detalladamente cómo funciona. Para explorar un puerto TCP específico, por ejemplo, el puerto 6789, en un host objetivo, nmap enviará un segmento TCP SYN con el puerto de destino 6789 a dicho host. Pueden obtenerse entonces tres posibles resultados:

- *El host de origen recibe un segmento TCP SYNACK del host objetivo*. Dado que esto significa que hay una aplicación ejecutándose con TCP en el puerto 6789 del host objetivo, nmap devuelve "open" (puerto abierto).
- *El host de origen recibe un segmento TCP RST procedente del host objetivo*. Esto significa que el segmento SYN ha alcanzado el host objetivo, pero este no está ejecutando una aplicación con TCP en el puerto 6789. Pero el atacante sabrá como mínimo que el segmento destinado al puerto 6789 del host no está bloqueado por ningún cortafuegos existente en la ruta entre los hosts de origen y de destino. (Los cortafuegos se estudian en el Capítulo 8.)
- *El origen no recibe nada*. Probablemente, esto significa que el segmento SYN fue bloqueado por un cortafuegos intermedio y nunca llegó al host objetivo.

nmap es una potente herramienta que puede detectar "las brechas en el muro", no solo en lo relativo a los puertos TCP abiertos, sino también a los puertos UDP abiertos, a los cortafuegos y sus configuraciones e incluso a las versiones de aplicaciones y sistemas operativos. La mayor parte de esta tarea se lleva a cabo manipulando los segmentos de gestión de la conexión TCP [Skoudis 2006]. Puede descargar nmap en la dirección www.nmap.org.

Con esto completamos nuestra introducción a los mecanismos de control de errores y de control de flujo de TCP. En la Sección 3.7, volveremos sobre TCP y estudiaremos más detalladamente el control de congestión de TCP. Sin embargo, antes de eso, vamos a dar un paso atrás y vamos a examinar los problemas de control de congestión en un contexto más amplio.

3.6 Principios del control de congestión

En la sección anterior hemos examinado tanto los principios generales como los específicos de los mecanismos de TCP utilizados para proporcionar un servicio de transferencia de datos fiable en lo que se refiere a la pérdida de paquetes. Anteriormente habíamos mencionado que, en la práctica, tales pérdidas son normalmente el resultado de un desbordamiento de los buffers de los routers a medida que la red se va congestionando. La retransmisión de paquetes por tanto se ocupa de un síntoma de la congestión de red (la pérdida de un segmento específico de la capa de transporte) pero no se ocupa de la causa de esa congestión de la red (demasiados emisores intentando transmitir datos a una velocidad demasiado alta). Para tratar la causa de la congestión de la red son necesarios mecanismos que regulen el flujo de los emisores en cuanto la congestión de red aparezca.

En esta sección consideraremos el problema del control de congestión en un contexto general, con el fin de comprender por qué la congestión es algo negativo, cómo la congestión de la red se manifiesta en el rendimiento ofrecido a las aplicaciones de la capa superior y cómo pueden aplicarse diversos métodos para evitar la congestión de la red o reaccionar ante la misma. Este estudio de carácter general del control de la congestión es apropiado porque, puesto que junto con la transferencia de datos fiable, se encuentra al principio de la lista de los diez problemas más importantes de las redes. En la siguiente sección se lleva a cabo un estudio detallado sobre el algoritmo de control de congestión de TCP.

3.6.1 Las causas y los costes de la congestión

Iniciemos este estudio de carácter general sobre el control de congestión examinando tres escenarios con una complejidad creciente en los que se produce congestión. En cada caso, veremos en primer lugar por qué se produce la congestión y el coste de la misma (en términos de recursos no utilizados

por completo y del bajo rendimiento ofrecido a los sistemas terminales). Todavía no vamos a entrar a ver cómo reaccionar ante una congestión, o cómo evitarla, simplemente vamos a poner el foco sobre la cuestión más simple de comprender: qué ocurre cuando los hosts incrementan su velocidad de transmisión y la red comienza a congestionarse.

Escenario 1: dos emisores, un router con buffers de capacidad ilimitada

Veamos el escenario de congestión más simple posible: dos hosts (A y B), cada uno de los cuales dispone de una conexión que comparte un único salto entre el origen y el destino, como se muestra en la Figura 3.43.

Supongamos que la aplicación del host A está enviando datos a la conexión (por ejemplo, está pasando datos al protocolo de la capa de transporte a través de un socket) a una velocidad media de λ_{in} bytes/segundo. Estos datos son originales en el sentido de que cada unidad de datos se envía solo una vez al socket. El protocolo del nivel de transporte subyacente es un protocolo simple. Los datos se encapsulan y se envían; no existe un mecanismo de recuperación de errores (como por ejemplo, las retransmisiones), ni se realiza un control de flujo ni un control de congestión. Ignorando la sobrecarga adicional debida a la adición de la información de cabecera de las capas de transporte e inferiores, la velocidad a la que el host A entrega tráfico al router en este primer escenario es por tanto λ_{in} bytes/segundo. El host B opera de forma similar, y suponemos, con el fin de simplificar, que también está enviando datos a una velocidad de λ_{in} bytes/segundo. Los paquetes que salen de los hosts A y B atraviesan un router y un enlace de salida compartido de capacidad *R*. El router tiene buffers que le permiten almacenar paquetes entrantes cuando la tasa de llegada de paquetes excede la capacidad del enlace de salida. En este primer escenario, suponemos que el router tiene un buffer con una cantidad de espacio infinita.

La Figura 3.44 muestra el rendimiento de la conexión del host A en este primer escenario. La gráfica de la izquierda muestra **la tasa de transferencia por conexión** (número de bytes por segundo en el receptor) como una función de la velocidad de transmisión de la conexión. Para una velocidad de transmisión comprendida entre 0 y *R*/2, la tasa de transferencia en el receptor es igual a la velocidad de transmisión en el emisor (todo lo que envía el emisor es recibido en el receptor con un retardo finito). Sin embargo, cuando la velocidad de transmisión es mayor que *R*/2, la tasa de transferencia es de solo *R*/2. Este límite superior de la tasa de transferencia es una consecuencia de compartir entre dos conexiones la capacidad del enlace. El enlace simplemente no puede proporcionar paquetes a un receptor a una velocidad de régimen permanente que sea mayor que *R*/2. Independientemente de lo altas que sean las velocidades de transmisión de los hosts A y B, nunca verán una tasa de transferencia mayor que *R*/2.

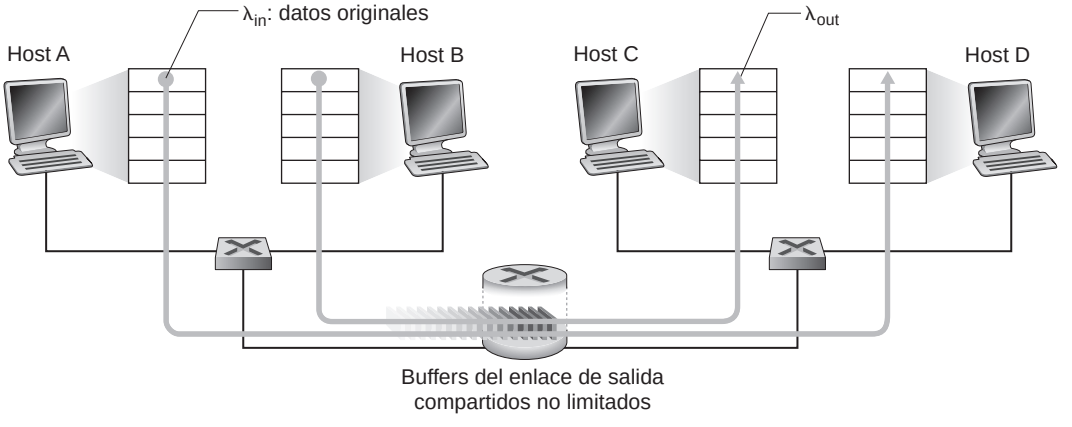

Figura 3.43 ♦ Escenario de congestión 1: dos conexiones que comparten un único salto con buffers de capacidad ilimitada.

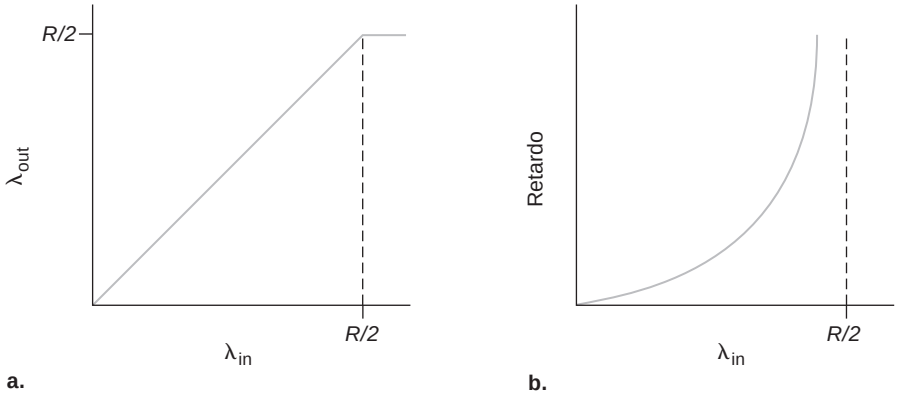

Figura 3.44 ♦ Escenario de congestión 1: tasa de transferencia y retardo en función de la velocidad de transmisión del host.

Alcanzar una tasa de transferencia por conexión de *R*/2 podría realmente parecer algo positivo, porque el enlace se utiliza completamente en suministrar paquetes a sus destinos. Sin embargo, la gráfica de la derecha de la Figura 3.44 muestra la consecuencia de operar cerca de la capacidad del enlace. A medida que la velocidad de transmisión se aproxima a *R*/2 (desde la izquierda), el retardo medio se hace cada vez más grande. Cuando la velocidad de transmisión excede de *R*/2, el número medio de paquetes en cola en el router no está limitado y el retardo medio entre el origen y el destino se hace infinito (suponiendo que las conexiones operan a esas velocidades de transmisión durante un periodo de tiempo infinito y que existe una cantidad infinita de espacio disponible en el buffer). Por tanto, aunque operar a una tasa de transferencia agregada próxima a *R* puede ser ideal desde el punto de vista de la tasa de transferencia, no es ideal en absoluto desde el punto de vista del retardo. *Incluso en este escenario (extremadamente) idealizado, ya hemos encontrado uno de los costes de una red congestionada: los grandes retardos de cola experimentados cuando la tasa de llegada de los paquetes se aproxima a la capacidad del enlace.*

Escenario 2: dos emisores y un router con buffers finitos

Ahora vamos a modificar ligeramente el escenario 1 de dos formas (véase la Figura 3.45). En primer lugar, suponemos que el espacio disponible en los buffers del router es finito. Una consecuencia de esta suposición aplicable en la práctica es que los paquetes serán descartados cuando lleguen a un buffer que ya esté lleno. En segundo lugar, suponemos que cada conexión es fiable. Si un paquete que contiene un segmento de la capa de transporte se descarta en el router, el emisor tendrá que retransmitirlo. Dado que los paquetes pueden retransmitirse, ahora tenemos que ser más cuidadosos al utilizar el término *velocidad de transmisión*. Específicamente, vamos a designar la velocidad a la que la aplicación envía los datos originales al socket como λ_{in} bytes/segundo. La velocidad a la que la capa de transporte envía segmentos (que contienen los datos originales *y* los datos retransmitidos) a la red la denotaremos como λ'_{in} bytes/segundo. En ocasiones, λ'_{in} se denomina **carga ofrecida** a la red.

