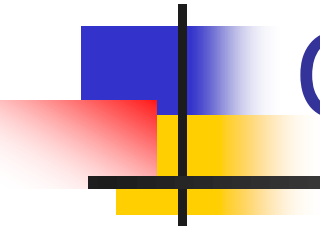
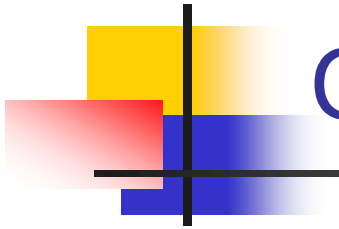


Capítulo 6

La capa de transporte CT



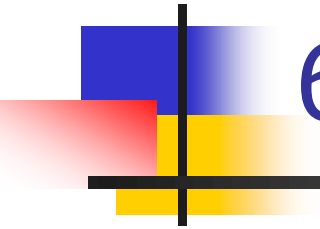
Contenido

6.1 El servicio de transporte

6.2 Elementos de los protocolos de transporte

6.3 El protocolo de Internet UDP

6.4 El protocolo de Internet TCP



6.1 El servicio de transporte

6.1.1 Servicios proporcionados a las capas superiores

- Conocer el funcionamiento de la CT es especialmente útil para quienes desarrollan aplicaciones de red
- Esta capa da un transporte de datos confiable **extremo a extremo** independientemente de la o las redes físicas
- Los usuarios de los servicios de la CT son los procesos de la capa de aplicación
- Para esto, la CT recibe los servicios de la capa de red CR

Entidad de transporte

- La CT está implementada en hardware y en software
- Esta entidad puede estar en:
  - El kernel del sistema operativo
  - Un proceso de usuario independiente
  - Una librería de las aplicaciones de red, o
  - La tarjeta de red

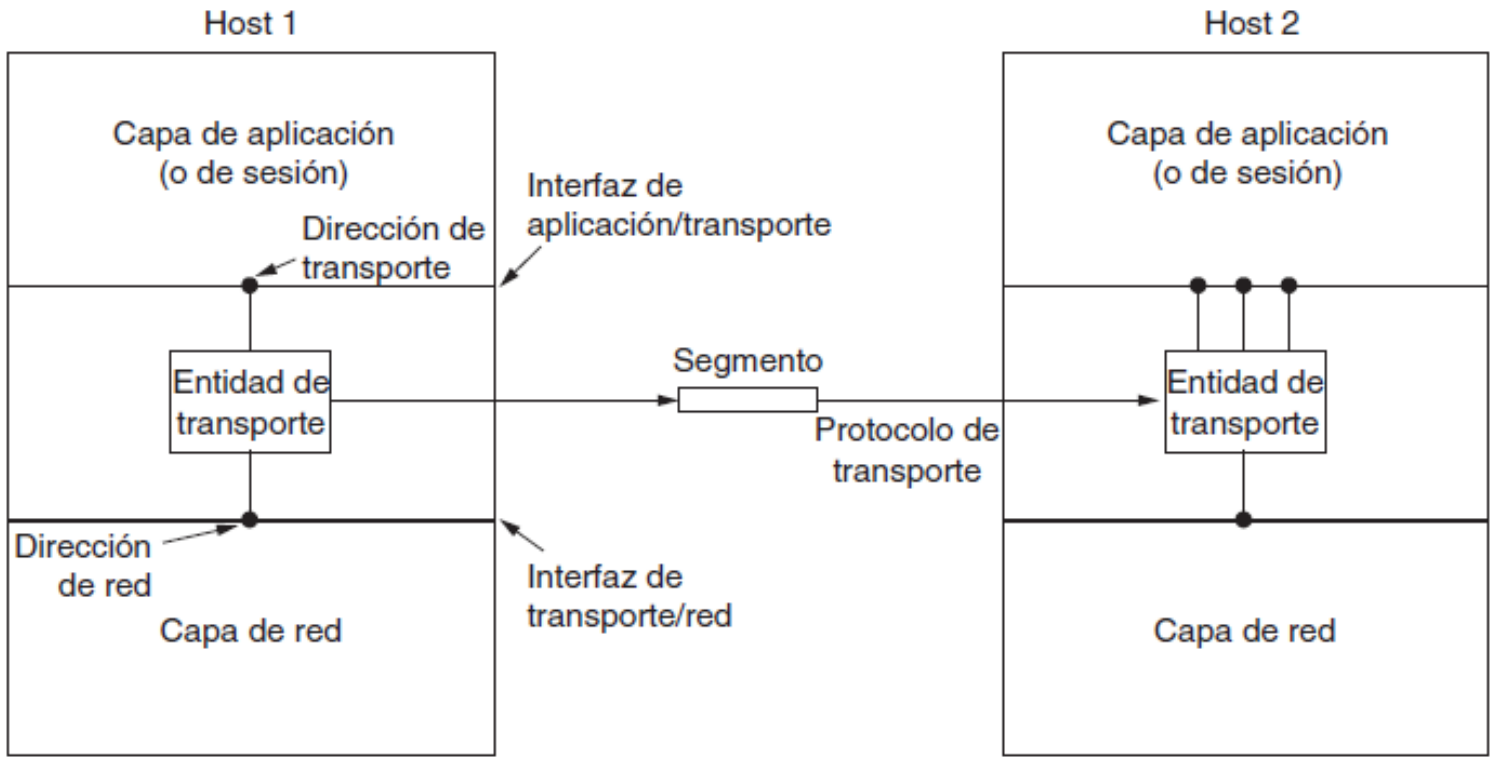
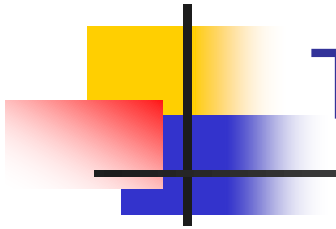


Figura 6-1. Las capas de red, transporte y aplicación.



Tipos de servicio de la CT

- Similar a los servicios de la CR, hay dos tipos de servicio que se ofrecen a la capa de aplicación CA:
  - orientado a la conexión
  - sin conexión

¿Dos capas que ofrecen los mismos servicios?

- La capa de red CR y la CT dan servicios sin conexión y orientado a conexión
- La CR se ejecuta en los enrutadores de los operadores
- Los usuarios finales no controlan la CR
- El proveedor a veces no cumple el compromiso de dar un servicio confiable: pierden paquetes, dañan paquetes, o la red se cae
- Se tiene que poner otra capa sobre la CR para mejorar la calidad del servicio
- Si accidentalmente se pierde una conexión de red, la CT establece automáticamente otra conexión de red con la CT remota

- La CT facilita a los usuarios crear aplicaciones de red sin tener que lidiar con la subred
- Subred: medios de transmisión y dispositivos de los proveedores del servicio de transmisión
  - Empresas públicas de telecomunicaciones PSTN

  - ISPs

6.1.2 Primitivas de servicio de transporte

- Para que las aplicaciones puedan acceder al servicio de transporte, la CT facilita algunas primitivas de servicio
- Primitivas del servicio de orientado a conexión

Primitiva	**Paquete enviado**	**Significado**
LISTEN	Ninguno	Se bloquea
CONNECT	Req	Intenta establecer
SEND	Datos	
RECEIVE	Ninguno	Se bloquea
DISCONNECT	Req	Intenta liberar

Diferencias entre los servicios de transporte y de red

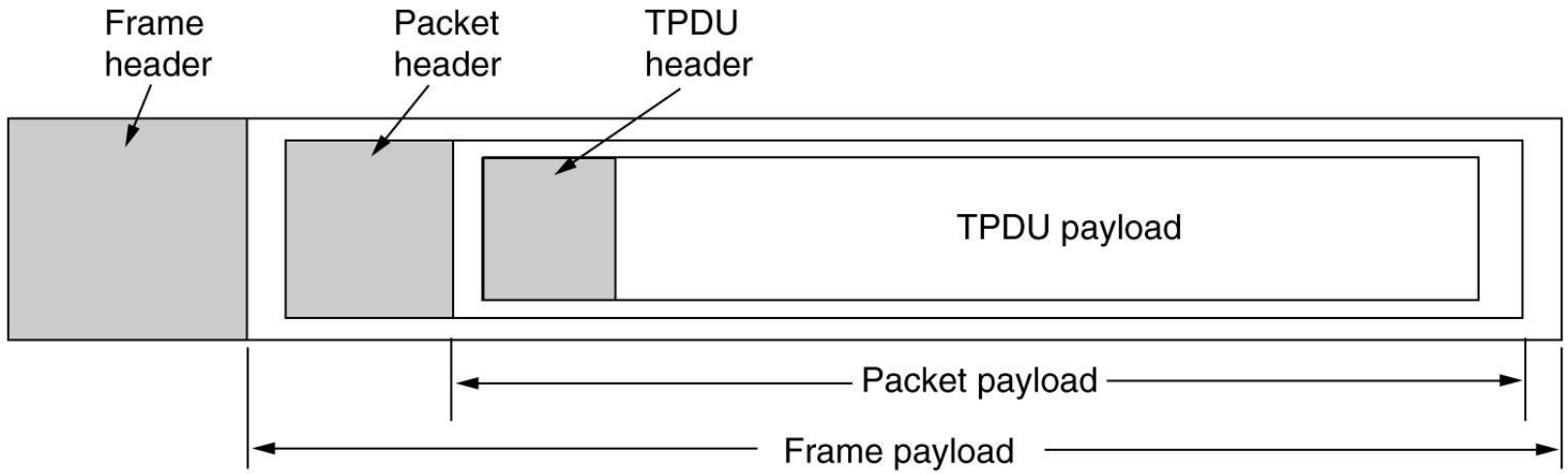
- **Primera diferencia**
- El servicio de la CR depende de las redes físicas. Estas pierden paquetes, producen errores, o se caen. Servicio no confiable
- En cambio, el servicio de CT orientado a conexión sí es confiable
- CT da un servicio confiable sobre una red no confiable
- CT oculta los defectos de la CR para que los procesos de usuario sientan un flujo de bits libre de errores y pérdidas
- **Segunda diferencia**
- Los programadores tratan con las primitivas de transporte, que son fáciles de usar. Casi nunca con las primitivas de red

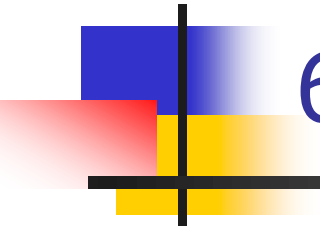
Funciones de las entidades de transporte

1. Confirmaciones ACKs
2. Controles de errores y pérdida de paquetes
3. Manejo de temporizadores
4. Retransmisiones

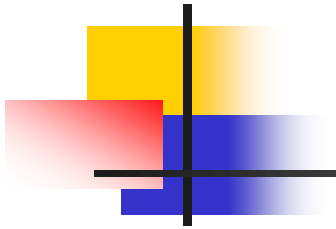
- Son funciones transparentes para las aplicaciones

Anidamiento





6.2 Elementos de los protocolos de transporte



- El servicio de transporte se implementa mediante un protocolo de transporte
- Los protocolos de capa 4 se parecen a los protocolos de capa 2 porque controlan:
  - Errores
  - Secuenciación
  - Flujo

Diferencias entre los dos protocolos

- En la capa 2, dos sistemas se conectan P2P a través de un enlace físico
- En la capa 4 los dos sistemas se conectan *end-to-end* a través de una subred
- En capa 2 el enrutador no necesita especificar la dirección del otro enrutador
- En capa 4 hay que indicar la dirección destino
- En capa 2 es más sencillo que en capa 4 establecer conexiones
- La subred almacena y reenvía paquetes. Esto produce complicaciones: retardos, pérdida, duplicación

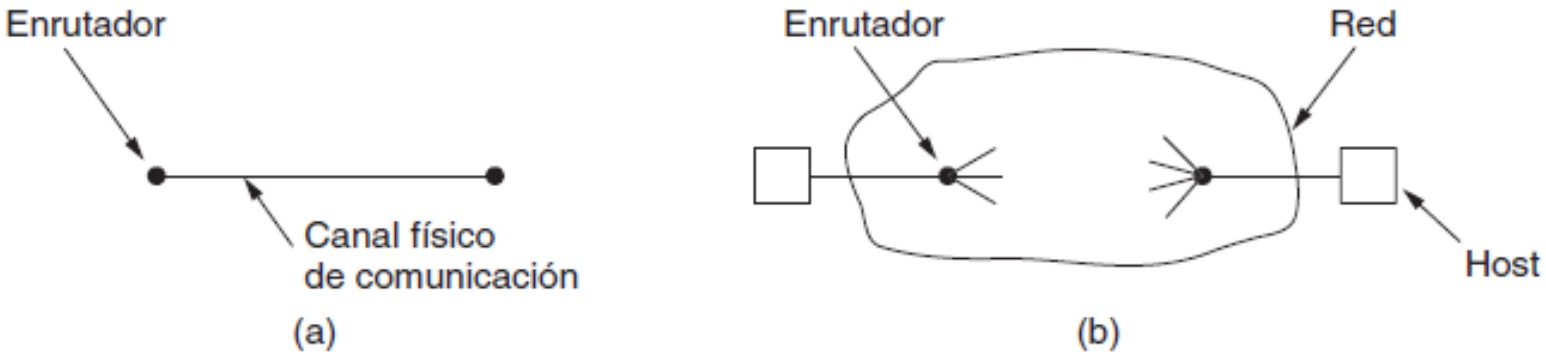
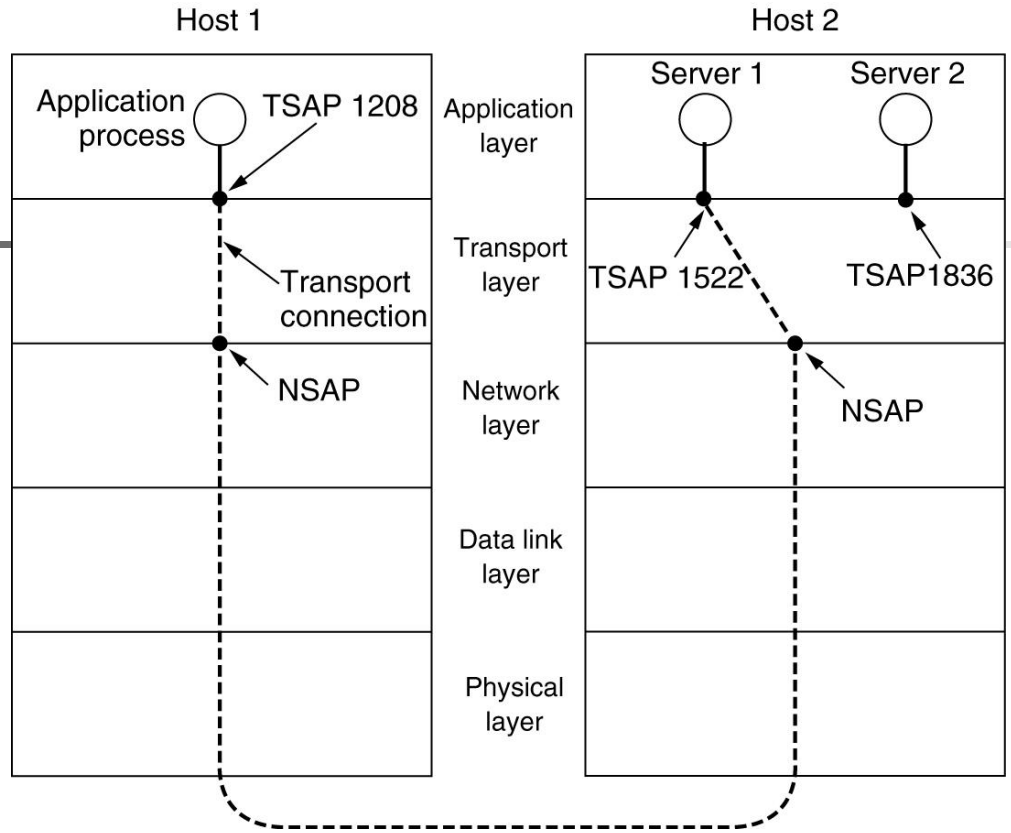


Figura 6-7. (a) Entorno de la capa de enlace de datos. (b) Entorno de la capa de transporte.

