# Redes

## Tema 3: Capa de transporte

José Carlos Cabaleiro Domínguez

Escola Técnica Superior de Enxeñería

# Índice

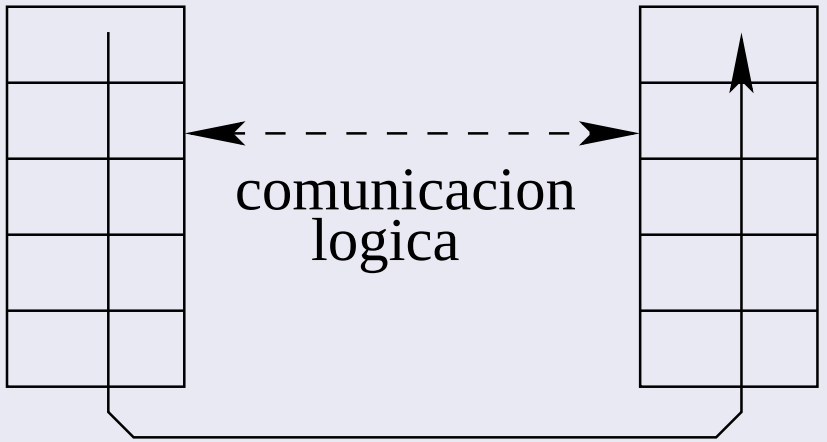
# Índice

# Introducción

## Capa de transporte

- Prepara los mensajes de las aplicaciones para ser transmitidos
- En destino, recupera los mensajes y los entrega a las aplicaciones
- Solo está implementada en los sistemas finales
- Proporciona una comunicación lógica entre procesos

Capa de aplicacion
Capa de transporte
Capa de red
Capa de enlace
Capa fisica



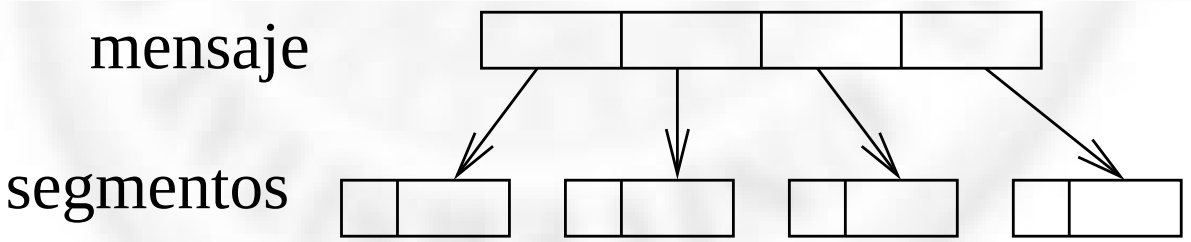
# Capa de transporte

## En TCP/IP

- Prepara los mensajes para transmitirlos por un canal no fiable (red de datagramas, IP, servicio de mejor esfuerzo)
- Protocolos de transporte TCP y UDP

## TCP en origen

- Fragmentar los mensajes en segmentos
- Añadirles la cabecera de la capa de transporte



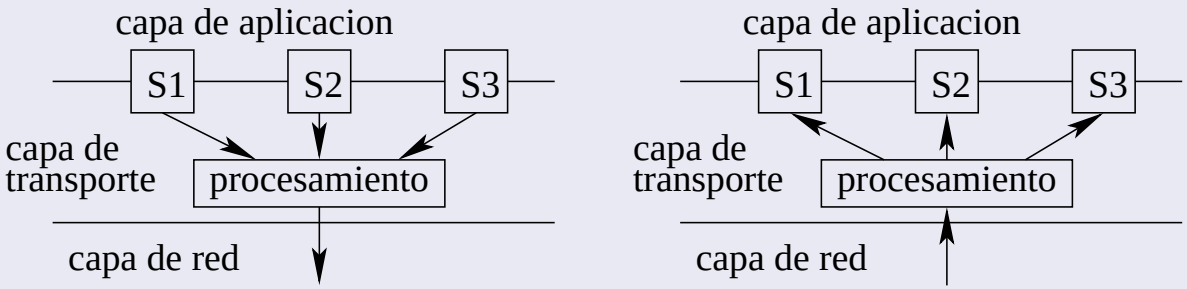

## UDP en origen

- Añadirles la cabecera de la capa de transporte

# Multiplexación y demultiplexación

## Interacción entre las aplicaciones y la capa de transporte

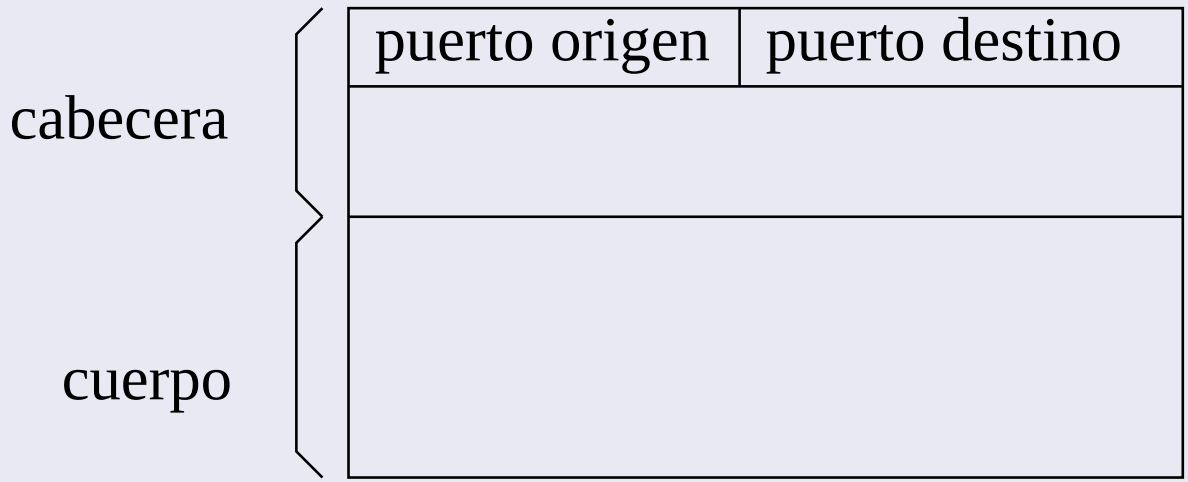
- Los mensajes pasan de la capa de aplicación a la capa de transporte a través de un socket
  - Los procesos escriben y leen del socket
  - La capa de transporte recoge los mensajes del socket y los traslada al socket destino
- Multiplexación: recorrer todos los sockets abiertos, procesar los mensajes y enviarlos a la capa de red
- Demultiplexación: recoger los segmentos que llegan de la capa de red, reconstruir los mensajes y colocarlos en los sockets destino



# Multiplexación y demultiplexación

## Interacción entre las aplicaciones y la capa de transporte

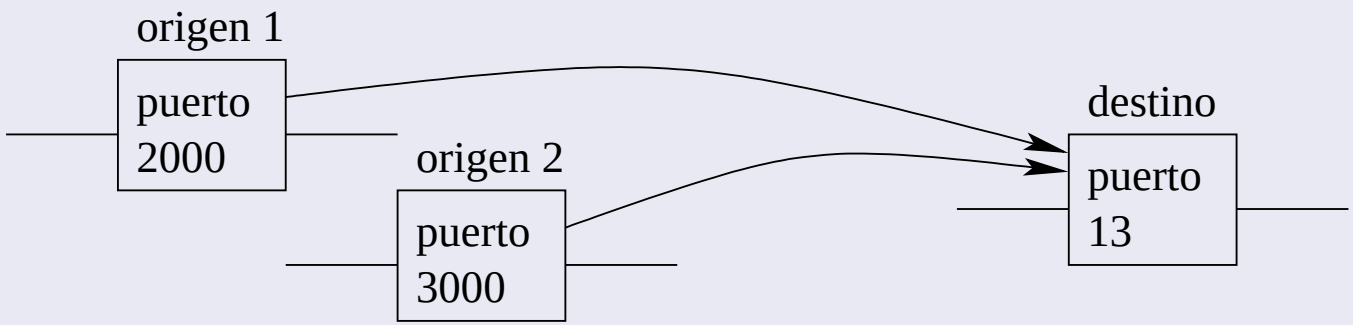
- Identificación del socket destino: a través de los números de puerto de la cabecera del segmento
  - Enteros de 16 bits
  - De 0–1023 usados por el administrador para los servicios bien conocidos



# Multiplexación y demultiplexación

## Multiplexación con sockets sin conexión (en UDP)

- Identificación del socket: la pareja dirección IP destino y puerto destino
- Segmentos de distintos hosts que se entreguen en el mismo puerto son recogidos por el mismo proceso
- El puerto origen se usa para que el proceso sepa a quién responder

