Tema 3: El protocolo TCP

Capítulos:

- Introducción
- Puertos y conexiones
- Control de flujo. Recuperación de errores
- Control de congestión
- Formato de un segmento TCP
- Establecimiento y cierre de una conexión
- Conclusiones

Bibliografía
"Internetworking with TCP/IP", *Cap. 12.*

Introducción

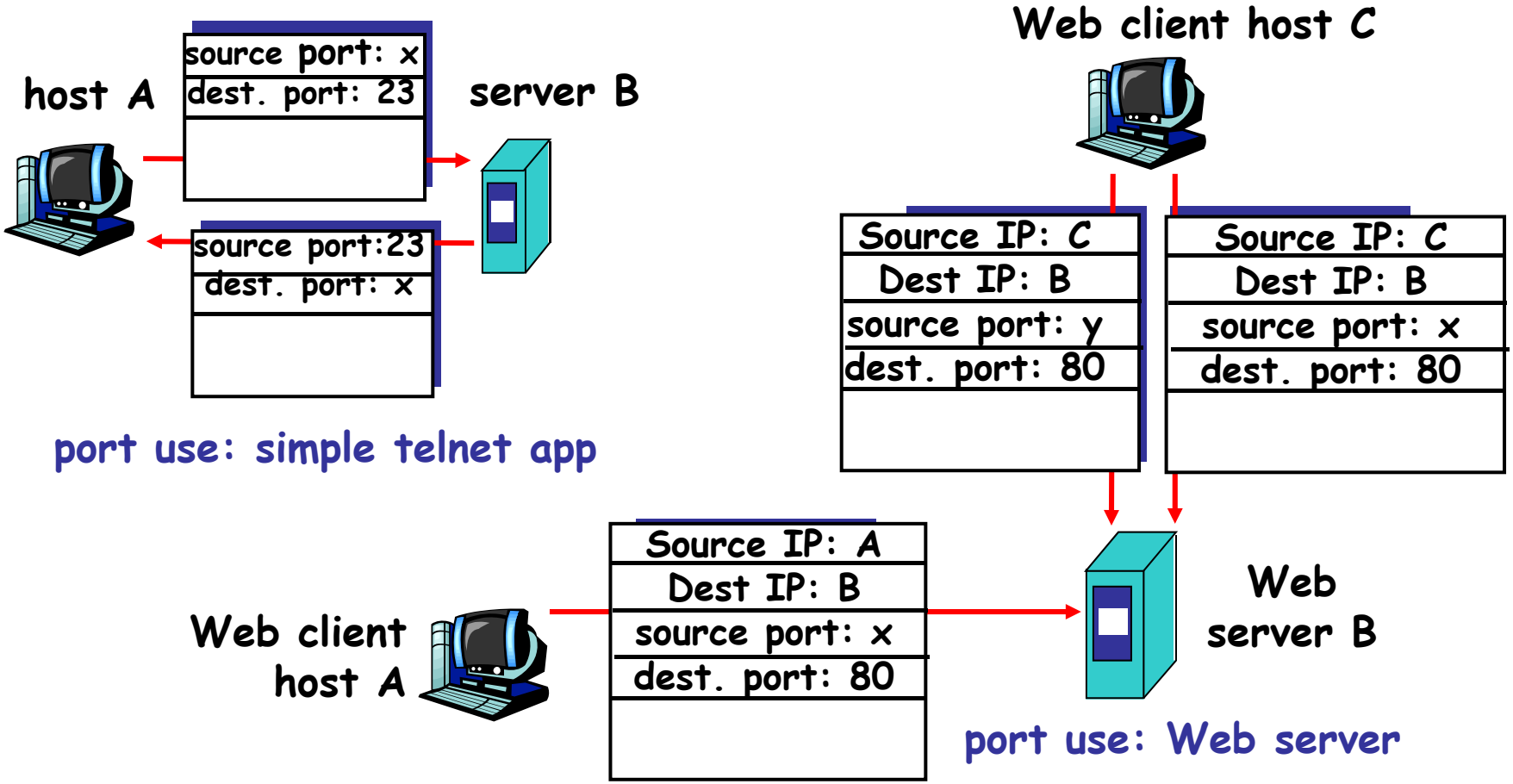
- Ninguno de los protocolos vistos hasta ahora proporciona **fiabilidad** en la comunicación de datos
- Si las aplicaciones la necesitan, tienen dos opciones:
  - Añadir los mecanismos necesarios en la propia aplicación
  - Utilizar un protocolo intermedio que se encargue de esta tarea
- La arquitectura TCP/IP ofrece un protocolo de transporte que proporciona la fiabilidad deseada:
  - Protocolo TCP (Transmission Control Protocol)
- Al igual que UDP, utilizará el **mecanismo de puertos**.

Características de TCP

- Proporciona un **servicio orientado a la conexión**, **fiable** y **ordenado** (comunicación punto-a-punto)
  - Tres fases (similar al servicio telefónico) :
    - Establecimiento de conexión
    - Transferencia de datos
    - Cierre de conexión.
- El flujo de datos es tratado como una **secuencia de bytes** (byte stream) → NO distingue fronteras entre mensajes
- El software del protocolo decide cómo dividir (o agrupar) las unidades de datos que la aplicación le transfiere.
- Por motivos de flexibilidad, el protocolo **no especifica la interfaz con la aplicación** (sockets interface)

Puertos y conexiones

- Asignación de puertos **estática** ó **dinámica**
- Se puede utilizar el mismo puerto para TCP y UDP

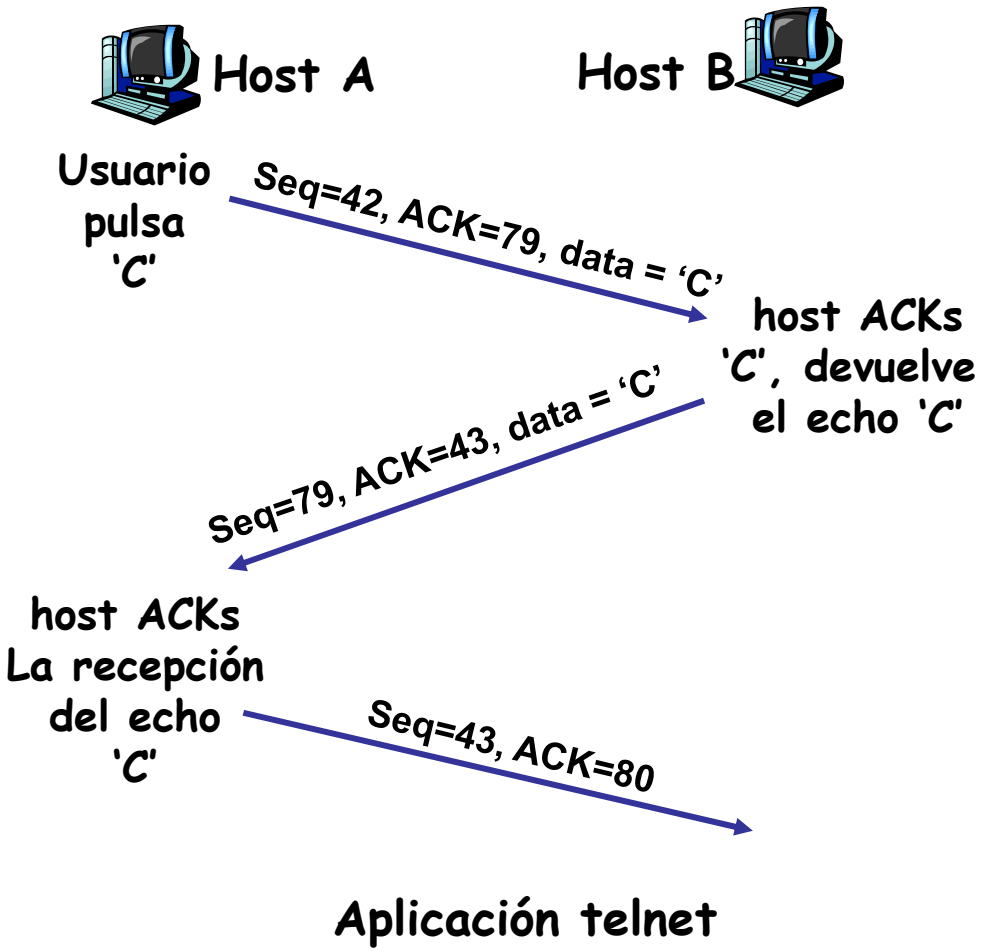


Control de flujo. Recuperación de errores

- Unidad de datos TCP: **segmento**
- Control de flujo extremo a extremo mediante **ventana deslizante** (protocolos punto-a-punto pipeline)
  - La Parada y Espera puede resultar muy poco eficiente
- Tamaño máximo de la ventana de transmisión variable (puede ser cero)
- Reconocimientos a nivel de octetos y no de segmentos
- Conexión TCP **full-duplex** → Dos flujos de datos independientes (4 ventanas)
- Se permite el empleo de **datos urgentes**, que no están sujetos al control de flujo