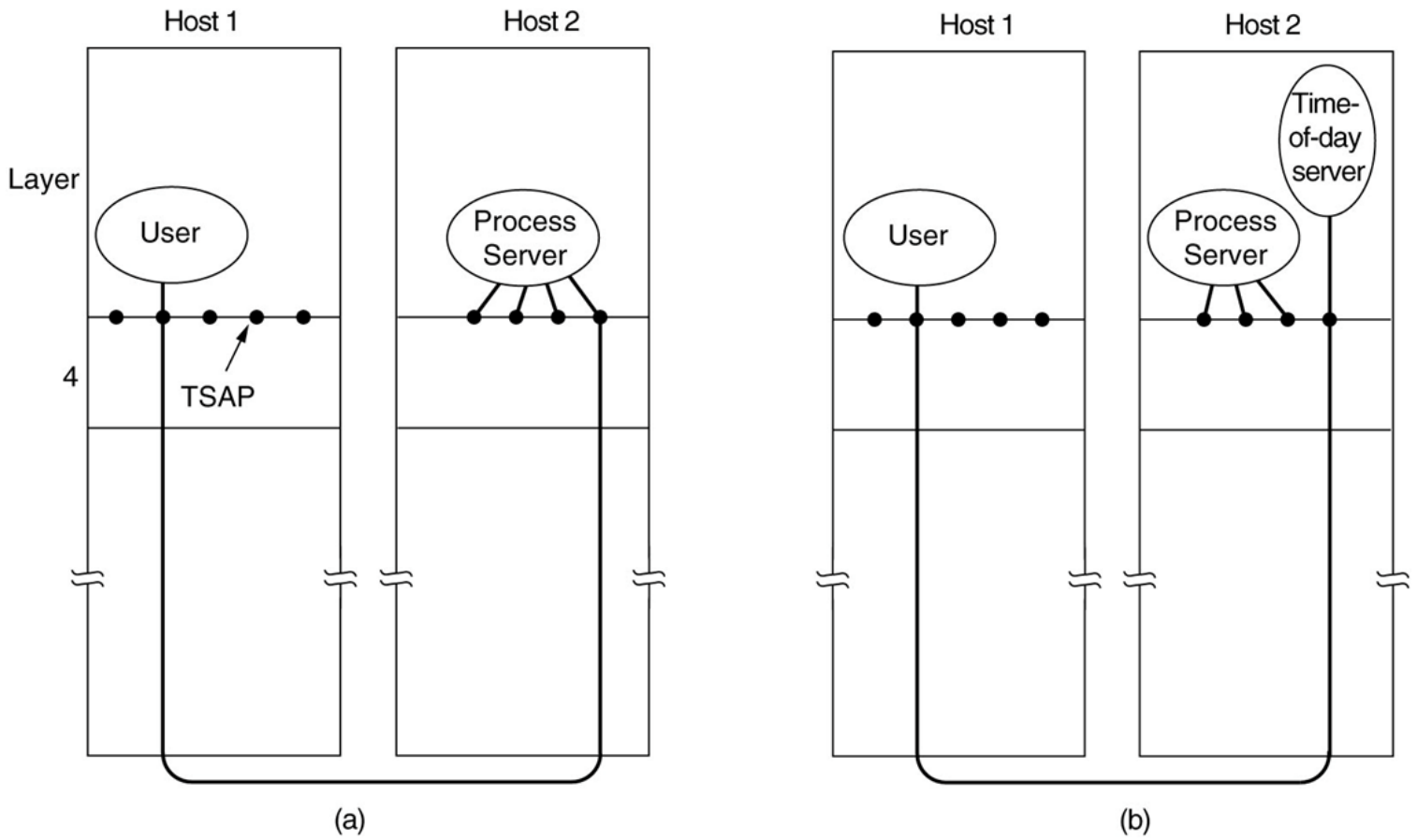
6.2.1 Direccionamiento

- A más de la dirección de red, los procesos, local y remoto que se comunican necesitan una dirección de transporte
- Los procesos pueden estar a la escucha de solicitudes de conexión
- En Internet estas direcciones se llaman puertos o TSAP



- El proceso en el host 1 sabe que el servidor de hora del día está conectado al TSAP 1522
- Hasta 1023 son servicios bien conocidos /etc/services

Servidor de procesos *inetd*
Apoderado o proxy de servidores de menor uso



6.2.2 Establecimiento de una conexión

- La red puede perder o duplicar paquetes. Esto complica el manejo de las conexiones
- Un usuario establece una conexión con un banco para transferir dinero
- Si cada paquete de la transacción se duplica, se haría más de una transferencia de dinero

Forma de solucionar duplicación de paquetes

- Uso de direcciones de transporte desechables
  - Cada vez que se requiere una transacción se genera un nueva dirección de transporte
  - Al cerrar la conexión se desecha la dirección y no se vuelve a usar
  - Si aparece un paquete duplicado, con una dirección anterior, éste se descarta

6.2.3 Liberación de una conexión

- Es más fácil liberar que establecer una conexión
- Hay 2 estilos de terminar una conexión:
  - Asimétrica
  - Simétrica

Liberación asimétrica

- Análogo al sistema telefónico
- Si una parte cuelga se interrumpe la conexión
- Es abrupta y puede haber pérdida de datos

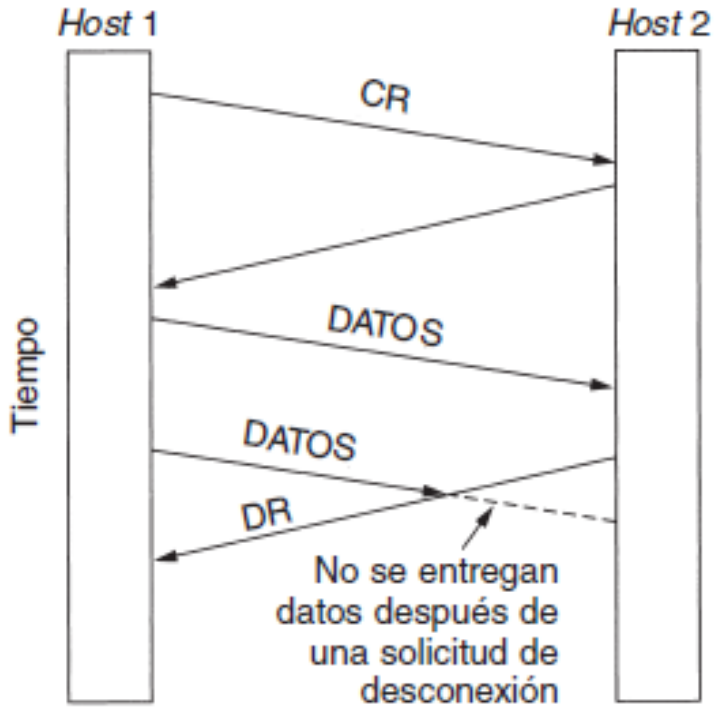
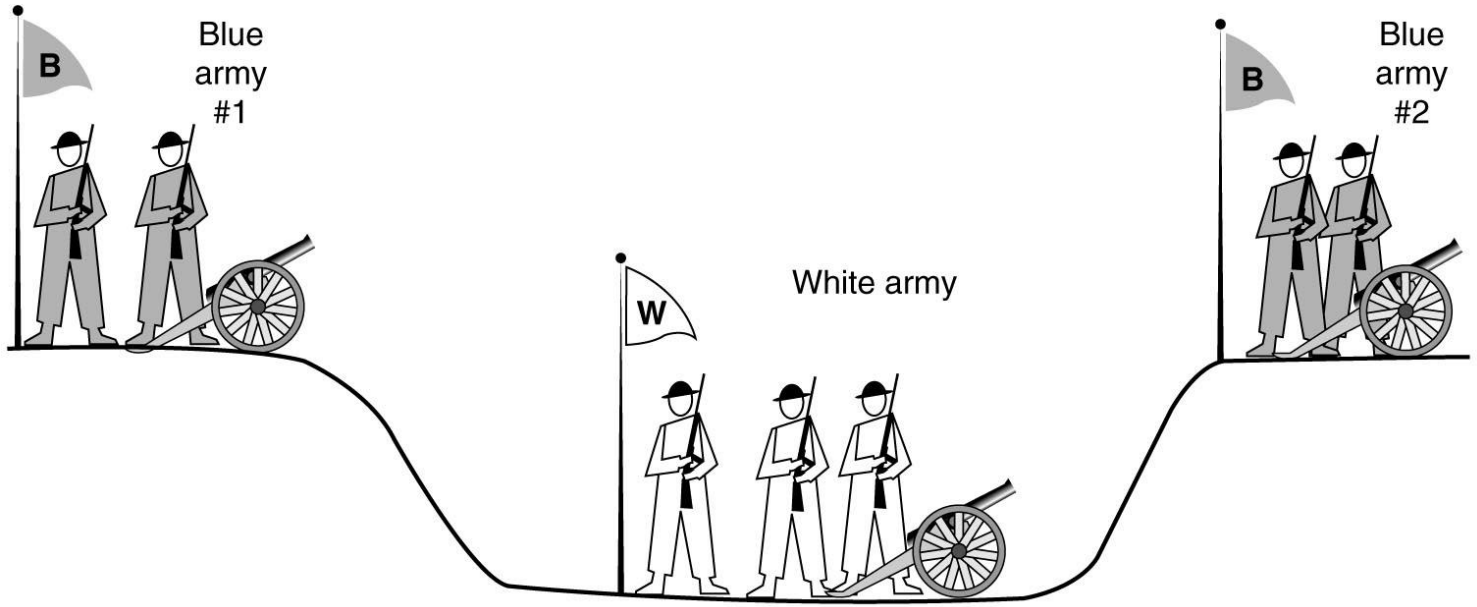


Figura 6-12. Desconexión abrupta con pérdida de datos.

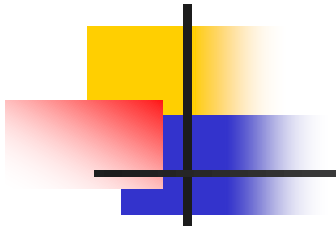
CR connection request; DR disconnection requiest

Liberación simétrica

- Protocolo más refinado
- Evita pérdida de datos
- Consta de 2 conexiones unidireccionales
- Cada dirección se libera por separado
- Un host puede recibir datos luego de enviar un DR



Los ejércitos azules deben sincronizarse para atacar simultáneamente al ejército blanco para poder vencer



- Problema: el emisor del mensaje final nunca sabrá que su mensaje llegó
- Si ninguna de las partes sabe que la otra está lista para desconectares, nunca ocurrirá la desconexión
- El protocolo podría fallar
- Un acuerdo de tres vías por lo general funciona bien

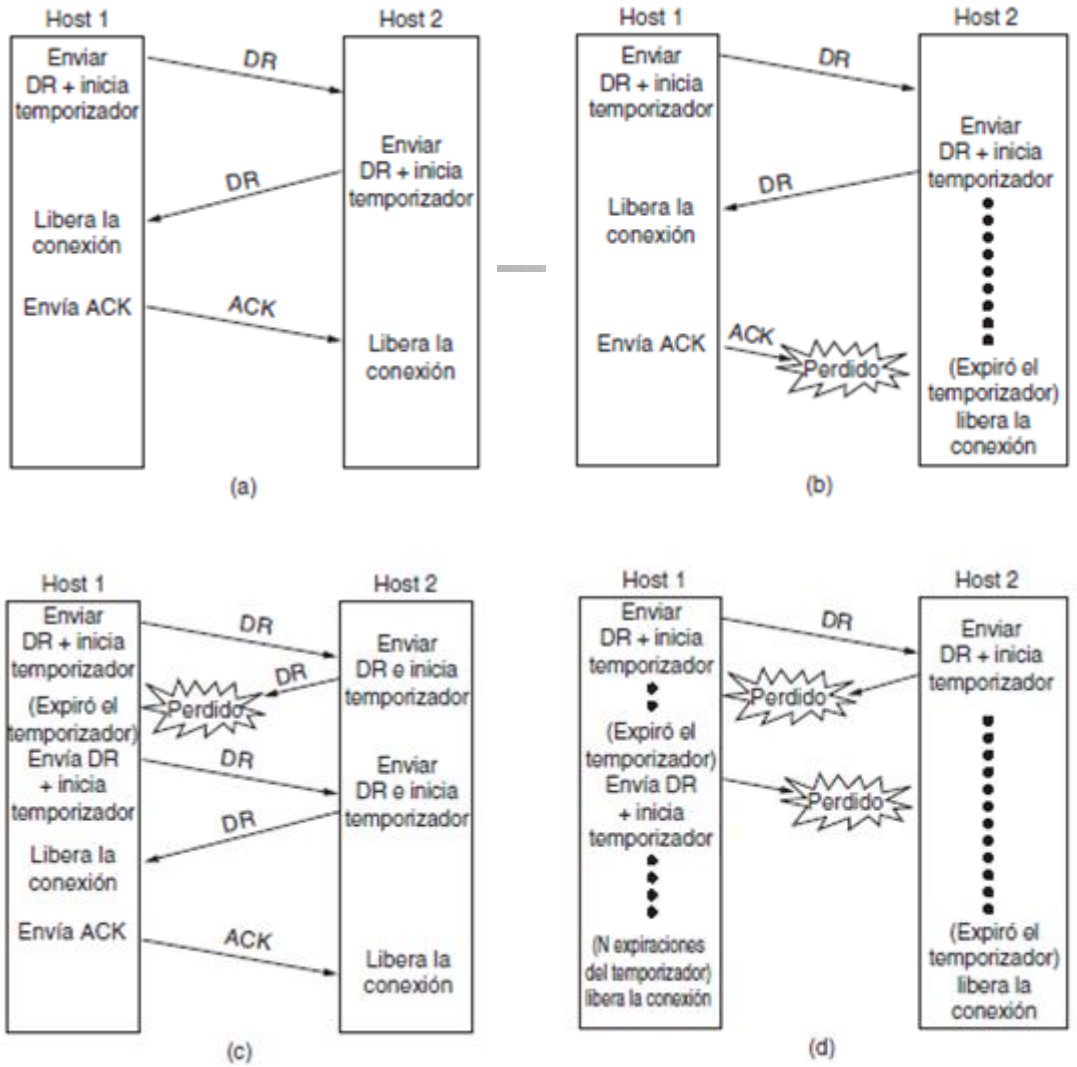
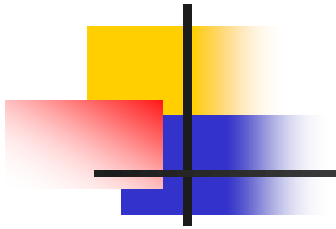


Figura 6-14. Cuatro escenarios de un protocolo para liberar una conexión. (a) Caso normal del acuerdo de tres vías. (b) Pérdida del último ACK. (c) Respuesta perdida. (d) Respuesta perdida y pérdida de los segmentos DR subsecuentes.

