Tema 4: El nivel de transporte

Bibliografía: [Kurose10], Capítulo 3 (excepto 3.4 y 3.6)

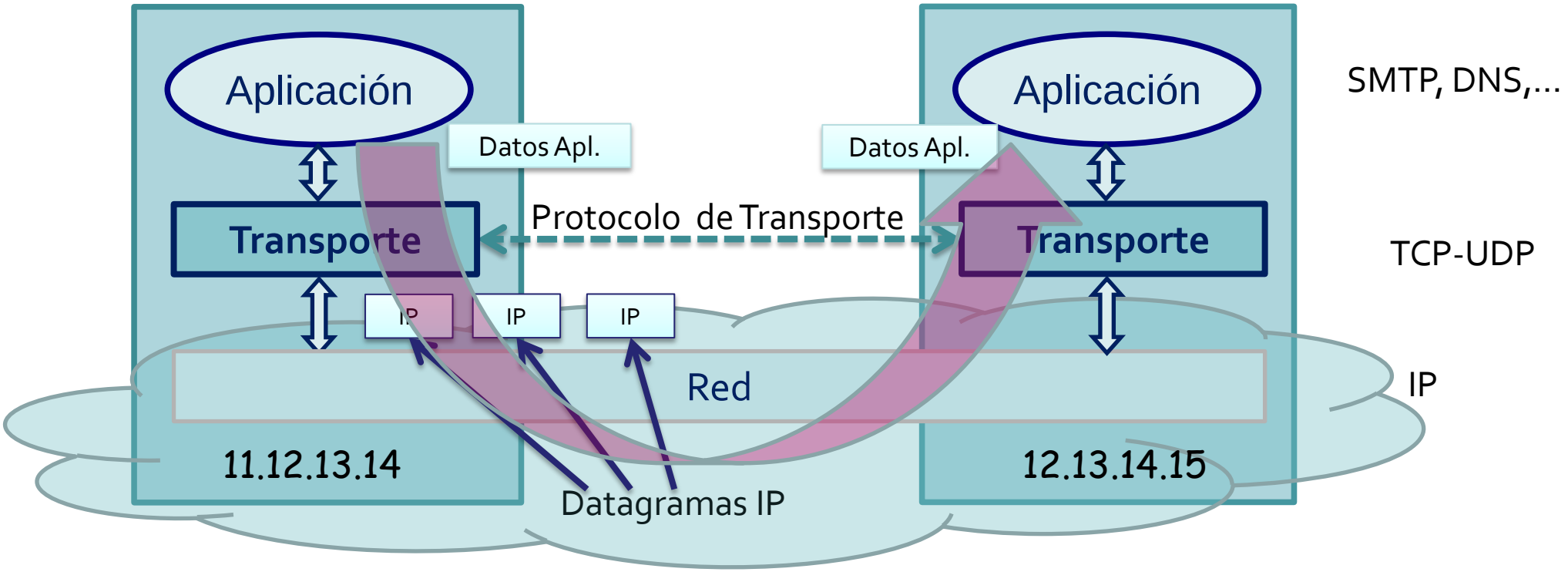
Objetivos

- Justificar la existencia del nivel de transporte
- Comprender el funcionamiento del servicio de transporte sin conexión (UDP)
- Aprender el funcionamiento básico del servicio de transporte orientado a la conexión (TCP)
  - En particular los mecanismos TCP para:
    - Control de flujo y de error
    - Control de la congestión

Índice

1. Servicios del nivel de transporte

- Proporciona **comunicación lógica** entre dos procesos de aplicación
- En una arquitectura de red puede haber varios protocolos de transporte disponibles
  - En Internet: TCP y UDP

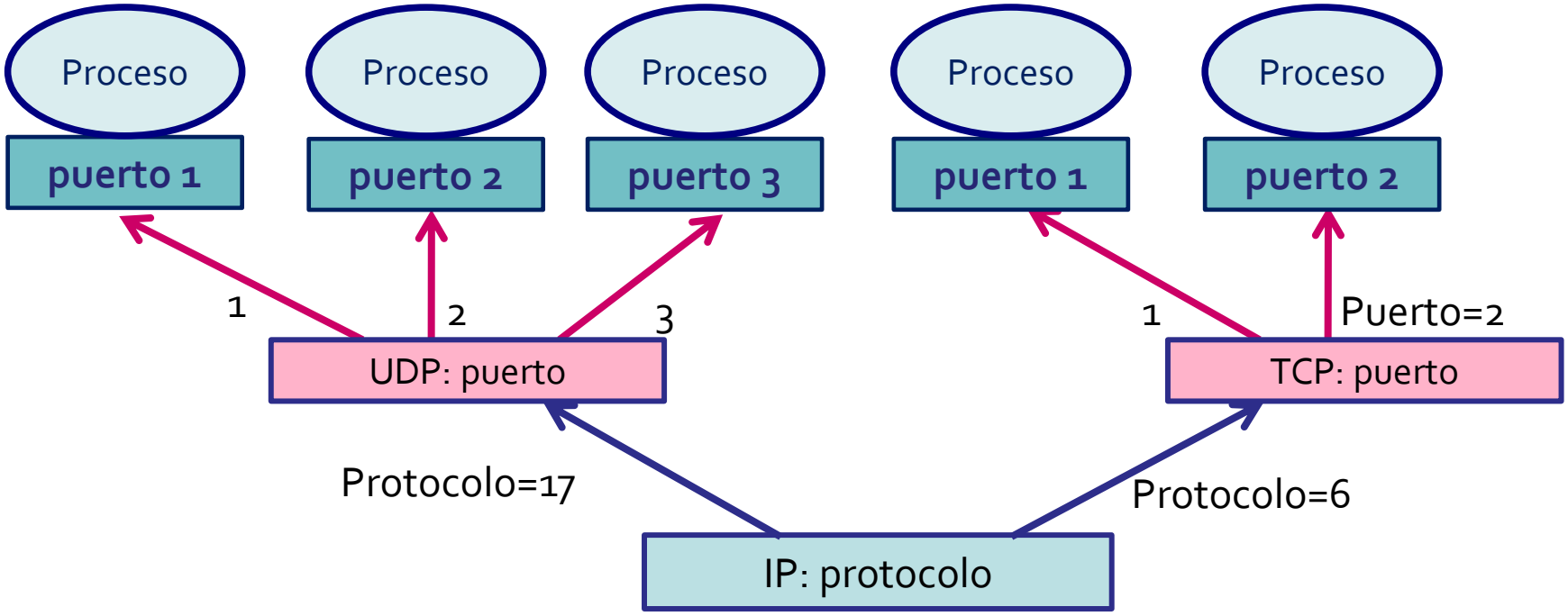


Direccionamiento de transporte

- Direccionamiento de red (IP):
  - Comunicación lógica entre hosts
- Direccionamiento de transporte:
  - Comunicación lógica entre procesos
  - Se apoya en los servicios del nivel de red, puede mejorarlos

Demultiplexación

- Receptor: la información de las cabeceras permite entregar los datos al proceso correcto
  - *Socket:* Punto de acceso a los servicios de transporte.
  - Los *atributos* que diferencian los distintos *sockets* en Internet son la dirección IP y el número de puerto



Números de puerto

- Valor de 16 bits (RFC 3232)
  - Cada servidor emplea un puerto diferente
  - El valor en el cliente no es importante

http://www.iana.org/assignments/port-numbers

Puertos en UDP

- Creación de *sockets* UDP

```
DatagramSocket cliente = new DatagramSocket();
DatagramSocket servidor = new DatagramSocket(6428);
```

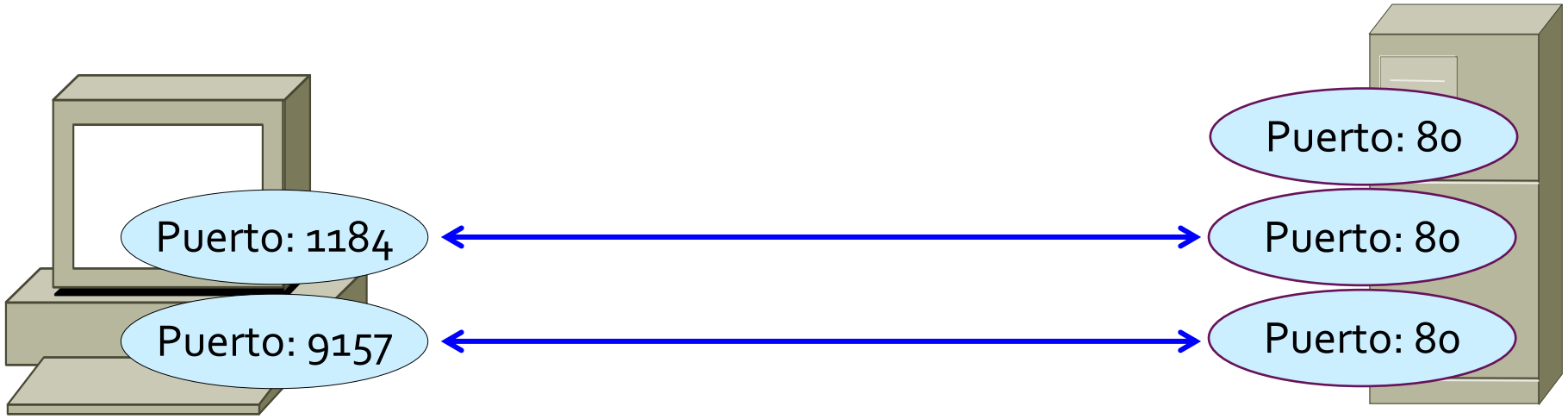
- Un *socket* UDP se identifica por dos valores:
  - (**Dirección IP, número de puerto**)

Puertos en TCP

- Creación de *sockets* TCP
  - `Socket cliente = new Socket("www.serv.com",80);`
  - `ServerSocket servidor = new ServerSocket(80);`
  - `Socket servidorconectado = servidor.accept();`
- Un *socket* TCP se identifica por al menos 2 valores:
  - **Dirección IP y número de puerto local**
- … y un *socket* TCP conectado por 4 valores
  - **Dirección IP y puerto local**
  - **Dirección IP y puerto remoto**

Demultiplexado con conexión

- Un mismo número de puerto TCP puede emplearse simultáneamente en distintas conexiones
  - Ej: (158.42.53.27, 1184, 158.42.53.5, 80) y (158.42.53.27, 9157, 158.42.53.5, 80)



Índice

UDP (*User Datagram Protocol*, RFC 768)

- Servicio de transferencia NO fiable
  - Sin conexión: sólo transferencia de datos
    - Cada segmento se trata de forma independiente
  - Sin garantía de entrega, ni orden
  - Sobrecarga mínima
- Permite difusiones
- Aplicaciones UDP:
  - Flujos multimedia
  - DNS, SNMP, RIP
- Transferencia fiable sobre UDP:
  - Debe implementarla la propia aplicación

Encapsulado de segmentos UDP

Hasta 64 kB

APLICACIÓN:

