4. LA CAPA DE TRANSPORTE

4.1 La capa de transporte y sus servicios

- Un protocolo de la capa de transporte proporciona una **comunicación lógica** entre procesos de aplicación que se ejecutan en host diferentes.
- Los procesos de aplicación utilizan la comunicación lógica proporcionada por la capa de trasporte para enviarse mensajes entre sí, sin preocuparse de los detalles de la infraestructura física utilizada para transportar los mensajes.
- Los protocolos de la capa de transporte están implementados en los sistemas terminales, pero no en los routers de la red.
- En el lado emisor, la capa de transporte convierte los mensajes que recibe procedentes de un proceso de aplicación emisor en paquetes de la capa de transporte, conocidos como **segmentos** de la capa de transporte en la terminología de Internet. Esto se hace dividiendo los mensajes de la aplicación en fragmentos más pequeños y añadiendo una cabecera de la capa de transporte a cada fragmento con el fin de crear el segmento de la capa de transporte. A continuación, la capa de trasporte pasa el segmento a la capa de red del sistema terminal emisor, donde el segmento se encapsula dentro de un paquete de la capa de red (datagrama) y se envía al destino.
- En el lado del receptor, la capa de red extrae el segmento de la capa de transporte del datagrama y lo sube a la capa de transporte. A continuación, esta capa procesa el segmento recibido, poniendo los datos del segmento a disposición de la aplicación de recepción.
- Para las aplicaciones de red puede haber más de un protocolo de la capa de transporte disponible (TCP, UDP).

4.1.1 Relaciones entre las capas de transporte y de red

- Mientras que un protocolo de la capa de transporte proporciona una comunicación lógica entre *procesos* que se ejecutan en host diferentes, un protocolo de la capa de red proporciona una comunicación lógica entre *host*.
- Los protocolos de la capa de transporte residen en los sistemas terminales. Dentro de un sistema terminal, el protocolo de la capa de transporte lleva los mensajes desde los procesos de la aplicación a la frontera de la red (la capa de red) y viceversa, pero no tiene nada que ver con cómo se transmiten los mensajes dentro del núcleo de la red.
- Una red de computadoras puede emplear distintos protocolos de transporte, ofreciendo cada uno de ellos un modelo de servicio distinto a las aplicaciones.
- Los servicios que un protocolo de transporte puede proporcionar a menudo están restringidos por el modelo de servicio del protocolo de la capa de red subyacente. Si el protocolo de la capa de red no proporciona garantías acerca del retardo ni del ancho de banda para los segmentos de la capa de transporte enviados entre host, entonces el protocolo de la capa de transporte no puede proporcionar ninguna garantía acerca del retardo o del ancho de banda a los mensajes de aplicación enviados entre procesos.
- No obstante, el protocolo de la capa de transporte *puede* ofrecer ciertos servicios incluso cuando el protocolo de red subyacente no ofrezca el servicio correspondiente en la capa de red.

4.1.2 La capa de transporte en Internet

- Internet y de forma más general TCP/IP, ponen a disposición de la capa de aplicación dos protocolos de la capa de transporte diferentes:
    - **UDP** (*UserDatagramProtocol*), que proporciona un servicio sin conexión no fiable a la aplicación que le invoca.

- **TCP** (*Transmision Control Protocol*), que proporciona a la aplicación que le invoca un servicio orientado a la conexión fiable.

- El protocolo de la capa de red de Internet es el protocolo **IP** (*Internet Protocol*). IP proporciona una comunicación lógica entre host. El modelo de servicio de IP es un **servicio de entrega de mejor esfuerzo** (**besteffort**). Esto quiere decir que IP hace todo lo que puede por entregar los segmentos entre los host que se están comunicando, pero no garantiza la entrega. Por estas razones, se dice que IP es un **servicio no fiable**. Además sabemos que todos los host tienen al menos una dirección IP asociada.
- La responsabilidad principal de UDP y TCP es ampliar el servicio de entrega de IP entre dos sistemas terminales a un servicio de entrega entre dos proceso que estén ejecutándose en los sistemas terminales.
- Extender la entrega host a host a una entrega proceso a proceso es lo que se denomina **multiplexación** y **demultiplexación de la capa de transporte**.
- UDP y TCP también proporcionan servicios de comprobación de la integridad de los datos al incluir campos de detección de errores en las cabeceras de sus segmentos.
- Estos dos servicios de la capa de transporte son los dos únicos servicios de que ofrece UDP. En particular, al igual que IP, UDP es un servicio no fiable, no garantiza que los datos enviados por un proceso lleguen intacto, o incluso que lleguen al proceso destino.
- TCP, por el contrario, ofrece a las aplicaciones varios servicios adicionales:
  - **Transferencia de datos fiable**:
    - Utilizando técnicas de control de flujo, números de secuencia, mensajes de reconocimiento y temporizadores.
    - TCP garantiza que los datos transmitidos por el proceso emisor sean entregados al proceso receptor, correctamente y en orden. De modo que TCP convierte el servicio no fiable de IP en un servicio de transporte fiable entre procesos.

  - **Control de congestión**:
    - Los mecanismos de control de congestión de TCP evitan que cualquier conexión TCP inunde con una cantidad de tráfico excesiva los enlaces y los routers existentes entre los host que se están comunicándose.
    - TCP se esfuerza en proporcionar a cada conexión que atraviesa un enlace congestionado la misma cuota de ancho de banda del enlace. Esto se consigue regulando la velocidad a la que los lados emisores de las conexiones TCP pueden enviar tráfico a la red.

- El tráfico UDP, por el contrario, no está regulado. Una aplicación que emplee el protocolo UDP puede enviar los datos a la velocidad que le parezca, durante todo el tiempo que quiera.

4.2 Multiplexación y demultiplexación

- Se va a centrar la explicación de este servicio básico de la capa de transporte en el contexto de Internet. Sin embargo, hay que destacar que un servicio de multiplexación es necesario en todas las redes de computadoras.
- En el host destino, la capa de transporte recibe segmentos procedentes de la capa de red que tiene justo debajo.
- La capa de transporte tiene la responsabilidad de entregar los datos contenidos en estos segmentos al proceso de la aplicación apropiada que está ejecutándose en el host.
- Recordemos que un proceso, como parte de una aplicación de red, puede tener uno o **más** sockets, por los que pasan los datos de la red al proceso y viceversa. Por tanto, la capa de transporte del host receptor realmente no entrega los datos realmente a un proceso, sino a un socket intermedio.

- Dado que en cualquier instante puede haber más de un socket en el host receptor, cada socket tiene asignado un identificador único. El formato de este identificador depende de si se trata de un socket UDP o de un socket TCP.
- Cada segmento de la capa de transporte contiene un conjunto de campos destinados a dirigir un segmento de entrada de la capa de transporte al socket apropiado. En el extremo receptor, la capa de transporte examina estos campos para identificar el socket receptor y a continuación envía el segmento a dicho socket. Esta tarea es lo que se denomina **demultiplexación**.

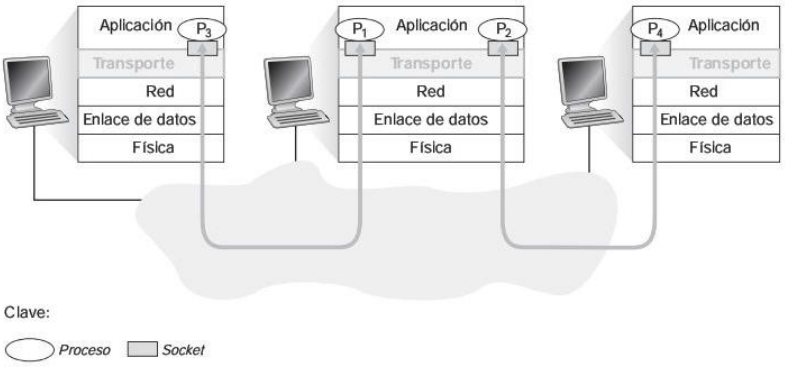


Figura 3.2 • Multiplexación y demultiplexación en la capa de transporte.

- La tarea de reunir los fragmentos de datos en el host origen desde los diferentes sockets, encapsulando cada fragmento de datos con la información de cabecera (la cual se utilizará después en el proceso de demultiplexación) para crear los segmentos y pasarlos a la capa de red es lo que se denomina **multiplexación**.
- Por tanto, la operación de multiplexación que se lleva a cabo en la capa de transporte requiere:
    1. Que los sockets tengan identificadores únicos.
    2. Que cada segmento tenga campos especiales que identifiquen al socket al que tiene que entregarse el segmento.

Figura 3.3 • Los campos número de puerto de origen y de destino en un segmento de la capa de transporte.

- Estos campos especiales son el campo **número de puerto de origen** y el campo **número de puerto destino**.
- Cada número de puerto es un número de 16 bits comprendido en el rango de 0 a 65535. Los números de puertos pertenecientes al rango de 0 a 1023 se conocen como **números de puertos bien conocidos** y son restringidos, lo que significa que están reservados para emplearlos por los protocolos de aplicación bien conocidos.
- Se puede encontrar la lista de números de puerto bien conocidos en el [RFC 1700] y su actualización en http://www.iana.org [RFC 3232].

Multiplexación y demultiplexación sin conexión

- Un socket UDP queda completamente identificado por una tupla que consta de una dirección IP destino y un número de puerto destino. En consecuencia, si dos segmentos UDP tienen diferentes direcciones IP y/o números de puerto de origen, pero la misma dirección IP de destino y el mismo número de puerto de destino, entonces los dos segmentos se enviarán al mismo proceso de destino a través del mismo socket de destino.

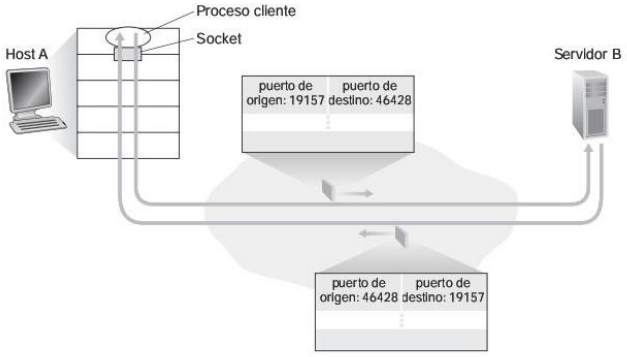


Figura 3.4 • Inversión de los números de puerto de origen y de destino.

Multiplexación y demultiplexación con conexión

- Una sutil diferencia entre un socket TCP y uno UDP es que el primero queda identificado por una tupla de cuatro elementos, que constan de dirección IP de origen, número de puerto de origen, dirección IP de destino, número de puerto de destino. Por tanto, cuando un segmento TCP llega a un host procedente de la red, el host emplea los cuatro valores para dirigir (demultiplexar) el segmento al socket apropiado.
- En particular y al contrario de lo que ocurre con UDP, dos segmentos TCP entrantes con direcciones IP de origen o números de puerto de origen diferentes (con la excepción de un segmento TCP que transporte la solicitud original de establecimiento de conexión) serán dirigidos a dos socket distintos.

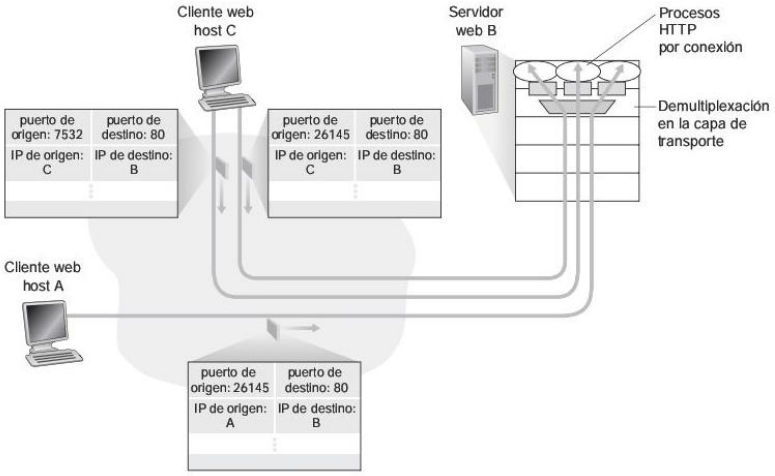


Figura 3.5 • Dos clientes utilizando el mismo número de puerto de destino (80) para comunicarse con la misma aplicación de servidor web.

Servidores web y TCP

- La Figura 3.5 muestra un servidor web que genera un nuevo proceso para cada conexión. Cada uno de estos procesos tiene su propio socket de conexión a través del cual llegan las solicitudes HTTP y se envían las respuestas HTTP.
- Sin embargo, se ha mencionado que no existe siempre una correspondencia uno a uno entre los sockets de conexión y los procesos. De hecho, los servidores web actuales de altas prestaciones a menudo solo utilizan un proceso y crean una nueva hebra con un nuevo socket de conexión para cada nueva conexión de un cliente. En un servidor así, en cualquier instante puede haber muchos sockets de conexión asociados al mismo proceso.
- Si el cliente y el servidor web están utilizando HTTP persistente, entonces mientras dure la conexión persistente, el cliente y el servidor intercambiarán mensajes HTTP a través del mismo sockets de servidor. Sin embargo, si el cliente y el servidor emplean HTTP no persistente, entonces se creará y cerrará una nueva conexión TCP para cada pareja solicitud/respuesta y por tanto, se creará y cerrará un nuevo socket para cada solicitud/respuesta.
- Esta creación y cierre de sockets tan frecuente puede afectar seriamente al rendimiento de un servidor web ocupado.

4.3 Transporte sin conexión: UDP [RFC 768]

- UDP hace casi lo mínimo que un protocolo de transporte debe hacer. Además de la función de [de]multiplexación y de algún mecanismo de comprobación de errores, no añade nada a IP.
- UDP toma los mensajes procedentes del proceso de la aplicación, asocia los campos correspondientes a los números de puerto de origen y destino para proporcionar el servicio de [de]multiplexación, añade dos campos pequeños más y pasa el segmento resultante a la capa de red. La capa de red encapsula el segmento de la capa de transporte en un datagrama IP y luego hace el mejor esfuerzo por entregar el segmento al host receptor. Si el segmento llega al host receptor, UDP utiliza el número de puerto de destino para entregar los datos del segmento al proceso apropiado de la capa de aplicación.
- Con UDP no tiene lugar una fase de establecimiento de la conexión entre las distintas entidades de la capa de transporte emisora y receptora previa al envío del segmento. Por esto, se dice que UDP es un protocolo sin conexión.
- Entonces. ¿No sería preferible emplear siempre TCP? La respuesta es no, ya que muchas aplicaciones están mejor adaptadas a UDP por las siguiente razones:

- Mejor control en el nivel de aplicación sobre qué datos se envían y cuando:
  - Con UDP, tan pronto como un proceso de la capa de aplicación pasa datos a UDP, UDP los empaqueta en un segmento UDP e inmediatamente entrega el segmento a la capa de red.
  - Puesto que las aplicaciones en tiempo real suelen requerir una velocidad mínima de transmisión, no permiten un retardo excesivo en la transmisión de los segmentos y pueden tolerar algunas pérdidas de datos, el modelo de servicio de TCP no se adapta demasiado bien a las necesidades de este tipo de aplicaciones.

