

Módulo 03

La Capa de Transporte

(Pt. 3)



Redes de Computadoras
Depto. de Cs. e Ing. de la Comp.
Universidad Nacional del Sur



Copyright

- Copyright © **2010-2022** A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU** Free Documentation License, versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página <http://www.gnu.org/copyleft/fdl.html>
- La versión transparente de este documento puede ser obtenida de la siguiente dirección:

<http://cs.uns.edu.ar/~ags/teaching>



Contenidos

- Servicios y protocolos de la capa de transporte
- Multiplexado y demultiplexado de segmentos
- Transporte no orientado a la conexión (**UDP**)
- Teoría de transporte confiable de datos
- Transporte orientado a la conexión (**TCP**)
- Establecimiento y cierre de conexiones
- Teoría de control de congestión
- Control de congestión en **TCP**



Transmission Control Protocol

- Es un protocolo **punto a punto**
- Permite intercambiar un **flujo de bytes**
- Implementa una **operatoria en pipeline**
 - Los mecanismos de control de flujo y de congestión determinan el tamaño de la ventana deslizante
- Puede requerir hacer uso de **almacenamiento intermedio** ya sea al enviar o al recibir
 - Suele ser obligatorio en el emisor pero llamativamente es opcional en el receptor



Transmission Control Protocol

- Posibilita la **transferencia bidireccional de datos**
 - Una misma conexión permite enviar y recibir datos.
 - El parámetro **MSS** denota el tamaño máximo de segmento aceptado por una dada implementación
- Es **orientado a la conexión**
 - Requiere una **fase previa de inicialización** antes de comenzar con el intercambio de información
- Implementa **control de flujo**
 - El emisor nunca satura de datos al receptor



Estructura de un segmento

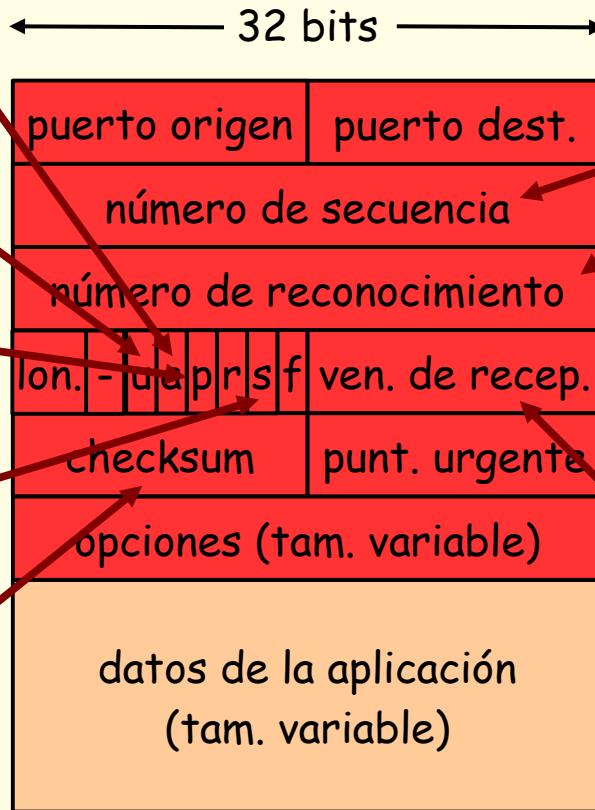
ACK denota que el nro. de reconocimiento es válido

URG permite marcar a los datos como urgentes

PSH solicita se procesen los datos del buffer

RST, SYN, FIN se utilizan para establecer y finalizar conexiones.

checksum
(análogo a **UDP**)



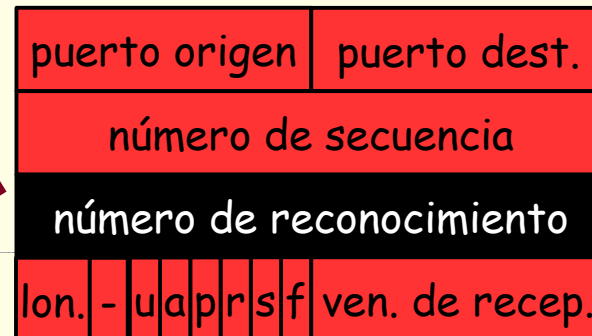
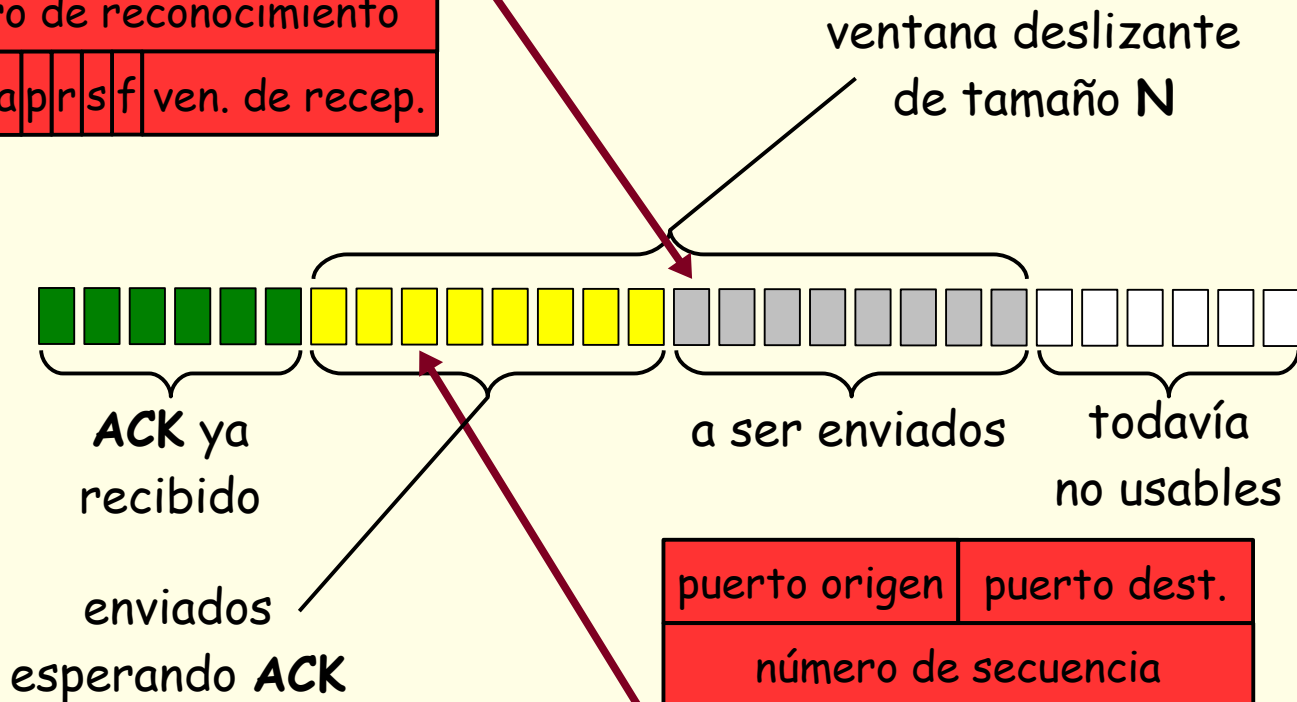
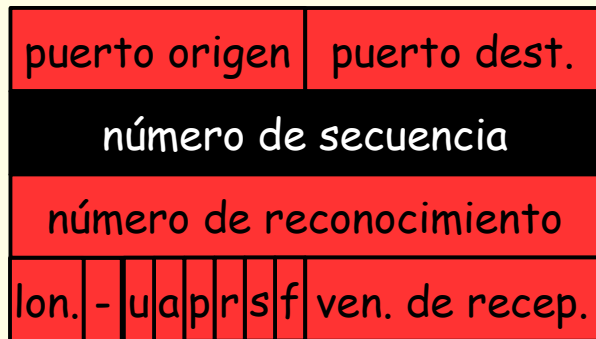
se cuenta a nivel de bytes, no a nivel de paquetes