El rendimiento de este segundo escenario dependerá en gran medida de cómo se realicen las retransmisiones. En primer lugar, considere el caso no realista en el que el host A es capaz de determinar de alguna manera (mágicamente) si el buffer en el router está libre o lleno y enviar entonces un paquete solo cuando el buffer esté libre. En este caso no se produciría ninguna pérdida, λ_{in} sería igual a λ'_{in} y la tasa de transferencia de la conexión sería igual a λ_{in}. Este caso se muestra en la Figura 3.46(a). Desde el punto de vista de la tasa de transferencia, el rendimiento es ideal: todo lo que se envía, se recibe. Observe que, en este escenario, la velocidad media de transmisión de host no puede ser mayor que *R*/2, ya que se supone que nunca tiene lugar una pérdida de paquetes.

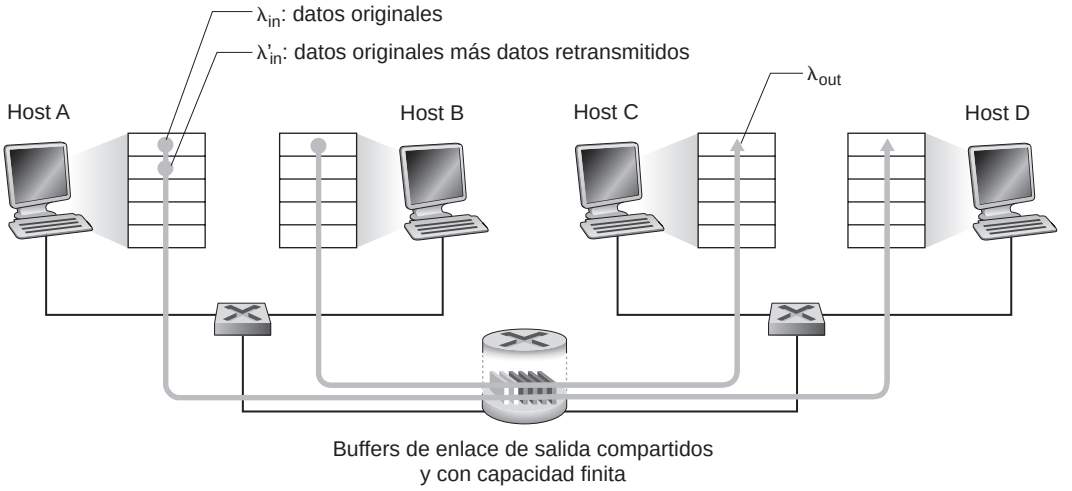

Figura 3.45 ♦ Escenario 2: dos hosts (con retransmisión) y un router con buffers con capacidad finita.

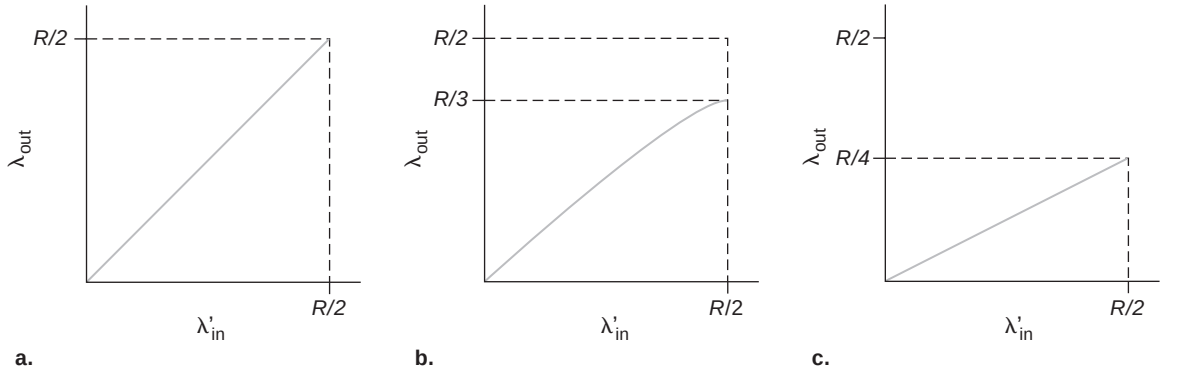

Figura 3.46 ♦ Escenario 2: rendimiento con buffers de capacidad finita.

Consideremos ahora un caso algo más realista en el que el emisor solo retransmite cuando sabe con seguridad que un paquete se ha perdido. De nuevo, esta es una suposición un poco exagerada; sin embargo, es posible que el host emisor tenga fijado su intervalo de fin de temporización en un valor lo suficientemente grande como para garantizar que un paquete que no ha sido reconocido es un paquete que se ha perdido. En este caso, el rendimiento puede ser similar al mostrado en la Figura 3.46(b). Para apreciar lo que está ocurriendo aquí, considere el caso en el que la carga ofrecida, λ'_{in} (la velocidad de transmisión de los datos originales más la de las retransmisiones), es igual a $R/2$. Según la Figura 3.46(b), para este valor de la carga ofrecida la velocidad a la que los datos son suministrados a la aplicación del receptor es $R/3$. Por tanto, de las 0,5R unidades de datos transmitidos, 0,333R bytes/segundo (como media) son datos originales y 0,166R bytes/segundo (como media) son datos retransmitidos. *Tenemos aquí por tanto otro de los costes de una red congestionada: el emisor tiene que realizar retransmisiones para poder compensar los paquetes descartados (perdidos) a causa de un desbordamiento de buffer.*

Por último, considere el caso en el que el emisor puede alcanzar el fin de la temporización de forma prematura y retransmitir un paquete que ha sido retardado en la cola pero que todavía no se ha perdido. En este caso, tanto el paquete de datos original como la retransmisión pueden llegar al receptor. Por supuesto, el receptor necesitará solo una copia de este paquete y descartará

la retransmisión. En este caso, el trabajo realizado por el router al reenviar la copia retransmitida del paquete original se desperdicia, ya que el receptor ya había recibido la copia original de ese paquete. El router podría haber hecho un mejor uso de la capacidad de transmisión del enlace enviando en su lugar un paquete distinto. *Por tanto, tenemos aquí otro de los costes de una red congestionada: las retransmisiones innecesarias del emisor causadas por retardos largos pueden llevar a que un router utilice el ancho de banda del enlace para reenviar copias innecesarias de un paquete.* La Figura 3.46(c) muestra la tasa de transferencia en función de la carga ofrecida cuando se supone que el router reenvía cada paquete dos veces (como media). Dado que cada paquete se reenvía dos veces, la tasa de transferencia tendrá un valor asintótico de *R*/4 cuando la carga ofrecida se aproxime a *R*/2.

Escenario 3: cuatro emisores, routers con buffers de capacidad finita y rutas con múltiples saltos

En este último escenario dedicado a la congestión de red tenemos cuatro hosts que transmiten paquetes a través de rutas solapadas con dos saltos, como se muestra en la Figura 3.47. De nuevo suponemos que cada host utiliza un mecanismo de fin de temporización/retransmisión para implementar un servicio de transferencia de datos fiable, que todos los hosts tienen el mismo valor de λ_{in} y que todos los enlaces de router tienen una capacidad de *R* bytes/segundo.

Consideremos la conexión del host A al host C pasando a través de los routers R1 y R2. La conexión A–C comparte el router R1 con la conexión D–B y comparte el router R2 con la conexión B–D. Para valores extremadamente pequeños de λ_{in}, es raro que el buffer se desborde (como en los dos escenarios de congestión anteriores), y la tasa de transferencia es aproximadamente igual a

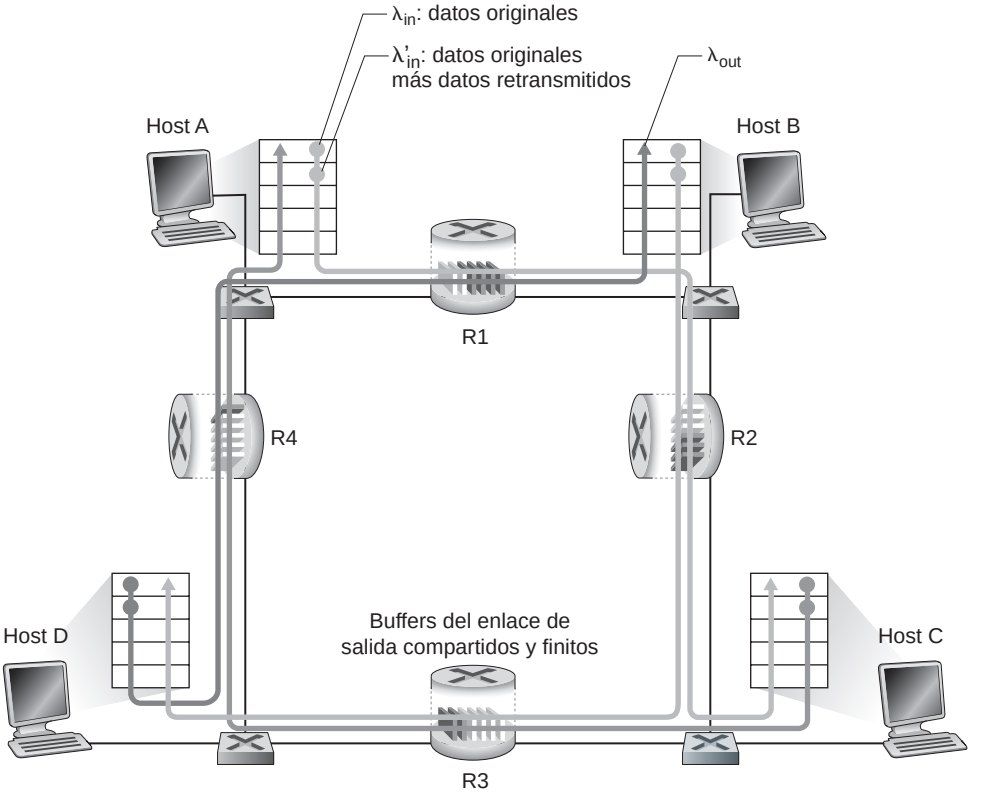

Figura 3.47 ♦ Cuatro emisores, routers con buffers finitos y rutas con varios saltos.

la carga ofrecida. Para valores ligeramente más grandes de λ_{in}, la correspondiente tasa de transferencia es también más grande, ya que se están transmitiendo más datos originales por la red y entregándose en su destino, y los desbordamientos siguen siendo raros. Por tanto, para valores pequeños de λ_{in}, un incremento de λ_{in} da lugar a un incremento de λ_{out}.

Una vez considerado el caso de tráfico extremadamente bajo, pasemos a examinar el caso en que λ_{in} (y por tanto λ'_{in}) es extremadamente grande. Sea el router R2. El tráfico de A–C que llega al router R2 (el que llega a R2 después de ser reenviado desde R1) puede tener una velocidad de llegada a R2 que es como máximo *R*, la capacidad del enlace de R1 a R2, independientemente del valor de λ_{in}. Si λ'_{in} es extremadamente grande en todas las conexiones (incluyendo la conexión B–D), entonces la velocidad de llegada del tráfico de B–D a R2 puede ser mucho mayor que la del tráfico de A–C. Puesto que los tráficos de A–C y B–D tienen que competir en el router R2 por el espacio limitado disponible en el buffer, la cantidad de tráfico de A–C que consiga atravesar con éxito R2 (es decir, que no se pierda por desbordamiento del buffer) será cada vez menor a medida que la carga ofrecida por la conexión B–D aumente. En el límite, a medida que la carga ofrecida se aproxima a infinito, un buffer vacío de R2 es llenado de forma inmediata por un paquete de B–D y la tasa de transferencia de la conexión A–C en R2 tiende a cero. Esto, a su vez, *implica que la tasa de transferencia terminal a terminal de A–C tiende a cero* en el límite correspondiente a una situación de tráfico intenso. Estas consideraciones dan lugar a la relación de compromiso entre la carga ofrecida y la tasa de transferencia que se muestra en la Figura 3.48.

