Bloque III: El nivel de transporte

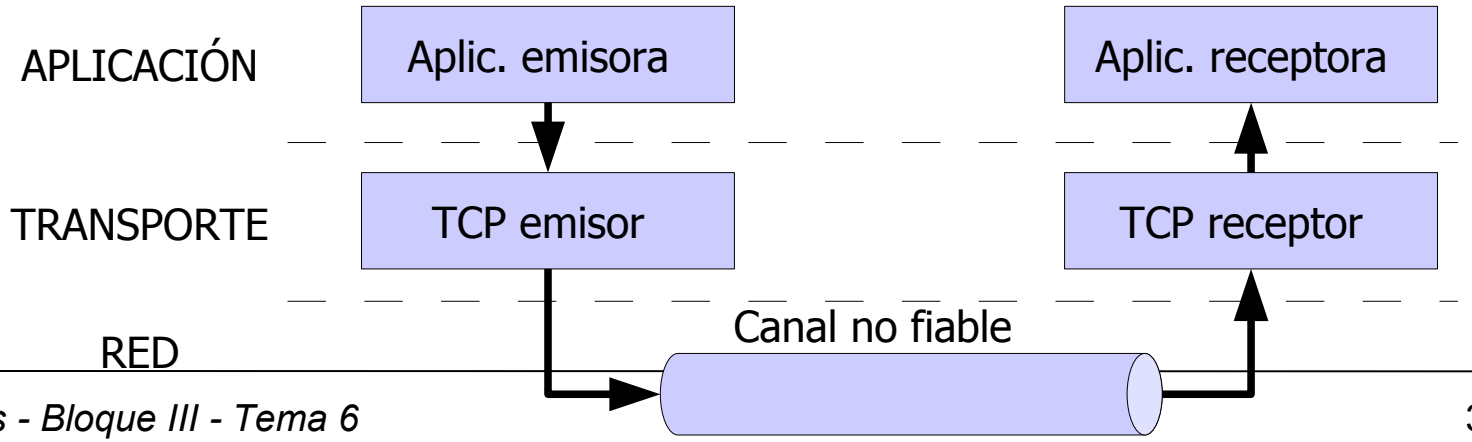
Tema 6: Intercambio de datos TCP

Índice

- Bloque III: El nivel de transporte
  - Tema 6: Intercambio de datos TCP
    - Introducción
      - Protocolos ARQ
      - Intercambio de datos TCP
    - Flujo de datos interactivo
      - ACKs retardados
      - Algoritmo de Nagle
    - Flujo de datos no interactivo
      - Control de flujo
      - Control de congestión
    - Temporizador de keepalive
- **Lecturas recomendadas:**
  - Capítulo 3, secciones 3.4, 3.5.3, 3.5.4, 3.5.5, 3.6.1 y 3.7 de “Redes de Computadores: Un enfoque descendente”. James F. Kurose, Keith W. Ross. Addison Wesley.
  - Capítulos 14, 15 y 17 de “TCP/IP Illustrated, Volume 1: The Protocols”, W. Richard Stevens, Addison Wesley.

Introducción

- Servicio de **transferencia fiable de datos**:
  - Los datos son recibidos correctamente (no se corrompen).
  - Los datos se reciben completos (no se pierde nada).
  - Los datos se entregan en el orden en el que fueron enviados.
- Problema: implementar un servicio de transferencia fiable de datos sobre un servicio (protocolo) de transferencia de datos **no fiable**.
  - Por ejemplo, TCP sobre IP.
  - Aunque este problema aparece en otros niveles: enlace y aplicación.

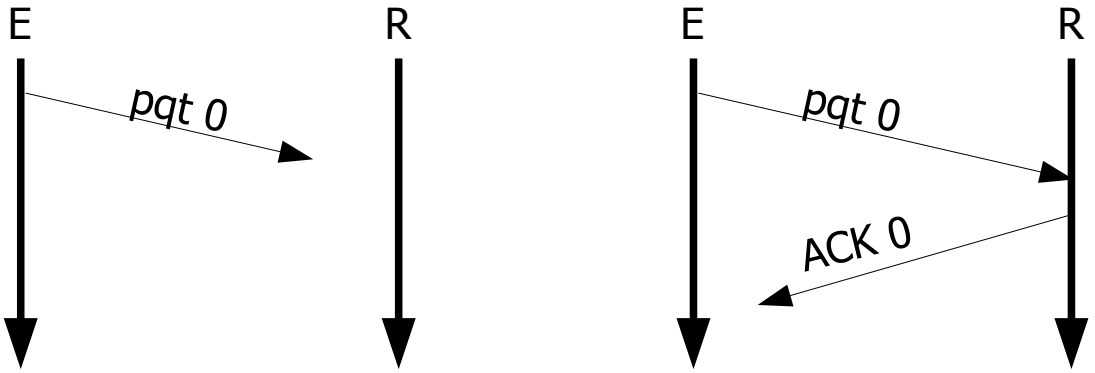


Introducción

- Si el canal es fiable → No hay nada que hacer
- Si el canal puede corromper los datos (pero mantiene el orden y no pierde datos), necesito:
  - Detección de errores (en el receptor): **checksum**.
  - Confirmación del receptor: confirmando los datos recibidos, positiva (**ACK**) y, opcionalmente, negativa (NAK).
  - **Retransmisión**: por parte del emisor, si un paquete no se ha recibido correctamente.
  - **Número de secuencia** para identificar los paquetes enviados y confirmados.
- Este tipo de protocolos se denominan ARQ (**Automatic Repeat reQuest**).

Introducción

- ¿Y si el canal también puede perder datos?