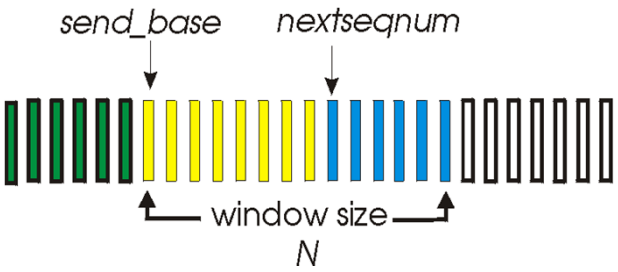
TCP: Números de secuencia y reconocimientos

- **Números de secuencia**:
  - Número de secuencia del primer octeto en el campo de datos del segmento
- **Reconocimientos**:
  - Número de secuencia del siguiente byte que se espera recibir
  - ACKs acumulativos
- ¿Cómo se procesan los **segmentos fuera de orden**?
  - TCP no especifica lo que hay que hacer → decisión de implementación.



TCP: Algoritmo simplificado

- **Versión simplificada del emisor TCP**
  - NO se considera control de flujo ni control de congestión.
  - Transferencia en un único sentido



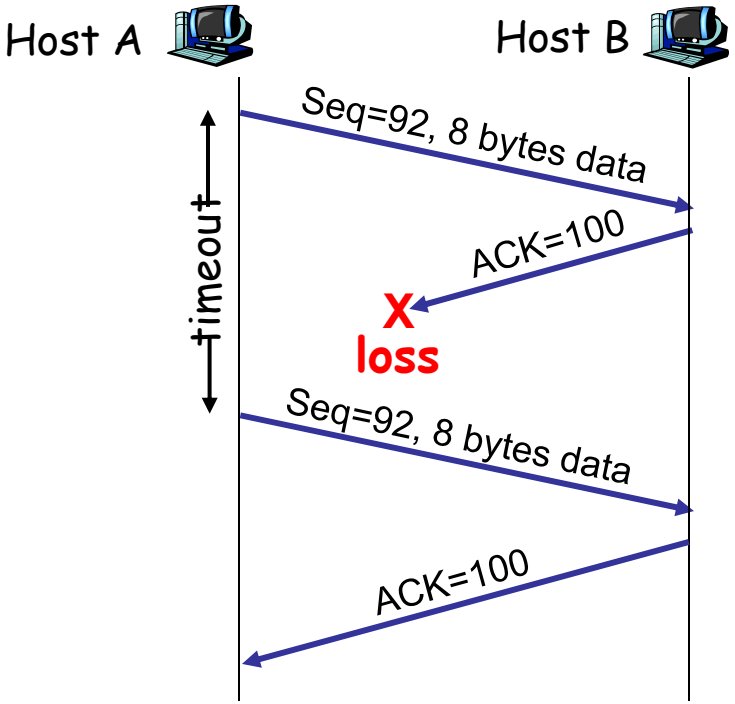

```
00  sendbase = initial_sequence number
01  nextseqnum = initial_sequence number
02
03   loop (forever) {
04     switch(event)
05     event: data received from application above
06          create TCP segment with sequence number nextseqnum
07          start timer for segment nextseqnum
08          pass segment to IP
09          nextseqnum = nextseqnum + length(data)
10      event: timer timeout for segment with sequence number y
11          retransmit segment with sequence number y
12          compute new timeout interval for segment y
13          restart timer for sequence number y
14      event: ACK received, with ACK field value of y
15          if (y > sendbase) { /* cumulative ACK of all data up to y */
16              cancel all timers for segments with sequence numbers < y
17              sendbase = y
18              }
19          else { /* a duplicate ACK for already ACKed segment */
20              increment number of duplicate ACKs received for y
21              if (number of duplicate ACKS received for y == 3) {
22                  /* TCP fast retransmit */
23                  resend segment with sequence number y
24                  restart timer for segment y
25              }
26    }  /* end of loop forever */
```

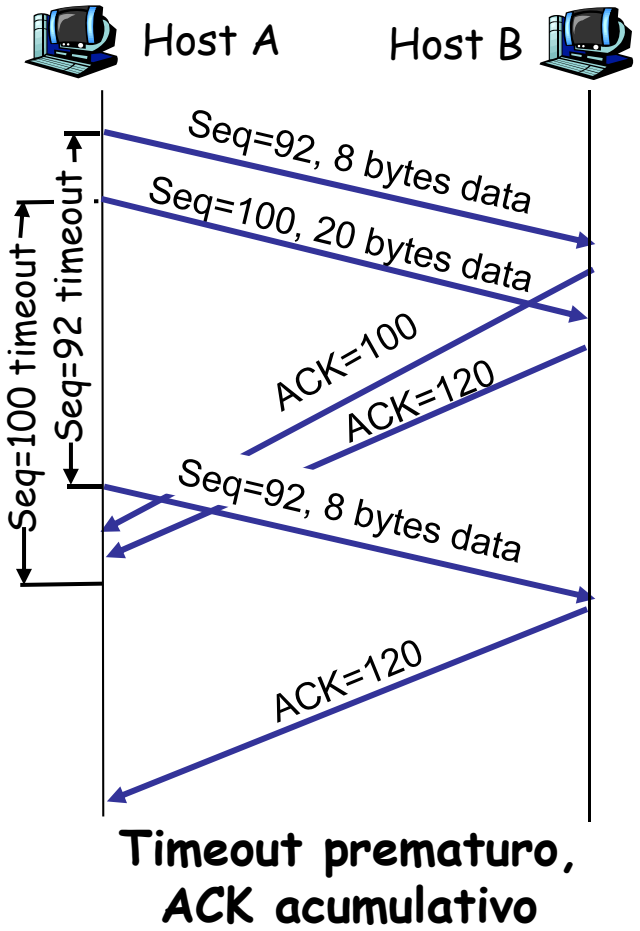
TCP: Generación de segmentos ACK

[RFC 1122, RFC 2581]

Evento	Acción en el receptor TCP
Llegada en orden de un segmento, **no hay huecos** y **todo** está previamente **reconocido**	**ACK retrasado** → Espera 500ms al siguiente segmento. Si no se genera, envía ACK
Llegada en orden de un segmento, **no hay huecos** y existe un **ACK retrasado pendiente**	Enviar inmediatamente un **ACK acumulativo**.
Llegada **fuera de orden** de segmento, Seq # mayor que el esperado **Hueco detectado**	Envío de un **ACK duplicado**, indicando el número de secuencia esperado
Llegada de un **segmento** que Parcial o completamente **llena el hueco**	Envío inmediato de un **ACK**, si el segmento comienza en la parte baja del hueco

TCP: Retransmisiones





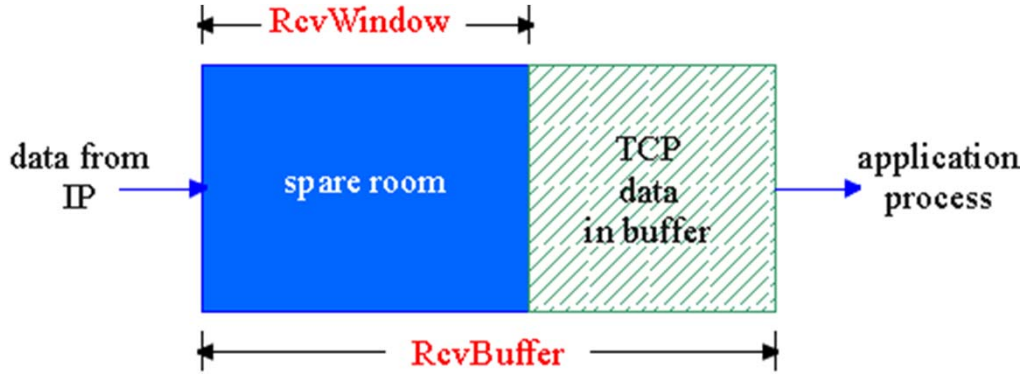
- **Las retransmisiones pueden originar duplicados**
  - Detección: Números de secuencia de los segmentos
- **La duración del temporizador es crítica**
  - Temporizador independiente para cada segmento