- Sin establecimiento de la conexión:
  - UDP inicia la transmisión sin formalidades preliminares. Por tanto, UDP no añade ningún retardo a causa del establecimiento de una conexión.

- Sin información del estado de la conexión:
  - UDP no mantiene información del estado de la conexión y no controla ningún parámetro relativo de control de congestión, número de secuencia y de reconocimiento. Por esta razón, un servidor dedicado a una aplicación concreta suele poder soportar más clientes activos cuando la aplicación se ejecuta sobre UDP que cuando lo hace sobre TCP.

- Poca sobrecarga debida a la cabecera de los paquetes

- Por tanto, UDP es preferible a TCP en aplicaciones (servidor de archivos remotos, flujos multimedia, Telefonía por Internet, Administración de res, protocolos de enrutamiento, traducción de nombres, etc.) tolerantes a la perdida de una pequeña cantidad de paquetes, por lo que una transferencia de datos no fiable no es absolutamente crítica para que la aplicación funcione correctamente.
- Como ya se ha dicho, UDP no proporciona mecanismos de control de congestión y estos mecanismos son necesarios para impedir que la red entre en un estado de congestión en el que se realice muy poco trabajo útil. Por tanto, la ausencia de un mecanismo de control de congestión en UDP puede dar lugar a altas tasas de pérdidas entre un emisor y un receptor UDP y al estrangulamiento de las sesiones TCP, lo cual es un problema potencialmente serio.
- Sin embargo, es posible que una aplicación disponga de un servicio fiable de transferencia de datos utilizando UDP. Esto puede conseguirse si las características de fiabilidad se incorporan a la propia aplicación. Pero se trata de una tarea nada sencilla que requiere una intensa tarea de depuración de aplicaciones. No obstante, incorporar los mecanismos de fiabilidad directamente en la aplicación permite que los procesos de aplicación puedan comunicarse de forma estable sin estar sujetos a las restricciones de la velocidad de transmisión impuestas por el mecanismo de control de congestión de TCP.

4.3.1 Estructura de los segmentos UDP [RFC 768]

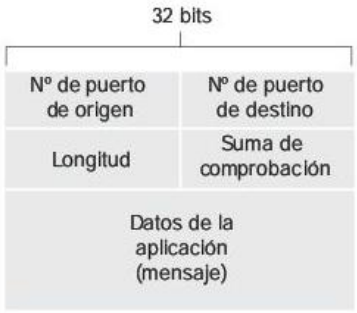


Figura 3.7 • Estructura de un segmento UDP.

4.3.2 Suma de comprobación UDP

- Proporciona un mecanismo de detección de errores.
- UDP en el lado del emisor calcula el complemento a 1 de la suma de todas las palabras de 16 bits del segmento, acarreando cualquier desbordamiento obtenido durante la operación de suma sobre el bit de menor peso.
- En receptor, las cuatro palabras de 16 bits se suman, incluyendo la suma de comprobación. Si no se han introducido errores en el paquete, entonces la suma en el receptor tiene que ser 0xFFFF. Si uno de los bits es un 0, entonces se sabe que el paquete contiene errores.

4.4 Principios de un servicio de transferencia de datos fiable

- La Figura 3.8 ilustra el marco de trabajo que se va a emplear en el estudio sobre la transferencia de datos fiable.

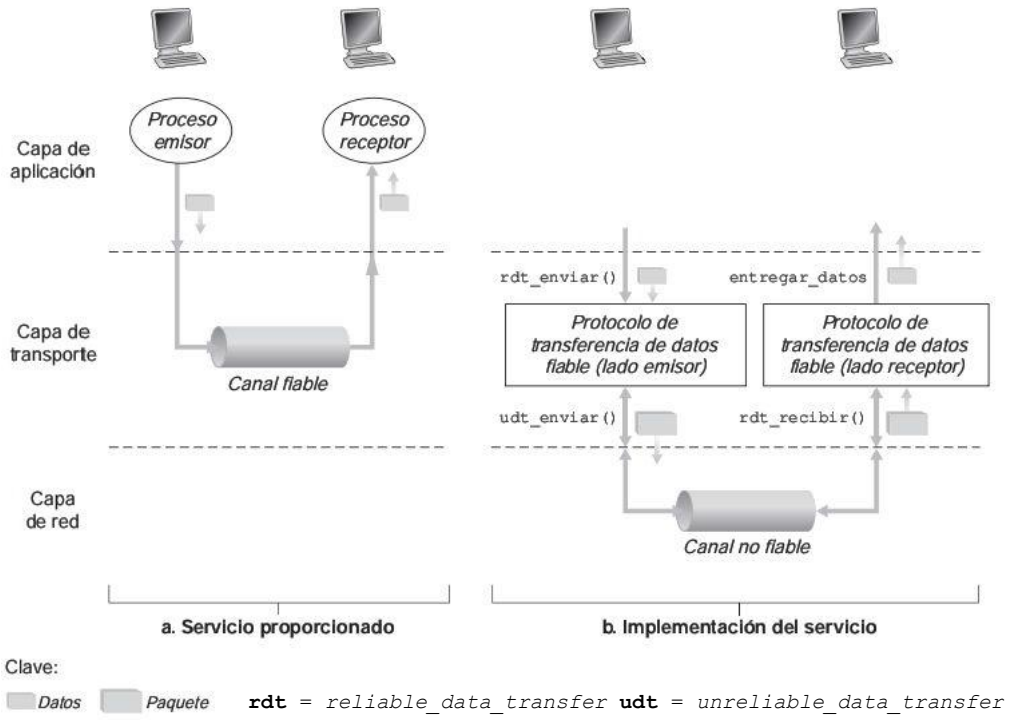


Figura 3.8 • Transferencia de datos fiable: modelo del servicio e implementación del servicio.

- La abstracción del servicio proporcionada a las entidades de la capa superior es la de un canal fiable a través del cual se pueden transferir datos.
- Disponiendo de un canal fiable, ninguno de los bits de datos transferidos está corrompido ni se pierde, y todos se entregan en el orden en que fueron enviados.
- Éste es precisamente el modelo de servicio ofrecido por TCP a las aplicaciones de Internet que lo invocan.
- Es la responsabilidad de un **protocolo de transferencia de datos fiable** implementar esta abstracción del servicio.
- Esta tarea es complicada por el hecho de que la capa que hay por debajo puede ser no fiable.

4.4.1 Construcción de un protocolo de transferencia de datos fiable

Transferencia de datos fiable sobre un canal totalmente fiable: `rdt1.0`

- En primer lugar se va a considerar el caso más simple en el que el canal subyacente es fiable.
- El protocolo en sí, que denominaremos `rdt1.0`, es trivial.

- En la Figura 3.9 se muestran las definiciones de las **máquinas de estados finitos (FSM, *Finite-State Machine*)** para el emisor y el receptor.
- El lado emisor simplemente acepta datos de la capa superior a través del suceso `rdt_enviar(datos)`, crea un paquete que contiene los datos mediante la acción `crear_paquete(datos)` y envía el paquete al canal mediante `udt_enviar(paquete)`.
- En el lado del receptor, `rdt` recibe un paquete del canal subyacente a través del suceso `rdt_recibir(paquete)`, extrae los datos del paquete mediante `extraer(paquete, datos)` y pasa los datos a la capa superior a mediante la acción `entregar_datos(datos)`.
- En este protocolo tan simple:
    - No existe ninguna diferencia entre una unidad de datos y un paquete.
    - Todo el flujo de paquetes va del emisor al receptor ya que disponiendo de un canal totalmente fiable no existe la necesidad en el lado del receptor de proporcionar ninguna realimentación al emisor, puesto que no hay nada que pueda ser incorrecto.
    - El receptor podría recibir los datos tan rápido como el emisor los enviara. Luego tampoco existe la necesidad de que el receptor le pida al emisor que vaya más despacio.

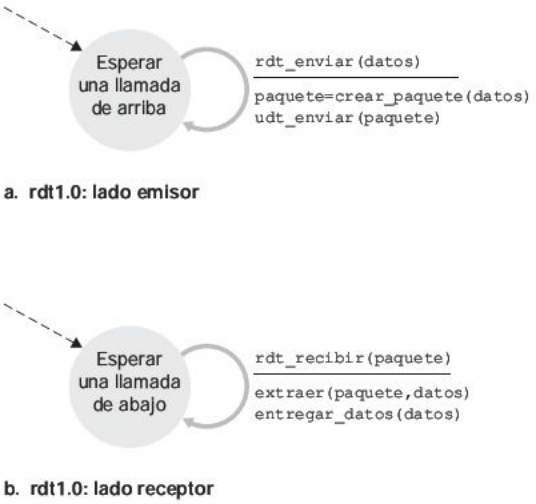


Figura 3.9 • `rdt1.0`: protocolo para un canal totalmente fiable.

Transferencia de datos fiable sobre un canal con errores de bit: `rdt2.0`

- Normalmente, los errores de bit se producen en los componentes físicos de una red cuando un paquete se transmite, se propaga o accede a un buffer.
- En una red de computadoras, los protocolos de transferencia de datos fiables basados en retransmisiones de **reconocimientos positivo**s y **reconocimientos negativos**, se conocen como **protocolos ARQ (*Automatic Repeat reQuest*)**.
- En los protocolos ARQ se requieren fundamentalmente tres capacidades de protocolo adicional para gestionar la presencia de errores de bit:
    - <u>Detección de errores</u>:
        - Se necesita un mecanismo que permita al receptor detectar que se han producido errores de bit.
        - Estas técnicas requieren que el emisor envíe al receptor bits adicionales y dichos bits también se tendrán en cuenta para el cálculo de la suma de comprobación del paquete de datos.

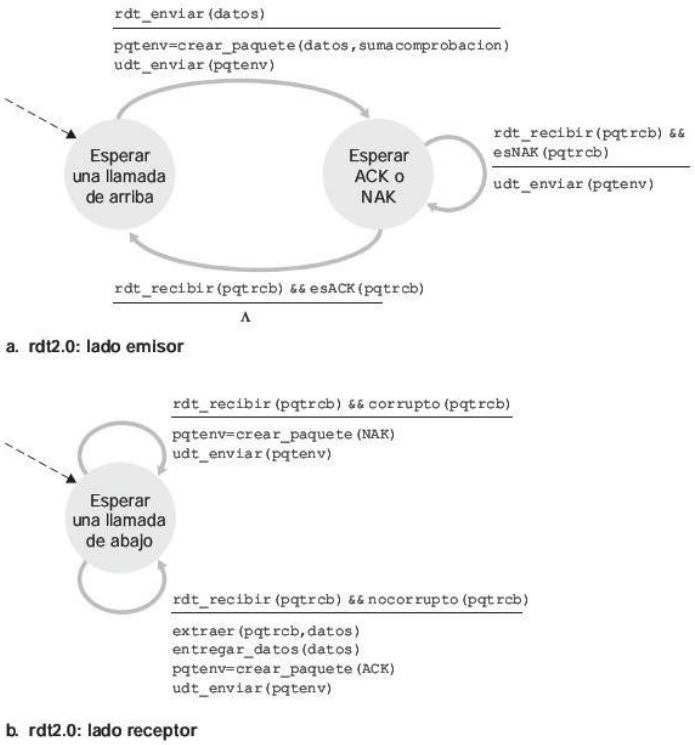


Figura 3.10 • `rdt2.0`: protocolo para un canal con errores de bit.

- Realimentación del receptor:
  - La única forma de que el emisor sepa lo que ocurre en el receptor es que el receptor envíe explícitamente información de realimentación al emisor.
  - Las respuestas de acuse de recibo o reconocimiento positivo (**ACK**) y reconocimiento negativo (**NAK**) son ejemplos de realimentación.

- Retransmisión. Un paquete que se recibe con errores en el receptor será retransmitido por el emisor.

- Los protocolos cómo `rdt2.0` funcionan pero tienen un defecto fatal. No tienen en cuenta la posibilidad de que el paquete ACK o NACK pueda estar corrompido.
- La cuestión más complicada es cómo puede recuperarse el protocolo de los errores en los paquetes ACK o NAK. La dificultad está en que si un paquete ACK o NAK está corrompido, el emisor no tiene forma de saber si el receptor ha recibido o no correctamente el último fragmento de datos transmitido.
- Consideremos tres posibilidades para gestionar paquetes ACK o NAK corruptos:
  - Una primera posibilidad podría ser que si el emisor no entiende la respuesta, éste le pregunte nuevamente al receptor, introduciendo un nuevo tipo de paquete (del emisor al receptor). A continuación, el receptor repetirá la respuesta. Sin embargo, nuevamente estamos ante el mismo problema.
  - Una segunda alternativa consistiría en añadir los suficientes bits de suma de comprobación como para permitir al emisor no solo detectar, sino también de recuperarse de los errores de bit.
  - Un tercer método consistiría simplemente en que el emisor reenviara el paquete de datos actual al recibir un paquete ACK o NAK alterado. Sin embargo este método introduce **paquetes duplicados** en el canal emisor-receptor.

    La principal dificultad con los paquetes duplicados es que el receptor no sabe si el último paquete ACK o NAK enviado fue recibido correctamente en el emisor. Por tanto, *a priori*, no puede saber si un paquete entrante contiene datos nuevos o es una retransmisión.

    Una solución sencilla a este problema consiste en añadir un nuevo campo al paquete de datos y hacer que el emisor numere sus paquetes de datos colocando un número de secuencia en este campo.
  - Las figuras 3.11 y 3.12 muestran la descripción de la máquina de estados finitos para `rdt2.1`, la versión revisada de `rdt2.0`.

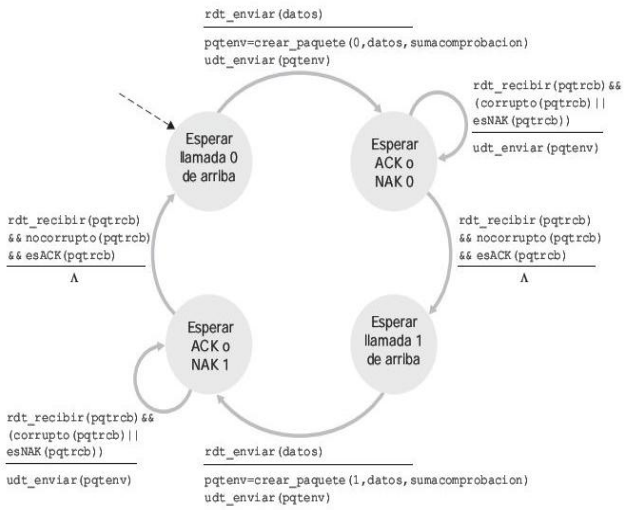


Figura 3.11 • Lado emisor de `rdt2.1`.