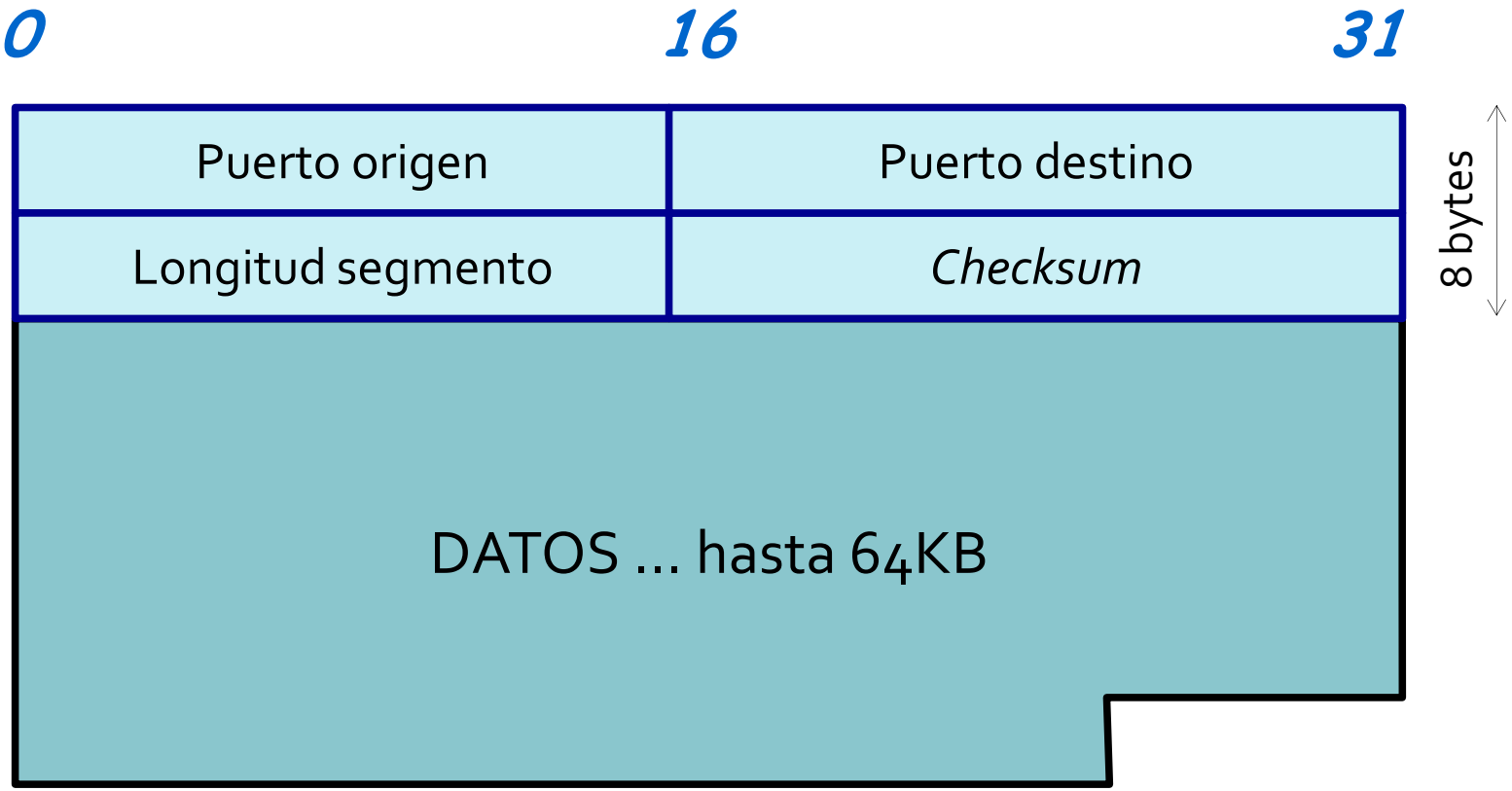
UDP:
Añade cabecera
Pasa a IP

SEGMENTO UDP

IP:
Añade cabecera
Envía a la red

DATAGRAMA **IP**

Formato de un segmento UDP



Objetivo: detectar errores en los segmentos transmitidos

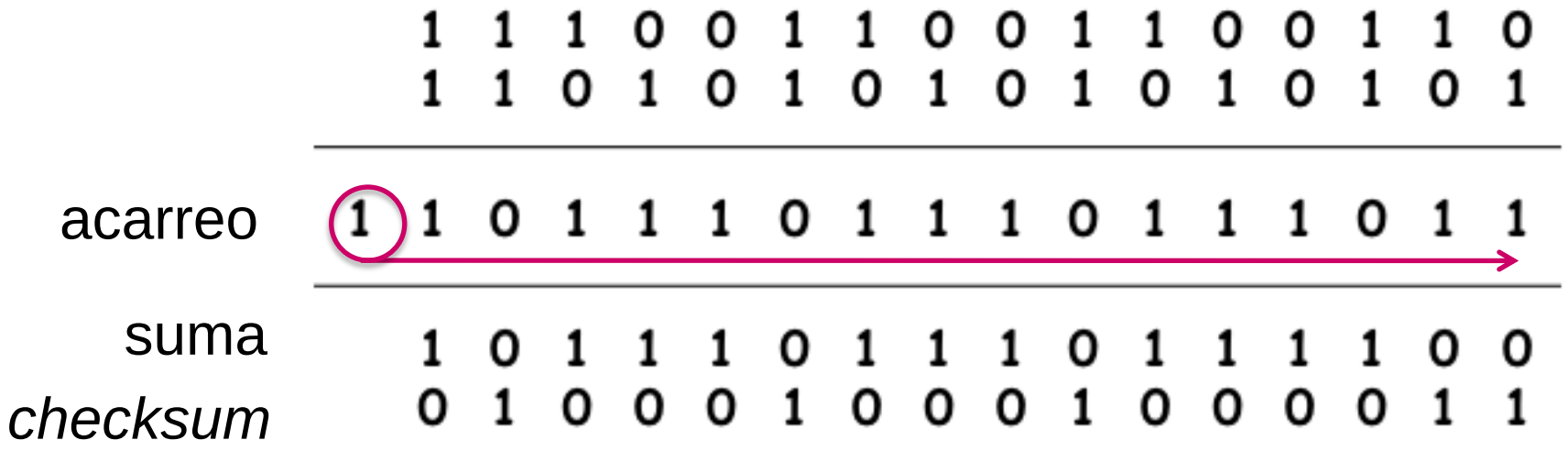
- **Emisor:**
  - Considera los datos como enteros de 16 bits
  - Realiza la suma de todas las palabras empleando aritmética en complemento a uno:
    - Los acarreos se propagan, sumándose al bit menos significativo
  - Calcula el complemento a uno del resultado y lo emplea como *checksum*

- **Receptor:**
  - Realiza nuevamente la suma
  - El resultado debe ser 0 representado por `1111111111111111`
  - En caso contrario, el mensaje se considera erróneo

http://www.netfor2.com/checksum.html

Ejemplo de cálculo del *checksum*

- Hay que propagar el acarreo
- Ejemplo: suma de 2 enteros de 16 bits

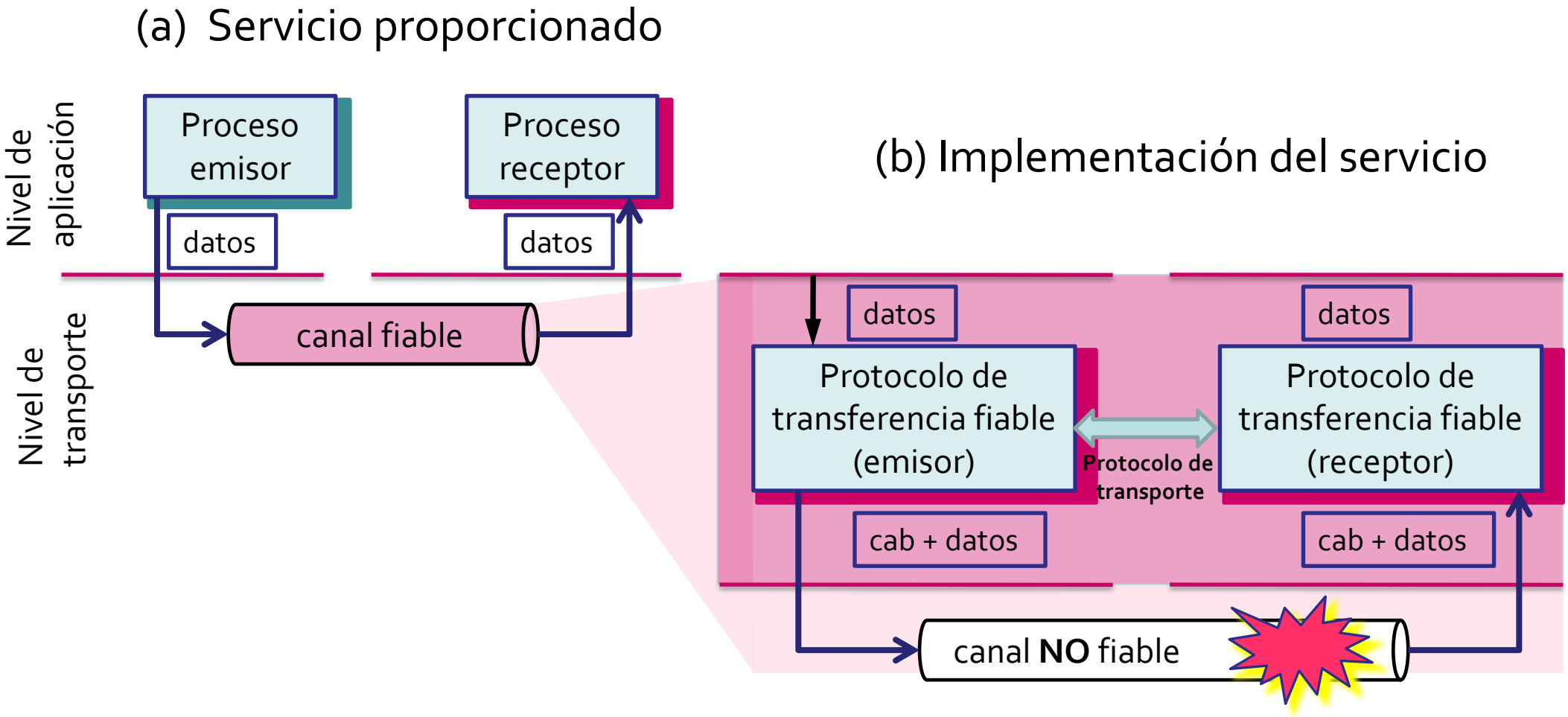


Índice

3. Transferencia fiable de datos

- La transferencia fiable es un problema que se puede plantear en distintos niveles de la arquitectura
- Problema:
  - ¿Cómo conseguir transferencia fiable sobre una red que puede perder segmentos?
- Solución:
  - Proporcionar mecanismos que garanticen la **detección** y **retransmisión** de los datos perdidos o dañados

Transferencia fiable de datos



- Canal fiable: entrega datos correctos y en orden
- Canal no fiable: errores de transmisión, congestión, encaminamiento, permutación, etc.

Protocolo ARQ (*Automatic Repeat reQuest*)

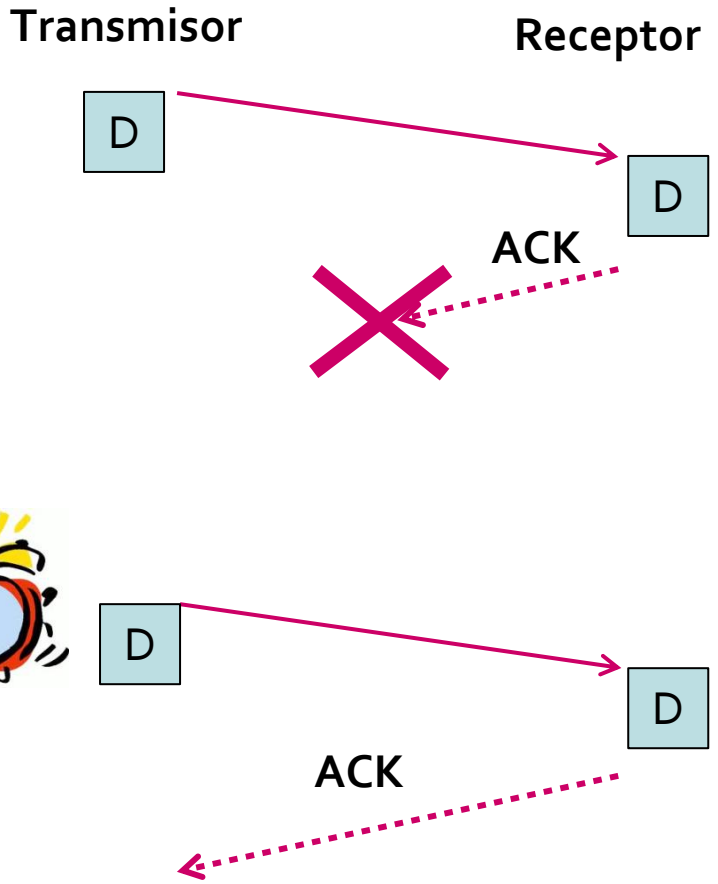
- Emisor y receptor deben resolver unas cuestiones básicas:
  - Emisor
    - ¿Se ha recibido correctamente el segmento?
  - Receptor
    - ¿El segmento recibido es correcto?
      - Evaluación de un código redundante (*checksum* o similar)
    - ¿Qué hacer en caso de que no lo sea?

