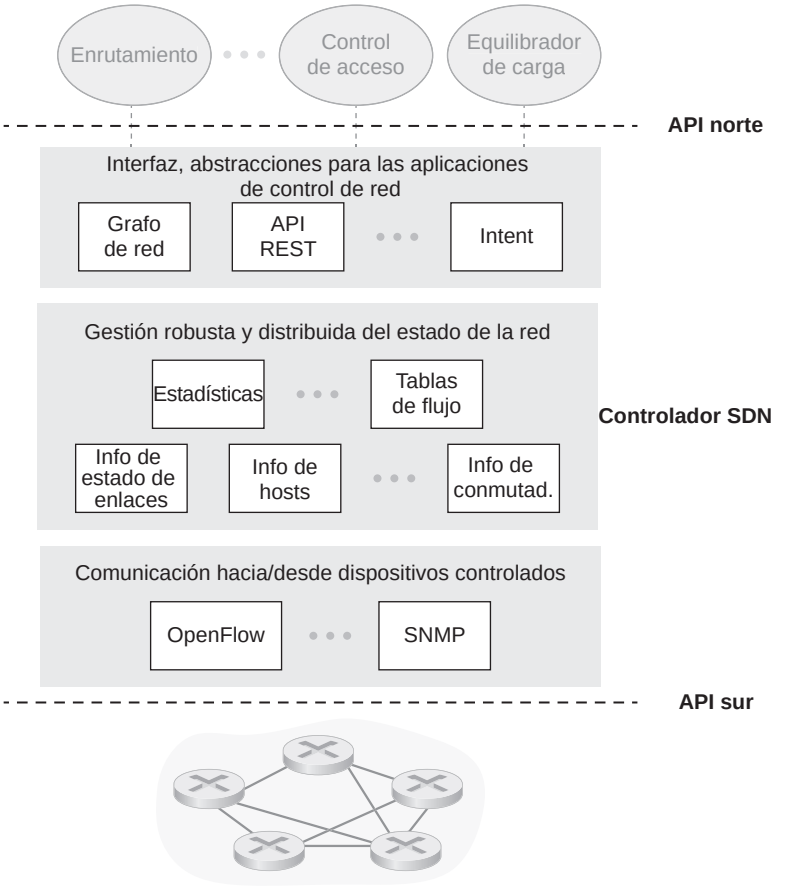


Figura 5.15 ♦ Componentes de un controlador SDN.

constituye la capa inferior de la arquitectura del controlador, como se muestra en la Figura 5.15. La comunicación entre el controlador y los dispositivos controlados cruza lo que se ha dado en llamar la interfaz "sur" del controlador (*southbound interface*). En la Sección 5.5.2 estudiaremos OpenFlow, un protocolo específico que proporciona esta funcionalidad de comunicaciones. OpenFlow está implementado en la mayoría de los controladores SDN, si no en todos.

- *Una capa de gestión del estado de la red.* Las decisiones últimas de control tomadas por el plano de control SDN —por ejemplo, configurar tablas de flujo en todos los conmutadores para conseguir el reenvío extremo a extremo deseado, para implementar un equilibrado de carga o para habilitar una determinada capacidad de cortafuegos —requiere que el controlador disponga de información actualizada sobre el estado de los hosts, enlaces, conmutadores y otros dispositivos controlados por SDN que haya en la red. La tabla de flujo de un conmutador contiene contadores cuyos valores también pueden ser aprovechados por las aplicaciones de control de red; por tanto, estos valores deberán estar disponibles para esas aplicaciones. Puesto que el fin último del plano de control es determinar las tablas de flujo de los diversos dispositivos controlados, un controlador puede también mantener una copia de dichas tablas. Todos estos elementos de información constituyen ejemplos del "estado" de la red que el controlador SDN mantiene.
- *La interfaz con la capa de aplicaciones de control de red.* El controlador interactúa con las aplicaciones de control de red a través de su interfaz "norte" (*northbound interface*). Esta API permite a las aplicaciones de control de red leer/escribir el estado de la red y las tablas de flujo

dentro de la capa de gestión del estado. Las aplicaciones pueden registrarse para que se las notifique cuando se produzcan sucesos de cambio de estado, con el fin de poder tomar acciones en respuesta a las notificaciones de sucesos de red enviadas desde los dispositivos controlados por SDN. Pueden proporcionarse diferentes tipos de APIs; como veremos más adelante, dos controladores SDN muy populares se comunican con sus aplicaciones utilizando una interfaz solicitud-respuesta tipo REST [Fielding 2000].

Ya hemos indicado, en diversas ocasiones, que se puede considerar que un controlador SDN está "lógicamente centralizado", es decir, que el controlador puede ser visto externamente (por ejemplo, desde el punto de vista de los dispositivos controlados por SDN y de las aplicaciones externas de control de red) como un único servicio monolítico. Sin embargo, estos servicios, y las bases de datos utilizadas para almacenar la información de estado, se implementan en la práctica mediante un conjunto *distribuido* de servidores, por razones de tolerancia ante fallos, de alta disponibilidad o de prestaciones. Al estar las funciones controladoras implementadas mediante un conjunto de servidores, hay que tomar en consideración la semántica de las operaciones internas del controlador (por ejemplo, el mantenimiento de un orden temporal lógico de los sucesos, la coherencia, el consenso y otros factores) [Panda 2013]. Esas preocupaciones son comunes en muchos sistemas distribuidos; en [Lamport 1989, Lampson 1996] puede encontrar algunas soluciones elegantes a estos desafíos. Los controladores modernos como OpenDaylight [OpenDaylight Lithium 2016] y ONOS [ONOS 2016] (véase el recuadro adjunto de En la práctica) han puesto un considerable énfasis en desarrollar la arquitectura de una plataforma controladora lógicamente centralizada, pero físicamente distribuida, que proporcione servicios escalables y alta disponibilidad, tanto a los dispositivos controlados como a las aplicaciones de control de red.

La arquitectura mostrada en la Figura 5.15 se asemeja mucho a la del controlador NOX originalmente propuesto en 2008 [Gude 2008], así como a la de los actuales controladores SDN OpenDaylight [OpenDaylight Lithium 2016] y ONOS [ONOS 2016] (véase el recuadro adjunto de En la práctica). Veremos un ejemplo de operación del controlador en la Sección 5.5.3. Pero primero vamos a examinar el protocolo OpenFlow, que reside en la capa de comunicaciones del controlador.

5.5.2 Protocolo OpenFlow

El protocolo OpenFlow [OpenFlow 2009, ONF 2016] opera entre un controlador SDN y un conmutador controlado por SDN u otro dispositivo que implemente la API OpenFlow que hemos estudiado anteriormente, en la Sección 4.4. El protocolo OpenFlow funciona sobre TCP, siendo su número de puerto predeterminado el 6653.

Entre los mensajes de mayor importancia que fluyen desde el controlador hacia el conmutador controlado podemos citar:

- *Configuración (configuration).* Este mensaje permite al controlador consultar y configurar los parámetros de configuración del conmutador.
- *Modificar estado (modify-state).* El controlador utiliza este mensaje para añadir/borrar o modificar entradas de la tabla de flujo del conmutador, así como para configurar las propiedades de los puertos del conmutador.
- *Leer estado (read-state).* El controlador usa este mensaje para recopilar estadísticas y valores de contadores de los puertos y tablas de flujo del conmutador.
- *Enviar paquete (send-packet).* El controlador usa este mensaje para enviar un paquete específico a través de un puerto de salida especificado del conmutador controlado. El propio mensaje contiene, en su carga útil, el paquete que hay que enviar.

Entre los mensajes que fluyen desde el conmutador controlado por SDN hacia el controlador podemos citar:

EN LA PRÁCTICA

RED GLOBAL DEFINIDA POR SOFTWARE DE GOOGLE

Recuerde del caso de estudio de la Sección 2.6 que Google tiene implantada una red de área extensa (WAN) dedicada, que interconecta sus centros de datos y sus clusters de servidores (en IXPs e ISPs). Esta red, denominada B4, tiene un plano de control SDN diseñado por Google y construido sobre OpenFlow. La red de Google es capaz de conseguir una utilización promedio de sus enlaces WAN del 70% (entre dos y tres veces superior a la tasa de utilización típica de los enlaces) y de dividir los flujos de aplicación entre múltiples rutas, basándose en la prioridad de las aplicaciones y en las demandas de flujo existentes [Jain 2013].

La red B4 de Google está particularmente bien adaptada a SDN: (*i*) Google controla todos los dispositivos, desde los servidores de frontera situados en IXPs e ISPs, hasta los routers que forman el núcleo de su red; (*ii*) las aplicaciones que más ancho de banda requieren son copias de datos a gran escala entre sitios, que pueden ceder la preferencia a las aplicaciones interactivas de mayor prioridad durante los periodos de congestión de recursos; (*iii*) al haber solo unas pocas docenas de centros de datos conectados, el control centralizado resulta factible.

La red B4 de Google utiliza conmutadores hechos a medida, cada uno de los cuales implementa una versión ligeramente ampliada de OpenFlow, con un agente local OpenFlow (OFA, *OpenFlow Agent*) que es similar en espíritu al agente de control que nos encontramos en la Figura 5.2. Cada OFA se conecta, a su vez, con un controlador OpenFlow (OFC, *OpenFlow Controller*) que reside en el servidor de control de la red (NCS, *Network Control Server*) a través de una red separada "fuera de banda", distinta de la red que transporta el tráfico entre centros de datos. El OFC proporciona, por tanto, los servicios utilizados por el NCS para comunicarse con los conmutadores que controla, de forma similar en espíritu a la capa inferior de la arquitectura SDN mostrada en la Figura 5.15. En B4, el OFC también realiza funciones de gestión de estado, manteniendo el estado de los nodos y enlaces en una base de información de red (NIB, *Network Information Base*). La implementación del OFC realizada por Google está basada en el controlador SDN ONIX [Koponen 2010]. Están implementados dos protocolos de enrutamiento: BGP para el enrutamiento entre centros de datos e IS-IS (un pariente cercano de OSPF) para el enrutamiento en el interior de un centro de datos. Se utiliza Paxos [Chandra 2007] para ejecutar réplicas en caliente de los componentes del NCS, con el fin de protegerse frente a posibles fallos.

Una aplicación de control de red especializada en ingeniería de tráfico (que descansa desde un punto de vista lógico sobre el conjunto de servidores de red) interactúa con estos servidores para proporcionar a los grupos de flujos de aplicación asignaciones globales de ancho de banda, que afectan a toda la red. Gracias a B4, SDN dio un importante salto adelante, adentrándose en las redes operacionales de un proveedor de red global. En [Jain 2013] puede encontrar una descripción detallada de B4.