La razón del eventual decrecimiento de la tasa de transferencia al aumentar la carga ofrecida es evidente cuando se considera la cantidad de trabajo desperdiciado realizado por la red. En el escenario descrito anteriormente en el que había una gran cantidad de tráfico, cuando un paquete se descartaba en un router de segundo salto, el trabajo realizado por el router del primer salto al encaminar un paquete al router del segundo salto terminaba siendo "desperdiciado". Para eso, hubiera dado igual que el primer router simplemente hubiera descartado dicho paquete y hubiera permanecido inactivo, porque de todos modos el paquete no llegaría a su destino. Aún más, la capacidad de transmisión utilizada en el primer router para reenviar el paquete al segundo router podría haber sido mejor aprovechada si se hubiera empleado para transmitir un paquete diferente (por ejemplo, al seleccionar un paquete para transmitirlo, puede resultar mejor para un router dar la prioridad a los paquetes que ya hayan pasado por varios routers anteriormente). *Por tanto, aquí nos encontramos con otro de los costes de descartar un paquete a causa de la congestión de la red: cuando un paquete se descarta a lo largo de una ruta, la capacidad de transmisión empleada en cada uno de los enlaces anteriores para encaminar dicho paquete hasta el punto en el que se ha descartado termina por desperdiciarse.*

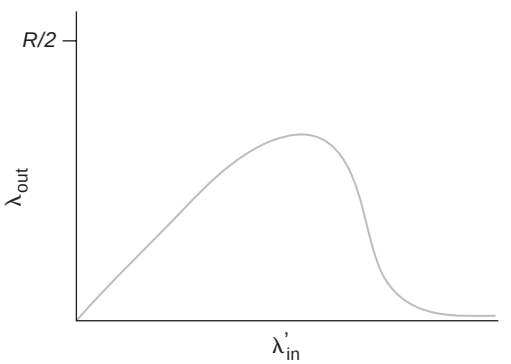

Figura 3.48 ♦ Escenario 3: rendimiento con buffers finitos y rutas con varios saltos.

3.6.2 Métodos para controlar la congestión

En la Sección 3.7 examinaremos en gran detalle el método específico de TCP para controlar la congestión. Ahora, vamos a identificar los dos métodos más comunes de control de congestión que se utilizan en la práctica y abordaremos las arquitecturas de red específicas y los protocolos de control de congestión que se integran en estos métodos.

En el nivel más general, podemos diferenciar entre las técnicas de control de congestión basándonos en si la capa de red proporciona alguna ayuda explícita a la capa de transporte con propósitos de controlar la congestión:

- *Control de congestión terminal a terminal*. En este método, la capa de red no proporciona soporte explícito a la capa de transporte para propósitos de control de congestión. Incluso la presencia de congestión en la red tiene que ser inferida por los sistemas terminales basándose únicamente en el comportamiento observado de la red (por ejemplo, la pérdida de paquetes y los retardos). En la Sección 3.7.1 veremos que TCP tiene que aplicar este método de control de congestión terminal a terminal, ya que la capa IP no proporciona ninguna realimentación a los hosts relativa a la congestión de la red. La pérdida de segmentos TCP (indicada por un fin de temporización o por la recepción de un triple paquete ACK duplicado) se toma como indicación de que existe congestión en la red, por lo que TCP reduce el tamaño de su ventana en consecuencia. También veremos una propuesta más reciente para abordar el control de congestión de TCP que utiliza valores de retardo de ida y vuelta crecientes como indicadores de que existe una mayor congestión de red.
- *Control de congestión asistido por la red*. En este método de control de congestión, los routers proporcionan una realimentación explícita al emisor y/o receptor informando del estado de congestión en la red. Esta realimentación puede ser tan simple como un único bit que indica que existe congestión en un enlace. Este método se aplicó en las tempranas arquitecturas de red SNA de IBM [Schwartz 1982] y DECnet de DEC [Jain 1989; Ramakrishnan 1990] y ATM [Black 1995]. También es posible proporcionar una realimentación de red más sofisticada. Por ejemplo, una forma del mecanismo de control de congestión de **ABR (*Available Bite Rate*)** en las redes **ATM** permite a un router informar al emisor de la velocidad máxima de transmisión de host que el router puede soportar en un enlace saliente. Como hemos mencionado anteriormente, las versiones predeterminadas de Internet de IP y TCP adoptan un método terminal a terminal para llevar a cabo el control de congestión. Sin embargo, veremos en la Sección 3.7.2 que, recientemente, IP y TCP pueden implementar de forma opcional un mecanismo de control de congestión asistido por la red.

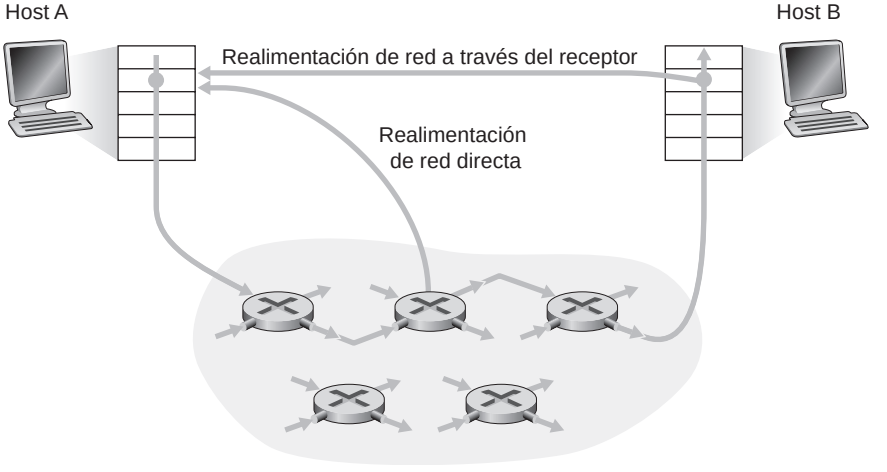

Figura 3.49 ♦ Dos rutas de realimentación de la información de la congestión asistida por la red.

En el mecanismo de control de congestión asistido por la red, la información acerca de la congestión suele ser realimentada de la red al emisor de una de dos formas, como se ve en la Figura 3.49. La realimentación directa puede hacerse desde un router de la red al emisor. Esta forma de notificación, normalmente, toma la forma de un paquete de asfixia o bloqueo (*choke packet*) (que esencialmente dice "¡Estoy congestionada!"). La segunda forma de notificación, más común, tiene lugar cuando un router marca/actualiza un campo de un paquete que se transmite del emisor al receptor para indicar que existe congestión. Después de recibir un paquete marcado, el receptor notifica al emisor la existencia de congestión. Observe que esta última forma de notificación tarda al menos un periodo igual al tiempo de ida y vuelta completo.

3.7 Mecanismo de control de congestión de TCP

En esta sección vamos a continuar con nuestro estudio de TCP. Como hemos visto en la Sección 3.5, TCP proporciona un servicio de transporte fiable entre dos procesos que se ejecutan en hosts diferentes. Otro componente clave de TCP es su mecanismo de control de congestión. Como hemos mencionado en la sección anterior, TCP tiene que utilizar un control de congestión terminal a terminal en lugar de un control de congestión asistido por la red, ya que la capa IP no proporciona una realimentación explícita a los sistemas terminales en lo tocante a la congestión de la red.

El método empleado por TCP consiste en que cada emisor limite la velocidad a la que transmite el tráfico a través de su conexión en función de la congestión de red percibida. Si un emisor TCP percibe que en la ruta entre él mismo y el destino apenas existe congestión, entonces incrementará su velocidad de transmisión; por el contrario, si el emisor percibe que existe congestión a lo largo de la ruta, entonces reducirá su velocidad de transmisión. Pero este método plantea tres cuestiones. En primer lugar, ¿cómo limita el emisor TCP la velocidad a la que envía el tráfico a través de su conexión? En segundo lugar, ¿cómo percibe el emisor TCP que existe congestión en la ruta entre él mismo y el destino? Y, tercero, ¿qué algoritmo deberá emplear el emisor para variar su velocidad de transmisión en función de la congestión percibida terminal a terminal?

Examinemos en primer lugar cómo un emisor TCP limita la velocidad a la que envía el tráfico a través de su conexión. En la Sección 3.5, hemos visto que cada lado de una conexión TCP consta de un buffer de recepción, un buffer de transmisión y varias variables (`UltimoByteLeido`, `VentRecepcion`, etc.). El mecanismo de control de congestión de TCP que opera en el emisor hace un seguimiento de una variable adicional, la **ventana de congestión**. Esta ventana, indicada como `VentCongestion`, impone una restricción sobre la velocidad a la que el emisor TCP puede enviar tráfico a la red. Específicamente, la cantidad de datos no reconocidos en un emisor no puede exceder el mínimo de entre `VentCongestion` y `VentRecepcion`, es decir:

```
UltimoByteEnviado – UltimoByteReconocido ≤ min{VentCongestion, VentRecepcion}
```

Con el fin de centrarnos en el mecanismo de control de congestión (en oposición al control de flujo), vamos a suponer que el buffer de recepción TCP es tan grande que la restricción de la ventana de recepción puede ignorarse; por tanto, la cantidad de datos no reconocidos por el emisor queda limitada únicamente por `VentCongestion`. Supondremos también que el emisor siempre tiene datos que enviar; es decir, todos los segmentos de la ventana de congestión son enviados.

La restricción anterior limita la cantidad de datos no reconocidos por el emisor y, por tanto, limita de manera indirecta la velocidad de transmisión del emisor. Para comprender esto imagine una conexión en la que tanto la pérdida de paquetes como los retardos de transmisión sean despreciables. En esta situación, lo que ocurre a grandes rasgos es lo siguiente: al inicio de cada periodo RTT, la restricción permite al emisor enviar `VentCongestion` bytes de datos a través de la conexión; al final del periodo RTT, el emisor recibe los paquetes ACK correspondientes a los datos. *Por tanto, la velocidad de transmisión del emisor es aproximadamente igual a VentCongestion/RTT bytes/*

segundo. Ajustando el valor de la ventana de congestión, el emisor puede ajustar la velocidad a la que transmite los datos a través de su conexión.

Veamos ahora cómo percibe un emisor TCP que existe congestión en la ruta entre él mismo y el destino. Definamos un "suceso de pérdida" en un emisor TCP como el hecho de que se produzca un fin de temporización o se reciban tres paquetes ACK duplicados procedentes del receptor (recuerde la exposición de la Sección 3.5.4 sobre el suceso de fin de temporización mostrado en la Figura 3.33 y la subsiguiente modificación para incluir la retransmisión rápida a causa de la recepción de tres paquetes ACK duplicados). Cuando existe una congestión severa, entonces uno o más de los buffers de los routers existentes a lo largo de la ruta pueden desbordarse, dando lugar a que un datagrama (que contenga un segmento TCP) sea descartado. A su vez, el datagrama descartado da lugar a un suceso de pérdida en el emisor (debido a un fin de temporización o a la recepción de tres paquetes ACK duplicados), el cual lo interpreta como una indicación de que existe congestión en la ruta entre el emisor y el receptor.

