Bloque III: El nivel de transporte

Tema 8: Retransmisiones y temporizadores en TCP

Índice

- Bloque III: El nivel de transporte
  - Tema 8: Retransmisiones y temporizadores en TCP
    - Retransmisiones
      - Control de congestión
      - Algoritmo para evitar la congestión
      - Algoritmo de recuperación y retransmisión rápida
    - Temporizadores
      - Temporizador de persistencia
      - Temporizador de keepalive

- **Referencias**
  - Capítulo 3 de “Redes de Computadores: Un enfoque descendente basdado en Internet”. James F. Kurose, Keith W. Ross. Addison Wesley, 2ª edición. 2003.
  - Capítulos 21, 22 y 23 de “TCP/IP Illustrated, Volume 1: The Protocols”, W. Richard Stevens, Addison Wesley, 1994.

Retransmisiones en TCP

- TCP proporciona un servicio fiable sobre un protocolo no fiable (IP) ➔ Se pueden perder datos y/o ACKs.
- Solución ➔ Esperar un tiempo la llegada del ACK (utilizando un temporizador) y si no se recibe, se retransmite el segmento.
- Problema: el RTT (Round Trip Time) es variable
  - ¿Cómo se determina el intervalo de timeout?
  - ¿Con qué frecuencia se producen retransmisiones?
    - **Exponential backoff ➔** Si no se recibe el ACK la primera retransmisión ocurrirá entre 1 y 1’5 segundos (ticks 500 milisegundos).
    - Las siguientes retransmisiones se duplica el tiempo: 3, 6, 12, 24, 48 y a partir de ahí 64 segundos.
    - Se finalizan las retransmisiones después de N intentos (normalmente, entre 5-10 minutos).

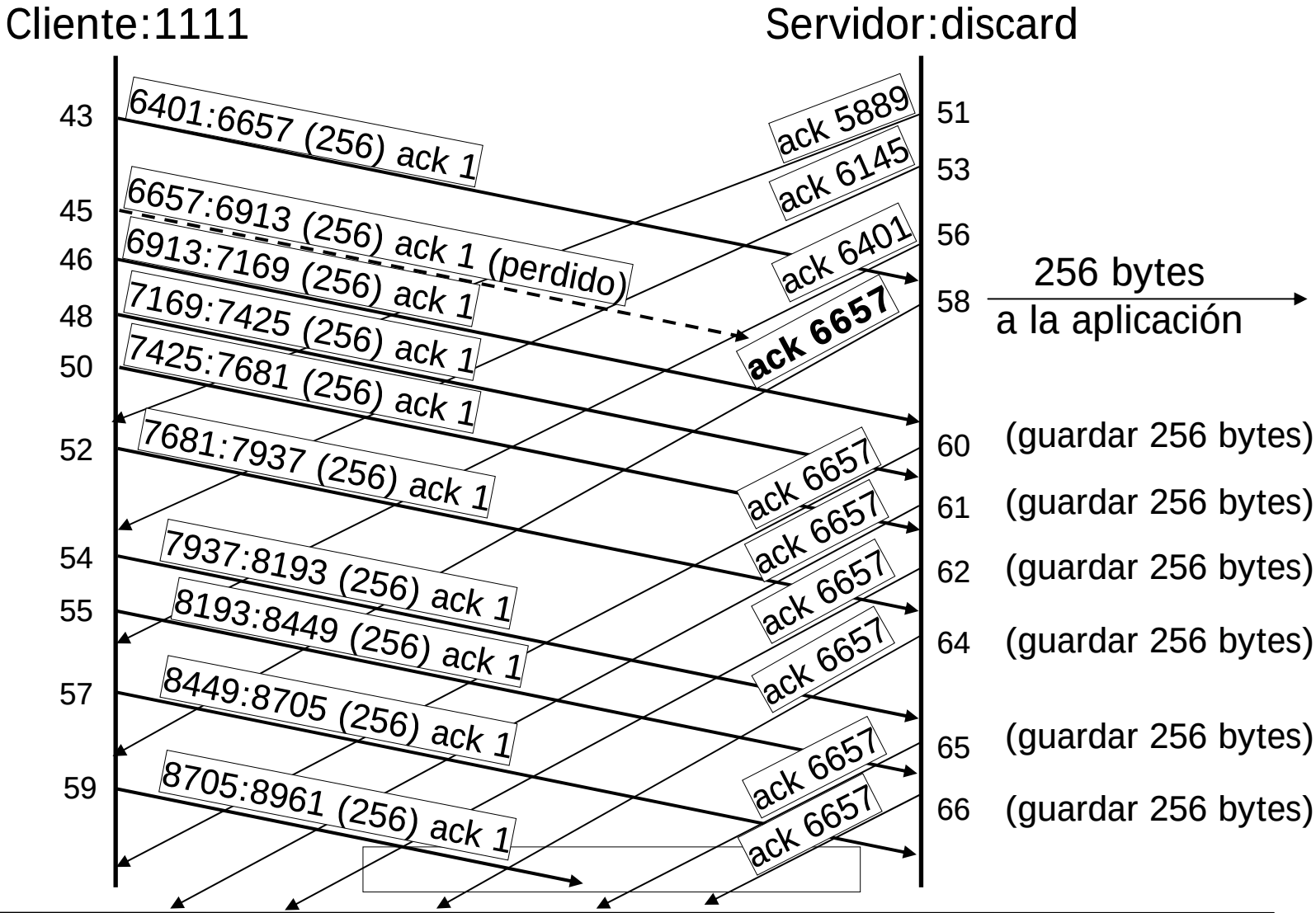
Estimación del RTT

- Resulta fundamental para la estrategia de timeout y retransmisiones de TCP el medir el RTT para la conexión, para así variar el timeout.
  - Problema: no hay una asignación uno-a-uno entre segmento y asentimiento
    - Sólo se mide el RTT para un segmento simultáneamente.
    - Su confirmación puede venir en un ACK acumulado ➔ estimación ligeramente superior
  - Se utiliza un estimador a partir de la media y la desviación típica de los retardos medidos (**estimador de Jacobson**).
    - La estimación se basa en ticks de reloj (p.e. 500 milisegundos) y no en tiempos absolutos.
  - Problema de ambigüedad en la retransmisión:
    - Al producirse una retransmisión (por timeout), es imposible saber si el ACK recibido es el correspondiente al segmento original o al retransmitido.
    - Solución ➔ Algoritmo de Karn:
    - Ante el problema de ambigüedad en la retransmisión NO deben usarse medidas de segmentos retransmitidos para estos cálculos.

