Tema 1.3

Protocolo TCP

(Transmission Control Protocol)

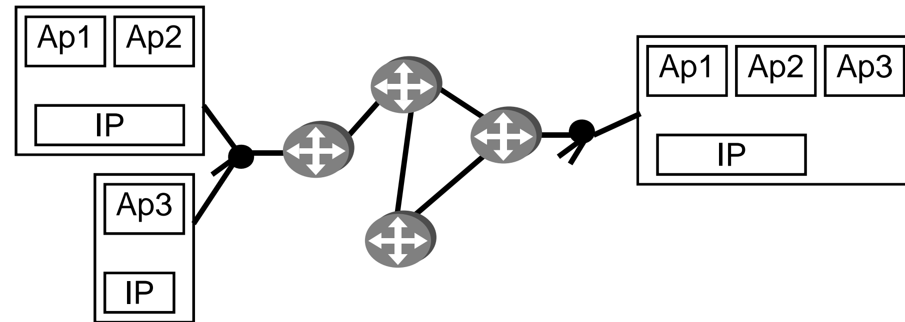
Capa de transporte.
Entrega confiable

Índice

- Introducción: capa de transporte confiable ………………. 3
- Servicio ofrecido …………………….. 4
- Formato de segmento TCP ……. 9
- Transmisión/Recepción de segmento TCP ………………………… 13
- Fases de conexión …………………. 14
- Control de errores, flujo y congestión ………………………………. 25
- Ejemplos ………………………………… 52
- Estados de conexión TCP ………. 53
- Bibliografía recomendada ………. 55

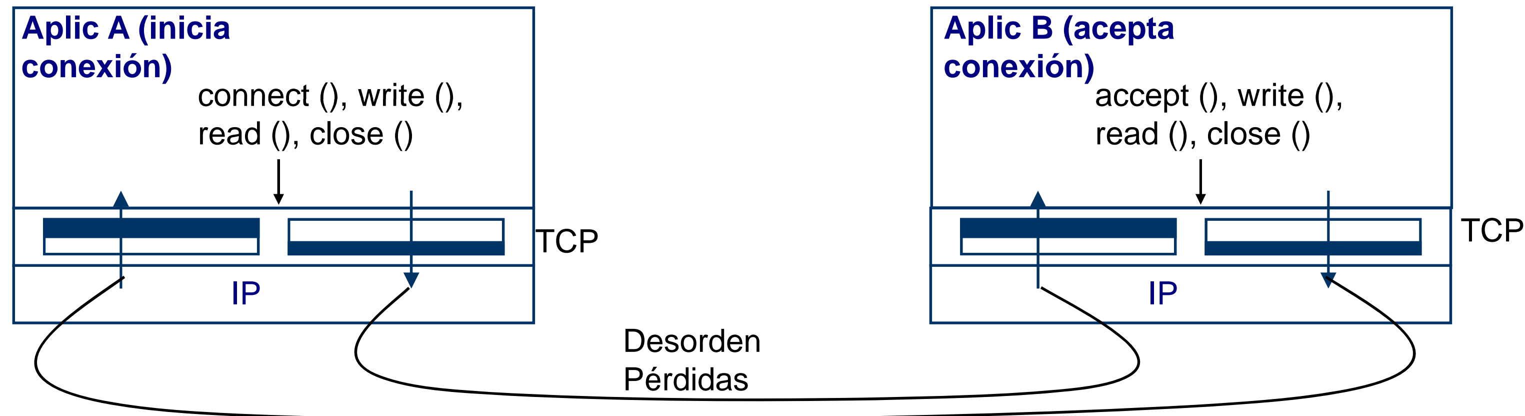
Introducción: Capa de transporte confiable

- La fiabilidad es requisito para muchas aplicaciones.
- Causas de pérdidas:
  - Fallo hardware.
  - Mala configuración dispositivos IP.
  - Descarte de datagramas en los routers, por situaciones temporales de congestión.
- Conclusión: Se requiere un protocolo de transporte que permita el intercambio confiable.



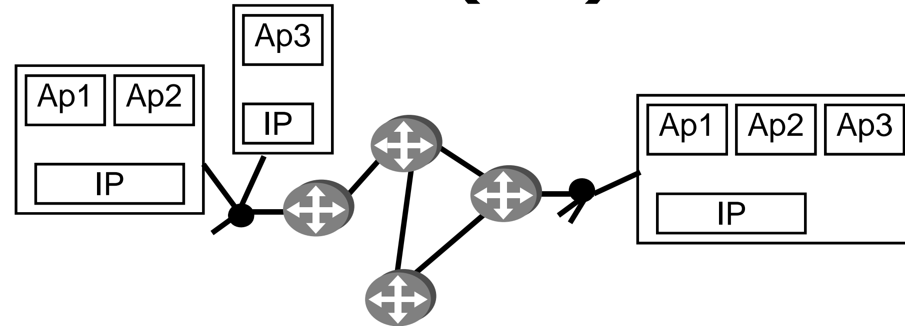
- TCP (*Transmission Control Protocol*).
  - El protocolo más complejo de la pila de protocolos TCP/IP.
  - Más del 90% del tráfico generado, es transportado por el protocolo TCP.

Servicio ofrecido (I)



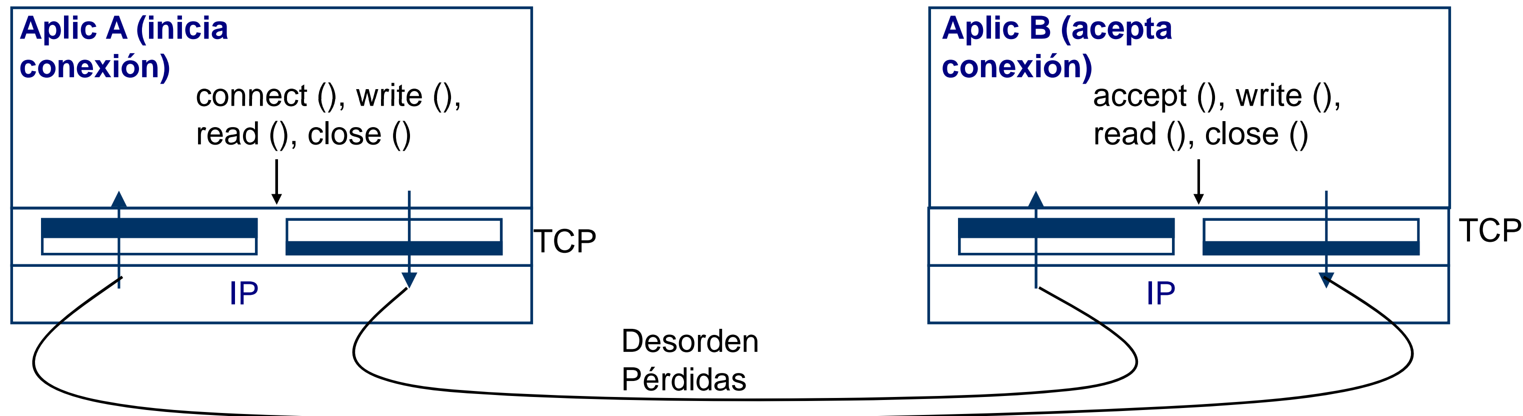
- Orientado a conexión.
  - Establecimiento de la conexión: extremo iniciador (`connect ()`) y extremo aceptador (`accept ()`).
  - Fase de envío de datos:
    - Bidireccional: tras el establecimiento de la conexión, la conexión es simétrica, pudiendo intercambiarse datos en ambos sentidos.
    - Primitivas `write ()` y `read ()`.
  - Cierre de la conexión. Primitiva `close ()`.
- Fiabilidad en la entrega de datos: Los datos enviados por un extremo (`write ()`), son entregados al otro (`read ()`), sin desorden.

Servicio ofrecido (II)



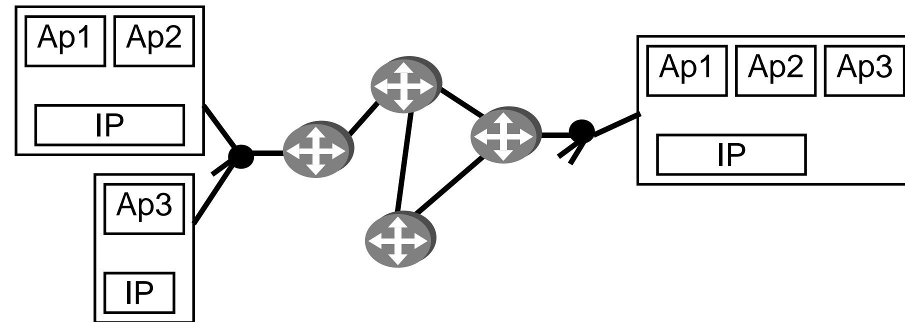
- Control de flujo (definición: el extremo receptor es capaz de controlar el volumen generado por el transmisor)
  - Hay dos receptores y dos transmisores => existen dos mecanismos de control de flujo:
    - Extremo A controla el flujo de datos B→A.
    - Extremo B controla el flujo de datos A→B.
- Control de congestión: TCP intenta (como protocolo de transporte, y por tanto en los extremos de la red) aliviar posibles situaciones de congestión que se produzcan en la red. Para ello:
  - Cuando el extremo A detecta la pérdida de segmentos TCP en sentido A→B, asume que existe congestión en algún punto de la red. En ese momento, A reduce la tasa de datos enviada.
  - La tasa enviada se recupera en el caso de que no se detecte pérdida de segmentos.
  - Existen dos mecanismos de control de congestión.
  - Nota: El mecanismo de control de congestión es necesario para TCP, que transporta más del 90% del tráfico.

Servicio ofrecido (III)



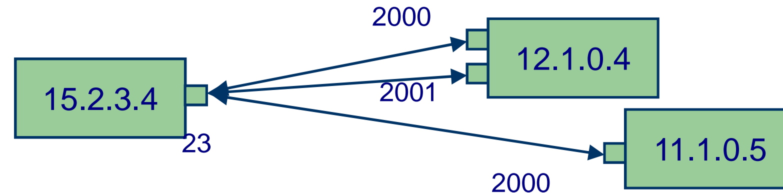
- TCP no reconoce ninguna estructura en los datos que transporta:
  - Identifica cada byte de datos con un número consecutivo.
  - No respeta la estructura de los datos entregados en la sentencia `write ()`. Ejemplo:
    - A.write (mensaje de 1000 bytes de datos). Bytes 24000 a 24999 (según numeración TCP A→B).
    - TCP transporta este mensaje en dos segmentos de 497 bytes (24000 a 24496) y 503 bytes (24497 a 24999).
    - Ambos segmentos llegan correctamente al extremo B.
    - Primera sentencia B.read () devuelve 150 bytes (24000 a 24149) .
    - Segunda sentencia B.read () devuelve 100 bytes (24150 a 24249) .
    - Tercera sentencia B.read () devuelve 750 bytes (24250 a 24999) .

Servicio ofrecido (IV)



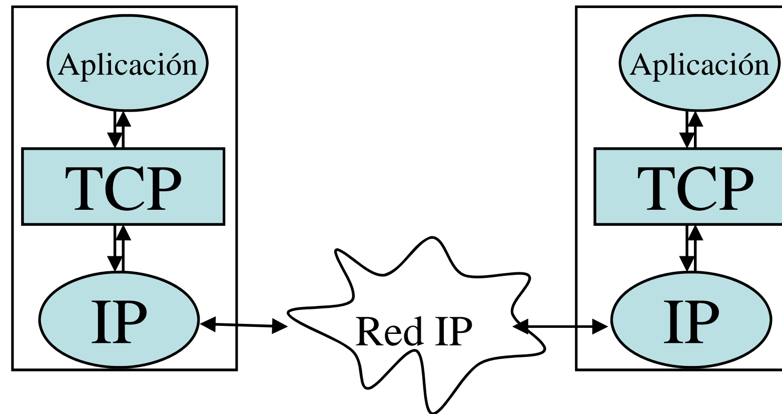
- Multiplexación y demultiplexación de conexiones TCP:
  - Cada aplicación en un PC se identifica mediante un número de puerto TCP (0...65535).
  - Una conexión TCP viene identificada por 4 valores (IP origen, Puerto origen, IP destino , puesto destino). Estos valores son indicados durante el establecimiento de la conexión.
    - El iniciador de la conexión: 1) debe conocer el IP/puerto en el que la aplicación destino acepta conexiones TCP. 2) debe reservar un número de puerto local.
    - El aceptador de la conexión: 1) debe registrarse ante el S.O. para recibir conexiones en un determinado puerto TCP.

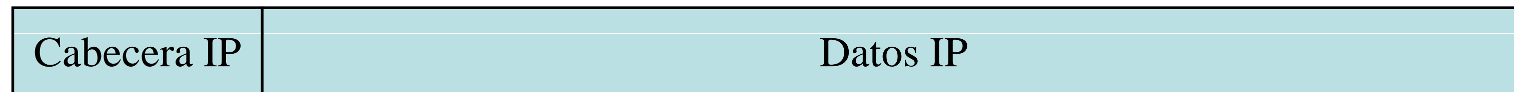
Servicio ofrecido (V)



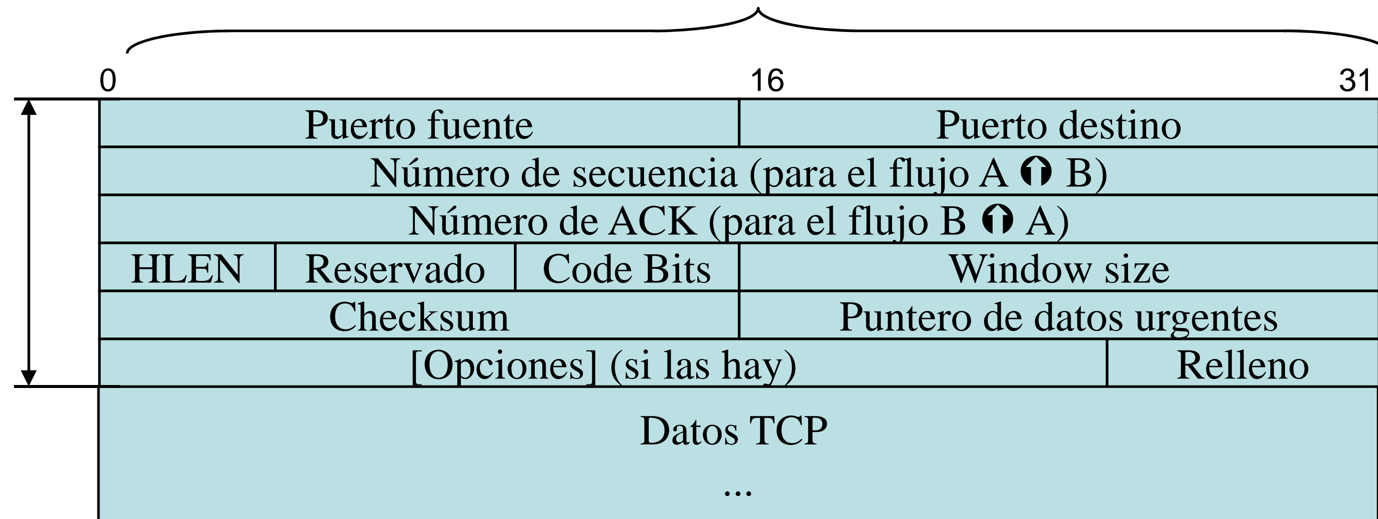
- Ejemplo: Aplicación cliente-servidor (Telnet)
  - Aplicación servidor (15.2.3.4): Registra como aceptador de conexiones TCP en puerto 23.
  - Aplicación cliente (12.1.0.4): Inicia conexión con (15.2.3.4 ; 23). En general, el Sistema Operativo reserva automáticamente un puerto TCP local libre [p.e. 2000] para esta conexión.
  - (IP origen ; puerto origen ; IP destino ; puerto destino) son únicos para cada conexión TCP.
- Tipos de puertos (ver http://www.iana.org/assignments/port-numbers)
  - Puertos bien conocidos (0...1023, *well-known ports*). Asignados por la IANA. En la mayoría de los sistemas se requieren permisos de root para registrar una aplicación como aceptadora de conexiones en estos puertos.
  - Puertos registrados (1024...49151). Listados por la IANA, pero en la mayoría de los sistemas pueden ser usados por aplicaciones ordinarias.
  - Puertos dinámicos o privados (49152...65535). Sin recomendaciones de la IANA.
- Estas mismas asignaciones de puertos se usan para puertos UDP, cuando ese servicio existe.