ARQ (*Automatic Repeat reQuest*)

- Errores
  - Detección: código detector de error
  - Recuperación basada en dos mecanismos complementarios
    - Reconocimientos dentro de un plazo de tiempo (*Acknowledgements*, ACKs)
      - El problema puede presentarse también en el ACK
    - Retransmisión si no llega el reconocimiento en un intervalo de tiempo acotado (*timeout*)

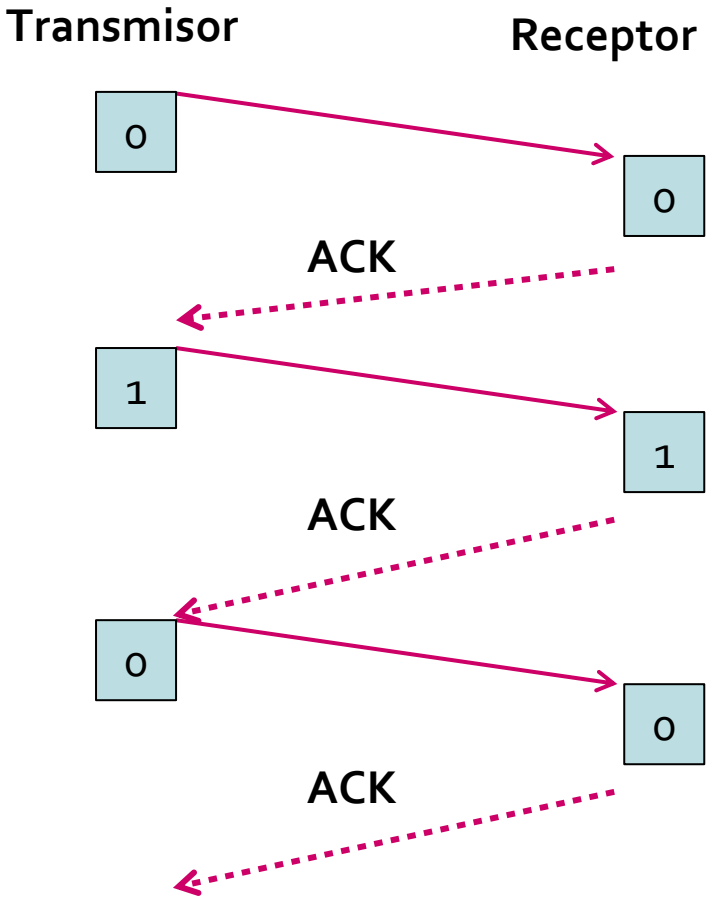
Detección de pérdida de segmento

- Emisor
  - Espera un tiempo limitado (temporizador)
  - Reenvía el segmento si no llega su reconocimiento
- ¿Cómo detecta el receptor que ha recibido un duplicado?

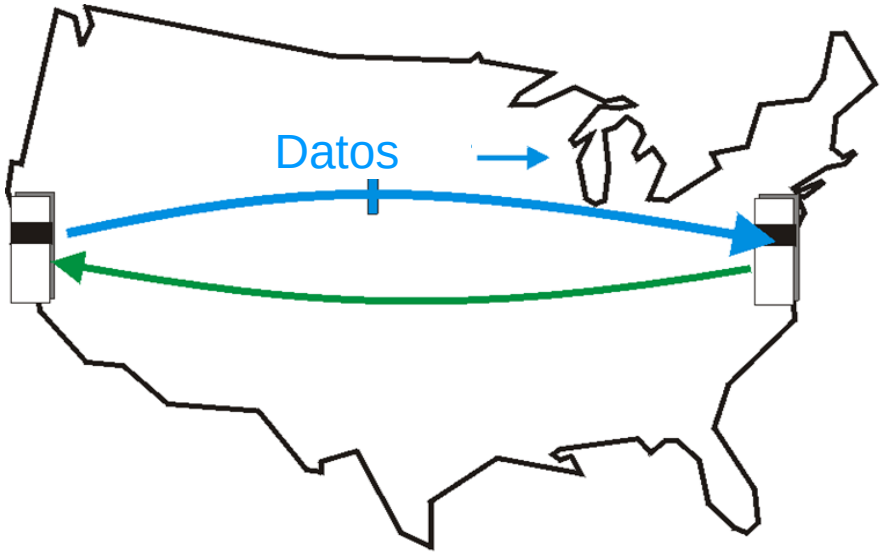


Detección de pérdida de segmento

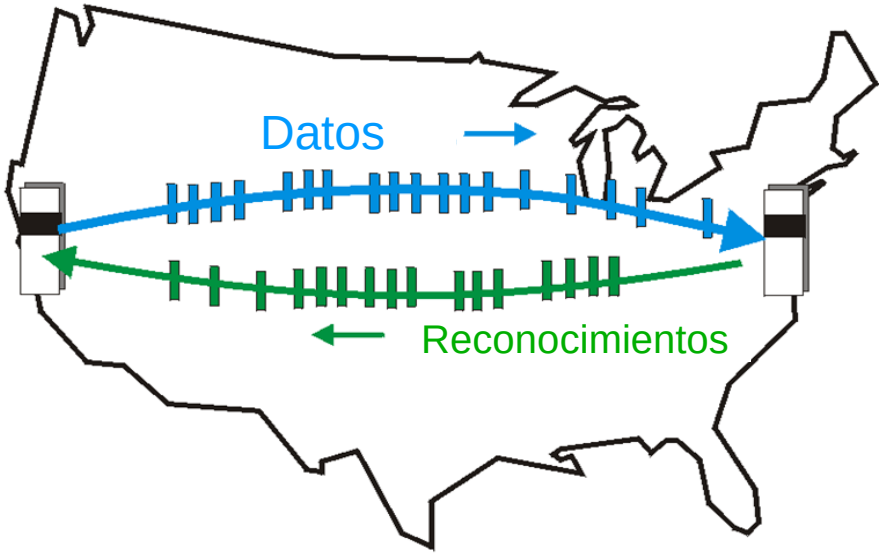
- El transmisor sólo puede tener un segmento pendiente de reconocimiento
  - **¡¡Simple pero ineficiente!!**
- Se pierde mucho ancho de banda



Comparación de prestaciones



Parada y espera



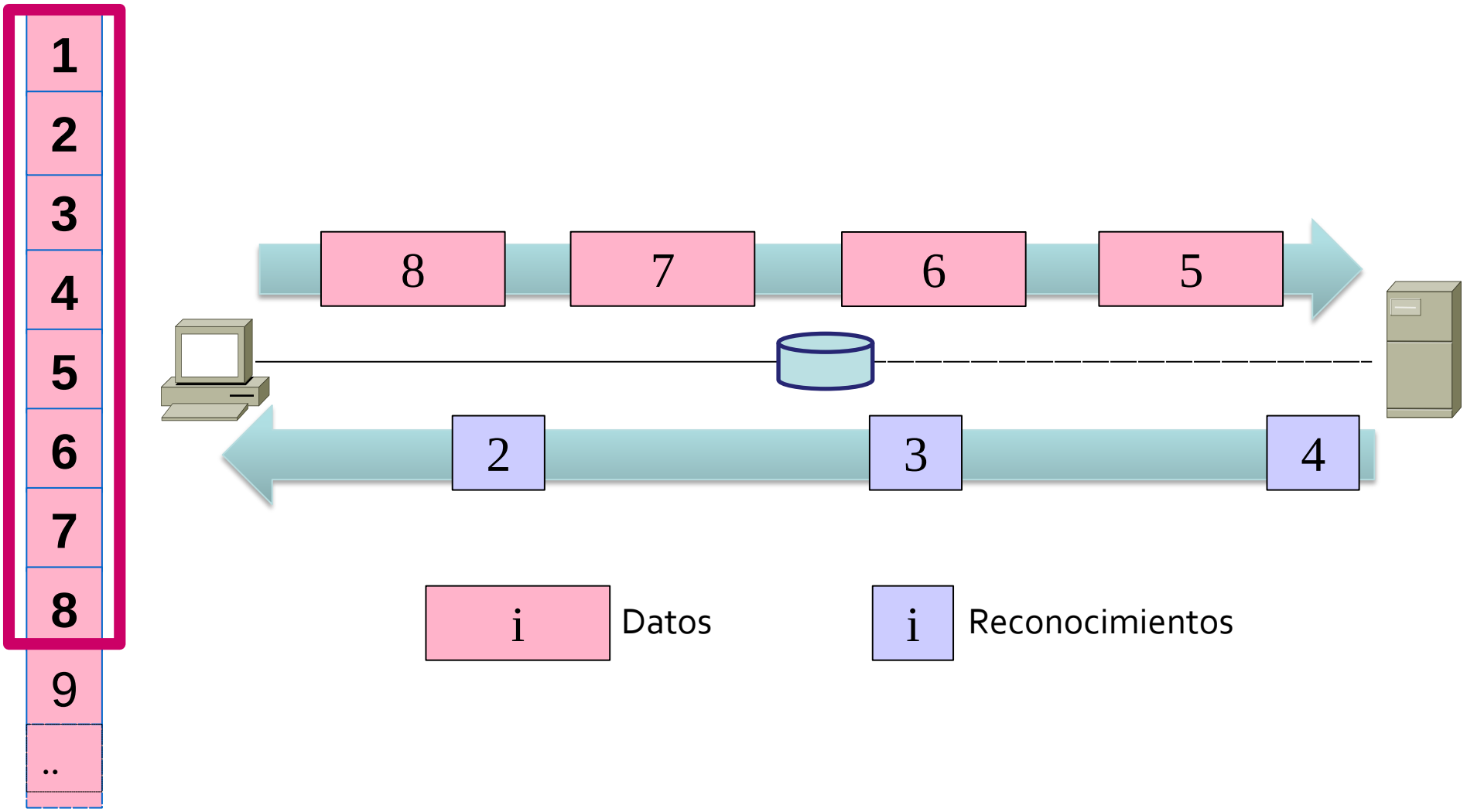
Ventana deslizante

Ventana deslizante

- Mejora de la eficiencia: **varios segmentos por RTT (*pipelining* o procesamiento en cadena)**
- Emisor
  - Almacena los segmentos pendientes de reconocimiento en la ventana de transmisión:
    - Tamaño variable limitado a N
  - La ventana incrementa su límite inferior al recibir ACKs y su límite superior con las nuevas transmisiones