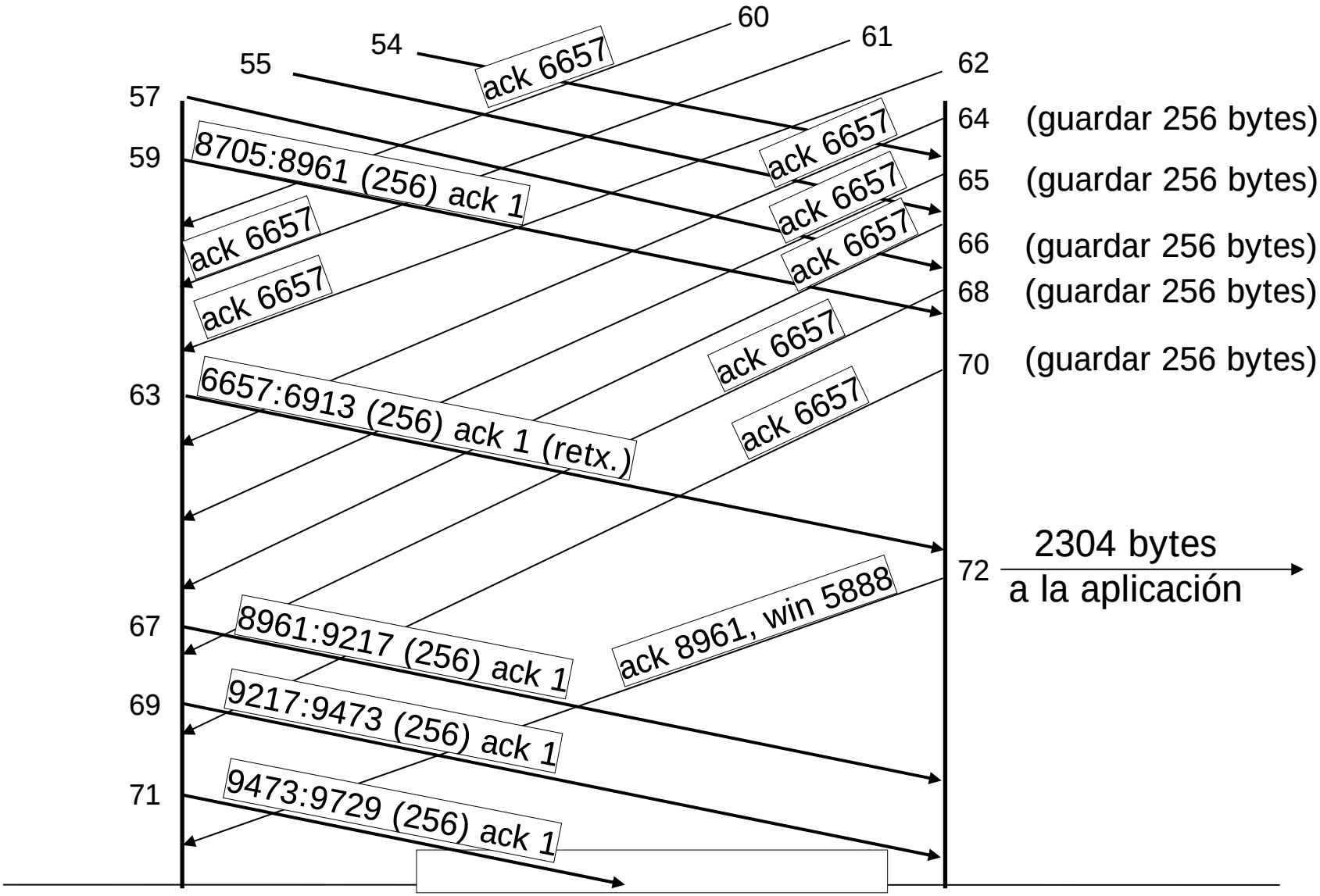
Control de congestión

- Dos posibilidades para “perder” paquete:
  - Saturación en un router
  - Error en el paquete (probabilidad mucho menor del 1%)
- Se asume que cuando se cuando se pierde un paquete es debido a que al menos un router saturado.
- Se utilizan dos indicadores para identificar un problema de congestión (pérdida de un paquete):
  - Ha vencido un timeout de retransmisión
  - Se han recibido ACKs duplicados

Control de congestión: Ejemplo



Control de congestión: Ejemplo



Congestión: Ejemplo

- Cuando el receptor recibe un segmento con mayor número de secuencia ➔ transmite el ACK del byte que espera recibir.
  - Almacena los datos en espera de poder entregarlos en orden.
- Cuando el transmisor recibe el tercer ACK duplicado (1 ó 2 se admiten como posible desorden causado por la red) sobre el mismo número de secuencia supone que se ha perdido sólo ese segmento.
  - Retransmite el segmento perdido y continúa la transmisión ➔ Algoritmo de recuperación y retransmisión rápida
- El receptor, al recibir el segmento perdido puede reordenar los segmentos, entregar todos los datos recibidos al usuario y asentirlos al transmisor.

Algoritmo para evitar la congestión

- Es un control de flujo que se impone el propio transmisor para evitar la congestión, frente a la ventana anunciada por el receptor para evitar la saturación del mismo.
- Se implementa conjuntamente con el algoritmo de inicio lento.
- Utiliza dos variables en bytes:
  - cwnd: ventana de congestión
  - ssthres: umbral de inicio lento