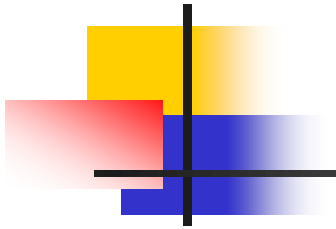
DR Disconnection Request

6.2.4 Control de flujo y almacenamiento en búfer en el manejo de las conexiones

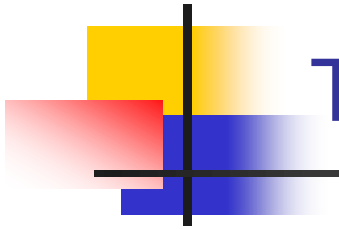
- El control de flujo en la CT es similar al de la capa 2
- Las CTs tienen una mecanismo para no desbordar al receptor:
  - Parada espera: El emisor envía un paquete y espera un ACK
  - Ventana deslizante: El emisor envía varios paquetes y espera un ACK



- La diferencia en la CT está en que un router tiene pocas líneas, y un host puede tener muchas conexiones
- Un enrutador no tiene CT
- En la capa 2 existen buffers por línea. En la CT existen buffers por conexión
- El emisor tiene buffers de salida que almacenan las TPDU hasta recibir ACK del receptor
- El receptor tiene buffers de entrada para almacenar las TPDU mientras se procesa una TPDU



- Se puede manejar:
  - Un grupo de buffers para todas las conexiones de transporte, o
  - Un grupo de buffers para cada conexión
- La cuestión es el tamaño de los buffers



Tamaño de los buffers

- Se asignan buffers por conexión
- Hay maneras según el caso:

1. TPDUs del casi del mismo tamaño: Cadena de buffers de tamaño fijo
2. TPDUs de tamaño variable: Cadena de buffers de tamaño variable
3. Un solo buffer grande circular

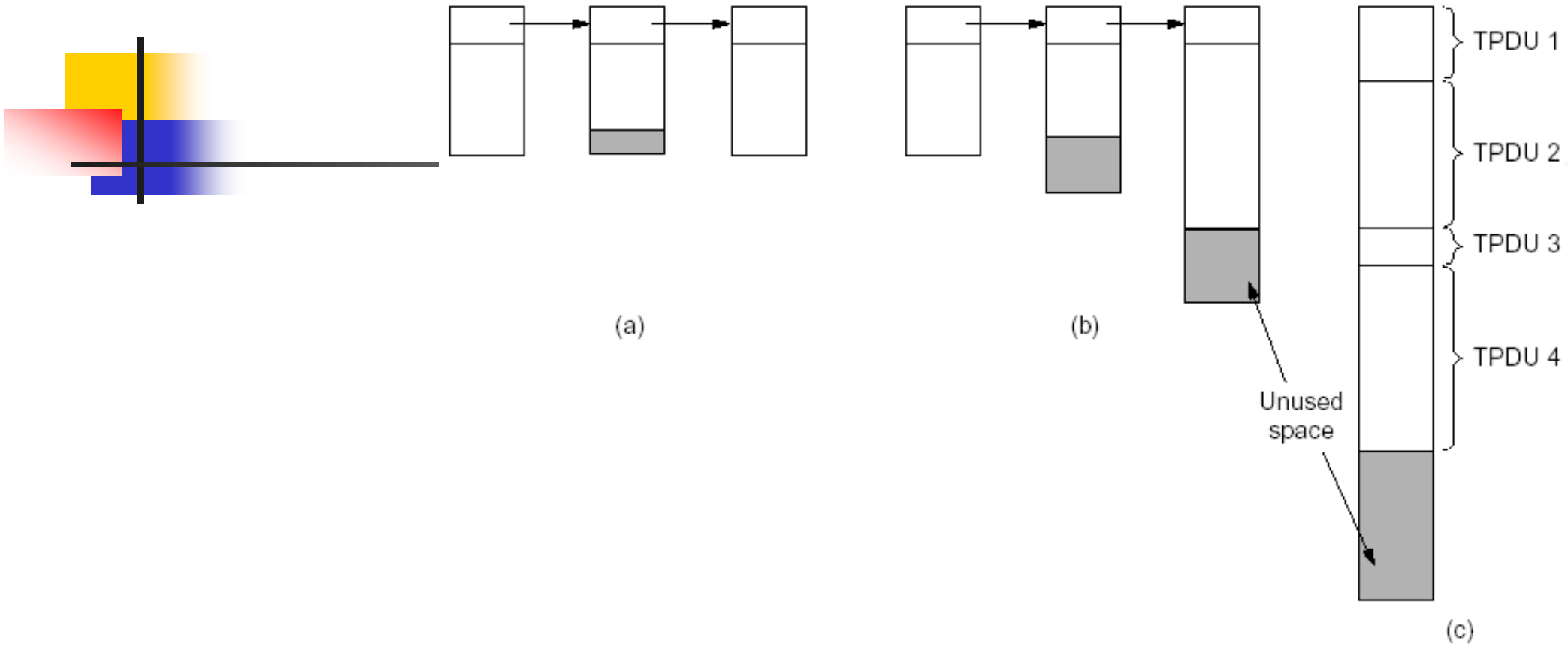
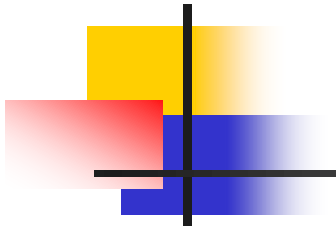
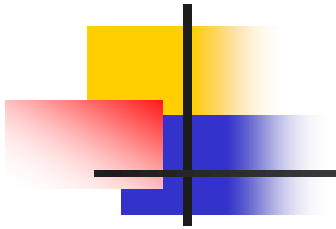


Fig. 6-15. (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.



- La CT permite que el host emisor pida al receptor que se provea de suficientes buffers para que pueda recibir información
- Cuando se abren conexiones se asigna un número de tráfico
- El número de buffers en el origen y destino depende del tráfico que cambia dinámicamente
- Para tráfico con ráfagas no se asigna un número fijo de buffers, sino se lo hace según la magnitud de la ráfaga



- La RAM es económica. Los hosts vienen con memoria suficiente. Los protocolos de CT no tienen este limitante
- El cuello de botella está el número y capacidad de las líneas de la subred

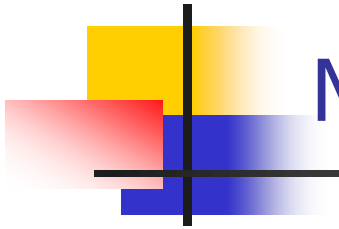
6.2.5 Multiplexación

- La multiplexión de varias conversaciones puede darse de tres maneras:

1. Un enlace físico
2. Un circuito virtual de capa 3 de OSI
3. Una conexión de capa 4

Multiplexión hacia arriba

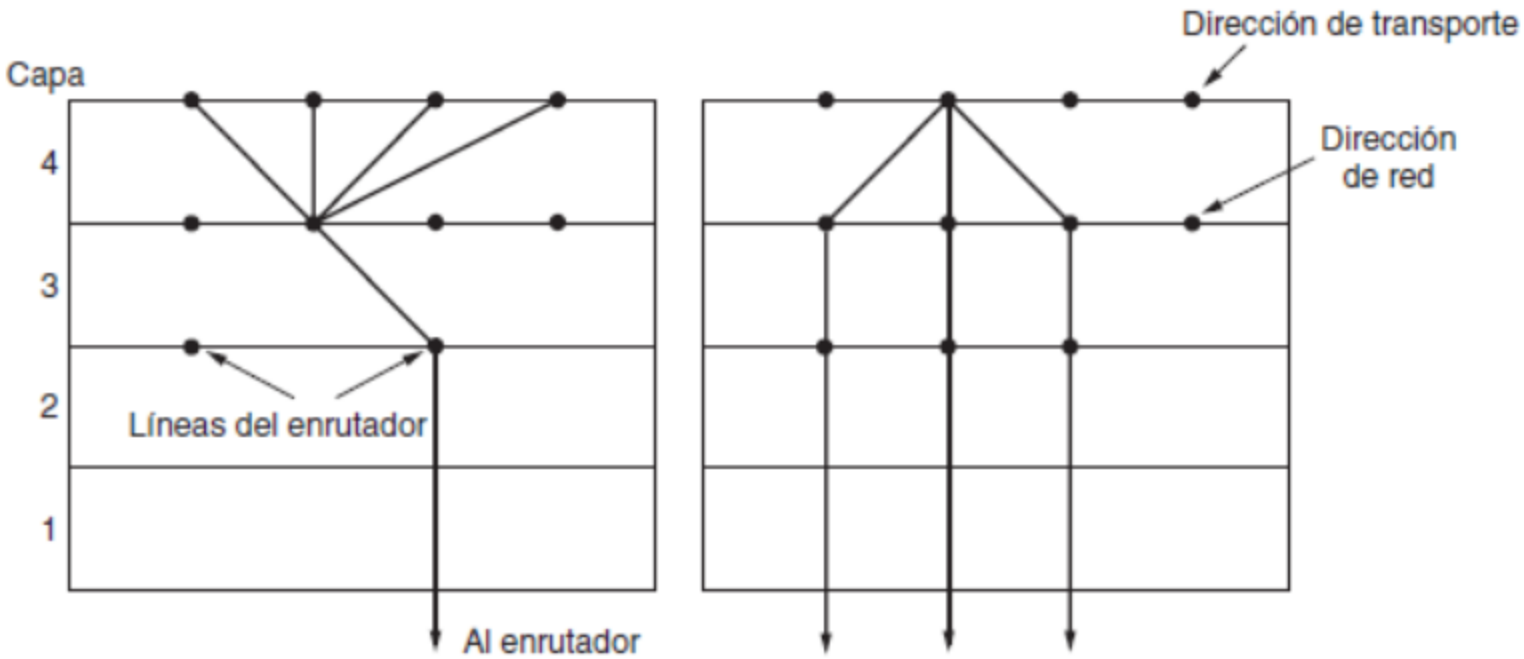
- En el Modelo TCP/IP, la capa de Internet no ofrece circuitos virtuales, solo datagramas (servicio sin conexión), por lo tanto no hay multiplexión
- En la CT surge la necesidad de multiplexar porque se tienen varios puertos y una sola dirección de red (IP en el modelo TCP/IP)
- Todas las conexiones de transporte tienen que compartir esta dirección



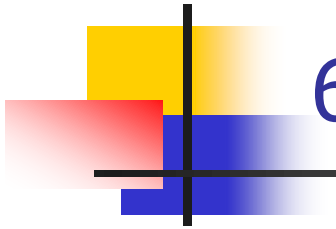
Multiplexión hacia abajo

- En el Modelo OSI, La subred provee circuitos virtuales
- Si un usuario necesita más ancho de banda del dado en un CV, se puede abrir más CVs y distribuir el tráfico entre ellos
- Para ello, se necesitan varias direcciones de red
- En el Modelo TCP/IP los hosts solo tienen una dirección de red y no hay multiplexión hacia abajo

Modelo OSI

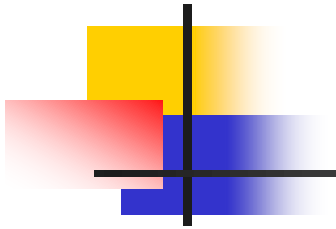


(a) Multiplexión hacia arriba. (b) Multiplexión hacia abajo.

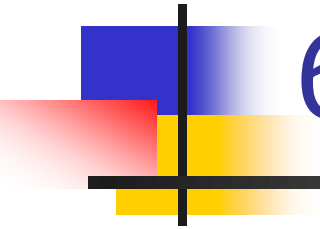


6.2.6 Recuperación de caídas

- Servidores, clientes y routers están sujetos a caídas
- Si la CR da servicio de datagramas, estos se pueden perder
- Si la CR da servicio orientado a conexión, se pueden perder los CVs
- Estos problemas lo maneja la CT
- Es deseable que los clientes sigan trabajando mientras un servidor que se recupera de una caída
- El problema es más complejo si se cae el host

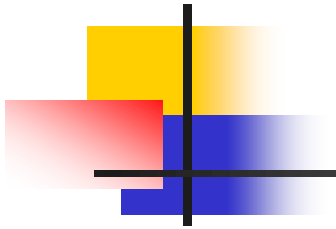


- Para recuperar su estado previo, el servidor difunde una *TPDU* a todos los hosts con los que hubo conexiones para:
  - Anunciar que acaba de reiniciarse y
  - Solicitar información del estado de todas las conexiones abiertas



6.3 El protocolo de Internet UDP

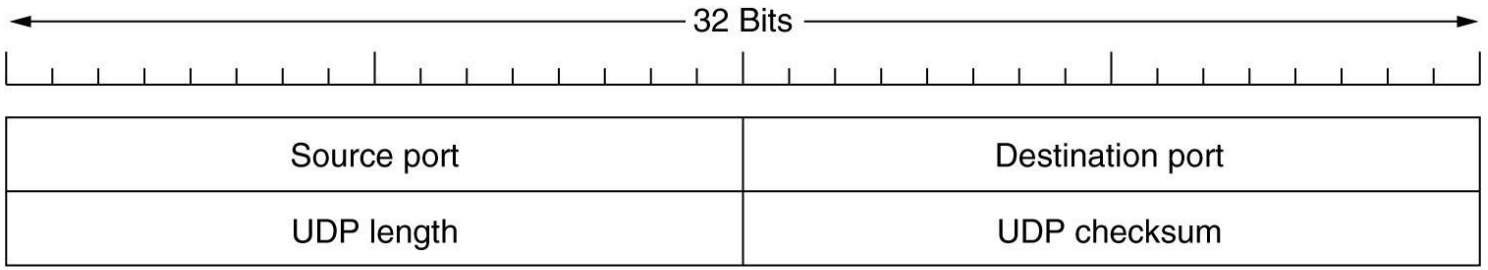
UDP User Datagram Protocol

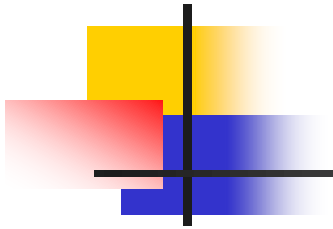


- Internet tiene dos protocolos en la capa de transporte:
  - Uno orientado a la conexión TCP
  - Sin conexión UDP

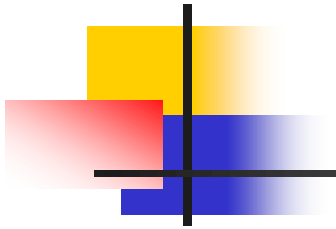
6.3.1 Introducción a UDP

- UDP permite que las aplicaciones funcionen sin conexión
- UDP divide los mensajes de las aplicaciones y los encapsula en segmentos
- Segmento: encabezado de 8 bytes + porción de mensaje (carga útil)
- ¿Por qué no usar solo datagramas IP?
- Se usa UDP en lugar de solo IP para indicar los puertos origen y destino
- Sin los puertos, la CT no sabría a qué aplicación enviar el segmento
- El puerto origen se necesita cuando se debe enviar una respuesta al emisor


32 Bits
Source port
Destination port
UDP length
UDP checksum



- UDP no realiza:
  - Control de flujo
  - Control de errores
  - Control de pérdidas
  - Retransmisión
- Esto le corresponde hacer a los procesos de usuario, si lo necesita
- UDP ofrece a las aplicaciones una interfaz al protocolo IP
- Multiplexa varios procesos utilizando puertos



- UDP es útil en ciertas aplicaciones cliente-servidor
- El cliente envía una solicitud corta al servidor
- El cliente espera una respuesta corta
- Si se pierde la solicitud o respuesta, el cliente intenta de nuevo
- Ejemplo: la aplicación DNS utiliza UDP: Obt_direccion_IP(nombre_de_host)