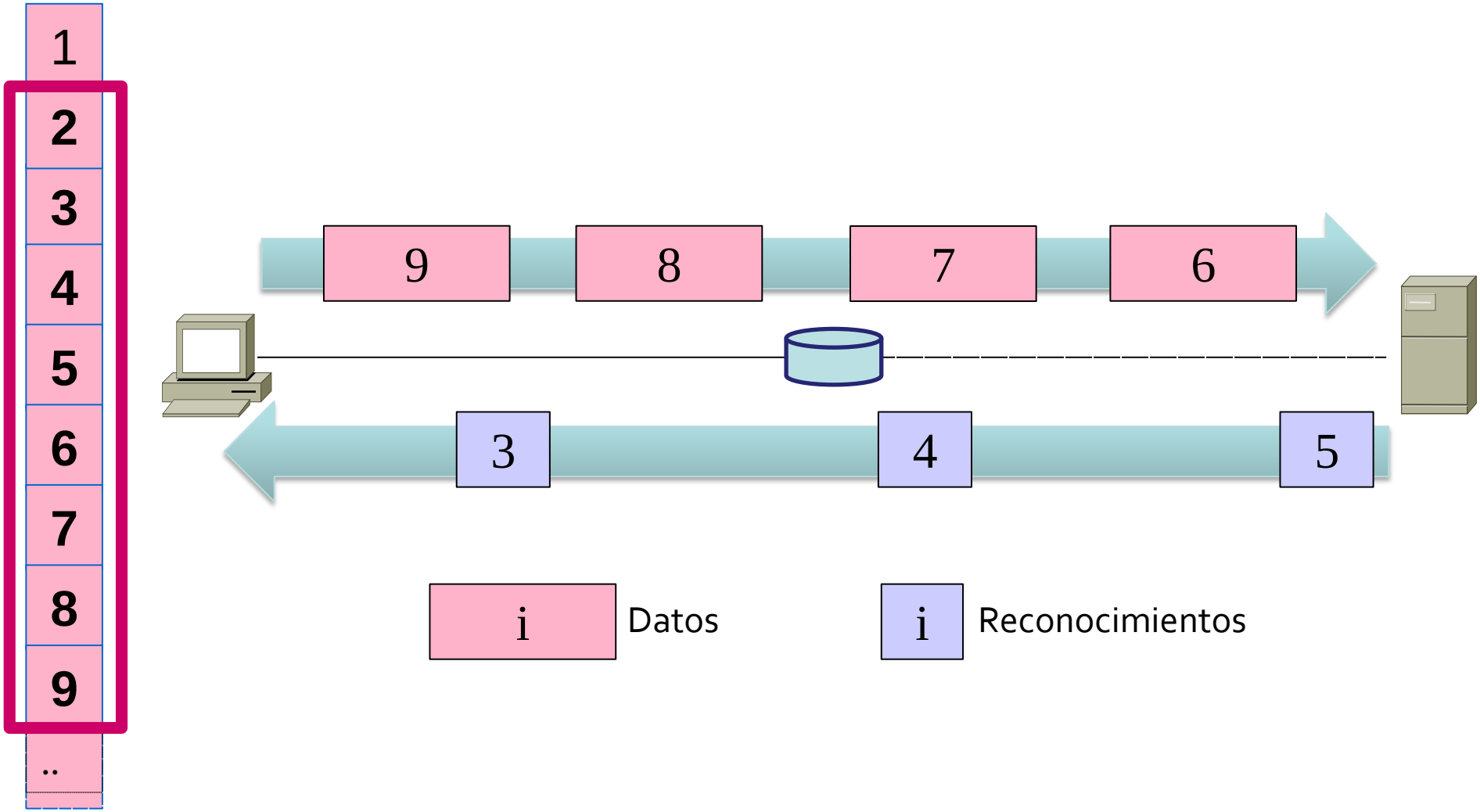
Funcionamiento ventana deslizante

Ventana máxima = 8



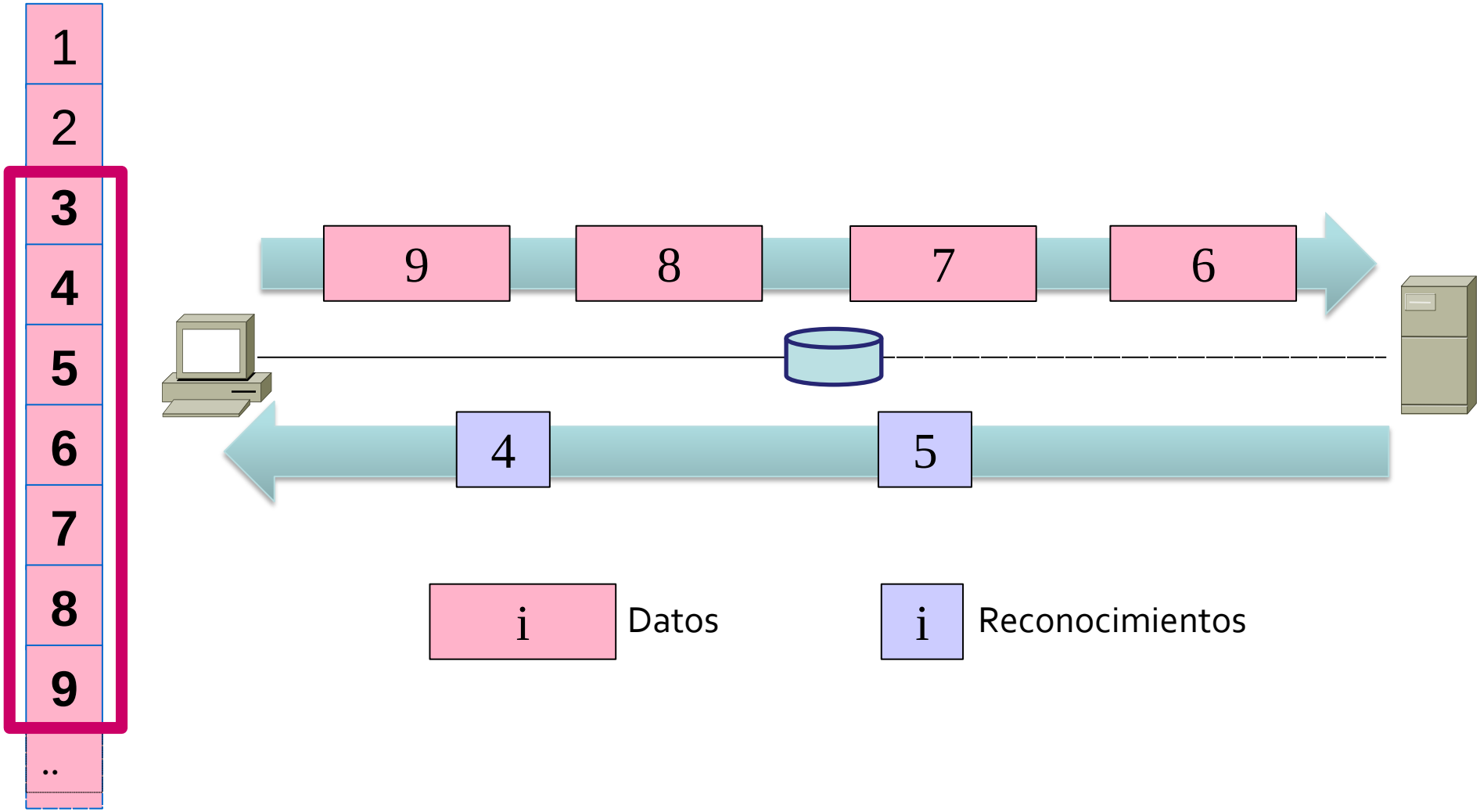
Funcionamiento ventana deslizante

Ventana máxima = 8

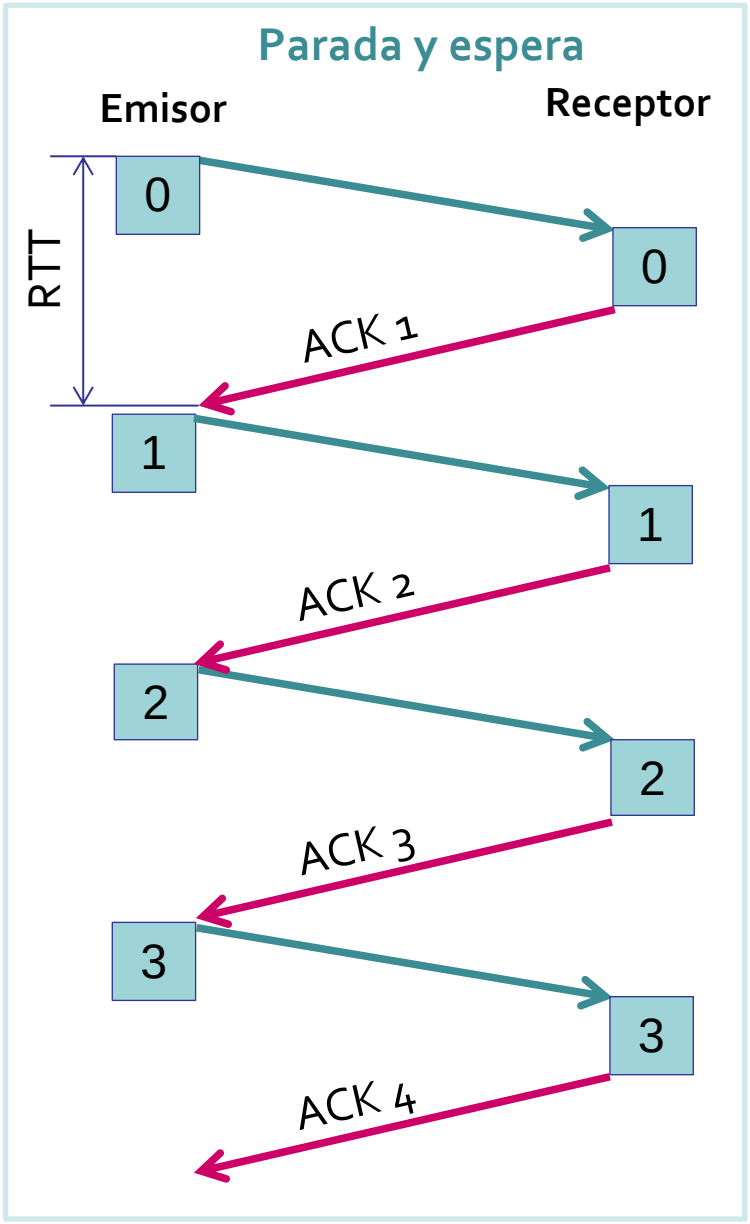


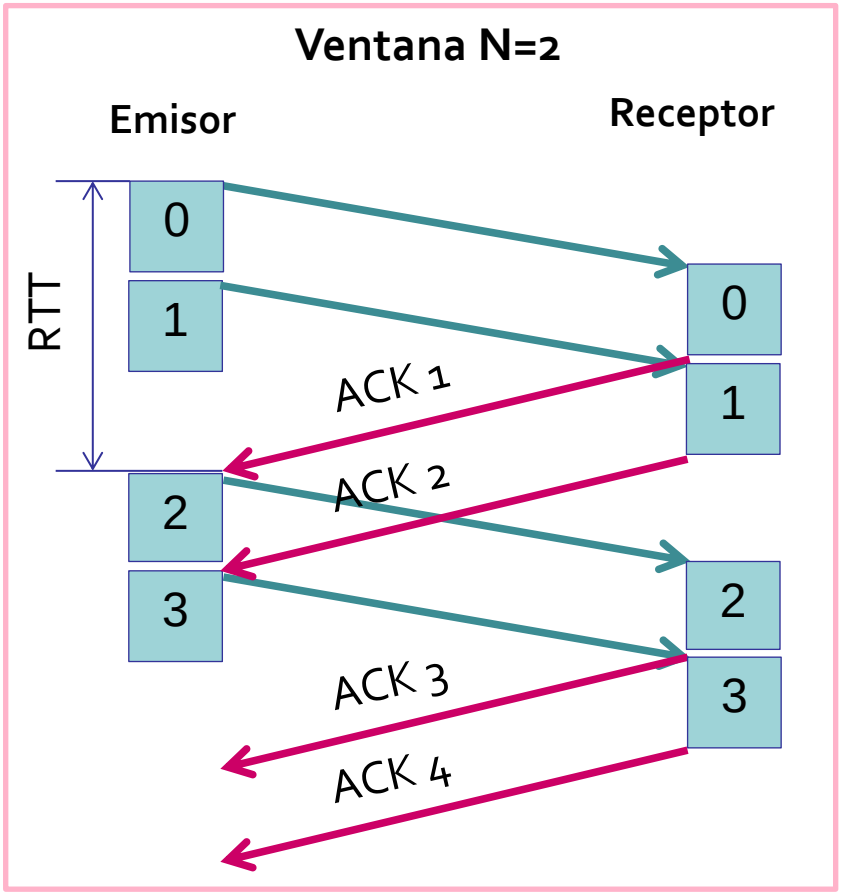
Funcionamiento ventana deslizante

Ventana máxima = 8



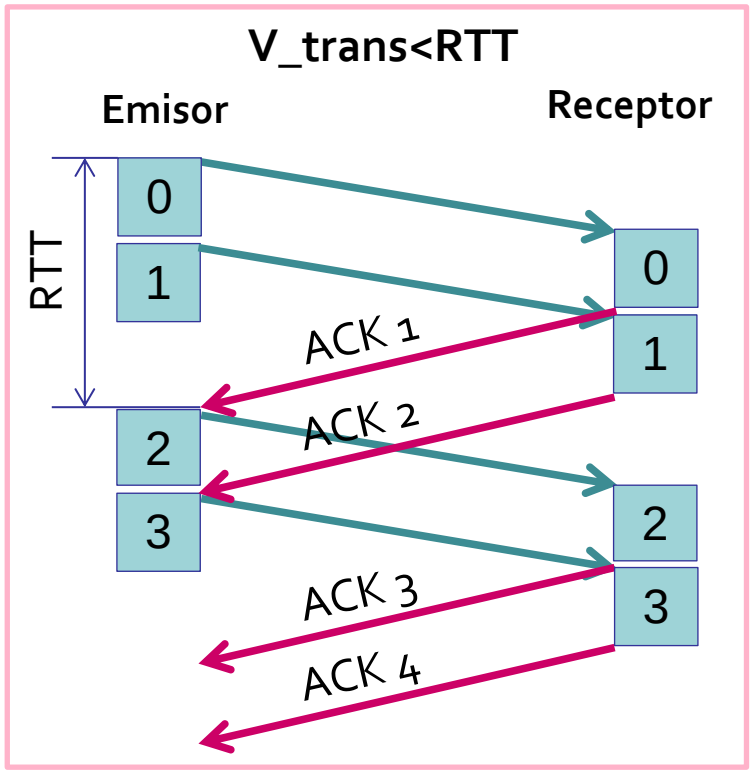
Comparación de prestaciones

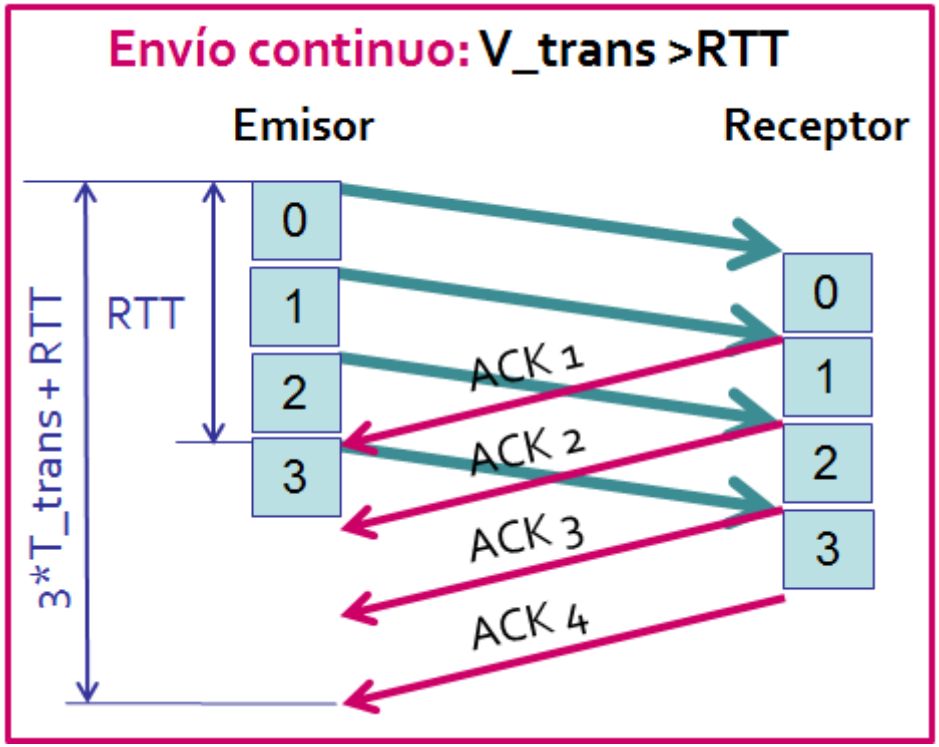




Dimensionamiento de ventana

Ventana deslizante





ventana (bits) ≥ RTT * Vtrans

Recepción en ventana deslizante

- Los segmentos pueden perderse o llegar desordenados
- ¿Y si llega un segmento fuera de orden?
  - Se rechaza: vuelta atrás
  - Se acepta: retransmisión selectiva
- Introducimos la **ventana de recepción**:
  - Tiene tamaño fijo
  - Incluye los números de secuencia de los segmentos que el receptor puede recibir

Resumen: Tamaño de ventanas

Ventana de Transmisión

- **Tamaño variable** (límite máximo a N)
- Segmentos con temporizador de retransmisión asociado

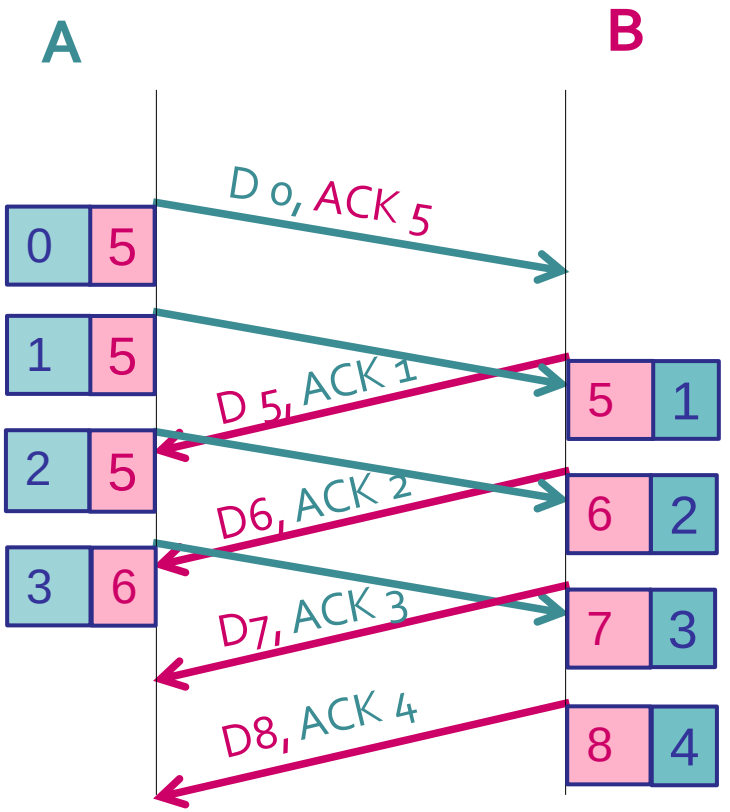
Ventana de Recepción

- **Vuelta atrás**
  - Recepción ordenada (tamaño 1)
- **Retransmisión selectiva**
  - Recepción fuera de orden (tamaño k, 1< k <= N)
- Reconocimientos acumulativos