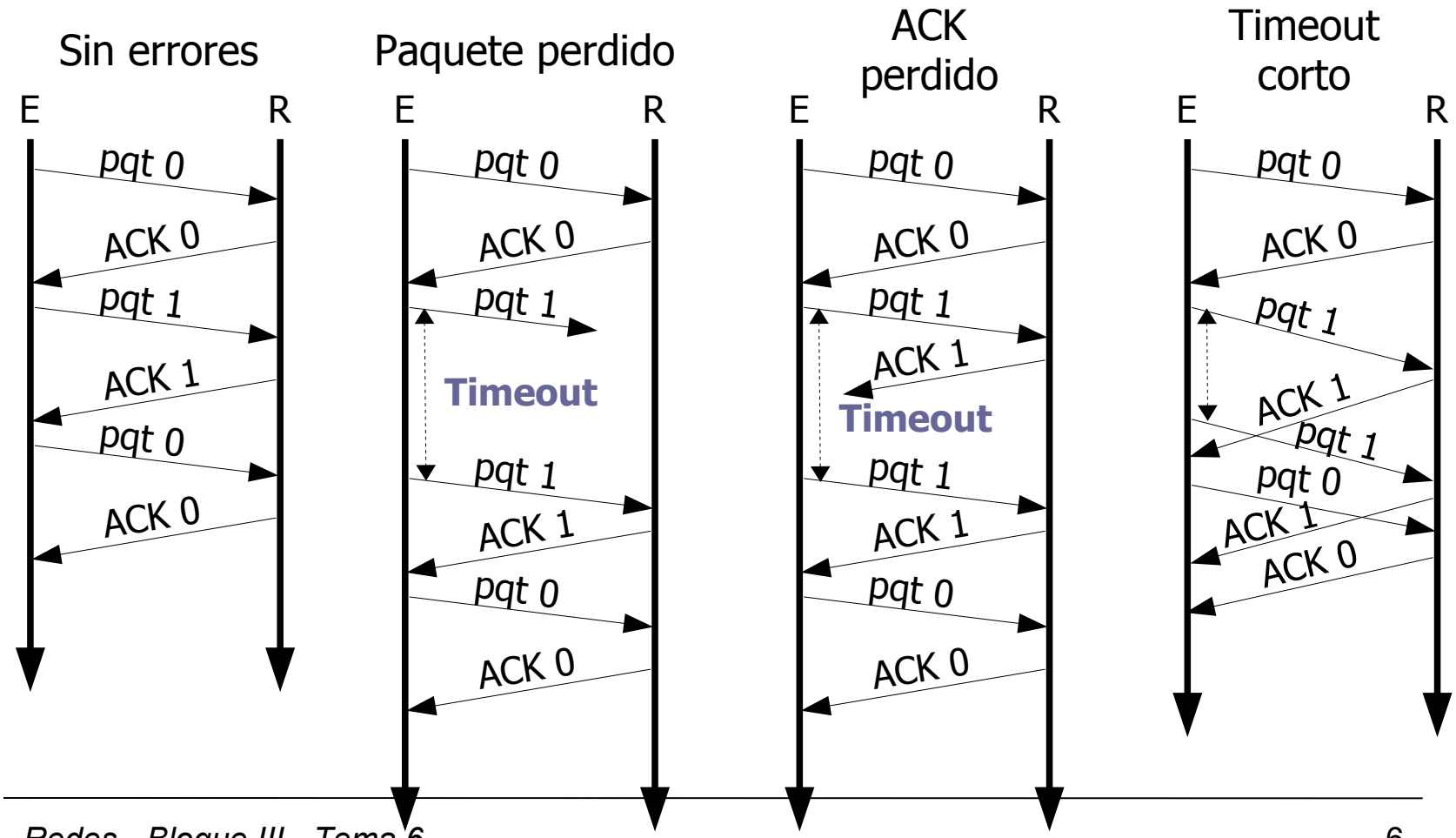


- Solución: esperar y retransmitir → Necesito un **temporizador** (o timeout) en el emisor para cada paquete de datos enviado.

- Esperar sí, pero ¿cuánto tiempo?
  - Idealmente: el tiempo de ida y vuelta (RTT – Round Trip Time) de un paquete en la red + un tiempo extra de procesamiento.
  - En muchas redes (p.e. Internet) el RTT no tiene un valor máximo.
  - Si espero demasiado → Tardo mucho en recuperarme del error.
  - Si espero poco → Envío muchos datos repetidos, para nada.

Protocolo ARQ de parada y espera

- Protocolo de **parada y espera**: el emisor no envía datos nuevos hasta confirmar que el receptor ha recibido correctamente los datos anteriores. También denominado de **bit alternante**.

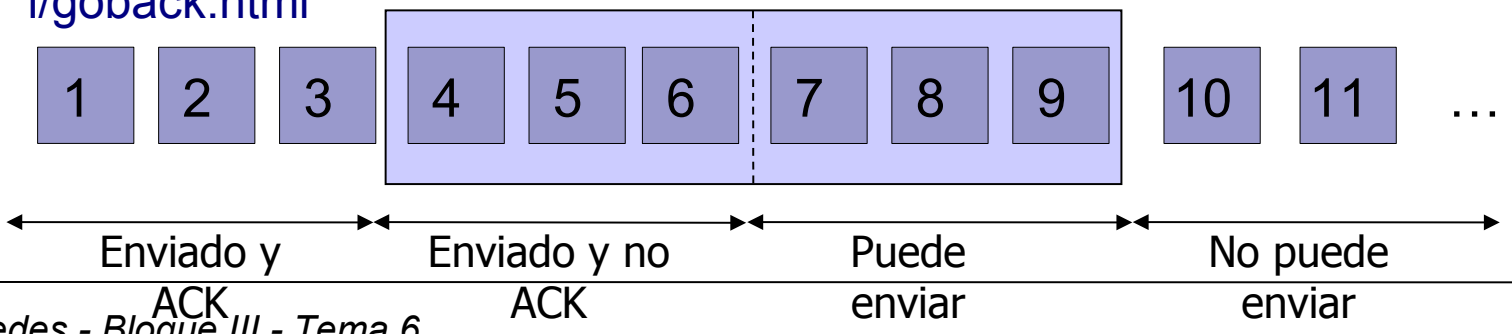


Protocolo ARQ de parada y espera

- Obtiene un rendimiento (velocidad) muy bajo.
- Solución: enviar varios paquetes sin esperar a los mensajes de confirmación → Procesamiento en cadena.
- ¿Qué necesito?
  - Aumentar el tamaño de los números de secuencia → Varios paquetes podrán estar en la red simultáneamente, sin confirmar.
  - El emisor necesitará un buffer para almacenar los paquetes transmitidos pero no confirmados.
  - El receptor necesitará un buffer para almacenar los paquetes recibidos correctamente (pero que la capa superior aún no puede procesar).
- Protocolos ARQ de procesamiento en cadena:
  - Retroceder N (GBN – Go-Back-N)
  - Repetición Selectiva (SR – Selective Repeat)

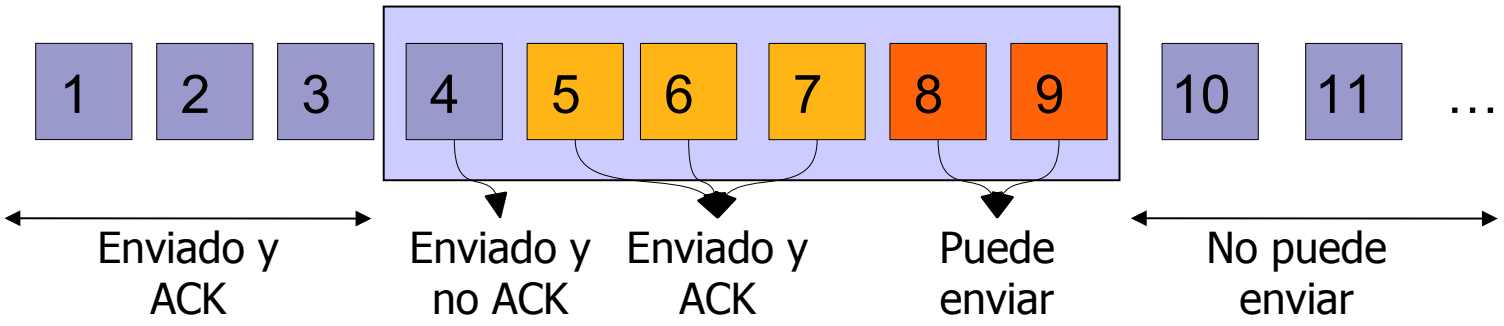
Protocolo ARQ retroceder N

- El emisor puede transmitir varios paquetes sin esperar a que estén confirmados.
- Se establece un máximo de N paquetes enviados sin confirmación.
- Se suelen representar los paquetes que se pueden enviar como una ventana que se va desplazando:
  - Cada vez que se recibe un ACK (nuevo) → Se puede enviar otro paquete.
  - Protocolo de **ventana deslizante**.
  - N = tamaño de ventana.
- El receptor no tiene buffer.
- Los números de secuencia son finitos → Son circulares.
- Sólo utiliza ACKs positivos, pero son **acumulativos**.
- http://computerscience.unicam.it/marcantoni/reti/applet/GoBackProtocol/goback.html



