Unidad 6 : La Capa de Transporte

Servicios suministrados a las capas superiores (Capa de Sesión).

El objetivo fundamental de la capa de Transporte es proporcionar un servicio eficiente, confiable y económico a sus usuarios, que son los procesos de la capa de aplicación. Para cumplir con este propósito, la capa de transporte hace uso de los servicios proporcionados por la capa de red. El *hardware* o el *software* de la capa de transporte que se encarga del trabajo se llama **entidad de transporte**. Ésta puede estar en el núcleo del sistema operativo, en un proceso de usuario independiente, en un paquete de biblioteca que forma parte de las aplicaciones de la red o en la tarjeta de interfaz de la red. Cuando la portadora presta un servicio de transporte confiable, la entidad de transporte reside en máquinas especiales de interfaz en la orilla de la subred a la que se conectan los *hosts*. La figura 1 muestra la relación entre las capas de red, transporte y aplicación.

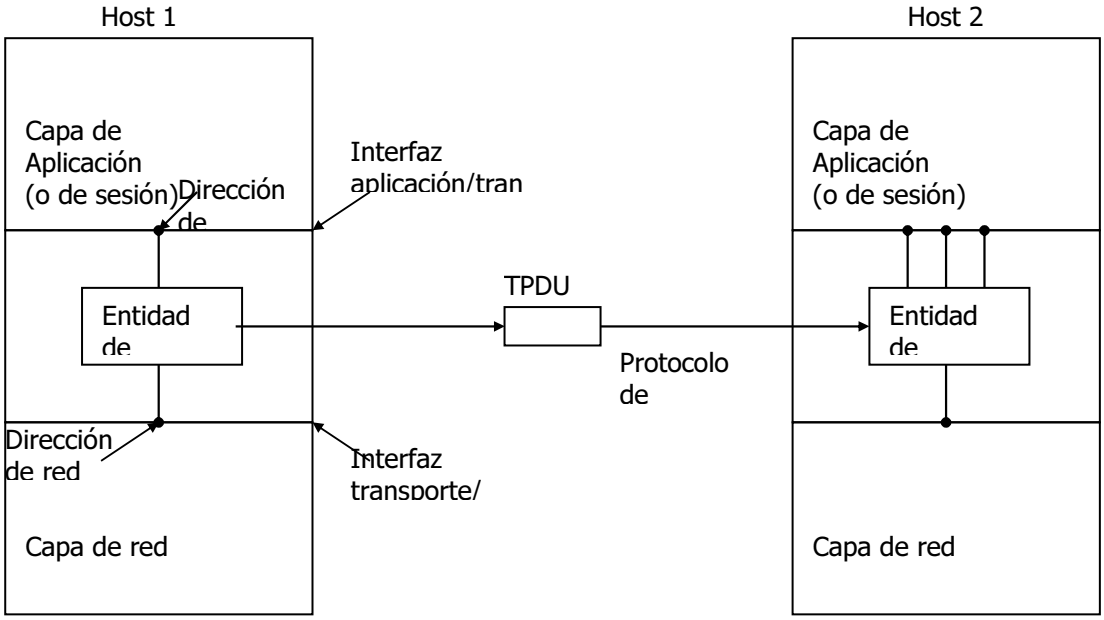


Figura 1. Las capas de red, transporte y aplicación

La existencia de la capa de transporte hace posible que el servicio de transporte sea más confiable que el servicio de red subyacente. La capa de transporte puede detectar y compensar paquetes perdidos y datos alterados. Las primitivas del servicio de transporte pueden diseñarse de modo que sean independientes de las primitivas del servicio de red. Es posible escribir programas de aplicación usando un grupo estándar de estas primitivas, y hacer que estos programas trabajen en una variedad amplia de redes, sin tener los problemas que surgen de trabajar con interfaces de subred diferentes y transmisiones no confiables. Así, esta capa cumple la función clave de aislar las capas superiores respecto de la tecnología, el diseño y las imperfecciones de la subred.

Por lo general, las cuatro capas inferiores pueden verse como el **proveedor del servicio de transporte** y las capas superiores son el **usuario del servicio de transporte**. Esta distinción hace que la capa de transporte sea el límite principal entre el proveedor y el usuario del servicio confiable de transmisión de datos.

La capa de Transporte como QOS (Quality of Service).

Otra manera de ver la capa de transporte es considerar que su función primaria es la de mejorar la **calidad del servicio** (**QoS**) proporcionada por la capa de red. Si el servicio de red es impecable, la capa de transporte tiene una tarea fácil; si, por el contrario, el servicio de red es malo, la capa de transporte tiene que construir un puente sobre el abismo que separa lo que quieren los usuarios de transporte y lo que la capa de red proporciona.

La QoS se caracteriza por varios parámetros específicos. Algunos de ellos son:

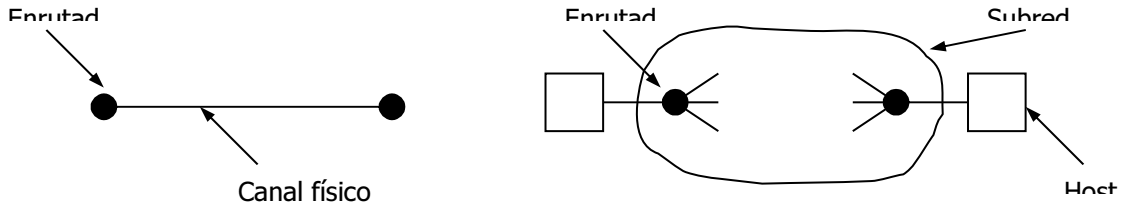
- *Retardo de establecimiento de la conexión*: es el tiempo que transcurre entre la solicitud de una conexión de transporte y la confirmación del usuario del servicio de transporte.
- *Probabilidad de falla de establecimiento de la conexión*: es la posibilidad de que una conexión no se establezca en el lapso de máximo retardo de establecimiento (por congestionamiento de la red, falta de espacio de tablas, etc.)
- *Rendimiento*: mide la cantidad de bytes de datos de usuario transferidos por segundo, medido durante algún intervalo de tiempo.
- *Retardo de tránsito*: mide el tiempo entre el envío de un mensaje por el usuario de transporte de la máquina de origen y su recepción por el usuario de transporte de la máquina de destino.
- *Protección*: provee un mecanismo para que el usuario especifique su interés en que la capa de transporte proporcione protección de lectura y modificación de los datos transmitidos contra terceros no autorizados.
- *Prioridad*: permite que el usuario indique qué conexiones son más importantes
- *Tasa de error residual*: mide la cantidad de mensajes perdidos o alterados como una fracción del total enviado.
- que otras y que, en el caso de un congestionamiento, las conexiones de alta prioridad sean atendidas antes que las de baja prioridad.
- *Tenacidad*: da la probabilidad de que la capa de transporte por sí misma termine instantáneamente una conexión debido a problemas internos o congestionamiento.

El servicio de transporte puede permitir que el usuario especifique valores preferidos, aceptables y mínimos para los parámetros de servicio en el momento de establecerse una conexión. Es responsabilidad de la capa de transporte examinar estos parámetros y, dependiendo de los tipos de servicio de red disponibles, determinar si puede proporcionar el servicio requerido. En algunos casos, la capa de transporte puede darse cuenta de que algunos parámetros son inalcanzables, en cuyo caso indica al solicitante que falló el intento de conexión, sin molestarse en comunicarse con el destino.

En otros casos, la capa de transporte sabe que no puede lograr la meta deseada, pero puede lograr una tasa menor, aunque aún aceptable. Entonces envía la tasa menor y la tasa mínima aceptable a la máquina remota, solicitando establecer una conexión. Si la máquina remota no pude manejar el valor propuesto, pero puede manejar un valor por encima del mínimo, puede hacer una contraoferta. Si no puede manejar ningún valor superior al mínimo, rechaza el intento de conexión. Por último se informa al usuario de origen si la conexión se estableció o rechazó, y en caso de haberse establecido, los valores de los parámetros acordados. Este procedimiento se llama **negociación abierta**. Una vez que se han negociado las opciones, permanecen así durante el resto de vida de la conexión.

Protocolos de Transporte

El servicio de transporte se implementa mediante un **protocolo de transporte** entre las dos entidades de transporte. Al igual que el protocolo de enlace de datos, se encarga del control de errores, la secuencia y el control de flujo. Pero las diferencias que existen entre ambos protocolos se deben a diferencias en los entornos en que operan cada uno de ellos. En la capa de enlace, dos enrutadores se comunican directamente mediante un canal físico mientras que, en la capa de transporte, este canal es reemplazado por la subred completa como se ve en la figura 2.



(a) (b)

Figura 2. (a) Entorno de la capa de enlace. (b) Entorno de la capa de transporte

Esta diferencia hace que la capa de transporte requiera un enfoque distinto del que se usa en la capa de enlace de datos. Por ejemplo, en la capa de transporte es necesario un direccionamiento explícito de los destinos, lo cual hace mas complicado el establecimiento inicial de la conexión; la gran cantidad de conexiones en la capa de transporte hace menos atractiva la idea de dedicar muchos *buffers* a cada una, etc.

Administración de la conexión

Direccionamiento

Cuando un proceso de aplicación desea establecer una conexión con un proceso de aplicación remoto, debe especificar a cual debe conectarse. El método que normalmente se emplea es definir direcciones de transporte en las que los procesos pueden estar a la escucha de solicitudes de conexión. En Internet, por ejemplo, estos puntos terminales son pares (dirección IP, puerto local). Usaremos el término **TSAP** (**punto de acceso al servicio de transporte**) para referirnos a los puntos terminales en general.

Si las direcciones TSAP fueran estables, todo funcionaría bien. Pero en general, los procesos de usuario quieren hablar con otros procesos de usuario que sólo existen durante un tiempo corto y no tiene una dirección TSAP conocida por adelantado. Es más, si puede haber muchos procesos de servidor, la mayoría de los cuales se usan pocas veces, sería un desperdicio tenerlos activos a todos, escuchando en una dirección estable todo el día.

Un esquema que da solución e este problema es el **protocolo inicial de conexión**, empleado por los *hosts* UNIX de Internet. En lugar de que cada servidor escuche en un TSAP bien conocido, cada máquina tiene un **servidor de procesos** especial que actúa como apoderado (*proxy*) de los servidores de menor uso y escucha en un grupo de puertos al mismo tiempo, esperando una solicitud de conexión TCP. Al obtener una solicitud entrante, el servidor de procesos genera el servidor solicitado, permitiéndole heredar la conexión con el usuario entrante. El nuevo servidor hace el trabajo requerido mientras que el servidor de procesos retoma a escuchar solicitudes nuevas.

Este protocolo funciona bien para aquellos servidores que pueden crearse cuando se necesita, pero hay muchas situaciones en las que los servicios existen independientemente del servidor de procesos. Para manejar esta situación se usa un esquema alterno donde existe un proceso especial llamado **servidor de nombres** o **servidor de direcciones**. En este modelo, al crearse un servicio nuevo, debe registrarse en el servidor de nombres, dando tanto su nombre de servicio como la dirección de su TSAP. El servidor de nombres registra esta información en su base de datos interna por lo que, cuando le llegan solicitudes posteriores, sabe las respuestas.