TCP: Control de flujo

Control de flujo

- **El emisor no enviará más datos de los que el receptor pueda procesar en cada momento.**

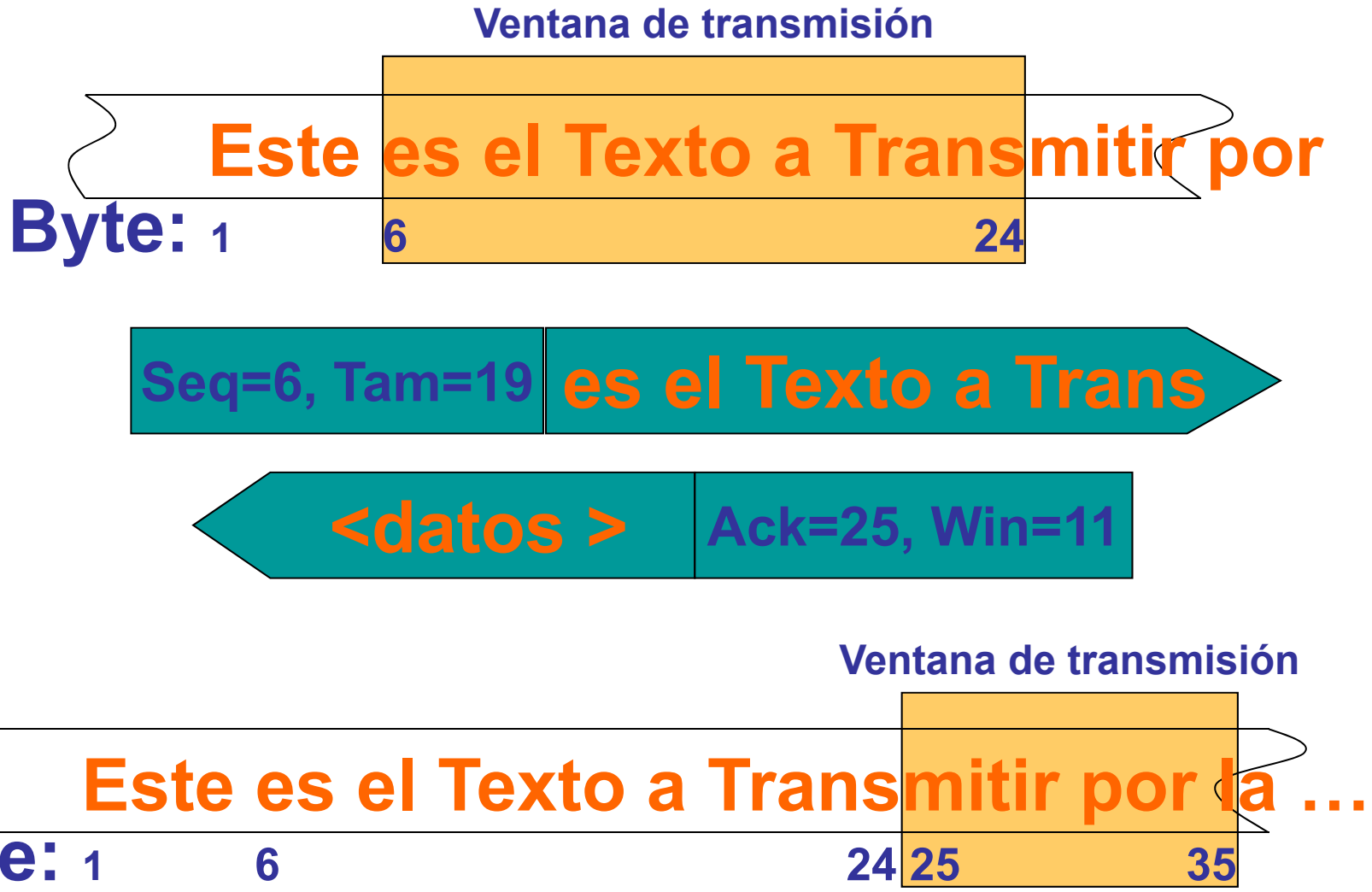
`RcvBuffer` = **Tamaño del buffer TCP del receptor**
`RcvWindow` = **Cantidad de espacio libre en el buffer**



Buffer del receptor

- **Receptor**: Informa explícitamente al emisor de la cantidad de espacio libre que dispone en el buffer
  - **Campo *RcvWindow*** (ventana RX) de la cabecera TCP
- **Emisor**: Mantiene la cantidad de datos enviados pendientes de reconocimiento por debajo de la indicación más reciente de *RcvWindow*

TCP: Un ejemplo de Control de flujo.



TCP: Temporizadores y retransmisión

- **¿Qué valor elegimos para el temporizador de un segmento TCP?**
  - Mayor que el RTT
    - El RTT (Round Trip Time) no es estable a lo largo de una conexión TCP
  - Si se elige un Timeout demasiado corto → Timeouts prematuros
  - Si se elige un Timeout demasiado largo → Lenta reacción a los segmentos perdidos → Disminución del throughput

- **¿Cómo se estima el valor del RTT?**
  - *RTT_medido*: Tiempo transcurrido desde el envío de un segmento hasta que se recibe su ACK
    - No se tienen en cuenta retransmisiones ni ACKs acumulados
  - Ya que *RTT_medido* variará a lo largo de la conexión, se propone obtener una estimación promediada del RTT
    - Basada en el historial reciente de medidas del RTT.

TCP: Temporizadores y retransmisión (I)

- **Estimación del RTT**
  - Media móvil ponderada
  - Valor recomendado para α=0.1

$$RTT\_estimado = (1-\alpha)*RTT\_estimado + \alpha*RTT\_medido$$

- **Cálculo del valor del Timeout**
  - RTT_estimado más un “margen de seguridad”
  - Cuanto más grande sea la variación del RTT_estimado mayor margen de seguridad deberíamos establecer.

$$Timeout = RTT\_estimado + 4*Deviation$$

$$Deviation = (1-\beta)*Deviation + \beta*|RTT\_medido-RTT\_estimado|$$

TCP: Temporizadores y retransmisión (II)

- Cuando se retransmite un segmento una o más veces, al recibir su reconocimiento, resulta imposible determinar a cuál de los segmentos enviados corresponde

- Esto origina problemas a la hora de establecer el intervalo de timeout
  - Si el reconocimiento se asocia al primer segmento enviado → el temporizador tenderá a crecer mucho si la red continúa perdiendo datagramas
  - Si el reconocimiento se asocia con el último segmento enviado → tendremos problemas de timeouts prematuros (timeout demasiado pequeño).

TCP: Temporizadores y retransmisión (III)

- Solución → **Algoritmo de *Karn***
  - Diferenciar en la estimación del valor del temporizador los segmentos transmitidos una única vez de los procedentes de una retransmisión.
- Cuando se produce la **pérdida** de un segmento se ignora su RTT, y se emplea una estrategia **back-off**
  - Nuevo_timeout = g * timeout, (generalmente g = 2)
- Cuando se transmite un segmento con éxito en el primer intento → se toma su RTT como estimación válida y deja de aplicarse el back-off.

TCP: Control de congestión (I)

- La congestión aparece por una sobrecarga en los nodos de conmutación (routers)
  - Pérdida de paquetes (desbordamiento de los buffers en los routers)
  - Incremento de los retardos de los paquetes (almacenamiento en colas)
- Lo que provoca un incremento en las retransmisiones, que no hacen más que agravar la situación.
- Control de congestión ≠ Control de flujo.
- TCP puede ayudar a solucionar la congestión, reduciendo la inyección de segmentos en la red.
- El estándar actual recomienda dos técnicas, relacionadas y fáciles de implementar:
  - **Slow-start** y…
  - **Congestion avoidance**.