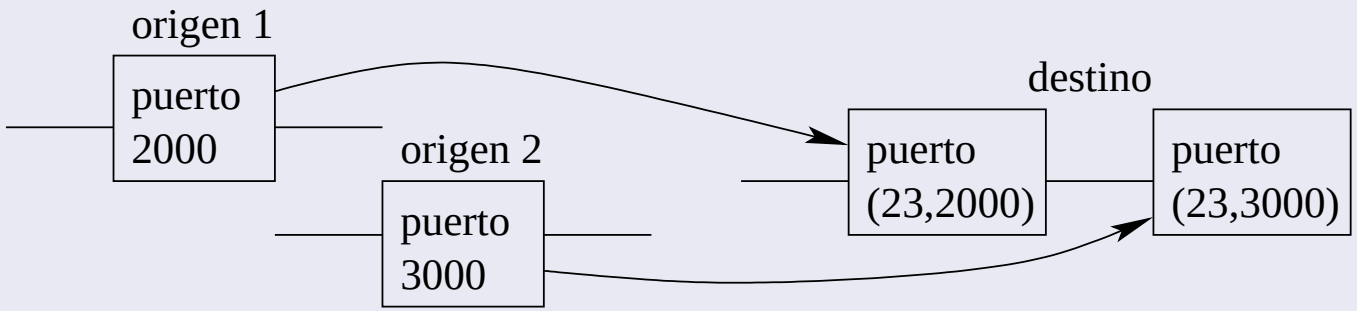


# Multiplexación y demultiplexación

## Multiplexación con sockets orientados a conexión (en TCP)

- Identificación del socket: la tupla de direcciones IP origen y destino y puertos origen y destino
- Segmentos de distintos hosts (o distintos puertos origen) que se entreguen en el mismo puerto van a sockets distintos

$\Longrightarrow$ varias conexiones al mismo puerto puedan ser atendidas por procesos o hilos diferentes

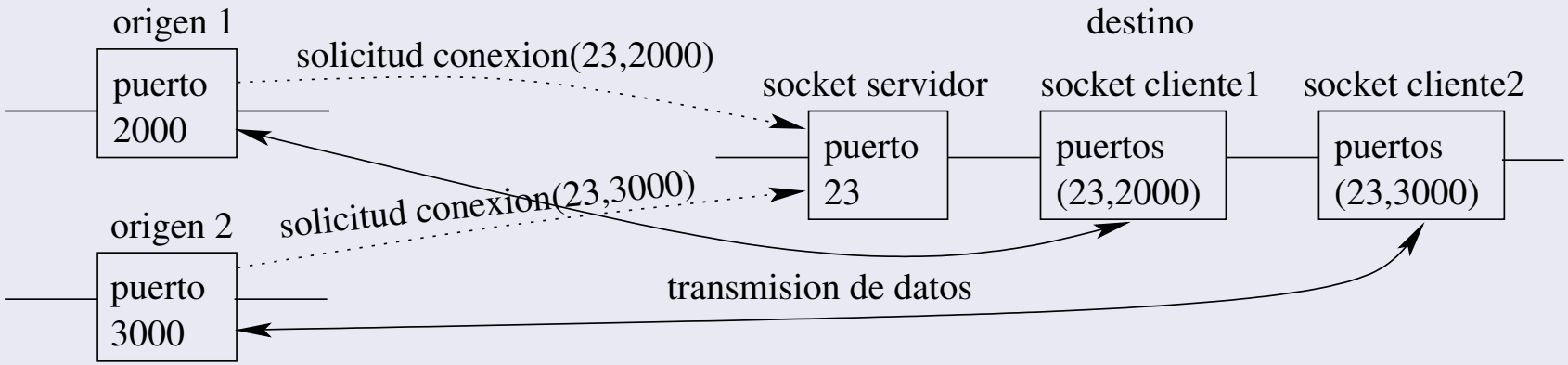


# Multiplexación con sockets orientados a conexión TCP

## Conexiones TCP

Dos tipos de sockets:

- Un socket de servidor: espera conexiones de clientes
- Varios sockets de conexión: se encargan de la transmisión de datos



# Índice

# Transporte no orientado a conexión: UDP

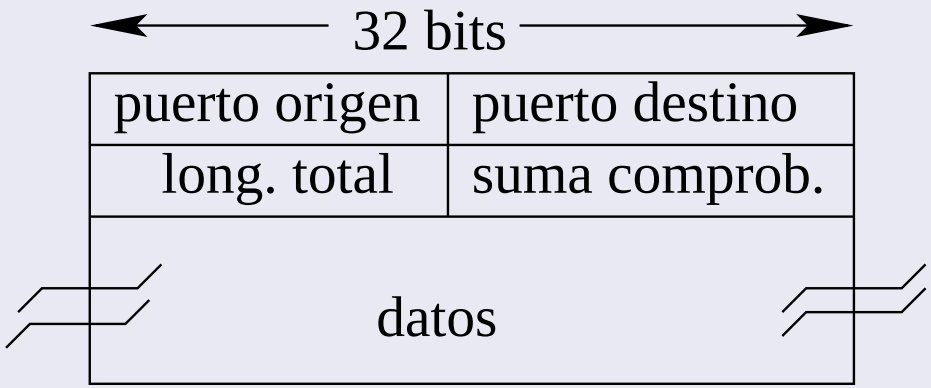
## Protocolo de datagramas de usuario

- Protocolo de transporte simple y poco sofisticado
- Hace casi lo mínimo que debería hacer un protocolo de transporte
  - Multiplexación/demultiplexación
  - Mecanismo de comprobación de errores
- Descrito en el RFC 768

# Transporte no orientado a conexión: UDP

## Protocolo de datagramas de usuario

- En origen, le añade una cabecera al mensaje para formar un segmento
- En destino, comprueba si el paquete llegó sin errores
  - Sin errores: entrega el paquete al socket
  - Con errores: en general se descarta
- Cabecera de 4 campos (8 bytes):
  - Puertos origen y destino
  - Longitud total del segmento (cabecera + datos) en bytes
  - Suma de comprobación incluyendo la pseudo-cabecera



# Transporte no orientado a conexión: UDP

## Características

- Sin conexión
  - No hay acuerdo previo entre emisor y receptor $\Longrightarrow$ no hay retardo en el establecimiento de la conexión
  - No hay retransmisiones en caso de errores
- Sin estado: cada segmento se envía con independencia de los demás
  - No hay segmentación (no hay información para reconstruir los mensajes)
  - Procesamiento más rápido y con menos recursos (más clientes)
- Cabeceras más pequeñas
- Más control de la aplicación
- Para aplicaciones que prefieren velocidad frente a fiabilidad: transmisión de audio, vídeo, DNS...

# Transporte no orientado a conexión: UDP

## Aplicaciones que usan TCP

- Correo electrónico (SMTP)
- Web (HTTP)
- Acceso a terminales remotos (telnet, SSH)
- Transferencia de archivos (FTP, SFTP)

## Aplicaciones que normalmente usan UDP

- Traducción de nombres (DNS)
- Protocolos de encaminamiento (RIP)
- Administración de red (SNMP)
- Servidor de archivos remoto (NFS)

## Aplicaciones que usan TCP o UDP

- Flujos multimedia
- Telefonía por Internet

## Cada vez se usa más TCP en aplicaciones multimedia, ¿?

- Aplicaciones multimedia: toleran pérdidas y responden mal a mecanismos de control de congestión
- Hay organizaciones que bloquean el tráfico UDP, porque no tienen mecanismos de control de congestión
  - Desbordamiento de paquetes en los routers
  - Estrangulamiento del tráfico TCP, que sí tiene mecanismos de control de congestión

# Índice

# Transmisión fiable

## Fundamentos

- Se basa en los protocolos de retransmisión: que retransmiten los paquetes con errores
- Protocolos ARQ *Automatic Repeat reQuest* (solicitud automática de repetición)
  - Parar y esperar: el emisor envía un paquete y espera la confirmación del receptor
  - Ventana deslizante: se pueden enviar varios paquetes antes de recibir las confirmaciones
    - Retroceder *N*
    - Repetición selectiva
- Se consideran enlaces bidireccionales (*full duplex*)
- Numeración de los paquetes 0, 1, ..., $2^n - 1$, $n$ número de bits para numerarlos