Para que la entidad de transporte sepa qué dirección de capa de red debe usar para establecer una conexión de red con la entidad de transporte remota que maneja el TSAP requerido, es necesario que la estructura de las direcciones TSAP sea jerárquica. Esto es, que la dirección consista de una secuencia de campos usados para dividir en porciones disconexas el espacio de direcciones. Una dirección TSAP verdaderamente universal podrías tener la siguiente estructura:

dirección = <galaxia><estrella><planeta><país><red><*host*><puerto>

Con este esquema es directa la localización de un TSAP en cualquier parte del universo conocido. Análogamente, si una dirección TSAP es una concatenación de una dirección NSAP (punto al servicio de acceso de red) y un puerto, entonces, cuando se le da una dirección TSAP a una entidad de transporte para conectarse a ella, usa la dirección NSAP contenida en la dirección TSAP para alcanzar la unidad de transporte remota adecuada.

Establecimiento de conexión

El establecimiento de una conexión parecería muy fácil, es suficiente que una entidad de transporte enviara una TPDU (**unidad de datos del protocolo de transporte**) CONNECTION REQUEST (solicitud de conexión) al destino y esperara una respuesta CONNECTION ACCEPTED. El problema ocurre cuando la red puede perder, almacenar o duplicar paquetes.

Uno de los mayores problemas es la existencia de duplicados retrasados, para solucionarlo es necesario diseñar un mecanismo para matar a los paquetes viejos que aún quedan "vagando" por ahí. Si se puede asegurar que ningún paquete viva más allá de cierto tiempo conocido, el problema se vuelve algo más manejable.

El tiempo de vida de un paquete puede restringirse a un máximo conocido usando una de las siguientes técnicas:

1. Diseño de subred restringido: incluye cualquier técnica que evite que los paquetes hagan ciclos, combinado con una manera de limitar el retardo por congestionamientos a través de la trayectoria más larga posible (ahora conocida).
2. Contador de saltos en cada paquete: consiste en incrementar el conteo de saltos cada vez que se reenvía el paquete. El protocolo de enlace de datos simplemente descarta cualquier paquete cuyo contador de saltos ha excedido cierto valor.
3. Marca de tiempo en cada paquete: requiere que cada paquete lleve la hora en la que fue creado, acordando los enrutadores descartar cualquier paquete que haya rebasado cierto tiempo predeterminado. Este método requiere que los relojes de los enrutadores estén sincronizados, posiblemente escuchando la WWV o alguna estación de radio que difunda la hora exacta periódicamente.

En la práctica es necesario garantizar no solo que el paquete está muerto, sino también que todos los acuses de recibo lo estén. Para ello se introduce T, que es un múltiplo pequeño del tiempo de vida de paquete máximo verdadero. El múltiplo depende del protocolo y simplemente tiene el efecto de hacer más grande a T. Si esperamos un tiempo T después del envío de un paquete, podemos estar seguros de que todos los rastros suyos han desaparecido, y que ni él ni sus acuses de recibo aparecerán repentinamente de la nada.

Teniendo limitados los tiempos de vida de los paquete, es posible proponer una manera a prueba de errores de establecer conexiones seguras.

Liberación de conexión

La liberación de una conexión es más fácil que su establecimiento. Hay dos estilos de terminación de una conexión: liberación asimétrica y liberación simétrica. La primera es la manera en que funciona el sistema telefónico: cuando una parte cuelga, se rompe la conexión. La segunda trata la conexión como dos conexiones unidireccionales distintas, y requiere que cada una se libere por separado. La liberación asimétrica es abrupta y puede resultar en la pérdida de datos (figura 3).