campo para publicitar el tamaño de la ventana de recepción

formato de un segmento **TCP**



Secuencia y reconocimiento

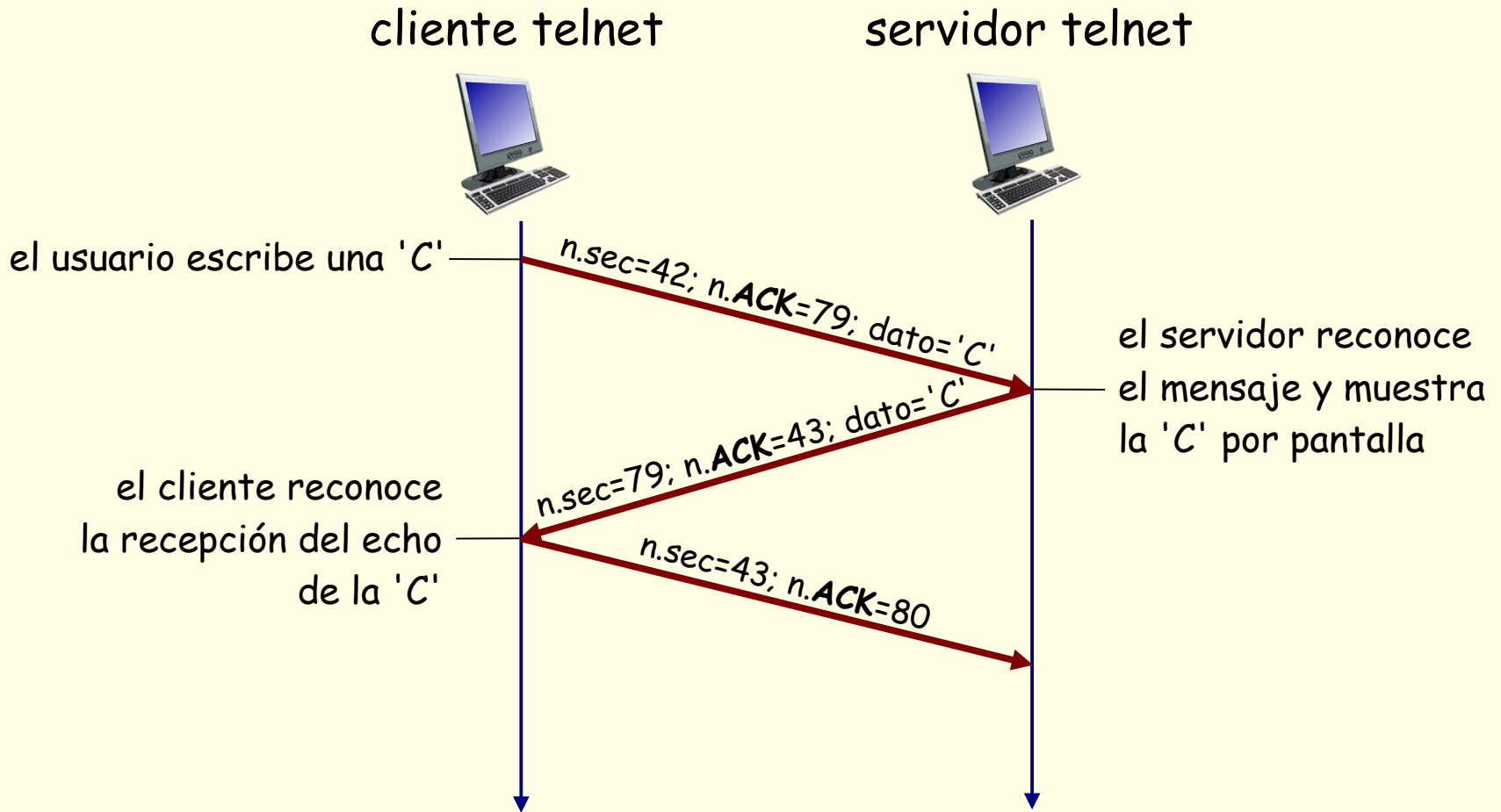


Secuencia y reconocimiento

- El **número de secuencia** de un segmento indica la posición del byte transferido dentro del flujo de bytes de la conexión
- El **número de reconocimiento** indica el próximo número de secuencia esperado del otro lado
 - **TCP** hace uso de **reconocimientos acumulativos**, esto es, al reconocer un cierto número de secuencia se reconocen implícitamente todos los anteriores
- La especificación formal nada dice acerca de qué hacer con los segmentos fuera de orden



Secuencia y reconocimiento



Tiempo de espera razonable

- ¿Qué valor resulta conveniente adoptar como tiempo de espera razonable (**TCP-timeout**)?
 - Al menos tiene que ser mayor que el **RTT**
 - Pero el **RTT** es un valor dinámico, el cual cambia en el tiempo
 - Un valor excesivamente bajo va a causar reenvíos prematuros absolutamente innecesarios
 - Pero un valor demasiado alto hace que sea muy costoso recuperarse ante una pérdida



Estimación del RTT

- ¿Cómo se puede estimar el valor del **RTT**?
 - ➔ Cada vez que se recibe un **ACK** como respuesta a un mensaje enviado previamente se puede obtener una nueva estimación del **RTT**
 - ➔ Es conveniente no hacer uso de las retransmisiones a la hora de estimar el **RTT**. ¿Por qué será?
 - ➔ La estimaciones obtenidas posiblemente oscilarán a lo largo del tiempo, por lo que hace falta concebir algún mecanismo que permita sopesar múltiples estimaciones



TCP RTT

- Para obtener una estimación adecuada del **RTT** se hace uso del alisado exponencial **EMA** (Exponentially-weighted Moving Average)

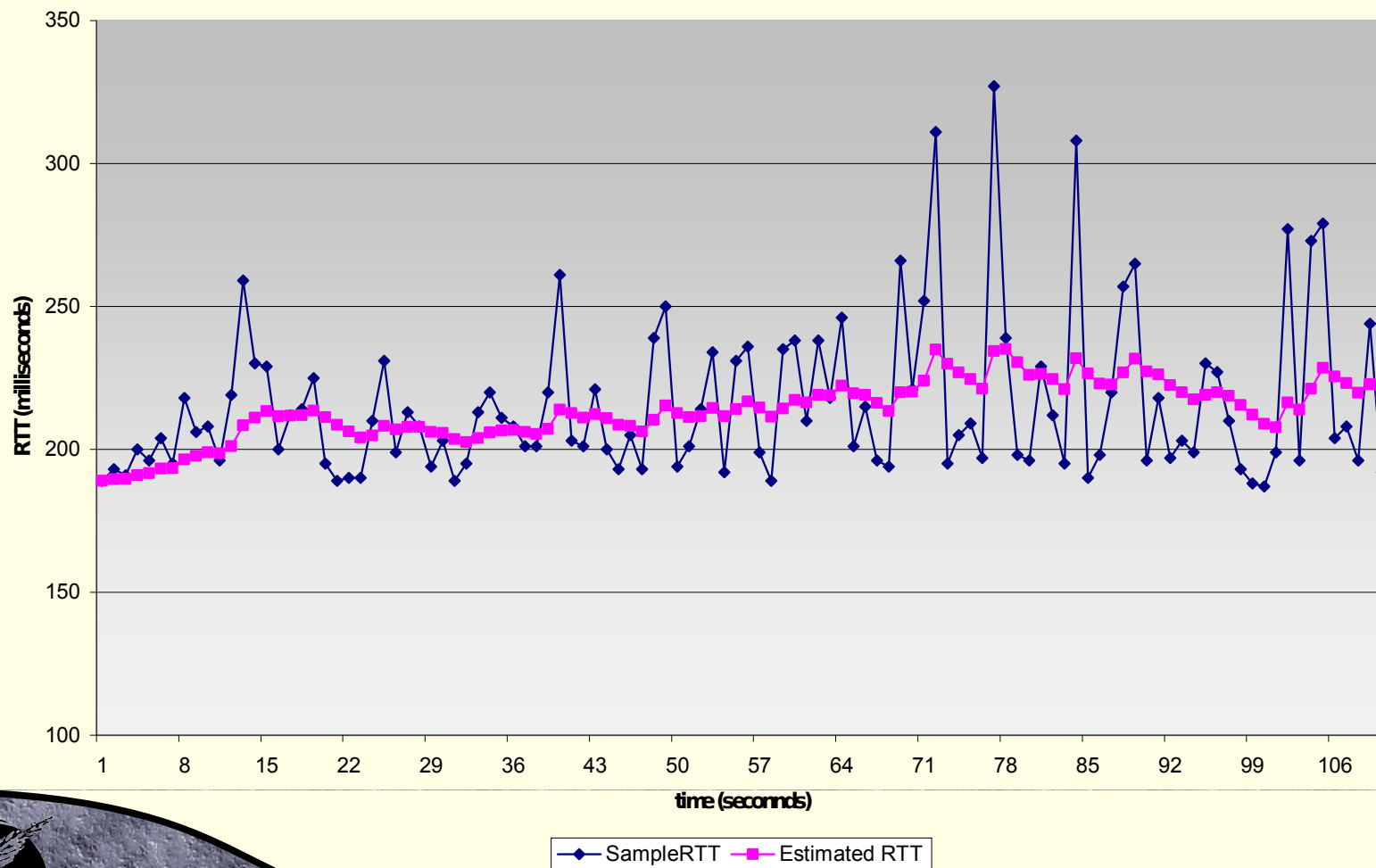
$$\text{RTT-estimado} = (1 - \alpha) \times \text{RTT-estimado} + \alpha \times \text{RTT-observado}$$