Ahora que ya hemos visto cómo se detecta la existencia de congestión en la red, vamos a considerar el mejor de los casos, cuando no existe congestión en la red, es decir, cuando no se producen pérdidas de paquetes. En este caso, el emisor TCP recibirá los paquetes de reconocimiento ACK correspondientes a los segmentos anteriormente no reconocidos. Como veremos, TCP interpretará la llegada de estos paquetes ACK como una indicación de que todo está bien (es decir, que los segmentos que están siendo transmitidos a través de la red están siendo entregados correctamente al destino) y empleará esos paquetes de reconocimiento para incrementar el tamaño de la ventana de congestión (y, por tanto, la velocidad de transmisión). Observe que si la velocidad de llegada de los paquetes ACK es lenta, porque por ejemplo la ruta terminal a terminal presenta un retardo grande o contiene un enlace con un ancho de banda pequeño, entonces el tamaño de la ventana de congestión se incrementará a una velocidad relativamente lenta. Por el contrario, si la velocidad de llegada de los paquetes de reconocimiento es alta, entonces el tamaño de la ventana de congestión se incrementará más rápidamente. Puesto que TCP utiliza los paquetes de reconocimiento para provocar (o temporizar) sus incrementos del tamaño de la ventana de congestión, se dice que TCP es **auto-temporizado**.

Conocido el *mecanismo* de ajuste del valor de `VentCongestion` para controlar la velocidad de transmisión, la cuestión crítica que nos queda es: ¿*cómo* debería un emisor TCP determinar su velocidad de transmisión? Si los emisores TCP de forma colectiva transmiten a velocidades demasiado altas pueden congestionar la red, llevándola al tipo de colapso de congestión que hemos visto en la Figura 3.48. De hecho, la versión de TCP que vamos a estudiar a continuación fue desarrollada en respuesta al colapso de congestión observado en Internet [Jacobson 1988] en las versiones anteriores de TCP. Sin embargo, si los emisores TCP son demasiado cautelosos y transmiten la información muy lentamente, podrían infrautilizar el ancho de banda de la red; es decir, los emisores TCP podrían transmitir a velocidades más altas sin llegar a congestionar la red. Entonces, ¿cómo pueden determinar los emisores TCP sus velocidades de transmisión de manera que no congestionen la red a la vez que hacen uso del todo el ancho de banda disponible? ¿Están los emisores TCP explícitamente coordinados, o existe un método distribuido en el que dichos emisores TCP puedan establecer sus velocidades de transmisión basándose únicamente en la información local? TCP responde a estas preguntas basándose en los siguientes principios:

- *Un segmento perdido implica congestión y, por tanto, la velocidad del emisor TCP debe reducirse cuando se pierde un segmento.* Recuerde que en la Sección 3.5.4 hemos visto que un suceso de fin de temporización o la recepción de cuatro paquetes de reconocimiento para un segmento dado (el paquete ACK original y los tres duplicados) se interpreta como una indicación implícita de que se ha producido un "suceso de pérdida" del segmento que sigue al segmento que ha sido reconocido cuatro veces, activando el proceso de retransmisión del segmento perdido. Desde el punto de vista del mecanismo de control de congestión, la cuestión es cómo el emisor TCP debe disminuir el tamaño de su ventana de congestión y, por tanto, su velocidad de transmisión, en respuesta a este suceso de pérdida inferido.

- *Un segmento que ha sido reconocido indica que la red está entregando los segmentos del emisor al receptor y, por tanto, la velocidad de transmisión del emisor puede incrementarse cuando llega un paquete ACK correspondiente a un segmento que todavía no había sido reconocido.* La llegada de paquetes de reconocimiento se interpreta como una indicación implícita de que todo funciona bien (los segmentos están siendo entregados correctamente del emisor al receptor y la red por tanto no está congestionada). Luego se puede aumentar el tamaño de la ventana de congestión.
- *Tanteo del ancho de banda*. Puesto que los paquetes ACK indican que la ruta entre el origen y el destino está libre de congestión y la pérdida de paquetes indica que hay una ruta congestionada, la estrategia de TCP para ajustar su velocidad de transmisión consiste en incrementar su velocidad en respuesta a la llegada de paquetes ACK hasta que se produce una pérdida, momento en el que reduce la velocidad de transmisión. El emisor TCP incrementa entonces su velocidad de transmisión para tantear la velocidad a la que de nuevo aparece congestión, retrocede a partir de ese punto y comienza de nuevo a tantear para ver si ha variado la velocidad a la que comienza de nuevo a producirse congestión. El comportamiento del emisor TCP es quizá similar a la del niño que pide (y consigue) una y otra vez golosinas hasta que se le dice "¡No!", momento en el que da marcha atrás, pero enseguida comienza otra vez a pedir más golosinas. Observe que la red no proporciona una indicación explícita del estado de congestión (los paquetes ACK y las pérdidas se utilizan como indicadores implícitos) y que cada emisor TCP reacciona a la información local de forma asíncrona con respecto a otros emisores TCP.

Ahora que ya conocemos los fundamentos del mecanismo de control de congestión de TCP, estamos en condiciones de pasar a estudiar los detalles del famoso **algoritmo de control de congestión de TCP**, que fue descrito por primera vez en el libro de [Jacobson 1988] y que está estandarizado en el documento [RFC 5681]. El algoritmo consta de tres componentes principales: (1) arranque lento (*slow start*), (2) evitación de la congestión (*congestion avoidance*) y (3) recuperación rápida (*fast recovery*). Los dos primeros componentes son obligatorios en TCP, diferenciándose en la forma en que aumentan el tamaño de la ventana de congestión en respuesta a los paquetes ACK recibidos. Vamos a ver brevemente que el arranque lento incrementa el tamaño de la ventana de congestión más rápidamente (¡contrariamente a lo que indica su nombre!) que la evitación de la congestión. El componente recuperación rápida es recomendable, aunque no obligatorio, para los emisores TCP.

Arranque lento

Cuando se inicia una conexión TCP, el valor de la ventana de congestión (`VentCongestion`) normalmente se inicializa con un valor pequeño igual a 1 MSS (tamaño máximo de segmento) [RFC 3390], que da como resultado una velocidad de transmisión inicial aproximadamente igual a MSS/ RTT. Por ejemplo, si MSS = 500 bytes y RTT = 200 milisegundos, la velocidad de transmisión inicial será aproximadamente de 20 kbps. Puesto que el ancho de banda disponible para el emisor TCP puede ser mucho más grande que el valor de MSS/ RTT, al emisor TCP le gustaría poder determinar rápidamente la cantidad de ancho de banda disponible. Por tanto, en el estado de **arranque lento**, el valor de `VentCongestion` se establece en 1 MSS y se incrementa 1 MSS cada vez que se produce el primer reconocimiento de un segmento transmitido. En el ejemplo de la Figura 3.50, TCP envía el primer segmento a la red y espera el correspondiente paquete ACK. Cuando llega dicho paquete de reconocimiento, el emisor TCP incrementa el tamaño de la ventana de congestión en 1 MSS y transmite dos segmentos de tamaño máximo. Estos segmentos son entonces reconocidos y el emisor incrementa el tamaño de la ventana de congestión en 1 MSS por cada uno de los segmentos de reconocimiento, generando así una ventana de congestión de 4 MSS, etc. Este proceso hace que la velocidad de transmisión se duplique en cada periodo RTT. Por tanto, la velocidad de transmisión inicial de TCP es baja, pero crece exponencialmente durante esa fase de arranque lento.

Pero, ¿cuándo debe finalizar este crecimiento exponencial? El algoritmo del arranque lento proporciona varias respuestas a esta cuestión. En primer lugar, si se produce un suceso de pérdida de

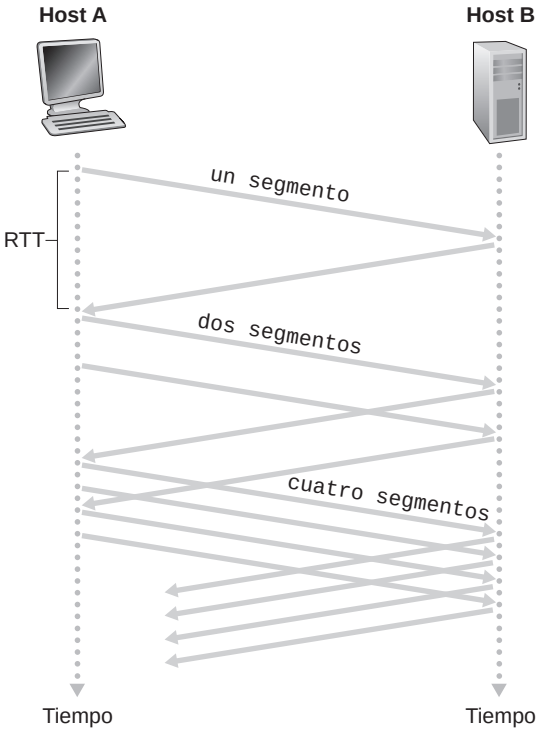

Figura 3.50 ♦ Fase de arranque lento de TCP.

paquete (es decir, si hay congestión) señalado por un fin de temporización, el emisor TCP establece el valor de `VentCongestion` en 1 e inicia de nuevo un proceso de arranque lento. También define el valor de una segunda variable de estado, `umbralAL`, que establece el umbral del arranque lento en `VentCongestion/2`, la mitad del valor del tamaño de la ventana de congestión cuando se ha detectado que existe congestión. La segunda forma en la que la fase de arranque lento puede terminar está directamente ligada al valor de `umbralAL`. Dado que `umbralAL` es igual a la mitad del valor que `VentCongestion` tenía cuando se detectó congestión por última vez, puede resultar algo imprudente continuar duplicando el valor de `VentCongestion` cuando se alcanza o sobrepasa el valor de umbral. Por tanto, cuando el valor de `VentCongestion` es igual a `umbralAL`, la fase de arranque lento termina y las transacciones TCP pasan al modo de evitación de la congestión. Como veremos, TCP incrementa con más cautela el valor de `VentCongestion` cuando está en el modo de evitación de la congestión. La última forma en la que puede terminar la fase de arranque lento es si se detectan tres paquetes ACK duplicados, en cuyo caso TCP realiza una retransmisión rápida (véase la Sección 3.5.4) y entra en el estado de recuperación rápida, que veremos más adelante. El comportamiento de TCP en la fase de arranque lento se resume en la descripción de la FSM del control de congestión de TCP de la Figura 3.51. El algoritmo de arranque lento tiene sus raíces en [Jacobson 1988];en [Jain 1986] se proponía, de forma independiente, un método similar al algoritmo de arranque lento.