6.3.2 Llamada a procedimiento remoto RPC

- Es enviar un mensaje a un proceso que corre en otro host y obtener respuesta, como si se llamara a una *función* en un lenguaje de programación
- RPC hace que las aplicaciones de red sean más fáciles de programar y usar
- Así, los detalles de conexión se ocultan al programador de aplicaciones
- RPC *Remorte Procedure Call* se ideó en 1984 y se usa en sistemas distribuidos

Pasos para realizar un RPC

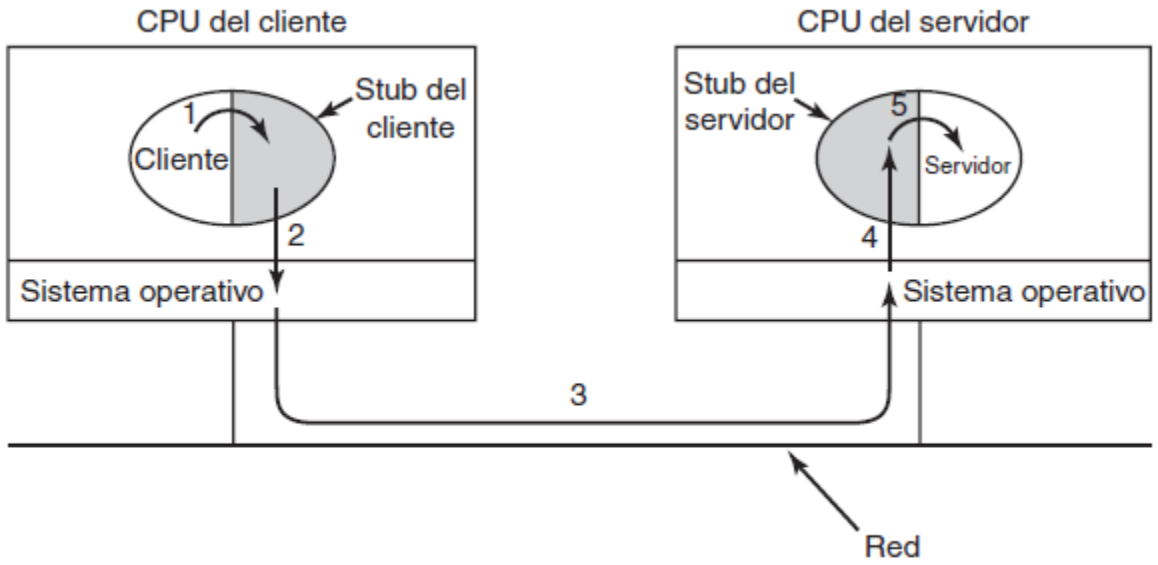
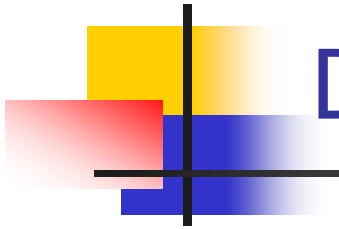


Figura 6-29. Pasos para realizar la llamada a un procedimiento remoto. Los stubs están sombreados.

Pasos para realizar un RPC

1. El cliente llama al *stub* del cliente
2. *El stub del cliente* empaqueta los parámetros en un mensaje y llama al SO para enviarlo al servidor
3. El SO envía el mensaje al servidor
4. En el servidor, el SO pasa el paquete entrante al *stub del servidor*
5. El *stub del servidor* llama al procedimiento servidor con los parámetros desempacados

- La respuesta sigue la misma ruta en forma opuesta
- No se utilizan sockets

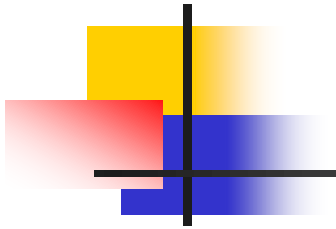


Desventajas de RPC

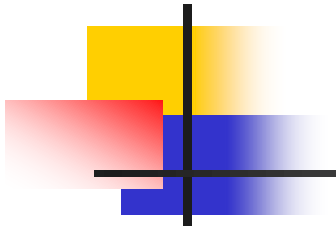
- Con RPC el paso de punteros es imposible porque los procesos residen en diferentes espacios de direcciones
- RPC se utiliza, con restricciones, muy ampliamente

6.3.3 Protocolo de transporte en tiempo real RTP

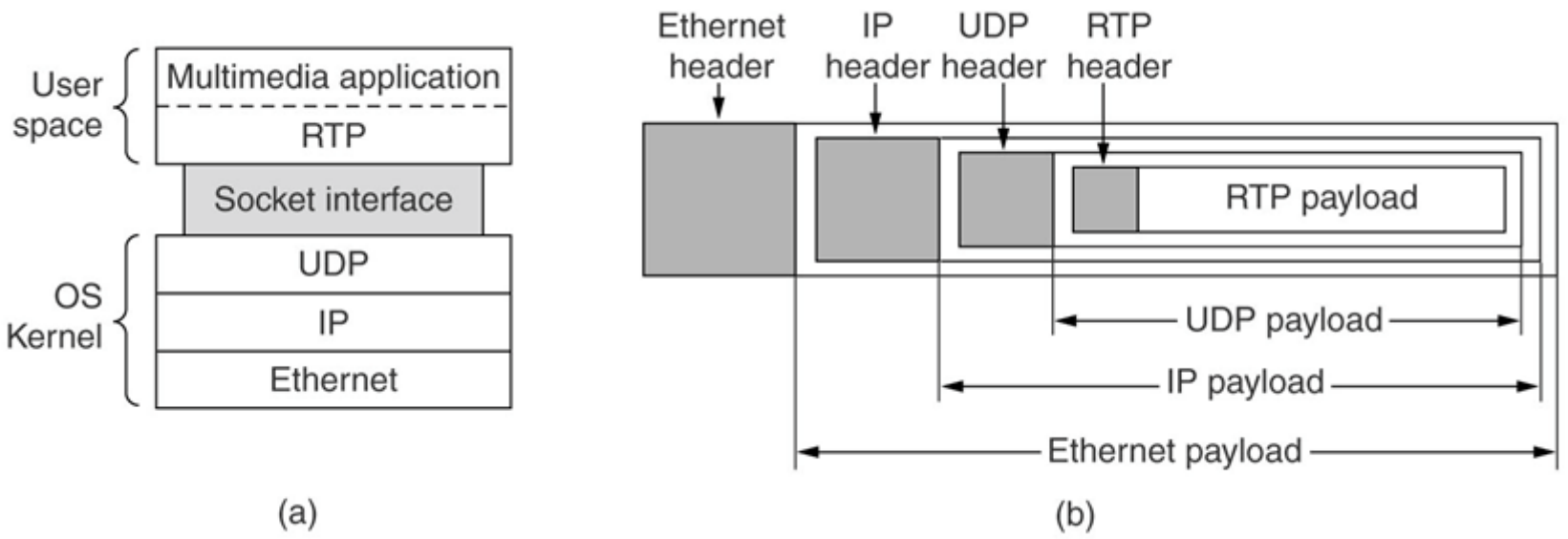
- *Tiempo real*: en vivo, momento actual, ahora
- *Información en tiempo real*: estar al tanto en forma inmediata del evento que ocurre ahora
- *Aplicación de tiempo real*: aplicación que informa de un evento con restricciones de retardo de transmisión y propagación
- *Tiempo de transmisión*: tiempo empleado por una estación para colocar todos los bits de una trama en el medio de transmisión
- *Tiempo de propagación*: Tiempo empleado por un bit en atravesar la red desde el origen al destino



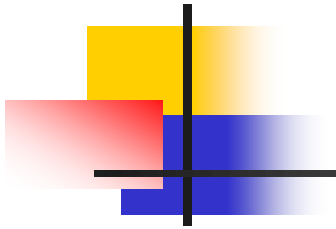
- UDP también se usa en aplicaciones en tiempo real
  - Sistemas de control de tráfico aéreo
  - Sistemas de monitoreo de signos vitales de un paciente
  - Aplicaciones para realizar operaciones financieras en la bolsa de valores
  - Sistemas de reserva de vuelos
  - Sistemas de control de procesos industriales: t^0, presión, rpm, . . .
  - Sistemas de control de generación distribución y consumo de energía eléc
  - Rastreo satelital para la industria del transporte
  - Videoconferencia



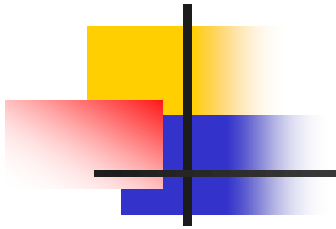
- Aplicaciones en tiempo real requieren un protocolo de transporte de tiempo real
- RTP Real-time Transport Protocol
- RTP se ejecuta sobre UDP



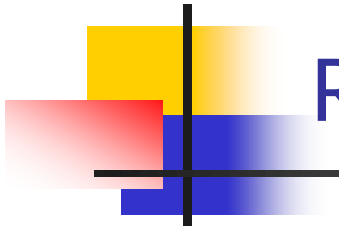
HDLC (High-Level Data Link Control) protocolo de capa de enlace punto a punto



- La aplicación de multimedia tiene **múltiples** flujos: audio, video y texto
- RTP multiplexa los flujos de tiempo real, función básica de RTP
- Luego se encapsula en paquetes RTP
- RTP es un protocolo independiente de la aplicación
- RTP se parece más a un protocolo de transporte
- RTP es un protocolo de transporte implementado en la capa de aplicación



- Cada paquete enviado en el flujo RTP tiene un número secuencial
- Si falta uno, el receptor lo aproxima por interpolación
- La retransmisión no es práctica, pues llegaría muy tarde para ser útil
- No hay garantía de entrega

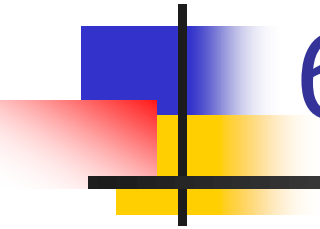


RTP no hace:

- Control de flujo
- Control de errores
- Confirmación de recepción
- Retransmisiones

Timestamping

- Marcación del tiempo
- Es una característica de las aplicaciones en tiempo real
- El destino almacena una pequeña cantidad de paquetes y lo reproduce unos milisegundos después del inicio del flujo
- Reduce el efecto de la fluctuación
- Permite que la sincronización de múltiples flujos: video, voz, música
- Ejemplo: TV digital estéreo y doblada en varios idiomas



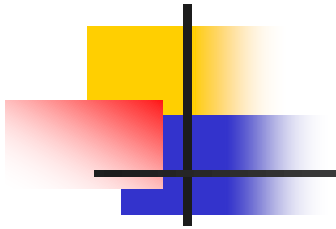
6.4 El protocolo de Internet TCP

TCP Transport Control Protocol

6.4.1 Introducción a TCP

- TCP (1981) da un flujo de bytes confiable de extremo a extremo a través de la subred no confiable
- La entidad TCP se implementa:
  - en el kernel del SO, que es lo más común
  - en un procedimiento de biblioteca
  - en la aplicación de usuario

- El tamaño máximo de un fragmento TCP, al igual que un segmento UDP es de 64 Kbytes = $64 \times 1024 - 1 = 2^{16} - 1 = 65.535$ bytes
- Los datos de usuario son 65535 bytes–el tamaño de la cabecera TCP o UDP
- IP soporta una carga de 65.535 bytes
- La aplicación de usuario funciona sobre un servicio orientado a la conexión de TCP que, a su vez recibe un servicio sin conexión de IP
- En la práctica, un segmento TCP lleva 1460 bytes de datos para que calce en una trama Ethernet, que puede llevar datagramas de hasta 1.500 bytes



- Funciones de TCP:

1. Enviar los segmentos sin causar congestionamiento en la red
2. Realizar retransmisiones de ser necesario
3. Ordenar los fragmentos que llegan al destino en desorden

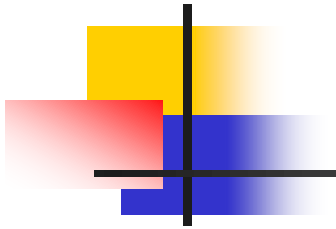
- Así, TCP proporciona confiabilidad en la comunicación, que no lo proporciona IP

6.4.2 El modelo del servicio TCP

- Emisor y receptor crean cada uno un punto terminal llamado socket
- Socket = dirección IP del host + puerto de 16 bits
- Un puerto es un TSAP
- Hay que establecer explícitamente una conexión entre sockets origen-destino

Primitiva	Significado
SOCKET	Crea un nuevo punto terminal de comunicación.
BIND	Asocia una dirección local con un socket.
LISTEN	Anuncia la disposición de aceptar conexiones; indica el tamaño de la cola.
ACCEPT	Establece en forma pasiva una conexión entrante.
CONNECT	Intenta establecer activamente una conexión.
SEND	Envía datos a través de la conexión.
RECEIVE	Recibe datos de la conexión.
CLOSE	Libera la conexión.

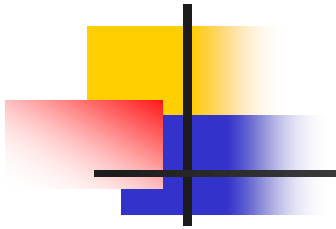
Figura 6-5. Las primitivas de socket para TCP.



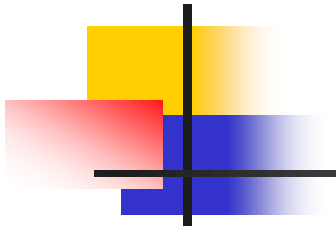
- Un socket puede usarse para varias conexiones (multiplexión)
- Dos o más conexiones pueden terminar en el mismo socket
- Esto se hace en servidores que atienden a múltiples clientes a la vez
- Las conexiones se identifican con los identificadores de los dos sockets
- Los número de puerto menores a 1024 se llaman ***puertos bien conocidos*** y se reservan para servicios estándar
- La lista de puertos bien conocidos se muestra a continuación

Puerto	Protocolo	Uso
20, 21	FTP	Transferencia de archivos.
22	SSH	Inicio de sesión remoto, reemplazo de Telnet.
25	SMTP	Correo electrónico.
80	HTTP	World Wide Web.
110	POP-3	Acceso remoto al correo electrónico.
143	IMAP	Acceso remoto al correo electrónico.
443	HTTPS	Acceso seguro a web (HTTP sobre SSL/TLS).
543	RTSP	Control del reproductor de medios.
631	IPP	Compartición de impresoras.

Figura 6-34. Algunos puertos asignados.