- *Flujo eliminado (flow-removed).* Este mensaje informa al controlador de que se ha eliminado una entrada de una tabla de flujo, por ejemplo debido a un fin de temporización o como resultado de un mensaje *modify-state* recibido.
- *Estado de puerto (port-status).* El conmutador usa este mensaje para informar al controlador de un cambio en el estado de un puerto.
- *Paquete entrante (packet-in).* Recuerde de la Sección 4.4 que los paquetes que llegan a un puerto del conmutador y que no se corresponden con ninguna entrada de la tabla de flujo se envían al controlador para su procesamiento adicional. Los paquetes para los que sí exista correspondencia también pueden ser enviados al controlador, como una de las posibles acciones que se ejecutan al detectarse la correspondencia. El mensaje *packet-in* se emplea para enviar tales paquetes al controlador.

En [OpenFlow 2009, ONF 2016] se definen mensajes OpenFlow adicionales.

5.5.3 Interacción entre los planos de datos y de control: ejemplo

Para cimentar nuestra comprensión de la interacción entre los dispositivos controlados por SDN y el controlador SDN, consideremos el ejemplo mostrado en la Figura 5.16, en el que se utiliza el algoritmo de Dijkstra (que hemos estudiado en la Sección 5.2) para determinar las rutas más cortas. El escenario SDN de la Figura 5.16 tiene dos diferencias importantes con respecto al escenario anterior de control por router de las secciones 5.2.1 y 5.3, en donde el algoritmo de Dijkstra estaba implementado en todos y cada uno de los routers y se usaba la técnica de inundación para transmitir a todos los routers de la red las actualizaciones del estado de los enlaces:

- El algoritmo de Dijkstra se ejecuta como una aplicación separada, fuera de los conmutadores de paquetes.
- Los conmutadores de paquetes envían las actualizaciones de estado de los enlaces al controlador SDN, en vez de intercambiárselas unos con otros.

En este ejemplo, supongamos que el enlace entre los conmutadores s1 y s2 se desactiva; supongamos también que está implementado un enrutamiento basado en la ruta más corta, por lo que las reglas de reenvío para los flujos entrantes y salientes en s1, s2 y s4 se verán afectadas; y supongamos por último que la operación de s3 no se ve afectada. Asumamos también que se utiliza OpenFlow como protocolo de la capa de comunicaciones y que la única función que realiza el plano de control es el enrutamiento basado en el estado de los enlaces.

1. El conmutador s1, al experimentar un fallo en el enlace entre él mismo y s2, notifica el cambio en el estado del enlace al controlador SDN, utilizando el mensaje *port-status* de OpenFlow.

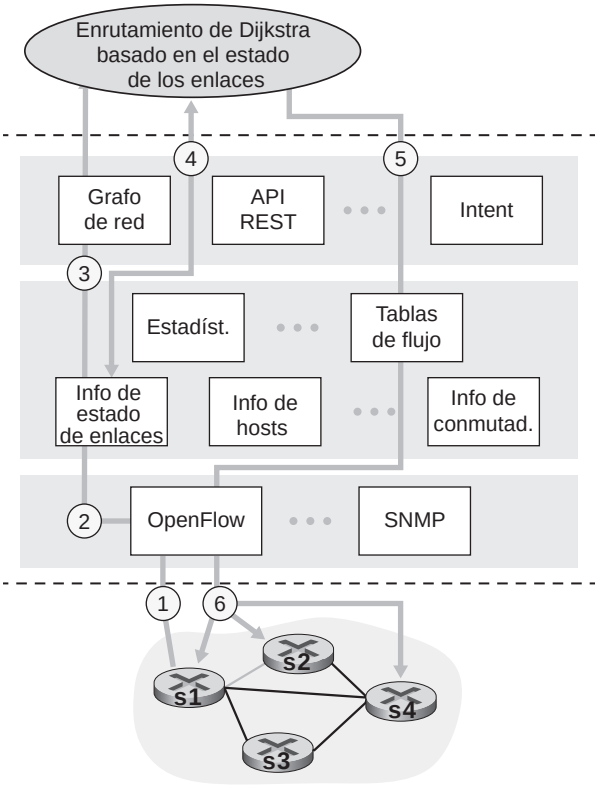


Figura 5.16 ♦ Escenario con controlador SDN: cambio en el estado de un enlace.

2. El controlador SDN recibe el mensaje OpenFlow que indica el cambio en el estado del enlace y envía una notificación al gestor del estado de los enlaces, que actualiza una base de datos de estado de los enlaces.
3. La aplicación de control de red que implementa el enrutamiento de Dijkstra basado en el estado de los enlaces, se había registrado previamente para ser notificada cuando hubiera cambios en el estado de los enlaces. Esa aplicación recibe la notificación del cambio en el estado del enlace.
4. La aplicación de enrutamiento basada en el estado de los enlaces interactúa con el gestor de estado de los enlaces para obtener información actualizada del estado de los enlaces; también puede consultar otros componentes de la capa de gestión del estado. Después calcula las nuevas rutas de coste mínimo.
5. La aplicación de enrutamiento basada en el estado de los enlaces interactúa entonces con el gestor de tablas de flujo, que determina las tablas de flujo que hay que actualizar.
6. El gestor de tablas de flujo utiliza entonces el protocolo OpenFlow para actualizar las entradas de las tablas de flujo de los conmutadores afectados: s1 (que ahora enrutará a través de s4 los paquetes destinados a s2), s2 (que ahora comenzará a recibir paquetes de s1 a través del conmutador intermedio s4) y s4 (que ahora debe reenviar los paquetes de s1 destinados a s2).

Este ejemplo es sencillo, pero ilustra cómo el plano de control SDN proporciona servicios del plano de control (en este caso, enrutamiento de la capa de red) que previamente se habían venido implementando mediante un control por router, ejercido en todos y cada uno de los routers de la red. Ahora podemos ver fácilmente cómo un ISP que utilice SDN podría pasar fácilmente de utilizar rutas de mínimo coste, a emplear una técnica de enrutamiento más especializada. De hecho, puesto que el controlador puede ajustar las tablas de flujo como desee, puede implementar *cualquier* forma de reenvío que quiera, simplemente cambiando su software de aplicación de control. Compare esta facilidad de modificación con el caso de un plano de control tradicional, con control por router, en el que habría que cambiar el software de todos los routers (que podrían haber sido suministrados al ISP por múltiples fabricantes independientes).

5.5.4 SDN: pasado y futuro

Aunque el enorme interés en SDN es un fenómeno relativamente reciente, las raíces técnicas de SDN, y en particular la separación de los planos de datos y de control, se remontan a mucho atrás. En 2004, diversas voces [Feamster 2004, Lakshman 2004, RFC 3746] argumentaron en favor de la separación de los planos de datos y de control de la red. [van der Merwe 1998] describe un marco conceptual de control para redes ATM [Black 1995] con múltiples controladores, cada uno de ellos controlando una serie de conmutadores ATM. El proyecto Ethane [Casado 2007] fue pionero en lo que respecta a la noción de una red de conmutadores Ethernet simples basados en flujo, con tablas de flujo de tipo correspondencia-acción, un controlador centralizado que gestionara la admisión de flujo y el enrutamiento, y el reenvío de los paquetes para los que no se encontrara correspondencia, desde el conmutador al controlador. Ese mismo año 2007 se puso en funcionamiento una red de más de 300 conmutadores Ethane. Después, Ethane evolucionó rápidamente para dar lugar al proyecto OpenFlow y, como dice el dicho, ¡el resto es historia!

Hay numerosos proyectos de investigación dirigidos a desarrollar futuras arquitecturas y capacidades SDN. Como hemos visto, la revolución SDN está conduciendo a la rápida sustitución de conmutadores y routers monolíticos y dedicados (que cuentan con plano de datos y plano de control) por hardware de conmutación sencillo y no diferenciado, al que se añade un sofisticado plano de control software. Una generalización de SDN a la que se conoce con el nombre de NFV (*Network Functions Virtualization*, virtualización de funciones de red) trata también de sustituir rápidamente los dispositivos intermediarios sofisticados (como por ejemplo los dispositivos con hardware dedicado y software propietario para la distribución y almacenamiento en caché de contenidos multimedia) por servidores, dispositivos de conmutación y dispositivos de almacenamiento simples y no diferenciados [Gember-Jacobson 2014]. Una segunda área de investigación también muy

importante trata de ampliar los conceptos de SDN desde el entorno interno a un sistema autónomo al entorno de operación entre sistemas autónomos [Gupta 2014].

EN LA PRÁCTICA

CASOS DE ESTUDIO DE CONTROLADORES SDN: LOS CONTROLADORES OPENDAYLIGHT Y ONOS

En los primeros días de SDN había un único protocolo SDN (OpenFlow [McKeown 2008; OpenFlow 2009]) y un único controlador SDN (NOX [Gude 2008]). Desde entonces, el número de controladores SDN ha crecido significativamente [Kreutz 2015]. Algunos controladores SDN son propietarios y específicos de una empresa, como por ejemplo ONIX [Koponen 2010], Contrail, de Juniper Networks [Juniper Contrail 2016] y el controlador de Google [Jain 2013] para su red de área extensa B4. Pero hay muchos más controladores que son de código abierto, implementados en diversos lenguajes de programación [Erickson 2013]. Más recientemente, el controlador OpenDaylight [OpenDaylight Lithium 2016] y el controlador ONOS [ONOS 2016] han recibido un considerable apoyo del sector. Ambos son de código abierto y están siendo desarrollados en colaboración con la Linux Foundation.

El controlador OpenDaylight

La Figura 5.17 presenta una vista simplificada de la plataforma controladora SDN OpenDaylight Lithium [OpenDaylight Lithium 2016]. El conjunto principal de componentes del controlador ODL se corresponde estrechamente con los que hemos presentado en la Figura 5.15.

Las *aplicaciones de servicio de red* son las aplicaciones que determinan cómo se llevan a cabo el reenvío y otros servicios del plano de datos, como el de cortafuegos y el de equilibrado de carga, en los conmutadores controlados. A diferencia del controlador canónico de la Figura 5.15, el controlador ODL tiene dos interfaces a través de las cuales las aplicaciones pueden comunicarse con los servicios nativos del controlador y entre sí: las aplicaciones externas se comunican con los módulos del controlador utilizando una API REST de solicitud-respuesta que se ejecuta sobre HTTP. Las aplicaciones internas se comunican entre sí a través de la capa SAL (*Service Abstraction Layer*, capa de abstracción de servicios). Es responsabilidad del diseñador de una aplicación controladora decidir si la implementa externa o internamente; la configuración concreta de aplicaciones mostrada en la Figura 5.17 es un mero ejemplo.