Evitación de la congestión

Al entrar en el estado de evitación de la congestión, el valor de `VentCongestion` es aproximadamente igual a la mitad de su valor en el momento en que se detectó congestión por última vez (podemos estar, por tanto, al borde de la congestión). En consecuencia, en lugar de duplicar el valor de `VentCongestion` para cada RTT, TCP adopta un método más conservador e incrementa el valor de `VentCongestion` solamente en un MSS cada RTT [RFC 5681]. Esto puede llevarse

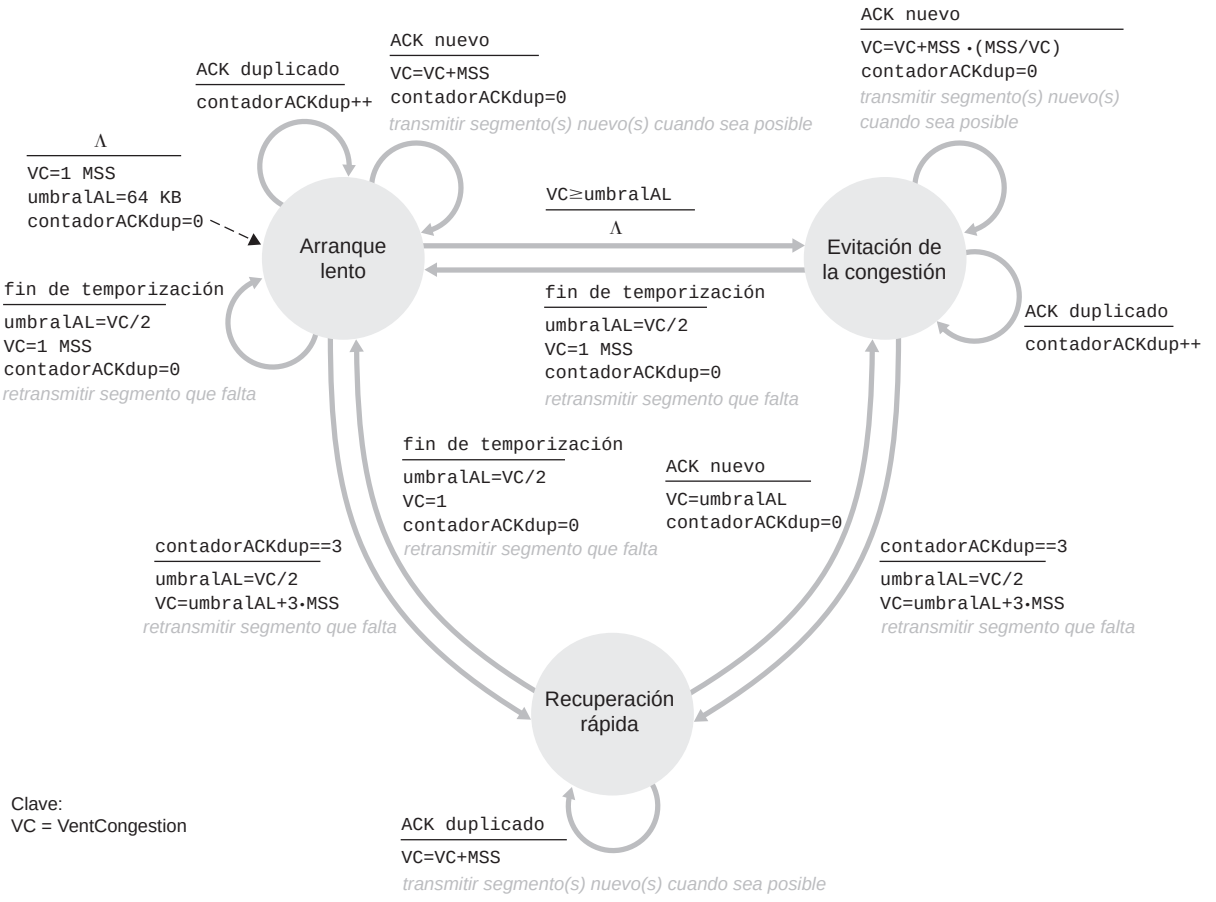

Figura 3.51 ♦ Descripción de la máquina de estados finitos del mecanismo de control de congestión de TCP.

a cabo de varias maneras. Un método habitual consiste en que un emisor TCP aumenta el valor de `VentCongestion` en MSS bytes (MSS/`VentCongestion`) cuando llega un nuevo paquete de reconocimiento. Por ejemplo, si MSS es igual a 1.460 bytes y `VentCongestion` es igual a 14.600 bytes, entonces se enviarán 10 segmentos en un RTT. Cada ACK que llega (suponiendo un ACK por segmento) incrementa el tamaño de la ventana de congestión en 1/10 MSS y, por tanto, el valor del tamaño de la ventana de congestión habrá aumentado en un MSS después de los ACK correspondientes a los 10 segmentos que hayan sido recibidos.

Pero, ¿en qué momento debería detenerse el crecimiento lineal (1 MSS por RTT) en el modo de evitación de la congestión? El algoritmo de evitación de la congestión de TCP se comporta del mismo modo que cuando tiene lugar un fin de temporización. Como en el caso del modo de arranque lento, el valor de `VentCongestion` se fija en 1 MSS y el valor de `umbralAL` se actualiza haciéndose igual a la mitad del valor de `VentCongestion` cuando se produce un suceso de pérdida de paquete. Sin embargo, recuerde que también puede detectarse una pérdida de paquete a causa de la llegada de tres ACK duplicados. En este caso, la red continúa entregando segmentos del emisor al receptor (como señala la recepción de paquetes ACK duplicados). Por tanto, el comportamiento de TCP ante este tipo de pérdida debería ser menos drástico que ante una pérdida de paquete indicada por un fin de temporización: TCP divide entre dos el valor de `VentCongestion` (añadiendo 3 MSS como forma de tener en cuenta los tres ACK duplicados que se han recibido) y configura el valor de `umbralAL`

para que sea igual a la mitad del valor que `VentCongestion` tenía cuando se recibieron los tres ACK duplicados. A continuación, se entra en el estado de recuperación rápida.

Nota de vídeo

Examen del comportamiento de TCP

Recuperación rápida

En la fase de recuperación rápida, el valor de `VentCongestion` se incrementa en 1 MSS por cada ACK duplicado recibido correspondiente al segmento que falta y que ha causado que TCP entre en el estado de recuperación rápida. Cuando llega un ACK para el segmento que falta, TCP entra de nuevo en el estado de evitación de la congestión después de disminuir el valor de `VentCongestion`. Si se produce un fin de temporización, el mecanismo de recuperación rápida efectúa una transición al estado de arranque lento después de realizar las mismas acciones que en los modos de arranque lento y de evitación de la congestión: el valor de `VentCongestion` se establece en 1 MSS

EN LA PRÁCTICA

DIVISIÓN TCP: OPTIMIZACIÓN DEL RENDIMIENTO DE LOS SERVICIOS EN LA NUBE

Para servicios en la nube como los de búsqueda, correo electrónico y redes sociales, resulta deseable proporcionar un alto nivel de capacidad de respuesta, idealmente proporcionando a los usuarios la ilusión de que esos servicios se están ejecutando dentro de sus propios sistemas terminales (incluyendo sus teléfonos inteligentes). Esto puede constituir todo un desafío, ya que los usuarios están ubicados a menudo a mucha distancia de los centros de datos responsables de servir el contenido dinámico asociado con los servicios en la nube. De hecho, si el sistema terminal está lejos de un centro de datos, entonces el RTT será grande, lo que podría dar lugar a un tiempo de respuesta excesivo, debido al arranque lento de TCP.

Como caso de estudio, considere el retardo a la hora de recibir la respuesta a una consulta de búsqueda. Normalmente, el servidor necesita tres ventanas TCP durante el arranque lento para suministrar la respuesta [Pathak 2010]. Por tanto, el tiempo desde que un sistema terminal inicia una conexión TCP, hasta que recibe el último paquete de la respuesta, es de aproximadamente 4 · RTT (un RTT para establecer la conexión TCP y tres RTT para las tres ventanas de datos), más el tiempo de procesamiento en el centro de datos. Estos retardos debidos al RTT pueden dar como resultado un retraso perceptible a la hora de devolver los resultados de las búsquedas, para un porcentaje significativo de las consultas. Además, puede haber un significativo porcentaje de pérdidas de paquetes en las redes de acceso, lo que provocaría retransmisiones TCP y retardos aún más grandes.

Una forma de mitigar este problema y mejorar la capacidad de respuesta percibida por el usuario consiste en (1) desplegar los servidores de *front-end* más cerca de los usuarios y (2) utilizar la **división TCP** (*TCP splitting*), descomponiendo la conexión TCP en el servidor de *front-end*. Con la división TCP, el cliente establece una conexión TCP con el *front-end* cercano, y el *front-end*mantiene una conexión TCP persistente con el centro de datos, con una ventana de congestión TCP muy grande [Tariq 2008, Pathak 2010, Chen 2011]. Con esta técnica, el tiempo de respuesta pasa a ser, aproximadamente, $4 \cdot RTT_{FE} + RTT_{BE}$ + tiempo de procesamiento, donde RTT_{FE} es el tiempo de ida y vuelta entre el cliente y el servidor de *front-end* y RTT_{BE} es el tiempo de ida y vuelta entre el servidor de *front-end* y el centro de datos (servidor de *back-end*). Si el servidor de *front-end* está próximo al cliente, entonces este tiempo de respuesta es aproximadamente igual a RTT más el tiempo de procesamiento, ya que RTT_{FE} es despreciable y RTT_{BE} es aproximadamente RTT. Resumiendo, la técnica de división TCP permite reducir el retardo de red, aproximadamente, de 4· RTT a RTT, mejorando significativamente la capacidad de respuesta percibida por el usuario, en especial para aquellos usuarios que estén lejos de su centro de datos más próximo. La división TCP también ayuda a reducir los retardos de retransmisión TCP provocados por las pérdidas de paquetes en las redes de acceso. Google y Akamai han hecho un amplio uso de sus servidores CDN en las redes de acceso (recuerde nuestra exposición en la Sección 2.6) para llevar a cabo la división TCP para los servicios en la nube que soportan [Chen 2011].

y el valor de `umbralAL` se hace igual a la mitad del valor que tenía `VentCongestion` cuando tuvo lugar el suceso de pérdida.

El mecanismo de recuperación rápida es un componente de TCP recomendado, aunque no obligatorio [RFC 5681]. Es interesante resaltar que una versión anterior de TCP, conocida como **TCP Tahoe**, establece incondicionalmente el tamaño de la ventana de congestión en 1 MSS y entra en el estado de arranque lento después de un suceso de pérdida indicado por un fin de temporización o por la recepción de tres ACK duplicados. La versión más reciente de TCP, **TCP Reno**, incorpora la recuperación rápida.

La Figura 3.52 ilustra la evolución de la ventana de congestión de TCP para Reno y Tahoe. En esta figura, inicialmente el umbral es igual a 8 MSS. Durante los ocho primeros ciclos de transmisión, Tahoe y Reno realizan acciones idénticas. La ventana de congestión crece rápidamente de forma exponencial durante la fase de arranque lento y alcanza el umbral en el cuarto ciclo de transmisión. A continuación, la ventana de congestión crece linealmente hasta que se produce un suceso de tres ACK duplicados, justo después del octavo ciclo de transmisión. Observe que el tamaño de la ventana de congestión es igual a $12 \cdot MSS$ cuando se produce el suceso de pérdida. El valor de `umbralAL` se hace entonces igual a $0{,}5 \cdot$ `VentCongestion` $= 6 \cdot$ MSS. En TCP Reno, el tamaño de la ventana de congestión es puesto a `VentCongestion` $= 9 \cdot$ MSS y luego crece linealmente. En TCP Tahoe, la ventana de congestión es igual a 1 MSS y crece exponencialmente hasta que alcanza el valor de `umbralAL`, punto a partir del cual crece linealmente.

La Figura 3.51 presenta la descripción completa de la máquina de estados finitos de los algoritmos del mecanismo de control de congestión de TCP: arranque lento, evitación de la congestión y recuperación rápida. En la figura también se indica dónde pueden producirse transmisiones de nuevos segmentos y dónde retransmisiones de segmentos. Aunque es importante diferenciar entre las retransmisiones/control de errores de TCP y el control de congestión de TCP, también lo es apreciar cómo estos dos aspectos de TCP están estrechamente vinculados.