- El demonio de cada servicio se conecta a su respectivo puerto; por ejemplo, el servidor ftp se conecta al puerto 21
- Hacerlo así podría llenar la memoria con demonios inactivos
- Un super demonio de Internet (*inetd* en Unix) se conecta a múltiples puertos
- Se puede tener demonios permanentes en los puertos más ocupados (ej. 80) e *inetd* en los demás
- Conexiones TCP son full duplex
- TCP no soporta la multidifusión ni difusión, UDP sí



- Una conexión TCP es un flujo de bytes, no un flujo de mensajes
- TCP no reconoce mensajes de usuario
- El flujo de bytes es agrupado en diferentes segmentos TCP
- Si el usuario emite 4 mensajes de 512 bytes en un flujo TCP, estos mensajes se envían en cuatro, dos o un segmento de 512 bytes, 1024 bytes, o 2048 bytes
- La CT en el receptor no distingue si llegaron uno, dos o cuatro mensajes. TCP solo detectó que llegó un flujo de bytes a través de diferentes segmentos
- La CA se encarga de tomar el flujo de bytes y construir los mensajes

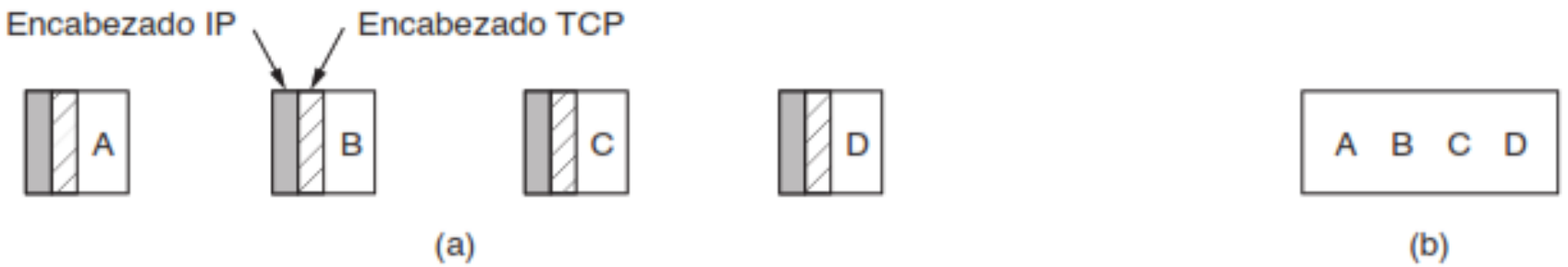
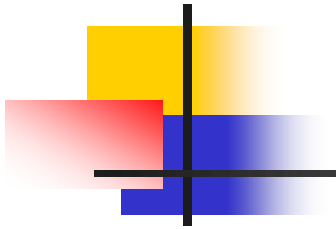


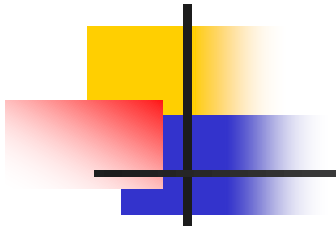
Figura 6-35. (a) Cuatro segmentos de 512 bytes que se envían como diagramas IP separados. (b) Los 2 048 bytes de datos que se entregan a la aplicación en una sola llamada READ.



- TCP envía los datos inmediatamente, o los almacena en un buffer para recolectar de la aplicación más datos y luego enviarlos
- Si la aplicación necesita que los datos se envíen inmediatamente activa el bit de la cabecera del segmento PUSH

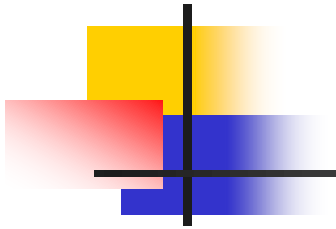
6.4.3 El protocolo TCP

- A cada byte de una **conexión** le corresponde un número de secuencia de 32 bits
- Antes, los enlaces entre enrutadores eran líneas telefónicas de $56Kbps$
- ¿En cuánto tiempo se agota la numeración?
- El número máximo es $2^{32} - 1 = 4.294'967.295 \approx 4.2\ GB$
- $56Kbps = 7.000\ bytes/s$
- Al número máximo $2^{32} - 1$ se llegaba en $\frac{(2^{32}-1)bytes}{7.000\ bytes/s} \approx 1\ semana$
- Los números de secuencia se usan en el mecanismo de ventana deslizante



- Un segmento TCP tiene una cabecera de 20 bytes (+ una parte opcional) seguido de 0 o más bytes de datos
- TCP decide el tamaño de los segmentos en función de dos restricciones:

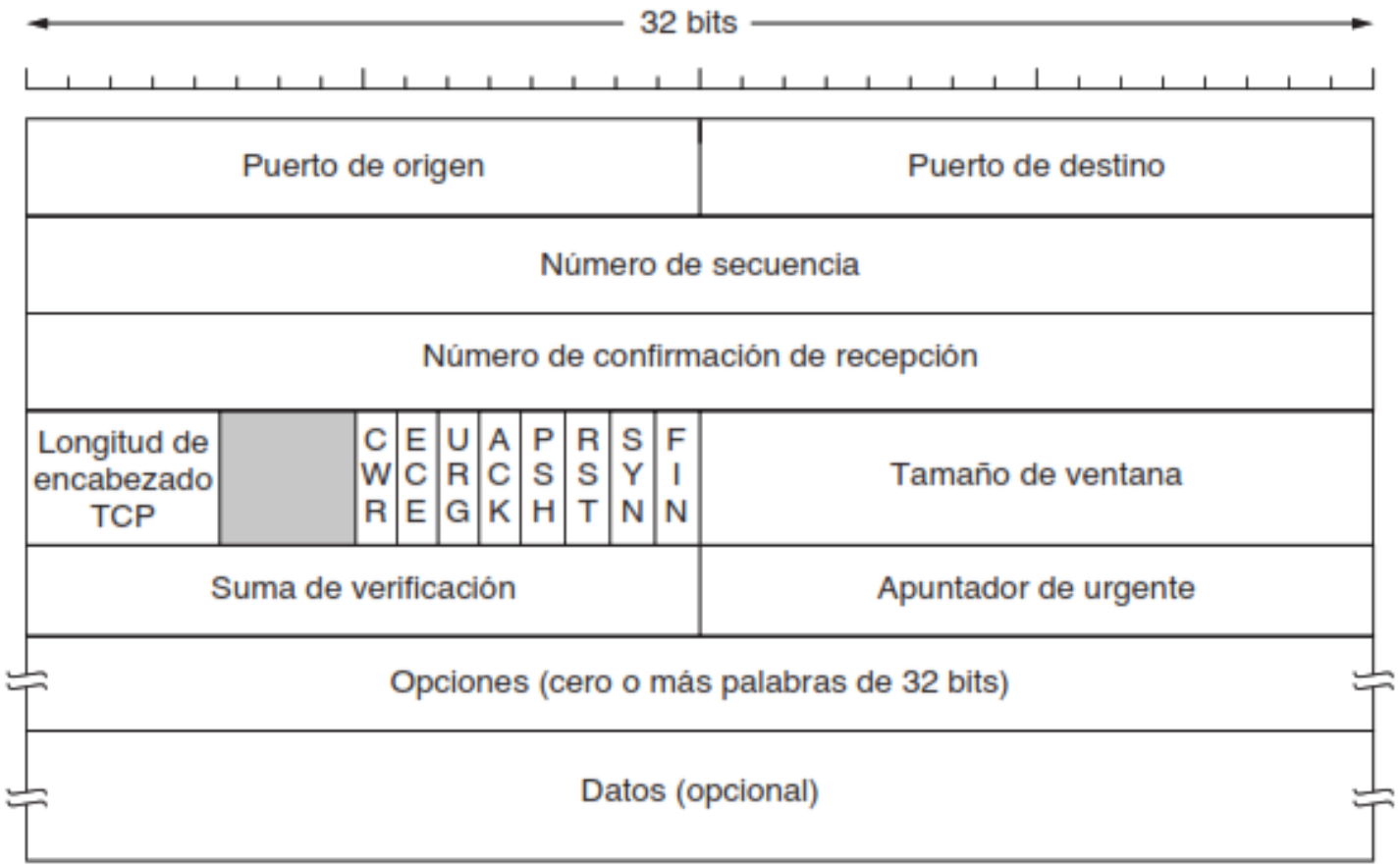
1. Que el segmento quepa en la carga útil de IP
2. Que quepa en la MTU de ruta (Path Maximum Transfer Unit) para evitar fragmentación. En la práctica MTU=1500 bytes (carga útil de Ethernet)

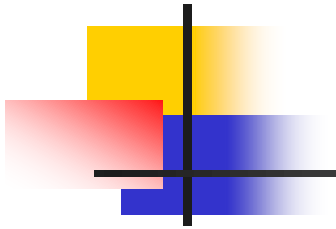


- TCP usa el protocolo de ventana deslizante:
  - Cuando el emisor transmite un segmento inicia un temporizador de retransmisión
  - Cuando el segmento llega al destino, TCP envía un ACK con o sin datos, indicando el siguiente número de secuencia a recibir (byte)
  - Si el temporizador expira, el emisor retransmite el segmento

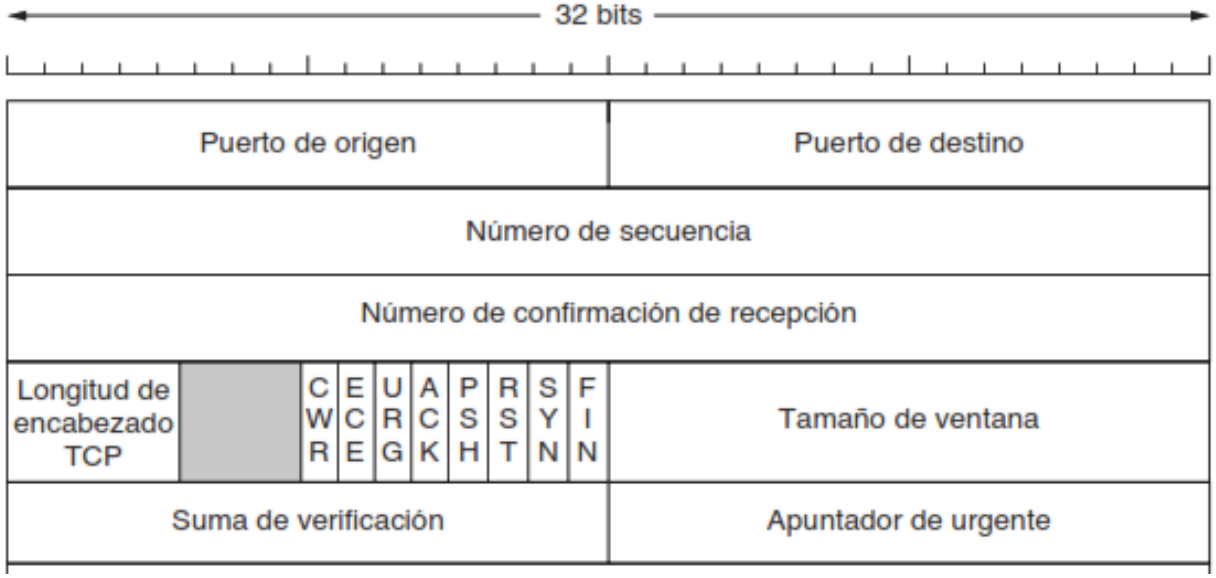
6.4.4 Encabezado del segmento TCP

20 bytes

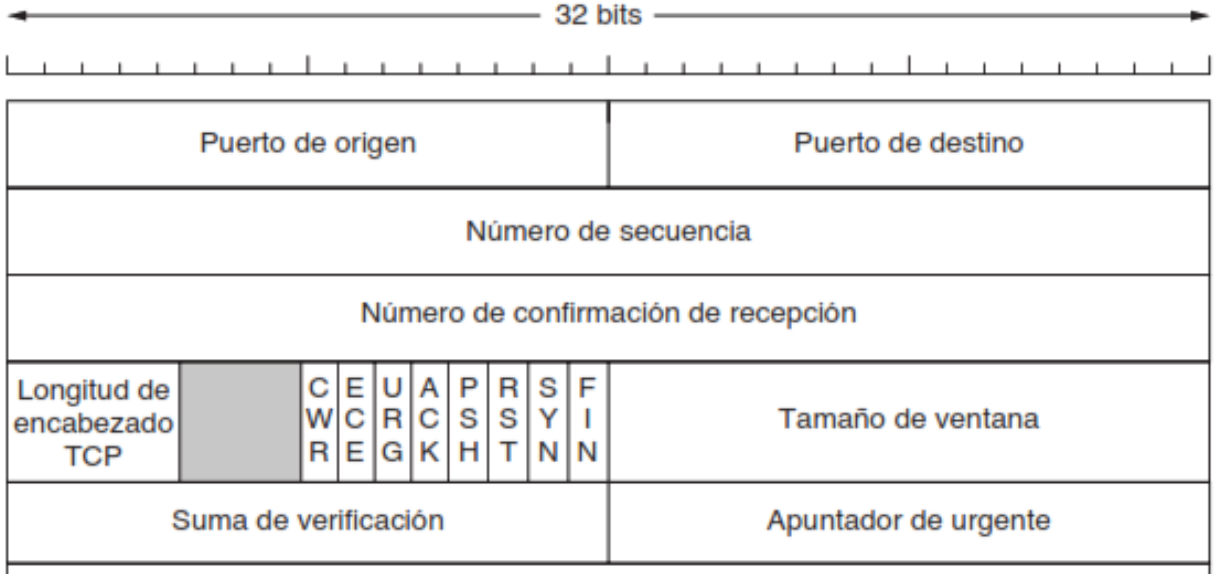




- Cada segmento comienza con un encabezado de formato fijo de 20 bytes
- El encabezado puede ir seguido de opciones
- Tras las opciones, si las hay, están los de datos de usuario
- Los segmentos sin datos se usan para ACK y mensajes de control

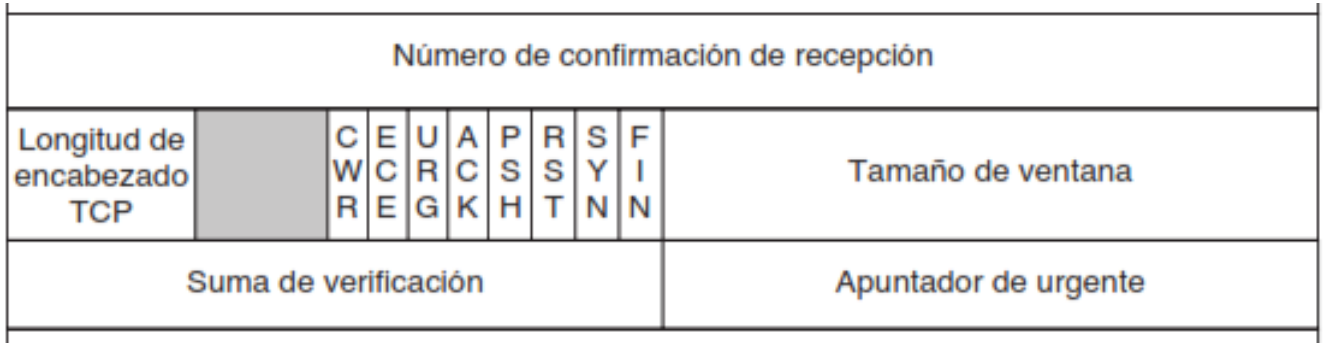


- *Puerto origen* y *puerto destino*. Son los puntos terminales locales de conexión
- Dirección IP + puerto forman un punto terminal único de 48 bits -socket local
- Los puntos terminales identifican la conexión
- El identificador de conexión se llama 5-tupla. Tiene 5 piezas de información: protocolo IP origen, puerto origen, IP destino, protocolo TCP, puerto destino
- *Número de secuencia*. Indica el primer byte del segmento