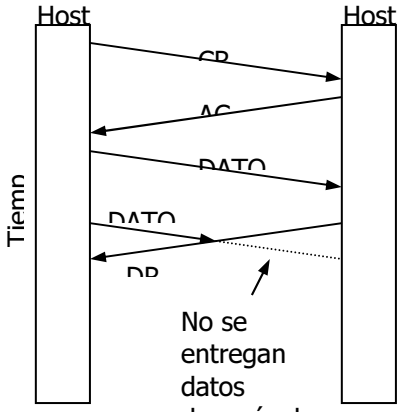
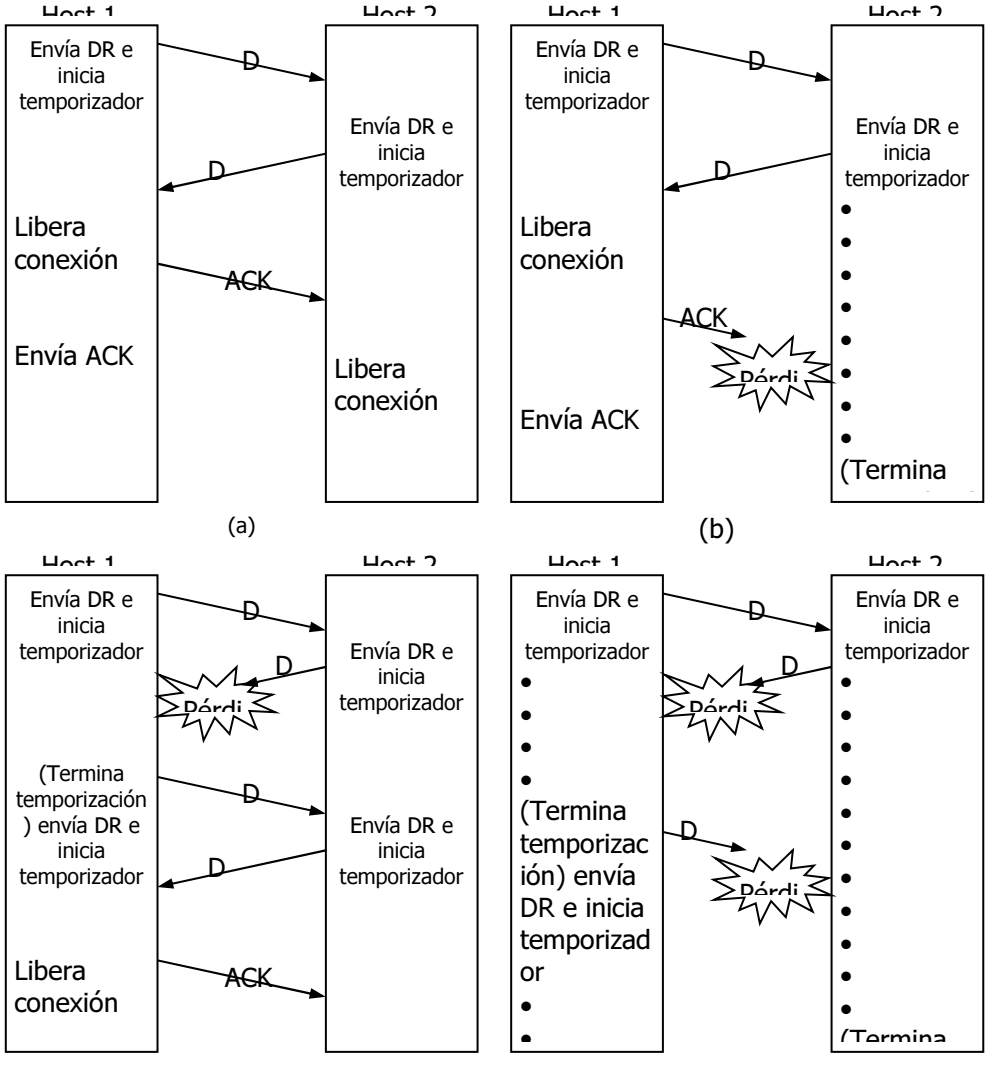


Figura 3. Desconexión abrupta con pérdida de datos.

Es obvio que requiere un protocolo de liberación más refinado para evitar la pérdida de datos. Una posibilidad es usar la liberación simétrica, en la que cada dirección se libera independientemente de la otra. Aquí, un *host* puede continuar enviando datos aún tras haber enviado una TPDU DISCONNECT.

Por desgracia, este protocolo no siempre funciona. De hecho, puede demostrarse que no existe un protocolo que funcione, ya que si ninguna de las partes está preparada para desconectarse hasta estar convencida de que la otra esta también preparada para desconectarse, la desconexión no ocurrirá nunca. En la figura 4 se ilustran cuatro situaciones de liberación usando un protocolo de



acuerdo de tres vías. Aunque este protocolo no es infalible, generalmente es adecuado.En la parte (a) se ve el caso normal en el que uno de los usuarios envía una TPDU DR (solicitud de desconexión) a fin de iniciar la liberación de una conexión. Al llegar, el receptor devuelve también una TPDU DR e inicia un temporizador, para el caso que se pierda su DR. Al llegar esta DR, el transmisor original envía de regreso una TPDU de ACK y libera la conexión.

Si se pierde la TPDU ACK, como muestra la parte (b), la situación se salva por el temporizador. Si éste expira, la conexión se libera de todos modos.

Figura 4. Cuatro situaciones de protocolo de liberación de una conexión. (a) Caso normal del protocolo de acuerdo de tres vías. (b) Pérdida del último ACK. (c) Respuesta perdida. (d) Respuesta perdida y pérdida de las DR subsecuentes

Si se pierde el segundo DR, el usuario que inicia la desconexión no recibirá la respuesta esperada, terminará de temporizar y comenzará de nuevo. En la parte (c) vemos la manera en que funciona esto, suponiendo que la segunda vez no se pierden algunas TPDU y que todas se entregan correctamente y a tiempo.

Como última situación, la parte (d) es semejante a la (c), excepto que suponemos que todos los intentos repetidos de retransmitir la DR también fallan debido a la pérdida de algunas TPDU. Tras *N* intentos, el retransmisor se da por vencido y libera la conexión. Mientras tanto, el receptor termina de temporizar y también sale.

Control de flujo y Almacenamiento temporal

En muchos aspectos el problema del control de flujo en la capa de transporte es igual que en la capa de enlace de datos, pero en otros es diferente. La similitud es que en ambas capas se requiere una ventana corrediza u otro esquema en cada conexión para evitar que un transmisor rápido abrume a un receptor lento. La diferencia principal es que un enrutador tiene relativamente pocas líneas, y que un *host* puede tener numerosas conexiones. Esto hace que sea impráctico implementar la estrategia de *buffers* de enlace de datos en la capa de transporte.

La entidad de transporte transmisora debe manejar *buffers*, porque cabe la posibilidad de que tengan que retransmitir. Si el receptor sabe que el transmisor pone en *buffer* todas las TPDU hasta que se reconocen, el receptor podría mantener un solo grupo de *buffers* compartido con todas las conexiones. Cuando entra una TPDU, se hace un intento por adquirir dinámicamente un *buffer* nuevo. Si hay uno disponible, se acepta la TPDU; de otro modo, se descarta. Dado que el transmisor está preparado para retransmitir las TPDU perdidas por la subred, no hay nada malo en hacer que el receptor se deshaga de las TPDU, el transmisor seguirá intentando hasta que recibe un reconocimiento.