Formato de segmento TCP (I)





Cabecera TCP
- tamaño variable, múltiplo de 4 bytes
- mínimo 20 bytes (lo más usual).
-Suponemos un segmento desde extremo A a B.



Formato de segmento TCP (II)

Cabecera TCP
- Suponemos un segmento transmitido desde extremo A a B.

0 16 31

Puerto fuente			Puerto destino	
Número de secuencia (para el flujo A → B)				
Número de ACK (para el flujo B → A)				
HLEN	Reservado	Code Bits	Window size	
Checksum			Puntero de datos urgentes	
[Opciones] (si las hay)				Relleno

- Puerto fuente (16 bits): Puerto TCP origen (aplicación en extremo A se asocia a ese puerto).
- Puerto destino (16 bits): Puerto TCP destino (aplicación en extremo B se asocia a ese puerto).
- Número de secuencia (SN, *Sequence Number*): $ID_{A \rightarrow B}$ del primer byte de datos transportado en este segmento.
- Número de ACK: $ID_{B \rightarrow A}$ del primer byte de datos que A espera recibir de B.

Importante: TCP numera (identifica) los bytes individualmente, con números consecutivos. No se numeran los segmentos.

Formato de segmento TCP (III)

Cabecera TCP
- Suponemos un segmento transmitido desde extremo A a B.

0		16	31
Puerto fuente		Puerto destino	
Número de secuencia (para el flujo A → B)			
Número de ACK (para el flujo B → A)			
HLEN	Reservado	Code Bits	Window size
Checksum		Puntero de datos urgentes	
[Opciones] (si las hay)			Relleno

- HLEN (*Header Length*, 4 bits): Tamaño de la cabecera TCP (que puede variar por el campo opciones) dividido por 4 (tamaño real = HLENx4 bytes).
  - Nota: El tamaño de cabecera TCP deba ser múltiplo de 4 bytes.
- Reservado (6 bits): No se utiliza.
- Code bits (6 bits):
  - bit URG: Este segmento lleva datos fuera de banda (la posición de estos datos vendrá entonces indicada por el puntero a datos urgentes).
  - bit ACK: Este segmento incluye en el campo ACK el ID del siguiente byte que se espera recibir.
  - bit PSH: Se solicita la operación PUSH.
  - bit RST: Reinicio de la conexión.
  - bit SYN: El campo SN incluye información para inicializar los números de secuencia (durante la fase de establecimiento de la conexión TCP).
  - bit FIN: Indica que el extremo A no enviará más datos (durante fase de cierre de conexión TCP).

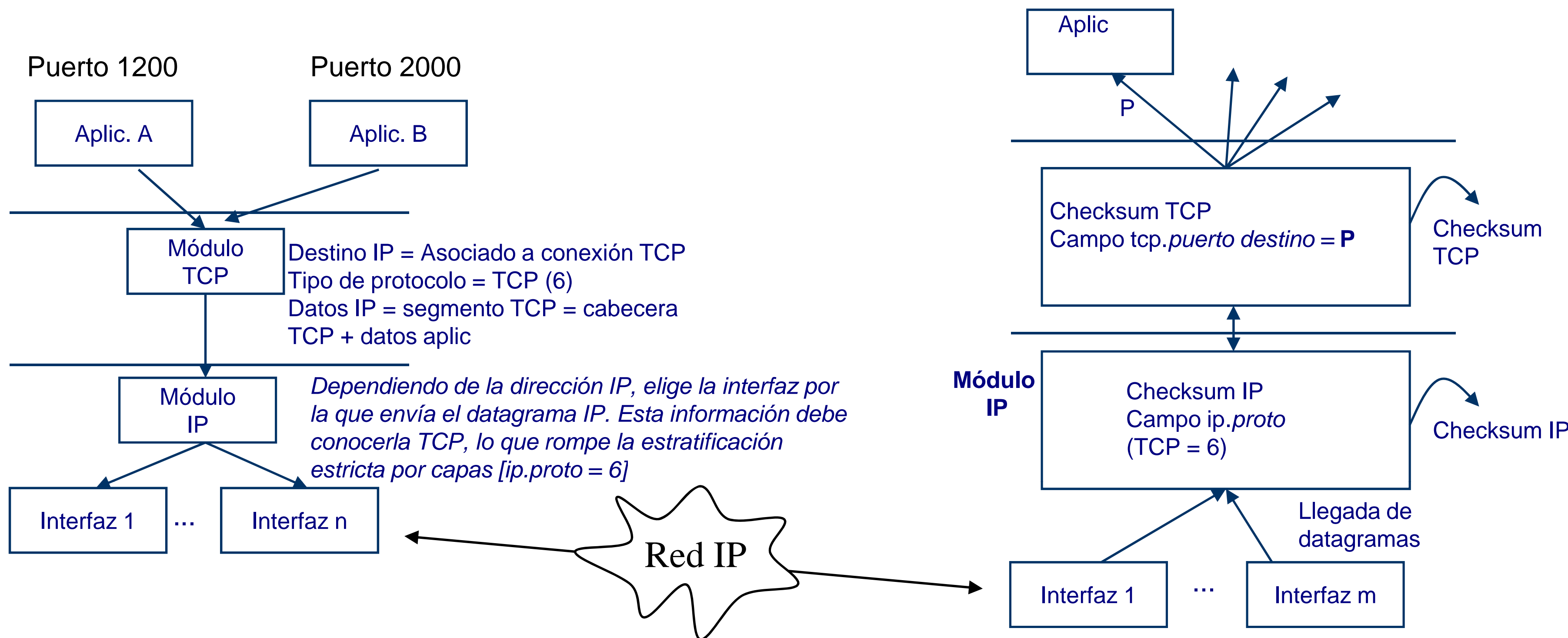
Formato de segmento TCP (IV)

Cabecera TCP
- Suponemos un segmento transmitido desde extremo A a B.

0			16	31
Puerto fuente			Puerto destino	
Número de secuencia (para el flujo A → B)				
Número de ACK (para el flujo B → A)				
HLEN	Reservado	Code Bits	Window size	
Checksum			Puntero de datos urgentes	
[Opciones] (si las hay)				Relleno

- *Window size* (16 bits): A indica a B el tamaño de ventana de transmisión que B debe fijar (mecanismo de control de flujo).
- Checksum (16 bits): Calculado a partir de:
  - IP destino.
  - IP origen (rompe la estratificación por capas!!!).
  - Cabecera TCP.
  - Datos TCP.
- Opciones (tamaño variable): Muy poco habitual. Veremos un ejemplo en el establecimiento de la conexión TCP.
- Relleno (tamaño variable): Con el objetivo de que el tamaño de la cabecera sea un múltiplo de 4 bytes.

Transmisión/recepción de un segmento TCP

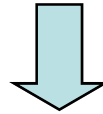


- Pregunta: qué sucede si un datagrama que transporta un segmento TCP se fragmenta? Cómo afecta esto a la capa TCP?

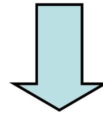
Fases de una conexión TCP

Establecimiento de conexión

(3 WHS, *3-way handshake*)



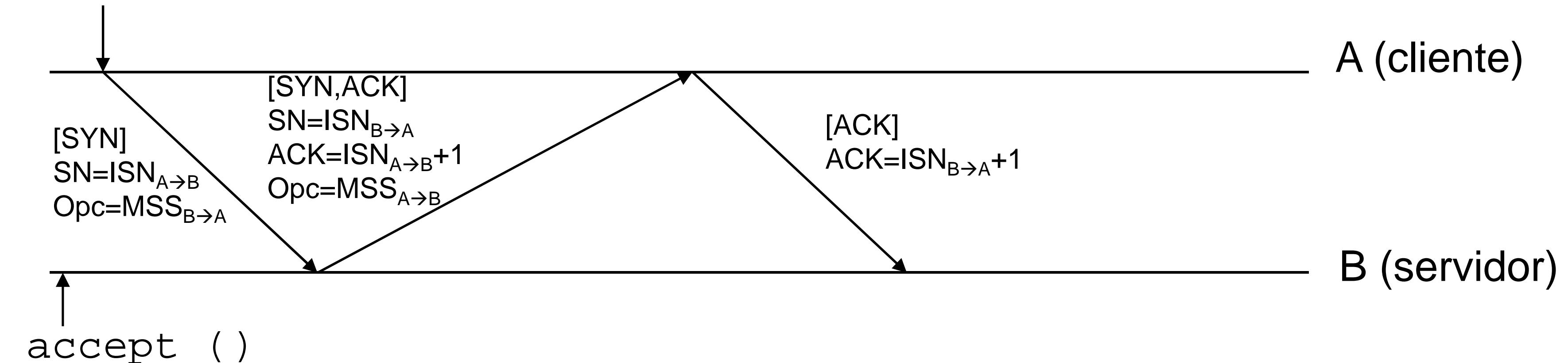
Fase de envío de datos



Fase de terminación de la conexión

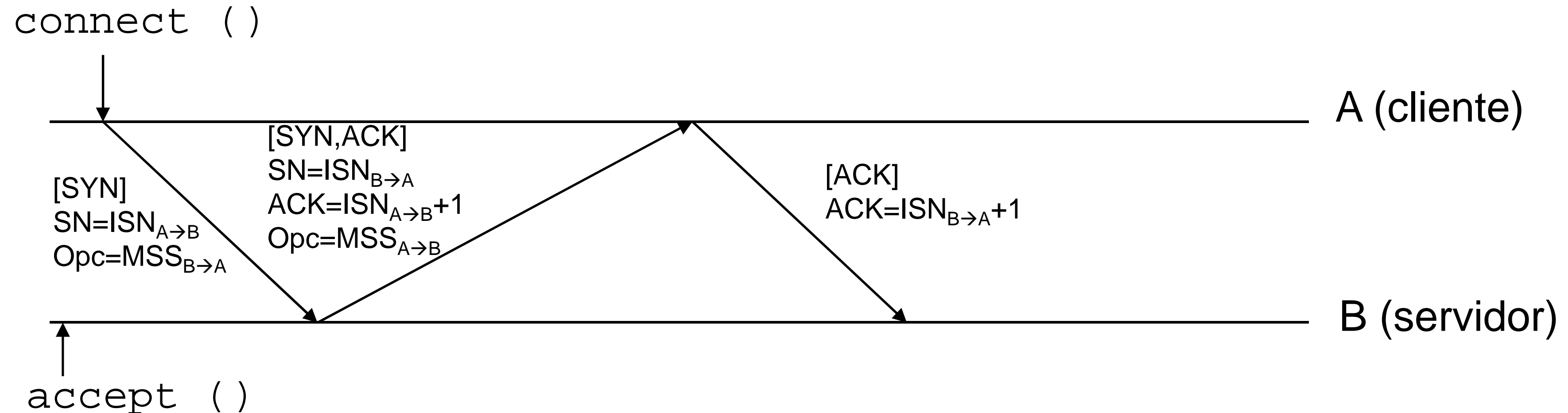
(requiere 4 segmentos)

Establecimiento de conexión TCP (I)



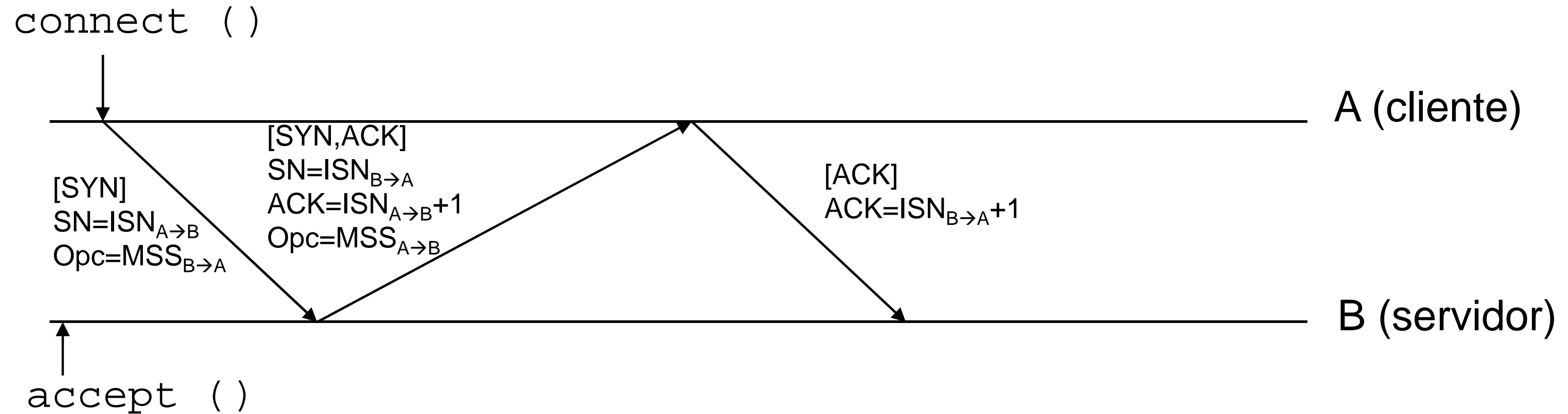
- Para el establecimiento de conexión TCP se emplea el algoritmo 3-WHS (3 Way Handshake, saludo a tres vías).
- 1º segmento (A→B, iniciación de conexión).
  - Segmento con puerto destino el indicado. Puerto origen indicado por la aplicación, o seleccionado por el S.O. entre los puertos libres (lo más habitual).
  - Flag SYN activado, para indicar que el contenido del SN es el 1º número de sucuencia.
  - SN = número aleatorio creado por el software TCP = ISN$_{A\rightarrow B}$.
  - Campo opciones = MSS$_{B\rightarrow A}$ (*Maximum Segment Size*) = tamaño máximo de campo de segmento que A está dispuesto a recibir (condiciona la forma en que B genera segmentos).