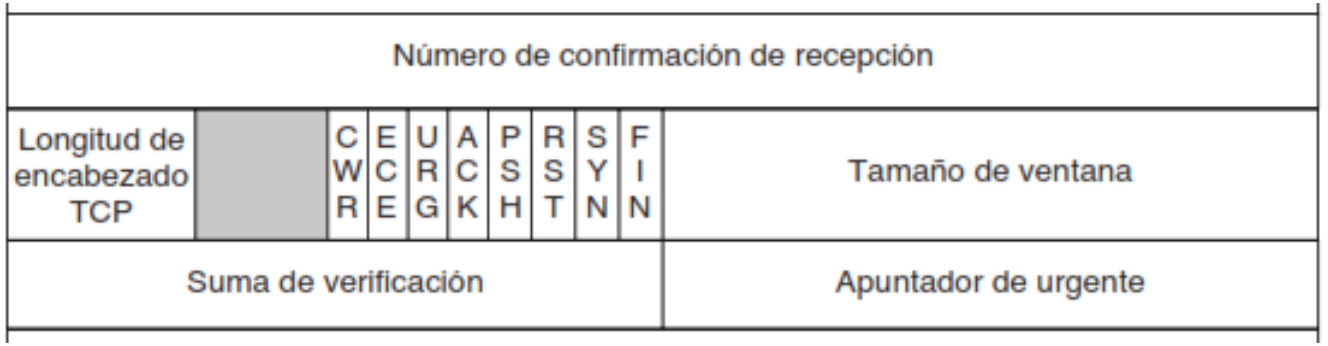


- *Número de confirmación de recepción* (ACK). Indica el número del primer byte del siguiente segmento esperado, no es el número del siguiente segmento
- *Longitud del encabezado*. Con 4 bits expresa en palabras de 32 bits. Máximo $15 \times \frac{32}{8} = 60\ bytes$
- Luego hay 4 bits que no se utilizan

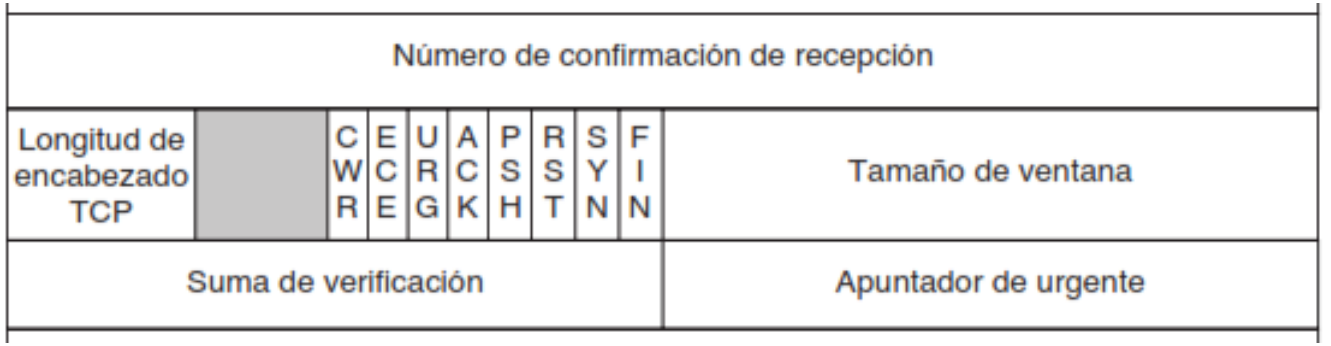
Ocho Indicadores de 1 bit



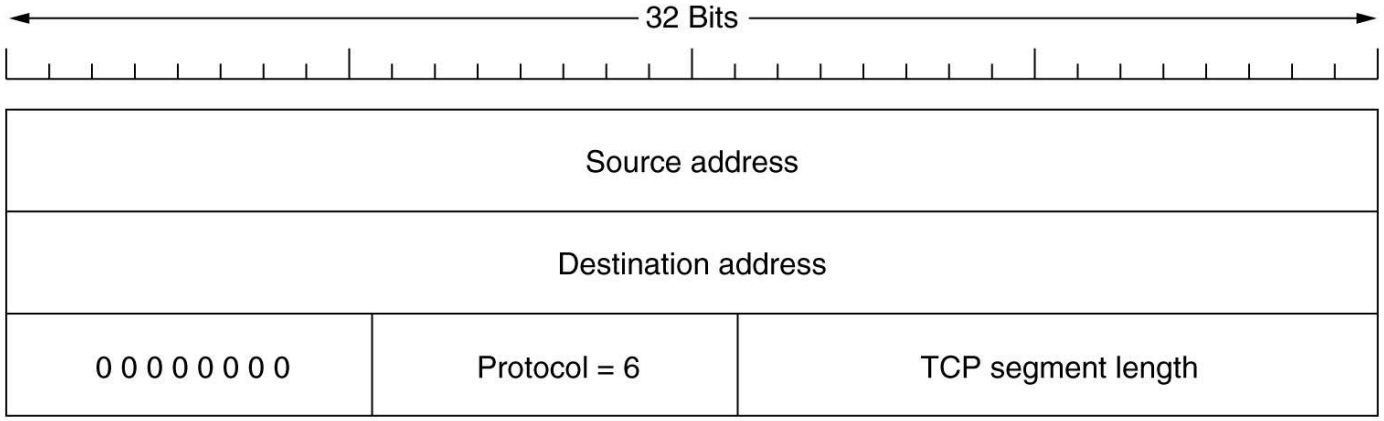
- CWR Congestion Window Reduced. La red avisa al receptor que hay congestión
- ECE. Explicit Congestion notification-Echo. Al mismo tiempo la red pide al emisor que baje la velocidad
- URG. Si el bit está en 1 indica que está activo el *Apuntador de urgente*
- ACK. Indica que el *Número de confirmación de recepción* es válido
- PSH. Push. La aplicación solicita a TCP que envíe el mensaje inmediatamente al destino y en el receptor que entregue inmediatamente a la aplicación



- RST Reset. Restablece la conexión debido a la caída del host
- SYN. Se usa para establecer una conexión
  - CONNECTION REQUEST: SYN = 1, ACK = 0
  - CONNECTION ACCEPTED: SYN = 1, ACK = 1
- FIN Finish. Se usa para liberar una conexión. Un emisor no tiene más datos que enviar pero puede seguir recibiendo datos
- Tamaño de ventana. (Capítulo 3. Canalización) El control de flujo se maneja usando una ventana de tamaño variable. A menor tamaño menor flujo
- El tamaño máximo de ventana es $2^{16} = 64\,KB$ antes de recibir el primer ACK
- Si es 0, el receptor necesita momentáneamente un descanso



- Suma de verificación. Considera el encabezado TCP, los datos y un pseudo encabezado que incluyen las direcciones IP
- La inclusión de direcciones IP viola la jerarquía de protocolos ya que esta corresponden a la capa de Internet
- UDP también incluye el pseudo encabezado en el checksum
- *Apuntador de urgente.* Trabaja con el bit *URG*. Indica el desplazamiento en bytes en el que están los datos urgentes a partir del *Número de secuencia* del byte actual

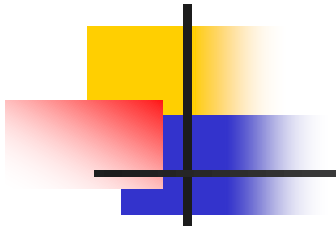


Pseudo encabezado

Campo de opciones

- El host especifica la carga útil TCP máxima que está dispuesto a aceptar
- Esta opción se indica al momento de establecer la conexión
- Segmentos grandes son más eficientes porque hay menos encabezados para más datos
- La carga útil predeterminada es 536 bytes
- Se requiere que todos los hosts de Internet acepten segmentos TCP de 536 + 20 bytes de cabecera

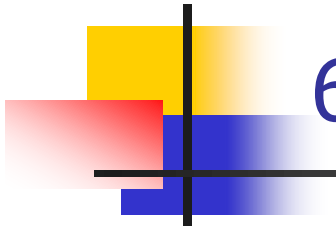
- En líneas con gran ancho de banda y/o alto retardo de propagación, la ventana de 16 bits es muy poco
- Si se tiene una línea de FO transoceánica de 600 Mbps, el tiempo de transmisión de una ventana de 16 bits $(2^{16}\ bytes\ =\ 64KB)$ es
- $\frac{64\ KB}{600\ Mbps/8} = 8 \times 10^{-4}s < 1ms$
- Si la distancia de una lugar a otro es unos 15.000 Km el retardo de propagación es de unos 50 ms
- El emisor estará ocioso $\left(1 - \frac{1ms}{50ms}\right) \times 100 = 98\%$ del tiempo
- Un tamaño de ventana mayor a 16 bits permitirá al emisor enviar más datos
- La opción *escala de ventana* permite al emisor y receptor negociar al momento del establecimiento de la conexión, el aumento de tamaño de la ventana hasta 14 bits más, para transmisiones de hasta 2^{30} bytes



- *Confirmación de recepción selectiva*. Permite al receptor indicar al emisor los rangos de números de secuencia que ha recibido
- Se utiliza después de haber perdido un paquete
- Así, el emisor sabe qué segmentos se deben retransmitir en forma selectiva

6.5.5 Establecimiento de una conexión TCP

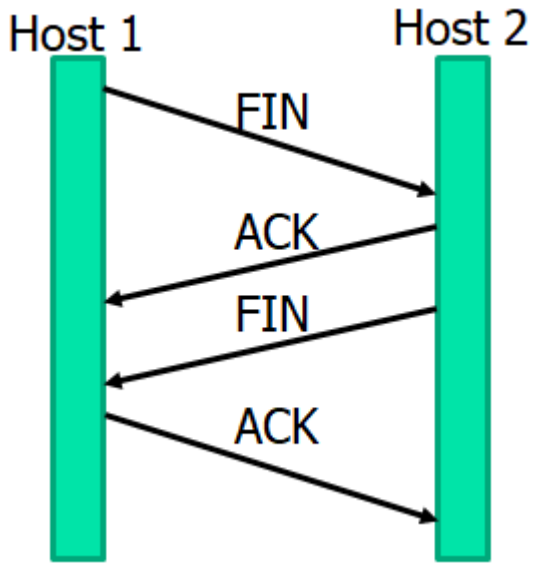
- Se usa un acuerdo de 3 vías
- El cliente solicita una conexión (*SYN=1, ACK=0*) indicando dirección IP y puerto destinos, y el tamaño máximo del segmento TCP a aceptar
- El cliente espera una respuesta del servidor
- El servidor espera una conexión entrante indicando el origen
- El servidor revisa si está esperando una conexión en el puerto destino especificado
- Si hay, acepta la conexión (*SYN=1, ACK=1*). Si no lo hay, el servidor envía una respuesta al cliente rechazando la conexión (*RST=1*)

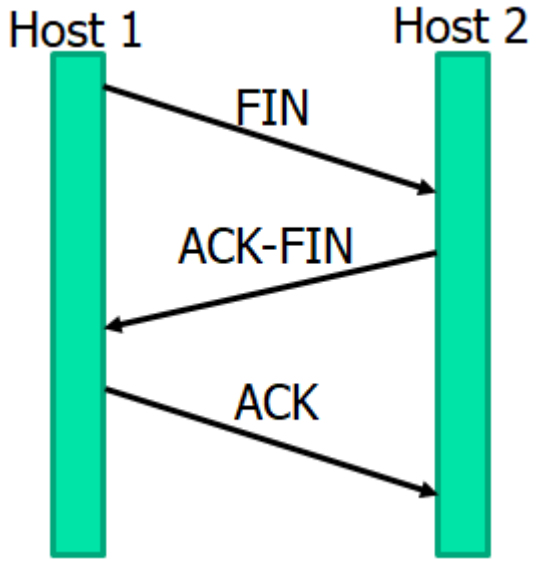


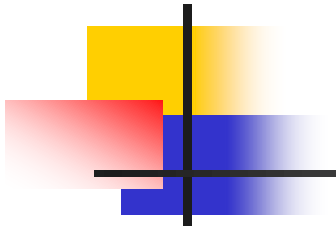
6.5.6 Liberación de una conexión TCP

- Las conexiones TCP son full dúplex, formadas con dos conexiones simplex
- Cada conexión simplex se libera independientemente
- Para liberar una conexión, una de las partes puede enviar un segmento TCP indicando que no tiene más datos por transmitir
- Al recibirse este segmento, ese sentido de la conexión se apaga
- Pero puede continuar un flujo indefinido de datos en el otro sentido
- Cuando ambos sentidos se apagan se libera la conexión

- Normalmente se necesitan 4 segmentos TCP para liberar una conexión: un FIN y un ACK para cada sentido
- Pero, es posible que el primer ACK y el segundo FIN estén contenidos en el mismo segmento, reduciéndose la cuenta total a 3.

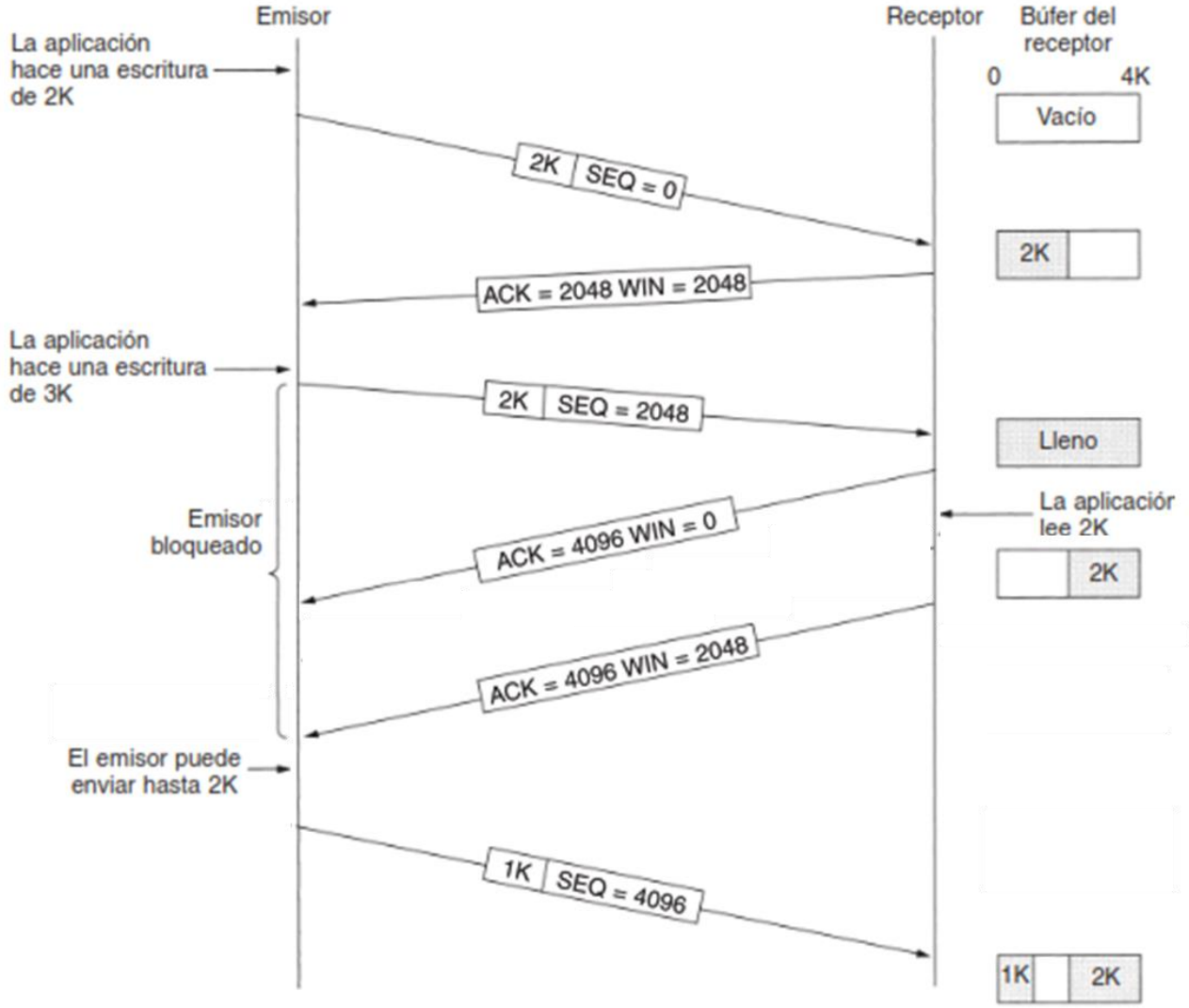


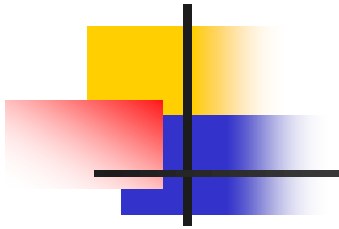




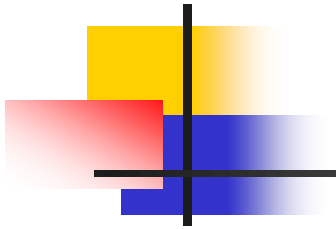
- Para evitar el problema de los 2 ejércitos, se usan temporizadores
- Si no llega una respuesta a un FIN en un máximo de dos tiempo de la vida de un paquete, el emisor de FIN libera la conexión
- Tarde o temprano el otro lado notará que nadie le está escuchando, también liberará la conexión
- Ésta no es una solución perfecta pero pocas veces ocurren problemas