Las *funciones básicas de servicio de red* de ODL forman el corazón del controlador y se corresponden estrechamente con las funciones de gestión del estado de la red que ya hemos visto en la Figura 5.15. La capa SAL es el sistema nervioso del controlador, que permite a las aplicaciones y a los componentes del controlador invocar sus mutuos servicios y suscribirse a los sucesos que generen. También proporciona una interfaz abstracta uniforme a los *protocolos de comunicaciones subyacentes* específicos de la capa de comunicaciones, incluyendo OpenFlow y SNMP (*Simple Network Management Protocol*, protocolo simple de gestión de red: un protocolo de gestión de red del que hablaremos en la Sección 5.7). OVSDB es un protocolo utilizado para gestionar la conmutación de centros de datos, un área de aplicación importante para la tecnología SDN. Hablaremos de las redes de centros de datos en el Capítulo 6.

El controlador ONOS

La Figura 5.18 presenta una vista simplificada del controlador ONOS [ONOS 2016]. De forma similar al controlador canónico de la Figura 5.15, podemos identificar tres capas en el controlador ONOS:

- *Protocolos y abstracciones norte.* Una característica original de ONOS es su infraestructura Intent, que permite a una aplicación solicitar un servicio de alto nivel (por ejemplo, establecer una conexión entre el host A y el host B o, a la inversa, no permitir que el host A y el host B se comuniquen) sin necesidad de conocer los detalles de cómo se implementa dicho servicio. La información de estado se proporciona a las aplicaciones de control de red a través de la API norte, bien síncronamente (mediante consultas) o

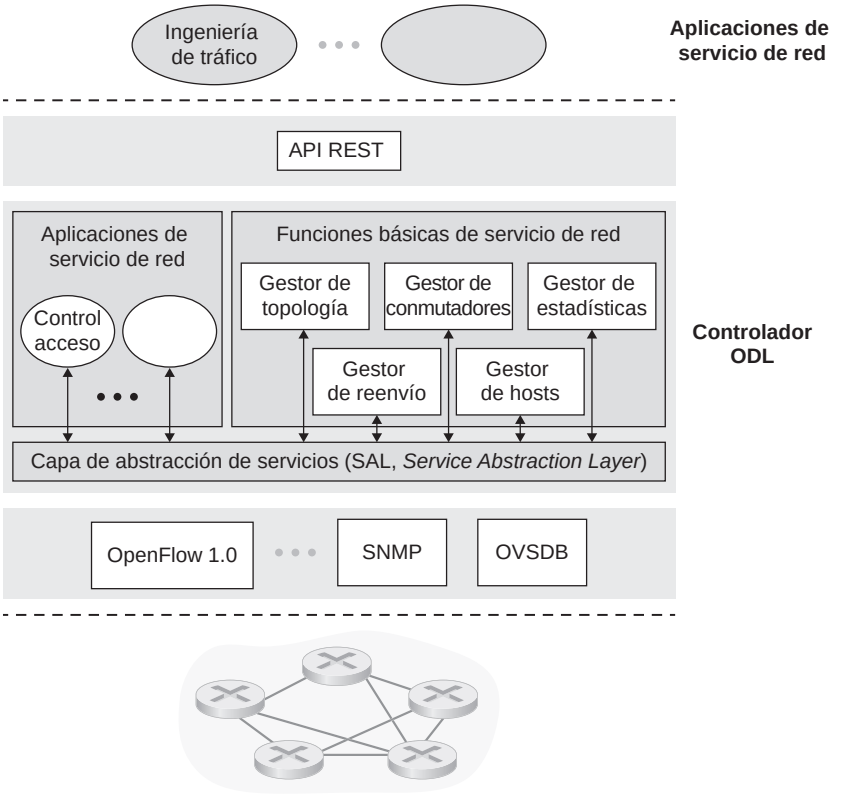


Figura 5.17 ♦ El controlador OpenDaylight.

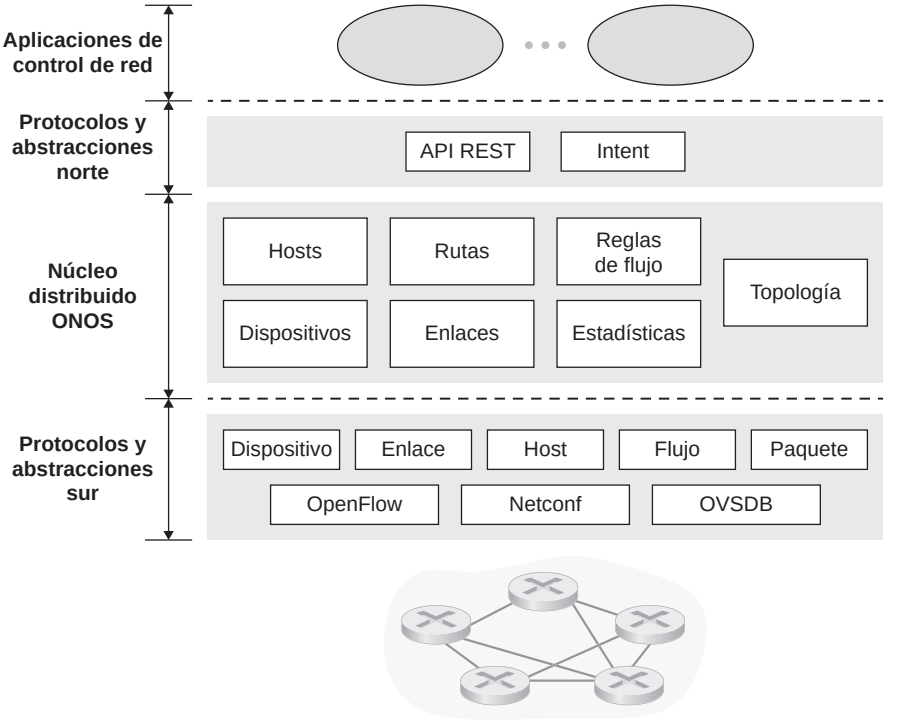


Figura 5.18 ♦ Arquitectura del controlador ONOS.

asíncronamente (a través de retrollamadas a procesos escucha, por ejemplo cuando cambia el estado de la red).

- *Núcleo distribuido.* El estado de los enlaces, hosts y dispositivos de la red se mantiene en el núcleo distribuido de ONOS. ONOS se implanta como un servicio en un conjunto de servidores interconectados, ejecutándose en cada servidor una copia idéntica del software ONOS; a mayor número de servidores, mayor capacidad de servicio. El núcleo de ONOS proporciona los mecanismos de replicación y coordinación del servicio entre instancias, proporcionando a las aplicaciones situadas por encima y a los dispositivos de red situados por debajo la abstracción de unos servicios del núcleo lógicamente centralizados.
- *Protocolos y abstracciones sur.* Las abstracciones sur enmascaran la heterogeneidad de los hosts, enlaces, conmutadores y protocolos subyacentes, permitiendo que el núcleo distribuido sea independiente de los dispositivos y de los protocolos. Debido a esta abstracción, la interfaz sur situada por debajo del núcleo distribuido está a un nivel lógico superior que en nuestro controlador canónico de la Figura 5.15 o en el controlador ODL de la Figura 5.17.

5.6 Protocolo de mensajes de control de Internet (ICMP)

Los hosts y los routers utilizan ICMP (*Internet Control Message Protocol*, protocolo de mensajes de control de Internet), especificado en [RFC 792], para intercambiarse información acerca de la capa de red. El uso más típico de ICMP es la generación de informes de error. Por ejemplo, al ejecutar una sesión HTTP podemos encontrarnos con un mensaje de error como "Red de destino inalcanzable". Este mensaje tiene su origen en ICMP. En algún punto, un router IP no ha podido encontrar una ruta hasta el host especificado en la solicitud HTTP, y dicho router ha creado y enviado un mensaje ICMP a nuestro host para informarle del error.

ICMP a menudo se considera parte de IP pero, en sentido arquitectónico, se encuentra justo encima de IP, ya que los mensajes ICMP son transportados dentro de datagramas IP. Es decir, los mensajes ICMP son transportados como carga útil de IP, al igual que los segmentos TCP o UDP son transportados como carga útil de IP. De forma similar, cuando un host recibe un datagrama IP con ICMP especificado como el protocolo de la capa superior (número de protocolo de la capa superior igual a 1), demultiplexa el contenido del datagrama hacia ICMP, al igual que demultiplexaría el contenido de un datagrama hacia TCP o UDP.

Los mensajes ICMP tienen un campo de tipo y un campo de código, y contienen la cabecera y los 8 primeros bytes del datagrama IP que ha dado lugar a la generación del mensaje ICMP (para que el emisor pueda determinar qué datagrama ha producido el error). En la Figura 5.19 se muestra una serie de tipos de mensajes ICMP seleccionados. Observe que los mensajes ICMP no solo se emplean para señalizar condiciones de error.

El famoso programa ping envía un mensaje ICMP de tipo 8 y código 0 al host especificado. El host de destino, al ver la solicitud de eco, devuelve una respuesta de eco ICMP de tipo 0 y código 0. La mayoría de las implementaciones de TCP/IP soportan el servidor ping directamente en el sistema operativo; es decir, el servidor no es un proceso. El Capítulo 11 de [Stevens 1990] proporciona el código fuente del programa cliente ping. Observe que el programa cliente necesita poder instruir al sistema operativo para generar un mensaje ICMP de tipo 8 y código 0.

Otro interesante mensaje ICMP es el mensaje de regulación del origen. Este mensaje rara vez se emplea en la práctica. Su propósito original era llevar a cabo el control de congestión (permitir a un router congestionado enviar un mensaje ICMP de este tipo a un host para forzarle a reducir su velocidad de transmisión). Hemos visto en el Capítulo 3 que TCP dispone de su propio mecanismo control de congestión que opera en la capa de transporte, sin utilizar ninguna realimentación de la capa de red, como el mensaje ICMP de regulación del origen.