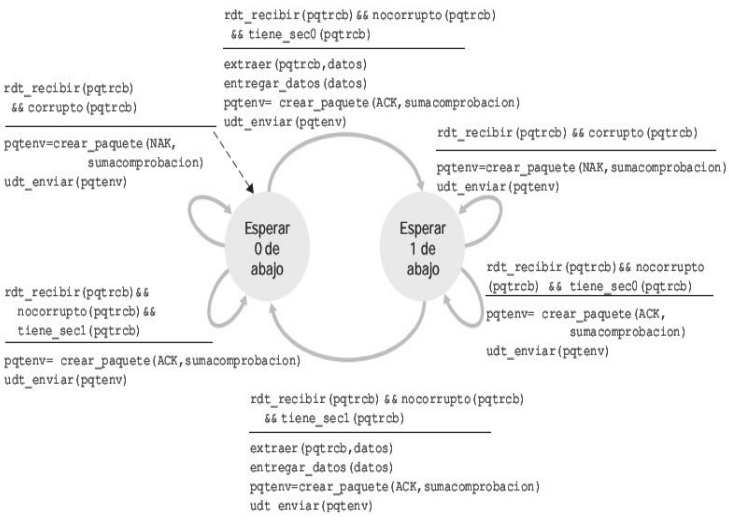


Figura 3.12 • Lado receptor de `rdt2.1`.

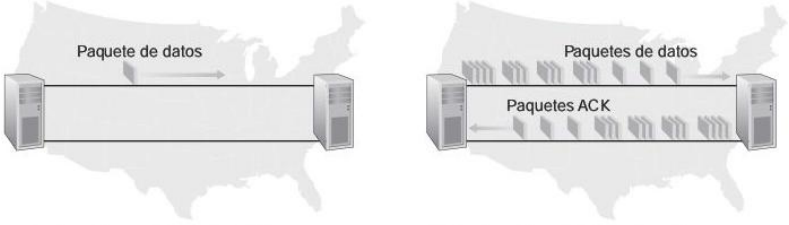
- Supongamos ahora que además de bits corrompidos, el canal subyacente también puede perder paquetes. Por tanto ahora el protocolo tiene que preocuparse por dos problemas más:
  - Cómo detectar la pérdida de paquetes:
    - Supongamos nuevamente que el emisor transmite un paquete de datos y bien el propio paquete o el mensaje ACK del receptor para ese paquete se pierde.
    - En cualquiera de estos dos casos, al emisor no le llega ninguna respuesta procedente del receptor.
    - Si el emisor está dispuesto a esperar el tiempo suficiente como para estar seguro de que se ha perdido un paquete, simplemente puede retransmitirlo.

  - Qué hacer cuando se pierde un paquete:
    - El emisor tiene que esperar al menos un tiempo igual al retardo de ida y vuelta entre el emisor y el receptor, más una cierta cantidad de tiempo que será necesaria para procesar un paquete en el receptor.
    - Idealmente el protocolo debería recuperarse de la pérdida de paquetes tan pronto como fuera posible, pero si espera un tiempo igual al retardo en el caso peor, eso significa una larga espera hasta iniciar el mecanismo de recuperación de errores.
    - El método que se adopta en la práctica es que el emisor seleccione juiciosamente un intervalo de tiempo tal que sea probable que un paquete se haya perdido, aunque no sea seguro que tal pérdida se haya producido. Si dentro de ese intervalo de tiempo no se ha recibido un ACK, el paquete se retransmite. Esto introduce la posibilidad de que existan **paquetes duplicados** en el canal emisor-receptor.
    - Desde el punto de vista del emisor, la retransmisión es la solución para todo.
    - La implementación de un mecanismo de retransmisión basado en tiempo requiere un **temporizador de cuenta atrás** que pueda interrumpir al emisor después de que haya transcurrido un determinado periodo de tiempo.

- Dado que los números de secuencia de los paquetes alternan entre 0 y 1, el protocolo rdt3.0 se denomina en ocasiones **protocolo de bit alternante**.

4.4.2 Protocolo de transferencia de datos fiable con procesamiento en cadena

- El protocolo rdt3.0 es un protocolo funcionalmente correcto, pero es muy improbable que haya alguien a quien satisfaga su rendimiento.
- La base del problema del rendimiento de rdt3.0se encuentra en el hecho de que es un protocolo de parada y espera.
- La solución a este problema de rendimiento concreto es, en lugar de operar en el modo parada y espera, el emisor podría enviar varios paquetes sin esperar a los mensajes de reconocimiento.



a. Funcionamiento de un protocolo de parada y espera.

b. Funcionamiento de un protocolo con procesamiento en cadena.

Figura 3.17 • Protocolo de parada y espera y protocolo con procesamiento en cadena.

- Dado que los muchos paquetes que están en tránsito entre el emisor y el receptor pueden visualizarse como el relleno de un conducto (*pipeline*), esta técnica se conoce como **pipelining** o **procesamiento en cadena**.
- Este procesamiento tiene las siguientes consecuencias en los protocolos de transferencia de datos fiables:
    - El rango de los números de secuencia tiene que incrementarse, dado que cada paquete en tránsito tiene que tener un número de secuencia único.
    - Los lados emisor y receptor de los protocolos pueden tener que almacenar en buffer más de un paquete.
    - El rango de los números de secuencia y los requisitos de buffer dependerán de la forma en que un protocolo de transferencia de datos responda a la perdida de paquetes y a los paquetes corrompidos o excesivamente retardados. Hay disponibles dos métodos que permiten la recuperación de errores mediante procesamiento en cadena:
        - **Retroceder N**
        - **Repetición selectiva**

4.4.3 Retroceder N (GBN, *Go-Back-N*)

- En un protocolo GBN, el emisor puede transmitir varios paquetes sin tener que esperar a que sean reconocidos, pero está restringido a no tener más de un número máximo permitido `N`, de paquetes no reconocidos en el canal.

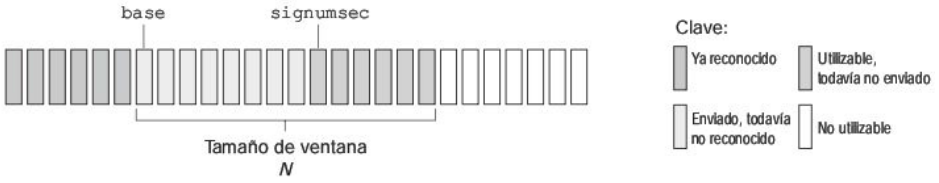


Figure 3.19 • Números de secuencia en el emisor en el protocolo Retroceder N.

- Si definimos:
    - La `base` como el número de secuencia del paquete no reconocido más antiguo.
    - `signunsec` como el número de secuencia más pequeño no utilizado.

  Entonces se puede identificar los cuatro intervalos en rango de los números de secuencia:
    - [`0`, `base-1`] corresponden a paquetes que ya han sido transmitidos y reconocidos.
    - [`base`, `signunsec - 1`] corresponden a paquetes que ya han sido enviados pero todavía no se han reconocido.
    - [`signunsec`, `base + N - 1`] se pueden emplear para los paquetes que pueden ser enviados de forma inmediata, en caso de que lleguen datos procedentes de la capa superior.
    - Los número de secuencia $\geq$ que `base + N` no pueden ser utilizados hasta que un paquete no reconocido que se encuentre actualmente en el canal sea reconocido.

- El rango de los números de secuencia permitidos para los paquetes trasmitidos pero todavía no reconocidos puede visualizarse como una ventana de tamaño `N` sobre el rango de los números de secuencia.
- Cuando el protocolo opera, esta ventana se desplaza hacia adelante sobre el espacio de los números de secuencia. Por esta razón `N` suele denominarse **tamaño de ventana** y el protocolo GBN se dice que es un **protocolo de ventana deslizante**.
- En la práctica, el número de secuencia de un paquete se incluye en un campo de longitud fija de la cabecera del paquete. Si k es el número de bit contenidos en el campo que especifica el número de secuencia del paquete, el rango de los números de secuencia será $[0, 2^k - 1]$.

- TCP utiliza un campo para el número de secuencia de 32 bits.
- Las Figuras 3.20 y 3.21 proporcionan una descripción ampliada de la máquina de estados finitos de un protocolo GBN:

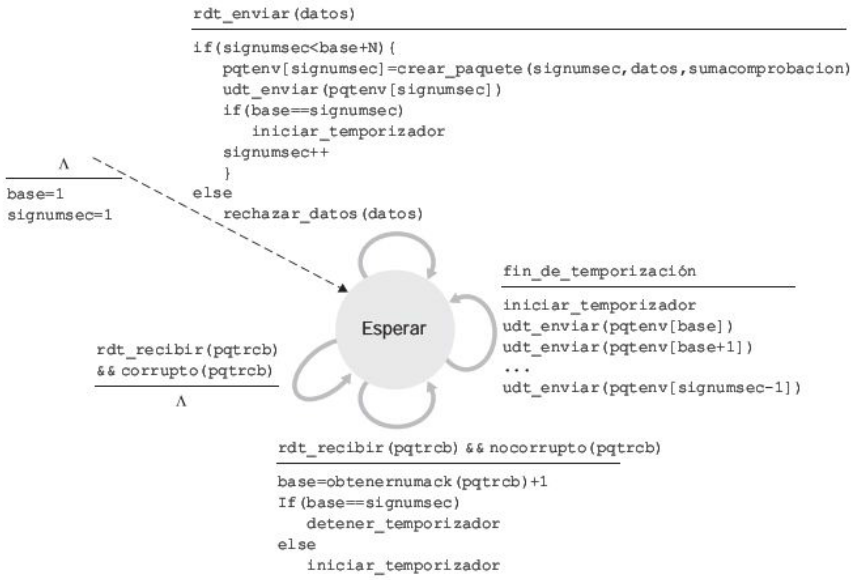


Figura 3.20 • Descripción de la FSM ampliada del lado de emisión de GBN.

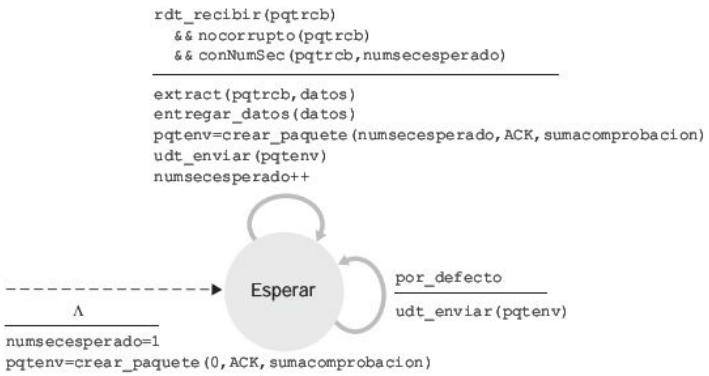


Figura 3.21 • Descripción de la FSM ampliada del lado receptor de GBN.

- El emisor del protocolo GBN tiene que responder a tres tipos de sucesos:
  - Invocación desde la capa superior.
  - Recepción de un mensaje de reconocimiento (**reconocimiento acumulativo**).
  - Un suceso de fin de temporización. El nombre de este protocolo, "Retroceder N", se deriva del comportamiento del emisor en presencia de paquetes perdidos o muy retardados. Como en los protocolos de parada y espera, se empleará un temporizador para recuperarse de la pérdida de paquetes de datos o de reconocimiento de paquetes. Si se produce un fin de temporización, el emisor reenvía todos los paquetes que haya transmitido anteriormente y que todavía no hayan sido reconocidos.

- Las acciones del receptor también son simples. Si un paquete con un número de secuencia n se recibe correctamente y en orden, el receptor envía un paquete ACK para el paquete n y entrega la parte de los datos del paquete a la capa superior. En todos los restantes casos, el receptor descarta el paquete y reenvía un mensaje ACK para el paquete recibido en orden más recientemente.

- Por tanto, el uso de confirmaciones acumulativas es una opción natural del protocolo GBN.
- La ventaja de descartar un paquete correctamente recibido pero fuera de orden es la simplicidad del almacenamiento en el buffer del receptor. Por el contrario, la desventaja es que la subsiguiente retransmisión de dicho paquete puede perderse o alterarse y por tanto, ser necesarias más retransmisiones.

4.4.4 Repetición selectiva (SR)

- Hay algunos escenarios en los que el propio GBN presenta problemas de rendimiento. En particular cuando el tamaño de la ventana y el producto ancho de banda-retardo son grandes puede haber muchos paquetes en el canal.
- En este caso, un único paquete erróneo podría hacer que el protocolo GBN retransmitiera una gran cantidad de paquetes, muchos de ellos de forma innecesaria. A medida que la probabilidad de errores en el canal aumenta, el canal puede comenzar a llenarse con estas retransmisiones innecesarias.
- Los protocolos de repetición selectiva evitan las retransmisiones innecesarias haciendo que el emisor únicamente retransmita aquellos paquetes que se sospeche que llegaron al receptor con error. Esta retransmisión individualizada y necesaria requerirá que el receptor conforme individualmente que paquetes ha recibido correctamente.
- El receptor de SR confirmará que un paquete se ha recibido correctamente tanto si se ha recibido en el orden correcto como si no. Los paquetes no recibidos en orden se almacenarían en el buffer hasta que se reciban los paquetes que faltan, momento en el que un lote de paquetes puede entregarse en orden a la capa superior.

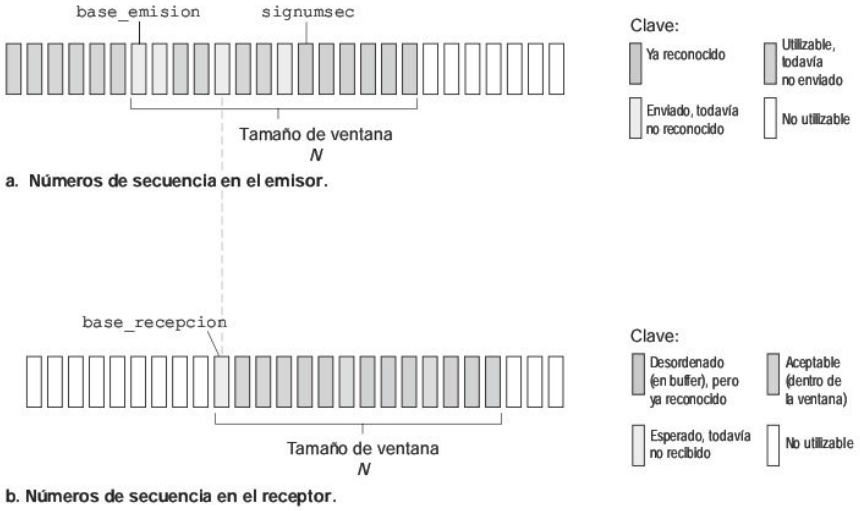


Figura 3.23 • Espacios de números de secuencia del lado emisor y del lado receptor en el protocolo de repetición selectiva (SR).