- **EMA** prioriza la última medición del **RTT** y penaliza cada vez mas a las mediciones anteriores de forma exponencial
 - Un valor usual para α es **0.125**



RTT observado vs. estimado

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



TCP timeout

- Una vez obtenida la estimación del **RTT** es posible elegir un valor adecuado de timeout
 - Al valor estimado de **RTT** se le debe incorporar un margen de seguridad adicional
 - Ante grandes cambios en el valor estimado, el margen de seguridad debería incrementarse

$$\text{desvmed-RTT} = (1 - \beta) \times \text{desvmed-RTT} + \beta \times | \text{RTT-observado} - \text{RTT-estimado} |$$

- Un valor usual para β es **0.25**

$$\text{TCP-timeout} = \text{RTT-estimado} + 4 \times \text{desvmed-RTT}$$



Transmisión confiable

- **TCP** implementa un canal de comunicación confiable por encima del canal no confiable provisto por la capa de red
 - Hace uso de las técnicas exploradas en la familia de protocolos **RDT** y de los protocolos **GBN** y **SR**
 - Usa un único temporizador para controlar las retransmisiones producto de las pérdidas
 - También se disparan retransmisiones al recibir mensajes de **ACK** duplicados



Emisor TCP (simplificado)

- Al recibir nuevos datos de la aplicación:
 - ➔ Crea un segmento con el número de secuencia que corresponda (esto es, el desplazamiento del primer byte a ser transmitido a través del flujo de bytes)
 - ➔ Programar el temporizador, en caso de no estar ya corriendo, usando el **TCP-timeout** antes calculado
- Al dispararse la alarma de temporizado:
 - ➔ Retransmitir el segmento que expiró
 - ➔ Reprogramar el temporizador



Emisor TCP (simplificado)

● Al recibir un mensaje de ACK:

- Si se trata de un mensaje de **ACK** asociado a un segmento para el cual se estaba esperando confirmación, marcar el segmento como confirmado y reiniciar el temporizador sólo en caso de contar con otros segmentos para los cuales se está aún esperando confirmación
- Por el momento asumiremos como simplificación que no se reciben mensajes de **ACK** por duplicado
- También ignoraremos el control de flujo y la gestión de las congestiones (por ahora)



Emisor TCP (simplificado)

“datos” recibidos de la aplicación

```
“segmento TCP” =  
    empaq(proxnumsec, “datos”, checksum);  
    enviar(“segmento TCP”);  
    proxnumsec = proxnumsec + largo(“datos”);  
    if (!“alarma puesta”)  
        poner(alarma);
```

λ

```
proxnumsec = primernumsec  
base = primernumsec
```



disparo(alarma)

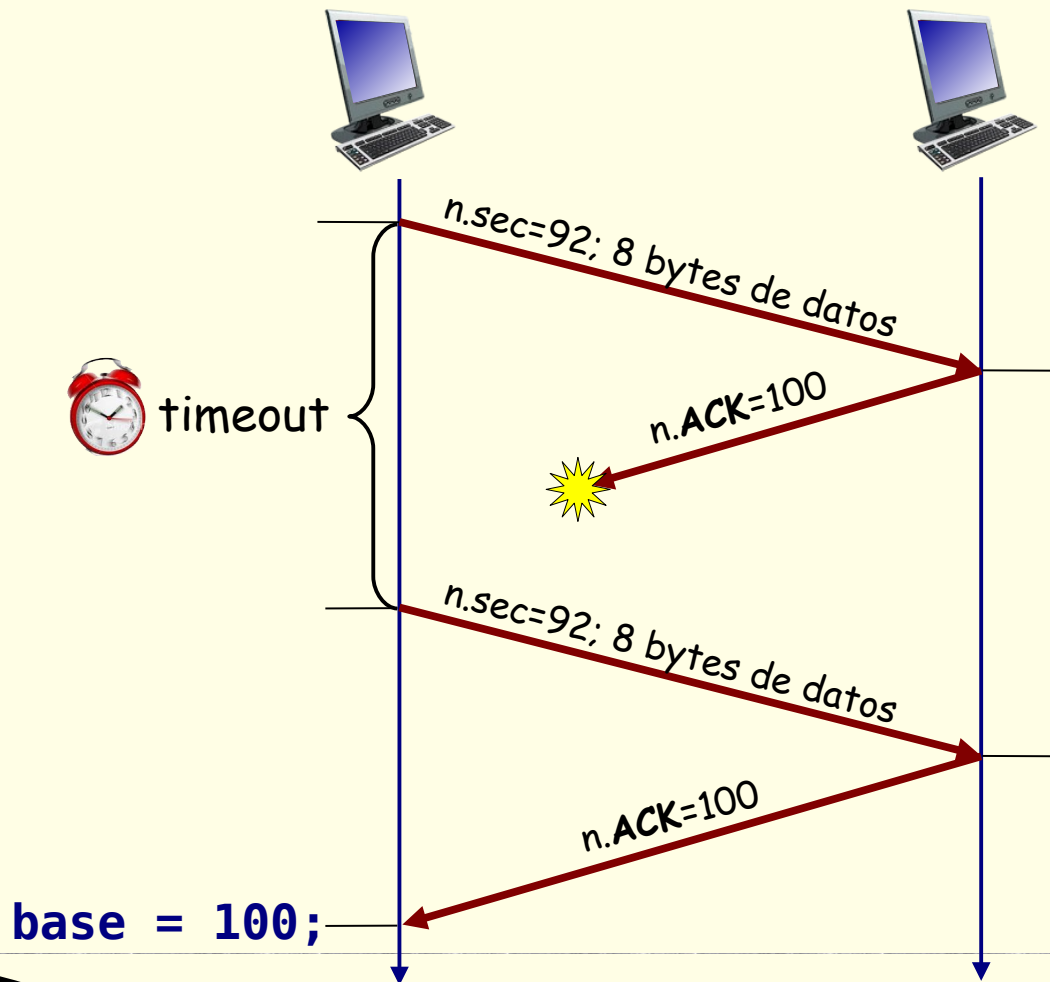
```
reenviar el segmento aún-  
no reconocido con menor-  
nro. de sec.;  
poner(alarma);
```

mensaje de ACK recibido
con nro. de ACK x

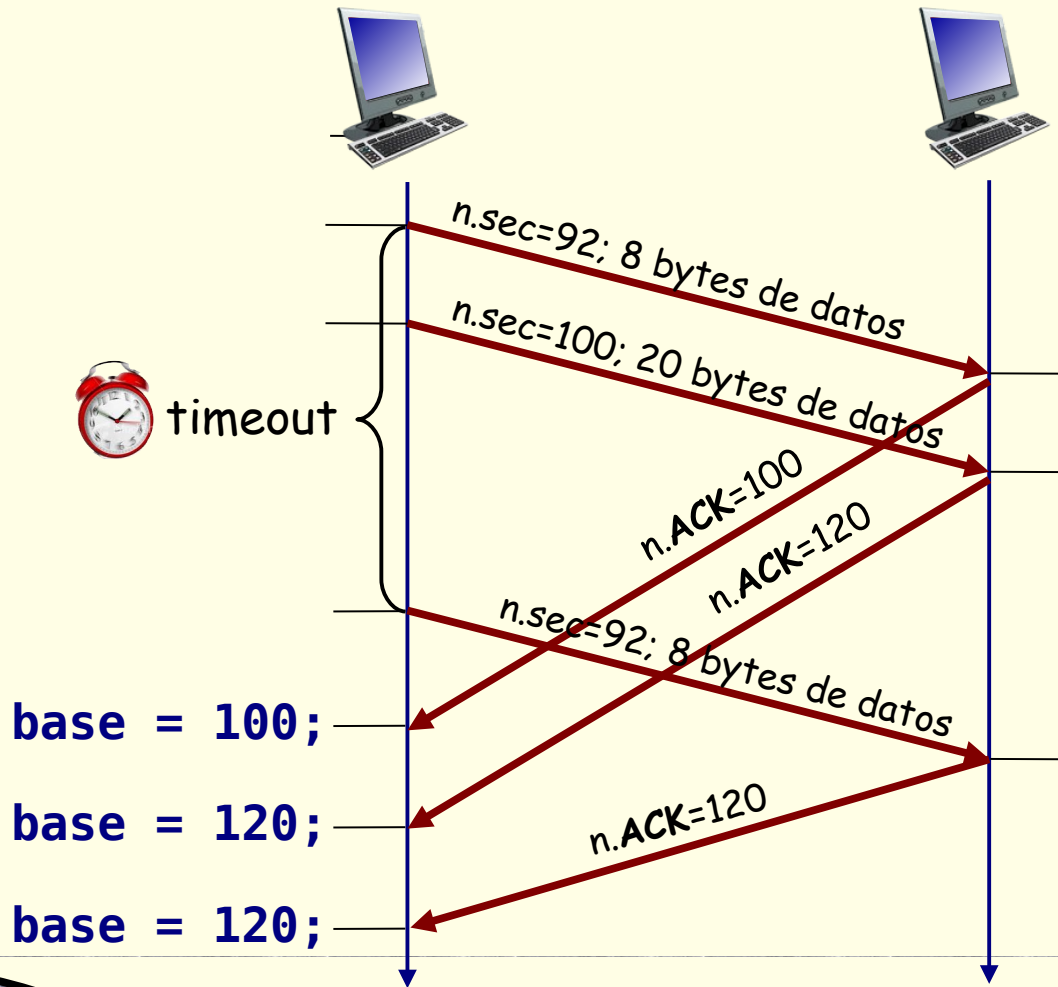
```
if (x > base)  
    base = x;  
    if (“quedan segmentos-  
        no reconocidos”)  
        poner(alarma);
```