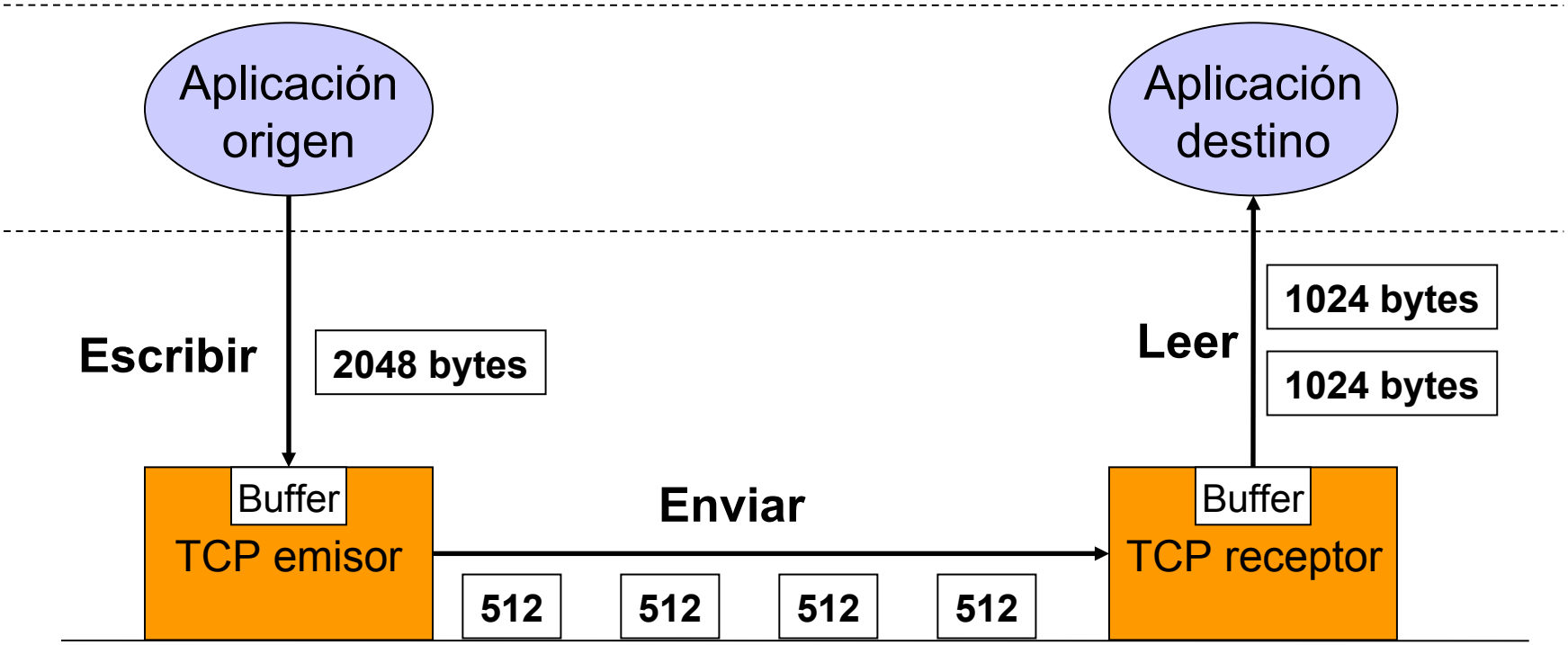
Protocolo ARQ de repetición selectiva

- Problema de retroceder N: un error en un paquete hace que se repitan otros (muchos) paquetes recibidos correctamente.
- Solución: que el emisor únicamente retransmita los paquetes erróneos → Repetición selectiva.
  - Los ACKs son individuales (selectivos - SACK).
  - Se utiliza una ventana, pero con algunos paquetes confirmados.
  - El receptor necesita un buffer.
  - Se utiliza un temporizador para cada paquete enviado.
- http://computerscience.unicam.it/marcantoni/reti/applet/SelectiveRepeatProtocol/selRepProt.html



Intercambio de datos TCP

- En TCP se consideran dos tipos de tráfico de datos:
  - **Interactivo**: gran número de segmentos de pequeño tamaño (menos de 10 bytes). Por ejemplo: telnet, ssh.
  - **No Interactivo**: segmentos de gran tamaño, normalmente el máximo permitido por las limitaciones de la red. Por ejemplo: HTTP, FTP, e-mail.



Intercambio de datos TCP

- Para implementar la fiabilidad, se basa en el modelo ARQ retroceder N:
  - Es un protocolo de ventana deslizante.
  - Los ACKs son acumulativos y positivos.
- Con algunos matices:
  - Cuando el receptor recibe un paquete fuera de orden:
    - No lo descarta, lo almacena en el buffer.
    - Envía un ACK del último paquete correcto.
  - Retransmisión rápida: si el emisor recibe tres ACKs repetidos → Retransmite (sólo) el paquete siguiente (al número de ACK).
  - El emisor mantiene un temporizador por cada grupo de paquetes enviado.

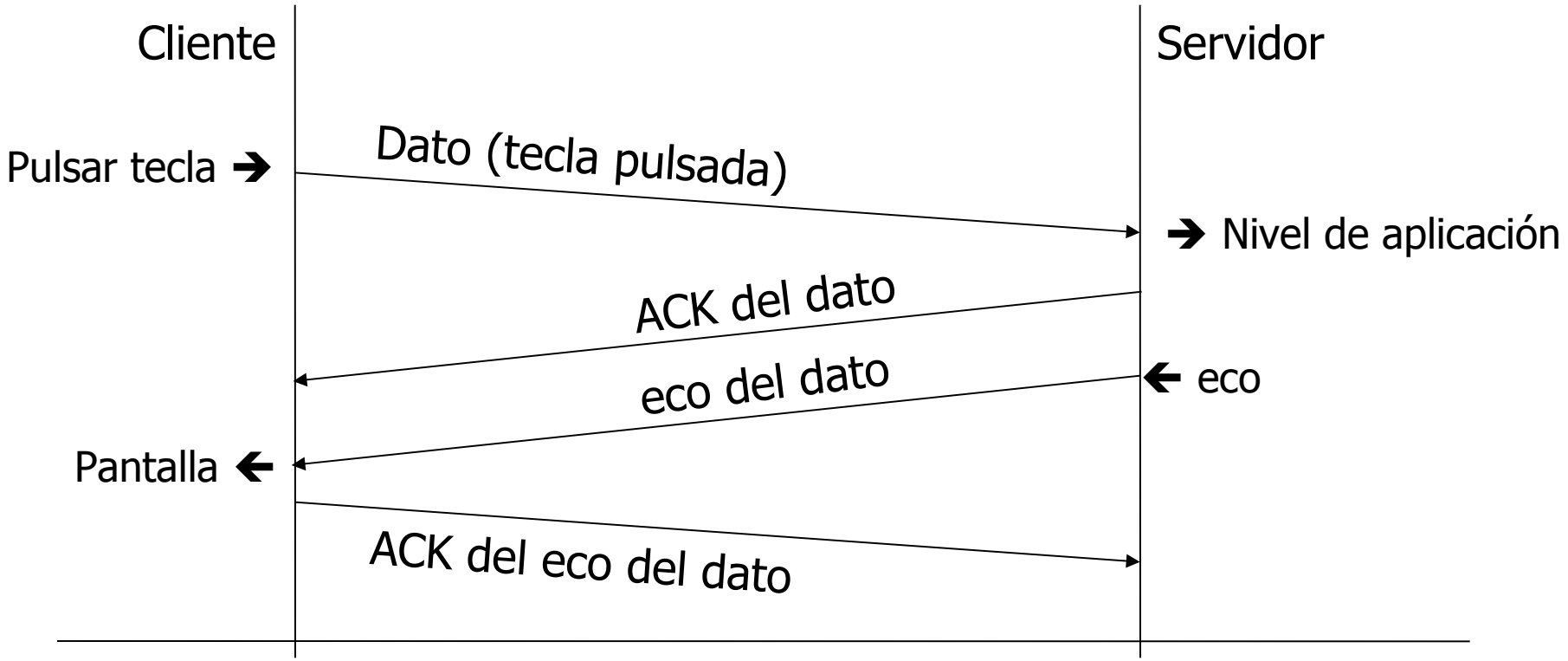
- En el RFC 2018 se propone un receptor TCP con confirmación selectiva (SACK).