- El emisor y el receptor no siempre tienen una visión idéntica de lo que se ha recibido correctamente y de lo que no. En los protocolos SR, esto significa que las ventanas del emisor y del receptor no siempre coinciden.
- La falta de sincronización ente las ventanas del emisor y del receptor tiene consecuencias importantes cuando nos enfrentamos con la realidad de un rango finito de números de secuencia.

1. *Datos recibidos de la capa superior.* Cuando se reciben datos de la capa superior, el emisor de SR comprueba el siguiente número de secuencia disponible para el paquete. Si el número de secuencia se encuentra dentro de la ventana del emisor, los datos se empaquetan y se envían; en caso contrario, bien se almacenan en el buffer o bien se devuelven a la capa superior para ser transmitidos más tarde, como en el caso del protocolo GBN.
2. *Fin de temporización.* De nuevo, se emplean temporizadores contra la pérdida de paquetes. Sin embargo, ahora, cada paquete debe tener su propio temporizador lógico, ya que sólo se transmitirá un paquete al producirse el fin de la temporización. Se puede utilizar un mismo temporizador hardware para imitar el funcionamiento de varios temporizadores lógicos [Varghese 1997].
3. *ACK recibido.* Si se ha recibido un mensaje ACK, el emisor de SR marca dicho paquete como que ha sido recibido, siempre que esté dentro de la ventana. Si el número de secuencia del paquete es igual a `base_emision`, se hace avanzar la base de la ventana, situándola en el paquete no reconocido que tenga el número de secuencia más bajo. Si la ventana se desplaza y hay paquetes que no han sido transmitidos con números de secuencia que ahora caen dentro de la ventana, entonces esos paquetes se transmiten.

Figura 3.24 • Sucesos y acciones en el lado emisor del protocolo SR.

1. *Se ha recibido correctamente un paquete cuyo número de secuencia pertenece al intervalo* `[base_recepcion, base_recepcion+N-1]`. En este caso, el paquete recibido cae dentro de la ventana del receptor y se devuelve al emisor un paquete ACK selectivo. Si el paquete no ha sido recibido con anterioridad, se almacena en el buffer. Si este paquete tiene un número de secuencia igual a la base de la ventana de recepción (`base_recepcion` en la Figura 3.22), entonces este paquete y cualquier paquete anteriormente almacenado en el buffer y numerado consecutivamente (comenzando por `base_recepcion`) se entregan a la capa superior. La ventana de recepción avanza entonces el número de paquetes suministrados entregados a la capa superior. Por ejemplo, en la Figura 3.26, cuando se recibe un paquete con el número de secuencia `base_recepcion = 2`, éste y los paquetes 3, 4 y 5 pueden entregarse a la capa superior.
2. *Se ha recibido correctamente un paquete cuyo número de secuencia pertenece al intervalo* `[base_recepcion -N, base_recepcion -1]`. En este caso, se tiene que generar un mensaje ACK, incluso aunque ese paquete haya sido reconocido anteriormente por el receptor.
3. *En cualquier otro caso.* Ignorar el paquete.

Figura 3.25 • Sucesos y acciones en el lado receptor del protocolo SR.

Mecanismo	Uso, comentarios
Suma de comprobación (checksum)	Utilizada para detectar errores de bit en un paquete transmitido.
Temporizador	Se emplea para detectar el fin de temporización y retransmitir un paquete, posiblemente porque el paquete (o su mensaje ACK correspondiente) se ha perdido en el canal. Puesto que se puede producir un fin de temporización si un paquete está retardado pero no perdido (fin de temporización prematura), o si el receptor ha recibido un paquete pero se ha perdido el correspondiente ACK del receptor al emisor, puede ocurrir que el receptor reciba copias duplicadas de un paquete.
Número de secuencia	Se emplea para numerar secuencialmente los paquetes de datos que fluyen del emisor hacia el receptor. Los saltos en los números de secuencia de los paquetes recibidos permiten al receptor detectar que se ha perdido un paquete. Los paquetes con números de secuencia duplicados permiten al receptor detectar copias duplicadas de un paquete.
Reconocimiento (ACK)	El receptor utiliza estos paquetes para indicar al emisor que un paquete o un conjunto de paquetes ha sido recibido correctamente. Los mensajes de reconocimiento suelen contener el número de secuencia del paquete o los paquetes que están confirmando. Dependiendo del protocolo, los mensajes de reconocimiento pueden ser individuales o acumulativos.
Reconocimiento negativo (NAK)	El receptor utiliza estos paquetes para indicar al emisor que un paquete no ha sido recibido correctamente. Normalmente, los mensajes de reconocimiento negativo contienen el número de secuencia de dicho paquete erróneo.
Ventana, procesamiento en cadena	El emisor puede estar restringido para enviar únicamente paquetes cuyo número de secuencia caiga dentro de un rango determinado. Permitiendo que se transmitan varios paquetes aunque no estén todavía reconocidos, se puede incrementar la tasa de utilización del emisor respecto al modo de operación de los protocolos de parada y espera. Veremos brevemente que el tamaño de la ventana se puede establecer basándose en la capacidad del receptor para recibir y almacenar en buffer los mensajes, o en el nivel de congestión de la red, o en ambos parámetros.

Tabla 3.1 • Resumen de los mecanismos para la transferencia de datos fiable y su uso.

4.5 Transporte orientado a la conexión: TCP [RFC 793], [RFC 1122], [RFC 1323], [RFC 2018], [RFC 2581]

- Para proporcionar una transferencia de datos fiable, TCP confía en muchos de los principios expuestos en la sección anterior, incluyendo los mecanismos de detección de errores, las retransmisiones, los reconocimientos acumulativos, los temporizadores y los campos de cabecera para los números de secuencia y reconocimiento.

4.5.1 La conexión TCP

- Se dice que TCP **está orientado a la conexión** porque antes de que un proceso de la capa aplicación pueda comenzar a enviar datos a otro, los dos procesos deben primero establecer una comunicación entre ellos, es decir, tienen que enviarse ciertos segmentos preliminares para definir los parámetros de la transferencia de datos que van a llevar a cabo a continuación. Como parte del proceso de establecimiento de la conexión TCP asociadas con la conexión TCP.
- Dado que el protocolo TCP se ejecuta únicamente en los sistemas terminales y no en los elementos intermedios de la red (routers y switches), los elementos intermedios de la red no mantienen el estado de la conexión TCP.
- Una conexión TCP proporciona un **servicio full-duplex**.
- Una conexión TCP casi siempre es una conexión punto a punto. La multidifusión, la transferencia de datos dese un emisor a muchos receptores en una única operación, no es posible con TCP.
- En el establecimiento de una conexión TCP:
  - El proceso que inicia la conexión es el proceso cliente y el otro el proceso servidor.
  - El proceso de la aplicación cliente informa en primer lugar a la capa de transporte del cliente que desea establecer una conexión con un proceso del servidor.

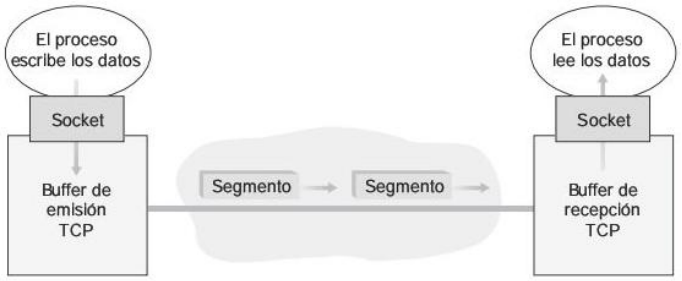


Figura 3.28 • Buffers de emisión y recepción de TCP.

  - La capa de transporte del cliente procede entonces a establecer una conexión con un proceso del servidor.
  - La capa de transporte del cliente procede entonces a establecer una conexión TCP con el TCP del servidor.
  - El cliente envía primero un segmento especial; el servidor responde con un segundo segmento TCP especial y por último el cliente responde de nuevo con un tercer segmento especial.
  - Los dos primeros segmentos no transportan datos de la capa de aplicación; el tercero de estos segmentos es el que puede llevar la carga útil.
  - Puesto que los tres segmentos son intercambiados entre dos host, este procedimiento suele denominarse **acuerdo en tres fases** (T***hree Way Handshake***).

- Una vez que se ha establecido la conexión TCP, los dos procesos de aplicación pueden enviarse datos el uno al otro.
- Consideremos la transmisión de datos desde el proceso cliente al proceso servidor:
  - El proceso cliente pasa un flujo de datos a través del socket (la puerta del proceso).
  - Una vez que los datos atraviesan la puerta, se encuentran en manos del protocolo TCP que se ejecuta en el cliente.
  - TCP dirige estos datos al **buffer de emisión** de la conexión, que es uno de los buffers que se definen durante el proceso inicial del acuerdo en tres fases.
  - De vez en cuando, TCP tomará fragmentos de datos del buffer de emisión.
  - La cantidad máxima de datos que pueden cogerse y colocarse en un segmento está limitado por el **tamaño máximo de segmento (MMS, *Máximum Segment Size*)**. Normalmente, el MMS queda determinado en primer lugar por la longitud de la trama más larga de la capa de enlace que el host emisor local puede enviar, que es la **unidad máxima de transmisión (MTU, *maximun Transmision Unit*)** y luego el MSS se establece de manera que se garantice que un segmento TCP, cuando se encapsula un datagrama, se ajuste a una única trama de la capa de enlace.

- Valores comunes de MTU son 1460 bytes, 536 bytes y 512 bytes.
- Observe que el MSS es la cantidad máxima de datos de la capa de aplicación en el segmento, no el tamaño máximo del segmento TCP incluyendo las cabeceras.
- TCP empareja cada fragmento de datos del cliente con una cabecera TCP, formando **segmentos TCP**.
- Los segmentos se pasan a la capa de red, donde son encapsulados por separado dentro de datagramas IP de la capa de red.
- Los datagramas se envían entonces a la red. Cundo TCP recibe un segmento en el otro extremo, los datos del mismo se colocan en el buffer de recepción de la conexión TCP.
- La aplicación lee el flujo de datos de este buffer. Cada lado de la conexión tiene su propio buffer de emisión y su propio buffer de recepción.

4.5.2 Estructura del segmento TCP

- El segmento TCP consta de campos de cabecera y un campo de datos.
- El campo de datos contiene un fragmento de los datos de la aplicación.
- Cuando TCP envía un archivo grande, normalmente divide el archivo en fragmentos de tamaño MSS. Sin embargo, las aplicaciones interactivas suelen transmitir fragmentos de datos que son más pequeños que el MSS.
- Al igual que con UDP, la cabecera incluye los **número de puerto de origen y de destino**.
- También, al igual que UDP, la cabecera incluye un **campo de suma de comprobación**.
- La cabecera de un segmento TCP también contiene los siguientes campos:

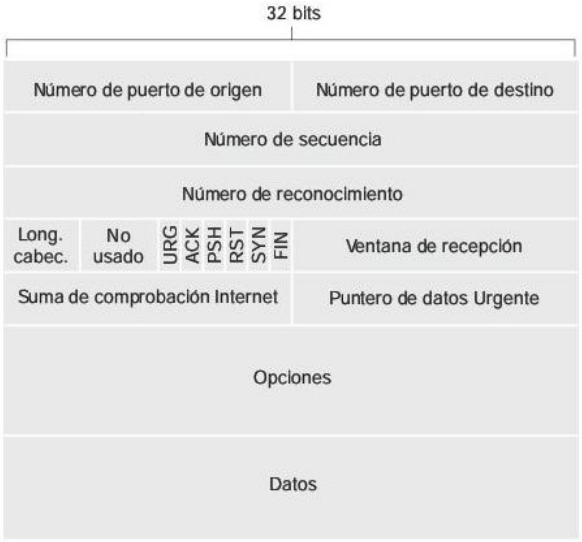


Figura 3.29 • Estructura del segmento TCP.

  - El campo **número de secuencia** y el campo **número de reconocimiento**. Son utilizados por el emisor y el receptor de TCP para implementar un servicio de transferencia fiable.
  - El campo **ventana de recepción** se utiliza para el control de flujo.
  - El campo **longitud de cabecera** de 4 bits, específica la longitud de la cabecera TCP en palabras de 32 bits. La cabecera TCP puede tener una longitud variable a causa del campo opciones de TCP.
  - El campo **opciones** es opcional y de longitud variable. Se utiliza cuando un emisor y un receptor negocian el tamaño de máximo de segmento (MSS) o como un factor de escala de la ventana en las redes de alta velocidad.
  - El campo indicador tiene 6 bits:
    - El bit **ACK**, se utiliza para indicar que el valor transportado en el campo de reconocimiento es válido, es decir, el segmento contiene un reconocimiento para un segmento que ha sido recibido correctamente.
    - Los bits **RST**, **SYN** y **FIN** se utilizan para el establecimiento y cierre de conexiones.
    - La activación del bit **PSH** indica que el receptor deberá pasar los datos a la capa superior de forma inmediata.
    - Por último, el bit **URG** se utiliza para indicar que hay datos en este segmento que la entidad de la capa superior del lado emisor ha marcado como urgente.
    - La posición de este último byte de estos datos urgente se indica mediante el campo **puntero de datos urgentes**.

Números de secuencia y números de reconocimiento

- Estos campos son una parte del servicio de transferencia de datos fiable de TCP.
- TCP percibe los datos como un flujo de bytes no estructurado pero ordenado.
- Los números de secuencia hacen referencia al flujo de bytes transmitido y no a la serie de segmentos transmitidos. El número de secuencia es por tanto el número del primer byte del segmento dentro del flujo de bytes.
- Consideremos ahora los números de reconocimiento, que son algo más complicados. Recordemos que TCP es una conexión **full dúplex**, de modo que el host A puede estar recibiendo datos del host B mientras envía datos al host B.
- Todos los segmentos que llegan procedentes del host B tienen un número de secuencia para los datos que fluyen de B a A. El número de reconocimiento que el host A incluye en su segmento es el número de secuencia del siguiente byte que el host A está esperando del host B.

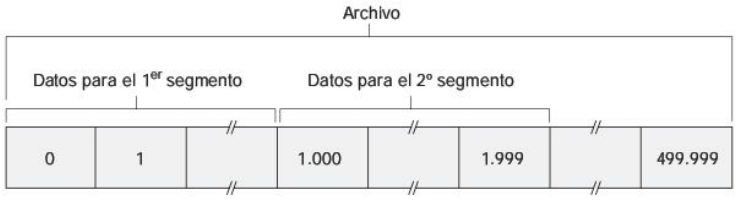


Figura 3.30 • División de los datos del archivo en segmentos TCP.

- Dado que TCP sólo confirma los bytes hasta el primer byte que falta en el flujo, se dice que TCP proporciona **reconocimientos acumulativos**.
- ¿Qué hace un host cuando no recibe los segmentos en orden a través de una conexión TCP?:
    1. El receptor descarta de forma inmediata los segmentos que no han llegado en orden.
    2. El receptor mantiene los bytes no ordenados y espera a que lleguen los bytes que faltan con el fin de rellenar los huecos.