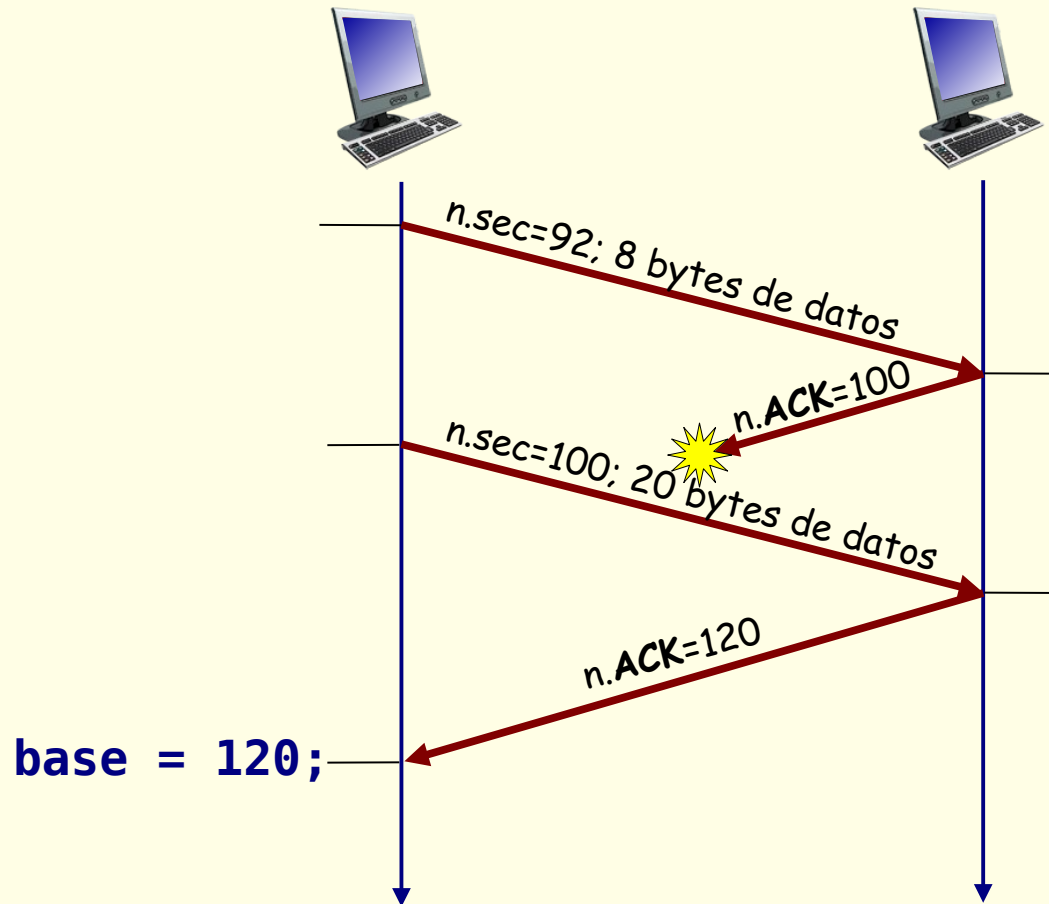
Pérdida de un ACK



Timeout prematuro



Reconocimiento acumulativo



Ajuste fino del temporizador

- La pérdida de segmentos se debe usualmente a la congestión en alguno de los routers del núcleo de la red
- Al producirse la retransmisión de un segmento a consecuencia del disparo del temporizador debemos tener esto en consideración:
 - ➔ Al volver a programar el temporizador a consecuencia de la retransmisión se debe **duplicar su duración**
 - ➔ Es decir, se dejará momentáneamente de lado el mecanismo de estimación basado en el **RTT**



Generación de confirmaciones

- Al llegar un segmento en orden, esto es, con el número de secuencia esperado, y además si ya se han confirmado todos los bytes anteriores del flujo de bytes:
 - Este caso se conoce como “**confirmación demorada**” (delayed **ACK**), pues se debe esperar hasta 500ms para ver si se reciben más segmentos
 - De no aparecer segmento alguno, se envía la confirmación al acabarse el tiempo de espera



Generación de confirmaciones

- Al llegar un segmento en orden, esto es, con el número de secuencia esperado, pero con la salvedad de que el último segmento aún no fue confirmado:
 - ➔ Enviar inmediatamente un único mensaje confirmando al mismo tiempo a ambos segmentos
 - ➔ Este caso se produce al llegar un segundo segmento mientras que con un segmento anterior se estaba ensayando una confirmación demorada



Generación de confirmaciones

- Al llegar un segmento fuera de orden el cual genera un hueco (faltan bytes en el medio):
 - ➔ Enviar inmediatamente un mensaje de confirmación duplicado indicando el número de secuencia del primer byte faltante
- Al llegar un segmento fuera de orden el cual completa parcial o totalmente el comienzo de un hueco anterior:
 - ➔ Enviar inmediatamente un mensaje de confirmación para la parte cubierta del hueco



Retransmisión anticipada

- Esperar hasta que se dispare la alarma asociada a un segmento implica tener que esperar una gran cantidad de tiempo
- El emisor podría darse cuenta de que se produjo algún inconveniente al detectar la recepción de confirmaciones duplicadas
 - ➔ El emisor suele enviar múltiples segmentos uno tras otro (mientras la ventana lo permita)
 - ➔ Al perderse un segmento llegarán múltiples mensajes de confirmación para el último segmento recibido