Por otro lado, si el transmisor sabe que el receptor siempre tiene espacio de *buffer*, no necesita retener copias de las TPDU que envía. Sin embargo, si el receptor no puede garantizar que se aceptará cada TPDU de entrada, el transmisor tendrá que usar *buffers* de todas maneras.

Aún si el receptor ha acordado manejar los *buffers*, queda la cuestión del tamaño. Se podría organizar los buffers de tres maneras:

- Si la mayoría de las TPDU tiene aproximadamente el mismo tamaño, se puede asignar el mismo tamaño a todos los *buffers*, con una TPDU por *buffer*. Pero, si hay una variación grande este método no conviene, puesto que si el tamaño se escoge igual a la TPDU más grande, se desperdiciará espacio cuando llegue una TPDU corta y, si se escoge menor que el tamaño máximo de TPDU, se requerirán varios *buffers* para las TPDU grandes, con la complejidad asociada.
- Se puede usar *buffers* de tamaño variable. La ventaja aquí es un mejor uso de la memoria, al costo de una gestión de *buffers* más complicada.

- Una tercera posibilidad es dedicar un solo *buffer* circular grande por conexión. Este sistema hace buen uso de la memoria cuando todas las conexiones tienen una carga alta, pero es deficiente si algunas conexiones son de poca carga.

La media óptima entre los *buffers* en el origen y los *buffers* en el destino depende del tipo de tráfico transportado por la conexión. Para un tráfico en ráfagas de bajo ancho de banda, es mejor mantener *buffers* en el transmisor; para alto ancho de banda, es mejor hacerlo en el receptor.

A medida que se abren y cierran conexiones, y a medida que cambia el patrón de tráfico, el transmisor y el receptor necesitan ajustar dinámicamente sus asignaciones de *buffers*. En consecuencia, el protocolo de transporte debe permitir a un *host* transmisor solicitar espacio de *buffer* en el otro extremo.

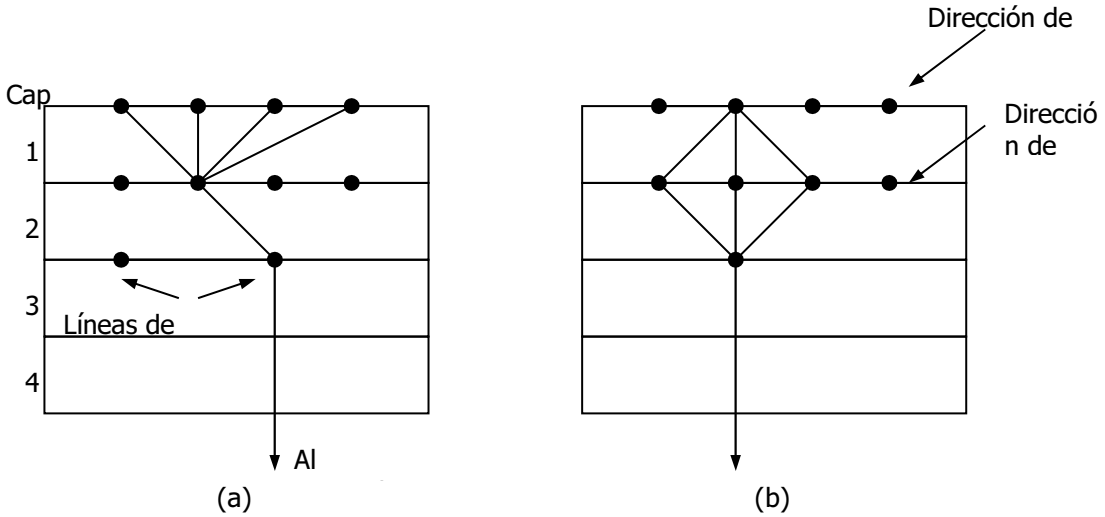
Una manera razonablemente general de manejar la asignación dinámica de *buffers* es desacoplarlos de los acuses de recibo. La gestión dinámica de *buffers* implica una ventana de tamaño variable. Inicialmente, el transmisor solicita una cierta cantidad de *buffers*, con base en sus necesidades percibidas. El receptor entonces otorga tantos *buffers* como puede. Cada vez que el transmisor envía una TPDU, debe disminuir su asignación, deteniéndose por completo al llegar la asignación a cero. Entonces el receptor incorpora tanto los reconocimientos como las asignaciones de *buffer* al tráfico en reversa.

Multiplexión

En la capa de transporte puede surgir la necesidad de multiplexión por varias razones. Por ejemplo, como consecuencia de una estructura de precios que penaliza fuertemente a las instalaciones que tienen muchos circuitos virtuales abiertos durante largos períodos de tiempo. Ante esta situación se realiza la multiplexión de diferentes conexiones de transporte en la misma conexión de red. Esta forma de multiplexión, llamada **multiplexión ascendente**, se muestra en la figura 5(a).

Figura 5. (a) Multiplexión ascendente. (b) Multiplexión descendente

La multiplexión también puede ser útil en la capa de transporte por otra razón, relacionada con las decisiones técnicas de la portadora, en lugar de sus decisiones de precio. Supongamos que se necesita una conexión de alto ancho de banda. Si la subred obliga a un control de flujo de ventana corrediza de *n* bits, se deberá detener el envío tan pronto estén pendientes $2^n - 1$ paquetes y se



deberá esperar que los paquetes se propaguen al *host* remoto y se reconozcan.