6.5.8 Política de transmisión de TCP

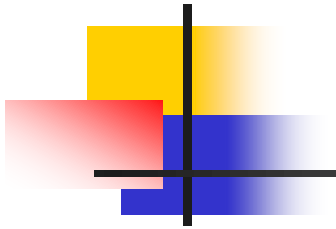




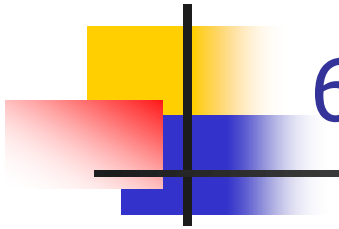
- Suponga que el receptor tiene un buffer de 4096 bytes (4K)
- La aplicación en el emisor hace una escritura de 2048 bytes (2K)
- El emisor envía un segmento de 2K, desde el byte 0 al byte 2047
- El receptor los recibe correctamente
- El receptor envía la confirmación de recepción del segmento
- La aplicación en el receptor aun no retira datos del buffer
- El receptor tiene solo 2K libres
- El receptor anuncia una ventana de 2K y el siguiente byte esperado
- La aplicación en el emisor hace una escritura de 3072 bytes (3K)



- EL emisor envía un segmento de 2K desde el byte 2048 al byte 4095
- El buffer en el receptor se llena
- El receptor avisa recibo de los datos y que el siguiente segmento a recibir es desde el byte 4096, pero al momento el buffer está lleno
- Luego la aplicación en el receptor lee del buffer 2K de datos
- El receptor avisa al emisor que el siguiente segmento a recibir es desde el byte 4096, y el espacio libre en buffer es de 2K
- El emisor envía el 1K restante iniciando desde el byte 4096



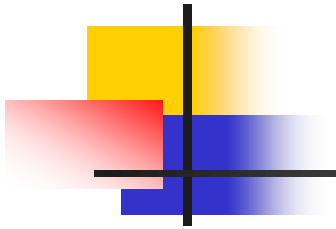
- No se requiere que el emisor envíe datos tan pronto como llegue de la aplicación
- Al inicio de la transmisión, el emisor podía esperar otros 2KB para transmitir un segmento con una carga útil de 4 KB
- Tampoco que se requiere que los receptores envíen confirmaciones de recepción tan pronto como sea posible
- El receptor podía esperar procesar todos los segmentos del buffer, y luego anunciar que tiene una ventana de 4 KB, y no sólo de 2KB



6.5.9 Control de congestión en TCP

- Cuando la carga inyectada a la red es mayor que la que puede manejar, se genera una congestión
- La capa IP intenta controlarla con los protocolos de enrutamiento
- Pero el trabajo pesado recae sobre TCP
- La solución a la congestión es disminuir la tasa de datos: No inyectar un paquete nuevo a la red hasta que salga uno viejo
- TCP lo soluciona manipulando dinámicamente los tamaños de las ventanas

- El primer paso es detectar la congestión
- La congestión se puede detectar observando las expiraciones de temporizador
- Expiración de temporizador implica la retransmisión de paquetes
- La retransmisión implica aumento de carga de la red
- La expiración de un temporizador por un paquete perdido, puede deberse a:
  - Ruido en la línea
  - Desbordamiento en un enrutador congestionado
- No hay pérdida de paquetes por errores de transmisión en líneas de fibra óptica. En las redes inalámbricas es diferente
- La mayoría de esas expiraciones en Internet se deben a congestión



- TCP toma acciones preventivas y acciones correctivas frente a la congestión
- Evitación o prevención de congestiones
  - Al establecer una conexión, emisor y receptor acuerdan un tamaño de ventana tal que no haya derrame en el receptor
- Pero aun puede ocurrir congestión interna en la red
  - El temporizador caduca y el emisor baja la velocidad

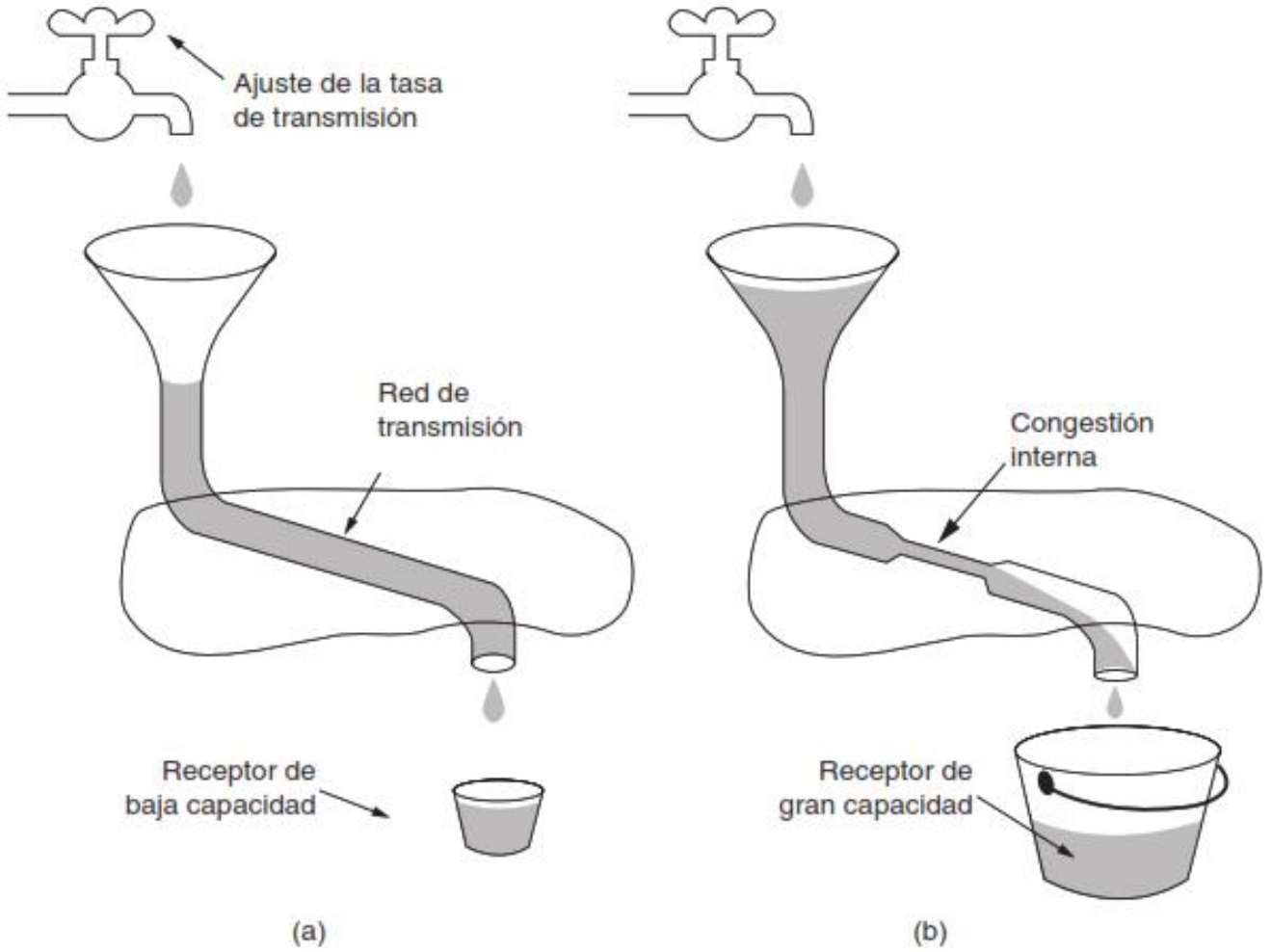
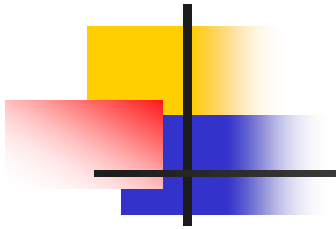
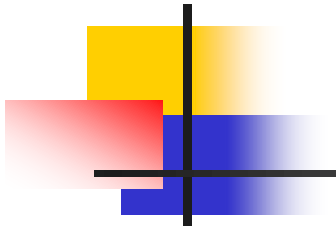


Figura 6-22. (a) Una red veloz que alimenta a un receptor de baja capacidad. (b) Una red lenta que alimenta a un receptor de alta capacidad.



- En Internet hay dos problemas potenciales
  - Capacidad de la red y
  - Capacidad del receptor
- La estrategia es manejarlos por separado
- Para ello, cada emisor mantiene dos ventanas:
  - La ventana de recepción que ha otorgado el receptor, y
  - La ventana de congestión
- Cada una indica la cantidad de bytes que puede enviar el emisor
- La cantidad que puede enviar el emisor es la menor de las dos ventanas



- Hay dos algoritmos que permiten determinar la cantidad de bytes que puede enviar el emisor sin producir congestión ni en el receptor ni en la red
  - Arranque lento
  - Algoritmo de Internet

Algoritmo de arranque lento

Tamaño de la ventana de congestión del emisor

Segmento confirmado

Tamaño de la ventana luego de la

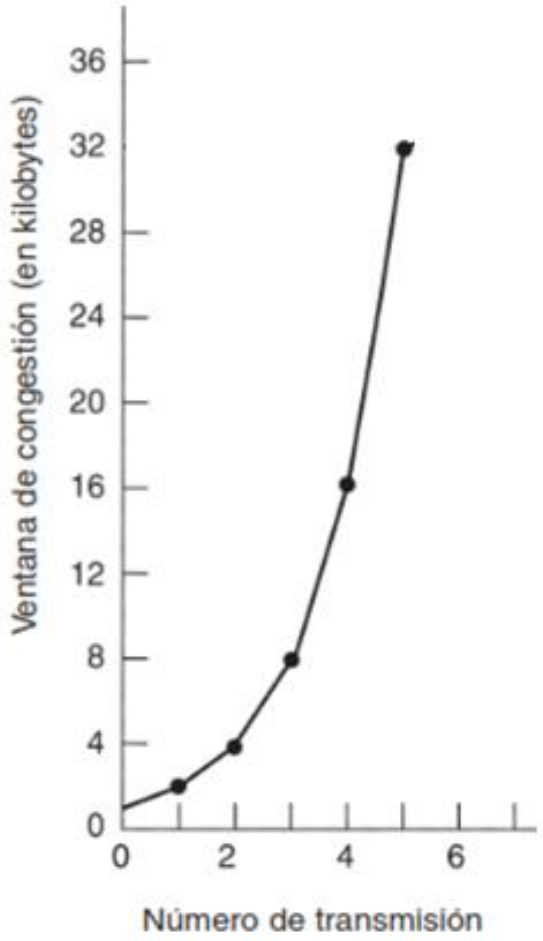
1ra. transmisión de la ventana

2da. transmisión de la ventana

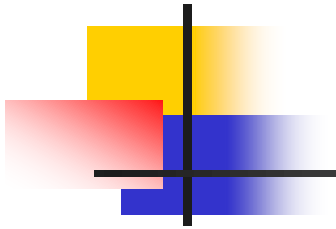
3ra. transmisión de la ventana

Algoritmo de arranque lento

- Al establecer una conexión, el emisor asigna a la ventana de congestión el tamaño de segmento máximo, entonces envía un segmento máximo
- Si se recibe la confirmación de recepción de este segmento antes de que expire el temporizador, el emisor aumenta el tamaño de la ventana en un segmento máximo y envía dos segmentos de tamaño máximo
- A medida que se confirma cada uno de estos segmentos, se aumenta el tamaño de la ventana de congestión en un segmento máximo
- Cada ráfaga confirmada duplica el tamaño de la ventana de congestionamiento



Ejemplo del algoritmo de Arranque Lento



- La ventana de congestión sigue creciendo exponencialmente hasta:
  - Ocurrir una expiración del temporizador, o
  - Alcanzar el tamaño de la ventana receptora

Algoritmo de Internet

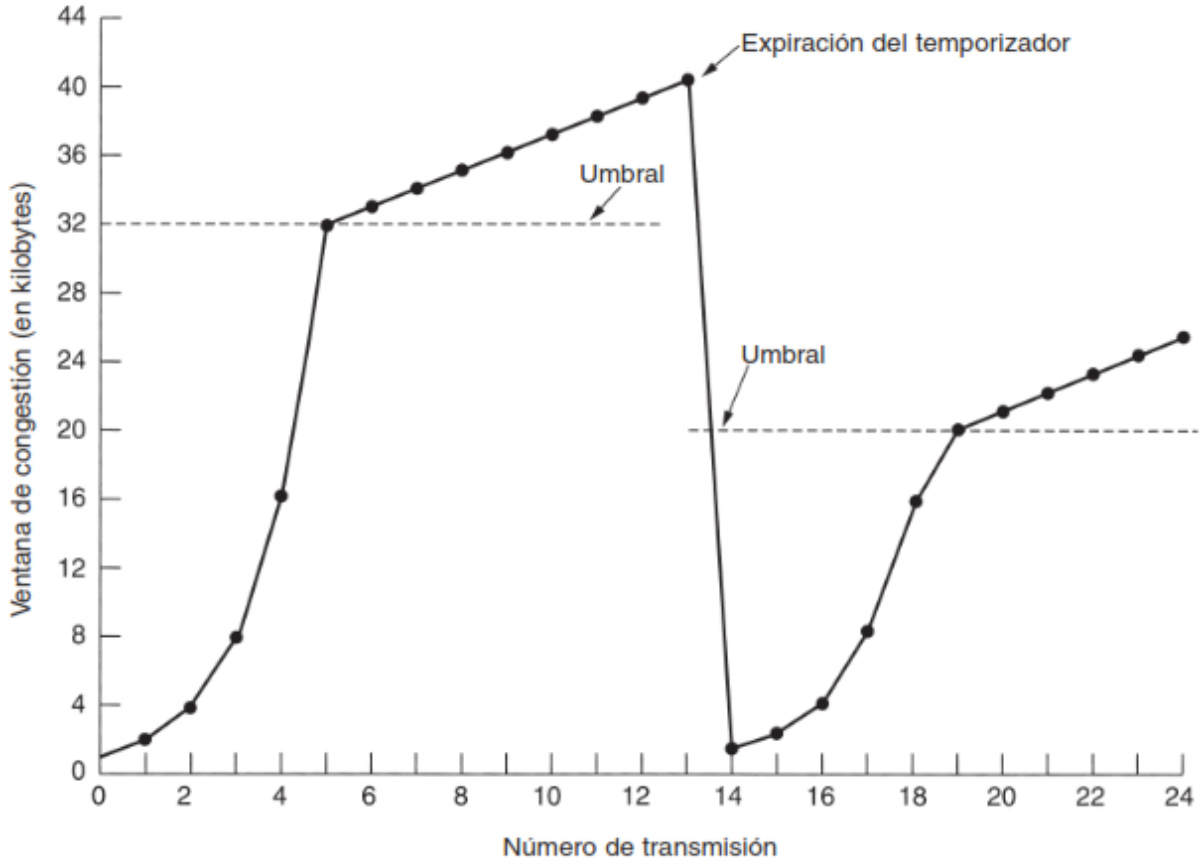


Figura 6-37. Ejemplo del algoritmo de congestión de Internet.