Intercambio de datos TCP

- Al tiempo de espera antes de retransmitir se le denomina: Retransmission Timeout (**RTO**).
- En TCP el RTO se calcula a partir del RTT (Round-Trip Time) que se estima continuamente durante toda una conexión TCP.
  - Cuando se envía un segmento, se mide el tiempo que tarda en recibirse su ACK.
- También existe la opción de Timestamp (TSOPT): 10 bytes
  - Campo Timestamp Value (TSval): el emisor indica el valor de su reloj en el momento de la transmisión.
  - Campo Timestamp Echo Reply (TSecr): el receptor copia el TSval en el segmento de respuesta.
- Estimación del RTT con TSOPT:
  - El emisor indica el TSval en los segmentos que envía.
  - Al preparar la respuesta (ACK), el receptor copia el TSval en el campo TSecr.
  - El emisor, al recibir el ACK, comprueba el reloj del sistema, le resta el TSecr y tiene la estimación del RTT.

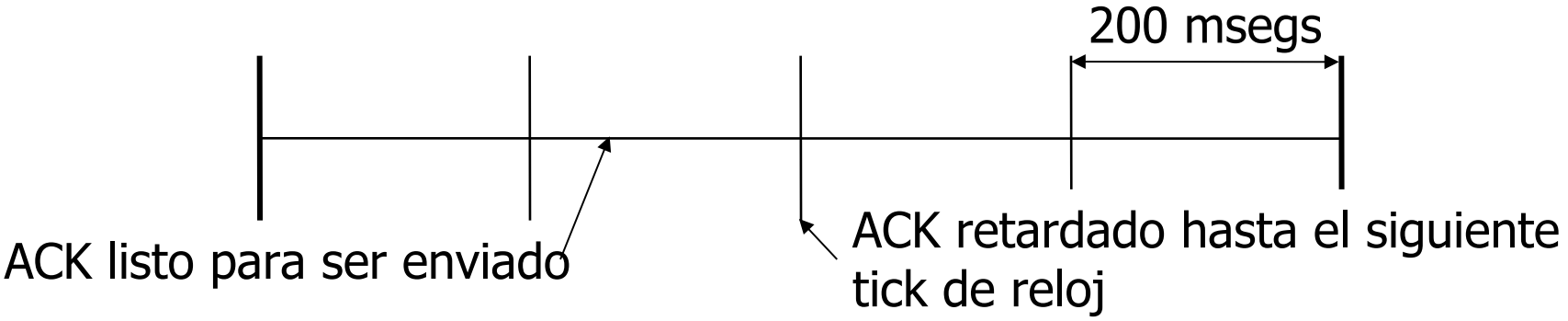
Flujo de datos interactivo

- Funcionamiento del ssh:
  - Envío de la tecla pulsada por el cliente
  - ACK de la tecla pulsada por el cliente
  - Eco de la tecla desde el servidor
  - ACK del eco



Flujo de datos interactivo

- **ACKs retardados**:
  - Objetivo: enviar el ACK + eco en un único datagrama.
- TCP no envía el ACK inmediatamente al recibir el dato, sino que retarda la salida del ACK esperando un tiempo para ver si hay datos para enviarlos con el propio ACK.
- El tiempo de espera es de 200 mseg:
  - No en valor absoluto, sino que se utiliza un reloj que da ticks cada 200 mseg.
- Tanto en el cliente como en el servidor se utilizan los acks retardados.



Flujo de datos interactivo

- El tráfico interactivo genera gran cantidad de paquetes de tamaño muy pequeño (1 byte datos + 20 cab. TCP + 20 cab. IP = 41 bytes) → **tinygrams**
  - En las redes de área local no presenta ningún problema
  - En las WAN supone una gran sobrecarga para la red.
- **Algoritmo de Nagle**:
  - Pretende resolver este problema, y se aplica en una conexión TCP de tráfico interactivo en una red de área extensa.
  - Enunciado: "*Una conexión TCP puede tener un único segmento pequeño que no haya sido confirmado. No se pueden enviar otros segmentos hasta recibir un ACK. En cambio, esos datos se almacenan y son enviados por TCP al llegar el ACK*."
  - Es auto-ajustable: cuanto más rápido lleguen los ACKs → más rápido se enviarán los datos.
- El algoritmo de Nagle convierte a TCP en un protocolo de parada y espera → En algunos casos puede no interesar, por lo que se puede controlar:
  - setTcpNoDelay(boolean) de la clase Socket.