Tipo ICMP	Código	Descripción
0	0	respuesta de eco (para ping)
3	0	red de destino inalcanzable
3	1	host de destino inalcanzable
3	2	protocolo de destino inalcanzable
3	3	puerto de destino inalcanzable
3	6	red de destino desconocida
3	7	host de destino desconocido
4	0	regulación del origen (control de congestión)
8	0	solicitud de eco
9	0	anuncio de router
10	0	descubrimiento de router
11	0	TTL caducado
12	0	Cabecera IP errónea

Figura 5.19 ♦ Tipos de mensajes ICMP.

En el Capítulo 1 hemos introducido el programa Traceroute, el cual nos permite trazar una ruta desde un host a cualquier otro host del mundo. Merece la pena resaltar que Traceroute se implementa con mensajes ICMP. Para determinar los nombres y las direcciones de los routers existentes entre el origen y el destino, el programa Traceroute del origen envía una serie de datagramas IP ordinarios al destino. Cada uno de estos datagramas transporta un segmento UDP con un número de puerto UDP poco probable. El primero de estos datagramas tiene un TTL de 1, el segundo de 2, el tercero de 3, y así sucesivamente. El origen también inicia sendos temporizadores para cada uno de los datagramas. Cuando el datagrama *n*-ésimo llega al router *n*-ésimo, este observa que el TTL del datagrama acaba de caducar. De acuerdo con las reglas del protocolo IP, el router descarta el datagrama y envía al origen un mensaje de advertencia ICMP (tipo 11, código 0). Este mensaje de advertencia incluye el nombre del router y su dirección IP. Cuando este mensaje ICMP llega de vuelta al origen, este obtiene el tiempo de ida y vuelta del temporizador, y obtiene también del propio mensaje ICMP el nombre y la dirección IP del router *n*-ésimo.

¿Cómo sabe un origen Traceroute cuándo dejar de enviar segmentos UDP? Recuerde que el origen incrementa el valor del campo TTL cada vez que envía un datagrama. Por tanto, uno de los datagramas terminará recorriendo el camino completo hasta el host de destino. Dado que ese datagrama contiene un segmento UDP con un número de puerto improbable, el host de destino devuelve al origen un mensaje ICMP de puerto inalcanzable (tipo 3, código 3). Cuando el host de origen recibe este mensaje ICMP, sabe que no tiene que enviar más paquetes de sondeo. (Realmente, el programa estándar Traceroute envía conjuntos de tres paquetes con el mismo TTL; es por ello que la salida de Traceroute proporciona tres resultados para cada TTL.)

De esta forma, el host de origen obtiene el número y la identidad de los routers que existen entre él y el host de destino, así como el tiempo de ida y vuelta entre los dos hosts. Observe que el programa cliente Traceroute tiene que poder instruir al sistema operativo para generar datagramas UDP con valores TTL específicos, y que el sistema operativo también tiene que ser capaz de notificarle la llegada de mensajes ICMP. Ahora que comprende cómo funciona Traceroute, puede volver atrás y practicar con él un poco más.

En RFC 4443 se ha definido una nueva versión de ICMP para IPv6. Además de reorganizar las definiciones existentes de tipos y códigos ICMP, ICMPv6 también añade nuevos tipos y códigos, requeridos por la nueva funcionalidad IPv6. Entre estos se incluyen el tipo "Packet too big" (paquete demasiado grande) y un código de error "unrecognized IPv6 options" (opciones IPv6 no reconocidas).

5.7 Gestión de red y SNMP

Habiendo finalizado nuestro estudio de la capa de red, y cuando ya solo nos queda por delante la capa de enlace, sabemos que una red está compuesta por muchos elementos hardware y software complejos, que interactúan unos con otros: elementos que van desde los enlaces, switches, routers, hosts y otros dispositivos que constituyen los componentes físicos de la red, hasta los muchos protocolos que controlan y coordinan estos dispositivos. Cuando una organización junta centenares o miles de esos componentes para formar una red, se vuelve todo un desafío el trabajo del administrador de la red, que no es otro que mantener la red funcionando. Hemos visto en la Sección 5.5 que el controlador lógicamente centralizado puede ayudar con este proceso, en un contexto SDN. Pero el desafío representado por la gestión de la red ha existido desde bastante antes que SDN, existiendo un rico conjunto de herramientas y técnicas de gestión de red que ayudan al administrador de la red a monitorizarla, gestionarla y controlarla. En esta sección estudiaremos esas herramientas y técnicas.

A menudo suele plantearse la pregunta de "¿Qué es la gestión de red?". Una definición de la gestión de red muy bien concebida, en una sola frase (aunque hay que reconocer que un poco larga), es la que da [Saydam 1996]:

> *La gestión de red incluye la implantación, integración y coordinación del hardware, el software y los elementos humanos para monitorizar, probar, sondear, configurar, analizar, evaluar y controlar los elementos y recursos de la red, con el fin de satisfacer los requisitos de tiempo real, de rendimiento operativo y de Calidad de Servicio, a un coste razonable.*

Dada esta amplia definición, vamos a cubrir en esta sección únicamente los rudimentos de la gestión de red: la arquitectura, los protocolos y la base de información que un administrador de red utiliza para llevar a cabo su tarea. No hablaremos de los procesos de toma de decisiones de los administradores, en los que hay que tener en cuenta la identificación de fallos [Labovitz 1997; Steinder 2002; Feamster 2005; Wu 2005; Teixeira 2006], la detección de anomalías [Lakhina 2005; Barford 2009], el diseño/ingeniería de la red para satisfacer los acuerdos de nivel de servicio (SLA, *Service Level Agreement*) contratados [Huston 1999a] y otros temas. Nuestro enfoque será, por tanto, deliberadamente reducido; el lector interesado puede consultar las referencias indicadas, el excelente libro de gestión de red de Subramanian [Subramanian 2000] y el tratamiento más detallado de la gestión de red disponible en el sitio web del presente libro.

5.7.1 El marco conceptual de la gestión de red

La Figura 5.20 muestra los componentes clave de la gestión de red:

- El **servidor de gestión** es una aplicación, normalmente con intervención humana, que se ejecuta en una estación central de gestión de red situada en el centro de operaciones de red (NOC, *Network Operations Center*). El servidor de gestión es el punto focal de la actividad de administración de la red; controla la recopilación, procesamiento, análisis y/o visualización de la información de gestión de la red. Es aquí donde se inician las acciones para el control del comportamiento de la red y donde el administrador interactúa con los dispositivos que forman la red.
- Un **dispositivo gestionado** es un equipo de red (incluyendo su software) que forma parte de una red gestionada. Un dispositivo gestionado puede ser un host, un router, un switch, un dispositivo

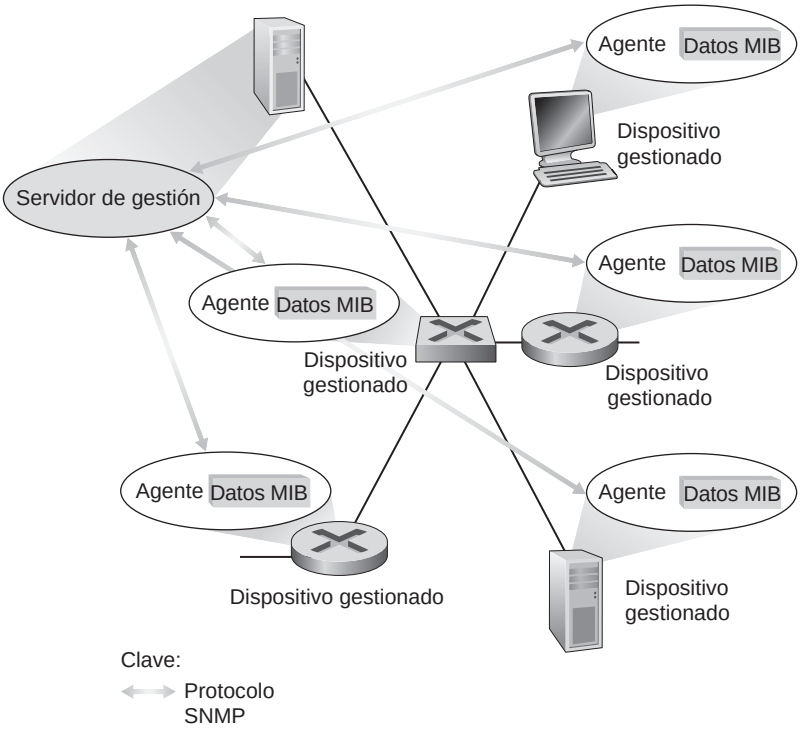


Figura 5.20 ♦ Elementos de gestión de red: servidor de gestión, dispositivos gestionados, datos MIB, agentes remotos, SNMP.

intermediario, un módem, un termómetro o cualquier otro dispositivo conectado a la red. Dentro de un dispositivo gestionado pueden existir diversos **objetos gestionados**. Estos objetos gestionados son los propios elementos hardware contenidos en el dispositivo gestionado (por ejemplo, una tarjeta de interfaz de red no es más que uno de los componentes de un host o un router) y los parámetros de configuración para los distintos elementos hardware y software (por ejemplo, un protocolo de enrutamiento interior al sistema autónomo, como OSPF).

- Cada objeto gestionado de un dispositivo gestionado tiene asociados distintos elementos de información, que se recopilan en una **base de información de gestión (MIB, *Management Information Base*)**; como veremos, los valores de estos elementos de información están a disposición del servidor de gestión (que en muchos casos también puede modificarlos). Un objeto de la MIB puede ser un contador, como por ejemplo el número de datagramas IP descartados en un router debido a errores en la cabecera del datagrama, o el número de segmentos UDP recibidos en un host; o puede ser información de carácter descriptivo, como por ejemplo la versión del software que se está ejecutando en un servidor DNS; o puede ser información de estado, como por ejemplo si un cierto dispositivo está funcionando correctamente; o puede ser información específica del protocolo, como por ejemplo una ruta hasta un destino. Los objetos MIB se especifican en un lenguaje de descripción de datos denominado SMI (*Structure of Management Information*, estructura de información de gestión) [RFC 2578; RFC 2579; RFC 2580]. Se utiliza un lenguaje de definición formal para garantizar que la sintaxis y la semántica de los datos de gestión de red estén bien definidas y sean no ambiguas. Los objetos MIB relacionados se agrupan en módulos MIB. A mediados de 2015, los distintos documentos RFC definían casi 400 módulos MIB, existiendo un número mucho mayor aún de módulos MIB privados (específicos del fabricante).
- En cada dispositivo gestionado reside también un **agente de gestión de red**, que es un proceso que se ejecuta en el dispositivo gestionado y que se comunica con el servidor de gestión, llevando

a cabo acciones locales en el dispositivo gestionado bajo control y por orden del servidor de gestión. El agente de gestión de red es similar al agente de enrutamiento que vimos en la Figura 5.2.