Transmisión bidireccional

- Los protocolos estudiados pueden funcionar en ambos sentidos
- Cada extremo actúa simultáneamente como transmisor y como receptor
- Este modo de funcionamiento permite enviar los **reconocimientos superpuestos** (*piggybacking*)

- (Sólo hemos dibujado los paquetes que se envían, no los que se reciben)
- Las flechas muestran el tránsito del último bit



Índice

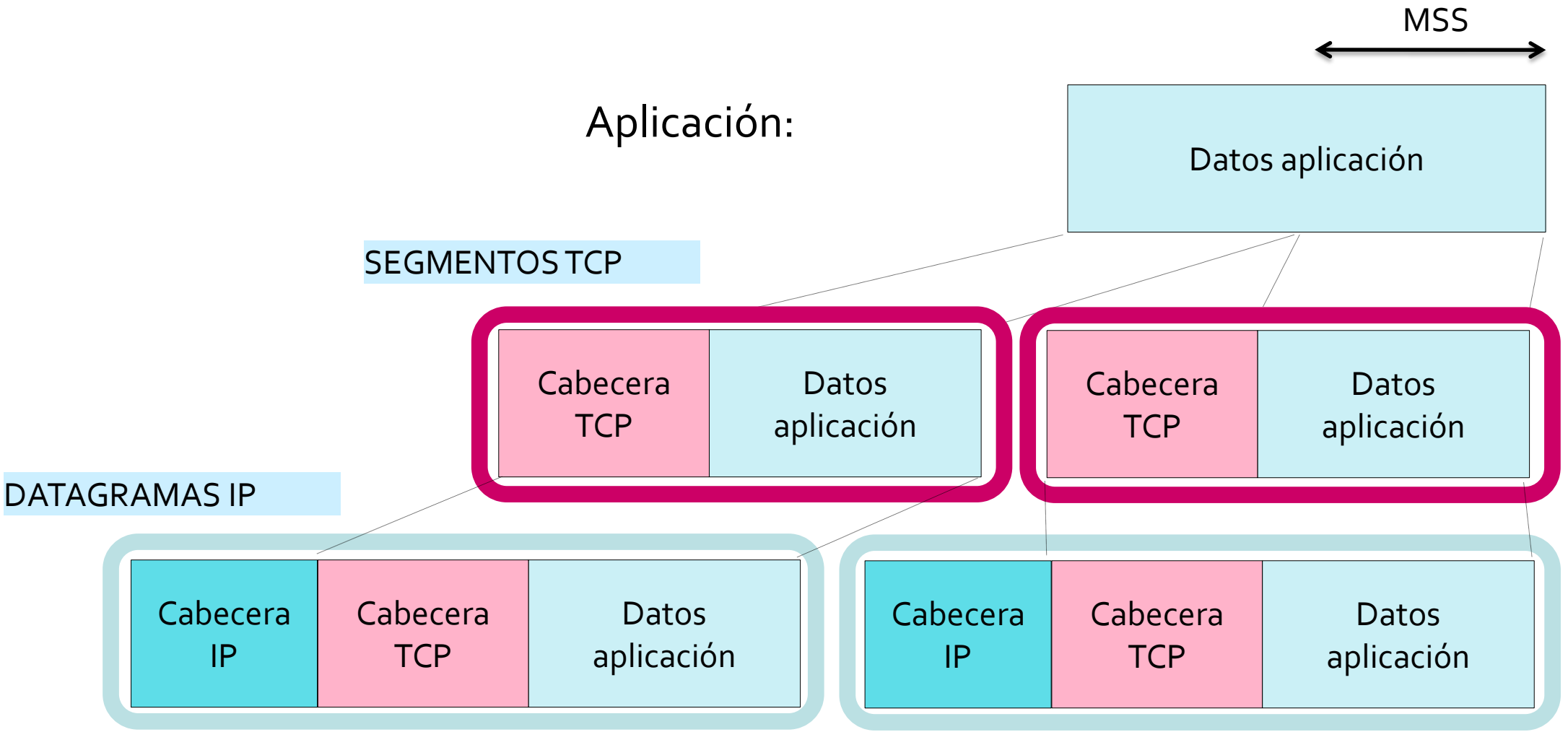
1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP
3. Fundamentos de la transferencia fiable de datos
4. Transporte orientado a la conexión: TCP
   1. Concepto
   2. Formato de un segmento
   3. Control de flujo y de error
   4. Gestión de una conexión TCP
   5. Opciones TCP
   6. El control de la congestión TCP

A10

TCP (*Transmission Control Protocol*, RFC's 793, 1122, 1323, 2018, 2581)

- Servicio orientado a la conexión:
  - Punto a punto
  - Control de flujo
  - Servicio fiable (retransmisiones)
- Flujo de bytes ordenado entre aplicaciones
  - No tiene en cuenta las fronteras entre mensajes
- Datos full dúplex
  - Flujo de datos bidireccional en la misma conexión
  - MSS: *maximum segment size*
- Control de la congestión
- Mucha información de estado asociada a la conexión!!