TCP: Control de congestión (II)

- Se define una **ventana de congestión** que regulará el caudal de salida de TCP.
  - Esta ventana, afecta al tamaño real de la ventana de transmisión

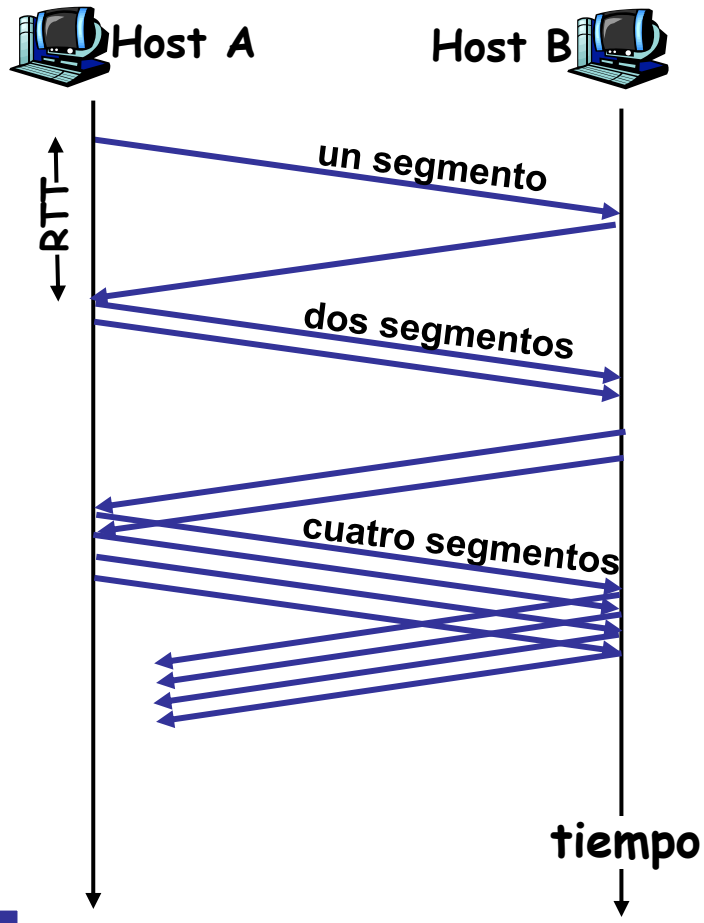
    ven_txon_permitida = min (ventana_recepción, ventana_congestión)
- Durante el funcionamiento normal (libre de congestión)
  - **Ventana_congestión = Ventana_recepción** (determinada por el control de flujo).
- El indicio de congestión se detecta por la pérdida de paquetes (vencimiento timeout):
  - En ese caso, se utilizará la ventana de congestión para reducir la ventana de transmisión
  - Además, para aquellos segmentos que permanecen en la ventana permitida, se aumenta el intervalo de timeout exponencialmente (back-off) → Algoritmo de Karn.

TCP: Control de congestión → **Slowstart**

Algoritmo Slowstart

```
// CW (Congestion Window)
CW = 1
Repeat
    if (ACK received) CW++
Until (evento de pérdida OR
       CW > threshold)
Go to Congestion avoidance
```

- Incremento exponencial (por RTT) en el tamaño de la ventana (no es lento!)
- Evento de pérdida:
  - Timeout (Tahoe TCP) y/o tres ACKs duplicados (Reno TCP)

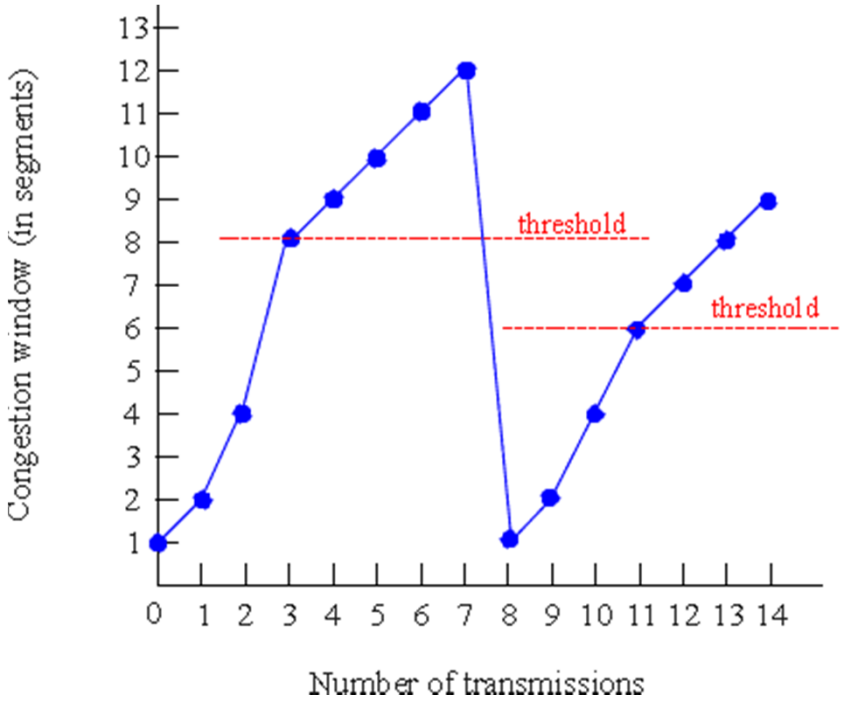


Inicialmente, "threshold" igual a la mitad de la ventana de recepción.

TCP: Control de congestión → **Congestion avoidance**

Congestion avoidance

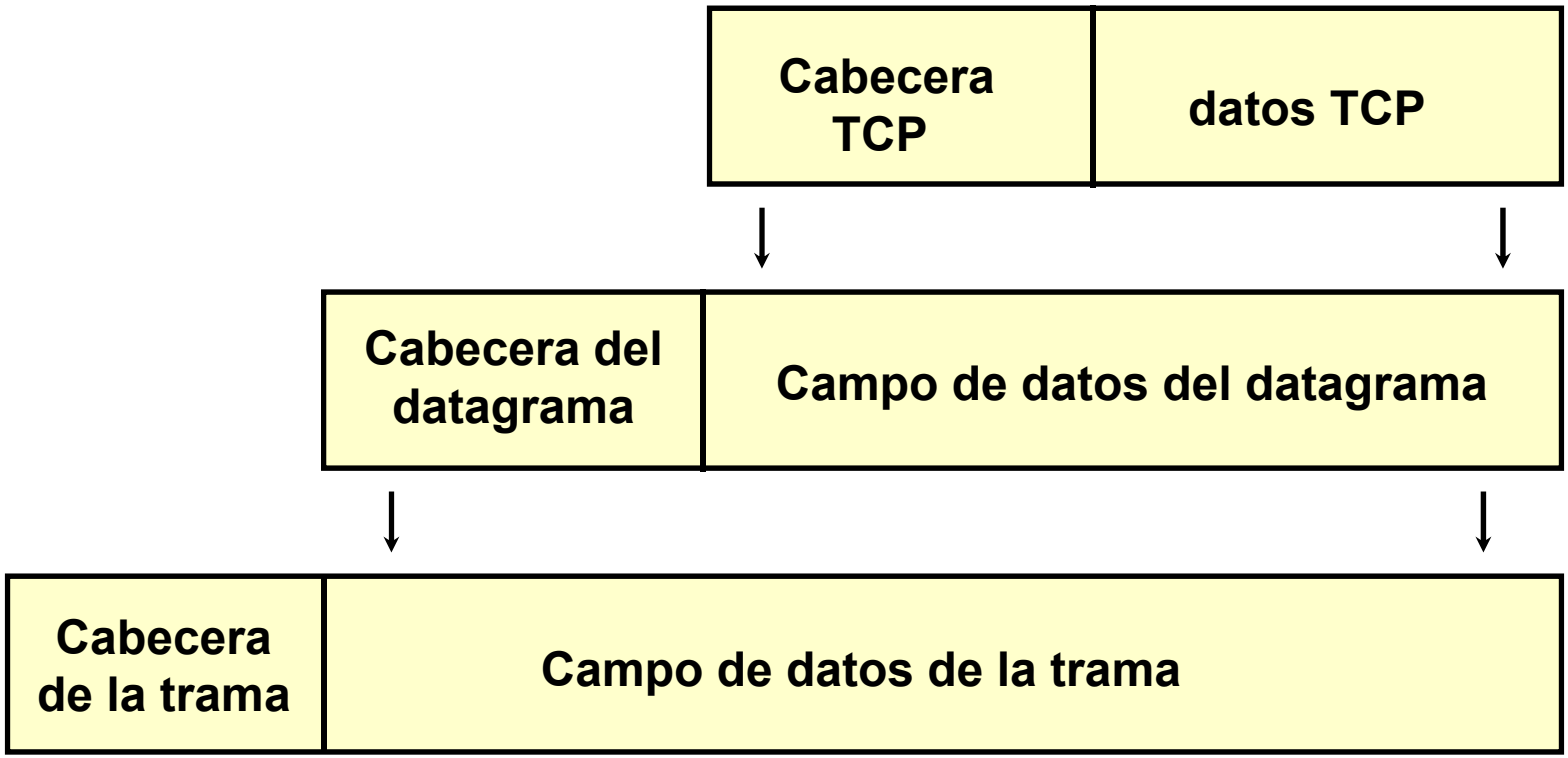
```
/* slow start terminado */
/* CW > threshold */
Hasta (evento de pérdida) {
  Cada CW segmentos ACKed:
     CW++
}
threshold = CW/2
Go to slow start
```