- El último componente del marco conceptual de la gestión de red es el **protocolo de gestión de red**. El protocolo se ejecuta entre el servidor de gestión y los dispositivos gestionados, permitiendo al servidor consultar el estado de los dispositivos gestionados y llevar a cabo acciones de manera indirecta en estos dispositivos a través de sus agentes. Los agentes pueden utilizar el protocolo de gestión de red para informar al servidor de gestión de que se han producido sucesos excepcionales (por ejemplo, fallos de componentes o violación de los umbrales de rendimiento). Es importante recalcar que el protocolo de gestión de red no se encarga él mismo de administrar la red; simplemente proporciona una serie de capacidades que un administrador puede utilizar para gestionar ("monitorizar, probar, sondear, configurar, analizar, evaluar y controlar") la red. Se trata de una distinción sutil, pero de gran importancia. En la siguiente sección hablaremos del protocolo SNMP (Simple Network Management Protocol, protocolo simple de gestión de red) de Internet.

5.7.2 El protocolo SNMP

El **protocolo simple de gestión de red** versión 2 (SNMPv2) [RFC 3416] es un protocolo de la capa de aplicación que se utiliza para transportar mensajes de información y control para la gestión de red entre un servidor de gestión y un agente que actúa por cuenta del servidor de gestión. El uso más común de SNMP es el modo de solicitud-respuesta, en el que un servidor de gestión SNMP envía una solicitud a un agente SNMP, el cual la recibe, lleva a cabo ciertas acciones y envía una respuesta a dicha solicitud. Normalmente, una solicitud se utilizará para consultar (recuperar) o modificar (establecer) los valores de objetos MIB asociados con un dispositivo gestionado. Un segundo uso común de SNMP es cuando un agente envía un mensaje no solicitado, conocido como mensaje TRAP, a un servidor de gestión. Los mensajes TRAP se utilizan para notificar a un servidor de gestión una situación excepcional (por ejemplo, la activación o desactivación de un enlace) que ha dado lugar a cambios en los valores de los objetos MIB.

SNMPv2 define siete tipos de mensajes, conocidos genéricamente como unidades de datos de protocolo (PDU, *Protocol Data Unit*), que se enumeran en la Tabla 5.2 y se describen a continuación. El formato de la PDU se muestra en la Figura 5.21.

- Las PDU `GetRequest`, `GetNextRequest` y `GetBulkRequest` son enviadas desde un servidor de gestión a un agente para solicitar el valor de uno o más objetos MIB del dispositivo gestionado por el agente. Los objetos MIB cuyos valores están siendo solicitados se especifican en la parte de las variables de la PDU. `GetRequest`, `GetNextRequest` y `GetBulkRequest` difieren en la granularidad de sus solicitudes de datos. `GetRequest` puede solicitar un conjunto arbitrario de valores MIB; pueden utilizarse múltiples `GetNextRequest`s para recorrer secuencialmente una lista o tabla de objetos MIB; `GetBulkRequest` permite devolver un gran bloque de datos, evitando la sobrecarga en que se incurre si se envian varios mensajes `GetRequest` o `GetNextRequest`. En los tres casos, el agente responde con una PDU `Response` que contiene los identificadores de objeto y sus valores asociados.
- Un servidor de gestión utiliza la PDU `SetRequest` para establecer el valor de uno o más objetos MIB en un dispositivo gestionado. Un agente responde con una PDU `Response` con el estado de error "noError", con el fin de confirmar que el valor ha sido definido.
- Un servidor de gestión utiliza una PDU `InformRequest` para notificar a otro servidor de gestión cierta información MIB que es remota al servidor receptor.
- La PDU `Response` es enviada normalmente por un dispositivo gestionado al servidor de gestión en respuesta a un mensaje de solicitud recibido de ese servidor. Con esta PDU se devuelve la información solicitada.

Tipo de PDU de SNMPv2	Emisor-receptor	Descripción
`GetRequest`	gestor a agente	Obtener el valor de una o más instancias de objeto MIB.
`GetNextRequest`	gestor a agente	Obtener el valor de la siguiente instancia de objeto MIB en una lista o tabla.
`GetBulkRequest`	gestor a agente	Obtener valores en un bloque grande de datos, por ejemplo, en una tabla de gran tamaño.
`InformRequest`	gestor a gestor	Informar a la entidad gestora remota de valores MIB remotos a su acceso.
`SetRequest`	gestor a agente	Establecer el valor de una o más instancias de objeto MIB.
`Response`	agente a gestor o gestor a gestor	Generado en respuesta a: `GetRequest`, `GetNextRequest`, `GetBulkRequest`, `SetRequest PDU`, o `InformRequest`
`SNMPv2-Trap`	agente a gestor	Informar al gestor del número de un suceso excepcional.

Tabla 5.2 ♦ Tipos de PDU de SNMPv2.

- El último tipo de PDU de SNMPv2 es el mensaje TRAP. Los mensajes TRAP son generados asíncronamente; es decir, no son generados en respuesta a una solicitud recibida, sino en respuesta a un suceso para el que el servidor de gestión requiere una notificación. El documento RFC 3418 define tipos de mensajes TRAP bien conocidos, como el arranque en frío o caliente de un dispositivo, la activación o desactivación de un enlace, la pérdida de un vecino o un suceso de fallo de autenticación. Una solicitud TRAP recibida no requiere ninguna respuesta por parte del servidor de gestión.

Dada la naturaleza solicitud-respuesta de SNMP, es conveniente señalar aquí que, aunque las PDU SNMP pueden ser transportadas por medio de muchos protocolos de transporte distintos, normalmente son transportadas en la carga útil de un datagrama UDP. De hecho, el documento RFC 3417

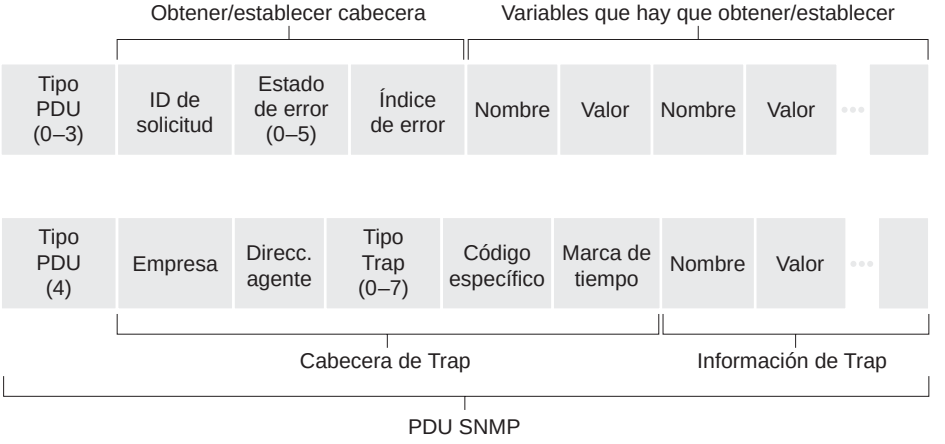


Figura 5.21 ♦ Formato de la PDU de SNMP.

establece que UDP es "la correspondencia de transporte preferida". Sin embargo, dado que UDP es un protocolo de transporte no fiable, no existe ninguna garantía de que una solicitud, o su respuesta, sea recibida por el destino. El servidor de gestión utiliza el campo ID de solicitud de la PDU (véase la Figura 5.21) para numerar las solicitudes que envía a un agente; la respuesta del agente toma su ID de solicitud de la solicitud recibida. Por tanto, el campo ID de solicitud puede ser utilizado por el servidor de gestión para detectar las solicitudes y respuestas perdidas. Dependerá del servidor de gestión decidir si retransmite una solicitud si no ha recibido la respuesta correspondiente transcurrido un cierto periodo de tiempo. En particular, el estándar SNMP no obliga a aplicar ningún procedimiento en concreto para las retransmisiones y ni siquiera indica si la retransmisión debe llevarse a cabo. Solo obliga a que el servidor de gestión "actúe de forma responsable respecto a la frecuencia y duración de las retransmisiones". ¡Por supuesto, esto lleva a preguntarse cómo debe actuar un protocolo "responsable"!

SNMP ha evolucionado en tres versiones. Los diseñadores de SNMPv3 han señalado que "SNMPv3 puede interpretarse como SNMPv2 con una serie de capacidades adicionales de seguridad y administración" [RFC 3410]. Realmente, existen cambios en SNMPv3 respecto de SNMPv2, pero en ninguna parte son más evidentes estos cambios que en el área de la administración y la seguridad. El papel central de la seguridad en SNMPv3 era particularmente importante, ya que la falta de una adecuada seguridad hizo que SNMP se usara principalmente para monitorizar, más que para cuestiones de control (por ejemplo, rara vez se emplea `SetRequest` en SNMPv1). De nuevo, vemos que la seguridad —un tema del que trataremos en el Capítulo 8— es un problema crítico, pero un problema cuya importancia, de nuevo, solo se ha reconocido un poco tarde, habiéndose "añadido" entonces las soluciones.

5.8 Resumen

Ya hemos completado estos dos capítulos dedicados al estudio del núcleo de la red: un estudio que comenzó con el análisis del plano de datos de la capa de red en el Capítulo 4 y que ha finalizado aquí, con el repaso al plano de control de dicha capa. Hemos visto que el plano de control es la lógica de red que controla no solo cómo se reenvía un datagrama entre routers, a lo largo de una ruta extremo a extremo que va desde el host de origen hasta el host de destino, sino también cómo se configuran y gestionan los componentes y servicios de la capa de red.

Hemos visto que hay dos enfoques generales para construir un plano de control: el *control por router* tradicional (en el que se ejecuta un algoritmo de enrutamiento en todos y cada uno de los routers y el componente de enrutamiento del router se comunica con sus homólogos de otros routers) y el control mediante *redes definidas por software* (SDN), en el que un controlador lógicamente centralizado calcula y distribuye las tablas de reenvío que deben utilizar todos y cada uno de los routers. Hemos estudiado dos algoritmos fundamentales de enrutamiento para calcular las rutas de menor coste en un grafo —el enrutamiento por estado de los enlaces y el enrutamiento de vectores de distancia— en la Sección 5.2; estos algoritmos encuentran su aplicación tanto en el control por router como en el control SDN. Dichos algoritmos son la base de dos protocolos de enrutamiento muy difundidos en Internet, OSPF y BGP, de los que hemos hablado en las secciones 5.3 y 5.4. En la Sección 5.5 hemos analizado el enfoque SDN de implementación del plano de control de la capa de red, investigando las aplicaciones de control de red SDN, el controlador SDN y el protocolo OpenFlow para la comunicación entre el controlador y los dispositivos controlados por SDN. En las secciones 5.6 y 5.7 hemos repasado algunas de las interioridades de la gestión de una red IP: ICMP (el protocolo de mensajes de control de Internet) y SNMP (el protocolo simple de gestión de red).