Control de congestión de TCP: retrospectiva

Una vez vistos los detalles de las fases de arranque lento, de evitación de la congestión y de recuperación rápida, merece la pena retroceder un poco para clarificar las cosas. Ignorando la fase inicial de arranque lento en la que se establece la conexión y suponiendo que las pérdidas están indicadas por la recepción de tres ACK duplicados en lugar de por fines de temporización, el control de congestión de TCP consiste en un crecimiento lineal (aditivo) de `VentCongestion` a razón de 1 MSS por RTT, seguido de un decrecimiento multiplicativo (división entre dos) del tamaño de la ventana, `VentCongestion`, cuando se reciben tres ACK duplicados. Por esta razón, suele decirse que el control de congestión de TCP es una forma de **crecimiento aditivo y decrecimiento multiplicativo**

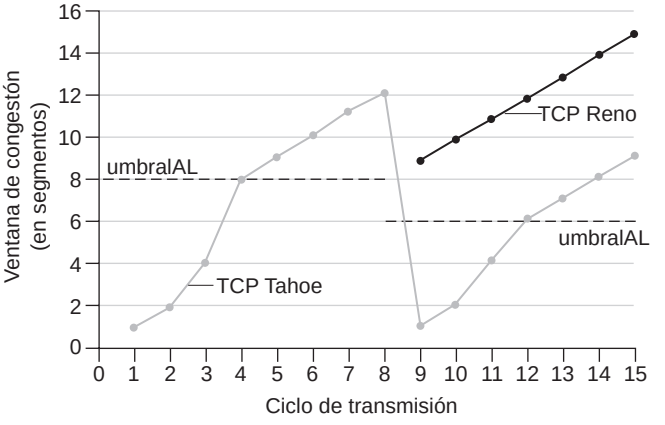

Figura 3.52 ♦ Evolución de la ventana de congestión de TCP (Tahoe y Reno).

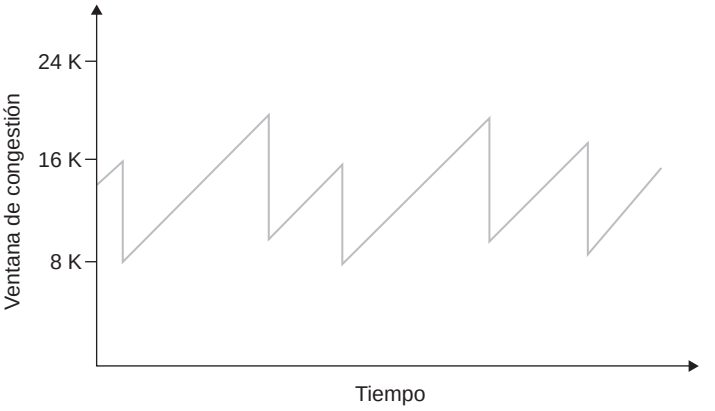

Figura 3.53 ♦ Control de congestión con crecimiento aditivo y decrecimiento multiplicativo.

(**AIMD, *Additive-Increase, Multiplicative-Decrease*)** de control de congestión. El control de congestión AIMD presenta un comportamiento en forma de "diente de sierra", como se muestra en la Figura 3.53, lo que también ilustra nuestra anterior intuición de que TCP "va tanteando" el ancho de banda. (TCP aumenta linealmente el tamaño de su ventana de congestión, y por tanto su velocidad de transmisión, hasta que tiene lugar la recepción de tres ACK duplicados. A continuación, divide entre dos su ventana de congestión, pero vuelve después a crecer linealmente, tanteando para ver si hay disponible ancho de banda adicional.)

Como hemos mencionado anteriormente, la mayor parte de las implementaciones TCP actuales emplean el algoritmo Reno [Padhye 2001]. Se han propuesto muchas variantes del algoritmo Reno [RFC 3782; RFC 2018]. El algoritmo TCP Vegas [Brakmo 1995; Ahn 1995] intenta evitar la congestión manteniendo una buena tasa de transferencia. La idea básica del algoritmo Vegas es (1) detectar la congestión en los routers existentes entre el origen y el destino *antes* de que se pierda un paquete y (2) reducir la velocidad linealmente cuando se detecta una pérdida inminente de paquetes. La pérdida inminente de un paquete se predice observando el RTT. Cuanto mayor es el RTT de los paquetes, mayor es la congestión en los routers. A finales de 2015, la implementación Linux Ubuntu de TCP proporcionaba arranque lento, evitación de congestión, recuperación rápida, retransmisión rápida y SACK, de manera predeterminada; también se proporcionan algoritmos alternativos de control de congestión, como TCP Vegas y BIC [Xu 2004]. En [Afanasyev 2010] se proporciona un resumen de las muchas variantes de TCP.

El algoritmo AIMD de TCP fue desarrollado basándose en un enorme trabajo de ingeniería y de experimentación con los mecanismos de control de congestión en redes reales. Diez años después del desarrollo de TCP, los análisis teóricos mostraron que el algoritmo de control de congestión de TCP sirve como un algoritmo de optimización asíncrona distribuido que da como resultado la optimización simultánea de diversos aspectos importantes, tanto de las prestaciones proporcionadas al usuario como del rendimiento de la red [Kelly 1998]. Desde entonces se han desarrollado muchos aspectos teóricos del control de congestión [Srikant 2004].

Descripción macroscópica de la tasa de transferencia de TCP

Visto el comportamiento en diente de sierra de TCP, resulta natural preguntarse cuál es la tasa de transferencia media (es decir, la velocidad media) de una conexión TCP de larga duración. En este análisis vamos a ignorar las fases de arranque lento que tienen lugar después de producirse un fin de temporización (estas fases normalmente son muy cortas, ya que el emisor sale de ellas rápidamente de forma exponencial). Durante un intervalo concreto de ida y vuelta, la velocidad a la que TCP envía datos es función del tamaño de la ventana de congestión y del RTT actual.

Cuando el tamaño de la ventana es *w* bytes y el tiempo actual de ida y vuelta es RTT segundos, entonces la velocidad de transmisión de TCP es aproximadamente igual a *w*/*RTT*. TCP comprueba entonces si hay ancho de banda adicional incrementando *w* en 1 MSS cada *RTT* hasta que se produce una pérdida. Sea *W* el valor de *w* cuando se produce una pérdida. Suponiendo que *RTT* y *W* son aproximadamente constantes mientras dura la conexión, la velocidad de transmisión de TCP varía entre $W/(2 \cdot RTT)$ y W/RTT.

Estas suposiciones nos llevan a un modelo macroscópico extremadamente simplificado del comportamiento en régimen permanente de TCP. La red descarta un paquete de la conexión cuando la velocidad aumenta hasta *W/RTT*; la velocidad entonces se reduce a la mitad y luego aumenta MSS/*RTT* cada *RTT* hasta que de nuevo alcanza el valor *W/RTT*. Este proceso se repite una y otra vez. Puesto que la tasa de transferencia (es decir, la velocidad) de TCP aumenta linealmente entre los dos valores extremos, tenemos

$$\text{tasa de transferencia media de una conexión} = \frac{0,75 \cdot W}{RTT}$$

Utilizando este modelo altamente idealizado para la dinámica del régimen permanente de TCP, también podemos deducir una expresión interesante que relaciona la tasa de pérdidas de una conexión con su ancho de banda disponible [Mahdavi 1997]. Dejamos esta demostración para los problemas de repaso. Un modelo más sofisticado que, según se ha comprobado, concuerda empíricamente con los datos medidos es el que se describe en [Padhye 2000].

TCP en rutas con un alto ancho de banda

Es importante darse cuenta de que el control de congestión de TCP ha ido evolucionando a lo largo de los años y todavía sigue evolucionando. Puede leer un resumen sobre las variantes de TCP actuales y una exposición acerca de la evolución de TCP en [Floyd 2001, RFC 5681, Afanasyev 2010]. Lo que era bueno para Internet cuando la mayoría de las conexiones TCP transportaban tráfico SMTP, FTP y Telnet no es necesariamente bueno para la Internet actual, en la que domina HTTP, o para una futura Internet que proporcione servicios que hoy ni siquiera podemos imaginar.

La necesidad de una evolución continua de TCP puede ilustrarse considerando las conexiones TCP de alta velocidad necesarias para las aplicaciones de computación reticular (*grid-computing*) y de computación en la nube. Por ejemplo, considere una conexión TCP con segmentos de 1.500 bytes y un *RTT* de 100 milisegundos y suponga que deseamos enviar datos a través de esta conexión a 10 Gbps. Siguiendo el documento [RFC 3649], observamos que utilizar la fórmula anterior para la tasa de transferencia de TCP, con el fin de alcanzar una tasa de transferencia de 10 Gbps, nos daría un tamaño medio de la ventana de congestión de 83.333 segmentos, que son *muchos* segmentos, lo que despierta el temor a que uno de esos 83.333 segmentos en tránsito pueda perderse. ¿Qué ocurriría si se produjeran pérdidas? O, dicho de otra manera, ¿qué fracción de los segmentos transmitidos podría perderse que permitiera al algoritmo de control de congestión de TCP de la Figura 3.51 alcanzar la velocidad deseada de 10 Gbps? En las cuestiones de repaso de este capítulo, mostraremos cómo derivar una fórmula que expresa la tasa de transferencia de una conexión TCP en función de la tasa de pérdidas (*L*), el tiempo de ida y vuelta (*RTT*) y el tamaño máximo de segmento (*MSS*):

$$\text{tasa de transferencia media de una conexión} = \frac{1,22 \cdot MSS}{RTT\sqrt{L}}$$

Con esta fórmula, podemos ver que para alcanzar una tasa de transferencia de 10 Gbps, el actual algoritmo de control de congestión de TCP solo puede tolerar una probabilidad de pérdida de segmentos de $2 \cdot 10^{-10}$ (o, lo que es equivalente, un suceso de pérdida por cada 5.000.000.000 segmentos), lo cual es una tasa muy baja. Esta observación ha llevado a una serie de investigadores a buscar nuevas versiones de TCP que estén diseñadas específicamente para estos entornos de alta

velocidad; consulte [Jin 2004; Kelly 2003; Ha 2008; RFC 7323] para obtener información sobre estos trabajos.

3.7.1 Equidad

Considere ahora *K* conexiones TCP, cada una de ellas con una ruta terminal a terminal diferente, pero atravesando todas ellas un enlace de cuello de botella con una velocidad de transmisión de *R* bps (con *enlace de cuello de botella* queremos decir que, para cada conexión, todos los restantes enlaces existentes a lo largo de la ruta de la conexión no están congestionados y tienen una capacidad de transmisión grande comparada con la capacidad de transmisión del enlace de cuello de botella). Suponga que cada conexión está transfiriendo un archivo de gran tamaño y que no existe tráfico UDP atravesando el enlace de cuello de botella. Se dice que un mecanismo de control de congestión es *equitativo* si la velocidad media de transmisión de cada conexión es aproximadamente igual a *R/K*; es decir, cada conexión obtiene la misma cuota del ancho de banda del enlace.

¿Es el algoritmo AIMD de TCP equitativo, teniendo en cuenta que diferentes conexiones TCP pueden iniciarse en instantes distintos y, por tanto, pueden tener distintos tamaños de ventana en un instante determinado? [Chiu 1989] proporciona una explicación elegante e intuitiva de por qué el control de congestión de TCP converge para proporcionar la misma cuota de ancho de banda de un enlace de cuello de botella a las conexiones TCP que compiten por el ancho de banda.

Consideremos el caso simple de dos conexiones TCP que comparten un mismo enlace, cuya velocidad de transmisión es *R*, como se muestra en la Figura 3.54. Suponemos que las dos conexiones tienen el mismo MSS y el mismo RTT (por lo que si tienen el mismo tamaño de ventana de congestión, entonces tienen la misma tasa de transferencia). Además, tienen que enviar una gran cantidad de datos y ninguna otra conexión TCP ni datagrama UDP atraviesan este enlace compartido. Asimismo, vamos a ignorar la fase de arranque lento de TCP y vamos a suponer que las conexiones TCP están operando en el modo de evitación de la congestión (AIMD) durante todo el tiempo.