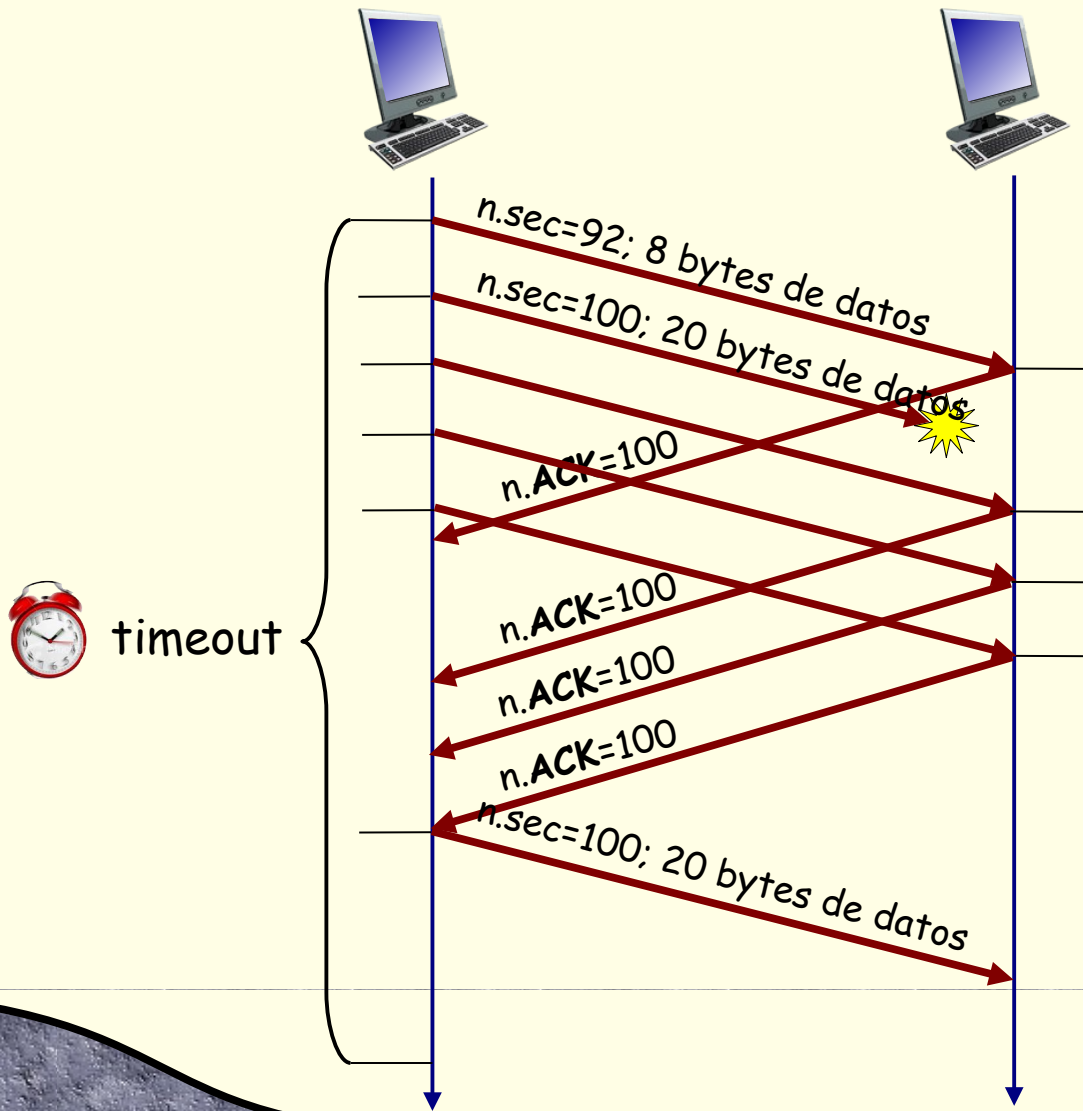


Retransmisión anticipada

- El emisor al observar **tres mensajes de confirmación repetidos** puede asumir que el segmento con ese número de secuencia se malogró
- La técnica de **retransmisión anticipada** (fast retransmit) consiste en retransmitir un cierto segmento antes de que se dispare la alarma del temporizador

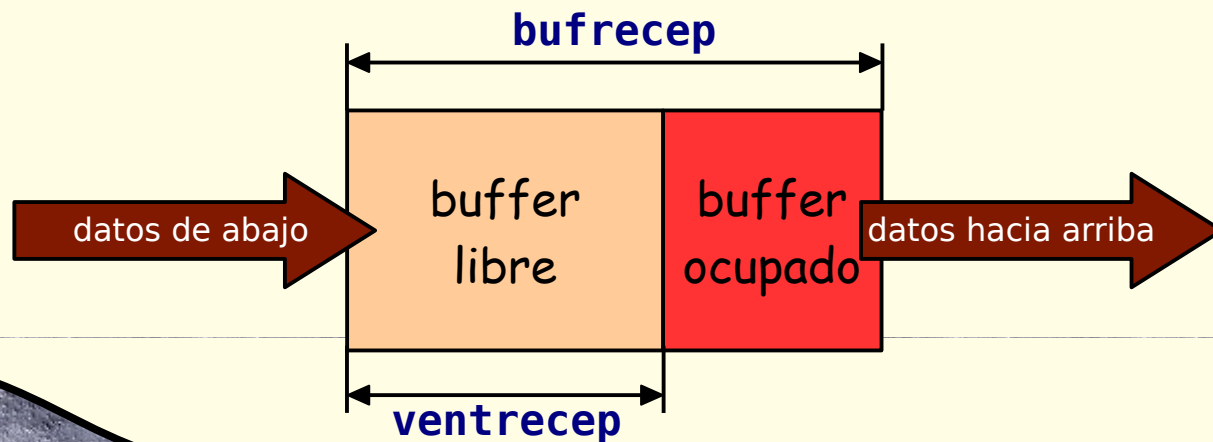


Retransmisión anticipada



Control de flujo en TCP

- El protocolo **TCP** estipula que el receptor cuente con un almacenamiento intermedio para alojar los segmentos que van llegando
 - La aplicación puede tomarse su tiempo en consumir los segmentos que van llegando
 - El control de flujo consiste en asegurar que el emisor no sature el almacenamiento intermedio del receptor



Control de flujo en TCP

- El receptor publicita el valor actual de la ventana de recepción **ventrecep**
 - En una implementación que descarte los segmentos recibidos fuera de orden el buffer libre se estima como **bufrecep – [ultbyterecep – ultbyteent]**
- El emisor limita la cantidad de paquetes enviados aún no reconocidos a ese valor
 - De esta forma se asegura que **nunca saturará al buffer del receptor**



Gestión de conexiones en TCP

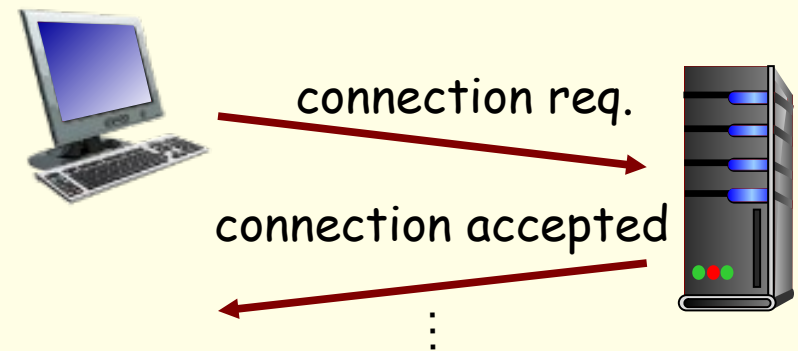
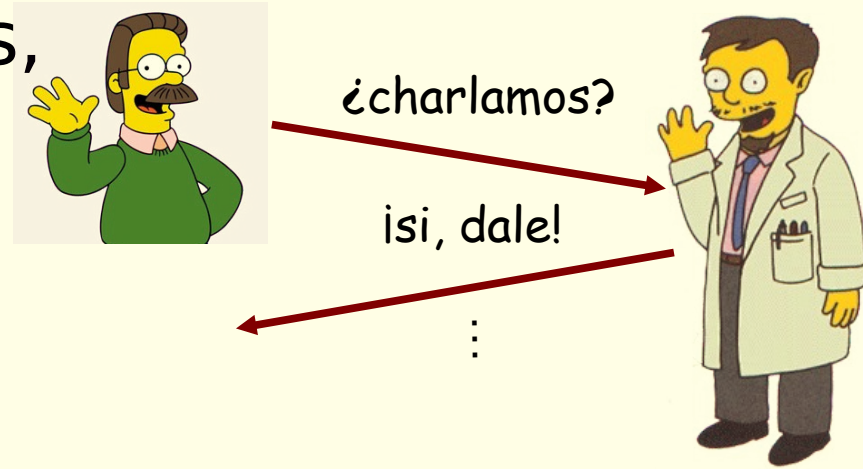
- Recordemos que **TCP** previo al comienzo del intercambio de segmentos debe establecer la conexión entre las partes involucradas
- Durante esta fase las partes acuerdan de forma mutua conectarse y deben inicializar las distintas variables **TCP** tales como:
 - ➔ Número de secuencia inicial (**base**, **proxnumsec**)
 - ➔ Almacenamientos intermedios (**bufrecep**)
 - ➔ Tamaño de la ventana de recepción (**ventrecep**)



Acuerdo mutuo

• Un protocolo de dos fases, ¿funcionará en una red de computadoras?