Habiendo completado nuestro estudio de la capa de red, vamos a descender un escalón más por la pila de protocolos, concretamente hasta la capa de enlace. Al igual que la capa de red, la capa de enlace también forma parte de todos y cada uno de los dispositivos conectados a la red. Pero en el siguiente capítulo veremos que la capa de enlace tiene la tarea mucho más localizada de transferir

los paquetes entre nodos situados en un mismo enlace o LAN. Aunque esta tarea puede parecer en principio trivial, comparada con las tareas que desempeña la capa de red, veremos que la capa de enlace implica una serie de cuestiones importantes y fascinantes, que nos mantendrán ocupados durante bastante tiempo.

Problemas y cuestiones de repaso

Capítulo 5 Cuestiones de repaso

SECCIÓN 5.1

R1. ¿A qué nos referimos al hablar de un plano de control basado en el control por router? En tales casos, cuando decimos que los planos de datos y de control de la red están implementados "monolíticamente", ¿qué queremos decir?

R2. ¿A qué nos referimos al hablar de un plano de control basado en un control lógicamente centralizado? En tales casos, ¿el plano de datos y el plano de control se implementan en el mismo dispositivo o en dispositivos diferentes? Explique su respuesta.

SECCIÓN 5.2

R3. Indique las similitudes y diferencias entre los algoritmos de enrutamiento centralizados y distribuidos. Proporcione un ejemplo de protocolo de enrutamiento que adopte un enfoque centralizado y otro descentralizado.

R4. Indique las similitudes y diferencias entre los algoritmos de enrutamiento de estado de los enlaces y por vector de distancias.

R5. ¿Qué es el problema de la "cuenta hasta infinito" en el enrutamiento por vector de distancias?

R6. ¿Es necesario que todos los sistemas autónomos utilicen el mismo algoritmo de enrutamiento interno? ¿Por qué o por qué no?

SECCIONES 5.3–5.4

R7. ¿Por qué se utilizan diferentes protocolos de enrutamiento internos de un sistema autónomo y entre sistemas autónomos en Internet?

R8. Verdadero o falso: cuando un router OSPF envía su información de estado de los enlaces, solo la envía a los vecinos directamente conectados. Explique su respuesta.

R9. ¿A qué se llama *área* en un sistema autónomo OSPF? ¿Por qué se introdujo el concepto de área?

R10. Defina los siguientes términos e indique en qué se diferencian: *subred*, *prefijo* y *ruta BGP*.

R11. ¿Cómo utiliza BGP el atributo NEXT-HOP? ¿Cómo utiliza el atributo AS-PATH?

R12. Describa cómo un administrador de red de un ISP de nivel superior puede implementar ciertas políticas al configurar BGP.

R13. Verdadero o falso: cuando un router BGP recibe un ruta anunciada por su vecino, debe añadir su propia identidad a la ruta recibida y luego enviar esa nueva ruta a todos sus vecinos. Explique su respuesta.

SECCIÓN 5.5

R14. Describa el papel principal de la capa de comunicaciones, de la capa de gestión del estado de la red y de la capa de aplicaciones de control de red en un controlador SDN.

R15. Suponga que quiere implementar un nuevo protocolo de enrutamiento en el plano de control SDN. ¿En qué capa implementaría ese protocolo? Explique su respuesta.

R16. ¿Qué tipos de mensajes fluyen a través de las API norte y sur de un controlador SDN? ¿Quién es el receptor de los mensajes que el controlador envía a través de la interfaz sur y quién envía mensajes al controlador a través de la interfaz norte?

R17. Describa el propósito de dos tipos de mensajes OpenFlow (a su elección) enviados desde un dispositivo controlado al controlador. Describa el propósito de dos tipos de mensajes OpenFlow (a su elección) enviados desde el controlador a un dispositivo controlado.

R18. ¿Cuál es el propósito de la capa de abstracción de servicios en el controlador SDN OpenDaylight?

SECCIONES 5.6–5.7

R19. Enumere cuatro tipos distintos de mensajes ICMP.

R20. ¿Qué dos tipos de mensajes ICMP se reciben en el host emisor que está ejecutando el programa *Traceroute*?

R21. Defina los siguientes términos en el contexto de SNMP: *servidor de gestión, dispositivo gestionado, agente de gestión de red y MIB.*

R22. ¿Cuál es el objetivo de los mensajes SNMP *GetRequest* y *SetRequest*?

R23. ¿Cuál es el objetivo del mensaje SNMP `Trap`?

Problemas

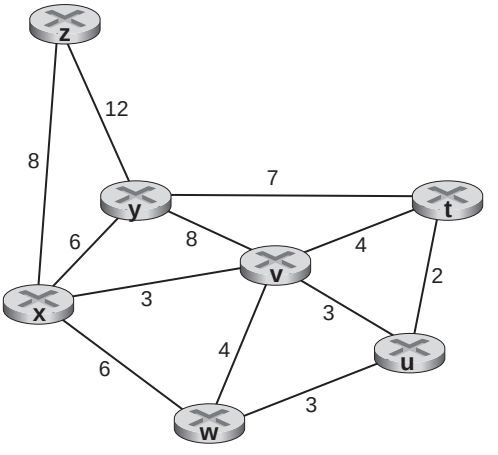
P1. En la Figura 5.3, enumere las rutas desde *y* hasta *u* que no contienen bucles.

P2. Repita el Problema P1 para las rutas de *x* a *z*, de *z* a *u* y de *z* a *w*.

Nota de vídeo

Algoritmo de Dijkstra: explicación y ejemplo

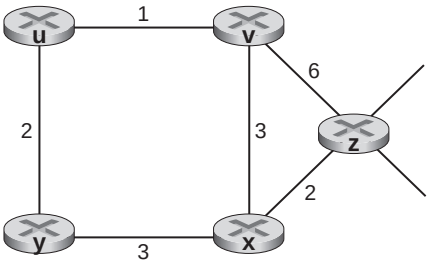
P3. Considere la siguiente red. Utilizando los costes de enlace indicados, aplique el algoritmo de la ruta más corta de Dijkstra para calcular la ruta más corta desde *x* a todos los restantes nodos de red. Indique cómo funciona el algoritmo elaborando una tabla similar a la Tabla 5.1.



P4. Considere la red mostrada en el Problema P3. Utilizando el algoritmo de Dijkstra y una tabla similar a la Tabla 5.1, haga lo siguiente:

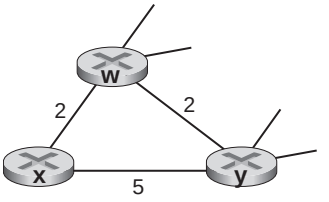
a. Calcule la ruta más corta desde *t* a todos los demás nodos de la red.
b. Calcule la ruta más corta desde *u* a todos los demás nodos de la red.
c. Calcule la ruta más corta desde *v* a todos los demás nodos de la red.
d. Calcule la ruta más corta desde *w* a todos los demás nodos de la red.
e. Calcule la ruta más corta desde *y* a todos los demás nodos de la red.
f. Calcule la ruta más corta desde *z* a todos los demás nodos de la red.

P5. Utilice la red que se muestra a continuación y suponga que cada nodo inicialmente conoce los costes hasta cada uno de sus vecinos. Utilizando el algoritmo de vector de distancias, especifique las entradas de la tabla de distancias para el nodo *z*.



P6. Considere una topología general (es decir, no la red concreta mostrada más arriba) y una versión síncrona del algoritmo de vector de distancias. Suponga que en cada iteración un nodo intercambia sus vectores distancia con sus vecinos y recibe los vectores distancia de ellos. Suponiendo que el algoritmo se inicia con cada nodo conociendo solo los costes de sus vecinos inmediatos, ¿cuál es el número máximo de iteraciones requerido antes de que el algoritmo distribuido converja? Justifique su respuesta.

P7. Considere el fragmento de red mostrado a continuación. *x* solo tiene dos vecinos conectados, *w* e *y*. *w* tiene una ruta de coste mínimo al destino *u* (no mostrado) de 5 e *y* tiene una ruta de coste mínimo a *u* de 6. Las rutas completas desde *w* e *y* a *u* (y entre *w* e *y*) no se muestran. Todos los costes de enlace de la red tienen valores enteros estrictamente positivos.



a. Indique el vector de distancias de *x* para los destinos *w, y* y *u*.

b. Indique un cambio en el coste del enlace para $c(x,w)$ o $c(x,y)$ tal que *x* informe a sus vecinos de una nueva ruta de coste mínimo a *u*, como resultado de ejecutar el algoritmo de vector de distancias.

c. Indique un cambio en el coste del enlace para $c(x,w)$ o $c(x,y)$ tal que *x no* informe a sus vecinos de una nueva ruta de coste mínimo a *u*, como resultado de ejecutar el algoritmo de vector de distancias.

P8. Considere la topología de tres nodos mostrada en la Figura 5.6. En lugar de tener los costes de enlace mostrados en dicha figura, los costes de enlace son $c(x,y) = 3$, $c(y,z) = 6$, $c(z,x) = 4$. Calcule las tablas de distancias después del paso de inicialización y después de cada iteración de una versión síncrona del algoritmo de vector de distancias (como hemos hecho anteriormente al explicar la Figura 5.6).

P9. Considere el problema de la cuenta hasta infinito en el enrutamiento por vector de distancias. ¿Se producirá dicho problema si disminuimos el coste de un enlace? ¿Por qué? ¿Qué ocurre si conectamos dos nodos que no tienen un enlace?

P10. Demuestre que al aplicar el algoritmo de vector de distancias en la Figura 5.6, cada valor del vector de distancias $D(x)$ es no creciente y finalmente se estabilizará en un número finito de pasos.