Una posible solución es hacer que la capa de transporte abra varias conexiones de red y distribuya el tráfico entre ellas por turno circular, como lo indica la figura 5(b). Esto se llama **multiplexión descendente**. Si hay disponibles varias líneas de salida, puede usarse también este tipo de multiplexión para aumentar aún más el desempeño.

Recuperación ante caídas

Si los hosts y los enrutadores están sujetos a caída, la recuperación de éstas se vuelve un tema importante. Si la entidad de transporte está por dentro de los hosts, la recuperación de caídas de red y enrutador, es directa.

El problema es más complicado cuando hay que recuperarse de caídas del host. En un intento por recuperar su estado previo, el servidor podría enviar una TPDU de difusión a todos los demás *hosts*, anunciando que acaba de caerse y solicitando a sus clientes que informen sobre el estado de todas las conexiones abiertas. Cada cliente puede estar en uno de dos estados: una TPDU pendiente, S1, o ninguna TPDU pendiente, S0. Con base en esta información de estado, el cliente debe decidir si retransmitirá o no la TPDU más reciente.

A primera vista parece obvio que el cliente debe retransmitir sólo si tiene una TPDU pendiente no reconocida, pero no es tan sencillo. Supongamos que la entidad de transporte del servidor envía primero un reconocimiento y luego ejecuta la actualización del proceso de aplicación, si la caída ocurre antes de hacer la escritura, el cliente recibirá el acuse de recibo y estará en el estado S0. Entonces el cliente no retransmitirá, pensando (incorrectamente) que la TPDU llegó. Por el contrario, si primero se hace la escritura y luego se envía el acuse de recibo, se generará la duplicación de una TPDU.

Sin importar como se programen el transmisor y el receptor, siempre habrá situaciones en las que el protocolo no podrá recuperarse correctamente. El servidor puede programarse de una de dos maneras: mandar acuse de recibo primero o escribir primero. El cliente puede programarse de cuatro maneras: siempre retransmitir la última TPDU, nunca retransmitir la última TPDU, retransmitir sólo en el estado S0 o retransmitir sólo en el estado S1. Esto da ocho combinaciones, pero para cada combinación existe algún grupo de eventos que hacen que falle el protocolo.

Estrategia usada por el host transmisor	Primero ACK luego escritura			Primero escritura luego ACK		
	C(W)	WC	(AW)	(WA)	AC	C(A)
Siempre retransmitir	IEN	UP	IEN	IEN	UP	UP
Nunca retransmitir	ERD	IEN	ERD	ERD	IEN	IEN
Retransmitir con S0	IEN	UP	ERD	ERD	UP	IEN
Retransmitir con S1	ERD	IEN	IEN	IEN	IEN	UP

BIEN: el protocolo funciona correctamente

DUP: el protocolo genera un mensaje duplicado

PERD: el protocolo pierde un mensaje

Hacer más elaborado el protocolo no sirve de nada. La conclusión es inevitable: la caída de un host y su recuperación no pueden hacerse transparentes a las capas superiores.

Capa de transporte en Internet

Internet tiene dos protocolos principales en la capa de transporte, uno orientado a conexiones (TCP) y el otro sin conexiones (UDP).

El **TCP** (**protocolo de control de transmisión**) se diseñó específicamente para proporcionar una corriente de bytes confiable a través de una subred no confiable. Una subred es diferente de una sola red porque las distintas partes pueden tener topologías, anchos de banda, retardos, tamaños de paquete y otros parámetros con grandes diferencias. El TCP se diseñó para adaptarse dinámicamente a las propiedades de la subred y para ser robusto ante muchos tipos de fallas.

Modelo de servicio TCP

El servicio TCP se obtiene haciendo que tanto el transmisor como el receptor creen puntos terminales, llamados sockets. Cada socket tiene un número (dirección) de socket que consiste en la dirección del *host*, llamado **puerto**. Puerto es el nombre en TCP de un TSAP. Para obtener el servicio TCP, debe establecerse explícitamente una conexión entre un socket de la máquina transmisora y un socket de la máquina receptora.

Todas las conexiones TCP son dúplex integral y punto a punto. Dúplex integral significa que el tráfico puede ir en ambos sentidos al mismo tiempo. Punto a punto significa que cada conexión tiene exactamente dos puntos terminales. El TCP no reconoce la multitransmisión ni la difusión.

Una conexión TCP es una corriente de bytes, no una corriente de mensajes. Los límites del mensaje no se conservan de extremo a extremo. Por ejemplo, si el proceso transmisor hace 4 escrituras de 512 bytes en una corriente TCP, estos datos pueden entregarse como 4 bloques de 512 bytes, 2 bloques de 1024 bytes, o de algún otro modo. No hay manera de que el receptor detecte las unidades en las que se escribieron los datos. Cuando una aplicación pasa datos al TCP, éste puede enviarlos de inmediato o guardarlos en un *buffer*, a discreción propia.

Otra característica interesante del TCP son los **datos urgentes**. Cuando un usuario interrumpe un cómputo remoto ya iniciado, la aplicación transmisora pone cierta información de control en la corriente de datos y se la da al TCP junto con la bandera URGENT. Este evento hace que el TCP deje de acumular datos y transmita de inmediato todo lo que tiene para esa conexión.

El protocolo TCP

Cada byte de una conexión TCP tiene su propio número de secuencia de 32 bits. Este número de secuencia se usa tanto para los acuses de recibo como para el mecanismo de ventana.