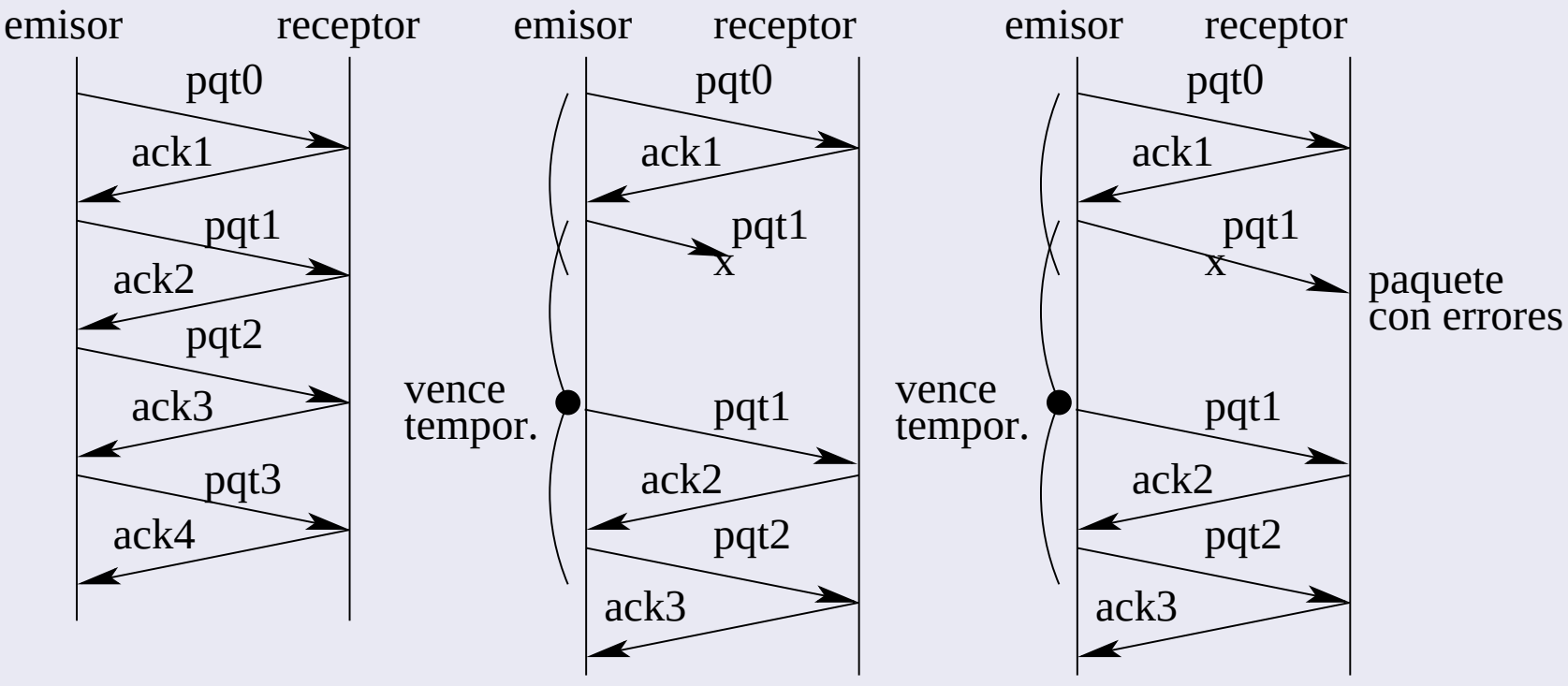
# Protocolo ARQ parar y esperar

## Casos

- Sin errores: el receptor devuelve un ACK con el número del siguiente paquete
- Pérdida de paquetes: el receptor no devuelve el ACK (timeout en el emisor) y el emisor retransmite el paquete
- Paquete con errores: similar al anterior
- Pérdida de un ACK: el emisor retransmite el paquete (timeout). El receptor recibe un duplicado, lo descarta y devuelve el ACK
- Timeout: paquetes y ACKs llegan con retraso $\Longrightarrow$ paquetes y ACKs duplicados
  - Paquetes duplicados: como antes
  - ACKs duplicados: se ignoran