Telnet: caso de estudio de los números de secuencia y de reconocimiento

- Telnet, definido en el documento [RFC 854], es un popular protocolo de la capa de aplicación utilizado para los inicios de sesión remotos.
- Se ejecuta sobre TCP y está diseñado para trabajar entre cualquier pareja de host.
- Telnet es una aplicación interactiva y los datos enviados a través de una conexión Telnet no están cifrados, lo que hace que sea vulnerable a los ataques de sniffer.
- Supongamos que el host A inicia una sesión Telnet con el host B:
    - Puesto que el host A inicia la sesión, se etiqueta como el cliente y el host B como el servidor.
    - Cada carácter escrito por el usuario se

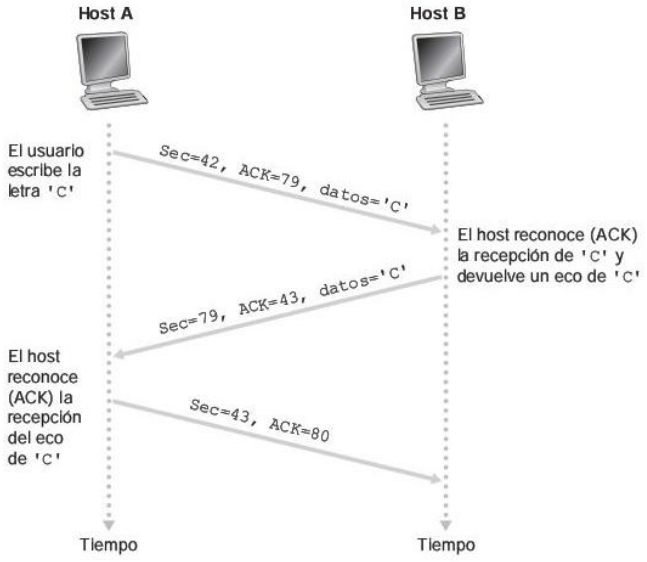


Figura 3.31 • Números de secuencia y de reconocimiento en una aplicación Telnet simple sobre TCP.

enviará al host remoto y éste devolverá una copia de cada carácter, que será mostrada en la pantalla del usuario.

- Este eco se emplea para garantizar que los caracteres vistos por el usuario ya han sido recibidos y procesados en el sitio remoto.
- Por tanto cada carácter atraviesa la red dos veces.
- La Figura 3.31 resume el escenario de escribir una única letra 'C', suponiendo que los números de secuencia inicales entre cliente y servidor son respectivamente 42 y 79.

4.5.3 Estimación del tiempo de ida y vuelta y fin de temporización

- TCP utiliza un mecanismo de fin de temporización/retransmisión para recuperarse de la pérdida de segmentos.
- El intervalo de fin de temporización debería ser mayor que el tiempo de ida y vuelta (**RTT**, **Round-Trip delay Time**) de la conexión, si fuera de otra manera, se enviarían retransmisiones innecesarias.

Estimación del tiempo de ida y vuelta

- El RTT de muestra, expresado como `RTTMuestra`, para un segmento es la cantidad de tiempo que transcurre desde que se envía el segmento (se pasa a IP) hasta que se recibe el correspondiente paquete de reconocimiento del segmento.
- En lugar de medir `RTTMuestra` para cada segmento transmitido, la mayor parte de las implementaciones TCP usan un algoritmo de retransmisión que monitoriza el retardo en cada conexión y ajusta el valor de RTO (*Retransmisison Time Out*) de acuerdo con ese valor.

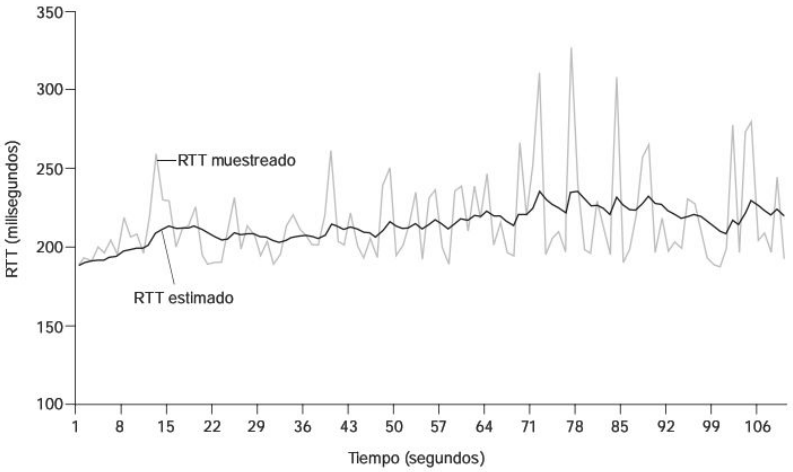


Figura 3.32 • Muestreo de RTT y estimación de RTT.

- La especificación del protocolo sugiere tomar muestras del tiempo de ida y vuelta, RTT, calculado como la diferencia de tiempo entre la emisión de un segmento y la recepción de su reconocimiento. Con esta información, TCP puede ajustar dinámicamente una variable que identifique el tiempo medio de ida y vuelta, de la siguiente forma:

$$RTT_{Estimado} = (1-\alpha) * RTT_{Estimado_{anterior}} + \alpha * RTT_{Muestra\_nuevo}$$

- El valor recomendado para α es 0.125 es decir $1/8$ [RFC 2988].
- En estadística, una media como esta se denomina **Media móvil exponencialmente ponderada** (**EWNA, *Exponential Weighted Moving Average***). El término exponencial aparece en EWMA porque el peso de un valor dado de `RTTMuestra` disminuye exponencialmente tan rápido como tienen lugar las actualizaciones.
- También es importante disponer de una medida de la variabilidad de RTT. [RFC 2988] define la variación de RTT, `RTTDev`, como una estimación de cuanto se desvía típicamente `RTTMuestra` de `RTTEstimado`:

$$RTT_{Desv} = (1-\beta) * RTT_{Desv} + \beta * |RTT_{Muestra} - RTT_{Estimado}|$$

- Si los valores de `RTTMuestra` presentan una pequeña fluctuación, entonces `RTTDev` será pequeño, por el contrario, si existe una gran fluctuación, `RTTDev` será grande.
- El valor recomendado para β es 0.25.

Definición y gestión del intervalo de fin de temporización para la retransmisión

- Evidentemente el intervalo tendrá que ser mayor o igual que `RTTEstimado` o se producirán retransmisiones innecesarias.
- Tampoco debería ser mucho mayor que `RTTEstimado` pues si un segmento se pierde, TCP no retransmitirá rápidamente el segmento, provocando retardos largos en la transferencia de datos.
- Por tanto, es deseable hacer el intervalo de fin de temporización igual a `RTTEstimado` más un cierto margen. El margen deberá ser más grande cuando la fluctuación en los valores de `RTTEstimado` sea grande y más pequeño en caso contrario.

$$IntervaloFinDeTemporizacion = RTT_{Estimado} + 4 * RTT_{Desv}$$

4.5.4 Transferencia de datos fiable

- TCP crea un **servicio de transferencia de datos fiable** sobre el servicio de mejor esfuerzo pero no fiable de IP. El servicio de transferencia fiable de datos de TCP garantiza que el flujo de datos que un proceso extrae de su buffer de recepción TCP no está corrompido, no contiene huecos, ni duplicados y está en orden.
- Los procedimientos de gestión del temporizador TCP recomendados [RFC 2988] utilizan un único temporizador de retransmisión, incluso aunque haya varios segmentos transmitidos y aun no reconocidos.
- En la siguiente exposición suponemos que solo se están enviando datos en una solo dirección, del host A al host B y que el host A está enviando un archivo grande.
- La Figura 3.33 presenta una descripción simplificada de un emisor TCP.

```
/* Suponga que el emisor no está restringido por los mecanismos de
control de flujo o de control de congestión de TCP, que el tamaño de
los datos procedentes de la capa superior es menor que el MSS y además
la transferencia de datos tiene lugar en un único sentido. */

SigNumSec = NumeroSecuenciaInicial
BaseEmision = NumeroSecuenciaInicial

loop (siempre) {
    switch(suceso)

        suceso: datos recibidos de la aplicación de la capa superior
            crear segmento TCP con número de secuencia SigNumSec
            if (el temporizador no se está ejecutando actualmente)
                iniciar temporizador
            pasar segmento a IP
            SigNumSec = SigNumSec+longitud(datos)
            break;

        suceso: fin de temporización del temporizador
            retransmitir el segmento aun no reconocido con
                el número de secuencia más pequeño
            iniciar temporizador
            break;

        suceso: ACK recibido, con valor de campo ACK igual a y
            if (y > BaseEmision) {
                BaseEmision = y
                if (existen actualmente segmentos aun no reconocidos)
                    iniciar temporizador
                }
            break;

    } /* fin del bucle siempre */
```

Figura 3.33 • Emisor TCP simplificado.

- Al producirse el primero de los sucesos más importantes TCP recibe datos de la aplicación, encapsula los datos en un segmento y pasa el segmento a IP. El intervalo de caducidad para este temporizador es `IntervaloFinDeTemporización`.
- El segundo suceso importante es el fin de temporización. TCP responde a este suceso retransmitiendo el segmento de ha causado el fin de la temporización y a continuación, reinicia el temporizador.
- El tercer suceso que tiene que gestionar el emisor TCP es la llegada de un segmento de reconocimiento ACK procedente del receptor.
  Al ocurrir este suceso, TCP compara el valor ACK `y` con su variable `BaseEmision`. La variable de estado TCP `BaseEmision` es el número de secuencia del byte de reconocimiento más antiguo de modo que `y` confirma la recepción de todos los bytes anteriores al número de byte `y`.
  Si `y >BaseEmision`, entonces el ACK está confirmando uno o más de los segmentos no reconocidos anteriores. Así, el emisor actualiza su variable `BaseEmision` y reinicia el temporizador si actualmente aún existen segmentos no reconocidos.

Algunos escenarios interesantes

- Primer escenario:

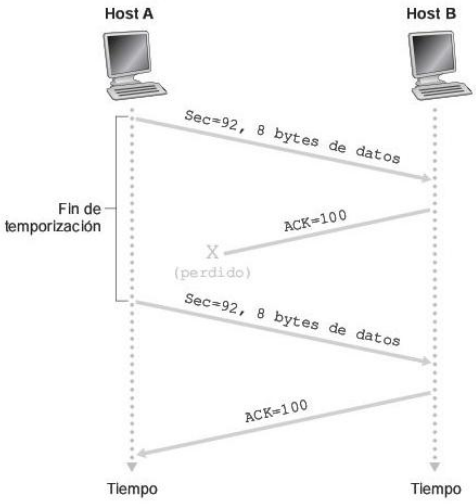


Figura 3.34 • Retransmisión debida a la pérdida de un paquete de reconocimiento ACK.

- Segundo escenario:

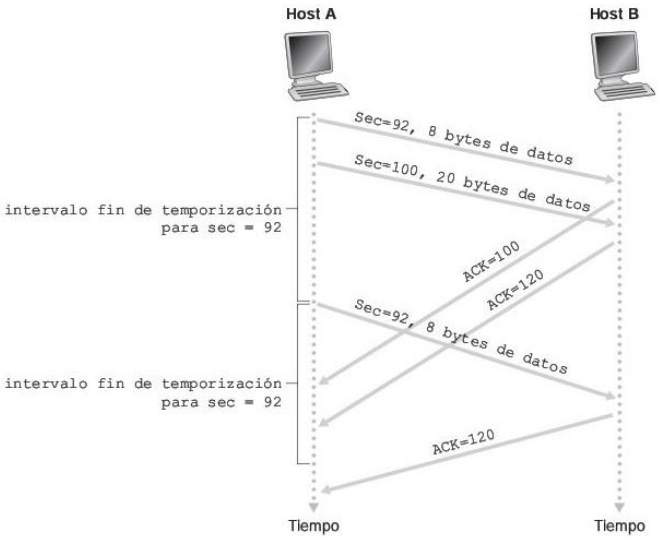


Figura 3.35 • Segmento 100 no retransmitido.

- Tercer escenario:

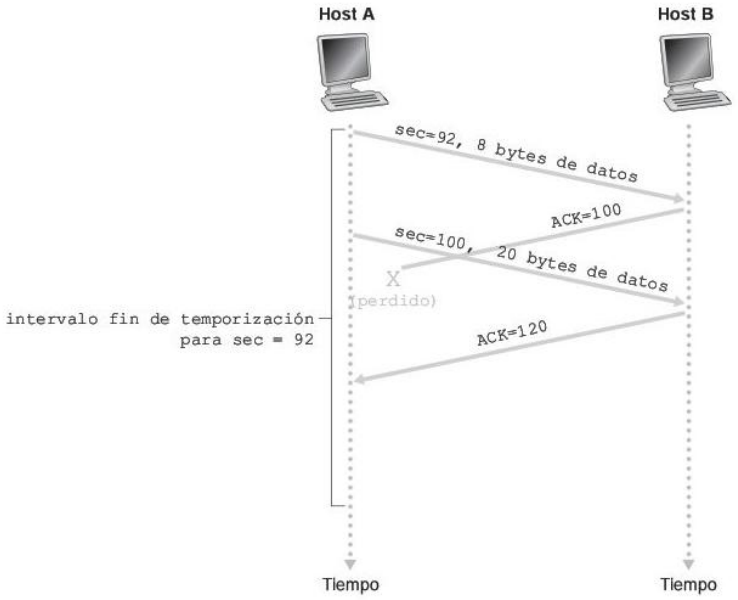


Figura 3.36 • Un reconocimiento acumulativo evita la retransmisión del primer segmento.

Duplicación del intervalo de fin de temporización

- Examinemos ahora algunas de las modificaciones que aplican la mayor parte de las implementaciones TCP.
- La primera de ellas está relacionada con la duración del intervalo de fin de temporización después de que el temporizador ha caducado.
- En esta modificación, cuando tiene lugar un suceso de fin de temporización TCP retransmite el segmento aun no reconocido con el número de secuencia más pequeño, como se ha descrito anteriormente. Pero cada vez que TCP retrasmite, define el siguiente intervalo de fin de temporización como dos veces el valor anterior, en lugar de obtenerlo a partir de los últimos valores de `RTTEstimado` y `RTTDesv`.
- Sin embargo, cuando el temporizador se inicia después de cualquiera de los otros dos sucesos (datos recibidos de aplicación y recepción de un ACK), el `IntervaloFindeTemporizacion` se obtiene a partir de los valores más recientes de `RTTEstimado` y `RTTDesv`.
- Estas modificaciones proporcionan una forma limitada de control de congestión.
- La caducidad del temporizador está causada muy probablemente por la congestión en la red. Cuando existe congestión, si los orígenes continúan retransmitiendo paquetes persistentemente, la congestión puede empeorar. En lugar de ello, TCP actúa de forma más diplomática, haciendo que los emisores retransmitan después de intervalos cada vez más grandes.