Establecimiento de conexión TCP (II)



- 2º segmento (B→A).
  - Flag SYN activado, para indicar que el contenido del SN es el 1º número de sucuencia. Flag ACK activado, para indicar que el campo ACK contiene el ID del próximo byte que B espera recibir.
  - SN = número aleatorio creado por el software TCP = $ISN_{B\rightarrow A}$.
  - ACK = $ISN_{A\rightarrow B}+1$ => el primer byte de datos que A transmita a B, tiene como identificador ($ISN_{A\rightarrow B}+1$).
  - Campo opciones = $MSS_{A\rightarrow B}$ (*Maximum Segment Size*) = tamaño máximo de campo de segmento que B está dispuesto a recibir (condiciona la forma en que B genera segmentos)

Establecimiento de conexión TCP (III)



- 3º segmento (A→B).
  - Flag ACK activado, para indicar que el campo ACK contiene el ID del próximo byte que A espera recibir.
  - ACK = ISN$_{B\rightarrow A}$+1 => el primer byte de datos que B transmita a A, tiene como identificador (ISN$_{B\rightarrow A}$+1).

Establecimiento de conexión TCP (IV)

- Tamaño máximo de segmento (*Maximum Segment Size*, MSS)
  - Definición: el MSS del extremo A, es el tamaño máximo **del campo datos** que introducirá en un segmento TCP.
  - Problema: Si TCP genera segmentos demasiado grandes, IP se verá obligada a fragmentarlos, lo cual genera ineficiencias.
  - Objetivo: La capa TCP debe generar segmentos:
    - lo más grandes posibles, para ganar eficiencia en el ratio cabecera/datos.
    - pero sin provocar que el datagrama se fragmente.
  - Pero... TCP se encuentra en los extremos, y desconoce los MTUs de las redes atravesadas!!!
  - Solución [RFC 879]: depende de la implementación del protocolo TCP. En general:
    - Si ambos extremos detectan que se encuentran en la misma red física (IP.origen AND Máscara == IP.destino AND Máscara), entonces asumen que comparten MTU, y acuerdan ambos extremos:

      MSS = MTU – 20 – 20

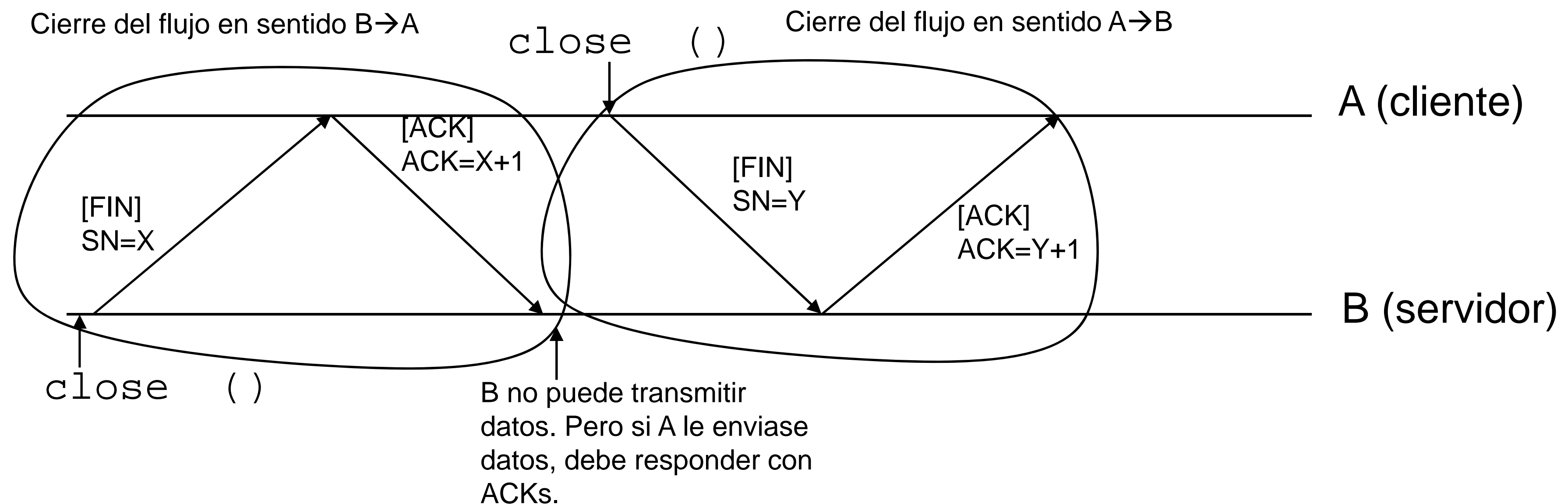
    - En caso contrario, acuerdan (MSS por defecto)

      MSS = 576 – 20 – 20 = 536 bytes
      (Nota: 576 es el MTU habitual en las redes X.25, ahora en desuso)

- El mecanismo MSS funciona razonablemente bien: el % de datagramas fragmentados es asumible en Internet (< 1% según algunos estudios).

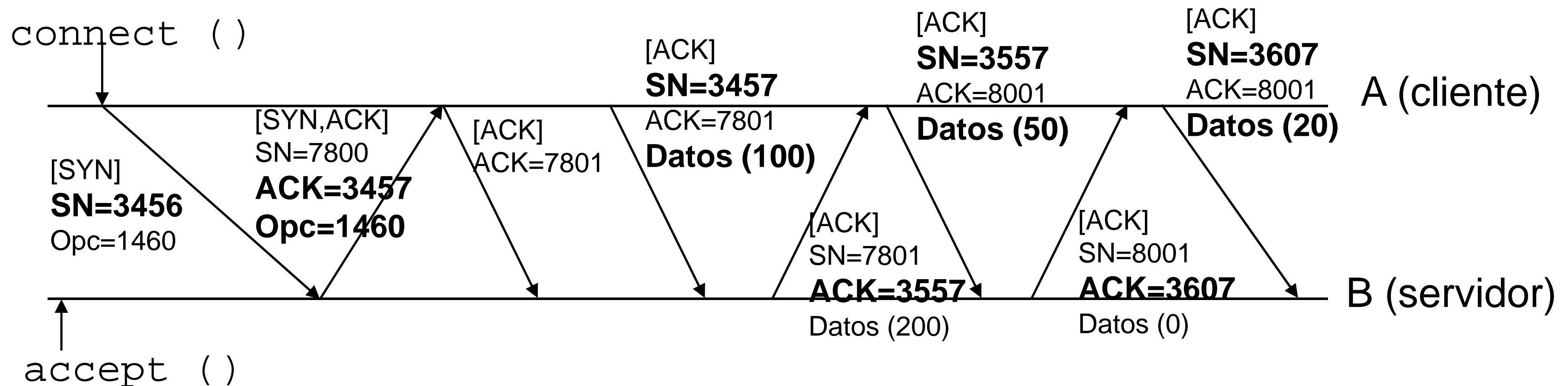
Cierre de conexión TCP

- El cierre de una conexión TCP puede tener varios motivos:
  - Llamada por parte de uno de los extremos a la primitiva `close ()`. Esto provoca el envío de 4 segmentos (figura inferior).
  - Por razones internas del software TCP => se envía un reset de la conexión (segmento con flag RES activo).
- Cierre mediante close ():
  - Cualquier extremo puede llamar a la función `close ()` (independientemente de quién inició la conexión).
  - Cierre de cada flujo por separado (segmento FIN y segmento ACK).



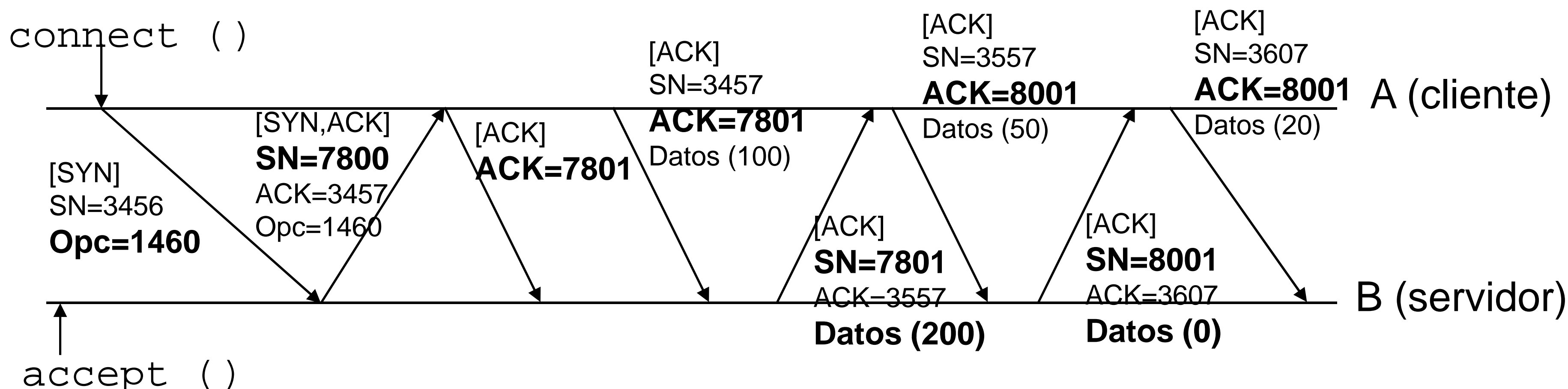
Fase de envío de datos (I)

- Tras el establecimiento de la conexión, ambos extremos son iguales.
- Datos en dos sentidos => dos flujos **independientes**.
- Numeraciones independientes:
  - Numeración datos flujo A→B. Cada byte de datos tiene su identificador $ID_{A\rightarrow B}$. El primer byte de datos del flujo tiene como ID = $ISN_{A\rightarrow B}+1$.
  - Numeración datos flujo B→A. Cada byte de datos tiene su identificador $ID_{B\rightarrow A}$. El primer byte de datos del flujo tiene como ID = $ISN_{B\rightarrow A}+1$.
- Flujo de datos sentido A→B:
  - Segmentos transmitidos por A hacia B:
    - Campo SN = $ID_{A\rightarrow B}$ del primer byte del campo de datos que viaja en este segmento.
    - Campo Datos = datos del flujo A→B.
  - Segmentos transmitidos por B hacia A:
    - Campo ACK = $ID_{A\rightarrow B}$ del próximo byte que B espera recibir. Ejemplo: Si envía el valor 1455698, quiere decir que han llegado correctamente todos los bytes desde $ISN_{A\rightarrow B}+1$ hasta 1455697, y que NO ha recibido el byte 1455698.



Fase de envío de datos (II)

- Flujo de datos sentido B→A:
  - Segmentos transmitidos por B hacia A:
    - Campo SN = $ID_{B \to A}$ del primer byte del campo de datos que viaja en este segmento.
    - Campo Datos = datos del flujo B→A.
  - Segmentos transmitidos por A hacia B:
    - Campo ACK = $ID_{B \to A}$ del próximo byte que A espera recibir. Ejemplo: Si envía el valor 9987634, quiere decir que han llegado correctamente todos los bytes desde $ISN_{B \to A}$+1 hasta 9987633, y que NO ha recibido el byte 9987634.
- Preguntas:
  - Conocido el campo SN de los segmentos transmitidos por A, ¿podemos deducir alguna información del campo ACK de esos mismos segmentos?, ¿y del campo SN de los segmentos que viajan en sentido contrario?, y ¿del campo ACK de los segmentos que viajan en sentido contrario?.

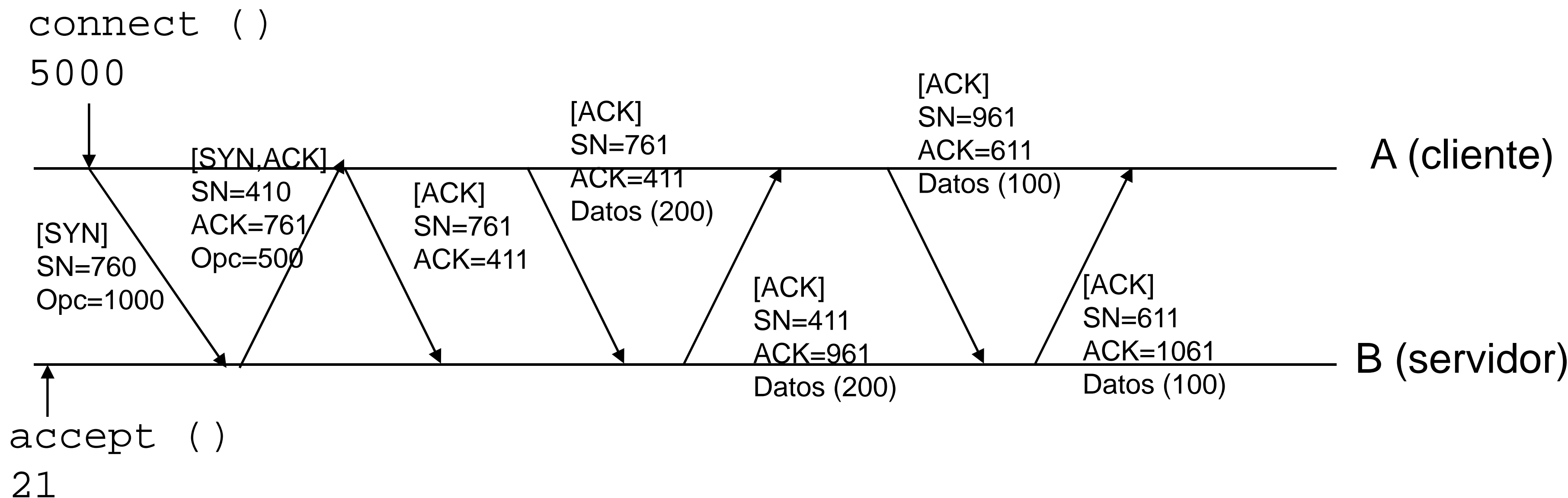


Ejemplo

- **Examen Diciembre 2003**: Un cliente HTTP establece una conexión TCP con un servidor HTTP. El extremo cliente informa de que el tamaño máximo de segmento que está dispuesto a recibir es de 1000 bytes, mientras que el extremo servidor informa que el tamaño máximo de segmento que está dispuesto a recibir es 500 bytes. Escriba el contenido de los campos de la cabecera TCP vacíos en la siguiente secuencia de intercambio de segmentos, suponiendo que no existe pérdida ni desorden en la entrega, salvo cuando sea explícitamente indicado.