# Protocolo ARQ parar y esperar

## Casos



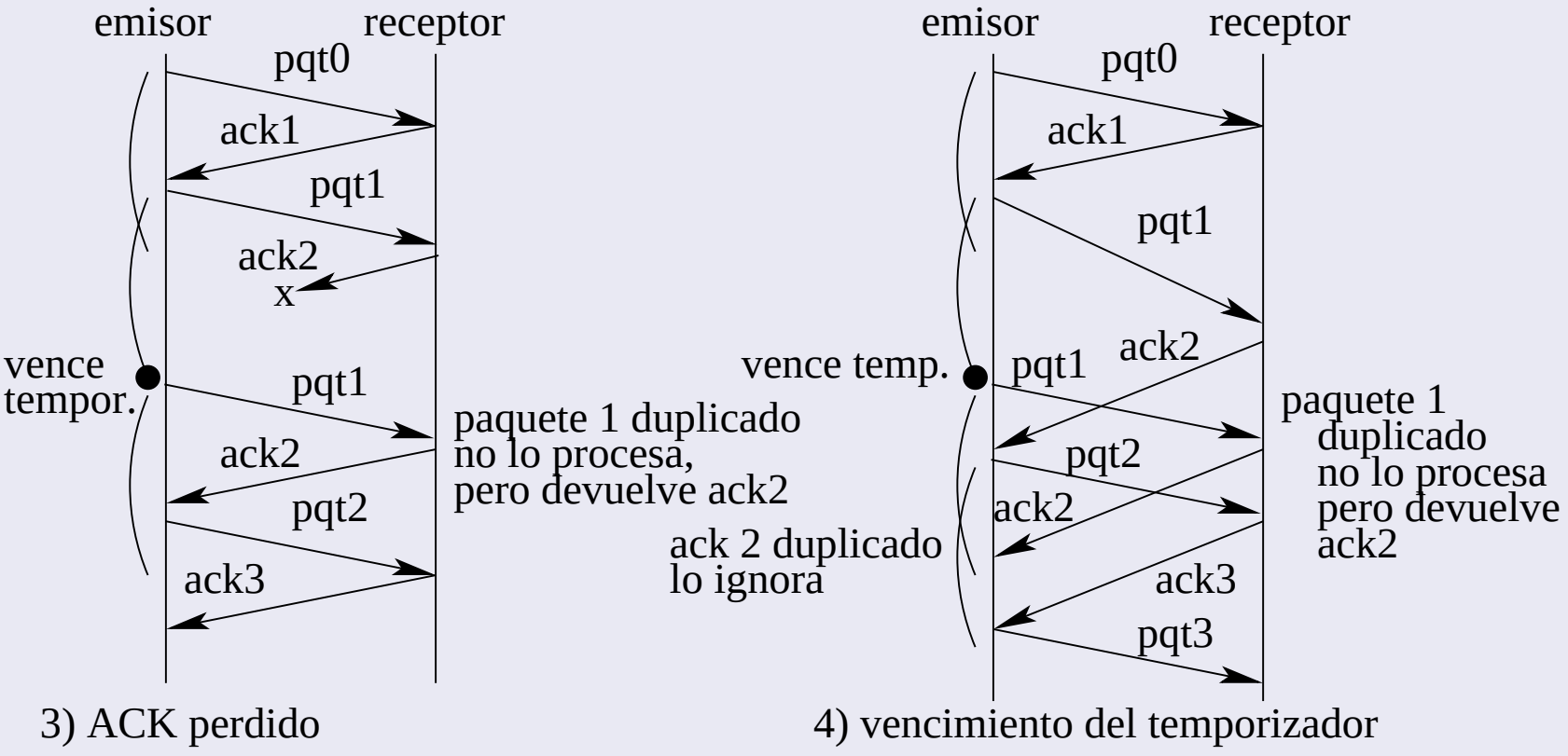

1) caso sin errores

2) caso con paquete perdido

2b) paquete con errores

# Protocolo ARQ parar y esperar

## Casos



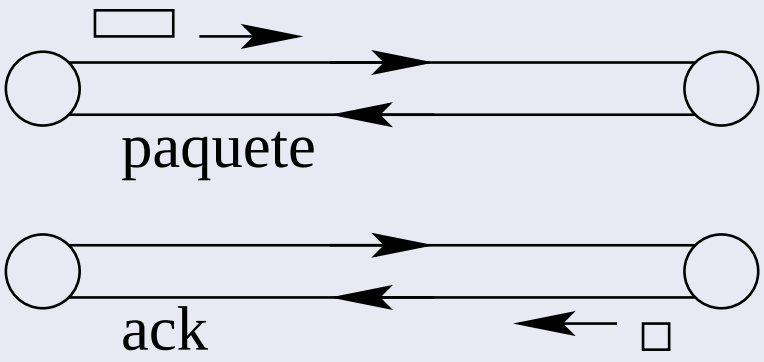
# Protocolo ARQ parar y esperar

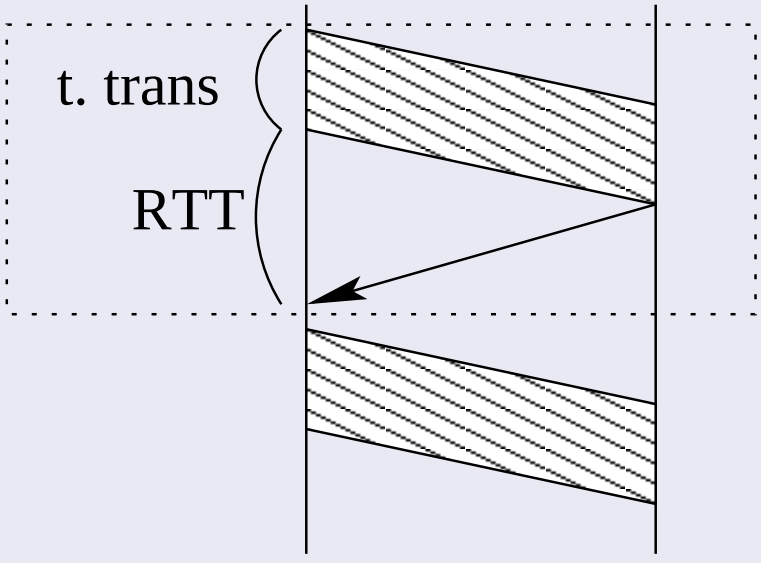
## Variantes

- NAK: indica la recepción de un paquete con errores $\Longrightarrow$ no se espera al timeout
- Dos ACKs iguales equivalen a un NAK: la recepción de un paquete con errores se devuelve el ACK del último paquete correcto
  - Si vence el temporizador, se envían continuamente duplicados de paquetes
- Tres ACKs iguales equivalen a un NAK: resuelve inconvenientes de la anterior
  - TCP lo usa con alguna variación

# Protocolo ARQ parar y esperar

## Inconveniente principal: poca utilización del enlace




Utilización del enlace por parte del emisor: $U = \frac{t_{trans}}{RTT + t_{trans}}$

Ejemplo: paquetes de 4.000 bits, por un enlace a 1 Gbps de 1.000 km con una velocidad de propagación de 200.000 km/s

# Protocolo ARQ de ventana deslizante

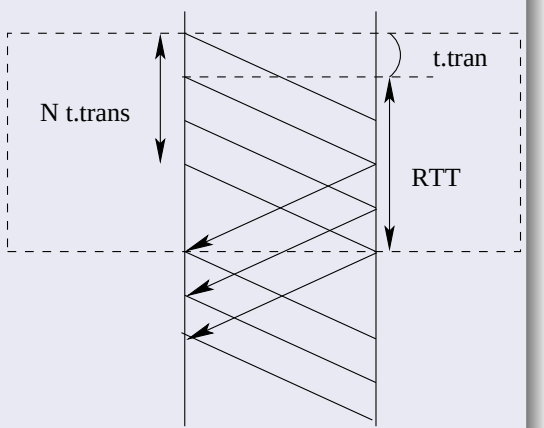
## Entubamiento

- El emisor envía un determinado número $N$ de paquetes antes de recibir las confirmaciones

## Utilización del enlace

- Tiempo útil en el emisor: $N \cdot t_{trans}$
- Tiempo total: $t_{trans} + RTT$
- $U = 1$ cuando $N \cdot t_{trans} \geq t_{trans} + RTT$

$$\Longrightarrow N \geq 1 + \frac{RTT}{t_{trans}}$$



# Protocolo ARQ de ventana deslizante

## Requerimientos

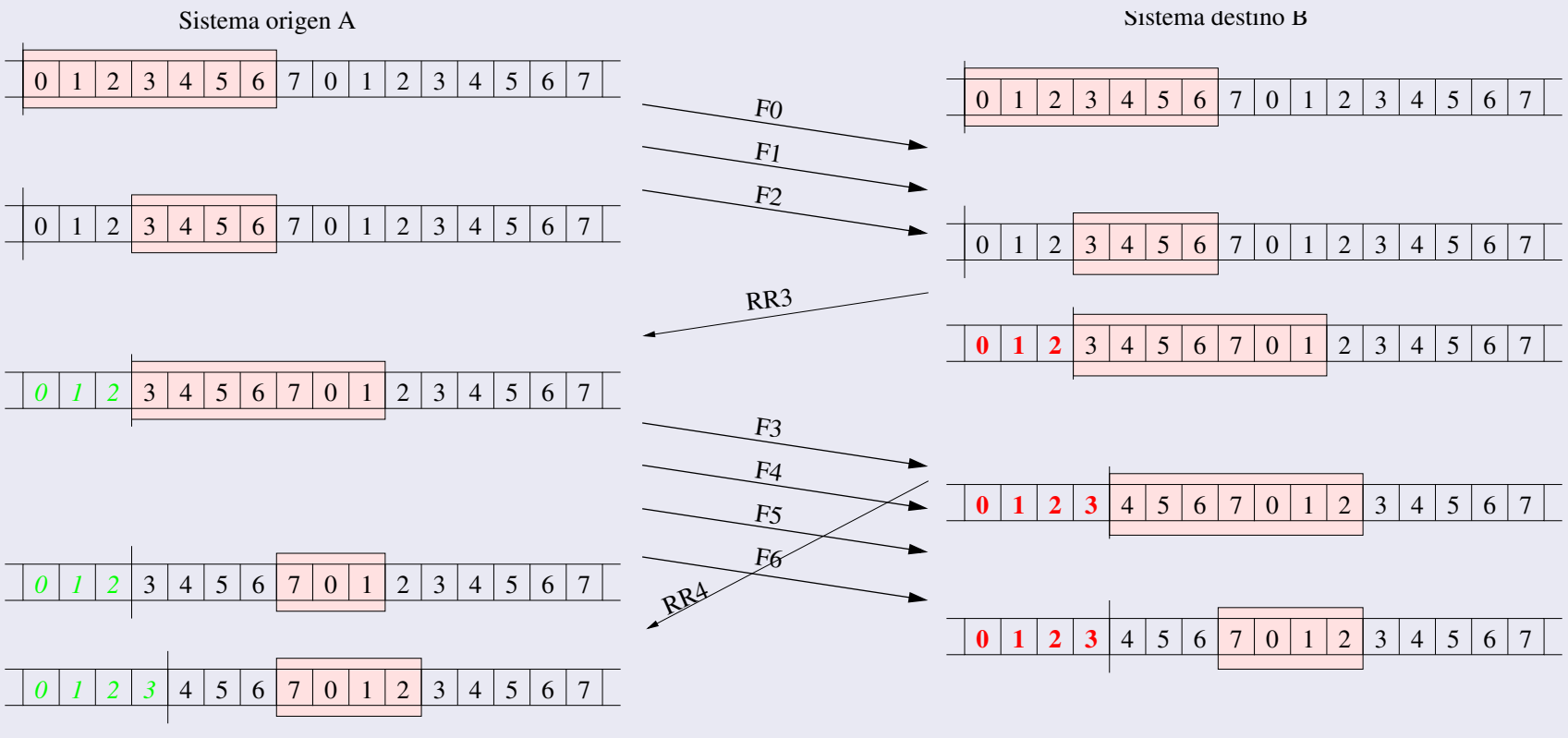
- El rango de números de secuencia debe abarcar al menos el doble del tamaño de la ventana emisora
- Emisor y receptor deben almacenar más de un paquete

## Ventanas

- Ventana emisora: es el conjunto de $N$ paquetes que el emisor puede enviar o están pendientes de confirmación
- Ventana receptora: es el conjunto de $N$ paquetes que el receptor puede aceptar o está procesando
- Estas ventanas se van desplazando a medida que el emisor (receptor) reciba (devuelva) las confirmaciones