Encapsulado de TCP



MSS=Maximum Segment Size

Problemas de IP y *soluciones de TCP*

IP

- Longitud datagrama limitada
- IP = política sin garantías
  1. Pérdida de datagramas
  2. Entrega desordenada y duplicados
  3. Errores en datagramas
- No identifica el proceso
- Congestión en la red

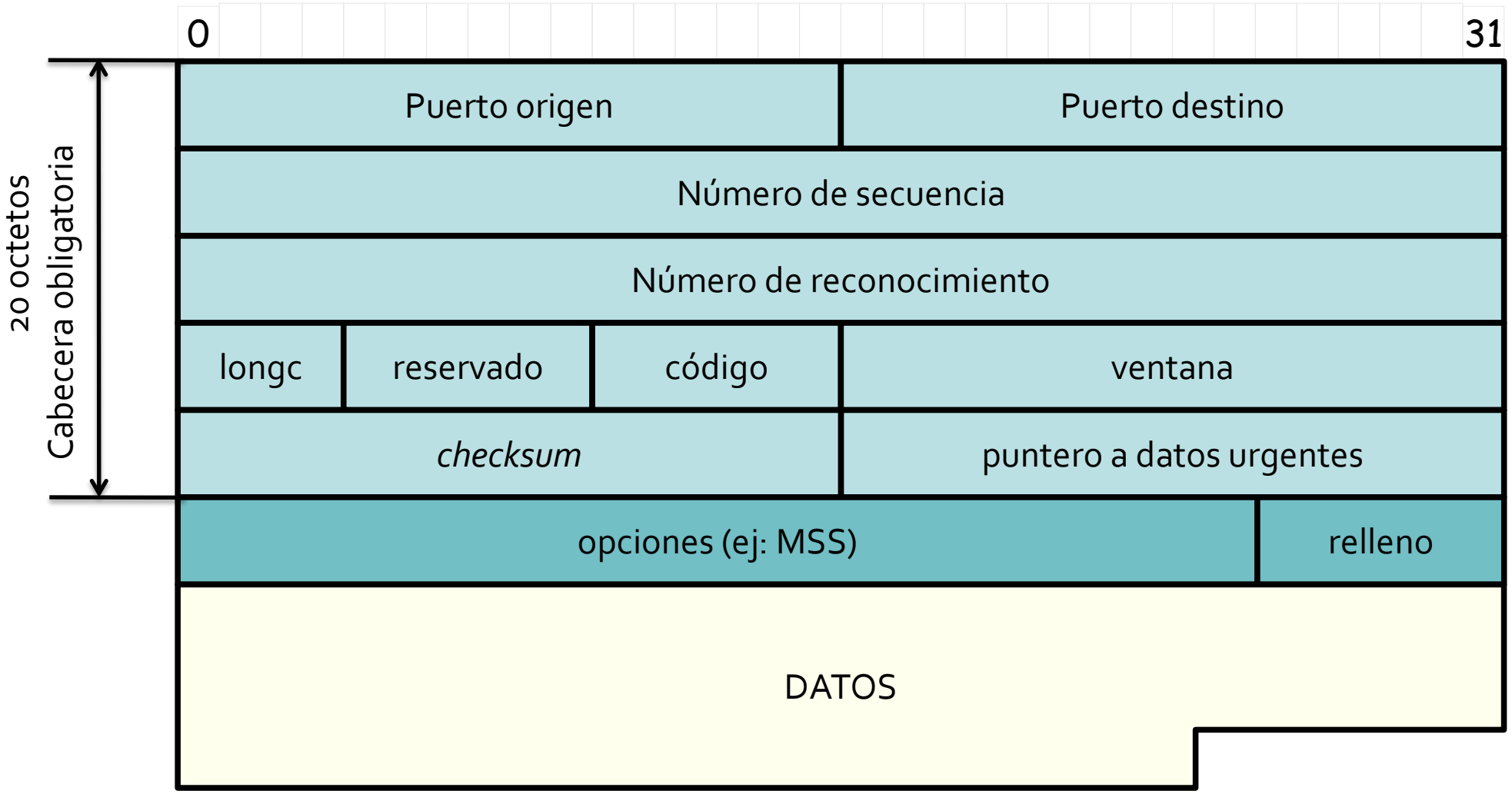
TCP

- Segmentación
- Control de flujo y de error
  1. Retransmisiones
  2. Numeración
  3. *Checksum*
- Puertos
- Control de la congestión

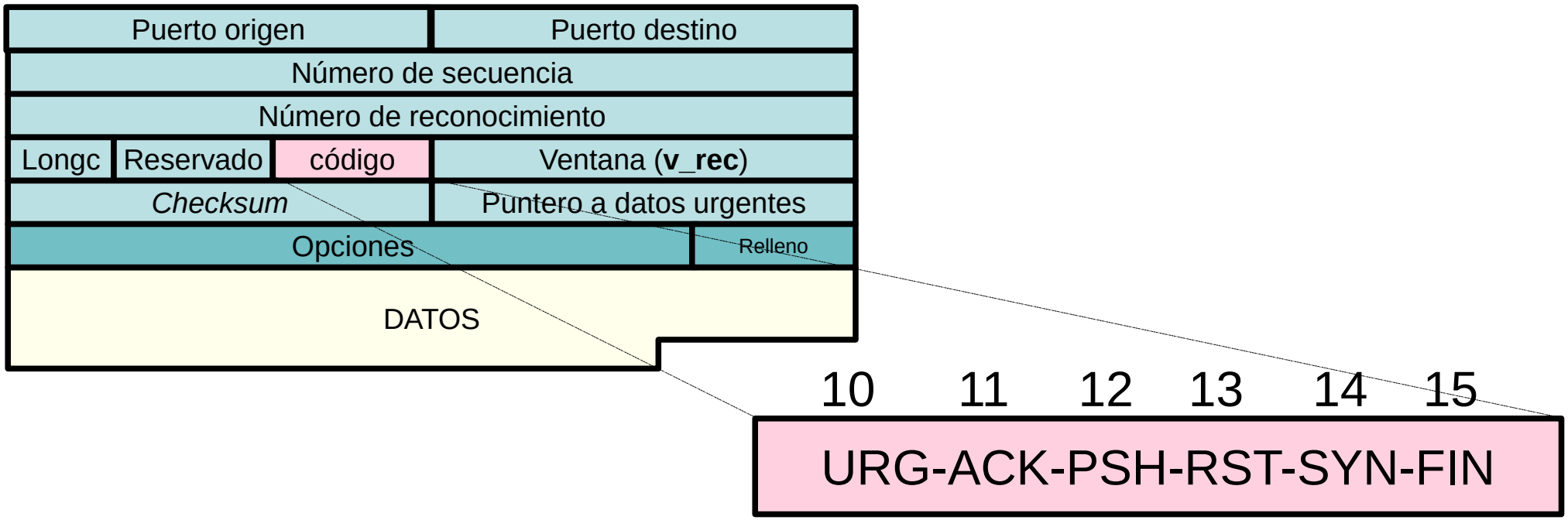
4.2. Formato de un segmento

La cuenta de datos se lleva en bytes:

- Todos los campos que se refieran a los datos contarán en bytes



Campo de código



URG	Puntero a datos urgentes es vàlido
ACK	El campo de reconocimiento es vàlido
PSH	El segmento solicita una operación PUSH
RST	Cierre abrupto de la conexión
SYN	Solicitud de inicio de conexión
FIN	Solicitud de cierre de conexión

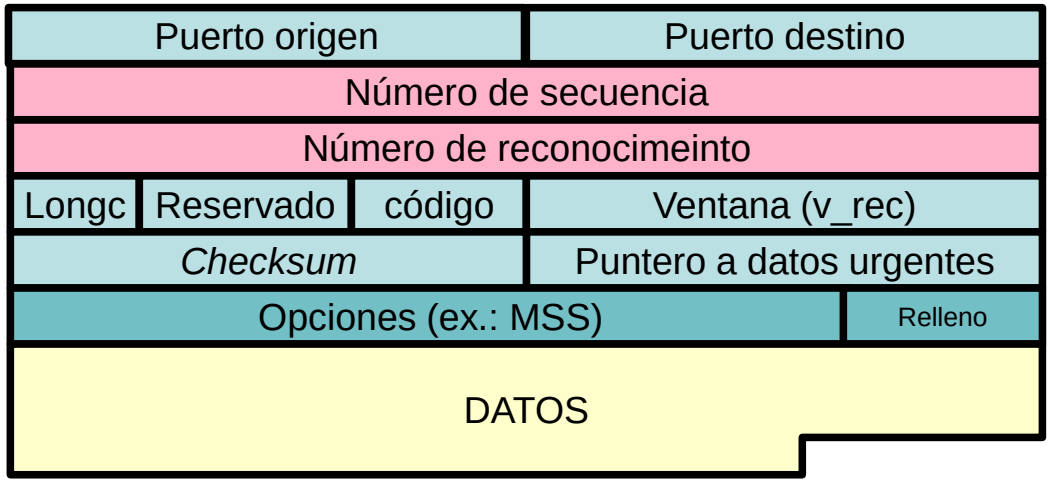
Poco empleado

Checksum en TCP

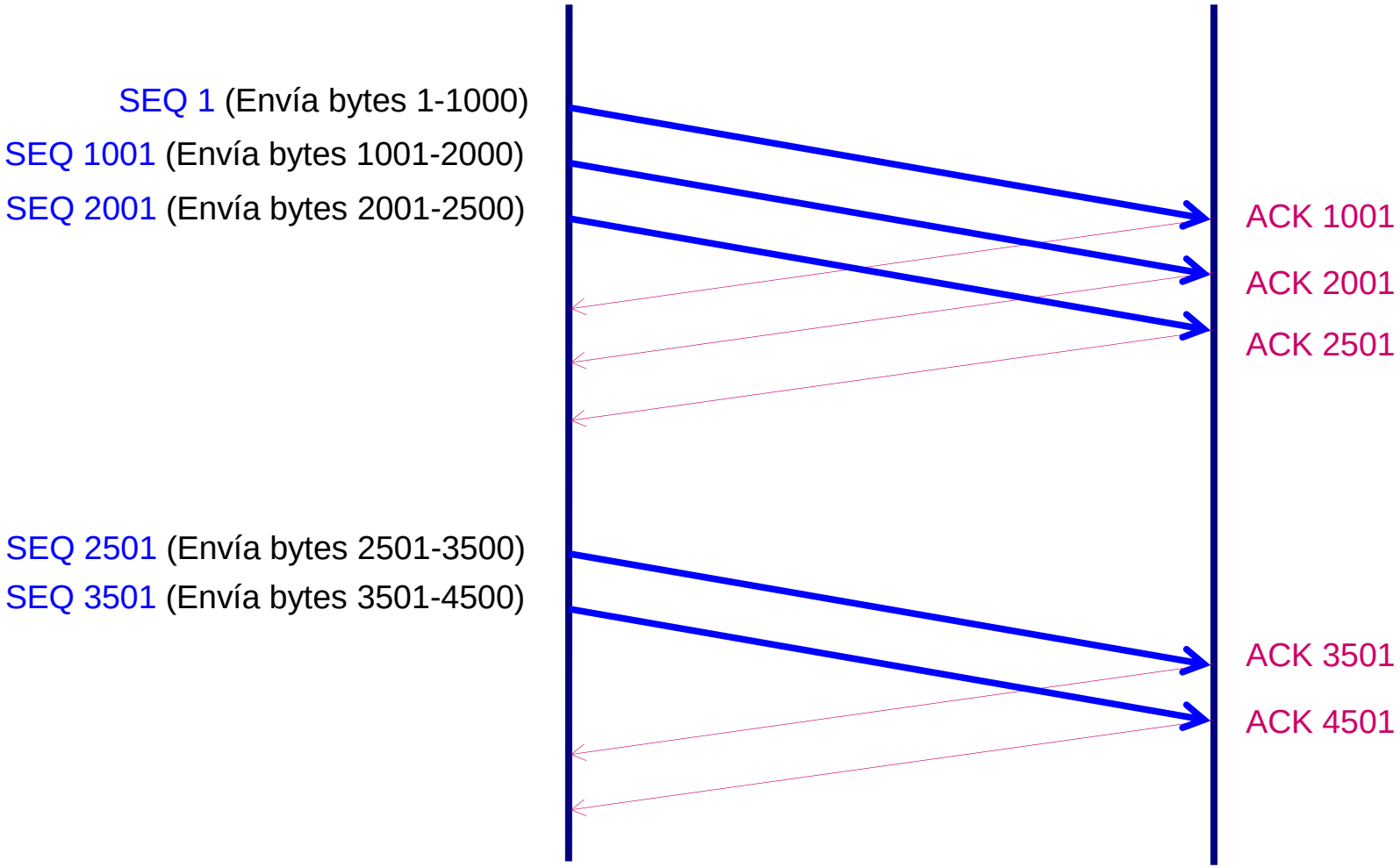
- Asegura que los segmentos no han sido modificados en tránsito
- Cálculo similar al *checksum* UDP
- Un segmento dañado se descarta
  - Tendrá que ser retransmitido

Numeración de segmentos

- En TCP se utilizan números de 32 bits
  - Número de secuencia:
    - Indica el número de orden del **primer byte de datos** que viaja en el segmento
  - Número de reconocimiento:
    - Indica el **byte** que el receptor espera recibir
    - Es **acumulativo**
    - Válido si el *flag* ACK = 1
  - Reconocimientos superpuestos (*piggybacking*)