P_{origen}	$P_{destino}$	Seq. Number	ACK	Options	Comentarios
5000	21	760			Petición de inicio de conexión
		410			2º mensaje de inicio de conexión
					3º mensaje de inicio de conexión
					Cliente envía 200 bytes de datos
					Servidor envía 200 bytes de datos
					Cliente envía 100 bytes de datos.
					Servidor envía 100 bytes de datos

Ejemplo TCP (resuelto)

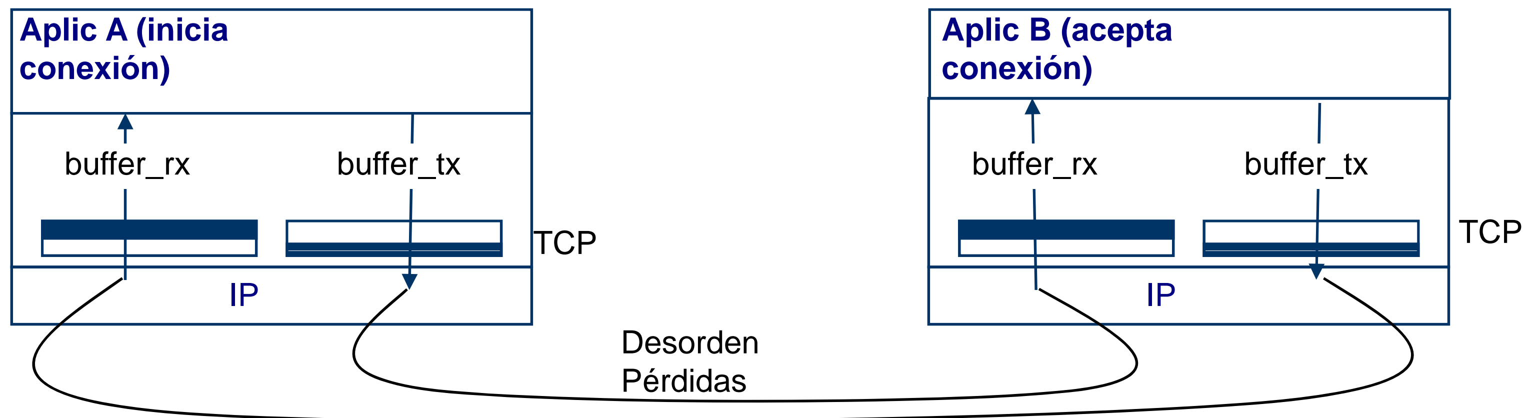


P_{origen}	$P_{destino}$	Seq. Number	ACK	Options	Comentarios
5000	21	760	---	*MSS=1000*	Petición de inicio de conexión
21	*5000*	410	*761*	*MSS=500*	2º mensaje de inicio de conexión
5000	*21*	*761*	*411*	---	3º mensaje de inicio de conexión
5000	*21*	*761*	*411*	---	Cliente envía 200 bytes de datos
21	*5000*	*411*	*961*	---	Servidor envía 200 bytes de datos
5000	*21*	*961*	*611*	---	Cliente envía 100 bytes de datos.
21	*5000*	*611*	*1061*	---	Servidor envía 100 bytes de datos

Introducción al control de errores, flujo y congestión

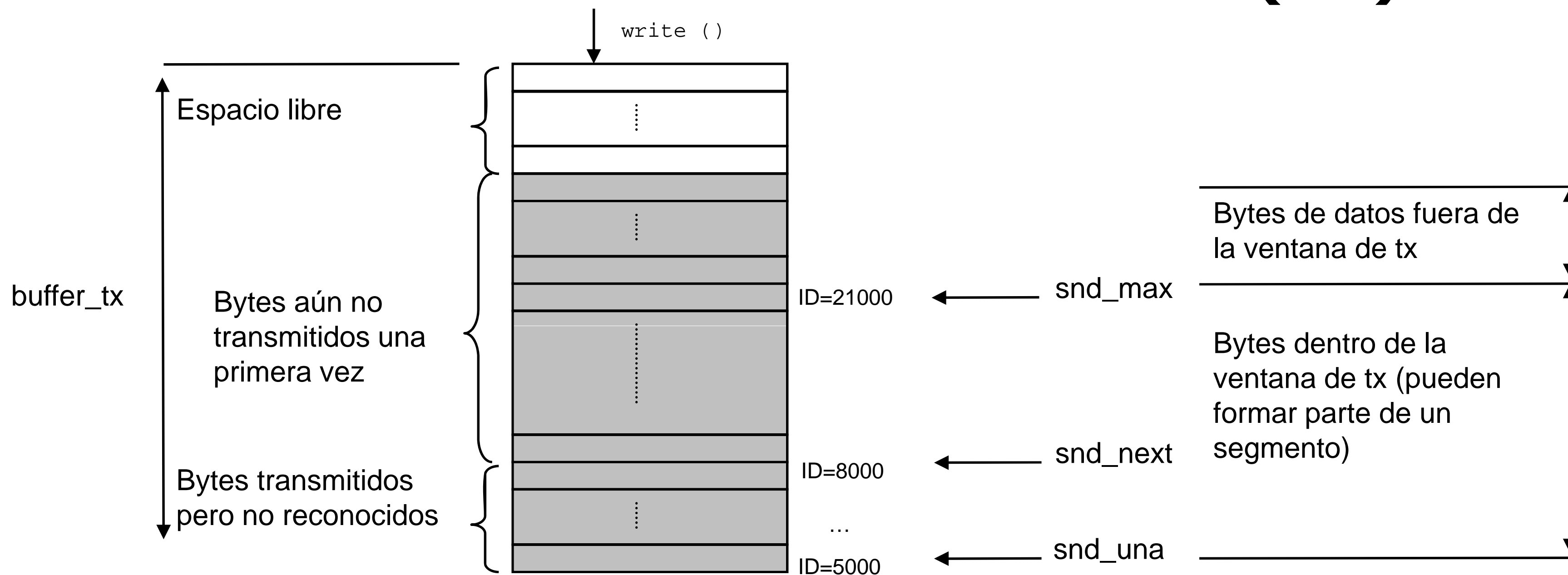
- Control de errores: Mecanismo de retransmisiones para el reenvío de segmentos en caso de pérdidas.
- Control de flujo: Mecanismo que permita a cada extremo receptor controlar el flujo enviado por el transmisor del extremo opuesto.
- Control de congestión: Mecanismo de auto-regulación del flujo de datos generado por los extremos transmisores, en función de la posible situación de congestión estimada en la red.
- Todos estos sistemas se implementan a través del mecanismo de ventana deslizante.

Ventana deslizante (I)



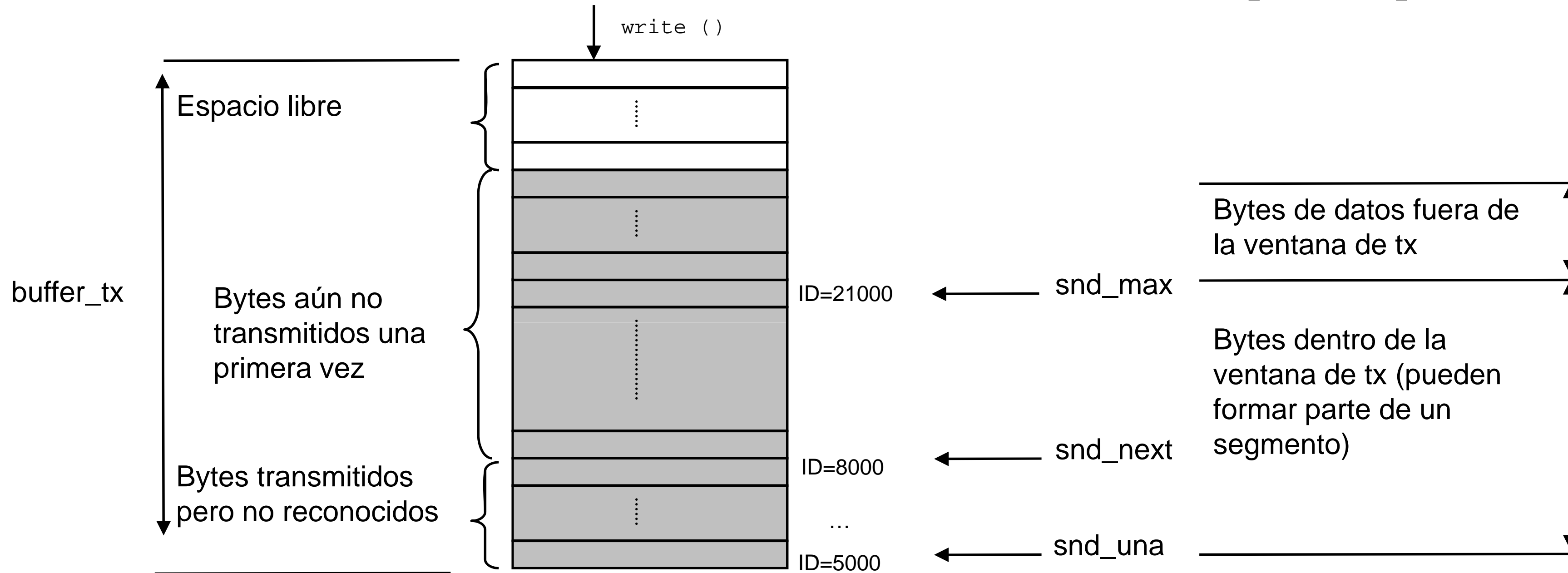
- Cada extremo de una conexión TCP mantiene dos bufferes para el almacenamiento de los datos:
- Buffer de recepción (buffer_rx):
  - Almacena los datos recibidos del otro extremo.
  - Los datos son eliminados del buffer cuando son leídos (`read ()`) por la aplicación.
  - Tamaño por defecto de buffer (en muchos SO): 256 Kbytes.
- Buffer de transmisión (buffer_tx):
  - Almacena los datos de aplicación procedentes de las llamadas `write ()`.
  - Los datos son eliminados del buffer, cuando se ha recibido asentimiento de su recepción.
  - Tamaño habitual de buffer: 256 Kbytes.

Ventana deslizante (II)



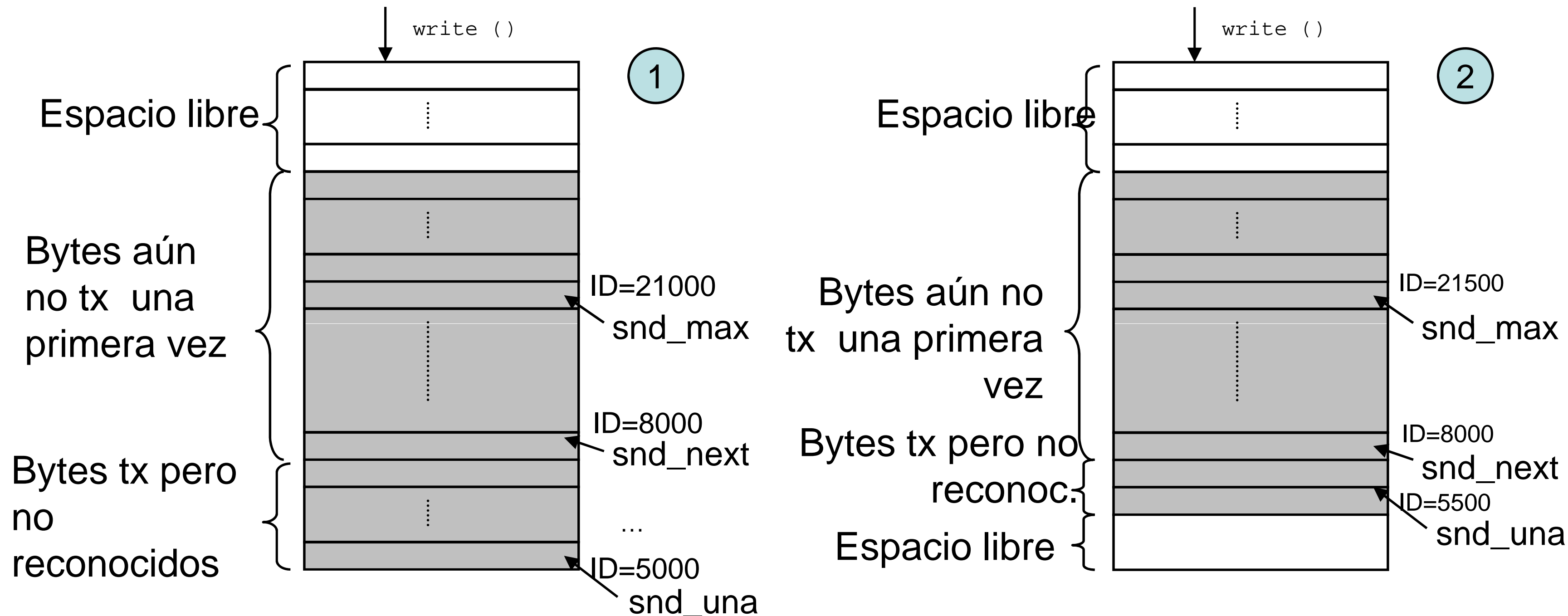
- **Definición** *ventana de transmisión de extremo A*: bytes del buffer_tx que pueden ser incluidos en segmento de datos (para transmisión o retransmisión).
- *snd_una*: posición de memoria donde se encuentra el primer byte que no ha sido asentido. Por tanto, marca el comienzo del buffer de transmisión y de la ventana de transmisión.
- *snd_next*: posición de memoria del primer byte de buffer_tx, que todavía no ha sido transmitido una primera vez. Salvo retransmisiones, indica el primer byte a incluir en el campo de datos del siguiente segmento con datos.
  - Todos los bytes dentro de la ventana de transmisión son enviados al otro extremo en sucesivos segmentos, sin espera de confirmación por el otro extremo (hasta que *snd_next* sea igual a *snd_max*).