# Protocolo ARQ de ventana deslizante

## Ejemplo



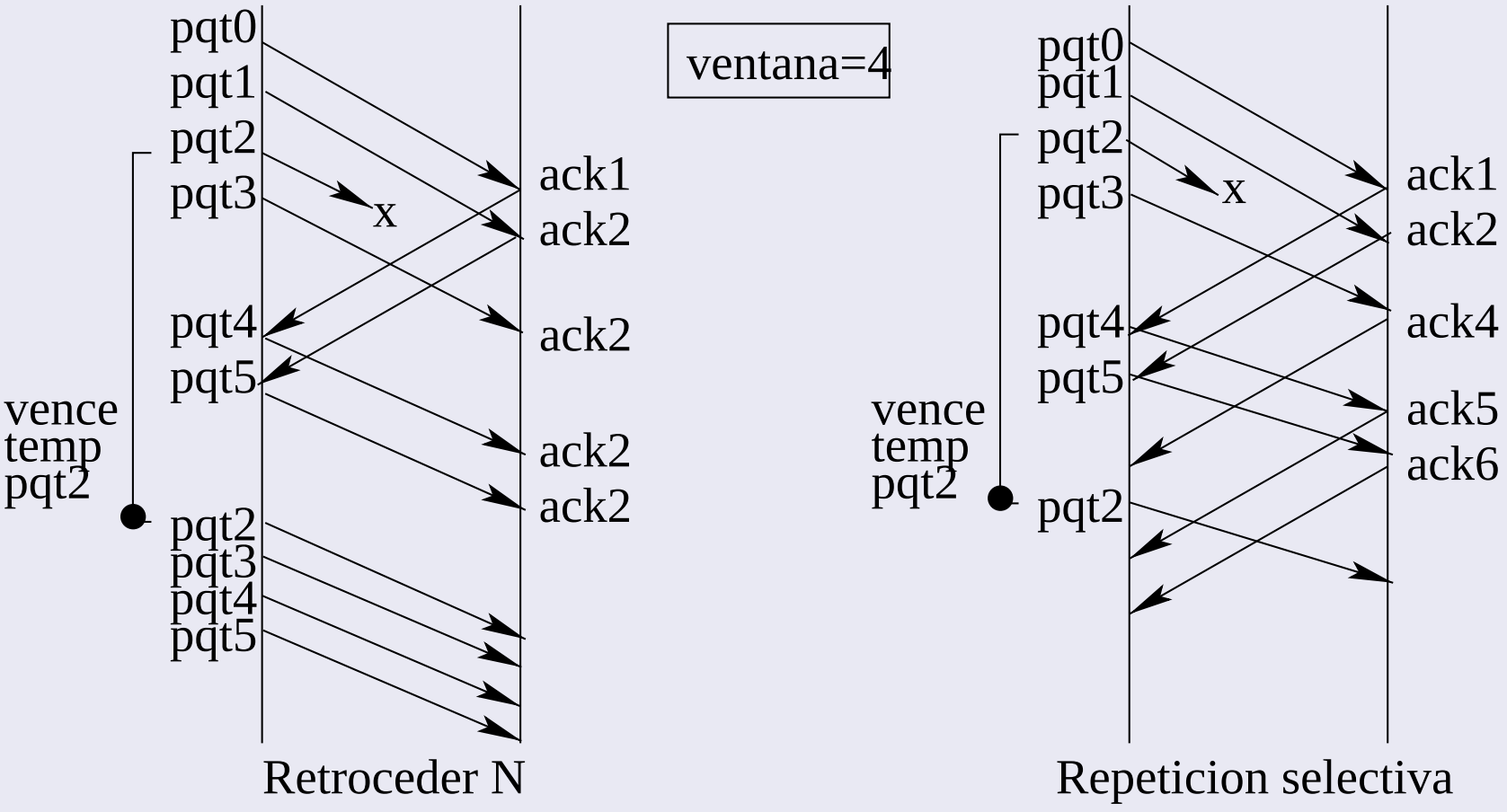
# Protocolo ARQ de ventana deslizante

## Tipos

- Retroceder *N* (*Go Back N*): el receptor sólo acepta paquetes en orden
  - Si un paquete llega con errores o no llega, se descartan los siguientes
  - Si expira el temporizador de un paquete, se retransmiten también los siguientes
  - ACKs acumulativos: un ACK implica un ACK a todos los paquetes previos
- Repetición selectiva: el receptor acepta paquetes fuera de orden
  - Solo se retransmiten los paquetes erróneos o que no llegan
  - Hay que enviar los ACKs para cada uno de lo paquetes recibidos

# Protocolo ARQ de ventana deslizante

## Retroceder *N* y repetición selectiva



# Índice

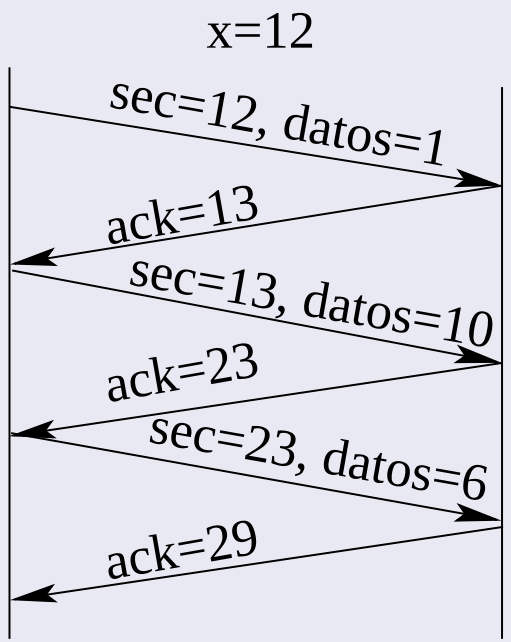
# TCP: protocolo de control de transmisión

## Transmisión fiable

- Aplicación de los principios de transmisión fiable

## Números de secuencia

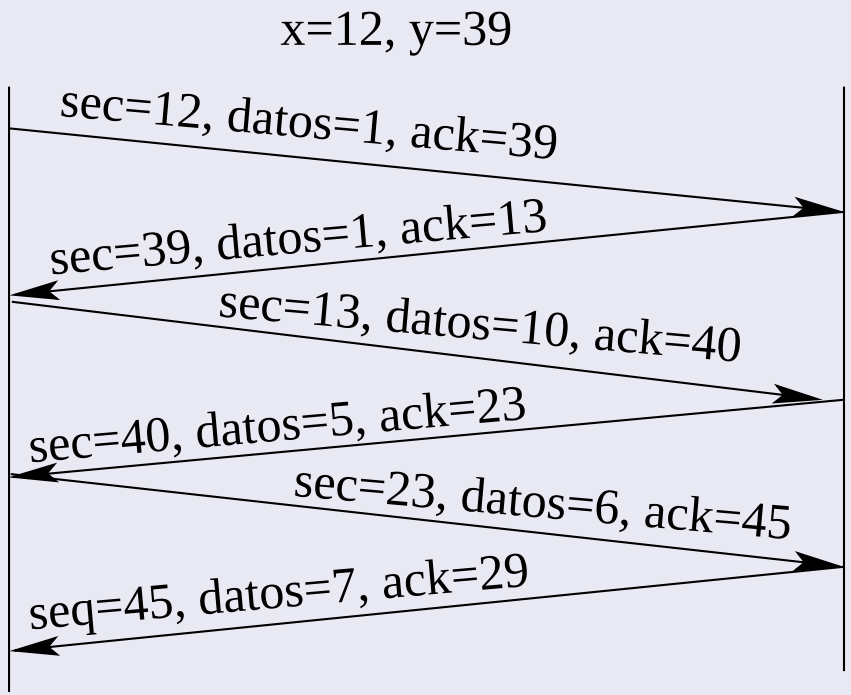
- Números de 32 bits que identifican bytes (no segmentos)
  - Se empieza por $x$ (aleatorio)
  - Se incrementa en $x + nbytes$
  - Los ACKs indican el siguiente byte que se desea recibir



# TCP: protocolo de control de transmisión

## Superposición (*piggybacking*)

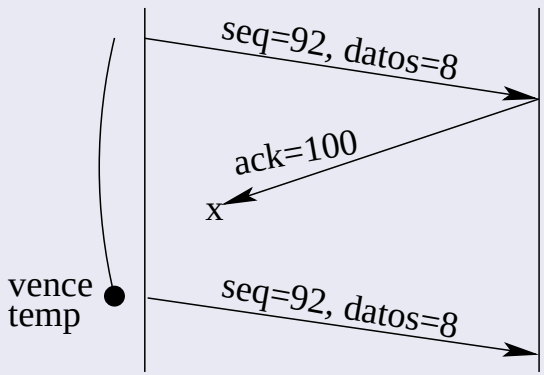
- En un mismo segmento ACK, se transmiten también datos
- Tiempo de espera máximo: si no tiene datos, transmite sólo el ACK



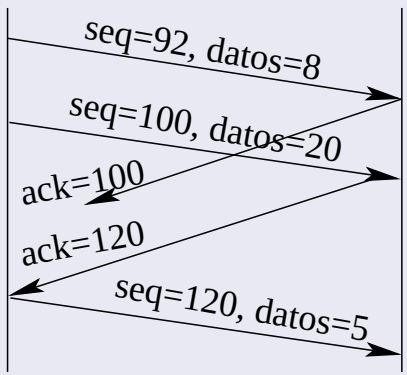
# TCP: protocolo de control de transmisión