Algoritmo para evitar la congestión

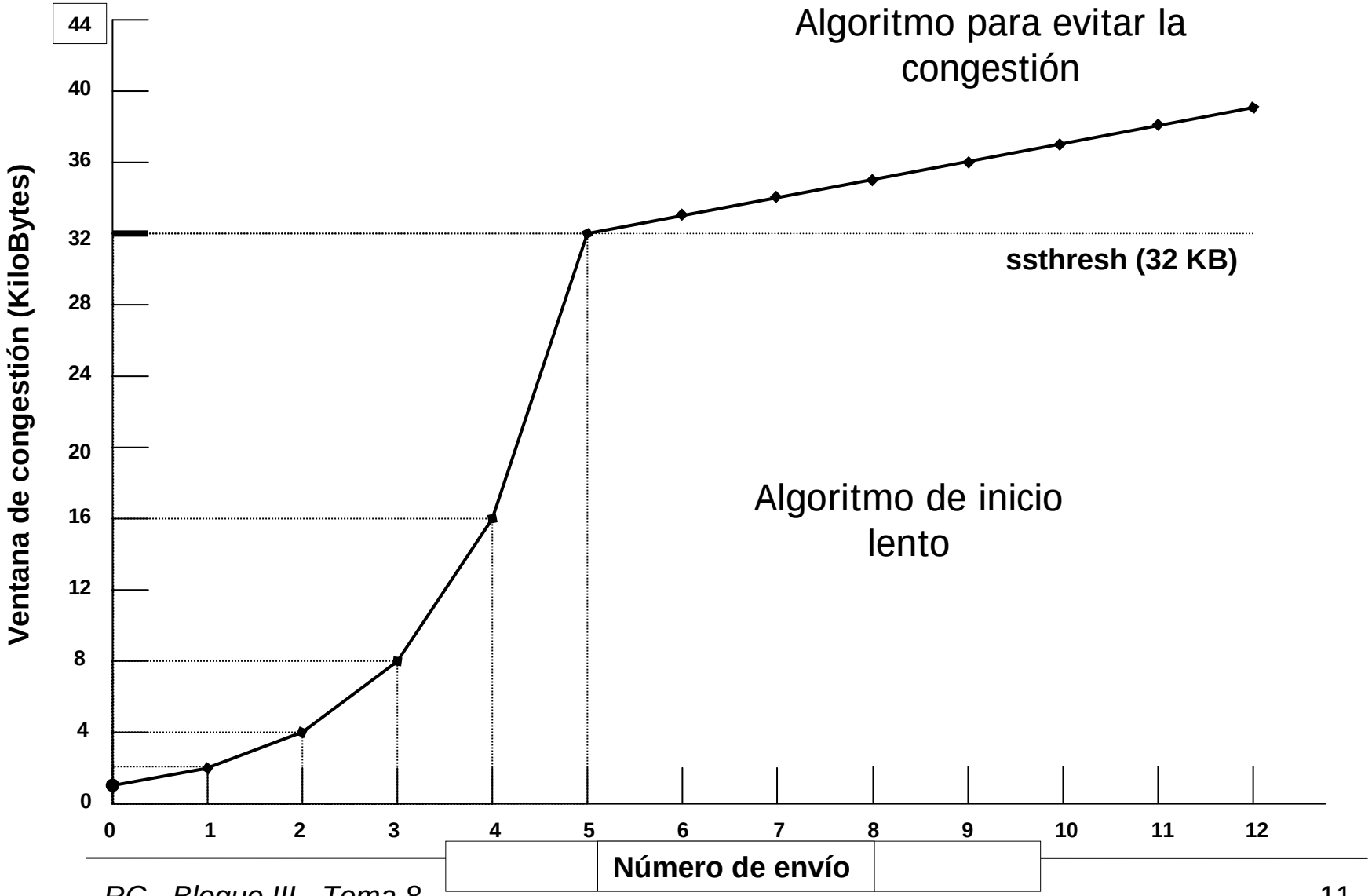
1. Inicialización:
   - cwnd = 1 segmento (en función del MSS)
   - ssthresh = 65535 bytes
2. TCP no podrá enviar más del mínimo de la ventana del receptor (win) y la ventana de congestión (cwnd).
3. Cuando se produce congestión (el timeout expira o se reciben tres ACKs repetidos) ➔ ssthresh = ½ de la ventana activa
   - Ventana activa = min(win, cwnd)
   - Valor mínimo de ssthresh es siempre de 2 segmentos.
   - Además, si es por timeout ➔ cwnd = 1 segmento
4. Cada vez que llega un ACK, se actualiza cwnd:
   a) Si cwnd <= ssthresh ➔ Algoritmo de inicio lento (aumento exponencial).

      cwnd = cwnd + tamaño segmento

   b) Si cwnd > ssthresh ➔ Se aumenta cwnd en una cantidad nunca superior a un segmento (aumento lineal). Se aplica la siguiente fórmula:

$$cwnd = cwnd + \frac{tamaño\ segmento^2}{cwnd} + \frac{tamaño\ segmento}{8}$$

Algoritmo para evitar la congestión



Recuperación y retransmisión rápida

- Modificaciones sobre el algoritmo para evitar la congestión propuestas por Jacobson (1990).
- Requisitos: cada vez que TCP recibe un segmento fuera de orden ➔ Genera un ACK (repetido) sin retardarlo.
- Si se reciben uno o dos ACKs repetidos ➔ Se asume que los segmentos se reciben desordenados.
- Si se reciben tres o más ACKs repetidos ➔ Se asume que se ha perdido un segmento y se retransmite el supuesto segmento perdido sin esperar a que venza el timeout (Algoritmo de retransmisión rápida).
- A continuación se aplica el algoritmo de control de congestión y no el algoritmo de inicio lento (Algoritmo de recuperación rápida).

Recuperación y retransmisión rápida

3. Si se detecta congestión por recibir tres ACKs repetidos:
   - ssthresh = ½ cwnd
   - Se retransmite el segmento perdido
   - cwnd = ssthresh + 3 segmentos
4. Cada vez que llega un ACK:
   a) Cada vez que se recibe otro ACK duplicado:
      - cwnd = cwnd + 1 segmento
      - Se transmite si lo permite cwnd
   b) Cuando se recibe un ACK que confirma datos nuevos (➔ la retransmisión se ha recibido):
      - cwnd = ssthresh (del paso 1)
      - Se reduce la tasa de transmisión a la mitad del valor que tenía cuando se produjo el fallo.

Control de congestión: Ejemplo

- En primer lugar se verá el establecimiento de conexión y como se inicializan los valores de cwnd y ssthresh.
- Además, en esta conexión se produce una pérdida en el primer SYN, y esto afecta directamente a los valores iniciales de cwnd y ssthresh:
  - cwnd = 256 bytes
  - ssthresh = 512 bytes

Segm. #	Acción			Variable	
	Enviar	Recibir	Observaciones	cwnd	ssthresh
			Inicialización	256	65535
	SYN				
			Timeout	256	512
	SYN				
		SYN, ACK			
	ACK				

Control de congestión: Ejemplo

Segm. #	Acción			Variable	
	Enviar	Recibir	Observaciones	cwnd	ssthresh
1	1:257				
2		ACK 257	Alg. inicio lento	512	512
3	257:513				
4	513:769				
5		ACK 513	Alg. inicio lento	768	512
6	769:1025				
7	1025:1281				
8		ACK 769	Alg. evitar cong.	885	512
9	1281:1537				
10		ACK 1025	Alg. evitar cong.	991	512