Ventana deslizante (III)



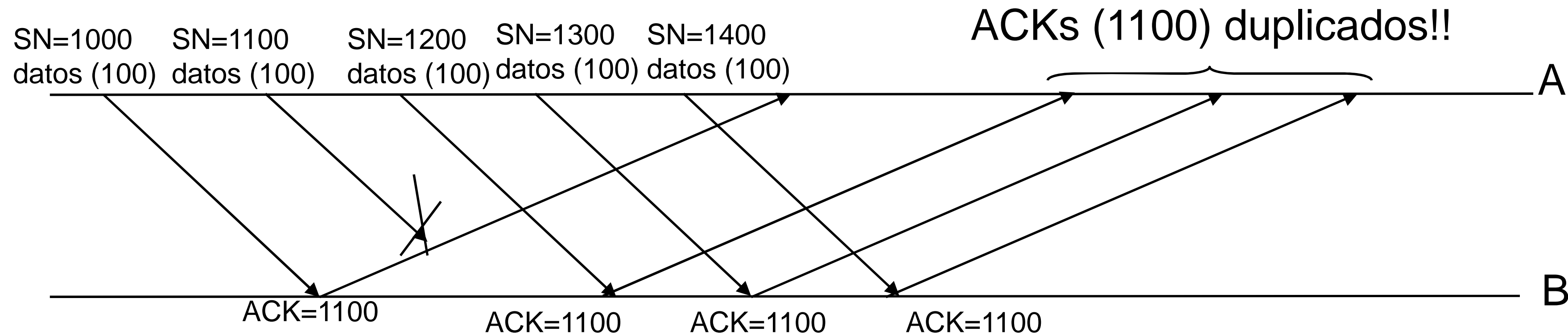
- *snd_wnd* = ventana advertida: tamaño máximo de ventana según el mecanismo de control de flujo.
- *snd_cwnd* = ventana de congestión: tamaño máximo de ventana según el mecanismo de control de congestión.
- *snd_max*: posición de memoria del primer byte *fuera* de la ventana de transmisión.
  - *snd_max = snd_una + [min (snd_wnd , snd_cwnd)]*
  - El tamaño de ventana de transmisión es el mínimo entre el permitido por control de flujo y por control de congestión.

Ventana deslizante (IV)



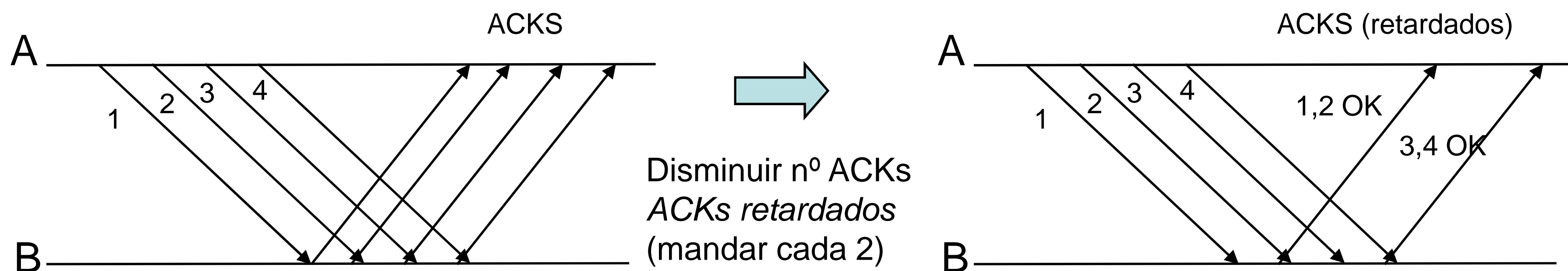
- Cuando A recibe un segmento con el flag ACK activado (lo habitual durante la fase de envío de datos), recalcula *snd_una* y *snd_max*.
  - snd_una = posición de memoria asociada a byte indicado por campo ACK, si supera el valor actual (en caso contrario, se descarta al tratarse de un segmento retrasado).
  - snd_max = snd_una + min (snd_wnd , snd_cwnd)
- Si el segmento asiente datos nuevos, *snd_una* (que marca el inicio de la ventana de transmisión) se incrementa => se dice que la ventana se "desliza".
- Ejemplo: si recibo segmento con ACK=5500 (asiente los bytes de ID = {5000...5499}) => *snd_una* se incrementa en 500 bytes => la ventana se "desliza" 500 bytes **(1)→(2)**.
- Nota: En las implementaciones reales, el buffer de tx y de rx se implementan como bufferes circulares.

Control de errores (I)



- Según el mecanismo de control de errores de TCP:
  - El ACK indica el ID del siguiente byte que se espera recibir => asiente todos los bytes de ID [ISN+1...ACK-1]. Un segmento es asentido, cuando se recibe un ACK con un ID mayor al último byte de datos del segmento.
  - El transmisor conoce cuál es el primer segmento perdido (Ejemplo: SN = 1100).
  - No sabe si los segmentos enviados después del 1º segmento perdido, han llegado correctamente o no (Ejemplo: qué sucede con los segmentos SN=1200,1300,1400?).
  - No conocerá esa información hasta que el 1º segmento llegue correctamente, y reciba el siguiente ACK (en nuestro ejemplo, recibiría ACK=1500, confirmando la llegada de todos los segmentos).

Control de errores (II)



- Envío de ACKs: Cada vez que se recibe un segmento:
  - Segmento incluye siguientes datos esperados, sin huecos en el buffer de recepción: mecanismo *ACK retardado (delayed ACKs)* = {se espera hasta 500 ms por el siguiente segmento, si no se recibe ningún segmento => se manda ACK}.
  - Segmento incluye siguientes datos esperados, sin huecos en el buffer de recepción, un ACK retardado pendiente: envío inmediato de ACK (que asiente ambos segmentos).
  - Segmento con datos nuevos, pero fuera de orden, con SN mayor del esperado, que genera un hueco en el buffer de recepción: enviar ACK (será ACK duplicado, que informará al transmisor de una situación anómala).
  - Segmento con datos que rellena un hueco en el buffer de recepción, que hace incrementar el ACK: enviar inmediatamente un ACK (no ACK retardado).

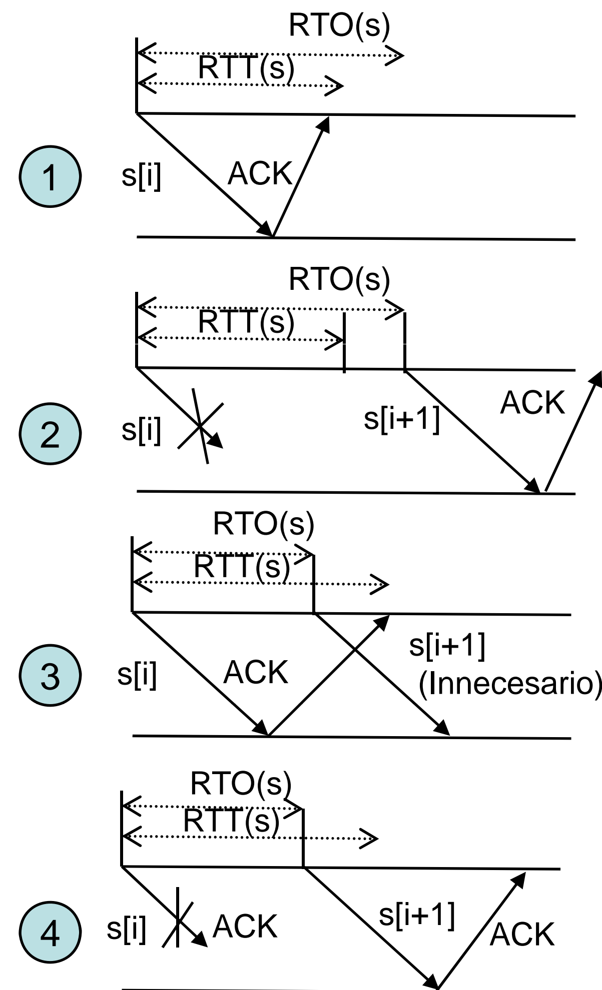
Nuevo

Nuevo

Nuevo

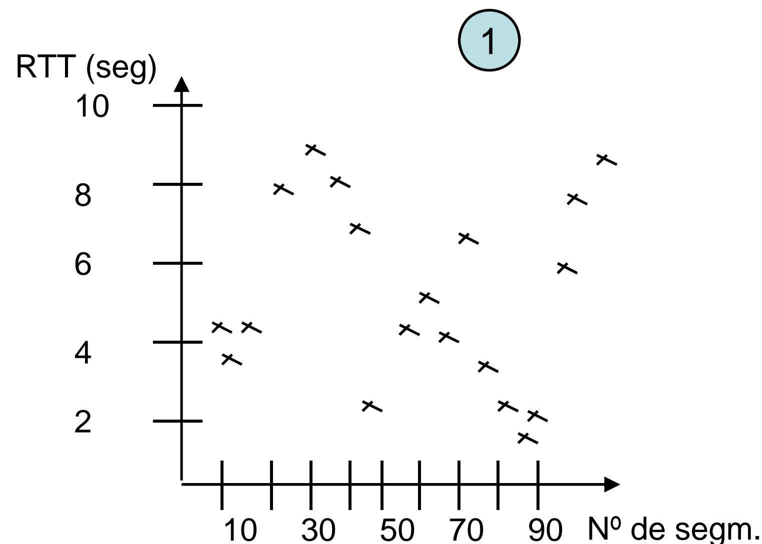
Control de errores (III)

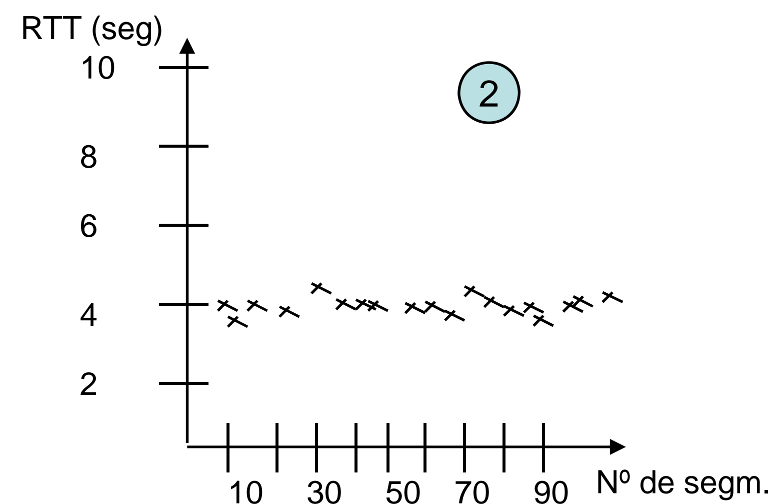
- Cuando se envía un segmento *s* con datos, por primera vez (s[1]), se inicia un temporizador:
  - Valor inicial del temporizador = RTO(s[1]) = (*Retransmission Time-Out*).
  - Si los bytes de datos enviados no son asentidos antes de que venza el temporizador, el segmento es retransmitido, y se lanza un nuevo temporizador de valor inicial RTO (s[2]).
  - ... RTOs[3] ...
  - ...
- ¿Qué valor se pone a los temporizadores?. El valor óptimo de un temporizador, tiene que ver con el tiempo de ida y vuelta entre los dos extremos de la conexión TCP (RTT, *Round Trip Time*)
  - Si RTO > RTT =>
    - (1) Si los datos llegan correctamente, no se envía retransmisión (correcto).
    - **(2) Si el segmento se ha perdido, la retransmisión comienza tarde.**
  - Si RTO < RTT =>
    - **(3) Si el segmento llega correctamente: se manda igualmente una retransmisión, porque el ACK no llega a tiempo.**
    - (4) Si el segmento se ha perdido, la retransmisión se realiza antes de tiempo (mejora?).
- Conclusión: El RTO de un segmento debe fijarse a un valor cercano, pero superior al RTT de la conexión.



Control de errores (IV)

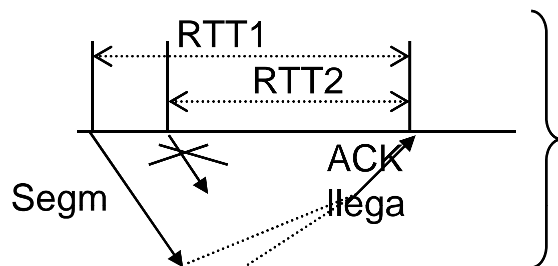
- ¿Qué valor se fija en el RTO de los segmentos a transmitir?
- **Solución 1**: Fijar un valor intermedio, igual en todas las conexiones TCP.
  - Los estudios indican: El RTT de distintas conexiones TCP en Internet puede variar desde menos un 1 ms hasta varios segundos!!! (4 órdenes de magnitud!!!).
  - Conclusión: Imposible fijar un valor que pueda ser válido para todas las conexiones TCP.
- **Solución 2**: El software TCP de cada conexión, debe estimar el RTT medio de la conexión, y usar esa estimación para calcular el RTO.
  - Los estudios indican: Hay conexiones TCP que tienen una gran variabilidad en el RTT de segmentos consecutivos **(1)**, y conexiones TCP donde los RTTs no se apartan mucho de la media **(2)**.
  - Conclusión: Algunas conexiones deben fijar un RTO superior y muy sobredimensionado sobre el RTT medido **(1)**, y en otras puede fijarse un RTO superior, pero muy cercano al RTT medio estimado **(2)**.
- **Solución 3** (EMPLEADA ACTUALMENTE): El software TCP de cada conexión, debe estimar el RTT medio, y estimar la desviación típica del RTT, y usar ambas para calcular el RTO.





Control de errores (V)

- Para estimar la media y la desviación típica del RTT, cada extremo debe obtener muestras del RTT.
- ¿Cómo obtengo muestras de RTT?
  - Puedo emplear todas las llegadas de ACK. Problema: cuando recibo un ACK de un segmento que ha sido retransmitido, se desconoce si el ACK corresponde al segmento original o a la retransmisión.



- Si se elige RTT1, y la realidad era que el ACK corresponde a la retransmisión => se elige un RTT mayor del real => RTO sobredimensionado.
- Si se elige RTT2, y la realidad era que el ACK corresponde al segmento original => se elige un RTT menor del real => RTO demasiado bajo.

- Solución: *Algoritmo de Karn*: Los ACKs de segmentos que se han retransmitido alguna vez, no se emplean para calcular nuevas muestras de RTT.