P11. Considere la Figura 5.7. Suponga que existe otro router *w*, conectado a los routers *y* y *z*. Los costes de todos los enlaces son los siguientes: $c(x,y) = 4$, $c(x,z) = 50$, $c(y,w) = 1$,

$c(z,w) = 1$, $c(y,z) = 3$. Suponga que se utiliza inversa envenenada en el algoritmo de enrutamiento por vector de distancias.

a. Cuando el enrutamiento por vector de distancias se estabiliza, los routers *w*, *y* y *z* se informan de sus respectivas distancias a *x*. ¿Cuáles son los valores de esas distancias?

b. Ahora suponga que el coste del enlace entre *x* e *y* aumenta a 60. ¿Se producirá un problema de cuenta hasta infinito aunque se utilice inversa envenenada? ¿Por qué? Si existe el problema de cuenta hasta infinito, entonces ¿cuántas iteraciones serán necesarias para que el enrutamiento por vector de distancias alcance de nuevo un estado estable? Justifique su respuesta.

c. ¿Cómo modificaría $c(y,z)$ para que no existiera el problema de cuenta hasta infinito si $c(y,x)$ cambia de 4 a 60?

P12. Describa cómo puede detectarse en BGP la existencia de bucles en las rutas.

P13. ¿Un router BGP elegirá siempre la ruta sin bucles con la longitud más corta de la secuencia AS-PATH? Justifique su respuesta.

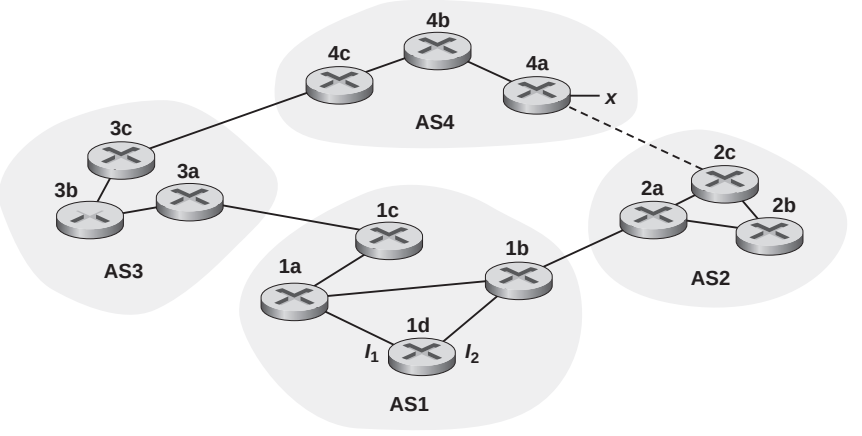
P14. Considere la red mostrada a continuación. Suponga que los sistemas autónomos AS3 y AS2 están ejecutando OSPF como protocolo de enrutamiento interno. Suponga que AS1 y AS4 están ejecutando RIP como protocolo de enrutamiento interno. Suponga por último que se utilizan eBGP e iBGP para el protocolo de enrutamiento entre sistemas autónomos. Además, inicialmente *no* existe enlace físico entre AS2 y AS4.

a. ¿De qué protocolo de enrutamiento aprende el router 3c acerca del prefijo *x*: OSPF, RIP, eBGP o iBGP?

b. ¿De qué protocolo de enrutamiento aprende el router 3a acerca de *x*?

c. ¿De qué protocolo de enrutamiento aprende el router 1c acerca de *x*?

d. ¿De qué protocolo de enrutamiento aprende el router 1d acerca de *x*?



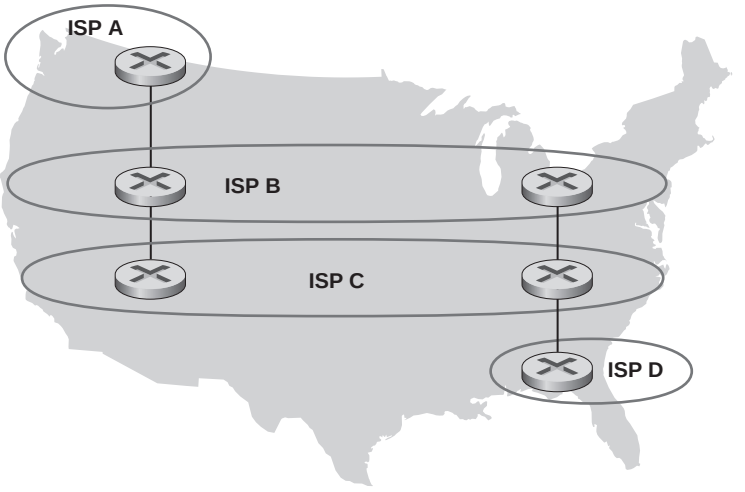
P15. Continuando con el problema anterior, una vez que el router 1d aprende acerca de *x* incluirá una entrada (*x*, *I*) en su tabla de reenvío.

a. Para esta entrada, ¿*I* será igual a I_1 o a I_2? Explique por qué en una frase.

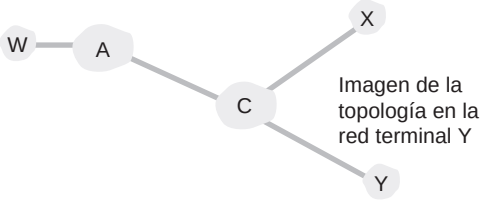
b. Ahora suponga que existe un enlace físico entre AS2 y AS4 (mostrado mediante una línea de puntos en la figura). Suponga que el router 1d aprende que *x* es accesible a través de AS2 y de AS3. ¿*I* será igual a I_1 o a I_2? Explique por qué en una frase.

c. Ahora suponga que existe otro sistema autónomo AS5, que conecta la ruta entre AS2 y AS4 (no se muestra en el diagrama). Suponga que el router 1d aprende que *x* es accesible a través de AS2 AS5 AS4, así como de AS3 AS4. ¿*I* será igual a I_1 o a I_2? Explique por qué en una frase.

P16. Considere la red mostrada a continuación. El ISP B proporciona un servicio troncal nacional al ISP A regional. El ISP C ofrece un servicio troncal nacional al ISP D regional. Cada ISP consta de un sistema autónomo. Los pares B y C se comunican entre sí por dos puntos utilizando BGP. Considere el tráfico que va de A a D. B preferiría manipular dicho tráfico hacia C por el enlace de la Costa Oeste (de modo que C tendría que absorber el coste de transportar el tráfico a través del país), mientras que C preferiría obtener ese tráfico a través del enlace con B de la Costa Este, en cuyo caso B tendría que transportar el tráfico a través del país. ¿Qué mecanismo BGP podría utilizar C para que B llevara el tráfico de A-a-D al punto de contacto de la Costa Este? Para responder a esta pregunta, tendrá que ahondar en la especificación de BGP.



P17. En la Figura 5.13, considere la información de rutas que llega a las redes terminales (*stub*) W, X e Y. Basándose en la información disponible en W y X, ¿cuáles son sus respectivas imágenes de la topología de la red? Justifique su respuesta. La imagen de la topología en Y se muestra a continuación.



P18. Considere la Figura 5.13. B nunca reenviaría tráfico destinado a Y a través de X basándose en el enrutamiento BGP. Pero existen algunas aplicaciones muy populares en las que los paquetes de datos se dirigen primero a X y luego fluyen hacia Y. Identifique una de tales aplicaciones y describa cómo los paquetes de datos siguen una ruta que no ha sido determinada por el enrutamiento BGP.

P19. En la Figura 5.13, suponga que existe otra red terminal V que es un cliente del ISP A. Suponga que B y C tienen una relación de pares y que A es cliente tanto de B como de C. Suponga también que A preferiría que el tráfico destinado a W procediera solo de B, y que el tráfico destinado a V procediera de B o de C. ¿Cómo podría anunciar A sus rutas a B y C? ¿Qué rutas del sistema autónomo recibe C?

P20. Suponga que los sistemas autónomos X y Z no están directamente conectados sino que se conectan a través del sistema autónomo Y. Suponga también que X tiene un acuerdo de comunicación entre pares con Y, y que Y tiene un acuerdo similar con Z. Por último, suponga

que Z desea transferir todo el tráfico de Y pero no el de X. ¿Permite BGP implementar esta política a Z?

P21. Considere las dos formas en que tiene lugar la comunicación entre una entidad gestora y un dispositivo gestionado: el modo solicitud-respuesta y TRAP. ¿Cuáles son las ventajas y los inconvenientes de estos dos métodos, en términos de (1) costes, (2) tiempo de notificación cuando se producen sucesos excepcionales y (3) robustez con respecto a los mensajes perdidos entre la entidad gestora y el dispositivo?

P22. En la Sección 5.7 hemos visto que era preferible transportar mensajes SNMP en datagramas UDP no fiables. ¿Por qué cree que los diseñadores de SNMP eligieron UDP en lugar de TCP como protocolo de transporte para SNMP?

Tarea de programación con socket

Al final del Capítulo 2 hay cuatro tareas de programación con sockets. A continuación le proponemos una quinta tarea, que utiliza ICMP, un protocolo del que hemos hablado en este capítulo.

Tarea 5: Ping ICMP

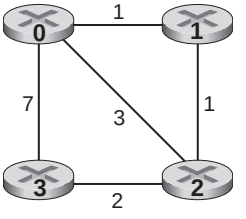
Ping es una popular aplicación de red que se utiliza para comprobar, desde una ubicación remota, si un determinado host está activo y es alcanzable. También se utiliza a menudo para medir la latencia entre el host cliente y el host de destino. Funciona enviando paquetes ICMP de "solicitud de eco" (es decir, paquetes ping) al host de destino y esperando a recibir paquetes ICMP de "respuesta de eco" (es decir, paquetes pong). Ping mide el RTT, contabiliza las pérdidas de paquete y calcula un resumen estadístico de múltiples intercambios ping-pong (el mínimo, la media, el máximo y la desviación estándar del tiempo de ida y vuelta).

En esta práctica de laboratorio tendrá que escribir su propia aplicación Ping en Python. La aplicación utilizará ICMP. Pero, para que el programa sea sencillo, no vamos a seguir exactamente la especificación oficial de RFC 1739. Observe que solo necesita escribir el lado del cliente del programa, ya que la funcionalidad necesaria en el lado del servidor está integrada en casi todos los sistemas operativos. Puede encontrar los detalles completos de esta tarea, así como fragmentos significativos del código Python, en el sitio web http://www.pearsonhighered.com/cs-resources.

Tarea de programación

En esta tarea de programación tendrá que escribir un conjunto "distribuido" de procedimientos que implemente un enrutamiento asíncrono distribuido por vector de distancias para la red que se muestra más abajo.