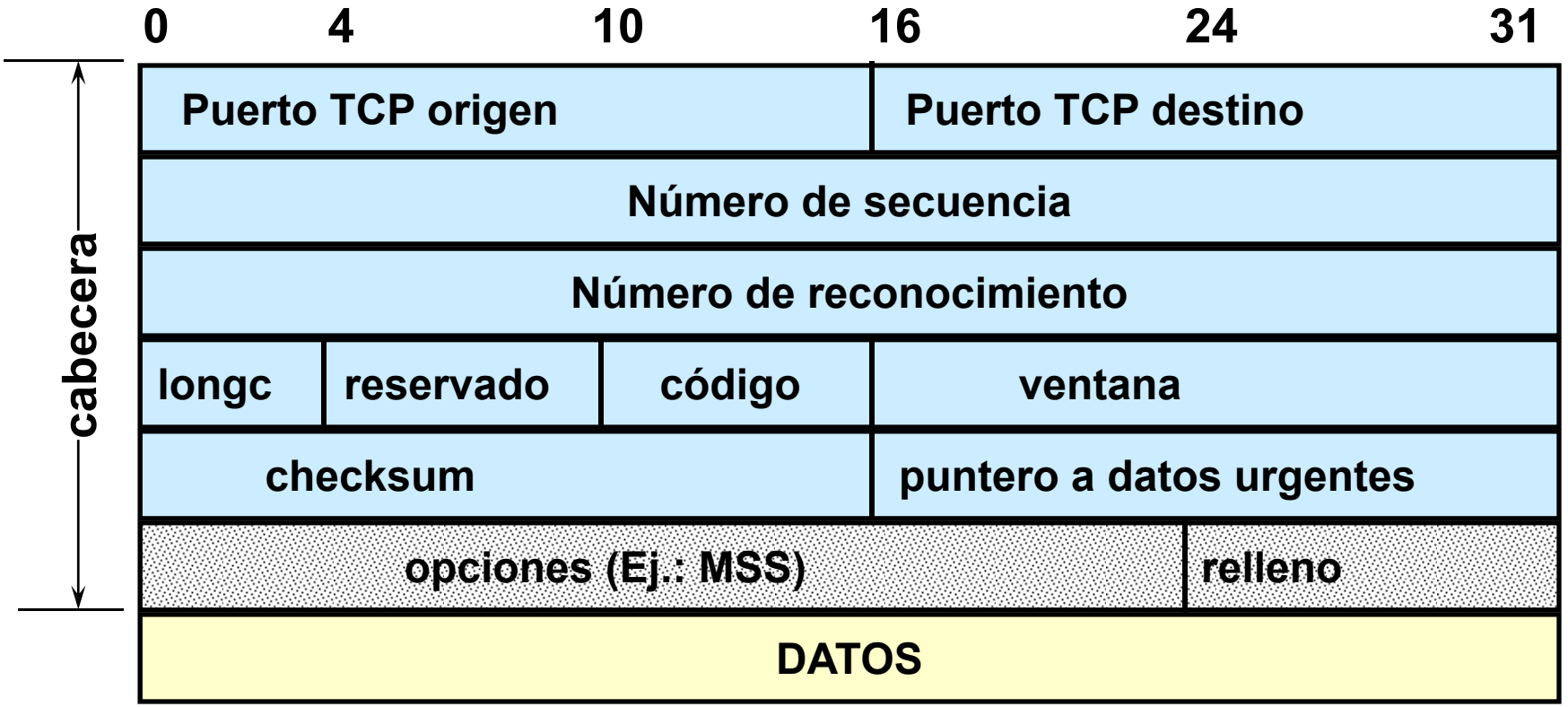
1: TCP Reno no realiza slow start (fast recovery) después de recibir tres ACKs duplicados

- **AIMD**: *Additive Increase, Multiplicative Decrease*
  - Incrementa *CW* en 1 por cada RTT
  - Divide por 2 el valor del *Threshold* cuando se detecta una pérdida

Encapsulado de mensajes TCP



Formato de un segmento TCP (I)



Formato de un segmento TCP (II)

- Cada segmento TCP se divide en: Cabecera y datos.
- **Puerto fuente** y **puerto destino** identifican a las aplicaciones, en los dos extremos de la conexión.
- **Número de secuencia** identifica la secuencia del primer octeto de datos del segmento en el flujo de datos del emisor.
- **Número de reconocimiento** indica cual es el siguiente octeto de datos que se espera recibir.
  - Es válido si el bit ACK (campo código) está activo.
- La **longitud de la cabecera** (longc) se expresa en palabras de 32 bits.

Formato de un segmento TCP (III)

- El campo **ventana** indica el tamaño del buffer del receptor (control de flujo).
  - Determina el tamaño de la ventana de transmisión.
- El **cheksum** se utiliza para la detección de errores.
  - Se aplica a todo el segmento (cabecera+datos).
  - Su cálculo es idéntico al de UDP (pseudo-cabecera).
- Las **opciones** permiten negociar algunos parámetros entre los TCPs de ambos extremos.
  - Ejemplo: Tamaño máximo de segmento (MSS) durante el establecimiento de conexión.

Formato de un segmento TCP (IV)

- El campo **reservado** no se utiliza.
- Los bits de **código** determinan el tipo de segmento y el significado de algunos de los campos de la cabecera.

bit (de izquierda a derecha)	Significado si está a uno
URG	*El puntero a datos urgentes es válido*
ACK	*El campo de reconocimiento es válido*
PSH	*Este segmento solicita una operación PUSH*
RST	*Cancelar la conexión*
SYN	*Establecimiento de conexión*
FIN	*El emisor llegó al final de su secuencia de datos*

TCP: Datos urgentes

- No siguen la secuencia de los datos “normales”
  - Se avisa al receptor de su llegada, independientemente de los datos que haya en cola
- Tienen interés para indicar **situaciones de excepción**
  - Ejemplo: Abortar programas en una terminal remota
- Después de haber leído todos los datos urgentes, la aplicación vuelve al modo normal de funcionamiento
- Los detalles de cómo TCP informa a la aplicación de la presencia de datos urgentes dependen del S.O.
- Los datos urgentes viajan entre los datos de un segmento
  - El puntero de datos urgentes indica dónde finalizan

TCP: Función de carga (PUSH)

- TCP elige la distribución de los datos en segmentos
  - **Por razones de eficiencia** suele acumular un número razonable de octetos antes de un envío
- Esto a veces **penaliza** el rendimiento de una aplicación.
  - Ejemplo: Terminal remota (TELNET).
- Para obligar a TCP a enviar inmediatamente los datos acumulados → **Operación PUSH**
- La aplicación puede solicitar una **transmisión inmediata**.
  - En general, es un parámetro de la conexión.
- Los datos se envían **sin esperar** a llenar un segmento.
  - El segmento generado lleva el bit PUSH del campo código a 1
- En recepción el TCP debe entregar los datos a la aplicación lo antes posible.
  - Por ejemplo, sin esperar a llenar el buffer de recepción.

Establecimiento/Cierre de una conexión TCP

- TCP ofrece un servicio orientado a la conexión:
  - **Establecimiento** de conexión:
    - Sincronización entre los dos extremos para iniciar un diálogo fiable.
    - Negociación de parámetros de la conexión (número de secuencia inicial, tamaño de ventana, tamaño máximo de segmento, etc).
  - **Transferencia** de datos.
  - **Cierre** de conexión:
    - Necesidad de realizar un cierre de conexión consensuado.
    - Liberación de los recursos reservados en el establecimiento.

- Para ello, **utiliza un servicio** de entrega (IP) **no fiable** !!.
  - Los procesos de establecimiento y cierre DEBERÁN garantizar aperturas y cierres correctos.