Control de errores (VI)

- Problema de aplicación del Algoritmo de Karn: respuesta ante incrementos bruscos del RTT de la red.
  - En una red rápida, el RTT calculado es pequeño. El RTO de los segmentos que se transmite también es pequeño.
  - Si de manera brusca, los segmentos sufren un mayor retraso, hasta que RTT > RTO => los segmentos son retransmitidos.
  - Lo ideal sería que paulatinamente, el RTO fuera creciendo para ajustarse a la nueva situación de la red. Sin embargo el algoritmo de Karn provoca que las nuevas muestras de RTT no sean empleadas, con lo que la situación de retransmisiones innecesarias se mantiene.

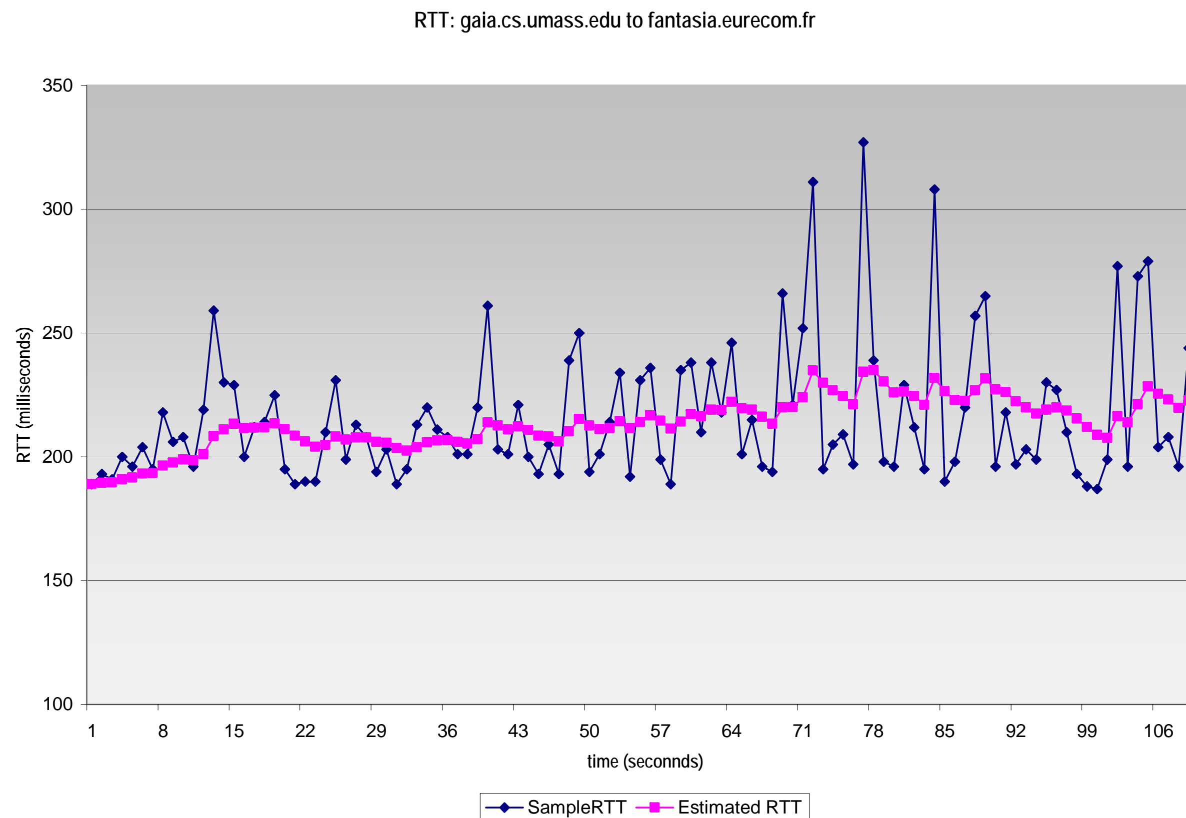
- **Solución:** ***Exponential backoff***: El RTO de un segmento retransmitido, es el doble del RTO de su transmisión anterior.
  - RTO (s[i+1]) = 2 x RTO (s[i])

Control de errores (VII)

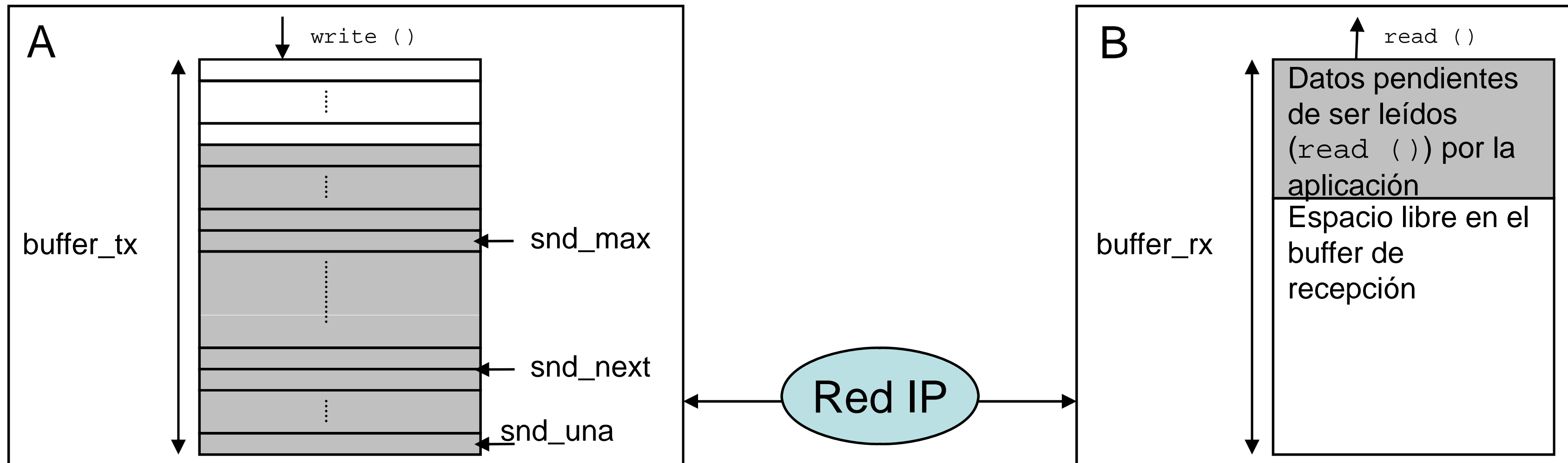
- Implementación del método de cálculo del RTO para cada segmento transmitido:
  - Cada vez que se recibe un ACK que asiente un segmento *s* que *no ha sido retransmitido* (algoritmo de Karn):
    - RTT_muestra = tiempo llegada ACK – tiempo transmisión segmento asentido.
    - err = RTT_muestra – rtt_medio
      - NOTA: rtt_medio = estimación de RTT medio proveniente de muestras anteriores.
    - rtt_medio = rtt_medio + g * err
      - NOTA: Valor de g empleado habitualmente: g=1/8
    - rtt_σ = rtt_σ + h * (|err| - rtt_σ)
      - NOTA: Valor de h empleado habitualmente: h=1/4
  - Cada vez que se transmite un segmento de datos s[i].
    - Si i = 1 (primera vez se transmite este segmento)
      - => RTO (s[i]) = a·rtt_medio + b· rtt_σ
      - NOTA: Habitualmente a=1, b=4.
    - Si i>1 (segmento retransmitido)
      - => RTO (s[i]) = 2 x RTO (s[i-1]) [*exponential backoff*]

Control de errores (VIII)

- El mecanismo de estimación del RTT medio, promedia las variaciones rápidas del RTT, y estima su desviación típica.



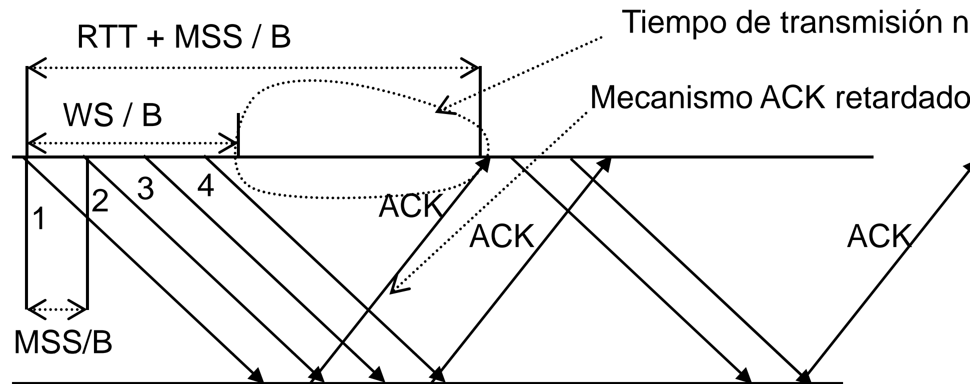
Control de flujo (I)



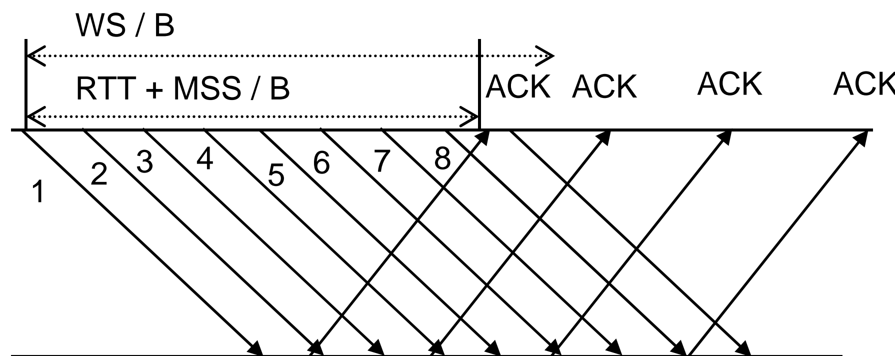
- Control de flujo: El campo *window size* es empleado por A para anunciar a B el tamaño máximo de ventana de transmisión que B debe tener. De esta manera limita el volumen de datos que B mandará sin que A envíe asentimientos.
  - Transmisión de un segmento: Se incluye en el campo *window size* el espacio libre en el buffer de recepción.
  - Recepción de un segmento:
    - *snd_wnd* = campo *window size* de segmento recibido.
    - se recalcula *snd_max = snd_una + min (snd_wnd , snd_cwnd).*
    - Si disminuye la ventana de transmisión, puede suceder que bytes de datos ya transmitidos, se queden ahora fuera de la ventana. En caso de que tengan que ser retransmitidos, tendrán que esperar a encontrarse dentro de la ventana de nuevo.

Control de flujo (II)

- ¿Cuál sería el tamaño mínimo de una ventana de transmisión, en un escenario ideal, para no desaprovechar capacidad de transmisión?
  - Suponiendo un transmisor con datos infinitos para transmitir, ancho de banda de transmisión B bps.
  - Segmentos de tamaño MSS. El receptor no se satura.
  - Red sin pérdidas, y con RTT fijo.



Un tamaño de ventana de transmisión pequeño, implica un desaprovechamiento del tiempo de transmisión.



Factor debido al mecanismo *delayed ACK*

- WS/ B = Tiempo en vaciarse ventana de transmisión
- Condición: WS / B > RTT + MSS / B =>

WS > RTT x B + MSS

- Se conoce como condición *bandwidth x delay product*

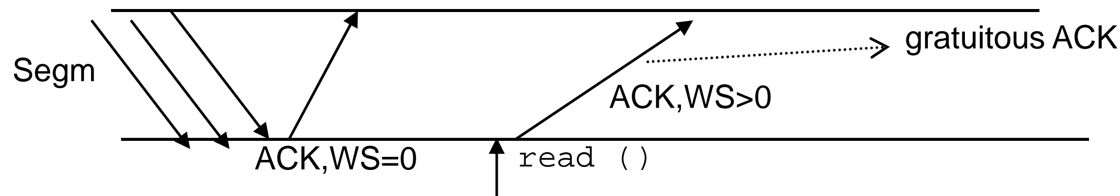
Control de flujo (III)

- El tamaño máximo de la ventana advertida por control de flujo es 64 KB (campo *Window Size* de 16 bits) => el tamaño máximo de la ventana de transmisión es 64 KB!
  - *TamVentanaTx = min (snd_wnd,snd_cwnd)*
- Problema:
  - Actualmente, en conexiones TCP sobre redes de alta capacidad, este valor puede ser inferior al producto *bandwidth-delay*.
  - Ejemplo: La siguiente tabla muestra los valores RTTxB para una conexión de RTT 100 ms.
  - Incluso para una Ethernet a 10 Mbps, la ventana de transmisión no es suficientemente para saturar la conexión!!!!
- Solución (extensión reciente de TCP, RFC 1323):
  - Se define la opción *window-scale-factor,* negociada durante el establecimiento 3-WHS. Indica una potencia de 2 por la que multiplicar el valor del campo *Window Size* en los subsiguientes segmentos TCP.

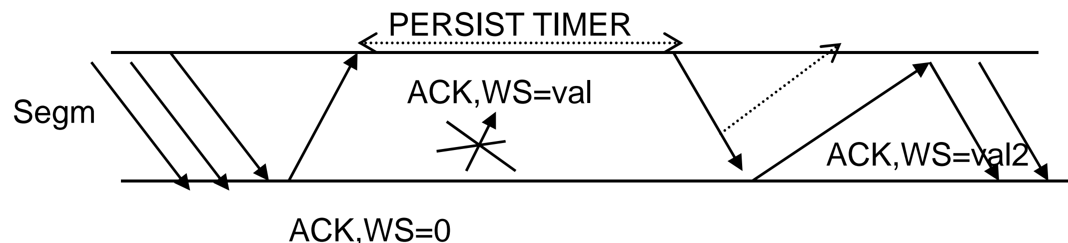
B	RTT x B (RTT=100ms)
E1 (2Mbps)	24,4 KB
Ethernet (10Mbps)	122 KB
Ethernet (100Mbps)	1.2 MB
SDH STM-1 (155Mbps)	1.8 MB
SDH STM-4 (622Mbps)	7.4 MB
SDH STM-8 (2.5Gbps)	29.6 MB

Control de flujo (IV)

- **Def: Cierre de ventana:** Cuando un extremo envía un segmento con WS=0, se dice que cierra la ventana del extremo opuesto.
- **Del lado del receptor:** Cuando una aplicación lee datos, que provocan nuevo espacio libre en el buffer de recepción, que permiten "abrir" una ventana cerrada => el extremo receptor envía un segmento *gratuitous acknowledgement,* con el nuevo valor WS, que abre la ventana.