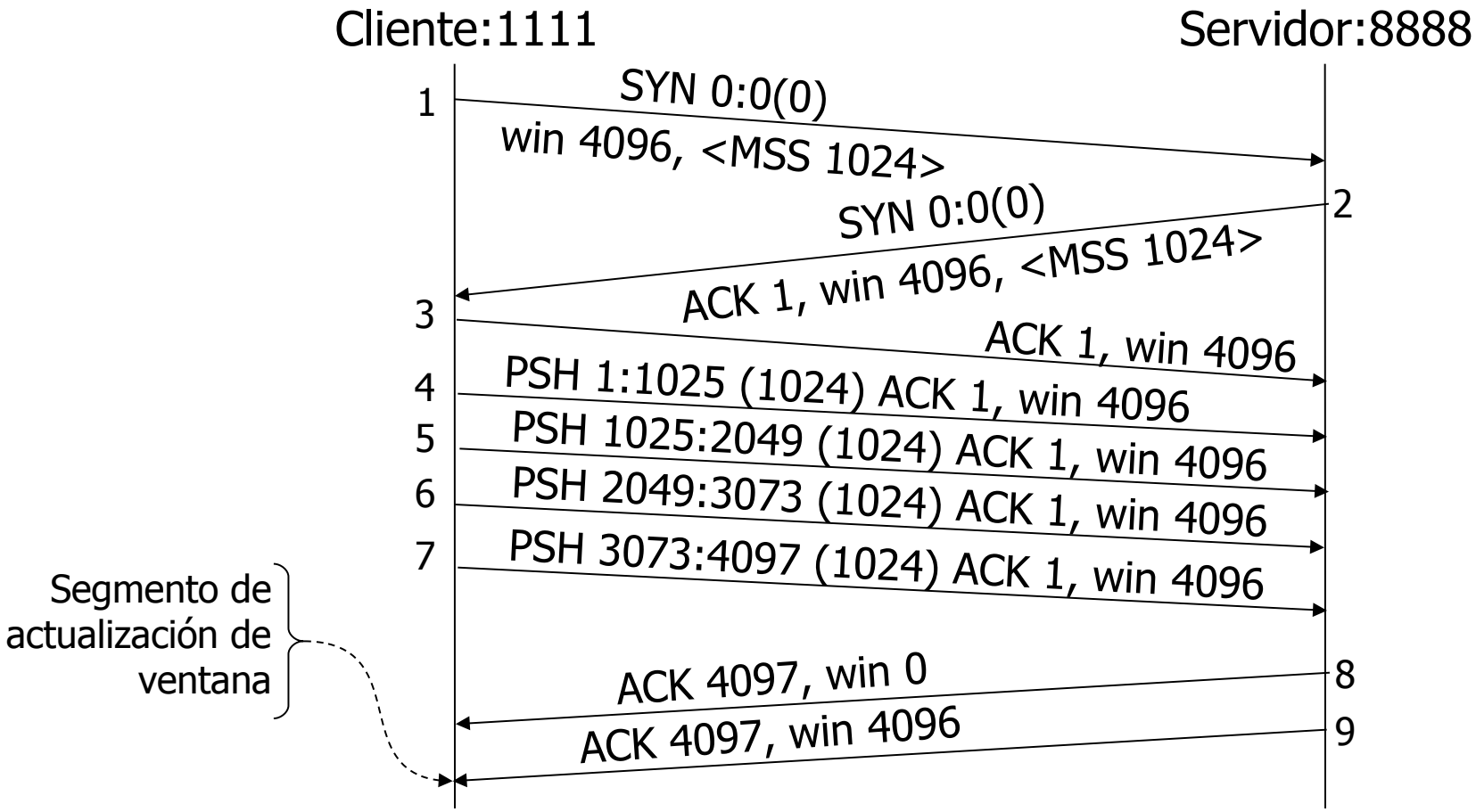
Flujo de datos no interactivo

- También llamado flujo de datos en masa (*bulk data flow*).
- En este tipo de tráfico se generan “pocos” segmentos, pero de gran tamaño.
- El principal problema a resolver en este tipo de tráfico es el **control de flujo**:
  - Evitar que un emisor rápido sature a un receptor lento.

- TCP utiliza una ventana deslizante: permite al emisor enviar múltiples paquetes antes de parar y esperar por el ACK, lo que da una mayor rapidez a este tipo de tráfico.
- Con un protocolo de ventana deslizante no es necesario confirmar todos los paquetes recibidos, sino que se pueden confirmar varios paquetes simultáneamente → ACKs acumulativos.