Tendrá que escribir las siguientes rutinas que se "ejecutarán" de forma asíncrona dentro del entorno emulado proporcionado para esta tarea. Para el nodo 0, escribirá las rutinas siguientes:



- *rtinit0()*. Se llamará a esta rutina una vez al principio de la emulación. *rtinit0()* no tiene argumentos. Debe inicializar su tabla de distancias en el nodo 0 para reflejar los costes directos

de 1, 3 y 7 a los nodos 1, 2 y 3, respectivamente. En la figura de encima, todos los enlaces son bidireccionales y los costes en ambas direcciones son idénticos. Después de inicializar la tabla de distancias y cualquier otra estructura de datos que necesiten sus rutinas del nodo 0, deberá entonces enviar a sus vecinos directamente conectados (en este caso, 1, 2 y 3) el coste de sus rutas de coste mínimo a los demás nodos de la red. La información de coste mínimo se envía a los nodos vecinos mediante un paquete de actualización de enrutamiento llamando a la rutina *tolayer2()*, como se describe en la tarea completa. El formato del paquete de actualización de enrutamiento también está descrito en la tarea completa.

- *rtupdate0(struct rtpkt *rcvdpkt)*. Se llamará a esta rutina cuando el nodo 0 reciba un paquete de enrutamiento que le haya enviado uno de sus vecinos directamente conectados. El parámetro **rcvdpkt* es un puntero al paquete que ha recibido. *rtupdate0()* es el "núcleo" del algoritmo de vector de distancias. Los valores recibidos en un paquete de actualización de enrutamiento procedente de algún otro nodo *i* contienen los costes de la ruta más corta actual de *i* hacia todos los demás nodos de la red. *rtupdate0()* utiliza estos valores para actualizar su propia tabla de distancias (como especifica el algoritmo de vector de distancias). Si su propio coste mínimo a otro nodo cambia como resultado de la actualización, el nodo 0 informa de este cambio del coste mínimo a sus vecinos directamente conectados enviándoles un paquete de enrutamiento. Recuerde que, en el algoritmo de vector de distancias, solo los nodos directamente conectados intercambiarán paquetes de enrutamiento. Por tanto, los nodos 1 y 2 se comunicarán entre sí, pero los nodos 1 y 3 no lo harán.

Para los nodos 1, 2 y 3 se definen rutinas similares. Por tanto, tendrá que escribir ocho procedimientos en total: *rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2()* y *rtupdate3()*. Estas rutinas implementarán conjuntamente un cálculo asíncrono y distribuido de las tablas de distancias para la topología y los costes mostrados en la figura anterior.

Puede encontrar todos los detalles acerca de esta tarea de programación, así como el código C que tendrá que crear en el entorno hardware/software simulado en http://www.pearsonhighered.com/cs-resource. También hay disponible una versión Java de la tarea.

Práctica de laboratorio con Wireshark

En el sitio web del libro, www.pearsonhighered.com/cs-resources, encontrará una práctica de laboratorio con Wireshark que examina el uso del protocolo ICMP en los comandos ping y traceroute.

UNA ENTREVISTA CON...

Jennifer Rexford

Jennifer Rexford es profesora en el departamento de Ciencias de la Computación de la Universidad de Princeton. Sus investigaciones se centran en conseguir que las redes de computadoras sean más fáciles de diseñar y gestionar, con un énfasis especial en los protocolos de enrutamiento. Entre 1996 y 2004 formó parte del departamento de Rendimiento y Gestión de Red de AT&T Labs–Research. Mientras trabajaba en AT&T diseñó técnicas y herramientas para la realización de medidas de red, ingeniería de tráfico y configuración de routers, que fueron implantadas en la red troncal de AT&T. Jennifer es coautora del libro "Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement," publicado por Addison-Wesley en mayo de 2001. Ha sido la moderadora de ACM SIGCOMM entre 2003 y 2007. Se graduó en Ingeniería Eléctrica en la Universidad de Princeton en 1991, doctorándose en Ingeniería Eléctrica y Ciencias de la Computación en la Universidad de Michigan en1996. En 2004, Jennifer fue galardonada con el premio Grace Murray Hopper de la ACM por su excelencia como joven profesional en el campo de las Ciencias de la Computación y fue incluida en la lista TR-100 del MIT de principales innovadores de menos de 35 años.

¿Podría describirnos uno o dos de los proyectos más excitantes en los que haya trabajado durante su carrera? ¿Cuáles fueron los principales desafíos de diseño?

Cuando trabajaba como investigadora en AT&T, varios de nosotros diseñamos una nueva forma de gestionar el enrutamiento en las redes troncales de los ISP. Tradicionalmente, los operadores de red configuran cada router individualmente, y estos routers ejecutan protocolos distribuidos para calcular las rutas a través de la red. Estábamos convencidos de que la gestión de red sería más simple y flexible si los operadores de red pudieran ejercer un control directo sobre el modo en que los routers reenvían el tráfico, basándose en una visión global de la topología y el tráfico de toda la red. La plataforma de control de enrutamiento RCP (*Routing Control Platform*) que diseñamos y construimos podía calcular las rutas para toda la red troncal de AT&T usando una única computadora barata y permitía controlar los routers heredados sin necesidad de modificaciones. Para mi, este proyecto fue muy excitante, porque tuvimos una idea provocadora, desarrollamos un sistema funcional y, al final, realizamos una implantación real en una red que estaba en operación. Si saltamos adelante unos cuantos años, vemos que las redes definidas por software (SDN) se han convertido en una tecnología dominante, y existen protocolos estándar (como OpenFlow) que facilitan mucho la tarea de decir a los conmutadores subyacentes qué tienen que hacer.

¿Cómo cree que deberían evolucionar en el futuro las redes definidas por software?

En una ruptura radical con respecto al pasado, el software del plano de control puede ser creado por muchos programadores diferentes, no solo por las empresas que desarrollan equipos de red. Sin embargo, a diferencia de las aplicaciones que se ejecutan en un servidor o en un teléfono inteligente, esas aplicaciones controladoras deben trabajar *juntas* para gestionar el mismo tráfico. Los operadores de red no desean realizar un equilibrado de carga con una parte del tráfico y un enrutamiento con otra parte del tráfico, sino que quieren realizar tanto el equilibrado de carga como el enrutamiento de un mismo tráfico. Las futuras plataformas controladoras

SDN deberían ofrecer buenas abstracciones de programación para *componer* múltiples aplicaciones controladoras escritas independientemente, de forma que trabajen juntas. Hablando en términos más generales, las buenas abstracciones de programación pueden hacer que resulte más fácil crear aplicaciones controladoras, sin tener que preocuparse de detalles de bajo nivel como las entradas de las tablas de flujo, los contadores de tráfico, los patrones de bits en las cabeceras de los paquetes, etc. Asimismo, aunque un controlador SDN está lógicamente centralizado, la red sigue consistiendo en una colección distribuida de dispositivos. Los controladores futuros deberían ofrecer buenas abstracciones para actualizar las tablas de flujo en toda la red, de modo que las aplicaciones puedan razonar acerca de lo que les sucede a los paquetes en tránsito mientras los dispositivos están siendo actualizados. La de las abstracciones de programación para el software del plano de control es un área excitante de investigación interdisciplinar entre las redes de computadoras, los sistemas distribuidos y los lenguajes de programación, con una posibilidad real de tener un impacto práctico en los años venideros.

¿Cuál cree que es el futuro de las redes y de Internet?

El de las redes es un campo excitante, porque las aplicaciones y las tecnologías subyacentes están cambiando constantemente. ¡Continuamente nos estamos reinventando a nosotros mismos! ¿Quién habría predicho hace solo diez años el predominio de los teléfonos inteligentes, que permiten a los usuarios móviles acceder tanto a las aplicaciones existentes, como a nuevos servicios basados en la ubicación? La aparición de la commputación en la nube está cambiando de modo fundamental la relación entre los usuarios y la aplicación que ejecutan, y los sensores y actuadores en red (la "Internet de las cosas") están haciendo posible una plétora de nuevas aplicaciones (¡y de vulnerabilidades de seguridad!). El ritmo de innovación es verdaderamente fascinante.

La red subyacente es un componente crucial de todas estas innovaciones. Y, sin embargo, la red constituye un notorio "obstáculo": limitando el rendimiento, comprometiendo la fiabilidad, restringiendo las aplicaciones y complicando la implantación y la gestión de servicios. Deberíamos intentar que la red del futuro sea tan invisible como el aire que respiramos, de modo que nunca obstaculice las nuevas ideas y los servicios valiosos. Para ello, necesitamos elevar el nivel de abstracción por encima de los protocolos y dispositivos de red individuales (¡y sus respectivos acrónimos!), de modo que podamos razonar acerca de la red y de los objetivos de alto nivel del usuario en su conjunto.

¿Qué personas le han inspirado profesionalmente?

Me he sentido inspirada durante mucho tiempo por Sally Floyd, del International Computer Science Institute. Sus investigaciones siempre eran pertinentes, centradas en los importantes desafíos a los que se enfrenta Internet. Profundizaba enormemente en cuestiones difíciles, hasta que entendía completamente el problema y el espacio de soluciones, y se dedicaba en cuerpo y alma a hacer que "las cosas avanzaran", como por ejemplo introduciendo sus ideas en estándares de protocolo y equipos de red. Asimismo, contribuía a la comunidad, prestando servicios profesionales en numerosas organizaciones de investigación y de estandarización y creando herramientas (como los ampliamente utilizados simuladores ns-2 y ns-3) que permiten a otros investigadores avanzar. Se jubiló en 2009, pero su influencia se dejará sentir en este campo durante muchos años.

¿Qué recomendaría a los estudiantes que quieran desarrollar una carrera profesional en el campo de las ciencias de la computación y las redes?

El de las redes es un campo inherentemente interdisciplinar. A las redes se les aplican técnicas derivadas de los avances en otras muchas disciplinas, de áreas tan diversas como la teoría de colas, la teoría de juegos, la teoría de control, los sistemas distribuidos, la optimización de redes, los lenguajes de programación, el aprendizaje de máquinas, los algoritmos, las estructuras de datos, etc. Creo que ser experto en un campo relacionado, o colaborar estrechamente con expertos en esos campos, es una excelente manera de reforzar los fundamentos de las redes,

para aprender a construir redes que sean merecedoras de la confianza de la sociedad. Más allá de las disciplinas teóricas, las redes son excitantes porque creamos sistemas reales que son utilizados por la gente real. Dominar la manera de diseñar y construir sistemas —obteniendo experiencia en sistemas operativos, arquitectura de computadoras, etc.— es otra fantástica manera de ampliar nuestros conocimientos de redes para ayudar a hacer del mundo un lugar mejor.