La gráfica de la Figura 3.55 muestra la tasa de transferencia de las dos conexiones TCP. Si TCP está diseñado para que las dos conexiones compartan equitativamente el ancho de banda del enlace, entonces la tasa de transferencia alcanzada debería caer a lo largo de la flecha que sale del origen con un ángulo de 45 grados (cuota equitativa de ancho de banda). Idealmente, la suma de las dos tasas de transferencia tiene que ser igual a *R* (evidentemente, que cada conexión reciba una cuota equitativa de la capacidad del enlace, pero igual a cero, no es una situación deseable). Por tanto, el objetivo debería ser que las tasas de transferencia alcanzadas se encuentren en algún punto próximo a la intersección de la línea de cuota equitativa de ancho de banda con la línea de utilización del ancho de banda completo mostradas en la Figura 3.55.

Suponga que los tamaños de ventana de TCP son tales que, en un instante determinado, las conexiones 1 y 2 alcanzan las tasas de transferencia indicadas por el punto *A* de la Figura 3.55. Puesto que la cantidad de ancho de banda del enlace conjunto consumido por las dos conexiones es menor que *R*, no se producirá ninguna pérdida y ambas conexiones incrementarán sus tamaños de ventana en

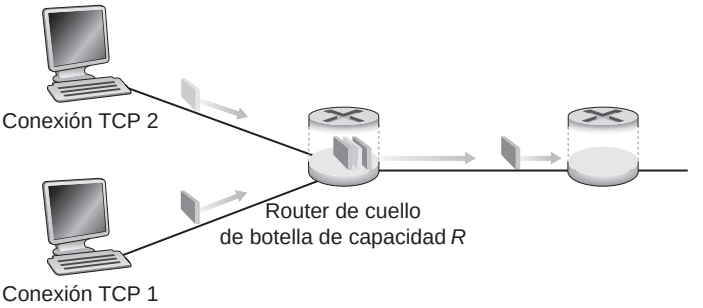

Figura 3.54 ♦ Dos conexiones TCP que comparten un mismo enlace cuello de botella.

1 MSS por RTT como resultado del algoritmo de evitación de la congestión de TCP. Por tanto, la tasa de transferencia conjunta de las dos conexiones sigue la línea de 45 grados (incremento equitativo para ambas conexiones) partiendo del punto *A*. Finalmente, el ancho de banda del enlace consumido conjuntamente por las dos conexiones será mayor que *R*, por lo que terminará produciéndose una pérdida de paquetes. Suponga que las conexiones 1 y 2 experimentan una pérdida de paquetes cuando alcanzan las tasas de transferencia indicadas por el punto *B*. Entonces las conexiones 1 y 2 reducen el tamaño de sus ventanas en un factor de dos. Las tasas de transferencia resultantes se encuentran por tanto en el punto *C*, a medio camino de un vector que comienza en *B* y termina en el origen. Puesto que el ancho de banda conjunto utilizado es menor que *R* en el punto *C*, de nuevo las dos conexiones incrementan sus tasas de transferencia a lo largo de la línea de 45 grados partiendo de *C*. Finalmente, terminará por producirse de nuevo una pérdida de paquetes, por ejemplo, en el punto *D*, y las dos conexiones otra vez reducirán el tamaño de sus ventanas en un factor de dos, y así sucesivamente. Compruebe que el ancho de banda alcanzado por ambas conexiones fluctúa a lo largo de la línea que indica una cuota equitativa del ancho de banda. Compruebe también que las dos conexiones terminarán convergiendo a este comportamiento, independientemente de su posición inicial dentro de ese espacio bidimensional. Aunque en este escenario se han hecho una serie de suposiciones ideales, permite proporcionar una idea intuitiva de por qué TCP hace que el ancho de banda se reparta de forma equitativa entre las conexiones.

En este escenario ideal hemos supuesto que solo las conexiones TCP atraviesan el enlace de cuello de botella, que las conexiones tienen el mismo valor de RTT y que solo hay una conexión TCP asociada con cada pareja origen-destino. En la práctica, estas condiciones normalmente no se dan y las aplicaciones cliente-servidor pueden por tanto obtener cuotas desiguales del ancho de banda del enlace. En particular, se ha demostrado que cuando varias conexiones comparten un cuello de botella común, aquellas sesiones con un valor de RTT menor son capaces de apropiarse más rápidamente del ancho de banda disponible en el enlace, a medida que este va liberándose (es decir, abren más rápidamente sus ventanas de congestión) y por tanto disfrutan de una tasa de transferencia más alta que aquellas conexiones cuyo valor de RTT es más grande [Lakshman 1997].

Equidad y UDP

Acabamos de ver cómo el control de congestión de TCP regula la velocidad de transmisión de una aplicación mediante el mecanismo de la ventana de congestión. Muchas aplicaciones multimedia,

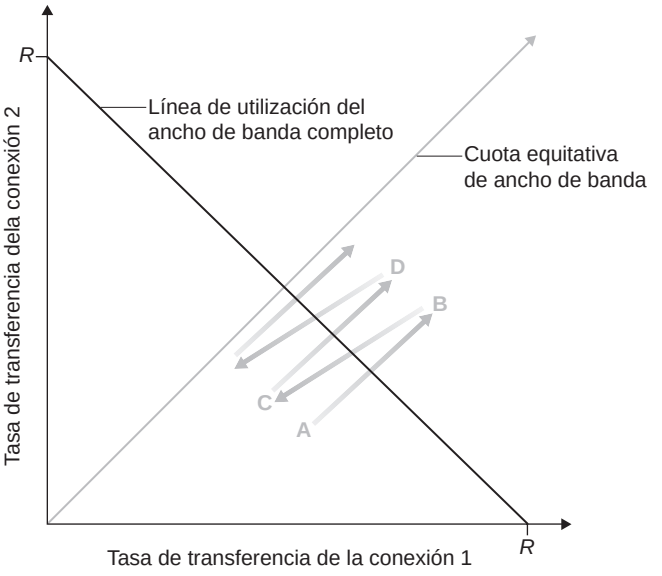

Figura 3.55 ♦ Tasa de transferencia alcanzada por las conexiones TCP 1 y 2.

como las videoconferencias y la telefonía por Internet, a menudo no se ejecutan sobre TCP precisamente por esta razón (no desean que su velocidad de transmisión se regule, incluso aunque la red esté muy congestionada). En lugar de ello, estas aplicaciones prefieren ejecutarse sobre UDP, que no incorpora un mecanismo de control de la congestión. Al ejecutarse sobre UDP, las aplicaciones pueden entregar a la red sus datos de audio y de vídeo a una velocidad constante y, ocasionalmente, perder paquetes, en lugar de reducir sus velocidades a niveles "equitativos" y no perder ningún paquete. Desde la perspectiva de TCP, las aplicaciones multimedia que se ejecutan sobre UDP no son equitativas (no cooperan con las demás conexiones ni ajustan sus velocidades de transmisión apropiadamente). Dado que el control de congestión de TCP disminuye la velocidad de transmisión para hacer frente a un aumento de la congestión (y de las pérdidas) y los orígenes de datos UDP no lo hacen, puede darse el caso de que esos orígenes UDP terminen por expulsar al tráfico TCP. Un área actual de investigación es el desarrollo de mecanismos de control de congestión para Internet que impidan que el tráfico UDP termine por reducir a cero la tasa de transferencia de Internet [Floyd 1999; Floyd 2000; Kohler 2006; RFC 4340].

Equidad y conexiones TCP en paralelo

Pero aunque se pudiera forzar al tráfico UDP a comportarse equitativamente, el problema de la equidad todavía no estaría completamente resuelto. Esto es porque no hay nada que impida a una aplicación basada en TCP utilizar varias conexiones en paralelo. Por ejemplo, los navegadores web a menudo utilizan varias conexiones TCP en paralelo para transferir los distintos objetos contenidos en una página web (en la mayoría de los navegadores puede configurarse el número exacto de conexiones múltiples). Cuando una aplicación emplea varias conexiones en paralelo obtiene una fracción grande del ancho de banda de un enlace congestionado. Por ejemplo, considere un enlace cuya velocidad es R que soporta nueve aplicaciones entrantes cliente-servidor, utilizando cada una de las aplicaciones una conexión TCP. Si llega una nueva aplicación y también emplea una conexión TCP, entonces cada conexión tendrá aproximadamente la misma velocidad de transmisión de $R/10$. Pero si esa nueva aplicación utiliza 11 conexiones TCP en paralelo, entonces la nueva aplicación obtendrá una cuota no equitativa de más de $R/2$. Dado que el tráfico web es el dominante en Internet, las conexiones múltiples en paralelo resultan bastante comunes.

3.7.2 Notificación explícita de congestión (ECN): control de congestión asistido por la red

Desde la estandarización inicial del arranque lento y de la evitación de la congestión a finales de la década de 1980 [RFC 1122], TCP ha implementado el tipo de control de congestión de extremo a extremo que hemos estudiado en la Sección 3.7.1: un emisor TCP no recibe ninguna indicación explícita de congestión por parte de la capa de red, deduciendo en su lugar la existencia de congestión a partir de la pérdida de paquetes observada. Más recientemente, se han propuesto, implementado e implantado extensiones tanto a IP como a TCP [RFC 3168] que permiten a la red señalizar explícitamente la congestión a un emisor y un receptor TCP. Este tipo de control de congestión asistido por la red se conoce con el nombre de **notificación explícita de congestión (ECN, *Explicit Congestion Notification*)**. Como se muestra en la Figura 3.56, están implicados los protocolos TCP e IP.

En la capa de red, se utilizan para ECN dos bits (lo que da un total de cuatro posibles valores) del campo Tipo de Servicio de la cabecera del datagrama IP (de la que hablaremos en la Sección 4.3). Una de las posibles configuraciones de los bits ECN es usada por los routers para indicar que están experimentando congestión. Esta indicación de congestión es transportada entonces en dicho datagrama IP hasta el host de destino, que a su vez informa al host emisor, como se muestra en la Figura 3.56. RFC 3168 no proporciona una definición de cuándo un router está congestionado; esa decisión es una opción de configuración proporcionada por el fabricante del router y sobre la cual decide el operador de la red. Sin embargo, RFC 3168 sí que recomienda que la indicación de

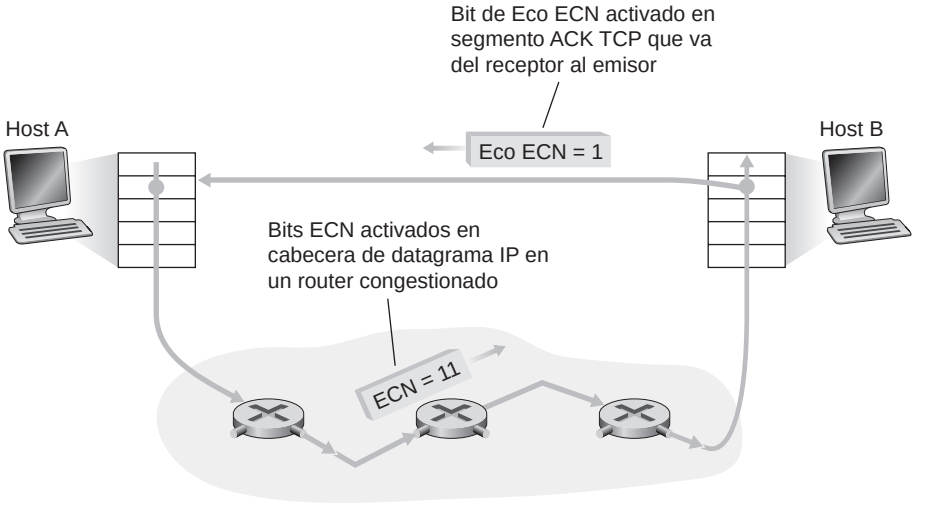

Figura 3.56 ♦ Notificación explícita de congestión: control de congestión asistido por la red.

congestión ECN solo se active cuando exista una congestión persistente. Una segunda configuración de los bits ECN es utilizada por el host emisor para informar a los routers de que el emisor y el receptor son compatibles con ECN, y por tanto capaces de llevar a cabo acciones es respuesta a las indicaciones de congestión de la red proporcionadas por ECN.