Control de congestión: Ejemplo

Segm. #	Acción			Variable	
	Enviar	Recibir	Observaciones	cwnd	ssthresh
58		ACK 6657	ACK nuevo	2426	512
59	8705:8961				
60		ACK 6657	ACK repetido 1	2426	512
61		ACK 6657	ACK repetido 2	2426	512
62		ACK 6657	ACK repetido 3	1792	1024
63	6657:6913		Retransmisión		
64		ACK 6657	ACK repetido 4	2048	1024
65		ACK 6657	ACK repetido 5	2304	1024
66		ACK 6657	ACK repetido 6	2560	1024
67	8961:9217				

Control de congestión: Ejemplo

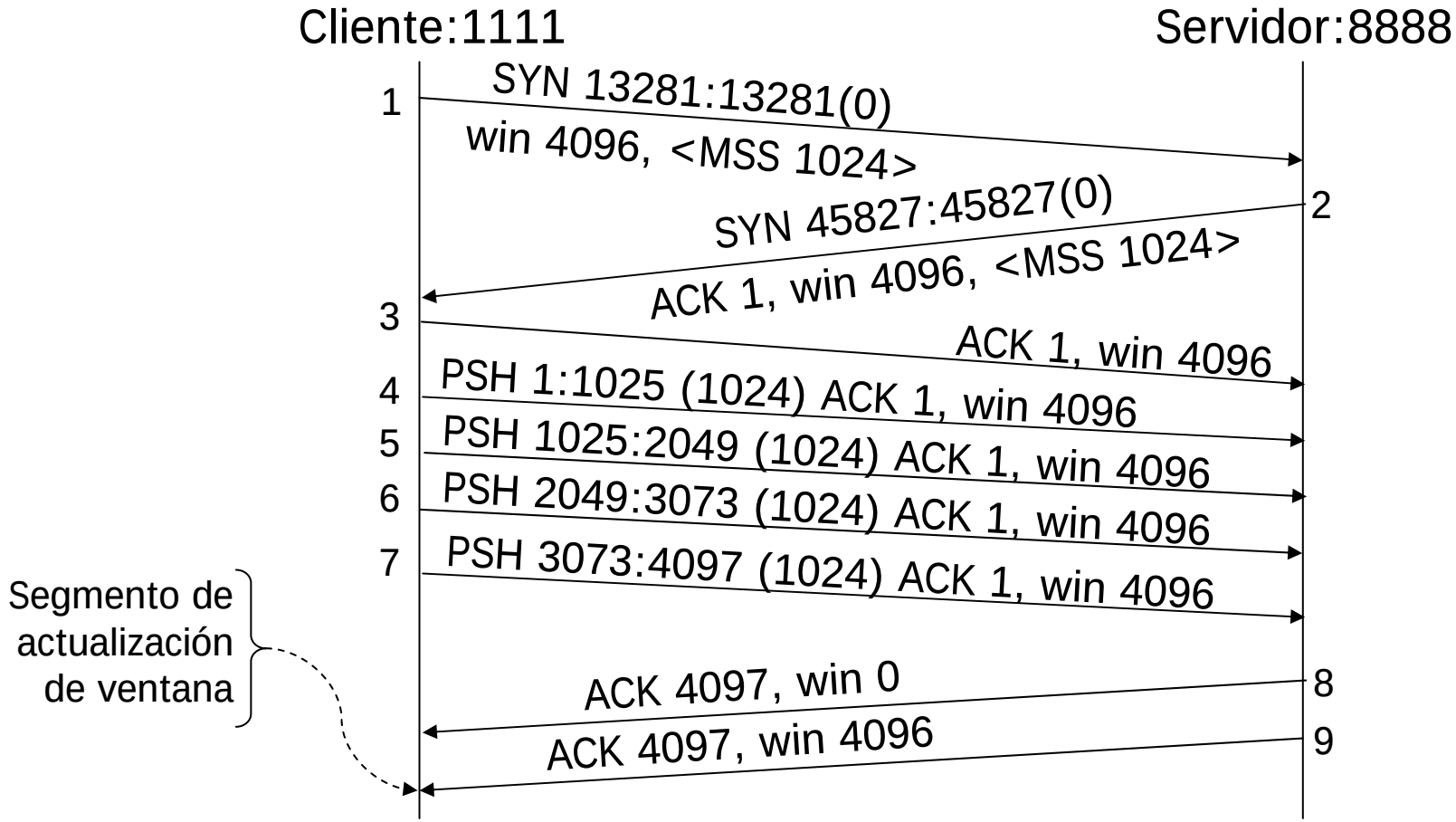
Segm. #	Acción			Variable	
	Enviar	Recibir	Observaciones	cwnd	ssthresh
68		ACK 6657	ACK repetido 7	2816	1024
69	9217:9473				
70		ACK 6657	ACK repetido 8	3072	1024
71	9473:9729				
72		ACK 8961	ACK nuevo	1280	1024

TCP: Temporizadores

- TCP gestiona 4 temporizadores diferentes con cada conexión:
  - Un temporizador de retransmisiones: se utiliza cuando se espera un ACK del otro extremo.
  - Un temporizador de persistencia: mantiene la información del tamaño de ventana, incluso si el otro extremo cierra su ventana de recepción.
  - Un temporizador de “keepalive”: dectecta cuando el otro extremo se reinicializa o está caído.
  - El temporizador 2MSL: mide el tiempo que la conexión está en el estado TIME-WAIT.