## Transferencia fiable de datos

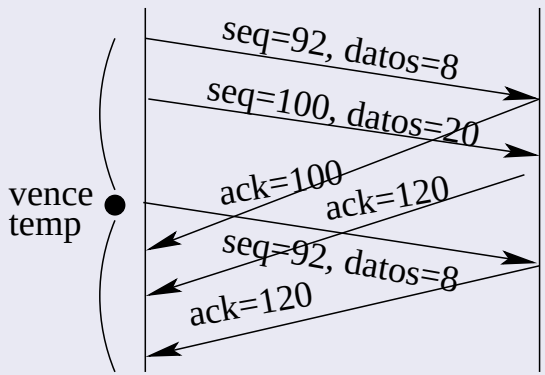
- El emisor usa temporizadores para la retransmisión
- Se usa ventana deslizante y ACKs acumulativos
- Se recomienda usar un temporizador único: cuando llega un ACK se reinicia
- Si un ACK no llega a tiempo, se retransmite solo ese segmento no confirmado, se reinicia el temporizador y se espera un ACK




perdida de ACK




ACK acumulativos




solo se retransmite uno

# TCP: protocolo de control de transmisión

## Estimación del tiempo de espera

$$\begin{aligned} \text{Temporizador} &= \text{EstimacionRTT} + 4\text{DevRTT} \\ \text{EstimacionRTT} &= (1-\alpha)\text{EstimacionRTT} + \alpha\text{MuestraRTT} \\ \text{DevRTT} &= (1-\beta)\text{DevRTT} + \beta|\text{MuestraRTT} - \text{EstimacionRTT}| \end{aligned}$$

- DevRTT es una medida de la variación del RTT y, en general, $\alpha = 0,125$ y $\beta = 0,25$
- La estimación del RTT es en base a segmentos transmitidos y confirmados (no retransmitidos)

## Duplicación del tiempo de espera

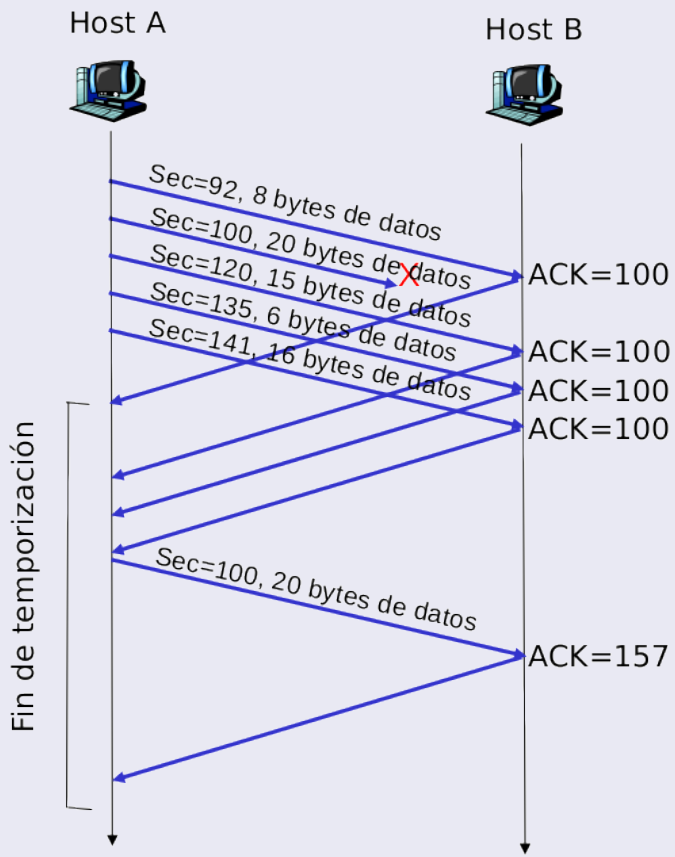
- Cuando se produce la expiración del temporizador, el emisor duplica el tiempo

# TCP: protocolo de control de transmisión

## Retransmisión rápida

Retransmisión de un paquete antes de que expire el temporizador

- 3 ACKs duplicados se interpretan como un NAK (el 4º)
- Se recibe un segmento con mayor número del esperado $\Longrightarrow$ el receptor envía un ACK duplicado
- Se repite la situación $\Longrightarrow$ el receptor envía otro ACK repetido $\Longrightarrow$ el emisor recibe 3 ACKs repetidos y realiza una *retransmisión rápida*



# TCP: protocolo de control de transmisión

## Pérdidas en TCP

- Expiración del temporizador: problema grave, se pierden los segmentos y los ACKs
- 3 ACKs duplicados: problema leve, al menos llegan los ACKs $\Longrightarrow$ *retransmisión rápida*

## ARQ intermedio entre retroceder *N* y repetición selectiva

- Retroceder *N*: ACKs acumulativos
- Repetición selectiva: aceptan segmentos fuera de orden y se retransmiten solo los necesarios

# TCP: protocolo de control de transmisión

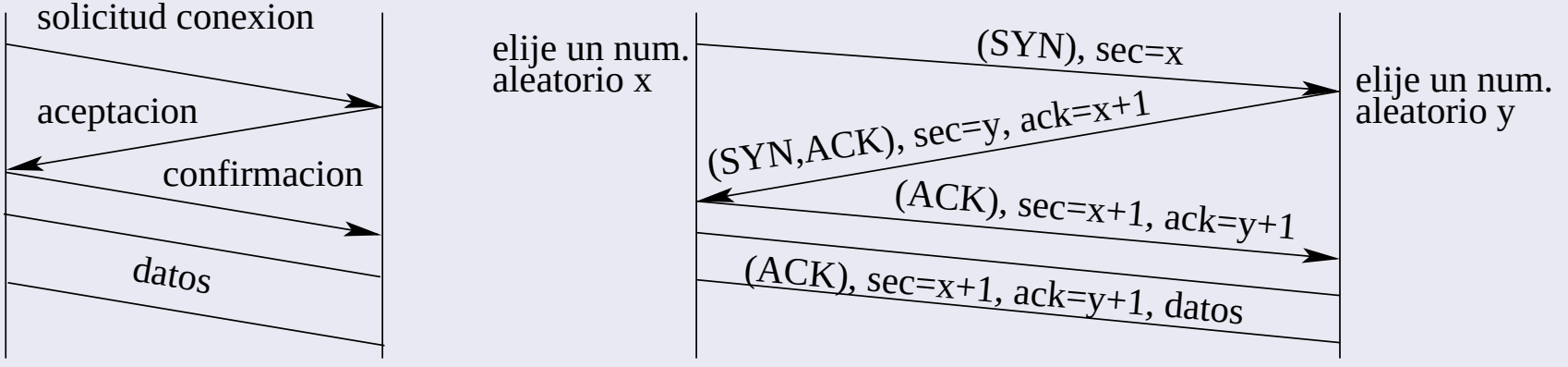
## Control de flujo

Mecanismo que permite al receptor indicar al emisor el ritmo al que puede recibir datos

- En el momento de la conexión, el receptor indica el tamaño de su ventana de recepción
- El emisor fija su ventana de envío a este valor
- Para ello existe un campo en la cabecera TCP
- El tamaño de ventana se puede modificar en cada transmisión

# TCP: protocolo de control de transmisión

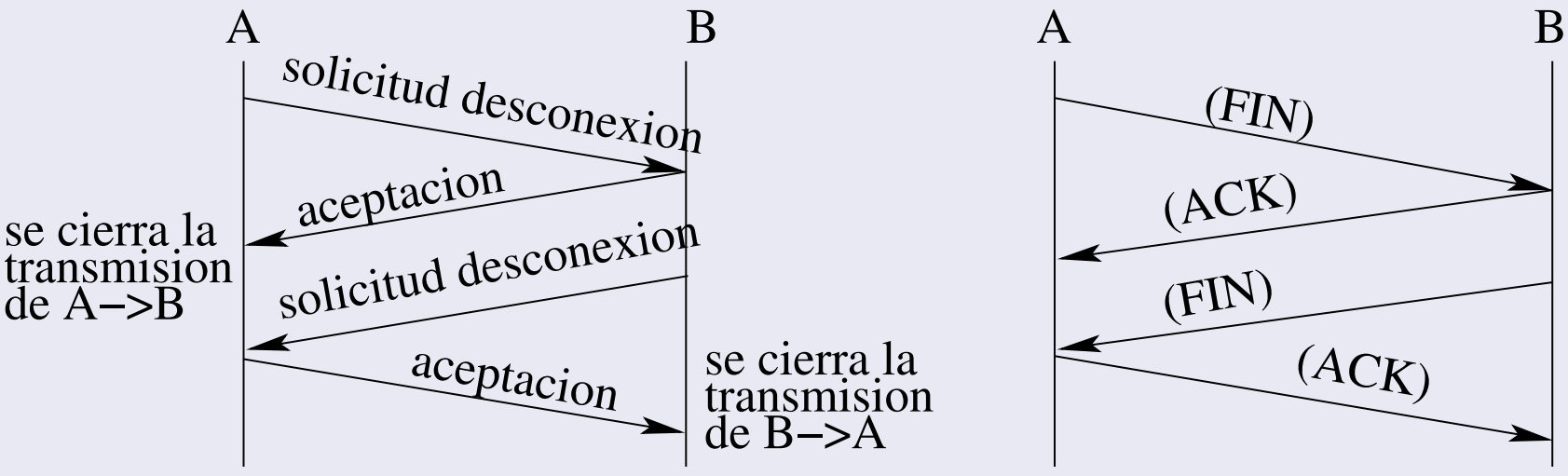
## Conexión: acuerdo en tres fases




- SYN = 1, cuando se envía por primera vez *x* o *y*
- ACK = 1, cuando se confirma un segmento
- En la tercera fase ya se pueden enviar datos