Como se muestra en la Figura 3.56, cuando la capa TCP en el host receptor detecta una indicación ECN de congestión en un datagrama recibido, informa de esa situación de congestión a la capa TCP del host emisor, activando el bit ECE (*Explicit Congestion Notification Echo*, bit de Eco ECN; véase la Figura 3.29) en un segmento ACK TCP enviado por el receptor al emisor. Al recibir un ACK con una indicación ECE de congestión, el emisor TCP reacciona a su vez dividiendo por dos la ventana de congestión, igual que lo haría al perderse un segmento mientras se usa retransmisión rápida, y activa el bit CWR (*Congestion Windows Reduced*, ventana de congestión reducida) en la cabecera del siguiente segmento TCP transmitido por el emisor hacia el receptor.

Además de TCP, también otros protocolos de la capa de transporte pueden hacer uso de la señalización ECN de la capa de red. El protocolo DCCP (*Datagram Congestion Control Protocol*, protocolo de control de congestión de datagramas) [RFC 4340] proporciona un servicio no fiable tipo UDP con control de congestión y baja sobrecarga administrativa, que emplea ECN. DCTCP (Data Center TCP, TCP para centros de datos) [Alizadeh 2010], que es una versión de TCP diseñada específicamente para redes de centros de datos, también utiliza ECN.

3.8 Resumen

Hemos comenzado este capítulo estudiando los servicios que un protocolo de la capa de transporte puede proporcionar a las aplicaciones de red. En uno de los extremos, el protocolo de capa de transporte puede ser muy simple y ofrecer un servicio poco sofisticado a las aplicaciones, poniendo a su disposición únicamente una función de multiplexación/demultiplexación para los procesos que se están comunicando. El protocolo UDP de Internet es un ejemplo de ese tipo de protocolo de la capa de transporte poco sofisticado. En el otro extremo, un protocolo de la capa de transporte puede proporcionar a las aplicaciones diversos tipos de garantías, como por ejemplo la de entrega fiable de los datos, garantías sobre los retardos y garantías concernientes al ancho de banda. De todos modos,

los servicios que un protocolo de transporte puede proporcionar están a menudo restringidos por el modelo de servicio del protocolo subyacente de la capa de red. Si el protocolo de la capa de red no puede proporcionar garantías sobre los retardos o de ancho de banda a los segmentos de la capa de transporte, entonces el protocolo de la capa de transporte no puede proporcionar garantías de retardo ni de ancho de banda a los mensajes intercambiados por los procesos.

En la Sección 3.4 hemos visto que un protocolo de la capa de transporte puede proporcionar una transferencia fiable de los datos incluso aunque el protocolo de red subyacente sea no fiable. Allí vimos que son muchas las sutilezas implicadas en la provisión de una transferencia fiable de los datos, pero que dicha tarea puede llevarse a cabo combinando cuidadosamente los paquetes de reconocimiento, los temporizadores, las retransmisiones y los números de secuencia.

Aunque hemos hablado de la transferencia fiable de los datos en este capítulo, debemos tener presente que esa transferencia fiable puede ser proporcionada por los protocolos de las capas de enlace, de red, de transporte o de aplicación. Cualquiera de las cuatro capas superiores de la pila de protocolos puede implementar los reconocimientos, los temporizadores, las retransmisiones y los números de secuencia, proporcionando así un servicio de transferencia de datos fiable a la capa que tiene por encima. De hecho, a lo largo de los años, los ingenieros e informáticos han diseñado e implementado de manera independiente protocolos de las capas de enlace, de red, de transporte y de aplicación que proporcionan una transferencia de datos fiable (aunque muchos de estos protocolos han desaparecido silenciosamente de la escena).

En la Sección 3.5 hemos examinado en detalle TCP, el protocolo fiable y orientado a la conexión de la capa de transporte de Internet. Hemos visto que TCP es bastante complejo, incluyendo técnicas de gestión de la conexión, de control de flujo y de estimación del tiempo de ida y vuelta, además de una transferencia fiable de los datos. De hecho, TCP es bastante más complejo de lo que nuestra descripción deja entrever; hemos dejado fuera de nuestra exposición, intencionadamente, diversos parches, correcciones y mejoras de TCP que están ampliamente implementadas en distintas versiones de dicho protocolo. De todos modos, todas estas complejidades están ocultas a ojos de la aplicación de red. Si el cliente en un host quiere enviar datos de forma fiable a un servidor implementado en otro host, se limita a abrir un socket TCP con el servidor y a bombear datos a través de dicho socket. Afortunadamente, la aplicación cliente-servidor es completamente inconsciente de toda la complejidad de TCP.

En la Sección 3.6 hemos examinado el control de congestión desde una perspectiva amplia, mientras que en la Sección 3.7 hemos mostrado cómo se implementa ese mecanismo de control de congestión en TCP. Allí vimos que el control de congestión es obligatorio para la buena salud de la red. Sin él, una red se puede colapsar fácilmente, sin que al final puedan transportarse datos de terminal a terminal. En la Sección 3.7 vimos que TCP implementa un mecanismo de control de congestión terminal a terminal que incrementa de forma aditiva su tasa de transmisión cuando se evalúa que la ruta seguida por la conexión TCP está libre de congestión, mientras que esa tasa de transmisión se reduce multiplicativamente cuando se producen pérdidas de datos. Este mecanismo también trata de proporcionar a cada conexión TCP que pasa a través de un enlace congestionado una parte equitativa del ancho de banda del enlace. También hemos examinado con cierta profundidad el impacto que el establecimiento de la conexión TCP y el lento arranque de la misma tienen sobre la latencia. Hemos observado que, en muchos escenarios importantes, el establecimiento de la conexión y el arranque lento contribuyen significativamente al retardo terminal a terminal. Conviene recalcar una vez más que, aunque el control de congestión de TCP ha ido evolucionando a lo largo de los años, continúa siendo un área intensiva de investigación y es probable que continúe evolucionando en los próximos años.

El análisis realizado en este capítulo acerca de los protocolos específicos de transporte en Internet se ha centrado en UDP y TCP, que son los dos "caballos de batalla" de la capa de transporte de Internet. Sin embargo, dos décadas de experiencia con estos dos protocolos han permitido identificar una serie de casos en los que ninguno de los dos resulta ideal. Los investigadores han dedicado, por tanto, grandes esfuerzos al desarrollo de protocolos adicionales de la capa de transporte, varios de los cuales son actualmente estándares propuestos por IETF.

El Protocolo de control de congestión para datagramas (DCCP, *Datagram Congestion Control Protocol*) [RFC 4340] proporciona un servicio no fiable con baja carga administrativa y orientado a mensajes, similar a UDP, pero que cuenta con un tipo de control de congestión seleccionado por la aplicación y que es compatible con TCP. Si una aplicación necesita una transferencia de datos fiable o semifiable, entonces el control de congestión sería implementado dentro de la propia aplicación, quizá utilizando los mecanismos que hemos estudiado en la Sección 3.4. DCCP está previsto para utilizarlo en aplicaciones tales como los flujos multimedia (véase el Capítulo 9) que pueden jugar con los compromisos existentes entre los requisitos de temporización y de fiabilidad en la entrega de datos, pero que quieran a la vez poder responder a situaciones de congestión en la red.

El protocolo QUIC (*Quick UDP Internet Connections*) de Google [Iyengar 2016], implementado en el navegador Chromium de Google, proporciona fiabilidad por medio de las retransmisiones, así como corrección de errores, configuración rápida de la conexión y un algoritmo de control de congestión basado en la velocidad que trata de ser compatible con TCP, todo ello implementado como un protocolo de la capa de aplicación por encima de UDP. A principios de 2015, Google informó de que aproximadamente la mitad de todas las solicitudes enviadas desde Chrome a los servidores Google se sirven a través de QUIC.

DCTCP (*Data Center TCP*) [Alizadeh 2010] es una versión de TCP diseñada específicamente para las redes de centros de datos y utiliza ECN para proporcionar un mejor soporte a la mezcla de flujos con tiempos de vida cortos y largos que caracterizan a las cargas de trabajo de los centros de datos.

El Protocolo de transmisión para control de flujos (SCTP, *Stream Control Transmission Protocol*) [RFC 4960, RFC 3286] es un protocolo fiable orientado a mensajes, que permite multiplexar diferentes "flujos" de nivel de aplicación a través de una única conexión SCTP (una técnica conocida con el nombre de "*multi-streaming*"). Desde el punto de vista de la fiabilidad, los diferentes flujos que comparten la conexión se gestionan de forma separada, de modo que la pérdida de paquetes en uno de los flujos no afecte a la entrega de los datos en los otros. SCTP también permite transferir datos a través de dos rutas de salida cuando un host está conectado a dos o más redes; también existe la posibilidad de la entrega opcional de datos fuera de orden, así como otra serie de características interesantes. Los algoritmos de control de flujo y de control de congestión de SCTP son prácticamente los mismos que en TCP.

El protocolo de Control de tasa compatible con TCP (TFRC, *TCP-Friendly Rate Control*) [RFC 5348] es un protocolo de control de congestión más que un protocolo completo de la capa de transporte. TFRC especifica un mecanismo de control de congestión que podría ser utilizado en algún otro protocolo de transporte, como DCCP (de hecho, uno de los dos protocolos seleccionables por la aplicación existentes en DCCP es TFRC). El objetivo de TFRC es suavizar el comportamiento típico en "diente de sierra" (véase la Figura 3.53) que se experimenta en el control de congestión de TCP, al mismo tiempo que se mantiene una tasa de transmisión a largo plazo "razonablemente" próxima a la TCP. Con una tasa de transmisión de perfil más suave que TCP, TFRC está bien adaptado a aplicaciones multimedia tales como la telefonía IP o los flujos multimedia, en donde es importante mantener ese perfil suave de la tasa de transmisión. TFRC es un protocolo "basado en ecuaciones" que utiliza la tasa medida de pérdida de paquetes como entrada para una ecuación [Padhye 2000] que permite estimar cuál sería la tasa de transferencia TCP si una sesión TCP experimentara dicha tasa de pérdidas. Entonces, dicha tasa de transferencia se adopta como objetivo de tasa de transmisión para TFRC.

Solo el futuro nos dirá si DCCP, SCTP, QUIC o TFRC serán adoptados ampliamente o no. Aunque estos protocolos proporcionan claramente una serie de capacidades mejoradas respecto a TCP y UDP, estos dos protocolos han demostrado ser a lo largo de los años "lo suficientemente buenos". El que un "mejor" protocolo termine venciendo a otro que es "suficientemente bueno" dependerá de una compleja mezcla de aspectos técnicos, sociales y empresariales.

En el Capítulo 1 hemos visto que una red de computadoras puede dividirse entre lo que se denomina la "frontera de la red" y el "núcleo de la red". La frontera de la red cubre todo lo que sucede en los sistemas terminales. Habiendo ya cubierto la capa de aplicación y la capa de transporte,