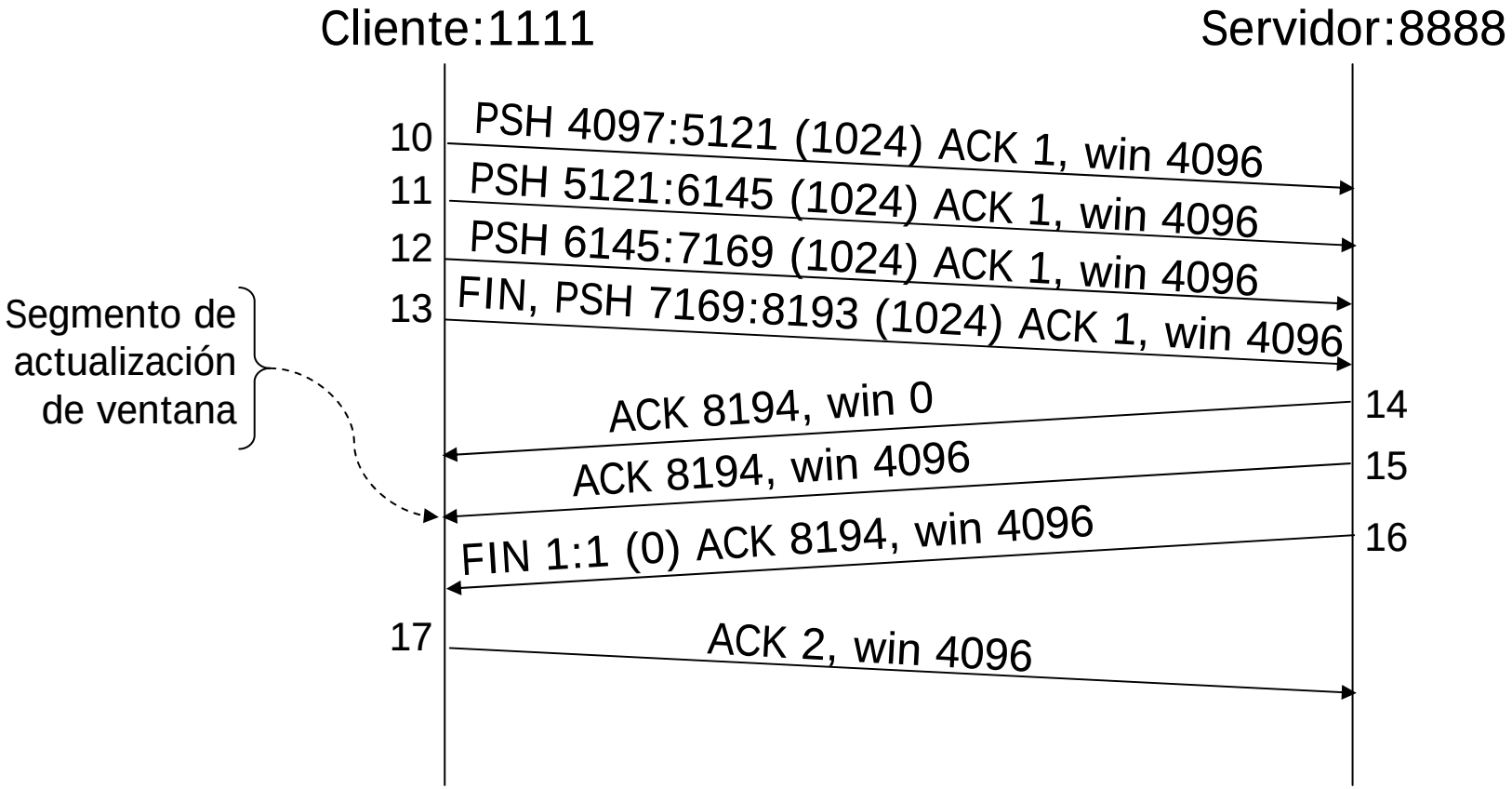
Temporizador de persistencia

- Emisor rápido, receptor lento



Temporizador de persistencia

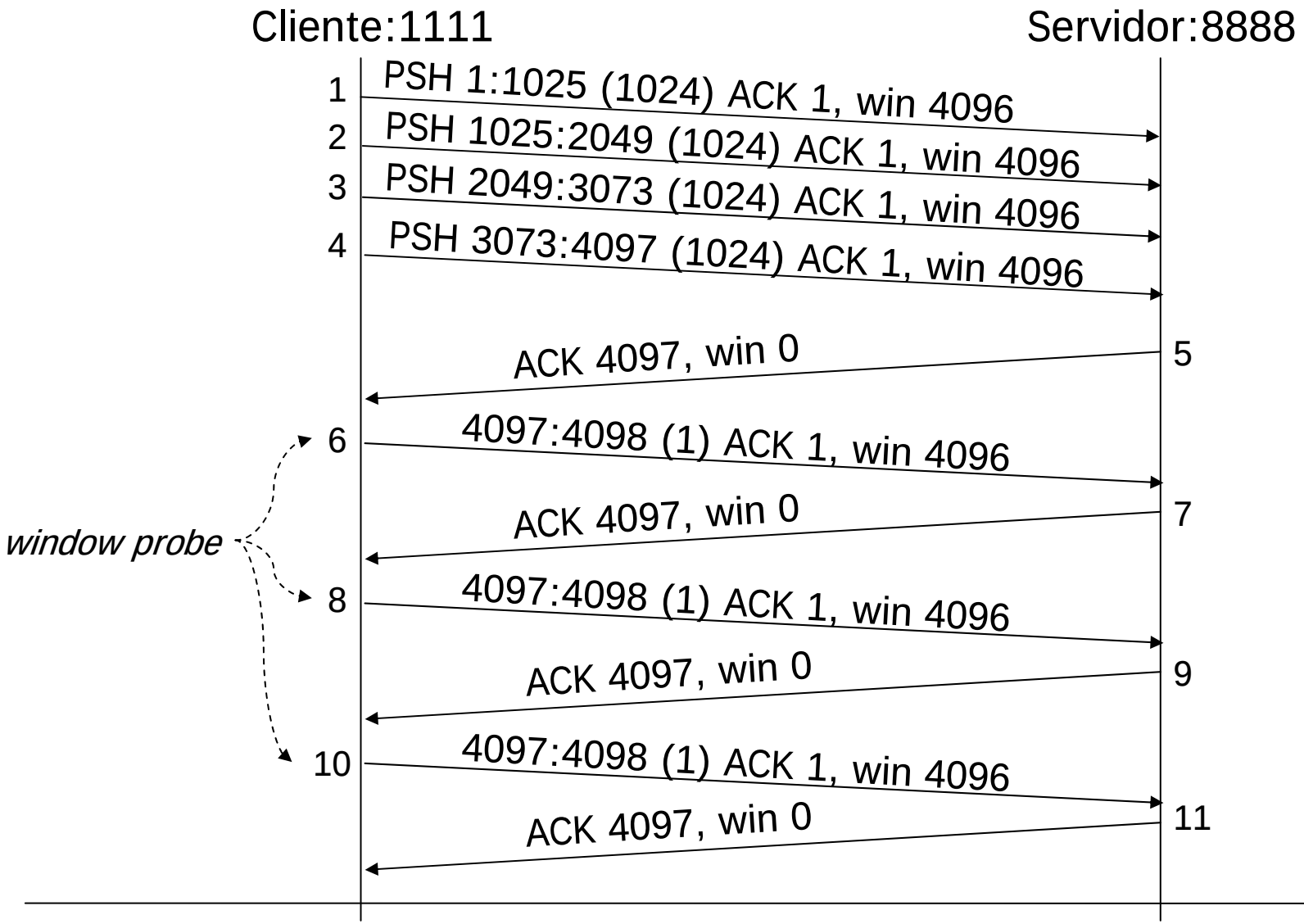
- Emisor rápido, receptor lento (continuación)



Temporizador de persistencia

- ¿Qué ocurre si se pierde el ACK del segmento 9?
  - El cliente está esperando que el servidor le actualice el tamaño de ventana.
  - El servidor ha actualizado la ventana y está esperando que le lleguen nuevos datos del servidor.
- ➔ Se entra en una situación de abrazo mortal.
- Para arreglarlo TCP, después de un tiempo sin que se abra la ventana, pregunta periódicamente si la ventana se ha actualizado utilizando unos segmentos especiales denominados: **window probes**.
- Los window probes no son más que segmentos de un byte utilizados para comprobar si realmente la ventana se ha modificado o no.