Retransmisión rápida

- Uno de los problemas con las retransmisiones generadas por los sucesos de fin de temporización es que el período de fin de temporización puede ser relativamente largo.
- Afortunadamente, el emisor puede a menudo detectar la pérdida de paquetes antes de que tenga lugar el suceso de fin de temporización, observando los ACK duplicados.
- Un **ACK duplicado** es un ACK que vuelve a reconocer un segmento para el que el emisor ya ha recibido un reconocimiento anterior.
- La Tabla 3.2 resume la política de generación de mensajes ACK en el receptor TCP [RFC 1122], [RFC 2581]:

Suceso	Acción del receptor TCP
Llegada de un segmento en orden con el número de secuencia esperado. Todos los datos hasta el número de secuencia esperado ya han sido reconocidos.	ACK retardado. Esperar hasta durante 500 milisegundos la llegada de otro segmento en orden. Si el siguiente segmento en orden no llega en este intervalo, enviar un ACK.
Llegada de un segmento en orden con el número de secuencia esperado. Hay otro segmento en orden esperando la transmisión de un ACK.	Enviar inmediatamente un único ACK acumulativo, reconociendo ambos segmentos ordenados.
Llegada de un segmento desordenado con un número de secuencia más alto que el esperado. Se detecta un hueco.	Enviar inmediatamente un ACK duplicado, indicando el número de secuencia del siguiente byte esperado (que es el límite inferior del hueco).
Llegada de un segmento que completa parcial o completamente el hueco existente en los datos recibidos.	Enviar inmediatamente un ACK, suponiendo que el segmento comienza en el límite inferior del hueco.

Tabla 3.2 • Recomendación para la generación de mensajes ACK en TCP

- Cuando un receptor TCP recibe un segmento con un número de secuencia que es mayor que el siguiente número de secuencia en orden esperado, detecta un hueco en el flujo de datos, es decir, detecta que falta un segmento.
- Este hueco podría ser el resultado de segmentos perdido o reordenados dentro de la red. Dado que TCP no utiliza paquetes NAK, el receptor no puede devolver al emisor un mensaje de reconocimiento negativo explícito. En su lugar, simplemente vuelve a reconocer (genera un ACK duplicado) al último byte de datos en orden que ha recibido.
- Puesto que un emisor suele enviar un gran número de segmentos seguidos, si se pierde uno de ellos, probablemente habrá muchos ACK duplicados seguidos.
- Si el emisor recibe tres ACK duplicados para los mismos datos, toma esto como una indicación de que el segmento que sigue al segmento que ha sido reconocido tres veces se ha perdido.

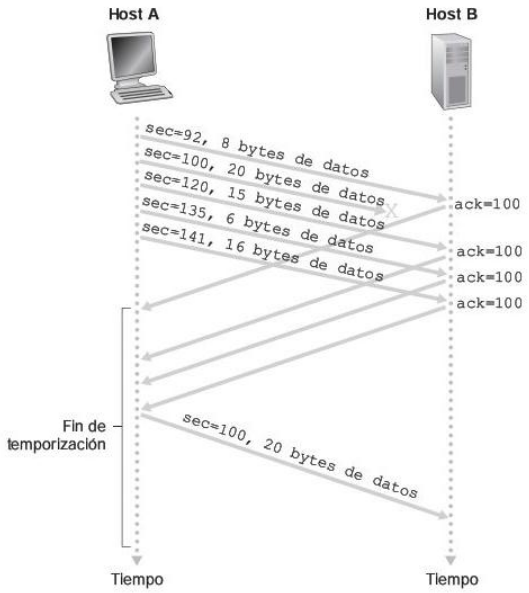


Figura 3.37 • Retransmisión rápida: retransmisión del segmento que falta antes de que caduque el temporizador del segmento.

- En el caso de que se reciban tres ACK duplicados, el emisor TCP realiza una **retransmisión rápida** [RFC 2581], reenviando el segmento que falta antes de que caduque el temporizador de dicho segmento.
- Para TCP con retransmisión rápida, el siguiente fragmento de código reemplaza al suceso de recepción de un ACK de la Figura 3.33:

```
suceso: ACK recibido, con un valor de campo ACK de y
          if (y > BaseEmision) {
              BaseEmision = y
              if (existen segmentos pendientes de reconocimiento)
                  iniciar temporizador
              }
          else { /* un ACK duplicado para un segmento ya reconocido */
             incrementar el número de mensajes ACK duplicados
                recibidos para y
             if (número de ACK duplicados recibidos para y==3)
                /* TCP con retransmisión rápida */
                reenviar el segmento con número de secuencia y
             }
                    break;
```

- Los procedimientos anteriores, se han desarrollado como resultado de más de 15 años de experiencia con los temporizadores TCP.

Retroceder N o repetición selectiva

- Recuerde que los reconocimientos de TCP son acumulativos y que los segmentos correctamente recibidos pero no en orden no son reconocidos individualmente por el receptor. En consecuencia, el emisor TCP solo necesitara mantener el número de secuencia más pequeño de un byte transmitido pero no reconocido (`BaseEmisión`) y el número de secuencia del siguiente byte que va a enviar (`SigNumSec`). En este sentido, TCP se parece mucho a un protocolo de tipo GBN, no obstante, existen algunas diferencias.
- Muchas implementaciones de TCP almacenan en buffer los segmentos recibidos correctamente pero no en orden. Una modificación propuesta para TCP es lo que se denomina **reconocimiento selectivo** [RFC 2018], que permite a un receptor TCP reconocer segmentos no ordenados de forma selectiva, en lugar de solo hacer reconocimientos acumulativos del último segmento recibido correctamente y en orden.
- Cuando se combina con la retransmisión selectiva, TCP se comporta como el protocolo SR selectivo. Por tanto, el mecanismo de recuperación de errores de TCP probablemente es mejor considerarlo como un hibrido de los protocolos GBN y SR.

4.5.5 Control de flujo

- TCP proporciona un **servicio de control de flujo** a sus aplicaciones para eliminar la posibilidad de que el emisor desborde el buffer del receptor. El control de flujo es por tanto un servicio de adaptación de velocidades.
- Un emisor TCP también puede atascarse debido a la congestión de la red IP, esta forma de control del emisor se define como un mecanismo de **control de congestión**. Por lo tanto, servicio de control de flujo ≠control de congestión.
- Vamos a suponer que la implementación de TCP es tal que el receptor TCP descarta los segmentos que no llegan en orden.

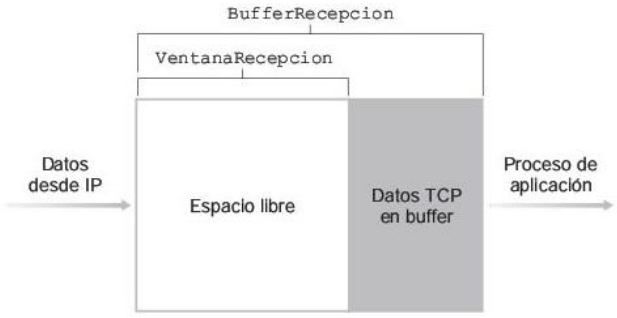


Figura 3.38 • La ventana de recepción y el buffer de recepción (`BufferRecepcion`).

- TCP proporciona un servicio de control de flujo teniendo que mantener el emisor una variable conocida como **ventana de recepción**.
- Informalmente, la ventana de recepción se emplea para proporcionar al emisor una idea de cuánto espacio libre hay disponible en el buffer del receptor.
- Puesto que TCP es una conexión full-duplex, el emisor de cada lado de la conexión mantiene una ventana de recepción diferente.
- Supongamos que el un host A está enviando un un archivo grande al host B a través de una conexión TCP. El host B asigna un buffer de recepción a estqa conexión; designamos al tamaño de este buffer como `BufferRecepcion`.
- De vez en cuando, el proceso de aplicación del host B lee el contenido del buffer. Definimos las siguiente variables:
  - `UltimoByteLeido`
  - `UltimoByteRecibido`

- Puesto que en TCP no está permitido desbordar el buffer asignado, tenemos que:

$$\texttt{UltimoByteRecibido} - \texttt{UltimoByteLeido} \leq \texttt{BufferRecepcion}$$

- La ventana de recepción se hace igual a:

```
VentanRecepcion = BufferRecepcion - [UltimoByteRecibido – UltimoByteLeido]
```

- El host B dice al host A la cantidad de espacio disponible que hay en el buffer de la conexión almacenando el valor actual de `VentanRecepcion` en el campo ventana de recepción de cada segmento que envía a A.
- Inicialmente, el host B establece que `VentanRecepcion = BufferRecepcion`.
- A su vez el host A controla dos variables y cuya diferencia es la cantidad de datos no reconocidos que el host A ha enviado a través de la conexión:

$$\texttt{UltimoByteEnviado} - \texttt{UltimoByteReconocido}$$

- Haciendo que el número de datos no reconocidos sea inferior al valor de `VentanaRecepción`, el host A podrá asegurarse de no estar desbordando el buffer de recepción en el host B:

$$\texttt{UltimoByteEnviado} - \texttt{UltimoByteReconocido} \leq \texttt{VentanaRecepción}$$

- Existe un problema técnico que se da cuando `VentanaRecepción = 0` y el host B no tiene nada que enviar al host A.
- Para resolver este problema, la especificación TCP requiere al host A que continúe enviando segmentos con un byte de datos cuando la longitud de la ventana de recepción de B es cero. Estos segmentos serán reconocidos por el receptor. Finalmente, el buffer comenzará a vaciarse y los ACK contendrán un valor de `VentanaRecepción` distinto de cero.

4.5.6 Gestión de la conexión TCP

- Vamos a ver como se establece y termina una conexión TCP.
- En primer lugar, veamos cómo se establece la conexión, suponiendo que hay un proceso en ejecución en un host cliente que desea iniciar una conexión con otro proceso que se ejecuta en otro host servidor:
  - Paso 1. TCP del lado cliente envía un segmento TCP especial al TCP del lado servidor. Este segmento especial no contiene datos de la capa de aplicación. Pero uno de los bits indicadores de la cabecera del segmento, el bit SYN, se pone a 1. Además, el cliente selecciona de forma aleatoria un número de secuencia inicial (`cliente_nsi`) y lo coloca en el campo número de secuencia del segmento TCP. Este segmento se encapsula dentro de un datagrama IP y se envía al servidor.
  - Paso2. Una vez que el datagrama IP que contiene el segmento **SYN TCP** llega al host servidor, el servidor extrae dicho segmento del datagrama, asigna los buffers y variables TCP a la conexión y envía un segmento de conexión concedida al cliente TCP.

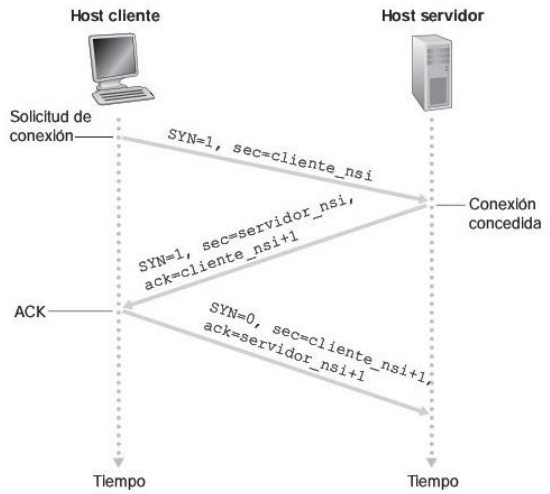


Figura 3.39 • El proceso de acuerdo en tres fases de TCP: intercambio de segmentos.

Este segmento de conexión concedida tampoco contiene datos de la capa de aplicación. Sin embargo, contiene tres fragmentos de información importantes de la cabecera del segmento:

I. El bit SYN se pone a 1.
II. El campo de reconocimiento se hace igual `cliente_nsi + 1`.
III. El servidor almacena en el campo número de secuencia el valor (`servidor_nsi`).

El segmento de conexión concedida se conoce como **segmento SYNACK**.

- Paso 3. Al recibir el segmento SYNACK, el cliente también asigna buffers y variables a la conexión. El host cliente envía entonces al servidor otro segmento de confirmación (el cliente hace esto almacenando el valor `servidor_nsi` en el campo de reconocimiento). El bit SYN se pone a cero.
  Esta tercera etapa del proceso de acuerdo en tres fases puede transportar datos del cliente al servidor dentro de la carga útil del segmento.

- Una vez completados estos tres pasos, los host cliente y servidor pueden enviarse segmentos que contengan datos el uno al otro.
- En cada uno de estos segmentos futuros, el valor del bit SYN será cero.
- Este procedimiento de establecimiento de la conexión suele denominarse proceso de **acuerdo en tres fases** (*Three Way Handsake*).
- Cualquiera de los dos procesos participantes en una conexión TCP pueden dar por terminada dicha conexión. Cuando una conexión se termina, los recursos (buffers y variables) de los host se liberan.
- Supongamos que el cliente decide cerrar la conexión:
  - Éste ejecuta un comando de cierre, haciendo que el cliente TCP envíe un segmento especial TCP al proceso servidor.
  - El segmento especial contiene un bit indicador en la cabecera del segmento, el bit FIN, puesto a 1.
  - Cuando el servidor recibe este segmento, devuelve al cliente un segmento de reconocimiento. El servidor entonces envía su propio segmento de desconexión, que tiene el bit FIN puesto a 1.
  - Por último, el cliente reconoce el segmento de desconexión del servidor. En esta situación, los recursos de ambos host quedan liberados.

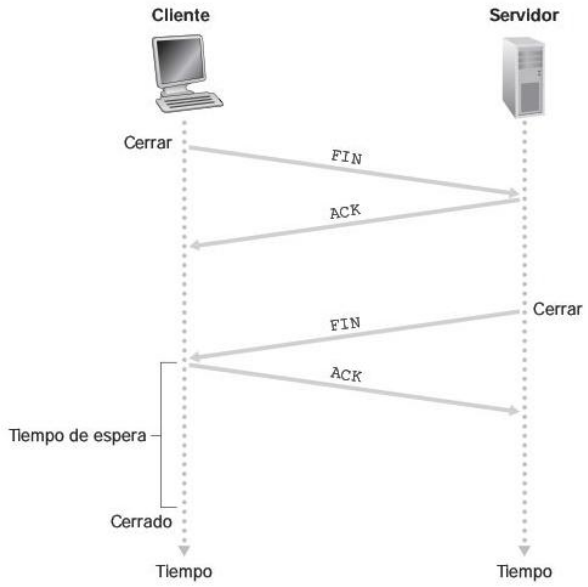


Figura 3.40 • Cierre de una conexión TCP.