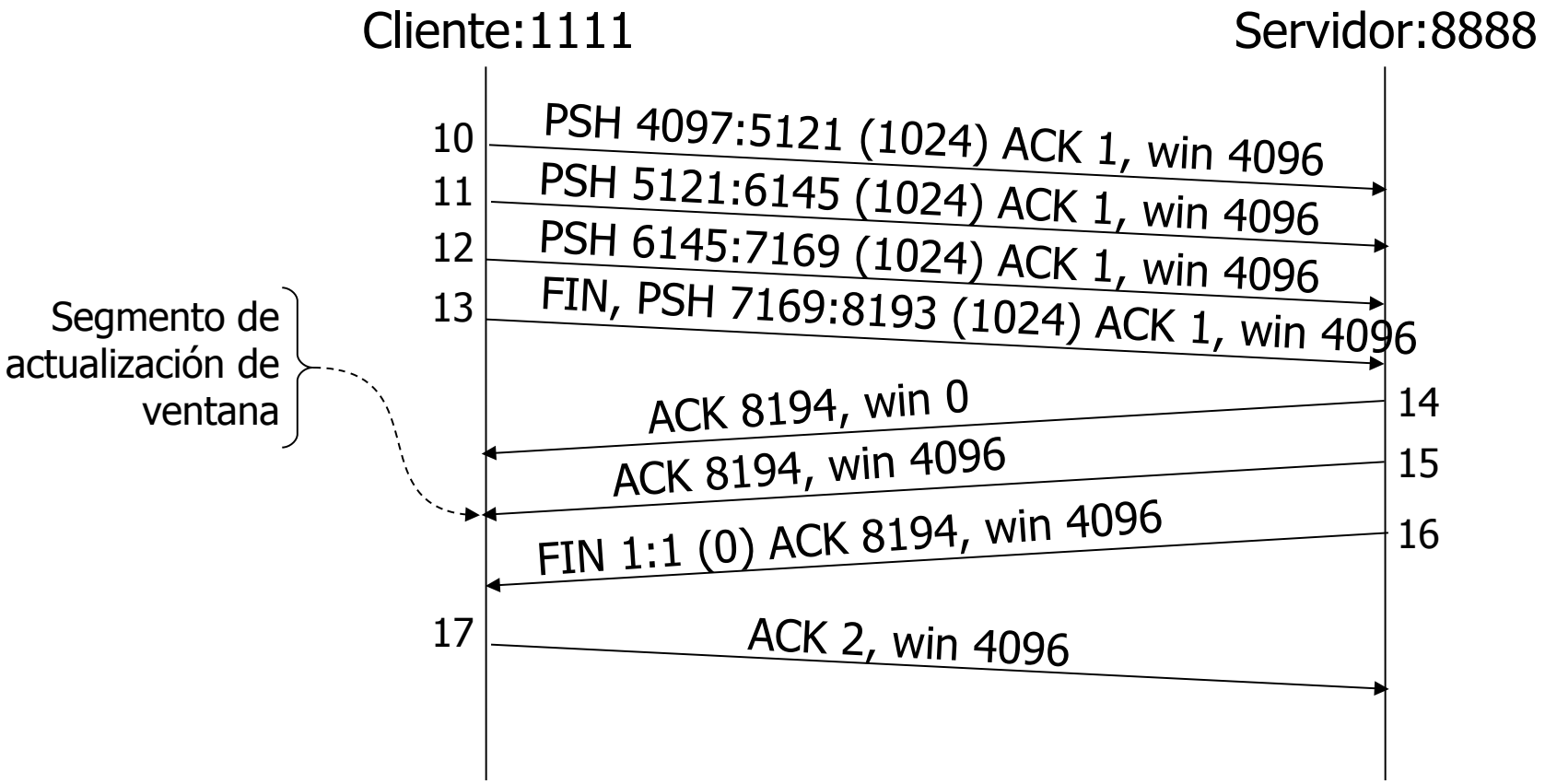
Control de flujo

- Emisor rápido, receptor lento



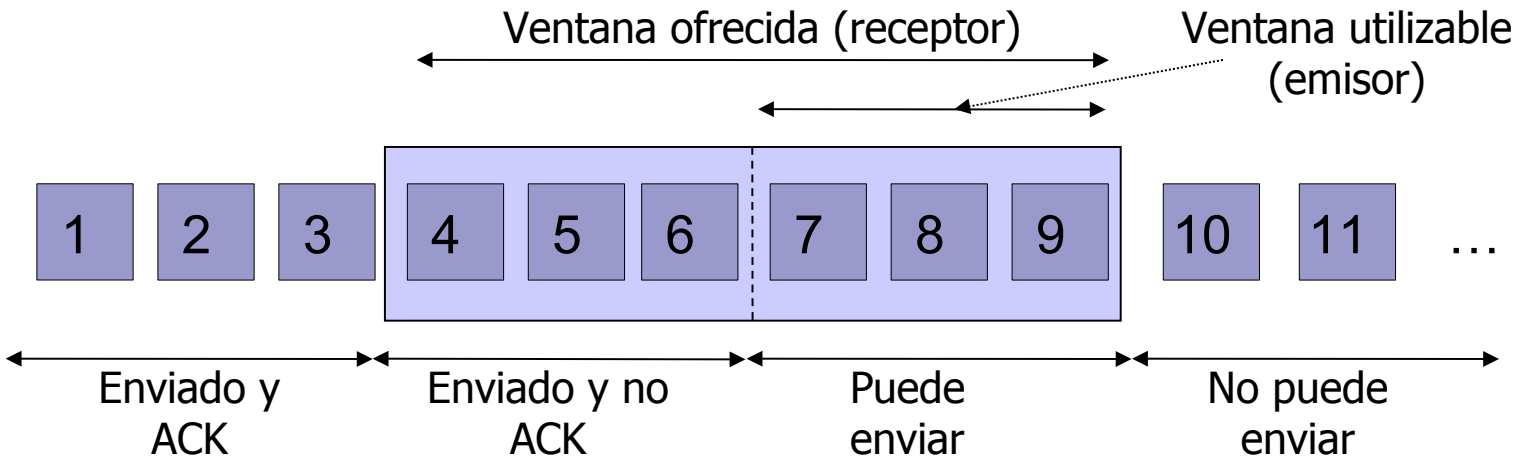
Control de flujo

- Emisor rápido, receptor lento (continuación)



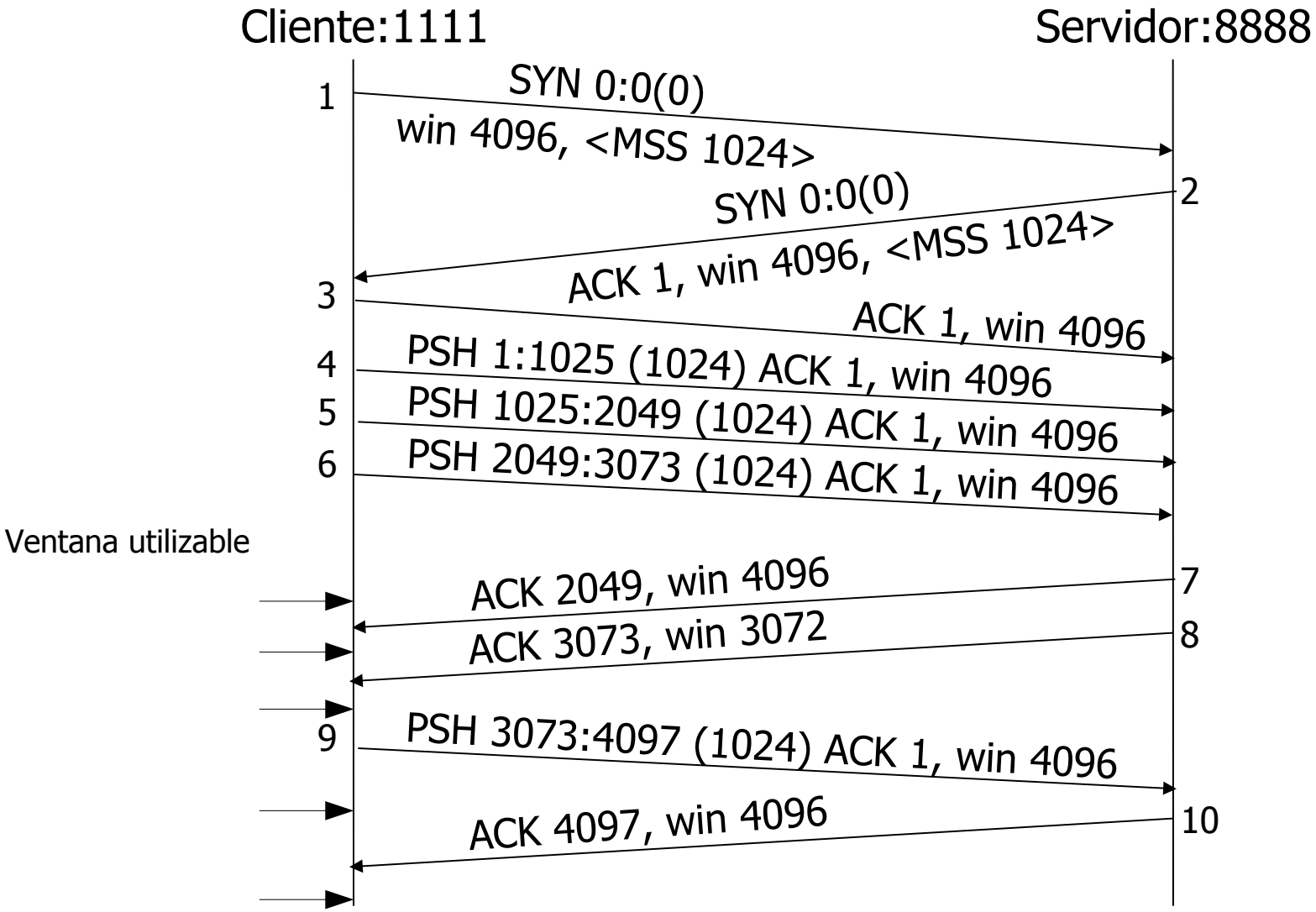
Control de flujo

- Funcionamiento de la ventana deslizante:

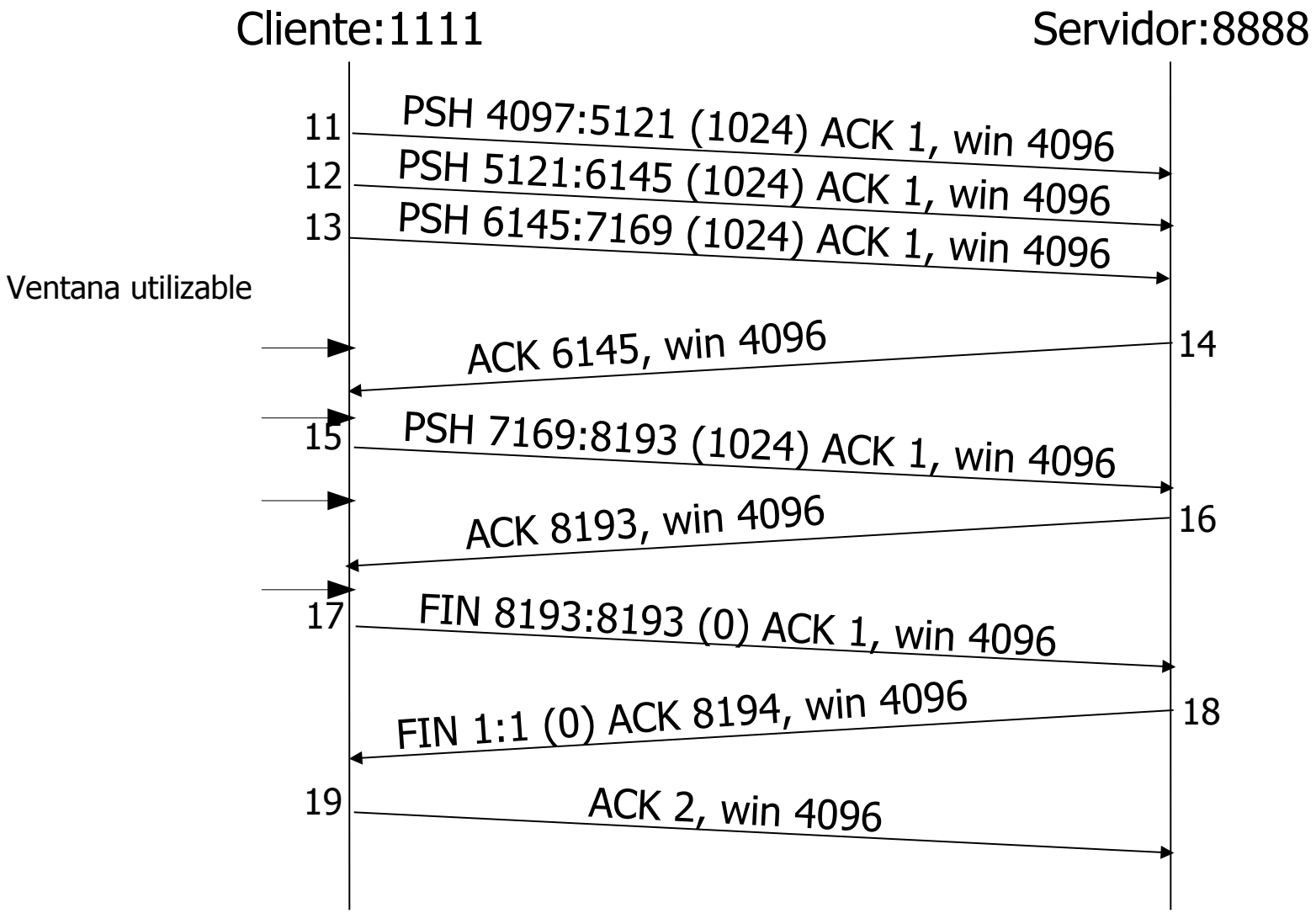


- **Ventana ofrecida** (receptor): número de bytes que indica el receptor que puede recibir en el momento de enviar el paquete (= win).
- **Ventana utilizable** (emisor): número de bytes que se pueden enviar inmediatamente.
  - Ventana utilizable = win – (sig. byte enviar – último ACK).

Control de flujo: Ejercicio



Control de flujo: Ejercicio



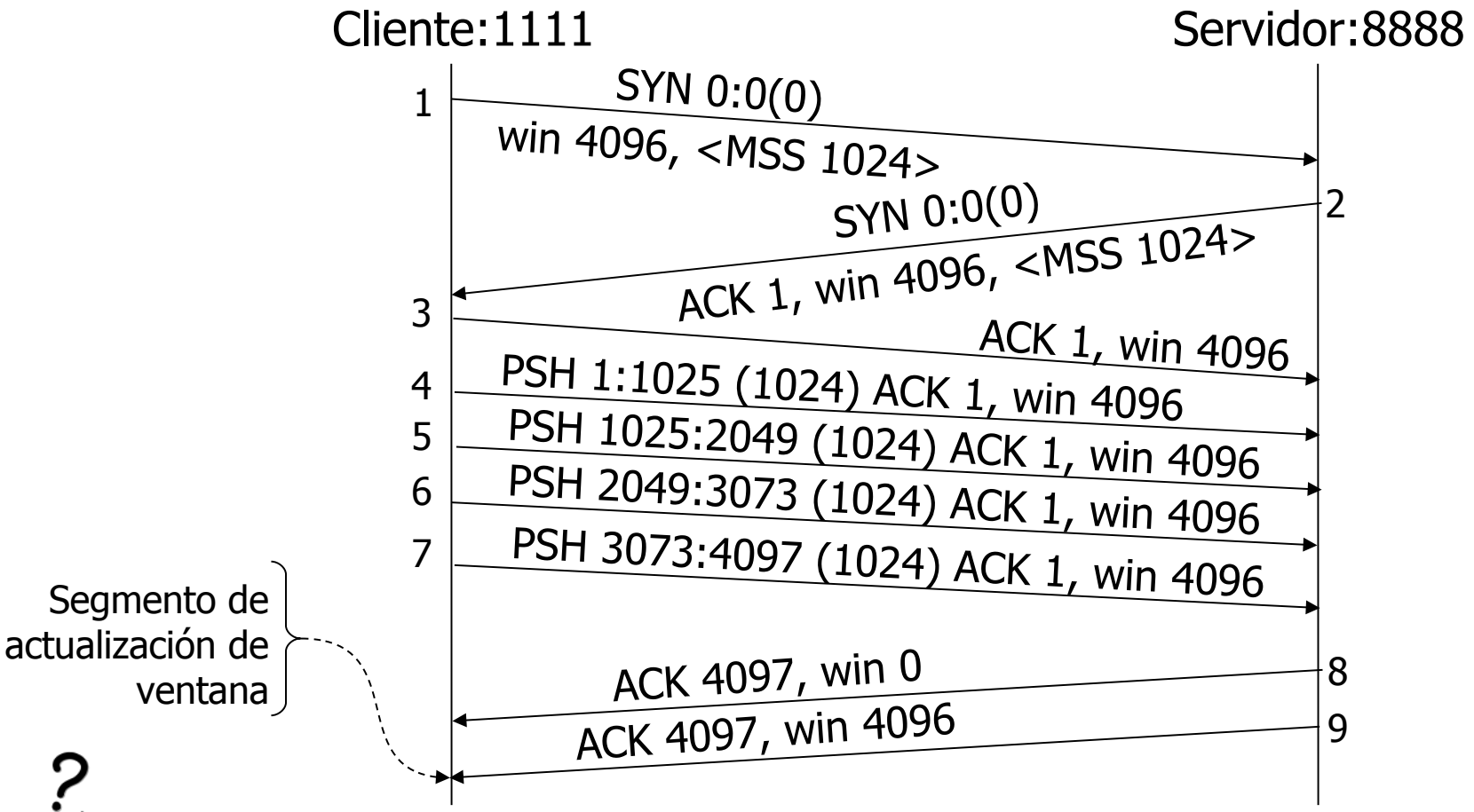
Control de flujo

- El envío de los mensajes TCP no es determinista, depende de múltiples factores: la carga de la red, la carga del receptor y emisor, …

- TCP utiliza ACKs acumulativos:
  - ACK 2049: confirma que se ha recibido correctamente hasta el 2048, y que el siguiente byte que se espera recibir es el 2049.
  - Segmentos 7, 14 y 16 son ACKs acumulativos.

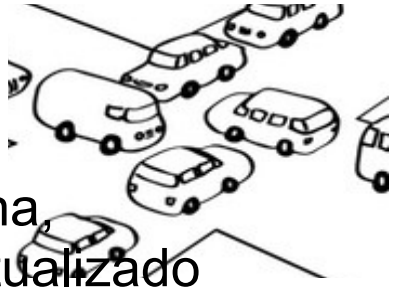
- El emisor no tiene por qué enviar una ventana completa de datos.
- El receptor no tiene que esperar a que se llene la ventana para enviar un ACK.

Control de flujo: Temp. de persistencia



- ¿Qué pasa si se pierde el segmento de actualización de ventana?