Ejemplo: numeración de segmentos



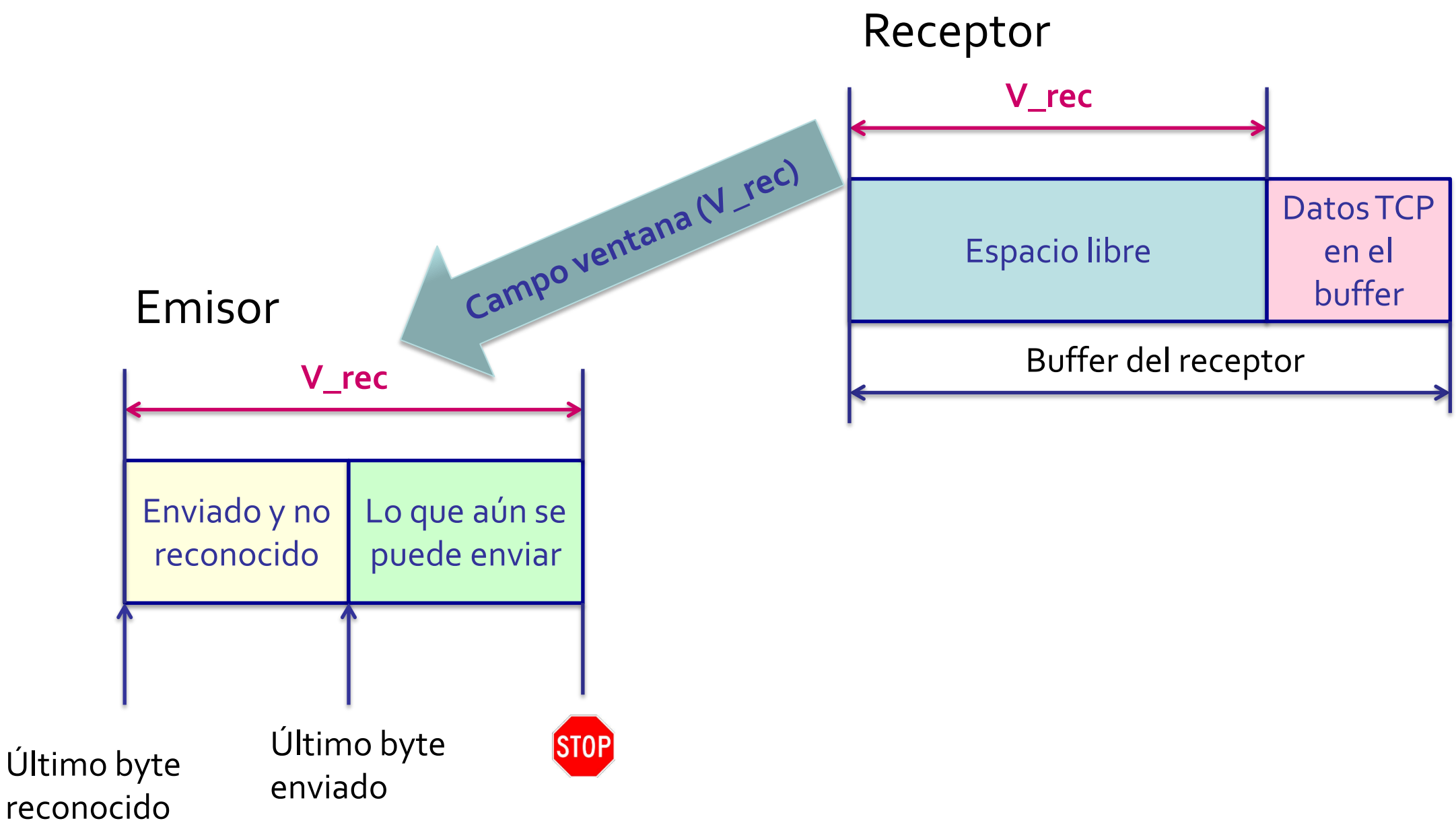
Como el tránsito de datos es bidireccional* los reconocimientos pueden viajar **superpuestos** (*piggybacking*)

**aunque no se muestra así en la figura*

4.3 Control de flujo y de error

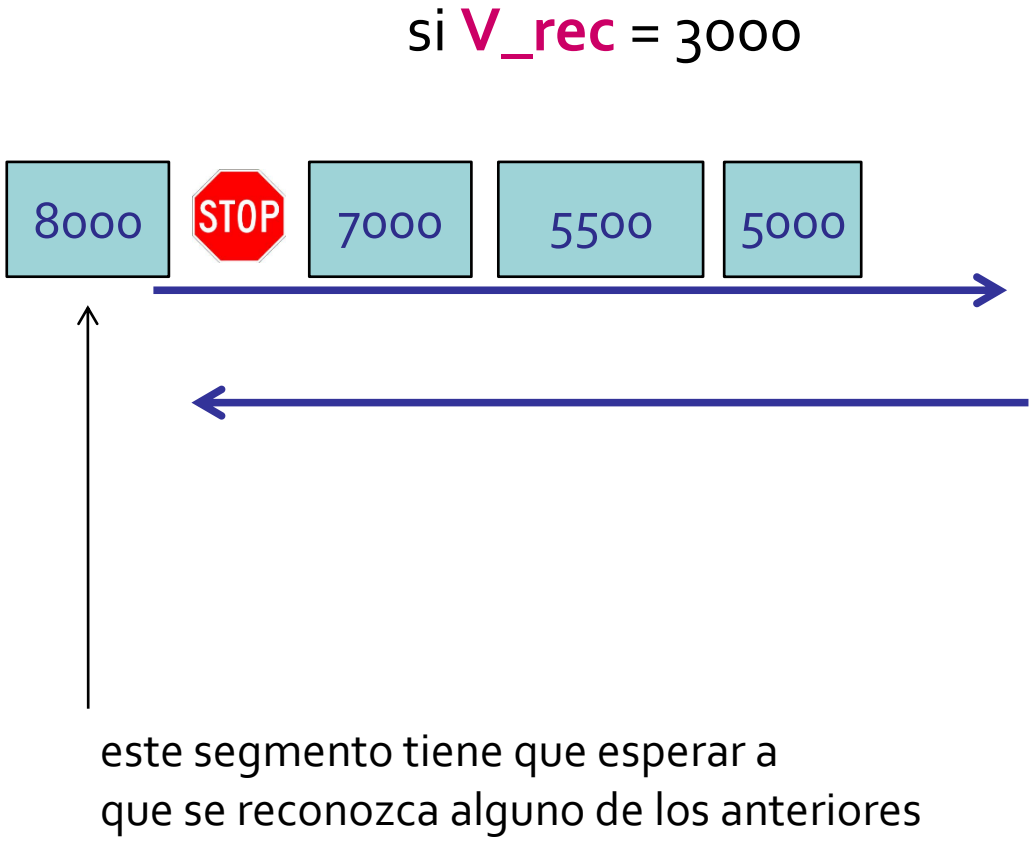
- Cada extremo dispone de un buffer de recepción
- Los datos recibidos correctamente y en orden se almacenan en el buffer
- El proceso de aplicación lee los datos "a su ritmo"
  - Si es lento en las lecturas, el emisor podría provocar un desbordamiento del buffer
  - Solución: control de flujo mediante una ventana de recepción (V_rec)
    - La ventana de recepción indica el espacio libre en el receptor
      - Es dinámica y puede tomar el valor cero

Control de flujo TCP



Control de flujo TCP

- La ventana de transmisión no puede ser mayor que la de recepción
  - No se envía ningún dato que no "quepa" en el receptor
  - No se pueden aceptar nuevos datos del nivel superior si se alcanza el límite de la ventana



Control de flujo TCP

- ¿Qué pasa si el receptor se queda sin espacio?
  - V_rec = 0
  - Problema
    - Posible bloqueo del emisor
  - Solución
    - El emisor puede aún enviar segmentos con 1 byte de datos
- Comparación con UDP
  - ¡¡Posible desbordamiento!!

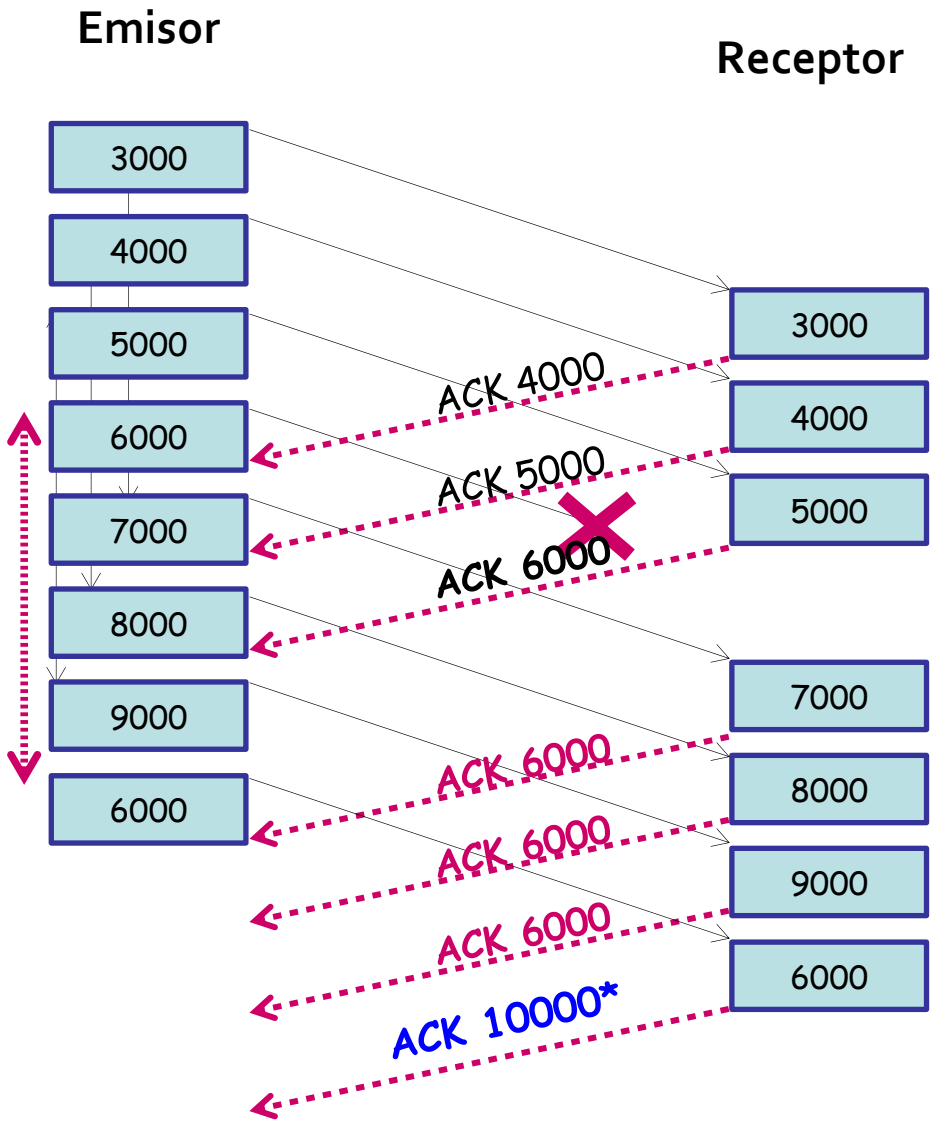
Demostración: Kurose

Control de error en TCP

- Utiliza ARQ:
  - Reconocimientos (ACK) y retransmisiones
- Detección de los segmentos erróneos: *checksum*
- **Ventana deslizante**
  - Ventana de transmisión de tamaño máximo variable
    - Fijado por el receptor (campo ventana, V_rec)
  - Si los datos no llegan en orden
    - Se pueden descartar: vuelta atrás
    - Se pueden aceptar: retransmisión selectiva

Reconocimientos en caso de pérdidas

- Reconocimientos acumulativos:
  - Cuando se recibe un segmento dañado, los segmentos siguientes NO se pueden reconocer!!!
- Tras la recepción de los segmentos 3000, 4000, 5000, 7000, 8000 y 9000 el reconocimiento será 6000
- Cada recepción puede generar un reconocimiento



* Suponiendo retransmisión selectiva

Generación de reconocimientos

- No es necesario enviar un ACK por cada segmento recibido
- Reconocimientos retardados (RFC 1122, 2581)
  - El ACK se puede retrasar hasta:
    - Recibir otro segmento (se reconocen uno de cada dos)
    - Enviar un segmento de datos en sentido contrario
    - Que venza un temporizador (cada 500 milisegundos)
- Se reduce el tráfico de reconocimientos