- Mientras se mantiene una conexión TCP, el protocolo TCP que se ejecuta en cada host realiza transiciones a través de los diversos **estados TCP**:

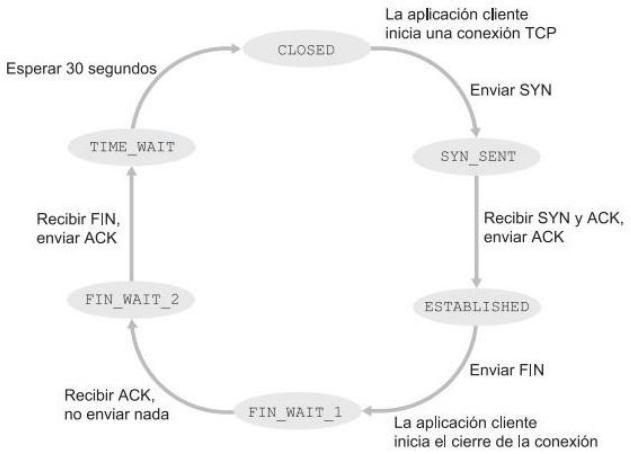


Figura 3.41 • Secuencia típica de estados TCP visitados por un cliente TCP.

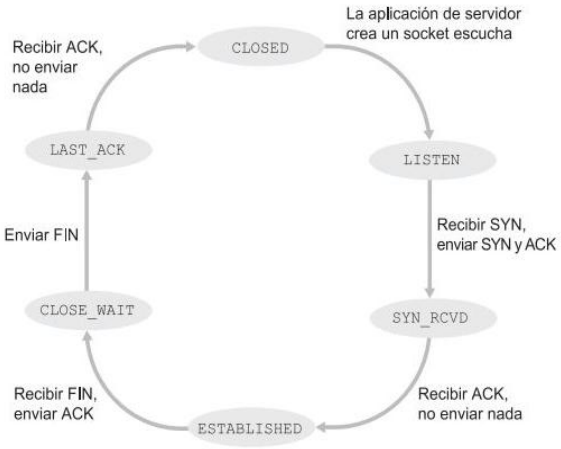


Figura 3.42 • Secuencia típica de estados TCP visitados por un servidor TCP.

- Hasta aquí, se ha supuesto que tanto el cliente como el servidor están preparados para comunicarse; es decir, que el servidor está escuchando en el puerto al que el cliente envía su segmento SYN. Consideremos lo que ocurre cuando un host recibe un segmento TCP cuyo número de puerto o cuya dirección IP de origen no se corresponde con ninguno de los sockets activos en el host.
- En este caso, el host enviará al origen un segmento especial de reinicio. Este segmento TCP tiene el bit indicador RST puesto a 1.

4.6 Principios de control de congestión

- La retransmisión de paquetes se ocupa de un síntoma de la congestión de la red, pero no se ocupa de la causa de esa congestión.
- 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.

4.6.1 Las causas y los costes de la cogestión

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

- Veamos el escenario de congestión más simple: dos host (A y B), cada uno de los cuales dispone de una conexión que comparte un único salto entre el origen y el destino:

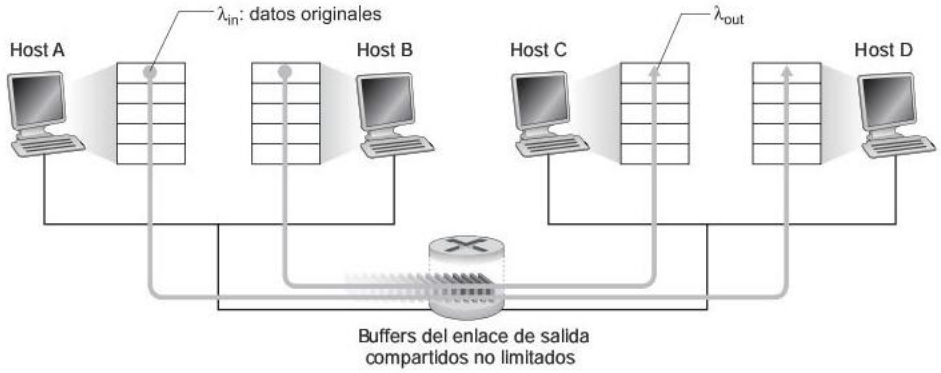


Figura 3.43 • Escenario de congestión 1: dos conexiones que comparten un único salto con buffers de capacidad ilimitada.

- Supongamos que la aplicación del host A está enviando datos a la conexión a una velocidad media de λ_{in} bytes/segundo.

- El protocolo de nivel de transporte subyacente es un protocolo simple. Los datos se encapsulan y se envían; no existe un mecanismo de recuperación de errores, ni se realiza un control de flujo, ni un control de congestión.
- El host B opera de forma similar y también está enviando datos a una velocidad de λ_{in} bytes/segundo.
- Los paquetes que salen de los host 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 gráfica 3.44 muestra el rendimiento de la conexión del host A:

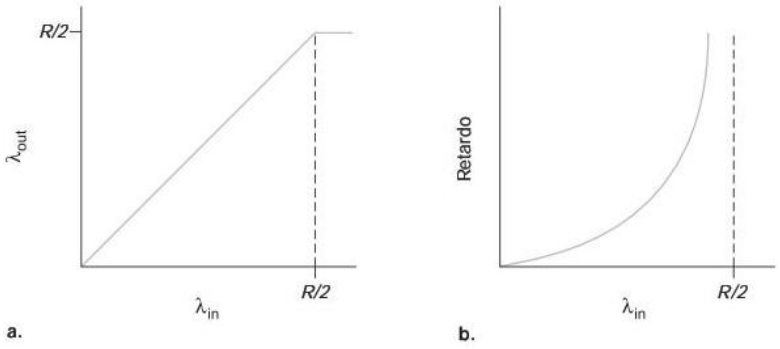


Figura 3.44 • Escenario de congestión1: tasa de transferencia y retardo en función de la velocidad de transmisión del host.

- La gráfica de la izquierda muestra la **tasa de transferencia por conexión** como una función de la velocidad de transmisión de la conexión.
- La gráfica de la derecha muestra la consecuencia de operar cerca de la capacidad del enlace. 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 desde el punto de vista del retardo.
- Incluso en un escenario idealizado, ya hemos encontrado unos 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 capa del enlace.

Escenario 2: dos emisores y un router con buffers finitos

- Ahora vamos a modificar ligeramente el escenario 1 de dos formas:

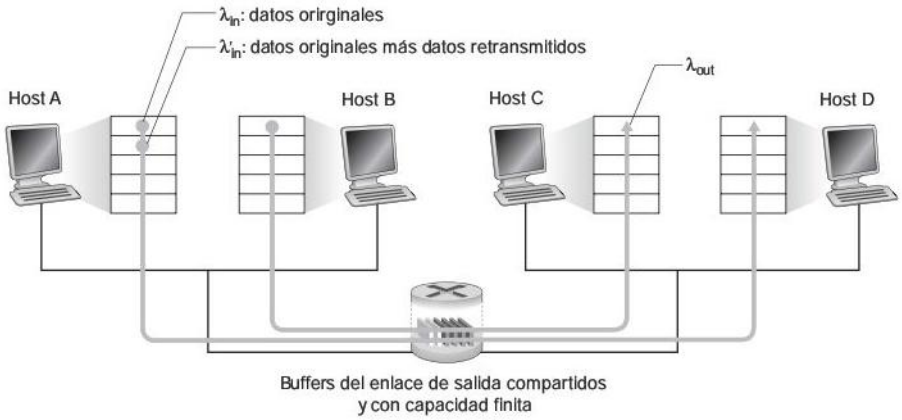


Figura 3.45 • Escenario 2: dos hosts (con retransmisión) y un router con buffers con capacidad finita.

- En primer lugar, suponemos que el espacio disponible en los buffers del router es finito.
- En segundo lugar, suponemos que cada conexión es fiable.
- 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 (datos originales + datos retransmitidos) a la red la denotaremos como λ'_{in} bytes/segundo. En ocasiones λ'_{in} se denomina **carga ofrecida**.
- El rendimiento de este segundo escenario dependerá en gran medida de cómo se realicen las transmisiones.
- En primer lugar, considere el caso no realista en el que el host A es capaz de determinar de alguna manera 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 producirá ninguna pérdida, λ_{in} será igual a λ'_{in} y la tasa de transferencia de la conexión sería igual a λ_{in} (Figura 3.46 a).

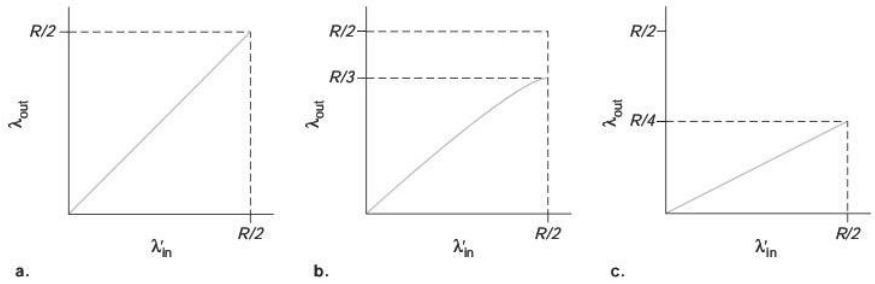


Figura 3.46 • Escenario 2: rendimiento con buffers de capacidad finita.

- Desde el punto de vista de la transferencia, el rendimiento es ideal: todo lo que se envía se recibe.
- Consideremos ahora un caso más realista en el que el emisor sólo retransmite cuando se sabe con seguridad que un paquete se ha perdido. En este caso, el rendimiento puede ser similar al de la Figura 3.46 b y en donde tenemos otro de los costes de una red congestionada: el emisor tiene que realizar retransmisiones para poder compensar los paquetes descartados 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 has sido retardado en la cola pero que todavía no se ha perdido, Figurea 3.46 c.
- 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.

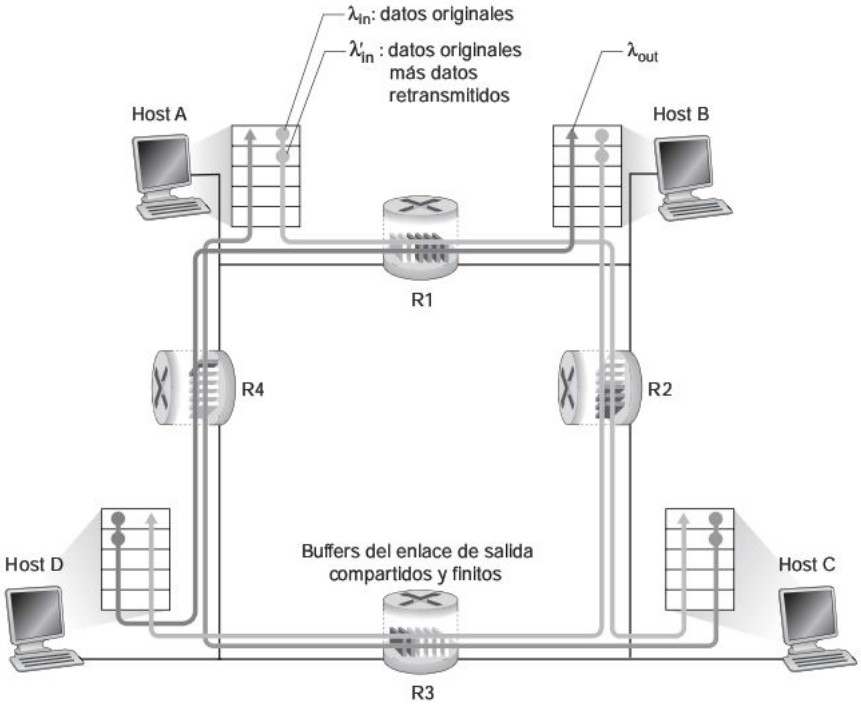


Figura 3.47 • Cuatro emisores, routers con buffers finitos y rutas con varios saltos.

- 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 pequeños de λ_{in}, un incremento de λ_{in} da lugar a un incremento de λ_{out}.
- Para el caso en que λ_{in} (y por tanto λ'_{in}) en que el tráfico es extremadamente grande, implica que la tasa de transferencia terminal a terminal de A-C tiende 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.

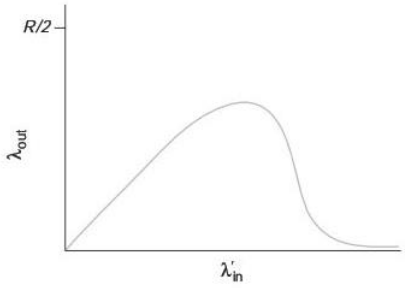


Figura 3.48 • Escenario 3: rendimiento con buffers finitos y rutas multisalto.

- 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.

4.6.2 Métodos para controlar la congestión

- Vamos a identificar los dos métodos más comunes de control de congestión que se utilizan en la práctica.
- 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.
        - Veremos que TCP tiene que aplicar necesariamente este método de control de congestión, ya que la capa IP no proporciona ninguna realimentación a los sistemas terminales relativa a la congestión de la red.
        - La pérdida de segmentos TCP se toma como indicación de que existe congestión en la red, por lo que TCP reduce el tamaño de su ventana en consecuencia.

    - **Control de congestión asistido por la red**:
        - En este método de control de congestión, los componentes de la capa de red proporcionan una realimentación explícita al emisor informando del estado de congestión en la red.
        - Esta realimentación puede ser tan simple como un único bit que indica que existe congestión en el enlace.
        - También es posible proporcionar una realimentación de red más sofisticada como el **ABR** (*Avalible bit-rate*) en ATM o el protocolo **XCP**.
        - En este mecanismo, 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:

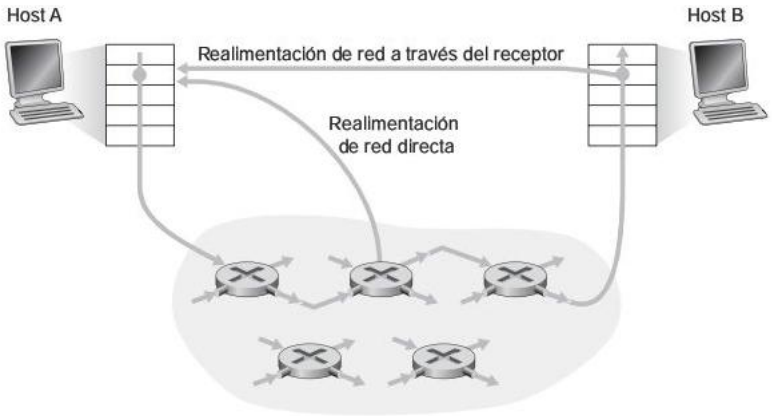


Figura 3.49 • Dos formas de realimentación de la información asistido por la red acerca de la congestión.

        - La realimentación directa puede hacerse desde un router de la red al emisor. Esta forma de notificación, toma la forma de un **paquete de asfixia o bloqueo** (***choke packet***).
        - La segunda forma tiene lugar cuando un router marca/actualiza un campo de un paquete que se transmite del emisor al receptor para indicar que hay congestión. Después de recibir un paquete marcado, el receptor notifica al emisor la existencia de la congestión.