Temporizador de persistencia



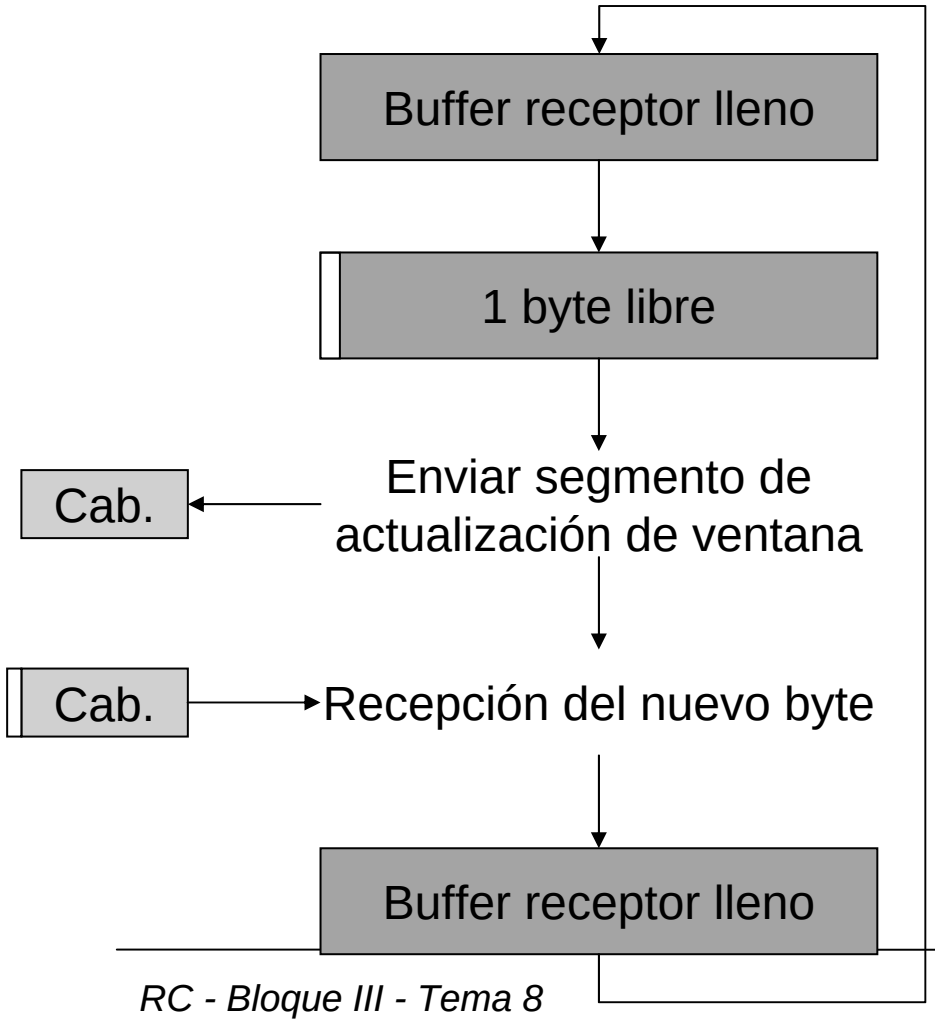
Temporizador de persistencia

- El temporizador de persistencia se activará en los siguientes intervalos: 5, 6, 12, 24, 48, 60, 60, … segundos.
  - Este temporizador se basa en el exponential backoff, pero limitado entre 5 y 60 segundos.
- Los window probes contienen 1 byte datos (nº secuencia 4097):
  - TCP permite enviar un byte de datos por encima del tamaño de la ventana.
  - El ACK del receptor NO asiente el byte del window probe (ACK 4097) ➔ Este byte se continúa retransmitiendo.
- ¿Cuándo se para la transmisión de window probes? Nunca, se continúan transmitiendo a intervalos de 60 segundos hasta que:
  - El receptor abre la ventana.
  - Se cierra la conexión por las aplicaciones.

Síndrome de la ventana tonta

- El control de flujo de TCP puede provocar que se intercambien pequeñas cantidades de datos, en lugar de esperar y enviar un segmento completo.
- Causas:
  - La aplicación receptora consume los datos muy lentamente
  - La aplicación emisora produce los datos muy lentamente
- En cualquier caso, los datos se envían con segmentos muy pequeños:
  - Uso ineficiente del ancho de banda
  - Incremento del procesamiento por parte de TCP

Síndrome de la ventana tonta



- Por ejemplo, la aplicación recupera los datos lentamente:
  1. El buffer del TCP receptor se llena
  2. El TCP receptor notifica al emisor que su ventana está cerrada
  3. La aplicación receptora lee un byte
  4. El TCP receptor envía un ACK al emisor para anunciarle que dispone de un byte libre
  5. El TCP emisor envía un segmento con un byte de datos
  6. Volvemos al punto 1

Solución de Clark (RFC 813)

- Solución de Clark: no enviar notificaciones de ventana pequeñas (p.e. 1 byte).
- En cambio, cerrar la ventana completamente hasta que:
  - Hay espacio para un segmento entero (MSS)
  - Se ha liberado la mitad del espacio del buffer del receptor (para buffers muy pequeños)
- La solución de Clark y el algortimo de Nagle son complementarios:
  - Algoritmo de Nagle: el emisor acumula datos hasta que hay “suficientes” datos para enviar.
  - Solución de Clark: el receptor consume datos hasta que se ha liberado espacio “suficiente” en el buffer para ser notificado