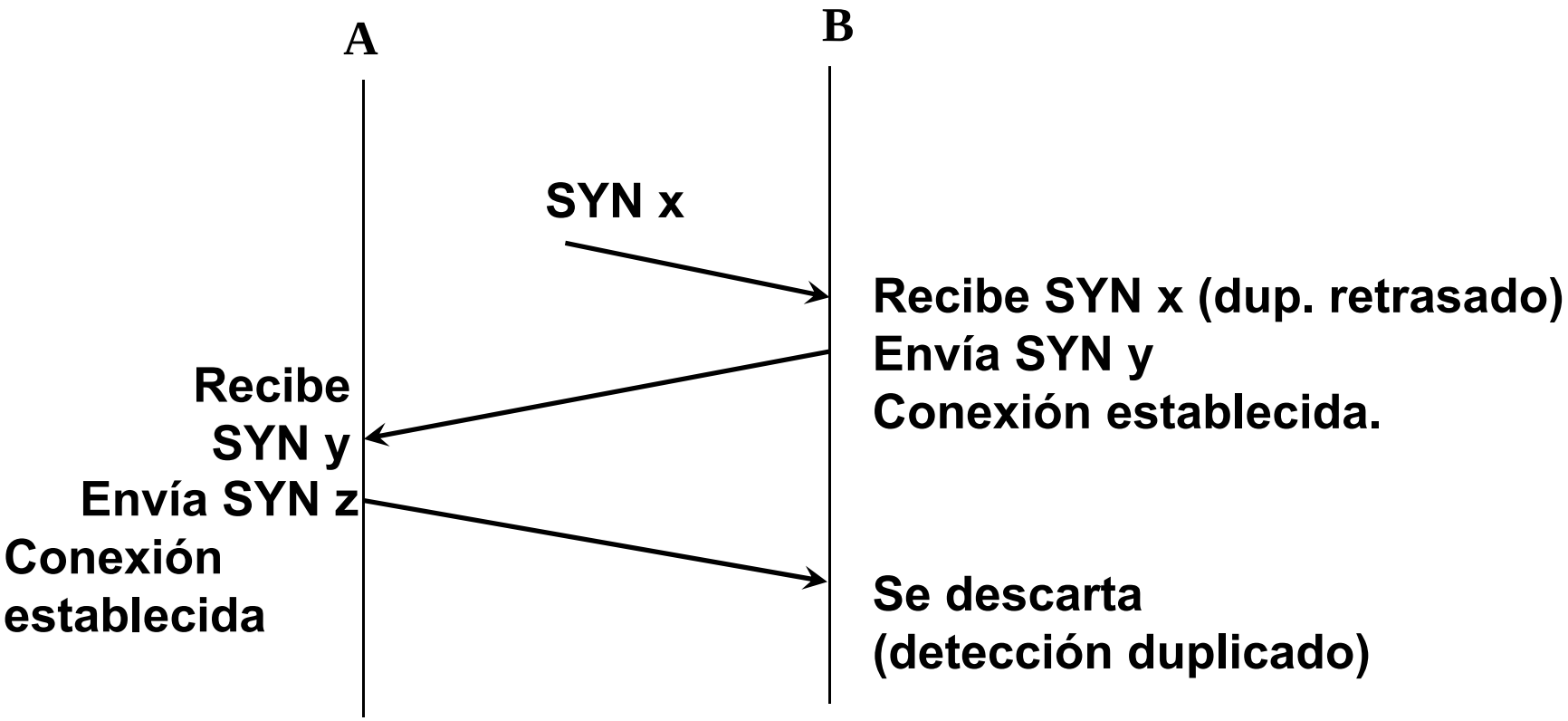
TCP: Establecimiento de una conexión

- **Sincronización** → Ambas partes deben ponerse de acuerdo para que la comunicación sea posible
  - **Apertura pasiva**: Uno de los extremos espera a que otro inicie la conexión. Normalmente se utiliza un puerto “bien conocido” (típicamente en aplicaciones servidor)
  - **Apertura activa**: El otro extremo solicita establecer una conexión. Normalmente utiliza un puerto asignado dinámicamente (típicamente en aplicaciones cliente)

El problema de los duplicados

- La existencia de temporizadores y el tiempo de respuesta no acotado pueden causar duplicados.
- Cuando el **duplicado pertenece** a una **conexión activa**
  - Si es de establecimiento de conexión se descarta.
  - Si es de datos sólo ocasiona problemas si los números de secuencia se repiten.
  - Para este último caso …
    - Tiempo de vida de los paquetes acotado (TTL).
    - Rango de números de secuencia elevado (2^{32}).
- Cuando el **duplicado pertenece** a una **conexión cerrada**
  - Si es de datos no ocasiona problemas.
    - Excepción → Nueva conexión con mismo identificador y número de secuencia en ventana de recepción.
    - Solución → Elegir el número de secuencia inicial de forma aleatoria
  - Si es de establecimiento de conexión, el protocolo a dos bandas falla...

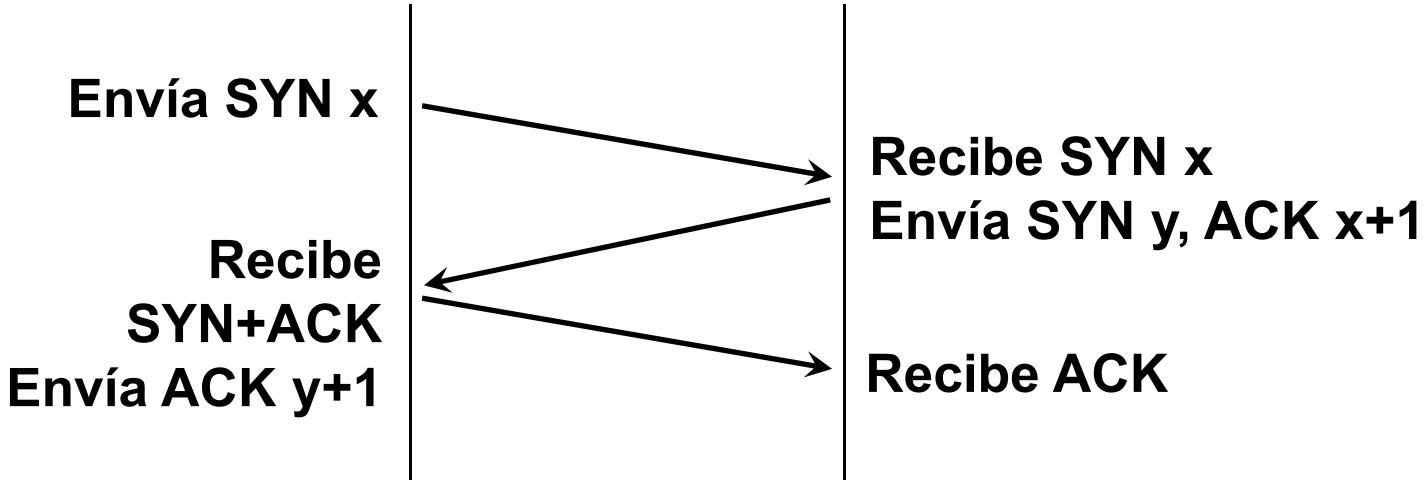
Protocolo a dos bandas



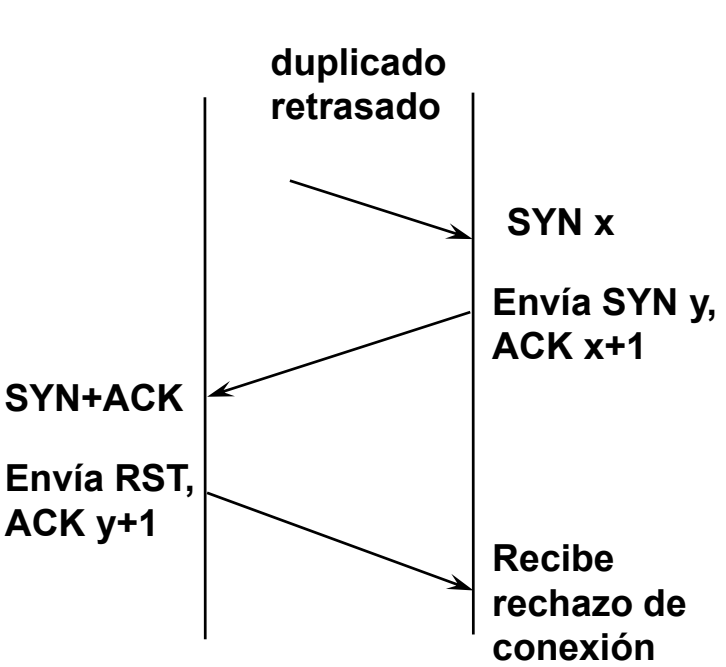
¡¡¡ Secuencias distintas !!!