4.6.3 Ejemplo de control de congestión asistido por la red: control de congestión en el servicio ABR de redes ATM

- Fundamentalmente, ATM emplea para la conmutación de paquetes una técnica basada en circuitos virtuales (**VC**, ***Virtual Circuit***). Esto significa que cada dispositivo de conmutación a lo largo de la ruta mantiene el estado del VC existente entre el origen y el destino.
- El tener información del estado de cada VC permite a un dispositivo de conmutación conocer el comportamiento de cada emisor individual y llevar a cabo acciones para el control de congestión específicas para cada origen.
- La información del estado de cada VC conservada en los conmutadores de la red hace que las redes ATM estén idealmente adaptadas para realizar un control de congestión asistido por la red.
- Cuando la red está poco cargada, el servicio ABR debería poder aprovecharse del ancho de banda de reserva disponible. Si la red está congestionada, el servicio ABR debería reducir su velocidad de transmisión a un valor mínimo predeterminado.

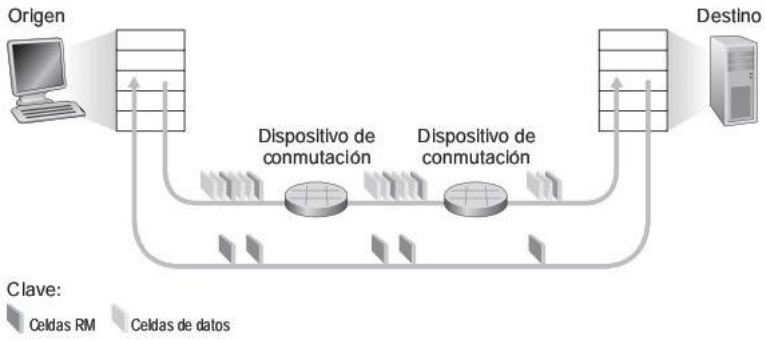


Figura 3.50 • Mecanismo de control de congestión del servicio ABR de ATM.

- Con el servicio ABR de ATM, las celdas de datos se transmiten desde un origen a un destino a través de una serie de dispositivos de conmutación intermedios.
- Intercaladas con las celdas de datos se encuentran las **celdas de gestión de recursos** (**celdas RM**, ***Resource-Managment***). Estas celdas RM se pueden utilizar para transportar información relativa al control de congestión entre los host y los dispositivos de conmutación.
- Cuando una celda RM llega a un destino, será enviada de vuelta al emisor.
- Un dispositivo de conmutación también puede generar una celda RM propia y enviarla directamente a un origen.
- El mecanismo de control de congestión del servicio ABR de ATM es un método basado en la velocidad. El emisor calcula una velocidad máxima a la que se puede transmitir y se autorregula de acuerdo a ello.
- ABR proporciona tres mecanismos para señalizar la información relativa a la congestión transmitida desde los dispositivos de conmutación al receptor:
    - Bit EFCI:
        - Cada celda de datos contiene un **bit EFCI** (***Explicit Forward Congestion Indication***).
        - Un dispositivo de conmutación de la red congestionado puede poner el bit EFCI de una celda de datos a 1 para indicar que existe congestión al host destino, el cual comprobará el bit EFCI de todas las celdas de datos recibidas.
        - Cuando llega una celda RM al destino, si la celda de datos recibida más recientemente tenía el bit EFCI a 1, entonces el destino pone el **bit de indicación de congestión** (**CI**, ***Congestion Indication***) de la celda RM a 1 y devuelve la celda RM al emisor.

- Bits **CI** y **NI** (***No Increase***):
  - La tasa de intercalado de las celdas RM es un parámetro ajustable, siendo el valor predeterminado una celda RM cada 32 celdas de datos.
  - Estas celdas RM disponen de un **bit indicativo de la congestión** (CI) y de un **bit de incremento** (NI) que un dispositivo de conmutación de la red congestionado puede configurar.
  - Específicamente, éstos, pueden poner el bit NI de una celda RM que le atraviese a 1 en condiciones de congestión leve y poner a 1 el bit CI bajo condiciones de congestión severa.

- Configuración de ER:
  - Cada celda RM también contiene un campo **ER** (***Explicit Rate***) de 2 byte.

- Un emisor ABR de una red ATM ajusta la velocidad a la que puede enviar las celdas en función de los valores de CI, NI y ER contenidos en las celdas RM devueltas.

4.7 Mecanismos de control de congestión de TCP

- TCP proporciona un servicio de transporte fiable entre dos procesos que se ejecutan en host diferentes.
- TCP tiene que utilizar un control de congestión terminal a terminal, 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.
- Pero este método plantea tres cuestiones:
  - ¿Cómo limita el emisor TCP la velocidad a la que envía el tráfico a través de su conexión?
    - 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 indica como `VentanaCongestion`, 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 `VentanaCongestion` y `VentanaRecepcion`, es decir:

$$\text{UltimoByteLeido} - \text{UltimoByteReconocido} \leq \min\{\text{VentanaCongestión}, \text{VentanaRecepcion}\}$$

    - Imagine una conexión en la que tanto la pérdida de paquetes como los retardos de transmisión sean despreciables. En esta situación, lo que ocurre es:
      - Al inicio de cada periodo RTT, la restricción (el buffer de recepción es tan grande que la restricción de la ventana de recepción puede ignorarse) permite al emisor enviar `VentanaCongestion` 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 `VentanaCongestion / 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.

- ¿Cómo percibe el emisor TCP que existe congestión en la ruta entre él mismo y el destino?
  - Definamos un suceso de perdida 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.
  - 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 sea descartado.
  - A su vez, el datagrama descartado da lugar a un suceso de pérdida en el emisor, el cual lo interpreta como una indicación de que existe congestión en la ruta entre el emisor y el receptor.
  - Cuando no se producen pérdidas de paquetes, el emisor TCP recibirá los paquetes de reconocimiento ACK correspondientes a los segmentos anteriores no reconocidos. TCP interpreta la llegada de estos paquetes ACK como una indicación de que todo está bien y empleará esos paquetes de reconocimiento para incrementar el tamaño de la ventana de congestión.
  - Puesto que TCP utiliza los paquetes de reconocimiento para provocar (o temporizar) sus incrementos de tamaño de la ventana de congestión, se dice que TCP es **auto-temporizado**.

- ¿Qué algoritmo deberá emplear el emisor para variar su velocidad de transmisión en función de la congestión percibida terminal a terminal?
  - TCP se basa en los siguientes principios:
    - Un segmento perdido implica congestión y por tanto, la velocidad del emisor TCP debe reducirse cuando se pierde un segmento.
    - 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.
    - Tanteo del ancho de banda. 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.
    - 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 ha variado la velocidad a la que comienza de nuevo a producirse congestión.

- Conocidos los fundamentos del mecanismo de control de congestión TCP, pasamos a estudiar el **algoritmo de control de congestión de TCP** [RFC 2581].
- El algoritmo consta de tres componentes principales:
  1. Arranque lento (**slow start**)
  2. Evitación de la congestión (**congestion avoidance**)
  3. Recuperación rápida (**fast recovery**)

Arranque lento

- Cuando se inicia una conexión TCP, el valor de la ventana de congestión normalmente se inicializa con un valor pequeño igual a 1 MSS [RFC 3390], que da como resultado una velocidad de transmisión inicial aproximadamente igual a MSS / RTT.

- 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 `VentanaCongestion` se establece a en 1 MSS y se incrementa 1 MSS cada vez que se produce el primer reconocimiento de un segmento transmitido.
- 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.
- ¿Cuándo debe finalizar este crecimiento exponencial?:
    1. Si se produce un suceso de pérdida de paquete señalado por un fin de temporización, el emisor TCP establece el valor de `VentanaCongestion` en 1 e inicia de nuevo un proceso de arranque lento.

       También define el valor de una segunda variable de estado que establece el umbral de arranque lento y que denominaremos `umbral` en `VentanaCongestion / 2`, la mitad del valor del tamaño de la ventana de congestión cuando se ha detectado que existe congestión.

       Dado que el `umbral` es igual a la mitad del valor que `VentanaCongestion` tenía cuando se detectó congestión por última vez, puede resultar imprudente continuar duplicando el valor de `VentanaCongestion` cuando se alcanza o sobrepasa el valor de `umbral`.

       Por tanto, cuando el valor de `VentanaCongestion` es igual a `umbral`, la fase de arranque lento termina y las transacciones TCP pasan al modo de evitación de la congestión.
    2. Si se detectan tres paquetes ACK duplicados, en cuyo caso TCP realiza una retransmisión rápida y entra en el estado de recuperación rápida.

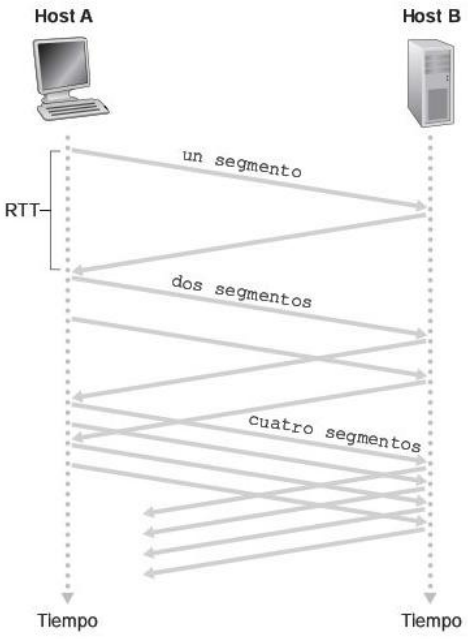


Figura 3.51 • Fase de arranque lento de TCP.

Evitación de la congestión

- Al entran en este estado, se puede estar al borde de la congestión. En consecuencia, se incrementa el valor de `VentanaCongestion` solamente en un MSS cada RTT [RFC 2581].
- Para ello, un método habitual consiste en que un emisor TCP aumenta el valor de `VentanaCongestion` en `MSS` / `VentanaCongestion` bytes cuando llega un nuevo paquete de reconocimiento.
- ¿En qué momento debería detenerse el crecimiento lineal en este modo?
    1. El algoritmo de evitación de la congestión de TCP se comporta del mismo modo que cuando tiene lugar un fin de temporización en el caso de arranque lento.
    2. Si se detecta una pérdida de paquete a causa de la llegada de tres ACK duplicados, la red continua entregando segmentos del emisor al receptor. Por tanto, TCP divide entre dos el valor de `VentanaCongestion` (añadiendo 3 MSS como forma de tener en cuenta los tres ACK duplicados) y configura el valor de `Umbral` para que sea igual a la mitad del valor que `VentanaCongestion` tenía cuando se recibieron los tres ACK duplicados. A continuación, se entra en el estado de recuperación rápida.

Recuperación rápida

- En esta fase, el valor de `VentanaCongestion` 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 `VentanaCongestion`.
- Si se produce un fin de temporización, el mecanismo de recuperación rápida efectúa una transición al estado de arranque lento y después de realizar las mismas acciones que en los modos de arranque lento y de evitación de la congestión.
- El mecanismo de recuperación rápida es un componente de TCP recomendado, aunque no obligatorio [RFC 2581].

Control de congestión de TCP: retrospectiva

- Ignorando la fase inicial de arranque lento y suponiendo que las pérdidas están indicadas por la recepción de tres ACK duplicados, el control de congestión de TCP consiste en un crecimiento lineal (aditivo) de `VentanaCongestion` a razón de 1 MSS por RTT, seguido de un decrecimiento multiplicativo (división entre 2) del tamaño de `VentanaCongestion`, cuando se reciben tres ACK duplicados.
- Por esta razón, suele decirse que el control de congestión de TCP es una forma de **crecimiento aditivos 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", lo que también ilustra que TCP va tanteando el ancho de banda:

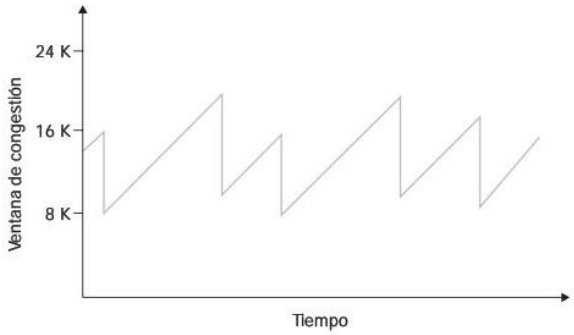


Figura 3.54 • Control de congestión con crecimiento aditivo y decrecimiento multiplicativo.

- La mayor parte de las implementaciones TCP actuales emplean el algoritmo Reno. Se han propuesto muchas variantes de este algoritmo [RFC 3782], [RFC 2018].
- El algoritmo TCP Vegas intenta evitar la congestión manteniendo una buena tasa de transferencia.
- El algoritmo AIMD de TCP fue desarrollado basándose en un enorme trabajo de ingeniería y experimentación con los mecanismos de control de congestión en redes reales. Diez años después del desarrollo de TCP.

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 de una conexión TCP de larga duración (ignorando las fases de arranque lento).
- 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 de 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 $W/(2 * RTT)$ y w/RTT.

- Puesto que la tasa de transferencia de TCP aumenta linealmente entre los dos valores extremos, tenemos que la tasa de transferencia media de una conexión es:

$$\frac{0{,}75 * W}{RTT}$$

El futuro de TCP

4.7.1 Equidad

- Considerando 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. Suponga que cada conexión está transfiriendo un archivo de gran tamaño y que no existe tráfico UDP atravesando el enlace de cuello de botella.

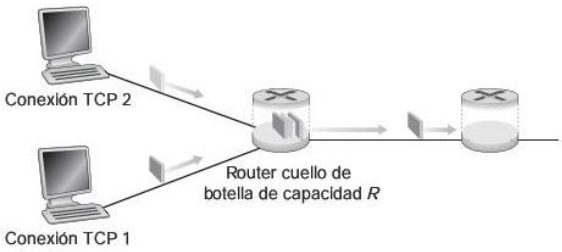


Figura 3.55 • Dos conexiones TCP que comparten un mismo enlace de cuello de botella.

- 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 de ancho de banda del enlace.
- ¿Es el algoritmo AIMD equitativo? Dah-Ming Chiu y Raj Jain en su *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*, proporcionan una explicación de por qué el control de congestión de TCP converge para proporcionar la misma cuota de ancho de banda de un enlace cuello de botella a las conexiones TCP que compiten por el ancho de banda.
- En la práctica, estas condiciones normalmente no se dan y las aplicaciones cliente-servidor pueden por tanto obtener cuotas desiguales de 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 éste va liberándose y por tanto disfrutan de una tasa de transferencia más alta que aquellas conexiones cuyo valor de RTT es más grande.

Equidad en UDP

- Desde la perspectiva de TCP, las aplicaciones 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.

Equidad y conexiones TCP en paralelo

- Pero aunque se pudiera forzar al tráfico UDP a comportarse intuitivamente, el problema de la equidad todavía no estaría completamente resuelto. Esto es porque no hay nada que impida una aplicación basad en TCP utilizar varias conexiones en paralelo.
- Cuando una aplicación emplea varias conexiones en paralelo obtiene una fracción grande del ancho de banda de un enlace congestionado.