La entidad TCP transmisora y la receptora intercambian datos en forma de segmentos. Un **segmento** consiste de una cabecera TCP fija de 20 bytes (más una parte opcional) seguida de cero o más bytes de datos. El *software* de TCP decide el tamaño de los segmentos; puede acumular datos de varias escrituras para formar un segmento, o dividir los datos en varios segmentos. Hay dos límites para el tamaño del segmento:

- Cada segmento, incluida la cabecera TCP, debe caber en la carga útil de 65.535 bytes del IP.
- Cada red tiene una **unidad máxima de transferencia** (**MTU**), y cada segmento debe caber en ella.

Las MTU, por lo general, son de varios miles de bytes y definen el límite superior del tamaño de segmento. Si un segmento pasa a través de una serie de redes sin fragmentarse y se topa con una

red cuya MTU es menor que el segmento, el enrutador de la frontera fragmenta el segmento en dos o más segmentos más pequeños.

El protocolo básico usado por las entidades del TCP es el protocolo de ventana corrediza. Cuando un transmisor envía un segmento, también inicia un temporizador. Cuando llega el segmento al destino, la entidad receptora devuelve un segmento (con datos, si existen) que contiene un número de acuse de recibo igual al siguiente número de secuencia que espera recibir. Si el temporizador del transmisor expira antes de la recepción del acuse de recibo, el transmisor envía de nuevo el segmento.

El TCP debe estar preparado para manejar y resolver algunos problemas que surgen debido a la fragmentación de segmentos, estos son:

- Es posible que llegue una parte del segmento transmitido y le entidad receptora envíe un acuse de recibo, pro la otra parte se pierda.
- Pueden llegar segmentos fuera de orden.
- Pueden retardarse algunos segmentos en tránsito durante tanto tiempo que el transmisor termina de temporizar y retransmite nuevamente.
- Siendo tantas la redes que constituyen Internet, es posible que un segmento se tope con una red congestionada o rota en alguna parte de su trayectoria.

La cabecera de segmento TCP

La figura 6 muestra la distribución de un segmento TCP. Cada segmento comienza con una cabecera de formato fijo de 20 bytes. La cabecera fija puede ir seguida de opciones de cabecera. Tras las opciones, pueden continuar hasta 65.535 –20 – 20 = 65.515 bytes de datos. Los segmentos sin datos son legales y se usan para acuses de recibo y mensajes de control.

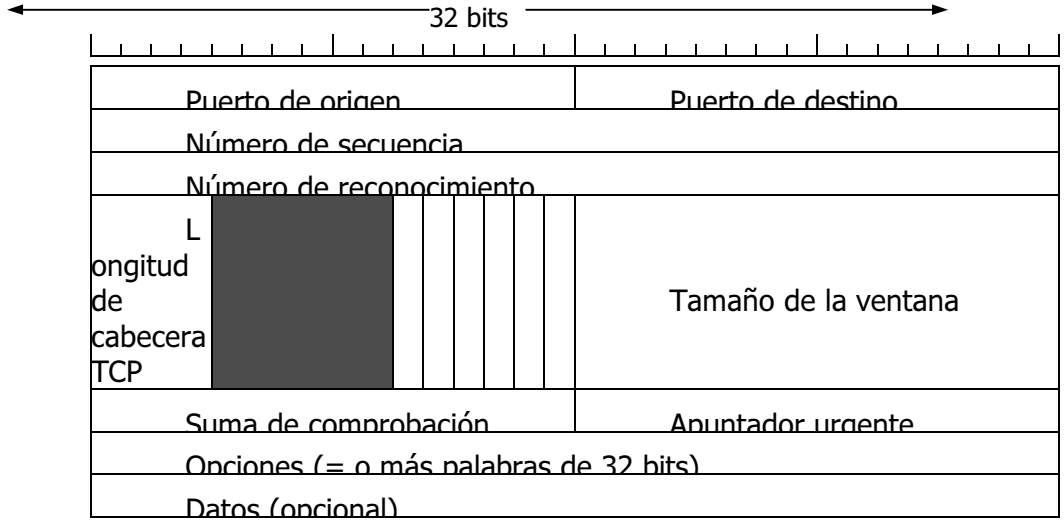


Figura 6. Cabecera TCP

Los campos *puerto de origen* y *puerto de destino* identifican los puntos terminales locales de la conexión.

Lon campos *número de secuencia* y *número de acuse de recibo* desempeñan sus funciones normales. El segundo especifica el siguiente byte esperado, no el último byte correctamente recibido.

La *longitud de cabecera TCP* indica la cantidad de palabras de 32 bits contenidas en la cabecera TCP. Esta información es necesaria porque el campo *opciones* es de longitud variable y la cabecera también.

A continuación viene un campo de 6 bits que no se usan.

Luego seis banderas de 1 bit:

- URG: sirve para indicar un desplazamiento en bytes a partir del número actual de secuencia en el que se encuentran los datos urgentes.
- ACK: se establece en 1 para indicar que el número de acuse de recibo es válido. Si es 0, el segmento no contiene acuse de recibo.
- PSH: se solicita al receptor entregar los datos a la aplicación a su llegada y no ponerlos en un *buffer*.
- RST: se usa para restablecer una conexión que se ha confundido debido a una caída de host u otra razón, también sirve para rechazar un segmento no válido o un intento de abrir una conexión.
- SYN: se usa para establecer conexiones. La solicitud de conexión tiene SYN = 1 y ACK = 0 para indicar que el campo de acuse de recibo incorporado no está en uso. La respuesta de conexión sí lleva un reconocimiento, por lo que SYN = 1 y ACK = 1.
- FIN: se usa para liberar una conexión; especifica que el transmisor no tiene más datos para enviar.