Temporizador de keepalive

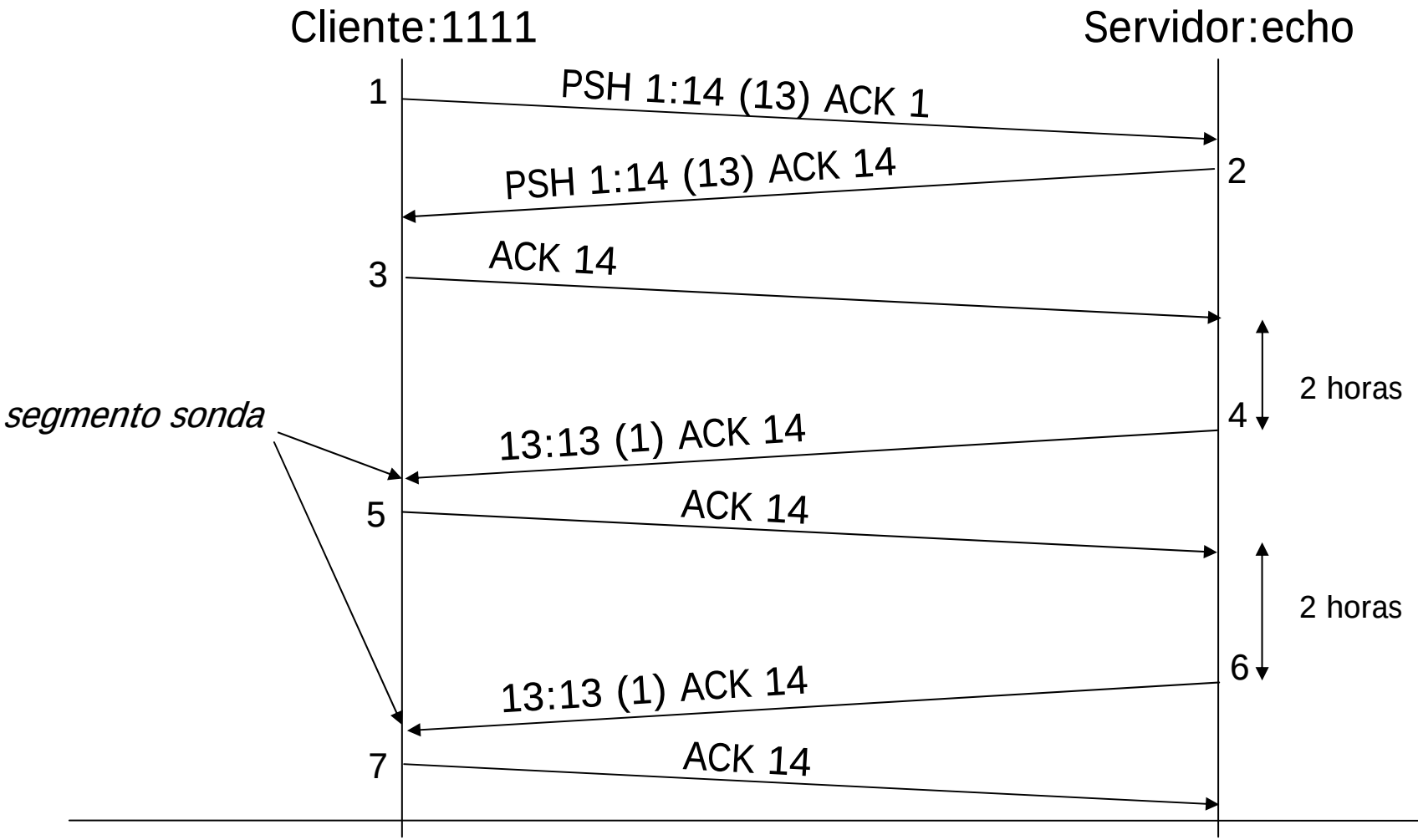
- En una conexión TCP sin intercambio de datos, no se produce ningún intercambio de paquetes (polling).
- Esto puede plantear problemas en situaciones de fallos del cliente (caídas, cliente inalcanzable o reinicializaciones).
- Para solucionar esto se encuentra el temporizador de keepalive (aunque no es parte del RFC de TCP).
- Tiene sentido en aplicaciones servidor que pueden liberar recursos si el cliente no está realmente conectado.
- Funcionamiento: después de 2 horas de inactividad, el servidor enviará un segmento sonda (similar al window probe).
  - Segmento sonda: segmento de un byte, correspondiente al último byte enviado.

Temporizador de keepalive

- El cliente puede estar en uno de estos cuatro estados:
  1. El cliente está levantado, funcionando y es alcanzable ➔ El cliente responderá correctamente al servidor y éste reinicia el timeout a 2 horas
     - También se reinicia el timeout si se produce intercambio de datos
  2. El host cliente se encuentra caído y aún está caído o reiniciazándose ➔ El servidor no obtendrá respuesta y lo reintentará 10 veces, en intervalos de 75 segundos (si no recibe respuesta se cierra la conexión).
  3. El host cliente se ha caído y se ha inicializado ➔ El cliente responderá con un segmento de reset y el servidor cerrará la conexión.
  4. El cliente está levantado y funcionando, pero no es alcanzable ➔ Este caso es idéntico al 2, y el servidor es incapaz de diferenciarlos.