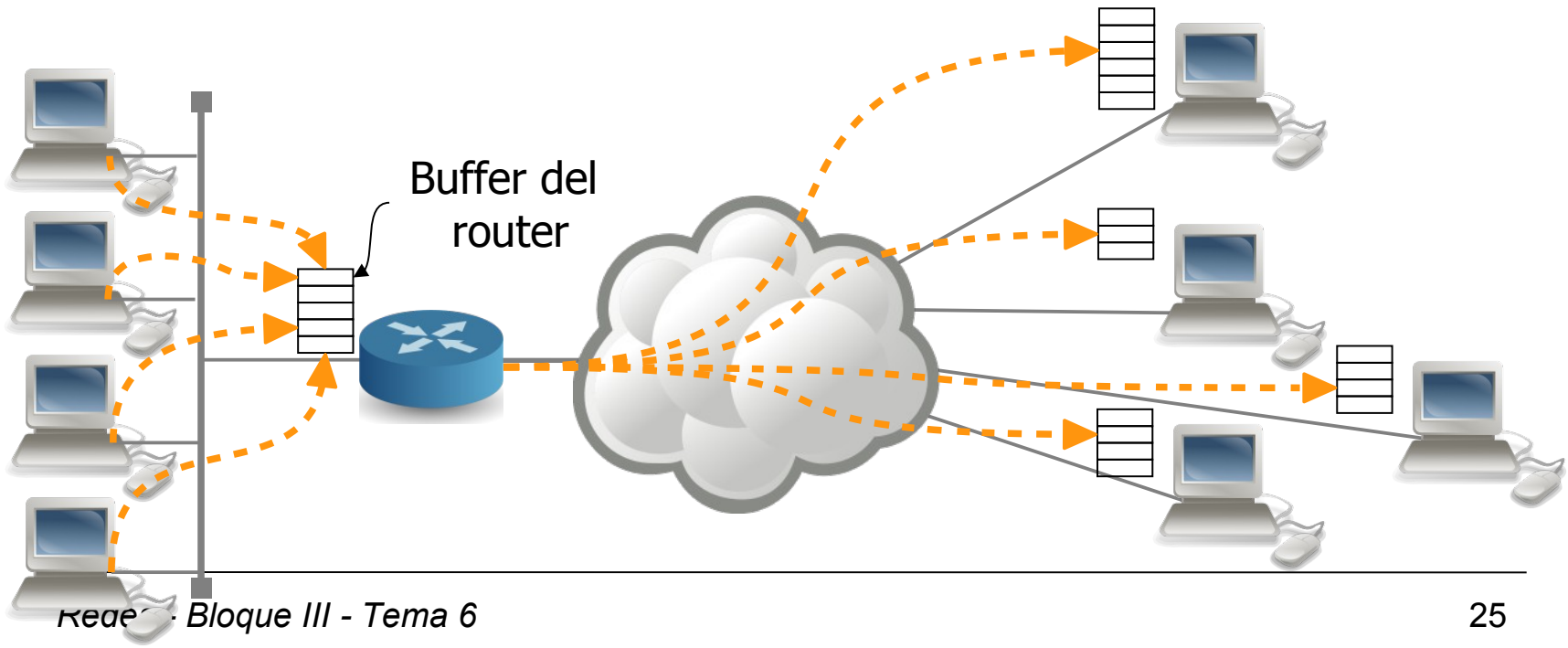
Control de flujo: Temp. de persistencia



- → Se entra en una situación de abrazo mortal.
- Solución: **Temporizador de persistencia**.
  - 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.
- **Window probes:** segmentos de un byte utilizados para comprobar si realmente la ventana se ha modificado o no.
  - Se envía el siguiente byte de datos.
  - El receptor puede confirmar o no el window probe (en función del espacio libre en el buffer).
- El temporizador se activa siguiendo un **exponential backoff**.
- Se continúa retransmitiendo hasta que:
  - El receptor abre la ventana.
  - Se cierra la conexión TCP por las aplicaciones.

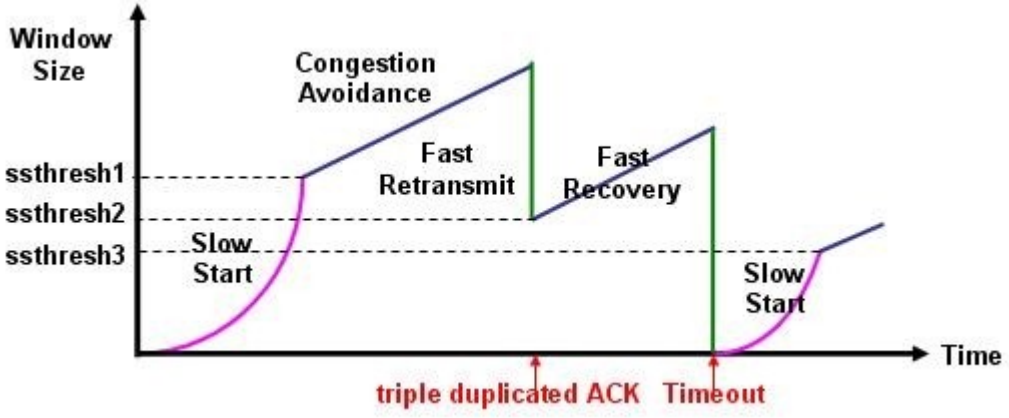
Control de congestión

- El control de flujo evita que el emisor llegue a saturar al receptor.
- Esto es correcto en una LAN, sin embargo en un entorno con routers intermedios el control de flujo no es suficiente → Los routers intermedios también se pueden saturar.
- Problema: los routers NO tienen nivel de transporte
- Solución: **Control de congestión**



Control de congestión

- Algoritmo para evitar la congestión → Se establece el **umbral de inicio lento**.
  - Por debajo del umbral, se aplica el algoritmo de inicio lento: "La velocidad a la que se inyectan paquetes nuevos a la red es la velocidad a la que se reciben ACKs del otro extremo"
  - Por encima, se aplica un crecimiento suavizado.



- http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/fairness/index.html

Control de congestión

- ¿Y qué pasa si algo va mal?

  1º: ¿Qué es que algo vaya mal? Cuando se pierde un paquete.
  - Dos posibilidades para “perder” un paquete:
    - Saturación en un router
    - Error en el paquete (probabilidad mucho menor del 1%)
  - Se asume que cuando se pierde un paquete es debido a que, al menos un router, está saturado.

  2º: ¿Cómo sé que algo va mal?
  - Se utilizan dos indicadores para identificar un problema de congestión (pérdida de un paquete):
    - Ha vencido un timeout de retransmisión (Fast Recovery).
    - Se han recibido ACKs duplicados (Fast Retransmit).

  3º: ¿Qué hago cuando algo va mal?
  - Retransmitir.
  - Y reducir la velocidad de transmisión.

- Todo esto se integra en el **algoritmo para evitar la congestión**.

Temporizador de keepalive

- En una conexión TCP sin intercambio de datos, no se produce ningún intercambio de paquetes (polling) → Problema en situaciones de fallos de uno de los extremos (normalmente el cliente).
- Solución: **temporizador de keepalive** → Mantiene la conexión activa (aunque no es parte del RFC de TCP).
- Permite liberar recursos al otro extremo, si realmente está desconectado.
- Funcionamiento: después de 2 horas de inactividad, el servidor enviará una sonda keepalive (similar al window probe).
  - Sonda keepalive: segmento de un byte, correspondiente al último byte enviado.
- El otro extremo puede estar en cuatro estados: ok, caído, reiniciado o no alcanzable.
- Se controla con (en /proc/sys/net/ipv4): tcp_keepalive_time (2h), tcp_keepalive_intvl (75 segs), tcp_keepalive_probes (9).