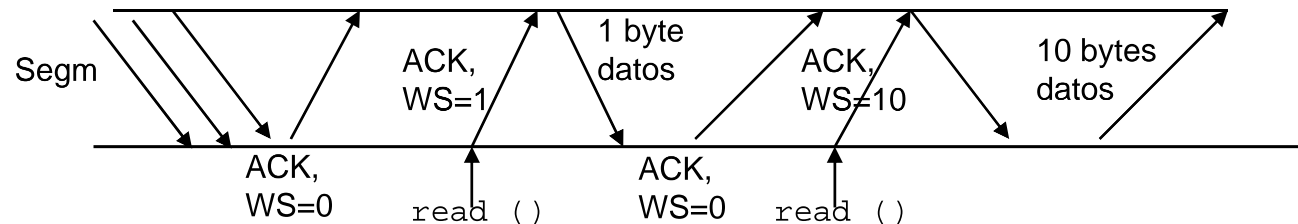


- Problema: Si se pierde el ACK que abre la ventana?
  - El transmisor no enviaría nada.
  - El receptor pensaría que el transmisor no tiene nada que enviar.
- **Solución:** (Del lado del transmisor) Un transmisor con una ventana cerrada, envía periódicamente (según el PERSIST TIMER, de 5 a 60 segundos habitualmente) segmentos de prueba (*window proves*). El receptor responde con un valor WS actualizado.



Control de flujo (V)

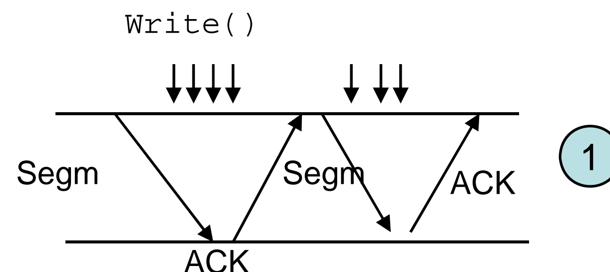
- Síndrome de la ventana tonta (*silly window syndrome,* SWS, RFC 813)
  - Aplicación transmisora rápida. Aplicación receptora más lenta.
    - Receptor cierra ventana del extremo transmisor.
    - Las lecturas de la aplicación, provocan la apertura de la ventana con tamaños pequeños => se generan segmentos de datos de pequeño tamaño (muy ineficiente).



- Solución al síndrome de la ventana tonta del lado del receptor:
  - Tras cerrar una ventana, el receptor no debe anunciar una ventana distinta de 0 hasta que no pueda anunciar una ventana de tamaño suficiente.
  - Habitualmente, el tamaño "suficiente" se fija a T=min(MSS,buffer_rx / 2)

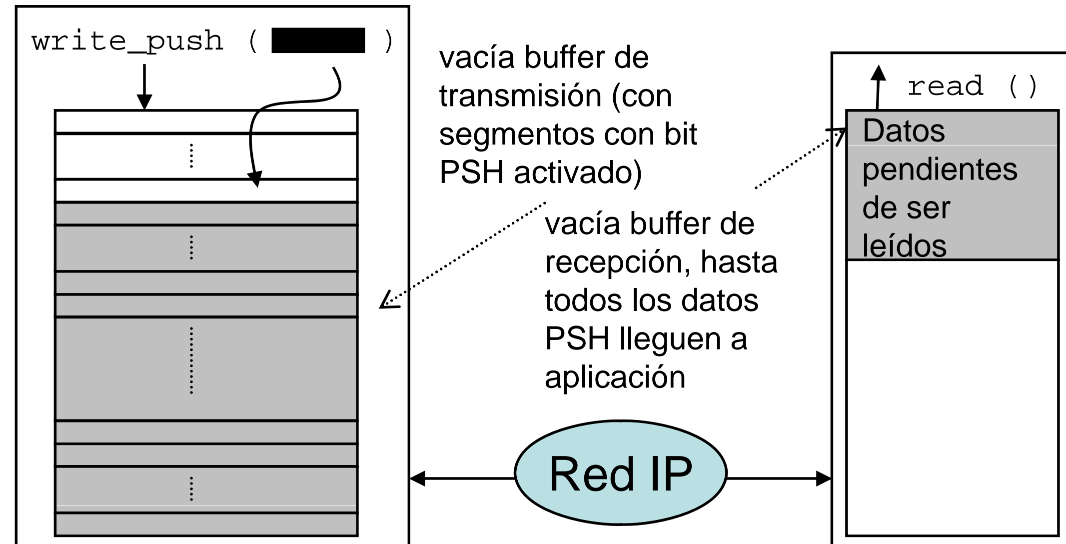
Control de flujo (VI)

- Solución al síndrome de la ventana tonta del lado del transmisor (algoritmo de Nagle, RFC 896 [John Nagle])
  - Método eficiente para evitar la transmisión de segmentos demasiado cortos.
- Algoritmo de Nagle:
  - Si no existen datos esperando ser asentidos => las peticiones `write ()`, independientemente del volumen de datos, pueden ser enviadas (p.e. en sesiones Telnet, las pulsaciones de teclas suelen cumplir esta condición).
  - Si existen datos esperando ser asentidos, todos los datos se mantienen en el buffer de transmisión hasta que suceda:
    - 1) Que todos los datos no asentidos, sean asentidos, o bien
    - 2) haya suficientes datos para ser transmitidos para llenar un segmento (MSS).
    - NOTA: Esto se aplica incluso aunque se solicite la operación PUSH.



Bit PSH: envío inmediato

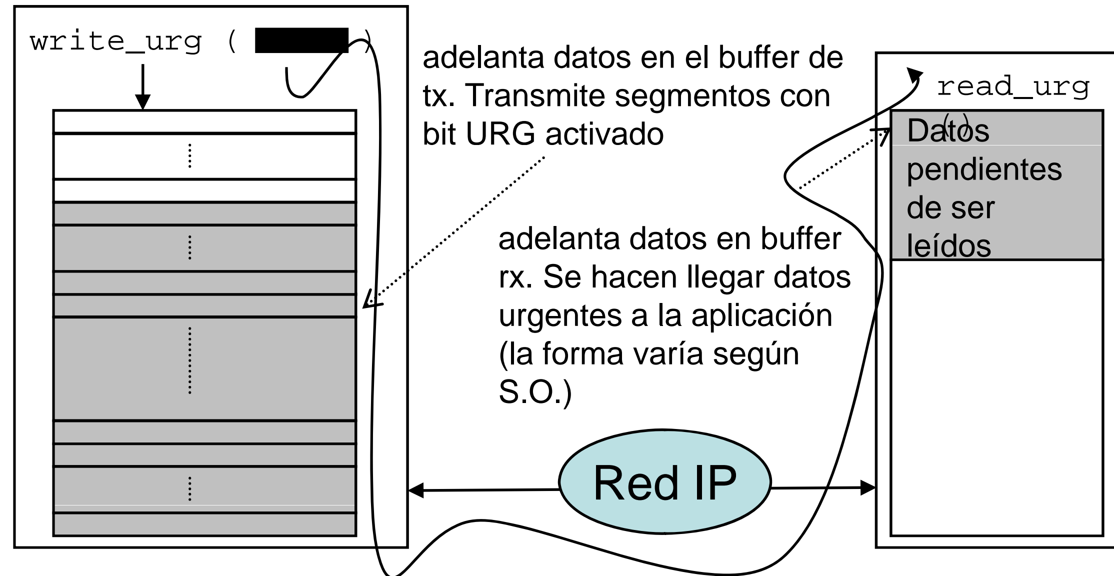
- Aplicación invoca transmisión de datos tipo `push ()`. Forzar el envío de datos, manteniendo la secuencia.



- Transmisor:
  - Los datos se integran en el buffer de transmisión.
  - El transmisor intenta vaciar el buffer de transmisión, enviando los datos del buffer en segmentos con el bit PSH activado (**siempre respetando lo marcado por la ventana de transmisión**).
- Receptor:
  - Los segmentos con el bit PSH activado se integran en el buffer de recepción.
  - Se hacen llegar a la aplicación todos los datos del buffer de recepción marcados con PSH, y los anteriores en la secuencia (ordenadamente).

Bit URG: datos fuera de banda

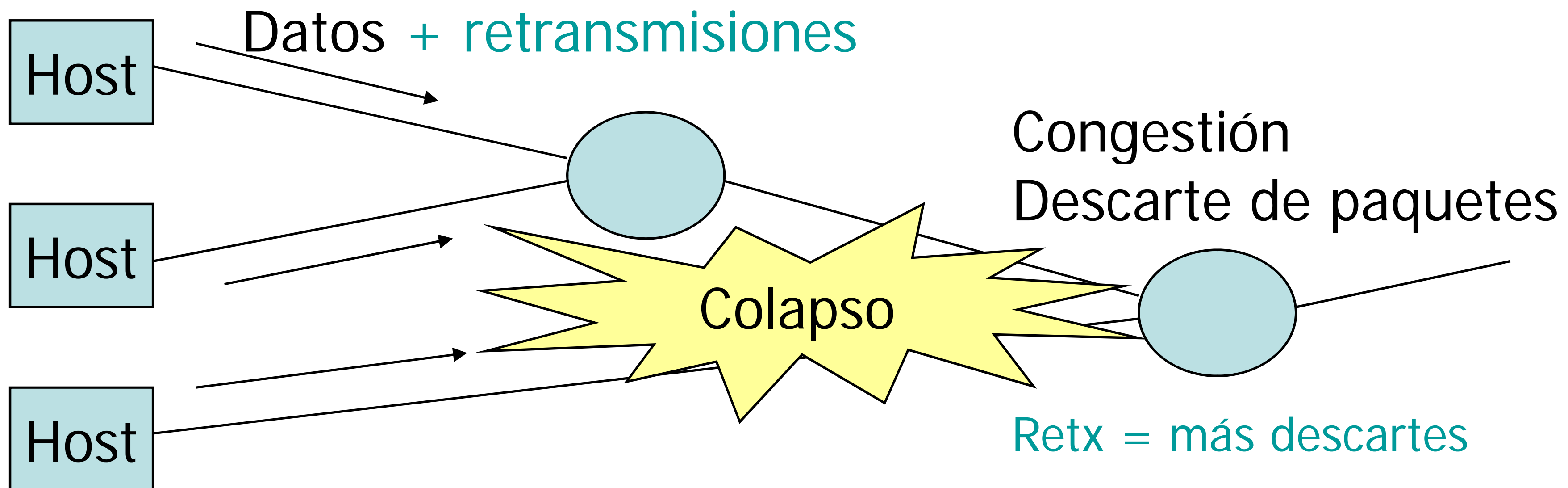
- Datos fuera de banda: datos no procesados dentro del buffer de transmisión o recepción. Son entregados a la aplicación destino separadamente.



- Transmisor:
  - Genera segmento, con los datos urgentes al comienzo de la parte de datos. En caso de que el mismo segmento envíe datos normales, el puntero URGENT POINTER apunta al final de los datos urgentes dentro del segmento. El segmento es enviado con el bit URG activado.
- Receptor (recibe segmento con bit URG = 1):
  - Los datos urgentes son enviados a la aplicación separadamente, adelantando a los datos del buffer de recepción.
  - El mecanismo concreto de comunicación con la aplicación receptora, depende del S.O. (p.e. llamada asíncrona).

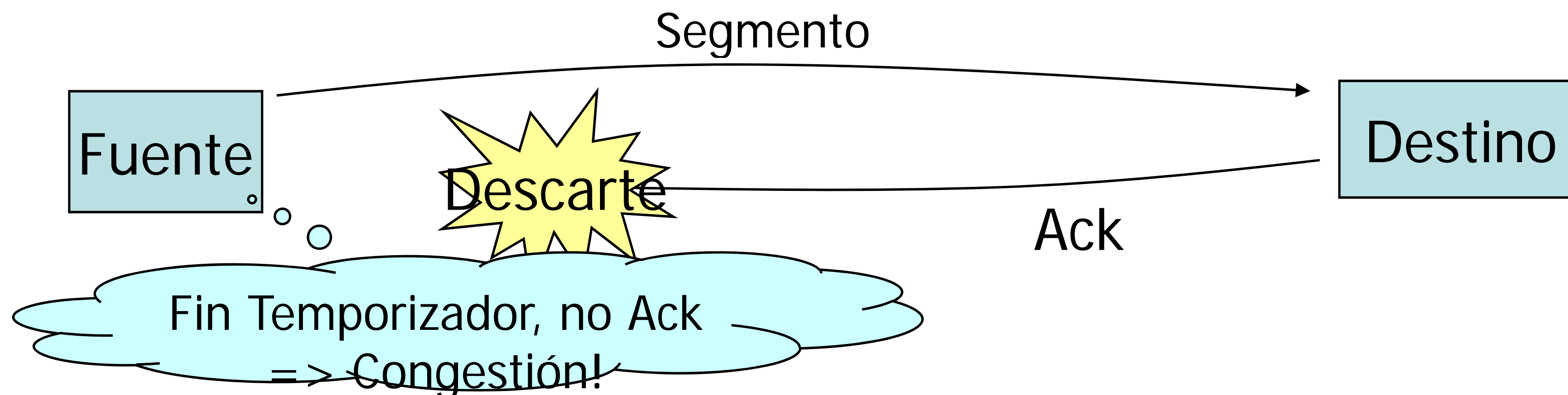
Control de congestión (I)

- La congestión es una situación que afecta a la red. TCP desconoce los detalles de la posible situación de congestión. El congestionamiento se muestra como un aumento del retraso en la llegada de ACKs (o no llegada de los mismos).
- Si en este caso reacciona según el mecanismo de retransmisión habitual, la situación de congestión probablemente empeore, pudiendo llegar al *colapso por congestionamiento.* Esta situación era habitual en Internet durante la década de los 80.



Control de congestión (II)

- Formas de manejar la congestión:
  - Algoritmos de encaminamiento dinámico (útiles si parte del tráfico se puede reencaminar, para evitar los *routers* en congestión).
  - Reserva de recursos (No!!).
  - Actuando sobre las fuentes de tráfico.
- Control de congestión en el protocolo TCP.
  - Los algoritmos de control de congestión son la razón de que la red Internet sea operativa hoy en día, a pesar de lo estadísticamente impredecible del tráfico generado por los usuarios.
  - La detección de congestión se basa en suponer que los siguientes hechos:
    - El aumento del RTT de los segmentos asentidos, respecto a anteriores segmentos.
    - La finalización de temporizadores de retransmisión.
  - ... son causados por congestión en la red.