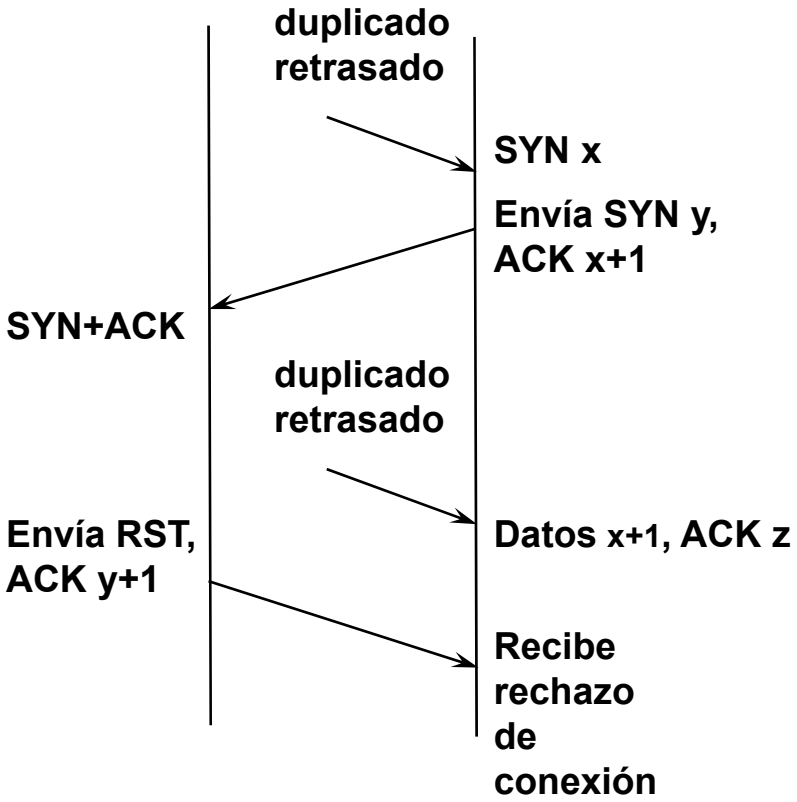
Protocolo a tres bandas

- Cada extremo deberá distinguir de forma inequívoca los duplicados retrasados de conexiones ya cerradas
- Los duplicados retrasados serán detectados utilizando reconocimientos de los números de secuencia iniciales que ambos extremos han seleccionado.



Protocolo a tres bandas (escenarios)

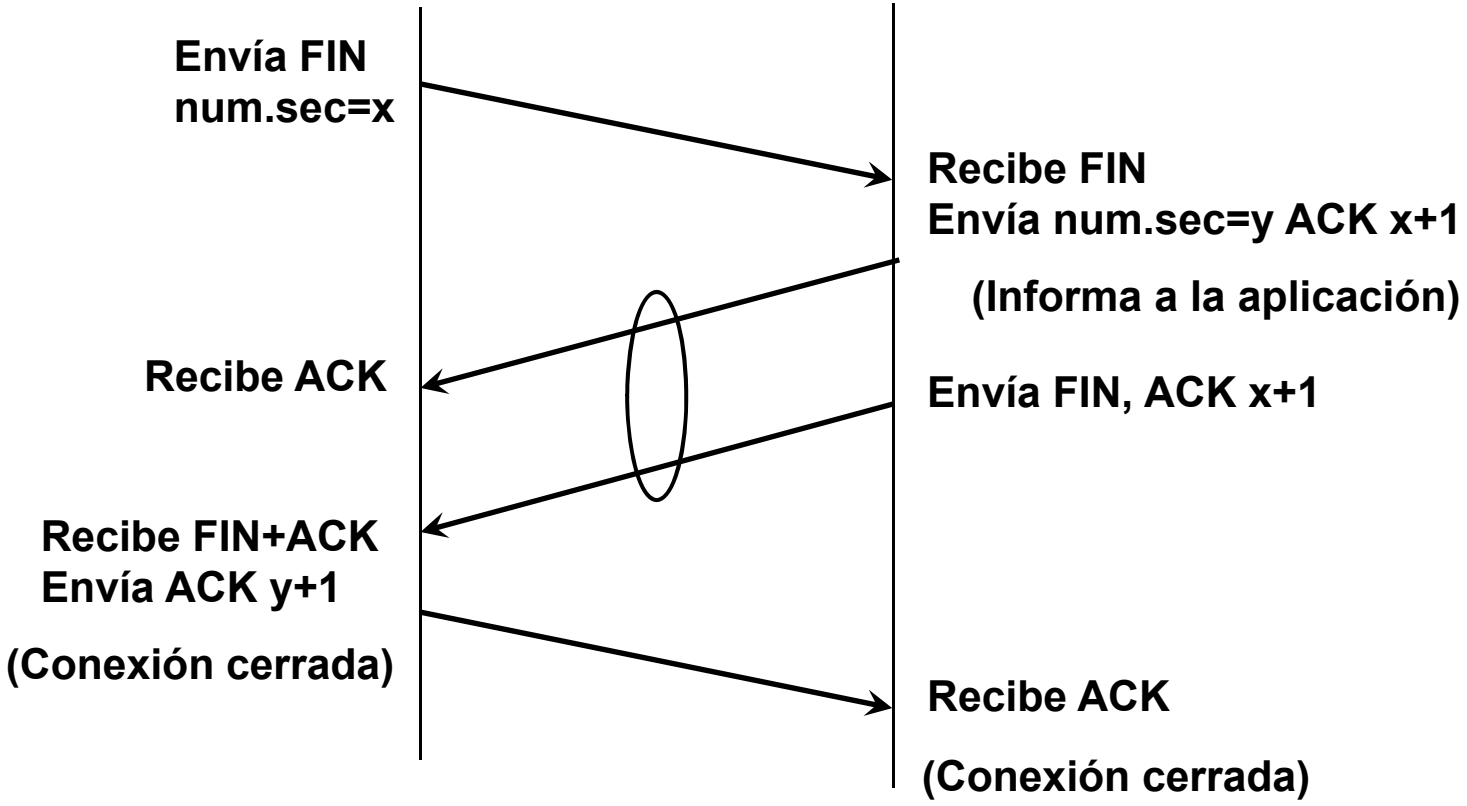




Cierre de una conexión TCP

- Cierre de conexión:
  - Utiliza **segmentos** con el **bit FIN activado**.
  - No pueden llevar datos
  - Requieren números de secuencia:
    - Evitar entrega desordenada
- Protocolo a tres bandas modificado:
  - Pretende conseguir un **cierre ordenado**.
- **Cierre independiente** de ambos sentidos.
- Tras recibir la solicitud de fin de conexión, aún se pueden enviar datos pendientes, que serán aceptados en el otro extremo.