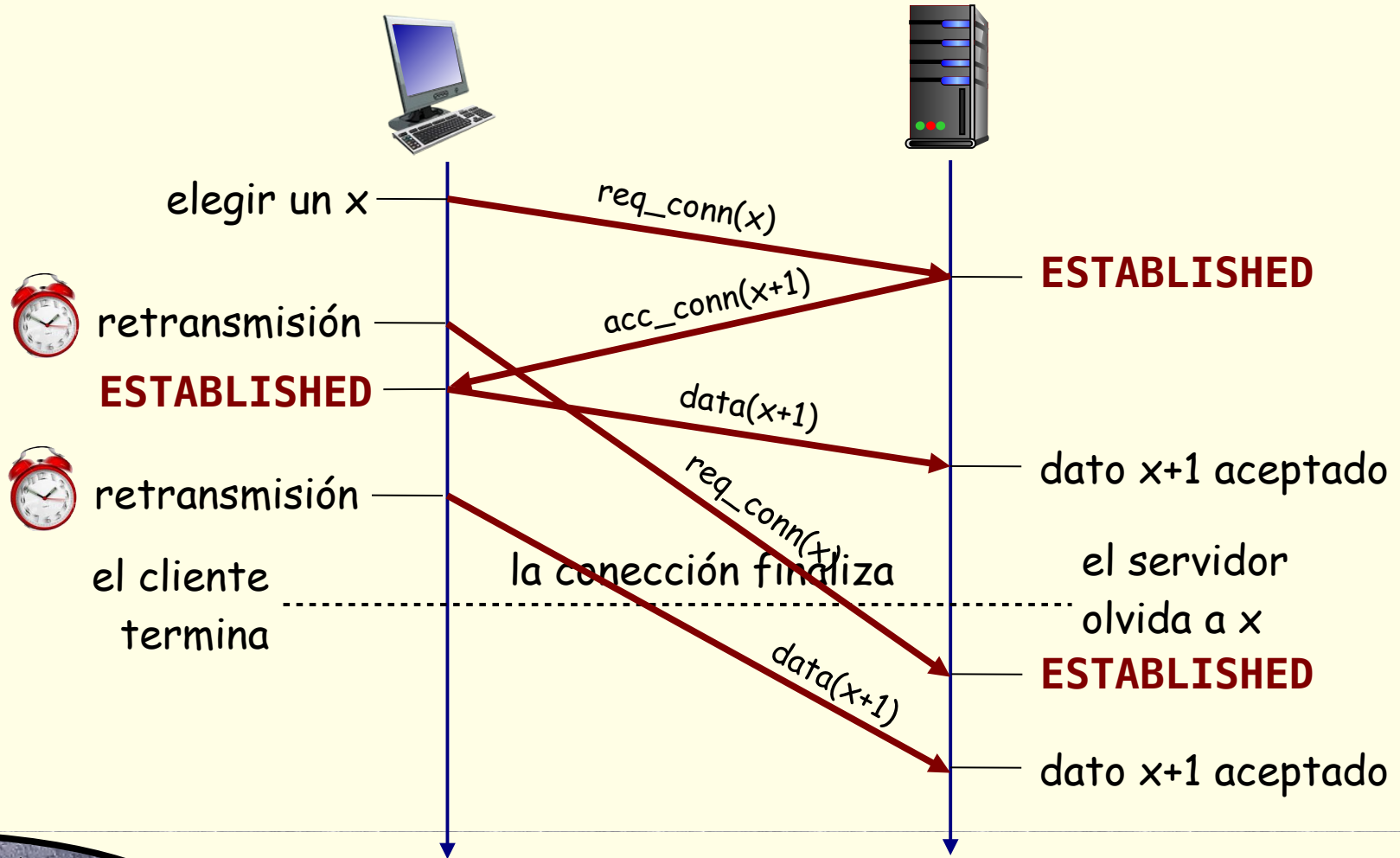
- Los retardos son variables
- Debemos contemplar eventuales retransmisiones
- Quizás los mensajes arriben fuera de orden