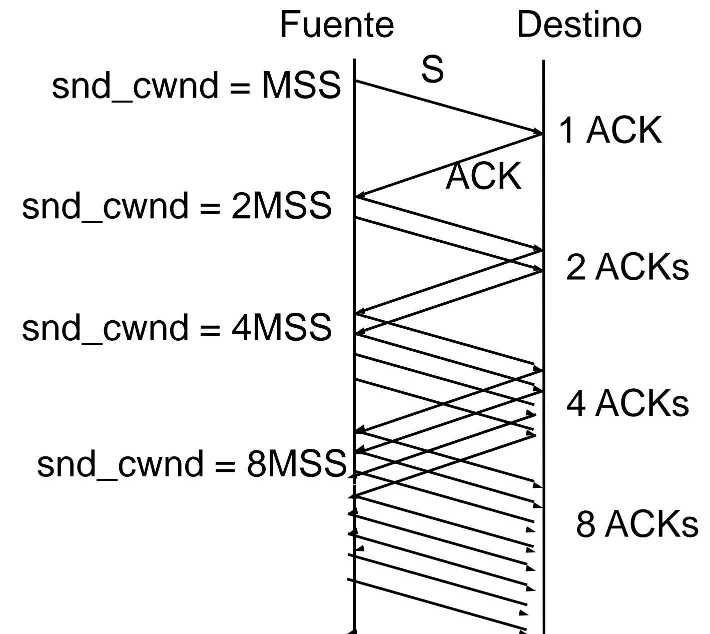


Control de congestión (III)

- Mecanismos de control de congestión implementados: Continúan mejorándose. Existen variadas implementaciones.
  - *Slow-start (SS) y congestion avoidance (CA)* (obligatorios)
  - TCP Tahoe (1988), SS, CA, *fast-retransmit*.
  - TCP Reno (1990), Tahoe + *fast recovery* + TCP *header prediction*.
  - TCP Sack: Reno + *selective* ACK.
- Los mecanismos TCP disminuyen el flujo de datos transmitido, en caso de posible congestión en la red. Lo hacen mediante dos mecanismos:
  - Actuando sobre *snd_cwnd*: *TamVentanaTx = min (snd_wnd , snd_cwnd*)
  - *Exponential backoff (*ya visto*)*: El RTO de un segmento retransmitido, es el doble del RTO de su transmisión anterior.
    - RTO (s[i+1]) = 2 x RTO (s[i])
- Veremos únicamente los mecanismos de *Slow-Start*, y *Congestion Avoidance* y *Fast Retransmit* correspondientes a la implementación TCP Tahoe.

Control de congestión TCP Tahoe (I)

- *Slow-Start*
  - Inicialmente, snd_cwnd = MSS.
  - Cada vez que recibe un ACK
    - snd_cwnd = snd_cwnd + MSS.
- Ejemplo. Suposiciones:
  - Transmisor tiene siempre datos para transmitir (segmentos de tamaño MSS).
  - Tamaño de ventana limitado por *snd_cwnd* (no por control de flujo)
  - Situación de congestión => segmentos llegan separados en tiempo al receptor => no se aplica el mecanismos *delayed ACK* => llega un ACK por cada segmento de datos.



- Solo SS => el crecimiento de la ventana es exponencial!! (MSS, 2xMSS, 4xMSS, 8xMSS, 16xMSS, 32xMSS...) => demasiado rápido => puede volver a entrar en congestión!

Control de congestión TCP Tahoe (II)

- Congestion Avoidance:
  - *Slow-start:* el crecimiento de la ventana no tiene nada de "*slow*".
  - *Congestion avoidance* intenta suavizar el crecimiento de *snd_cwnd* cuando nos recuperamos de una congestión.
  - Se define el parámetro *sshthresh* (inicialmente igual a 64 KB).
- SS + CA:
  - Si hay congestión debido a fin RTO, o a la recepción de un ACK duplicado
    - => sshthresh = max [2xMSS , (min (snd_wnd , snd_cwnd)) / 2]
    - Además, si la congestión es por fin de RTO => snd_cwnd = MSS.
  - Si snd_cwnd <= sshthresh => entramos en *SS*
    - Para cada ACK recibido: snd_cwnd += MSS.
  - Si snd_cwnd > sshthreah => entramos en CA
    - Para cada ACK recibido : snd_cwnd += MSS^2 / snd_cwnd.

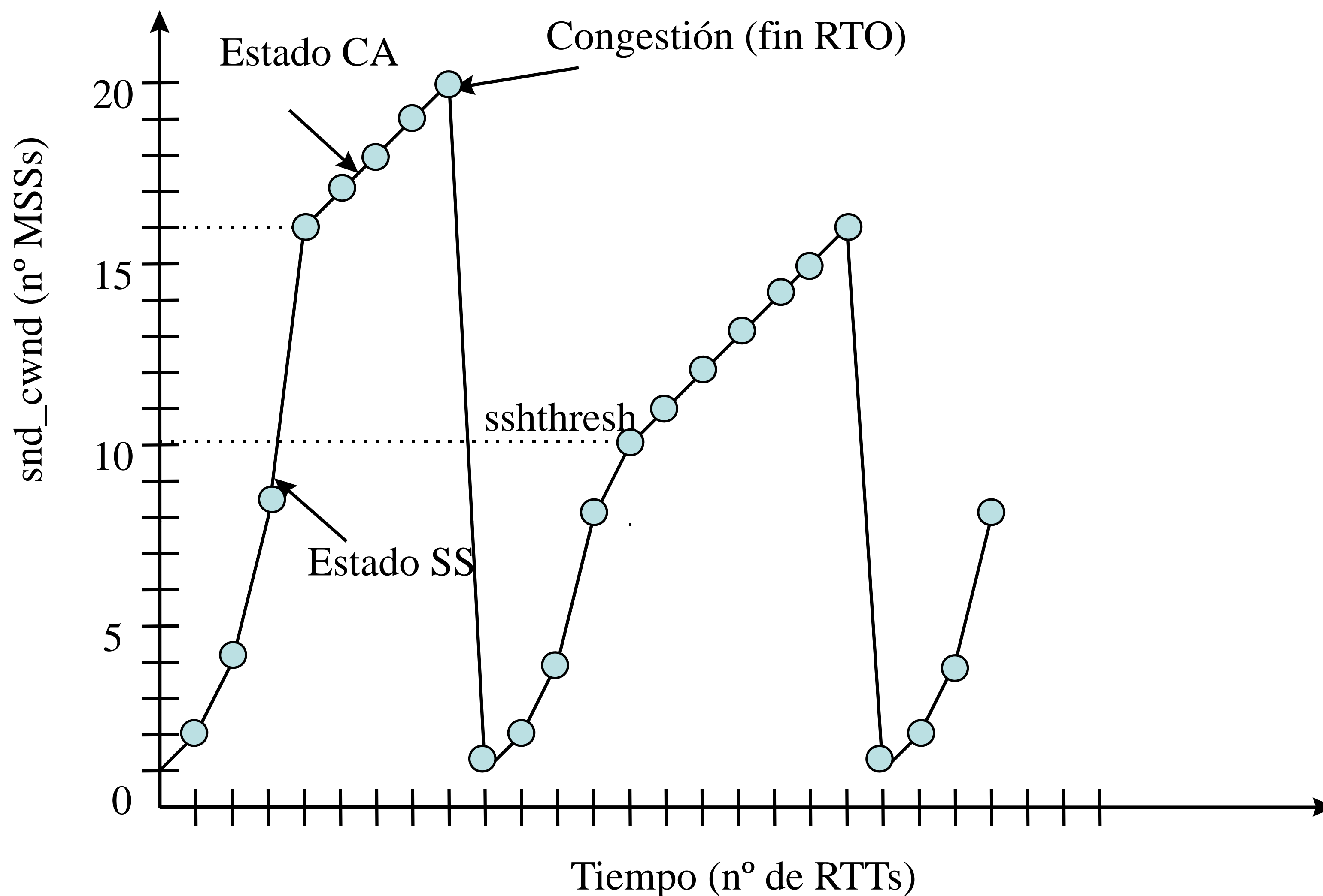
Control de congestión TCP Tahoe (III)

- Habitualmente, los temporizadores RTO son muy largos => se tarda tiempo en realizar retransmisiones.
- La pérdida de un segmento, se detecta usualmente mediante la llegada de ACKs duplicados.
- *Fast-Retransmit*:
  - Cuando se reciban 3 ACKs duplicados consecutivos, se retransmite el segmento que se sabe se ha perdido, aunque no haya finalizado su RTO.

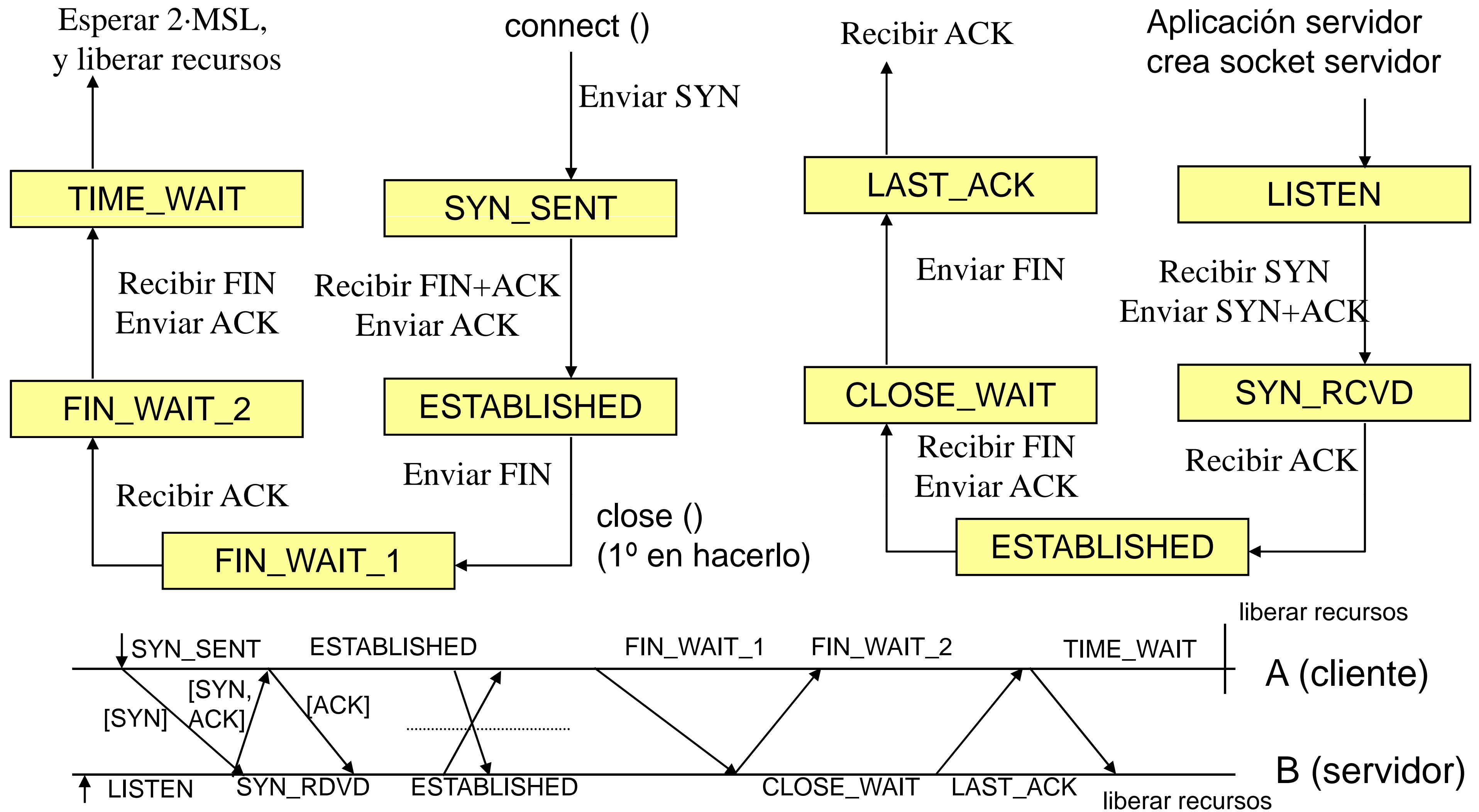
Control de congestión TCP Tahoe (IV)

Evento	Estado inicial	Acción de tx TCP	Comentario
Recibido ACK de datos no asentidos previamente	SS	snd_cwnd += MSS if (snd_cwnd > sshthresh) entrar en estado CA	snd_cwnd se doblará en cada RTT aprox.
Recibido ACK de datos no asentidos previamente	CA	snd_cwnd += MSS^2/ snd_cwnd	Incremento lineal de snd_cwnd (1 MSS cada RTT aprox.)
Pérdida detectada por 3 ACKs duplicados	SS o CA	Enviar segmento de ACK duplicado, aunque no haya saltado RTO	Fast retransmit
RTO finalizado	SS o CA	sshthresh = max [2xMSS , (min (snd_wnd , snd_cwnd)) / 2], snd_cwnd = 1 MSS, Entrar en estado SS	Entrar en SS

Ejemplo (SS-CA)



Estados conexión TCP (I)



Estados conexión TCP (II)

- TCP se describe más fácilmente como una máquina de estados.
- Estado TIME_WAIT:
  - Cuando una conexión TCP ha sido terminada (el último ACK ha sido enviado), quedan cosas pendientes:
    - ¿Qué sucede si el ACK se pierde? El último FIN sería reenviado, y debería ser asentido.
    - ¿Qué sucede si existen segmentos duplicados, que alcanzan el destino después de que la conexión se cierre? Si otra aplicación ha abierto una conexión que utiliza el mismo puerto => podrían generarse conflictos.
  - Solución: después de cerrar una conexión, antes de liberar los recursos (memoria, puerto TCP) se espera un tiempo igual a 2 veces el tiempo máximo de vida de un segmento en la red (MSL = *Maximum Segment Life*, habitualmente se fija a 120 segundos).
    - El único segmento que puede llegar en el estado TIME-WAIT es una retransmisión de segmento FIN.
    - Otras aplicaciones no pueden ser asignadas ese puerto TCP para otra conexión hasta después de 240 segundos.

Bibliografía recomendada

- Douglas E. Comer, “Internetworking with TCP/IP Vol. 1: Principles, Protocols, and Architecture. 5th Edition”, Prentice Hall 2006:
  - Capítulo 12: “Reliable Stream Transport Service (TCP)”
- RFCs:
  - 793: Especificación inicial del protocolo (1981).
  - 813: Administración de ventanas (1982).
  - 879: Maximum Segment Size (1983).
  - 1122: Aclara varios puntos (1989).
  - 896, 2001, 2581: Control de congestión.
  - Otros: 1323, ...
- Estadísticas fragmentación: http://www.caida.org/outreach/papers/2001/Frag-IMW/Frag-IMW.pdf