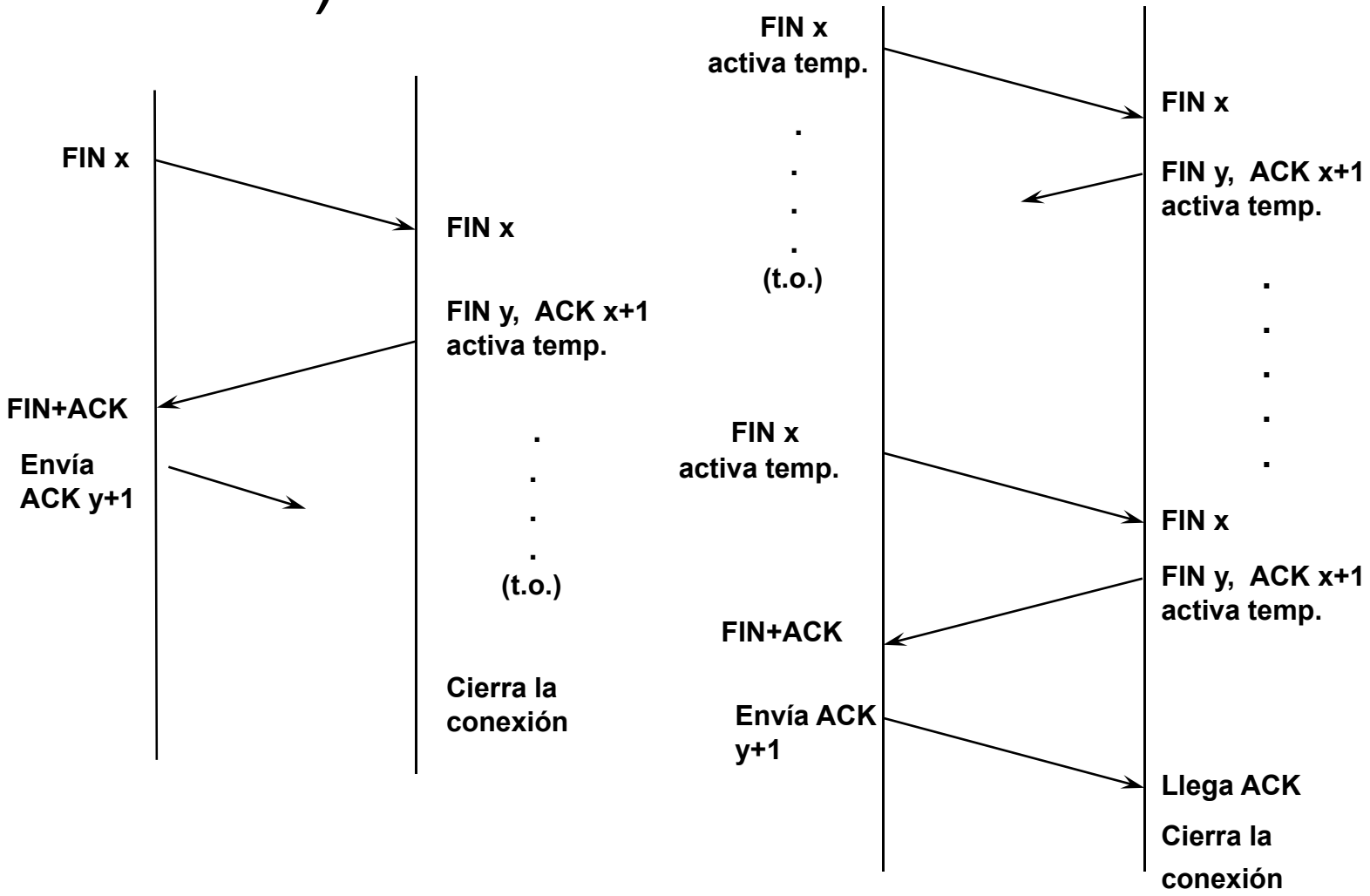
Cierre de conexión: Protocolo a tres bandas



Problemas en el cierre de una conexión

- No puede garantizarse un cierre ordenado:
  - Siempre puede perderse el último mensaje (no reconocido).
  - Ejemplo: El problema de los Generales Bizantinos.
- Solución de compromiso:
  - Se utiliza un temporizador asociado al segmento FIN inicial y otro para la espera del último reconocimiento (2xMSL)
- Conexión semi-abierta (host crashes):
  - Un extremo cae (Ej.: perdida de alimentación), dejando todas sus conexiones TCP abiertas.
  - Solución: Uso de temporizadores que detectan periodos de inactividad + intercambio de ACKs.

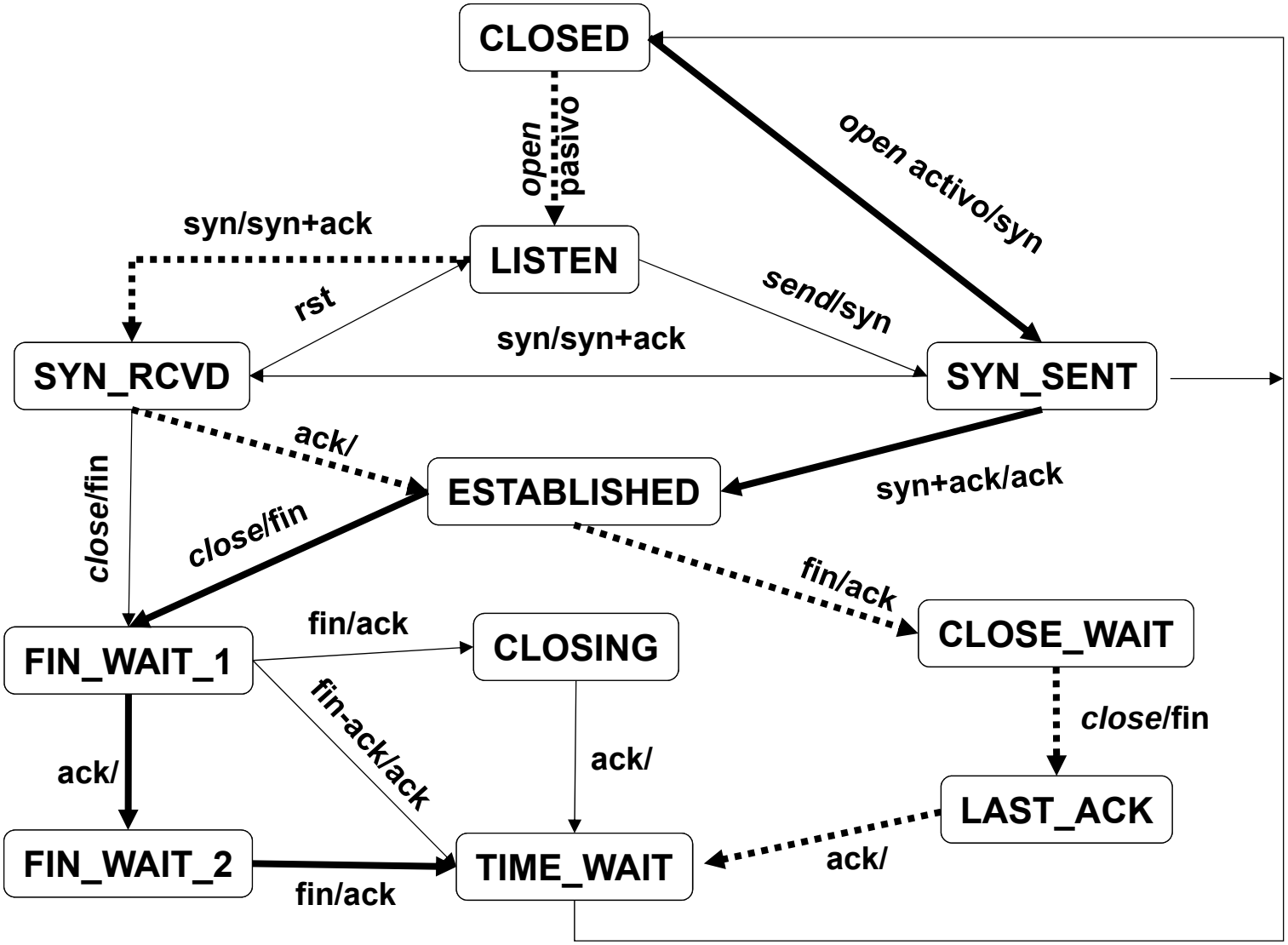
Problemas en el cierre de conexión (escenarios)



Cierre abrupto de una conexión TCP

- No se emplea en condiciones normales.
- Se solicita mediante un **segmento con el bit RST**.
- El otro extremo **cancela** inmediatamente la **conexión**.
  - Previa verificación de su secuencia y reconocimiento.
- TCP informa también a la aplicación.
- RST aborta la conexión en ambos extremos
  - Cesa inmediatamente la comunicación en ambos sentidos.
  - Se liberan los recursos asociados.

Máquina de estados TCP



Conclusiones sobre TCP (I)

- Protocolo más complejo.
- Servicio de transferencia de datos fiable y ordenado:
  - Corrige las pérdidas de paquetes:
    - Reconocimientos positivos.
    - Retransmisiones.
  - Soluciona los duplicados:
    - Números de secuencia.
    - Tiempo de vida acotado en los datagramas.
- Control de flujo mediante ventana deslizante y notificación de ventana.
- Control de congestión limitando el tamaño de la ventana de transmisión.

Conclusiones sobre TCP (II)

- Distinción entre diferentes destinos en la misma máquina: los puertos
- Establecimiento de conexión:
  - segmentos SYN.
  - protocolo a tres bandas.
- Liberación de ordenada de la conexión:
  - segmentos FIN.
  - protocolo a tres bandas.
- Liberación abrupta de la conexión:
  - segmentos RST.