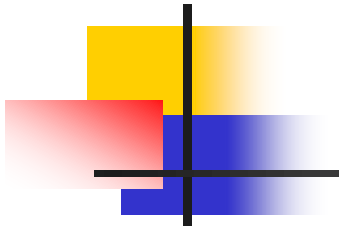
Algoritmo de Internet

Tamaño de la ventana de congestión del emisor

Segmento confirmado

Algoritmo de Internet

- Se usa el parámetro umbral, inicialmente de 64 KB, además de la ventana del receptor y de la ventana de congestión
- Al expirar el temporizador, se establece el umbral en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo
- Luego se usa el *algoritmo de arranque lento*
- Pero, el crecimiento exponencial termina al alcanzar el umbral
- Luego, las transmisiones exitosas aumentan linealmente la ventana de congestión; o sea, aumenta en un segmento por ráfaga, no en un segmento por segmento

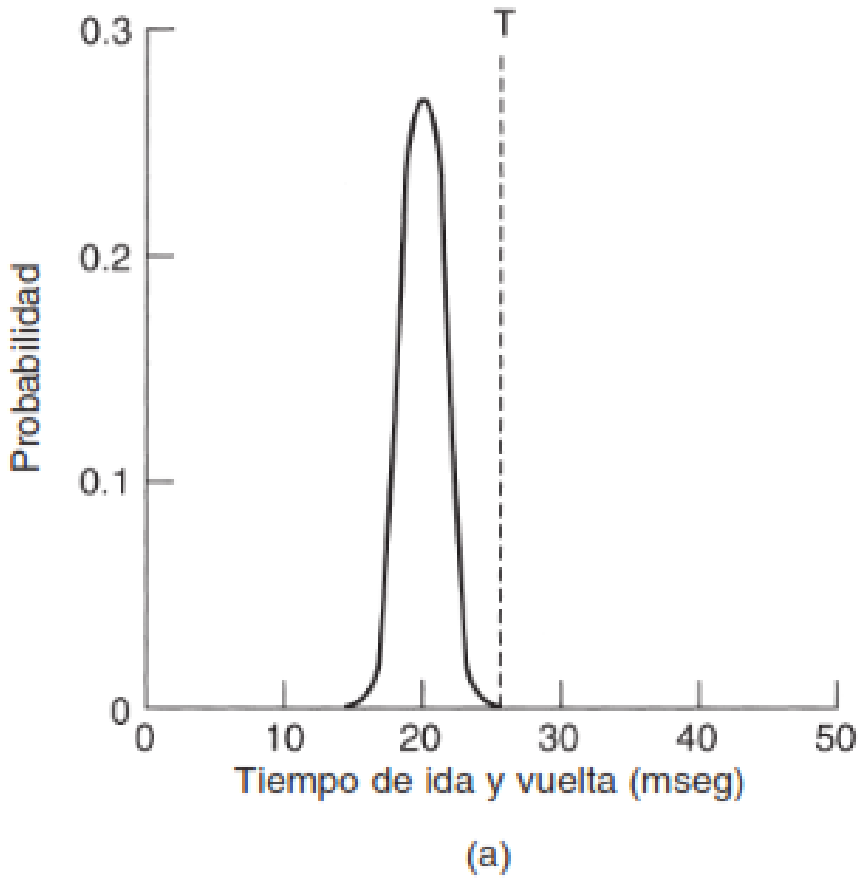


- Si no ocurren más expiraciones del temporizador, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor
- En este punto, dejará de crecer y permanecerá constante mientras no ocurran más expiraciones del temporizador

6.4.10 Administración de temporizadores del TCP

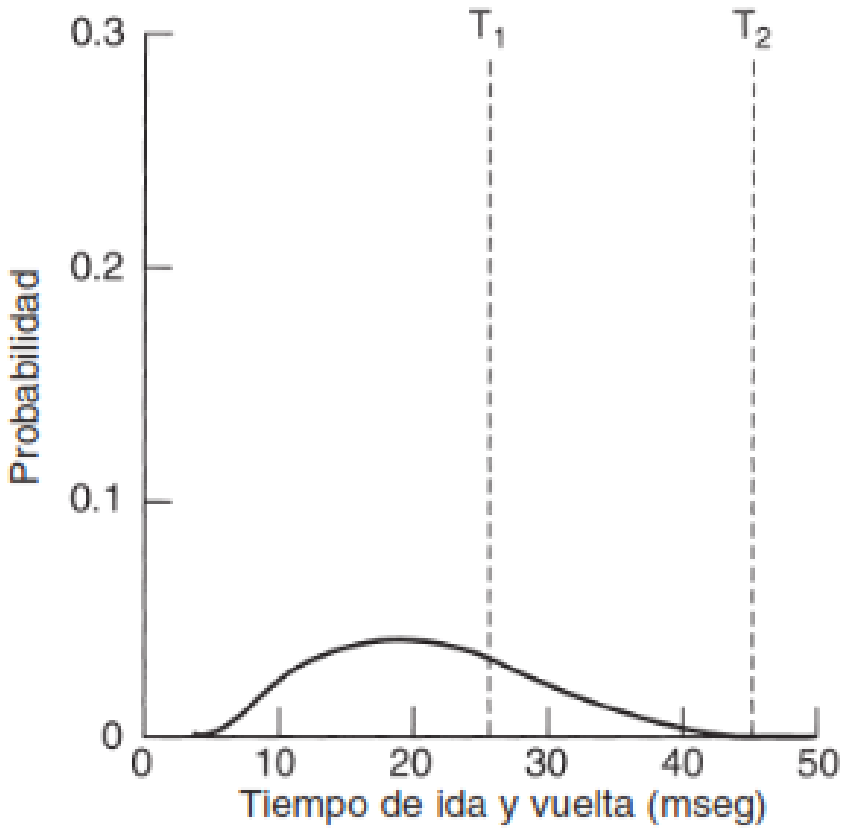
- Al enviarse un segmento se inicia el temporizador de retransmisiones
- Si la confirmación de recepción del segmento llega antes de expirar el temporizador, este se detiene
- Si el temporizador termina antes de llegar la confirmación de recepción, se retransmite el segmento y se reinicia el temporizador
- ¿Qué tanto debe durar el temporizador?
- Esta duración es mucho más predecible en el capa 2 que en la capa 4. Fig. 6.38(a)

Probabilidad del tiempo de llegada de la confirmación en la capa 2 vs el temporizador T

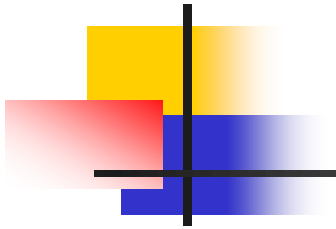


(a)

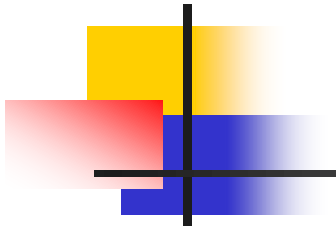
Probabilidad del tiempo de llegada de la confirmación en la capa 4 vs el el temporizador T1 o T2



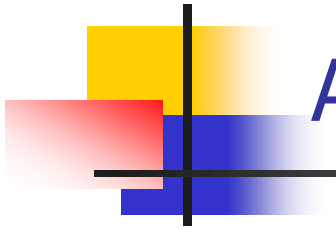
(b)



- En la capa 2 las confirmaciones de recepción pocas veces se retrasan
- La ausencia de confirmación de recepción generalmente se debe a la pérdida de esta o de la trama
- En TCP, es difícil la determinación del tiempo de ida y vuelta
- Si el intervalo de expiración es corto (T1) ocurren retransmisiones innecesarias, sobrecargando la red innecesariamente
- Si es largo (T2), el desempeño disminuye debido al mayor retardo de retransmisión de cada paquete perdido
- Por otro lado el tiempo de ida y vuelta puede variar rápidamente



- La solución es usar un algoritmo que ajuste dinámicamente el intervalo de expiración del temporizador
- Esto se logra con base en mediciones continuas del desempeño de la red
- El algoritmo que usa TCP es de Jacobson (1988)



Algoritmo de Jacobson

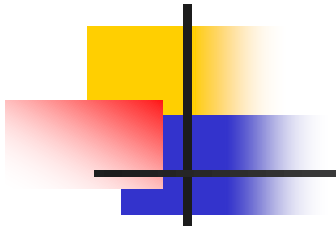
- TCP utiliza cuatro temporizadores diferentes:
- De retransmisión
- De persistencia
- De seguir con vida
- De espera cronometrada

Temporizador de retransmisión

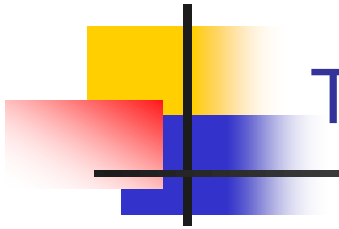
- Para cada conexión, TCP mantiene una variable RTT (Round-Trip Time), que es la mejor estimación actual del tiempo de ida y vuelta al origen
- Al enviar un segmento se inicia un temporizador para ver el tiempo de confirmación de recepción y para habilitar la retransmisión si se tarda demasiado
- Si llega la confirmación de recepción en un tiempo M, antes de expirar el temporizador, TCP actualiza RTT:
- $RTT = \alpha RTT + (1 - \alpha)M$
- α es el peso que se da al RTT anterior
- Por lo general $\alpha = 7/8$. Así, se da más peso a RTT anterior que a M
- Algunas implementaciones de TCP usa el temporizador de retransmisión $= 2RTT$

Temporizador de persistencia

- Este es diseñado para evitar el bloqueo
- El receptor envía una confirmación de recepción, indicando al emisor que espere (tamaño de ventana 0). Esta indicación llega al emisor
- Después el receptor actualiza la ventana, pero se pierde el paquete con la actualización
- Ahora el emisor y receptor esperan que el otro haga algo
- Cuando termina el temporizador de persistencia el emisor envía un sondeo al receptor



- La respuesta al sondeo da el tamaño de la ventana
- Si la ventana sigue en 0, se inicia otra vez el temporizador de persistencia
- Si la ventana es diferente de cero el emisor puede enviar datos



Temporizador de seguir con vida

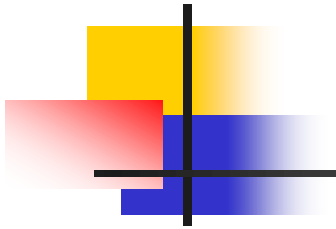
- Si una conexión está inactiva por mucho tiempo, este temporizador expira
- El un lado comprueba que el otro lado aun está allí
- Si no recibe respuesta, se termina la conexión

Temporizador de espera cronometrada

- Este opera durante el cierre de una conexión
- Tiene una duración del doble del tiempo máximo de vida del paquete
- Así se asegura que todos los paquetes desaparezcan antes del cierre

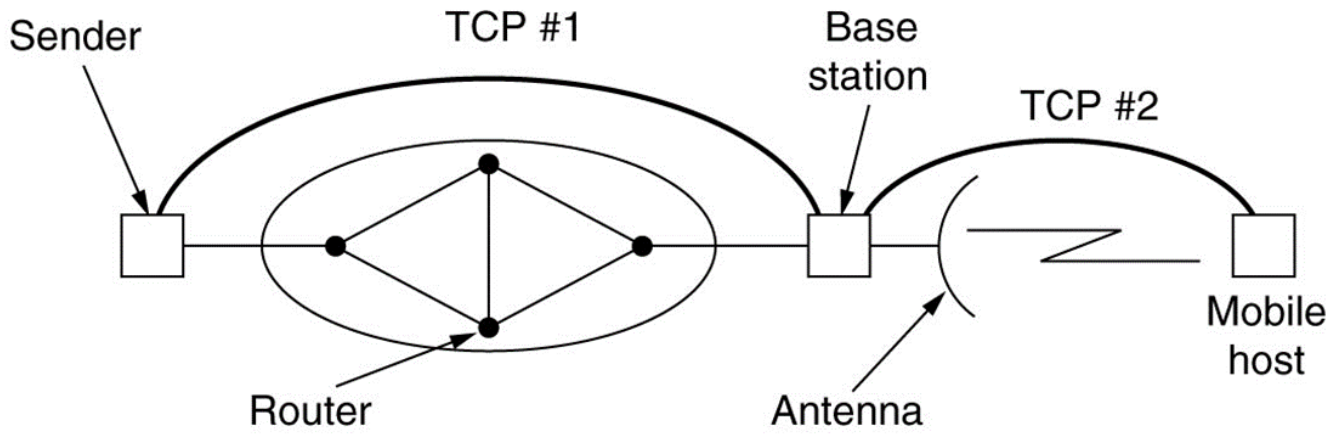
6.4.11 TCP y UDP inalámbricos

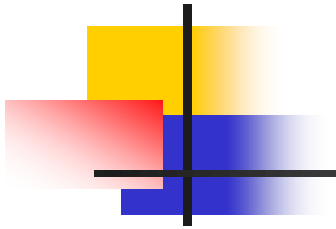
- Los protocolos de transporte deberían ser independientes de la tecnología de la capa de red subyacente
- TCP no debería preocuparse si IP opera sobre cobre, fibra o radio
- En la práctica sí importa, ya que las implementaciones TCP son optimizadas para redes cableadas, no para redes inalámbricas
- El problema principal es el algoritmo de control de congestión
- TCP supone que las expiraciones del temporizador ocurren por congestionamiento, no por paquetes perdidos
- Por tanto, TCP disminuye su velocidad (arranque lento) para reducir la carga a la red y aliviar la congestión



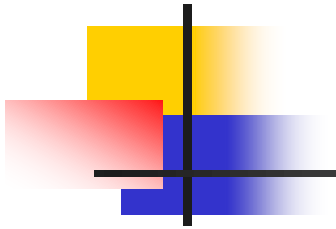
- Pero los enlaces inalámbricos pierden muchos paquetes, no por congestionamiento
- El enfoque adecuado para el manejo de paquetes perdidos es reenviarlos inmediatamente
- Al perderse un paquete por congestionamiento en una red cableada, el emisor debe reducir la velocidad
- En redes inalámbricas la reducción de la velocidad empeora las cosas, el emisor debe al menos mantener la velocidad, no reducirla
- Si el emisor no sabe si la red es cableada o inalámbrica, no se puede tomar la decisión correcta

- Con frecuencia, el camino desde el emisor al receptor no es homogéneo
- **TCP indirecto** es la solución propuesta
- La solución es dividir la conexión TCP en conexiones distintas





- La ventaja de este esquema es que ahora ambas conexiones son homogéneas
- Las expiraciones del temporizador en la red 1 reducen la velocidad del emisor
- Las expiraciones del temporizador en la red 2 aceleran la velocidad
- La desventaja: viola la semántica de TCP: conexión extremo a extremo



- La siguiente solución no quebranta la semántica de TCP
- Funciona haciendo modificaciones pequeñas en la estación base
- Un cambio es incluir un agente que almacena en caché los segmentos TCP que van al host móvil, y las confirmaciones de recepción que regresan de él
- Si el agente no ve la llegada de una confirmación de recepción antes de que el temporizador expire, retransmite este segmento sin indicar el origen de retransmisión