El control de flujo en el TCP se maneja usando una ventana corrediza de tamaño variable. El campo de *ventana indica* la cantidad de bytes que pueden enviarse comenzando por el byte que ya se ha enviado acuse de recibo. Si el campo de *ventana* es 0 indica que se han recibido los bytes hasta *número de acuse de recibo* –1, inclusive, pero que el receptor necesita un descanso y quisiera no recibir más datos por el momento.

Se proporciona una *suma de comprobación* para confiabilidad extrema. Es una suma de comprobación de la cabecera, los datos y la pseudocabecera conceptual.

Gestión de una conexión TCP

En el TCP se establecen las conexiones usando el protocolo de acuerdo de tres vías. Para establecer una conexión, un lado espera pasivamente una conexión entrante ejecutando las primitivas LISTEN y ACCEPT y especificando cierto origen o bien nadie en particular.

El otro lado, ejecuta una primitiva CONNECT especificando la dirección y el puerto IP con el que se desea conectar, el tamaño máximo de segmento TCP y algunos datos de usuario.

Al llegar el segmento al destino, la entidad TCP ahí revisa si hay un proceso que haya ejecutado un LISTEN en el puerto indicado en el campo *puerto de destino*. Si no lo hay, envía una contestación con el bit RST encendido para rechazar la conexión.

Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión; si la acepta, se devuelve un segmento de acuse de recibo.

Aunque las conexiones TCP son dúplex integral, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones simplex. Cada conexión simplex se libera independientemente de su igual. Para liberar una conexión, cualquiera de las partes puede enviar un segmento con el bit FIN establecido, lo que significa que no tiene más datos que transmitir. Al reconocerse el FIN, ese sentido se apaga. Sin embargo, puede continuar un flujo de datos indefinido en el otro sentido. Cuando ambos sentidos se han apagado, se libera la conexión. Normalmente se requieren cuatro segmentos TP para liberar una conexión, un FIN y un ACK para cada sentido.

Para evitar el problema de los dos ejércitos, se usan temporizadores. Si no llega una respuesta a un FIN en un máximo de dos tiempos de vida de paquete, el transmisor del FIN libera la conexión. Tarde o temprano el otro lado notará que ya nadie lo está escuchando, y también terminará su temporización.

Control de congestionamiento con TCP

Cuando la carga ofrecida a cualquier red es mayor que la que se puede manejar, se genera un congestionamiento. Aunque la capa de red intenta manejarlos, gran parte del trabajo recae sobre el TCP porque la solución real al congestionamiento es la disminución de la tasa de datos.

Antes de analizar la manera en que el TCP reacciona al congestionamiento, veamos qué hace para evitar que ocurra: al establecerse una conexión, se tiene que seleccionar un tamaño de ventana adecuado. El receptor puede especificar una ventana con base en su tamaño de *buffer*. Si el transmisor se ajusta a su tamaño de ventana, no ocurrirán problemas por desbordamiento de *buffers* en la terminal receptora, pero aún pueden ocurrir debido a congestionamientos internos en la red.

La solución de Internet es aceptar que existen dos problemas potenciales (capacidad de la red y capacidad del receptor) y manejarlos por separado. Para ello, cada transmisor mantiene dos ventanas: la ventana que ha otorgado el receptor y una segunda ventana, la **ventana de congestionamiento**; cada una refleja la cantidad de bytes que puede enviar el transmisor. La cantidad de bytes que pueden enviarse es la cifra menor de las dos ventanas. Por tanto, la ventana efectiva es el mínimo de lo que el transmisor piensa que es correcto y lo que el receptor piensa que está bien.

Gestión de temporizadores del TCP

El TCP usa varios temporizadores para hacer su trabajo. El más importante es el **temporizador de retransmisión**. Al enviarse un segmento, se inicia un temporizador de retransmisiones, si el acuse de recibo del segmento llega antes de expirar el temporizador, éste se detiene. Si, el temporizador termina antes de llegar el acuse de recibo, se retransmite el segmento. Para determinar el intervalo de terminación del temporizador debe usarse un algoritmo muy dinámico que ajuste constantemente el intervalo de terminación de temporización, con base en mediciones continuas del desempeño en la red. Ya que si se hace demasiado corto, ocurrirán retransmisiones innecesarias, cargando la red con paquetes inútiles, y si se hace demasiado largo, el desempeño sufrirá debido al gran retardo de retransmisión de cada paquete perdido.

Otro temporizador usado por el TCP es el **temporizador de persistencia**, para evitar el siguiente interbloqueo: el receptor envía un acuse de recibo con un tamaño de ventana de 0, indicando al transmisor que espere. Después, el receptor actualiza la ventana, pero se pierde el paquete con la actualización. Ahora, tanto el transmisor como el receptor están esperando que el otro haga algo. Cuando termina el temporizador de persistencia, el transmisor envía una prueba al receptor, la respuesta de la prueba indica el tamaño de la ventana. Si aún es de cero, se inicia nuevamente el temporizador de persistencia, si es diferente de cero, pueden enviarse datos.

Un tercer temporizador es el **temporizador de seguir con vida**. Cuando una conexión ha estado ociosa durante demasiado tiempo, el temporizador de seguir con vida puede terminar, haciendo que un lado compruebe que el otro aún está ahí. Si no recibe respuesta, se termina la conexión.

El último temporizador usado en cada conexión TCP es el que se usa en el estado *TIMED WAIT* durante el cierre; opera durante el doble del tiempo máximo de vida de paquete, para asegurar que, al cerrarse una conexión, todos los paquetes creados por ella hayan muerto.