# TCP: protocolo de control de transmisión

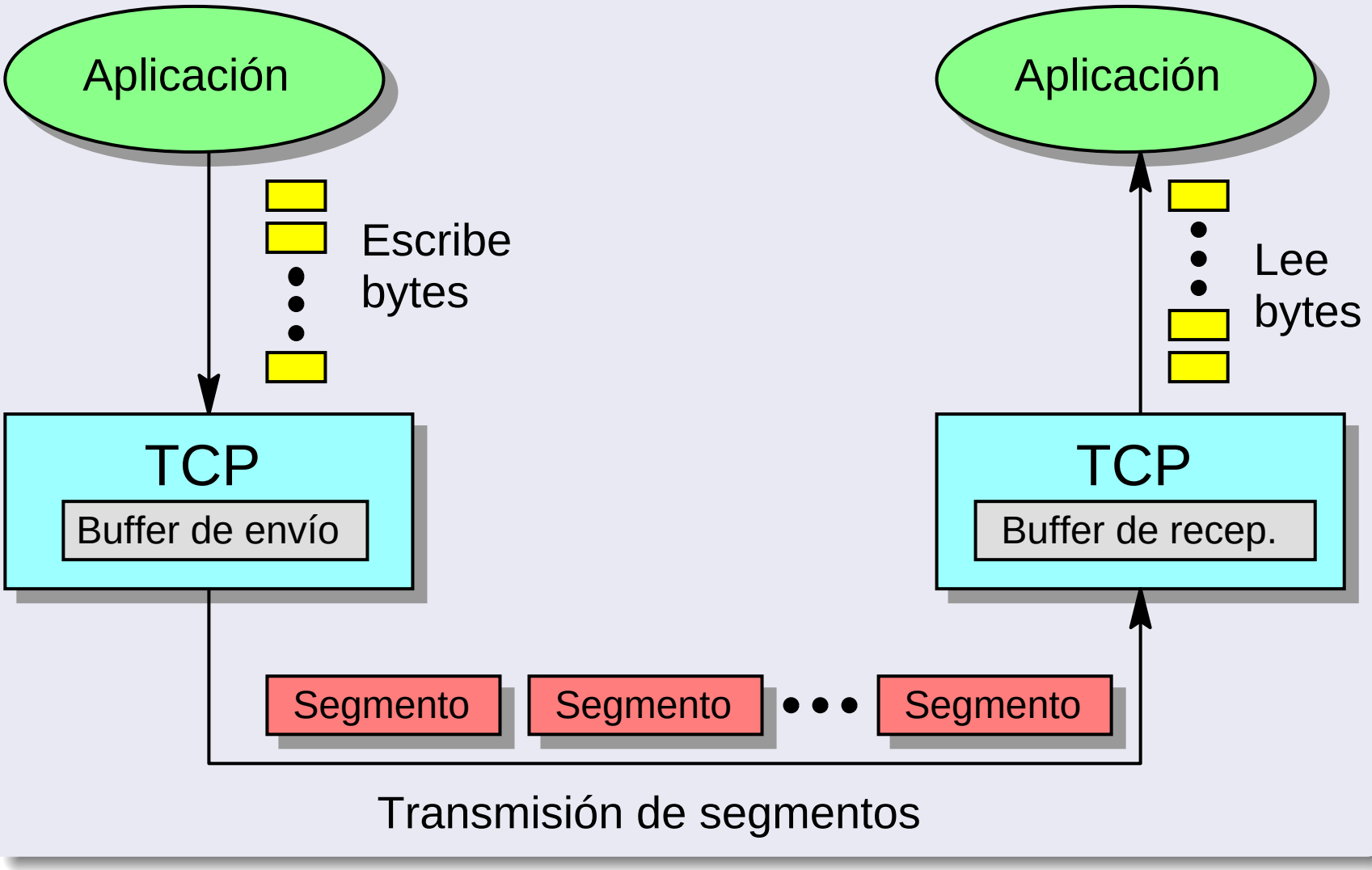
## Desconexión




- En dos fases: cada una para desconectar la transmisión en un sentido
- Se podría desconectar en un sentido y seguir transmitiendo en el otro
- FIN = 1, solicitud de desconexión
- ACK = 1, aceptación

# TCP: protocolo de control de transmisión

## Transmisión de datos



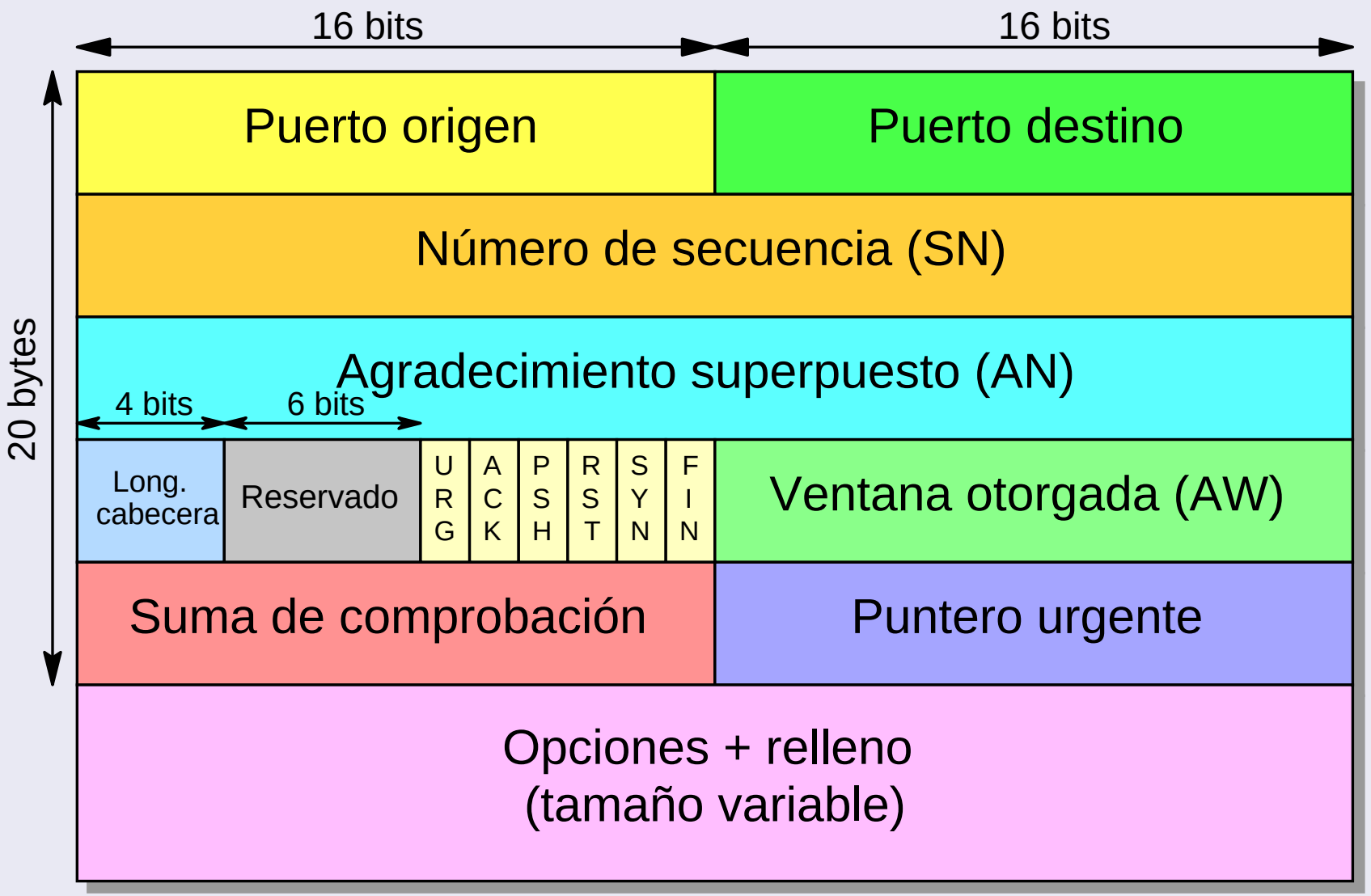
# TCP: protocolo de control de transmisión