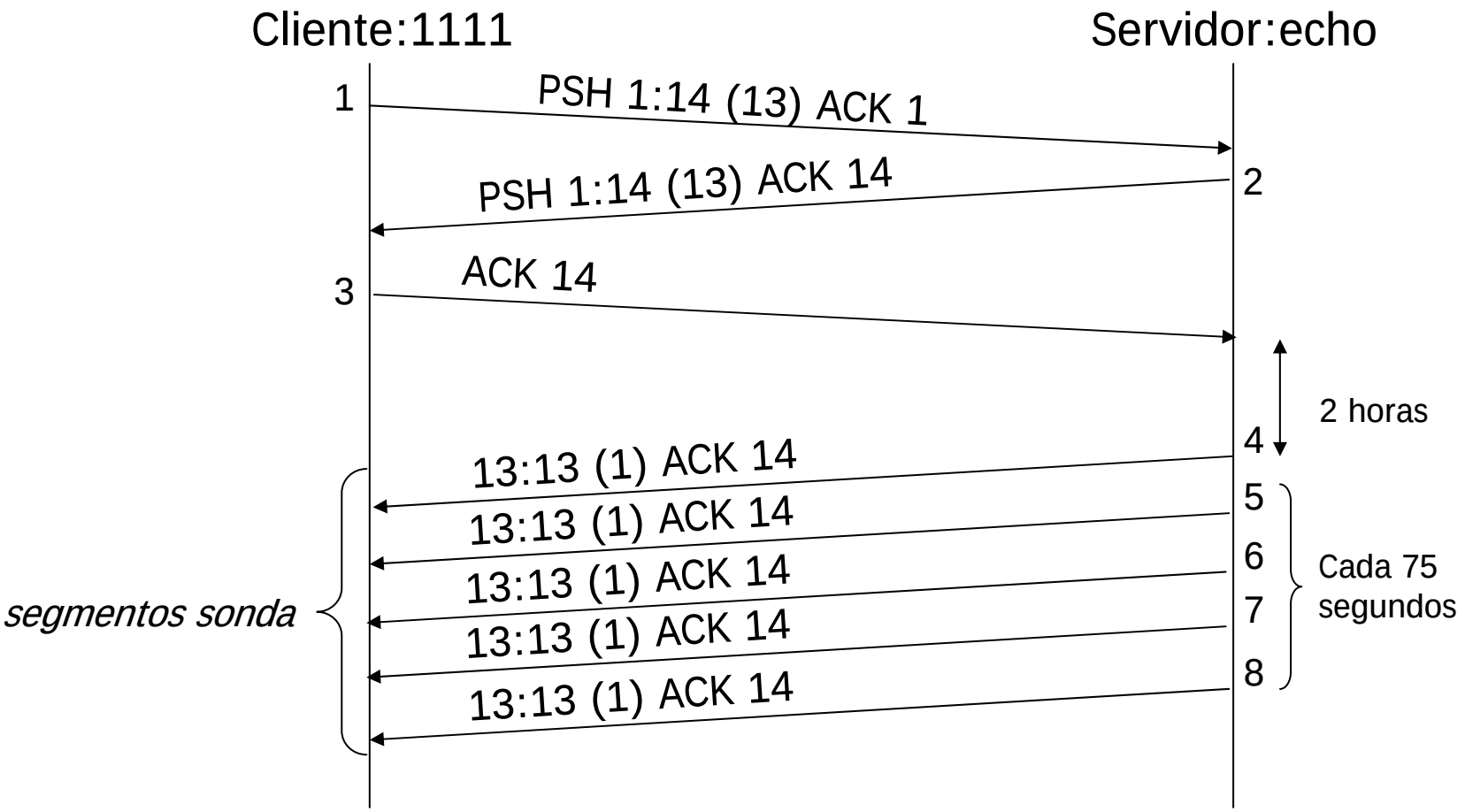
Temporizador de keepalive

- Caso 1: cliente OK



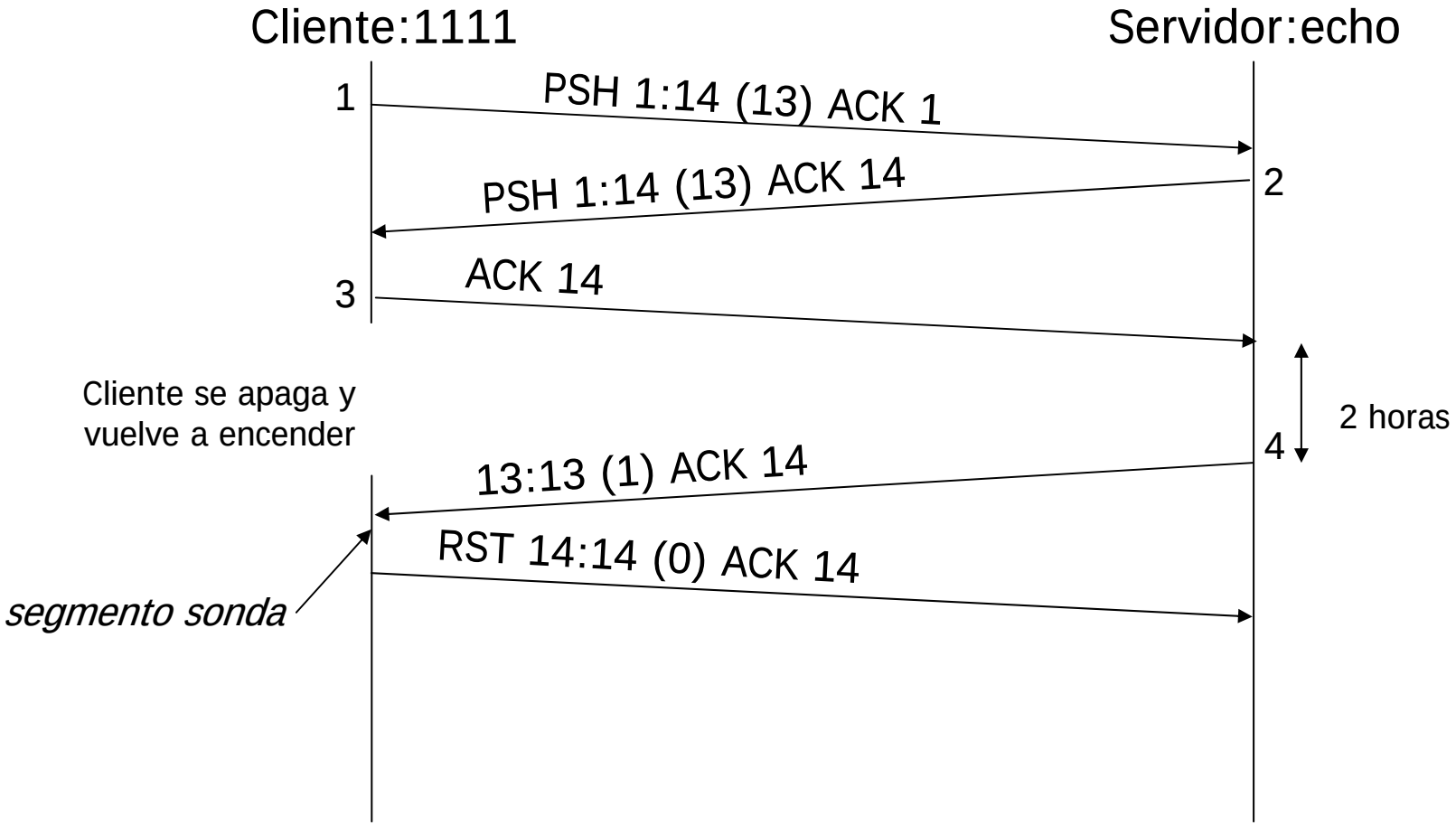
Temporizador de keepalive

- Caso 2: cliente caído



Temporizador de keepalive

- Caso 3: cliente caído y reiniciado



Temporizador de keepalive

- Caso 4: cliente inalcanzable (= caso 2)