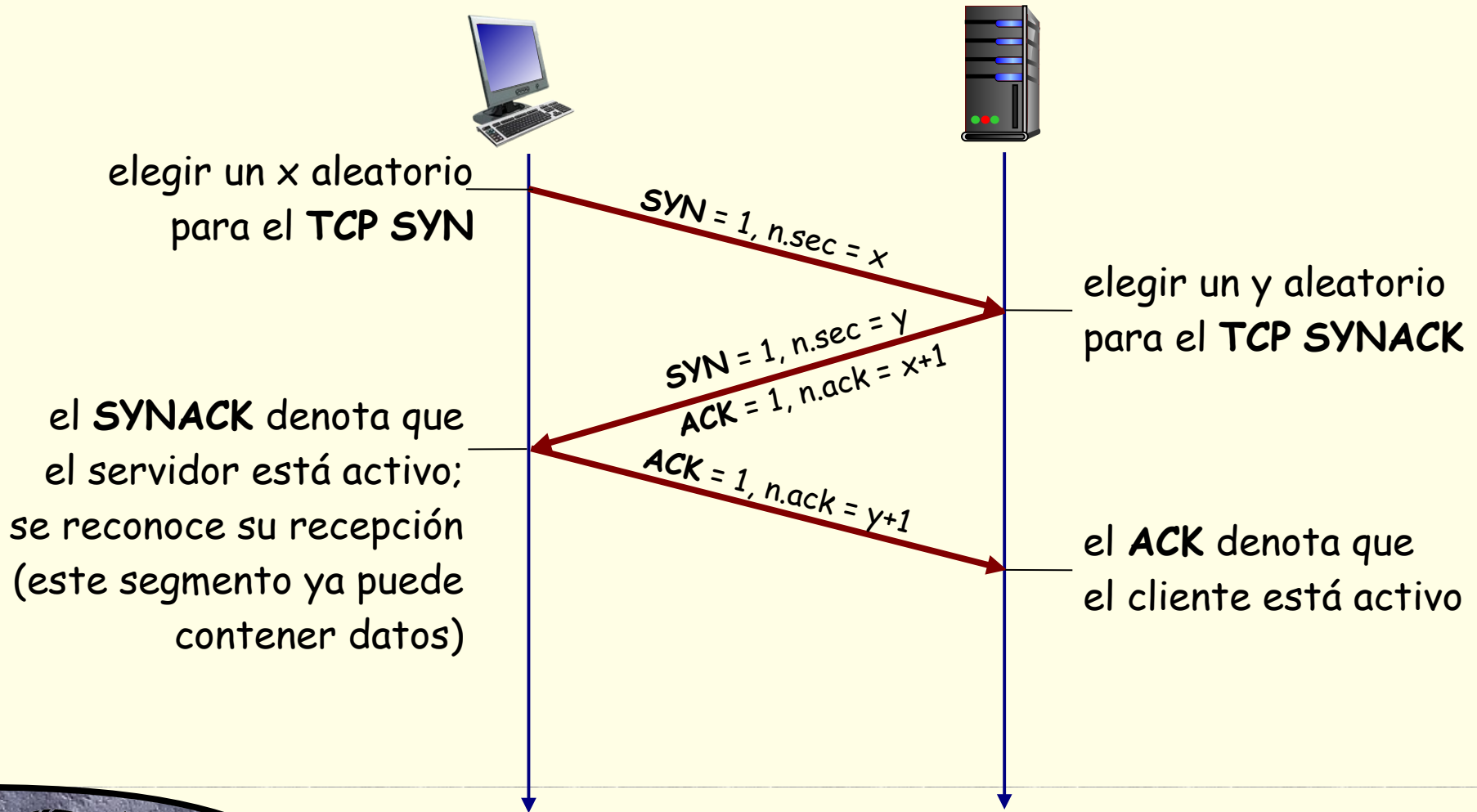
Saludos de dos fases



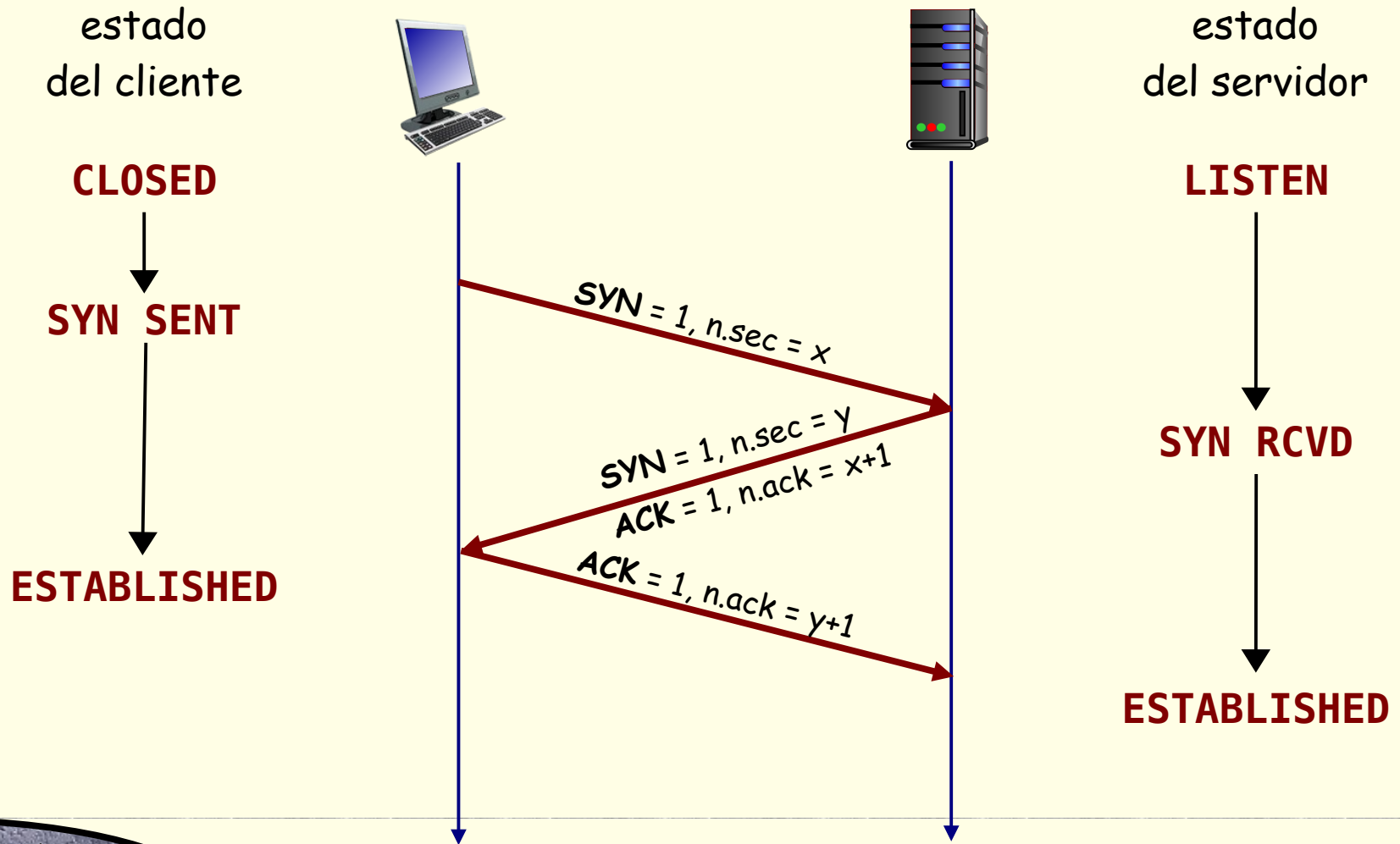
Saludos de dos fases



Saludo de tres fases



Saludo de tres fases



Saludo de tres fases

```
ServerSocket =  
    socket(AF_INET,SOCK_STREAM)  
serverSocket.bind((ip,port))  
serverSocket.listen(1)
```

λ

closed

```
ClientSocket =  
    socket(AF_INET,SOCK_STREAM)  
clientSocket.connect((ip,port))
```

$\text{SYN}(n.\text{sec}=x)$

$\text{SYN}(x)$

$\text{SYNACK}(n.\text{sec}=y, n.\text{ack}=x+1)$
se crea un nuevo socket para
comunicarse con ese cliente

listen

SYN
rcvd

SYN
sent

$\text{ACK}(n.\text{ack}=y+1)$

λ

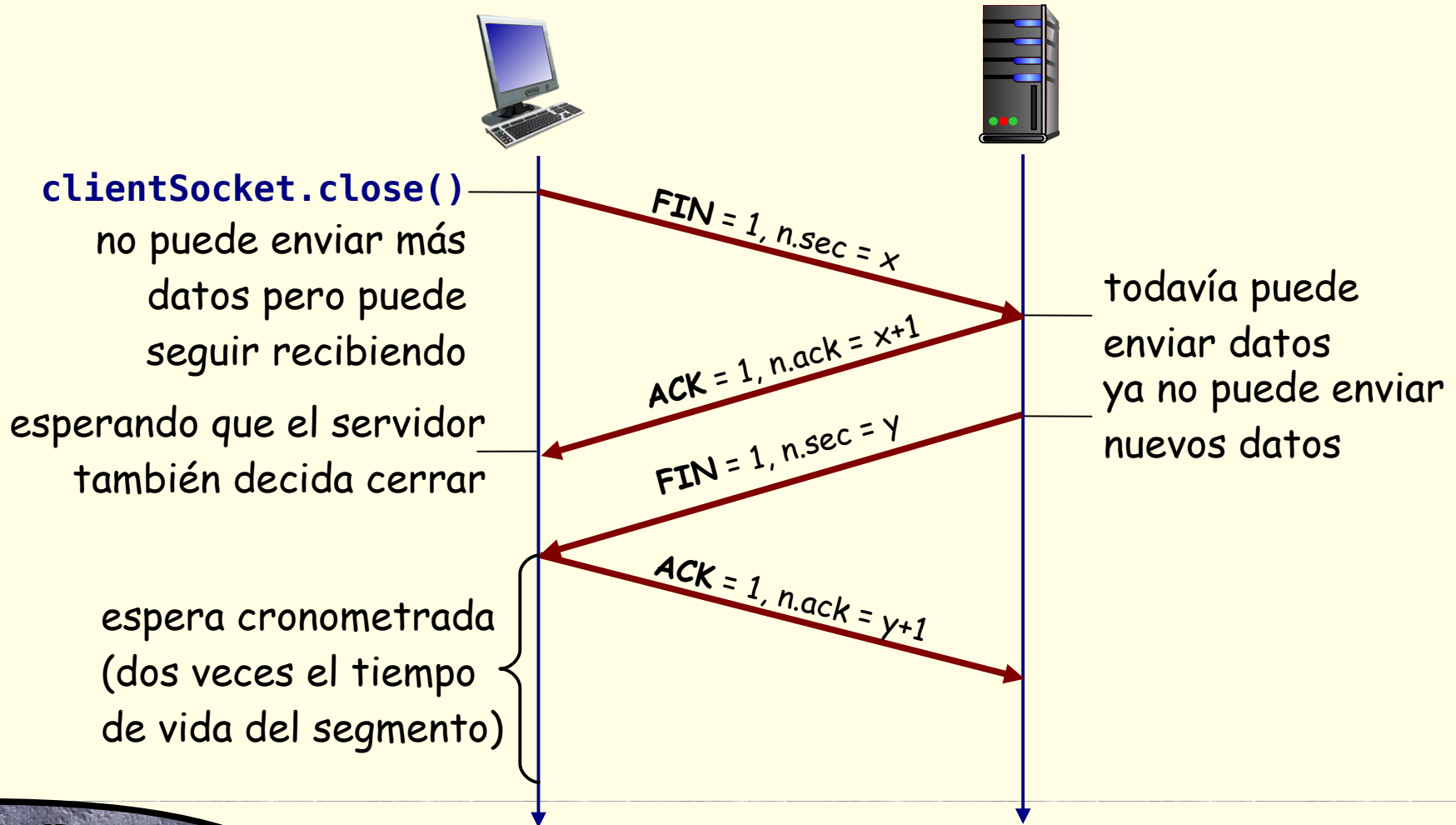
estab.