## Transmisión de datos

- Una vez establecida la conexión empieza la transmisión
- La aplicación va pasando los datos a la capa de transporte (escribiendo en el socket) que los va acumulando
- Mecanismos para disparar la transmisión de un segmento:
  - Segmentación: cuando el número de bytes supere al MSS (*maximun segment size*)
  - Cuando la aplicación fuerza el envío (*push*)
  - Cuando un temporizador llega a 0
- TCP genera el segmento y se lo pasa a IP

# TCP: protocolo de control de transmisión

## Estructura de la cabecera TCP



# Índice

# Control de congestión

## Asignación de recursos

- Los recursos de la red se deben repartir entre las diferentes peticiones

## Congestión

- Demasiados paquetes en la red producen retardos en las transmisiones y pérdida de muchos paquetes
- Usualmente por el desbordamiento de la memoria de los routers

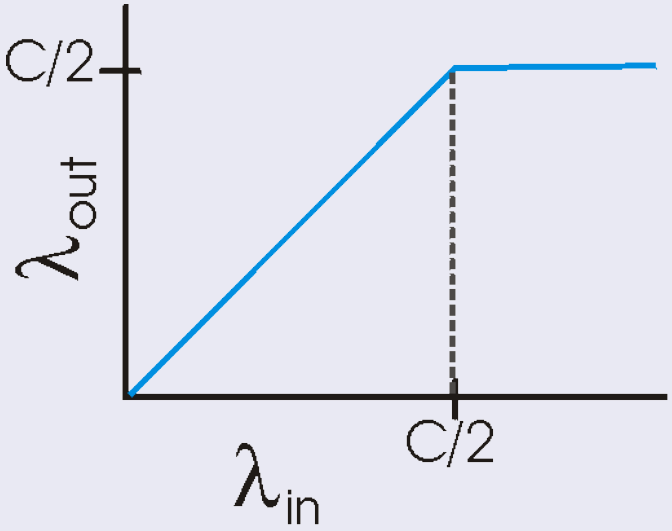
## Control de congestión

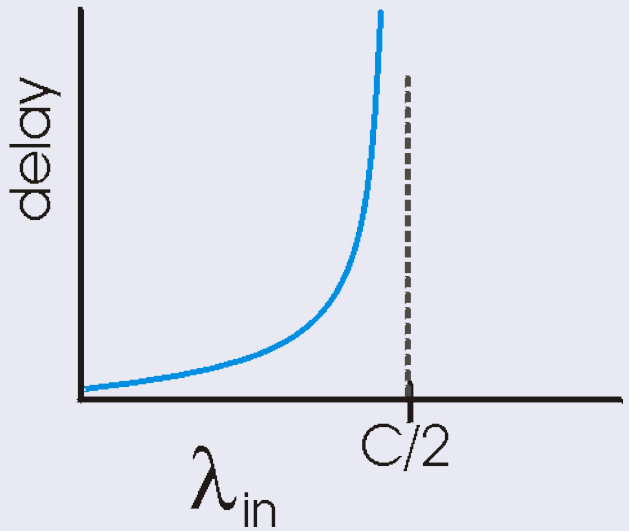
- Esfuerzos realizados por los elementos de la red para prevenir o responder a situaciones de congestión
  - Pre-reservar recursos para evitar congestión
  - Dejar que la congestión ocurra y resolverla entonces

# Control de congestión

## Origen de la congestión

- Escenario 1: dos emisores, un router con capacidad infinita y enlace compartido de velocidad C
  - Tasa de transmisión entre 0 y C/2: todo se recibe OK y con retardo finito
  - Tasa de transmisión mayor que C/2: el enlace no puede proporcionar paquetes a esa velocidad $\Rightarrow$ paquetes en la cola del router y aumenta el retardo

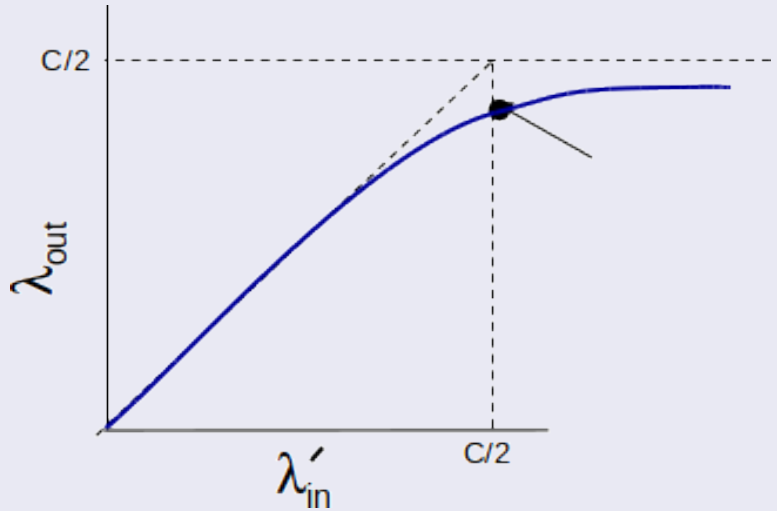


# Control de congestión

## Origen de la congestión

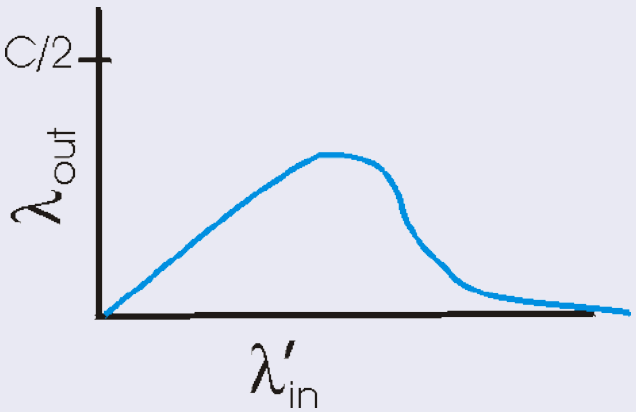
- Escenario 2: dos emisores, un router con memoria finita y enlace compartido de velocidad C
  - $\lambda'_{in}$: carga ofrecida, contiene datos originales y retransmitidos
  - Tasa de transmisión entre 0 y C/2: todo se recibe OK y con retardo finito
  - Tasa de transmisión mayor que C/2: la tasa entregada disminuye porque algunos son duplicados



# Control de congestión

## Origen de la congestión

- Escenario 3: varios emisores, routers con memoria finita y varios enlaces
  - Tasa de transmisión pequeña: todo se recibe OK y con retardo finito




- Tasa de transmisión elevada: los buffers de los routers se llenan y la tasa entregada disminuye
  - En el límite tiende a cero

## Necesidad de mecanismos de control de congestión

En TCP/IP el control de congestión recae principalmente en TCP

# Control de congestión en TCP

## Mecanismo en TCP

- El emisor adecúa su tasa de envío en función de la congestión que percibe
- Considera que hay congestión si:
  - Expira un temporizador
  - Se reciben 3 ACKs duplicados

## Procedimiento

- En el emisor se definen: la *ventana de congestión* y el RTT
- El RTT se estima periódicamente (tiempo desde el envío de un segmento hasta que llega su ACK)
- Se actualiza la ventana de congestión según el caso

$$\text{tasa de envío} = \frac{\text{ventana de congestión}}{RTT} \text{ bytes/segundo}$$

# Control de congestión en TCP

## Mecanismos para actualizar la ventana de congestión

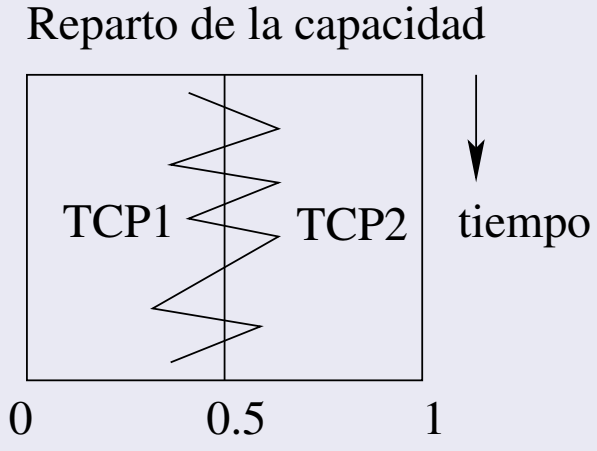
- Inicio lento: determina la capacidad inicial de la red
- Incremento aditivo/decremento multiplicativo (AIMD): situación normal
- Recuperación rápida: evita la fase de inicio lento cuando hay congestión

# Control de congestión en TCP

## Imparcialidad

Dos aplicaciones compartiendo un enlace de capacidad limitada

- Dos conexiones TCP se reparten la capacidad del enlace




- Una conexión TCP y una UDP, la UDP acaparará la mayor parte de la capacidad
- Si una aplicación usa una conexión TCP y la otra 9 conexiones TCP paralelas, la capacidad será 1/10, 9/10