$\text{SYNACK}(n.\text{sec}=y, n.\text{ack}=x+1)$

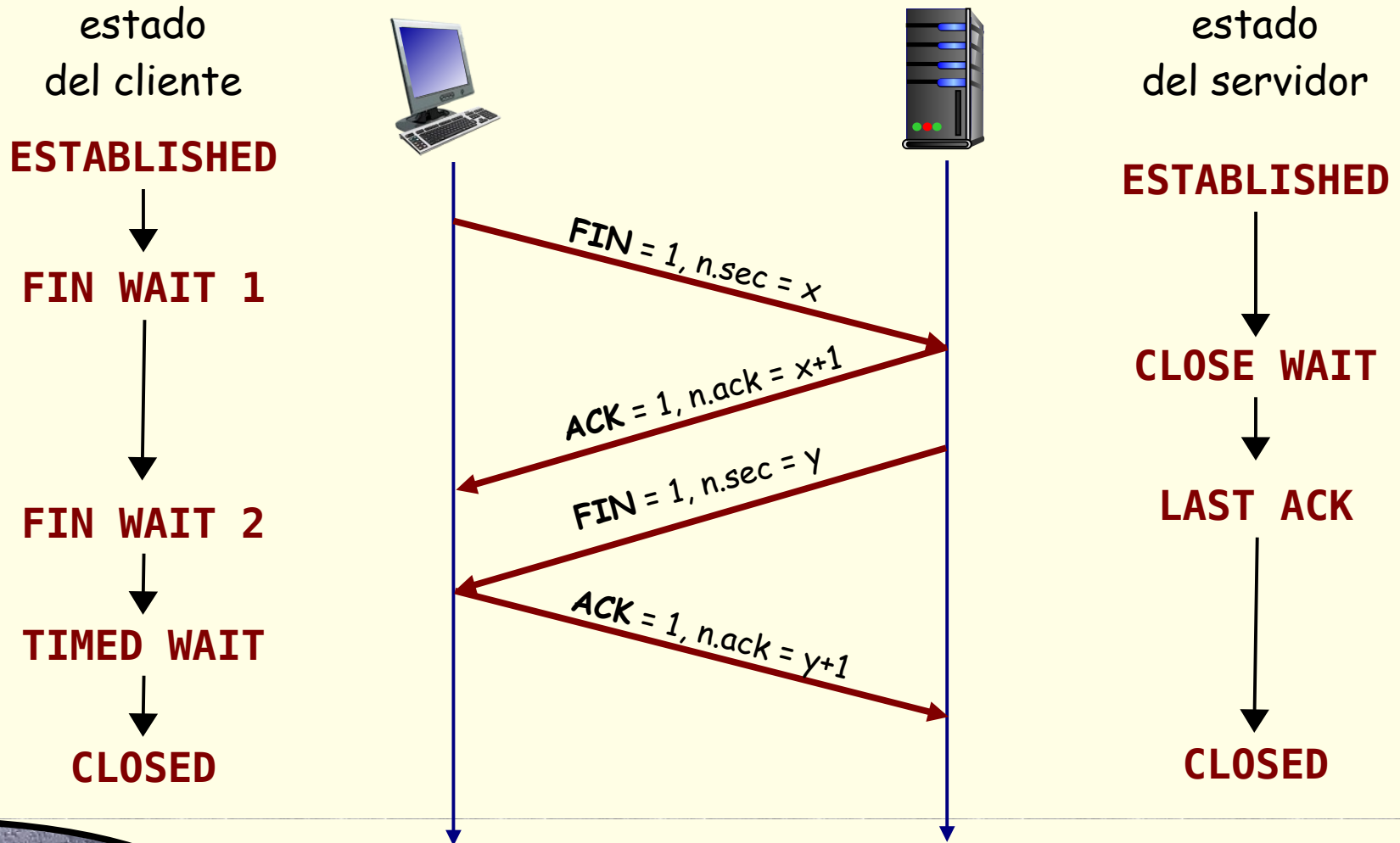
$\text{ACK}(n.\text{ack}=y+1)$



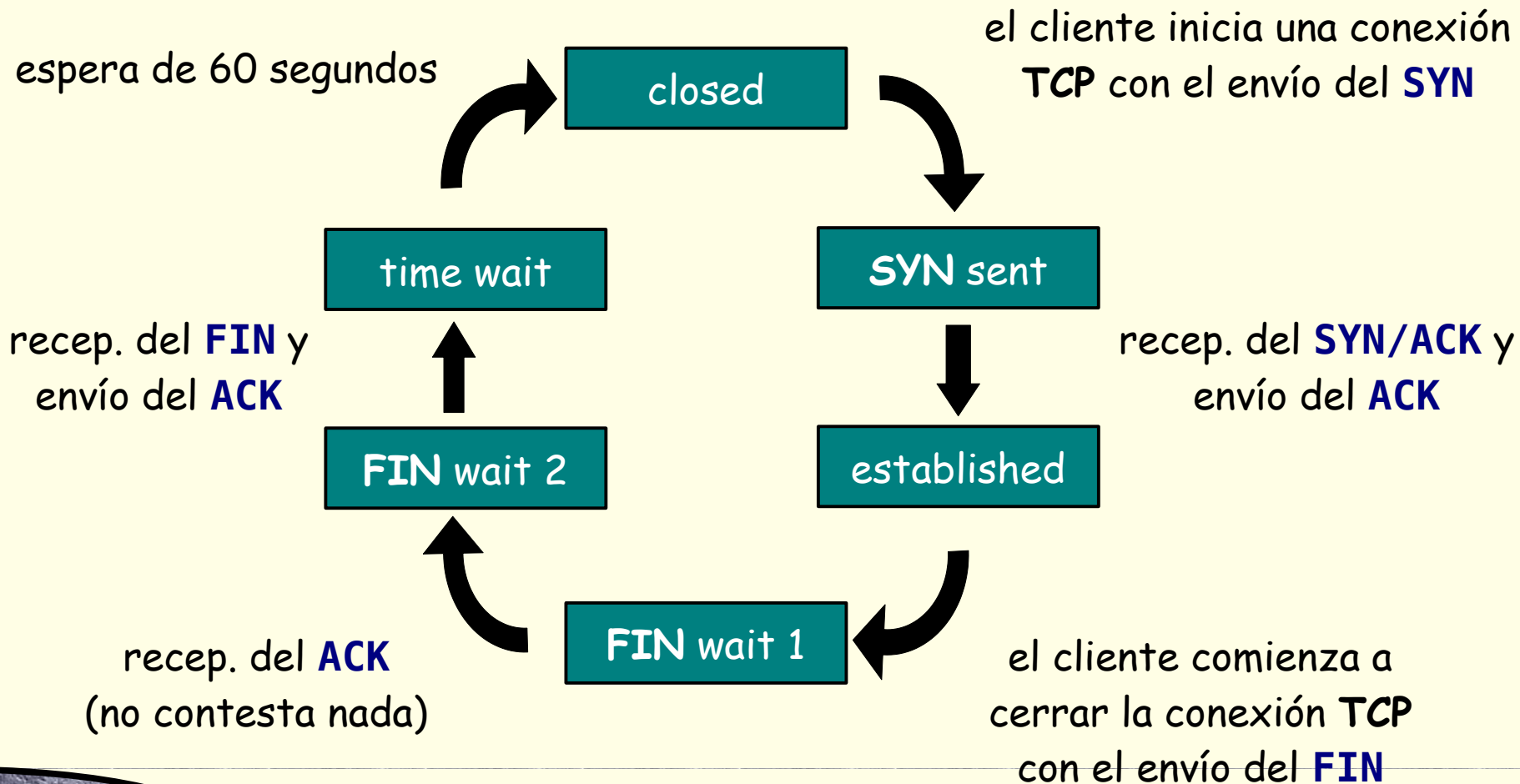
Terminación de una conexión



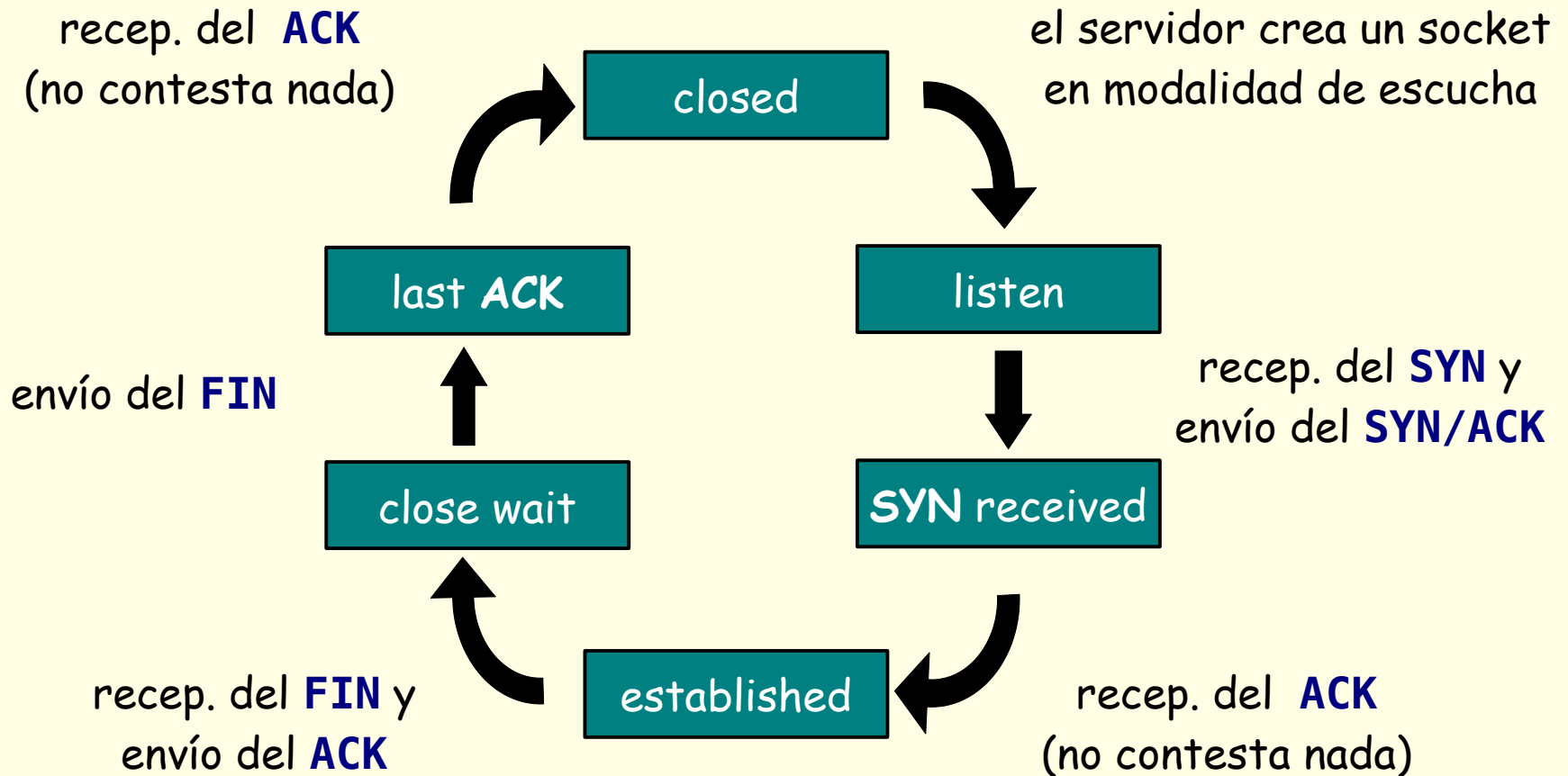
Terminación de una conexión



Síntesis (cliente)



Síntesis (servidor)



¿Preguntas?