Envío de datos: temporizador

- Al enviar un segmento se pone en marcha un temporizador (si no está ya en marcha)
  - Normalmente se emplea un único temporizador por RTT
- El reconocimiento del segmento desactiva su temporizador
- Si vence el temporizador se produce el reenvío de al menos un segmento
  - Todos en vuelta atrás

Temporizador y RTT

- ¿Cuál es el tiempo adecuado de espera para el temporizador (intervalo de *timeout)*?
  - Un intervalo mayor que el RTT
  - Pero …
    - El RTT es variable: depende de los retrasos de la red
      - Cantidad de tráfico en cada uno de los *routers*
      - Velocidades de transmisión y propagación de cada una de las líneas empleadas
    - Es imposible calcular el RTT a priori
- Utilizaremos una predicción basada en la experiencia

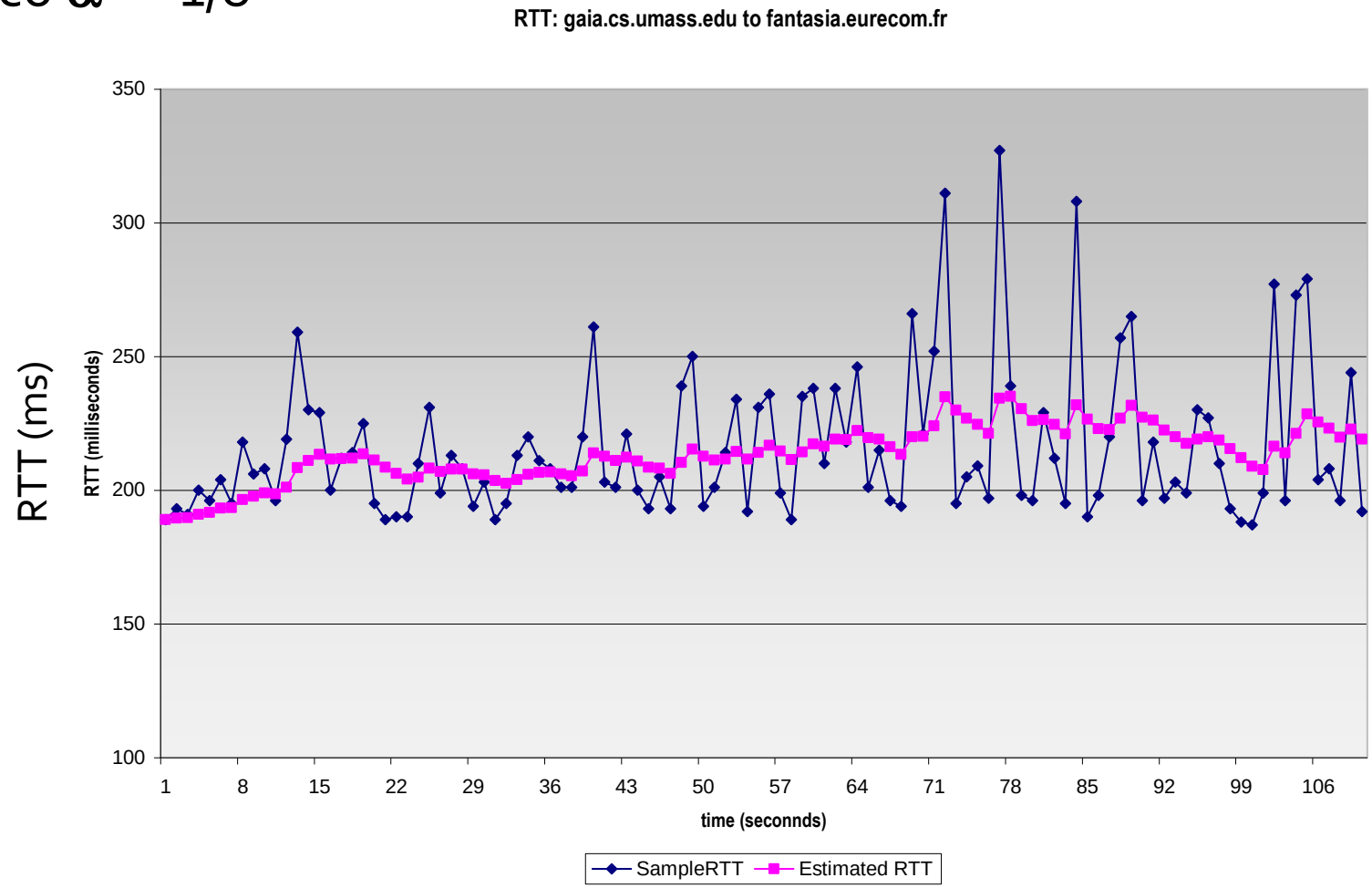
Estimación del RTT (RFC 6298)

- MuestraRTT
  - Tiempo desde que se transmite un segmento hasta que se recibe su ACK asociado
    - Ignora las retransmisiones
  - No se emplea directamente para calcular el RTT porque su valor puede ser atípico
    - Queremos una medida más suave que tenga en cuenta la historia anterior
      - La media es demasiado lenta en adaptarse a los cambios

Ejemplo: estimación del RTT

$$\texttt{EstimacionRTT} = (1-\alpha)*\texttt{EstimacionRTT} + \alpha*\texttt{MuestraRTT}$$

- La influencia de las muestras anteriores decrece exponencialmente rápido
- Valor típico α = 1/8



Tiempo (s)

Estimación del Timeout (RFC 6298)

- Intervalo de timeout: `EstimacionRTT` más un margen de seguridad

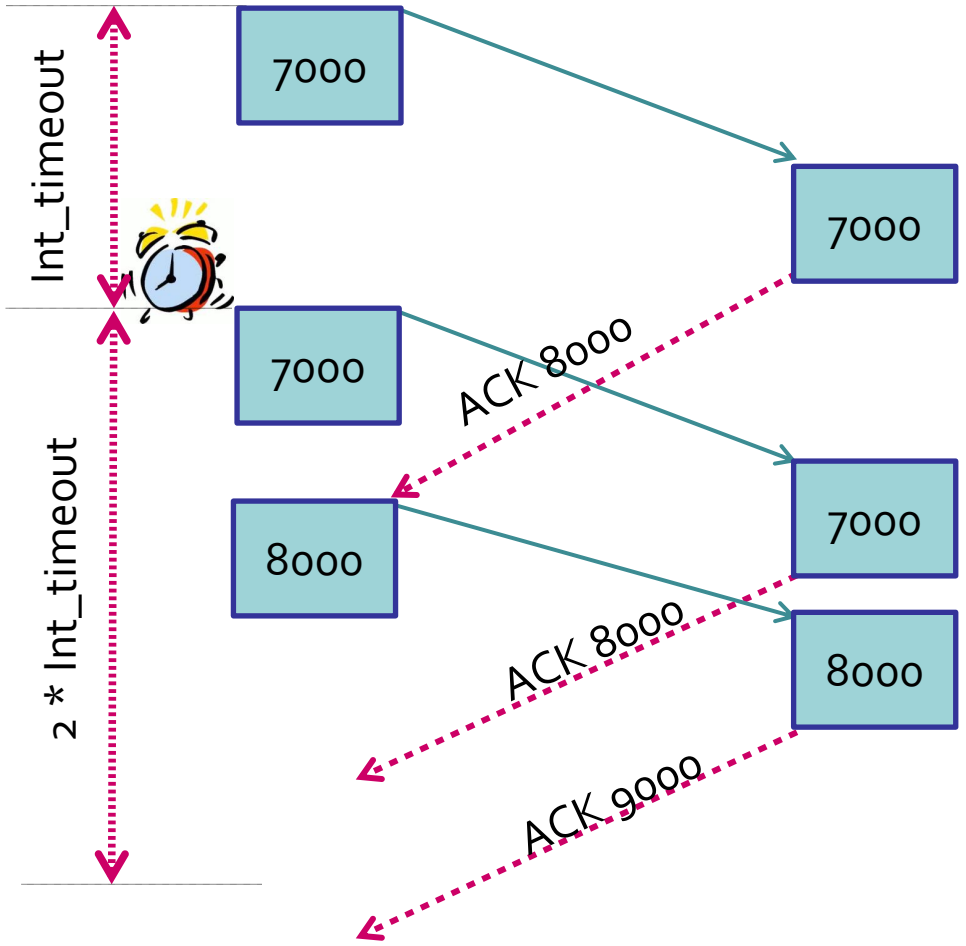
- DevRTT nos da una medida de la variabilidad de la muestra
  - `DevRTT = (1-β)*DevRTT + β*|MuestraRTT- EstimacionRTT|`
  - Valor recomendado β=1/4

"Margen de seguridad"

```
Intervalo_de_timeout = EstimacionRTT + 4 *DevRTT
```

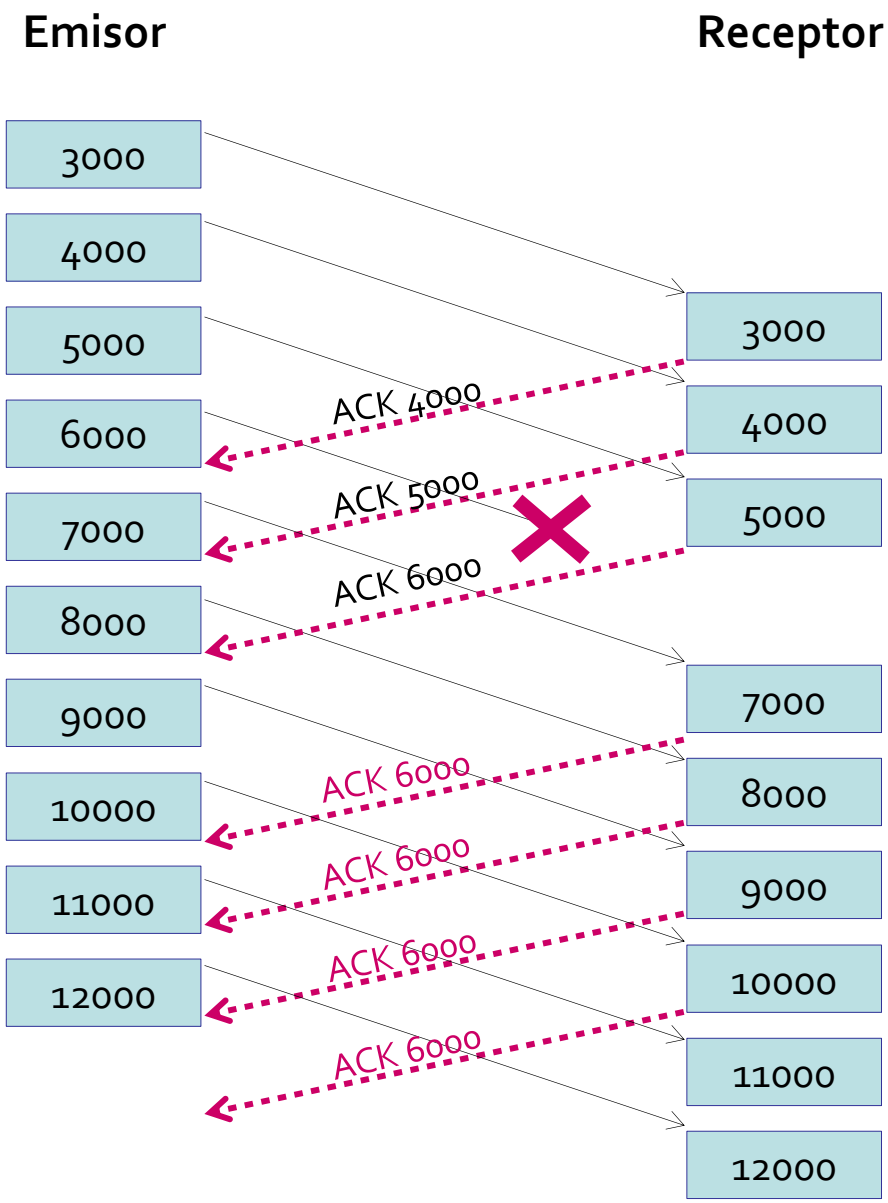
Duplicación del *timeout* (RFC 6298)

- Problema:
  - Ambigüedad en el RTT cuando hay retransmisiones
- Solución:
  - No tener en cuenta las medidas del RTT de los segmentos retransmitidos
  - **Al retransmitir doblar el valor del temporizador**
    - ***Exponential Backoff***



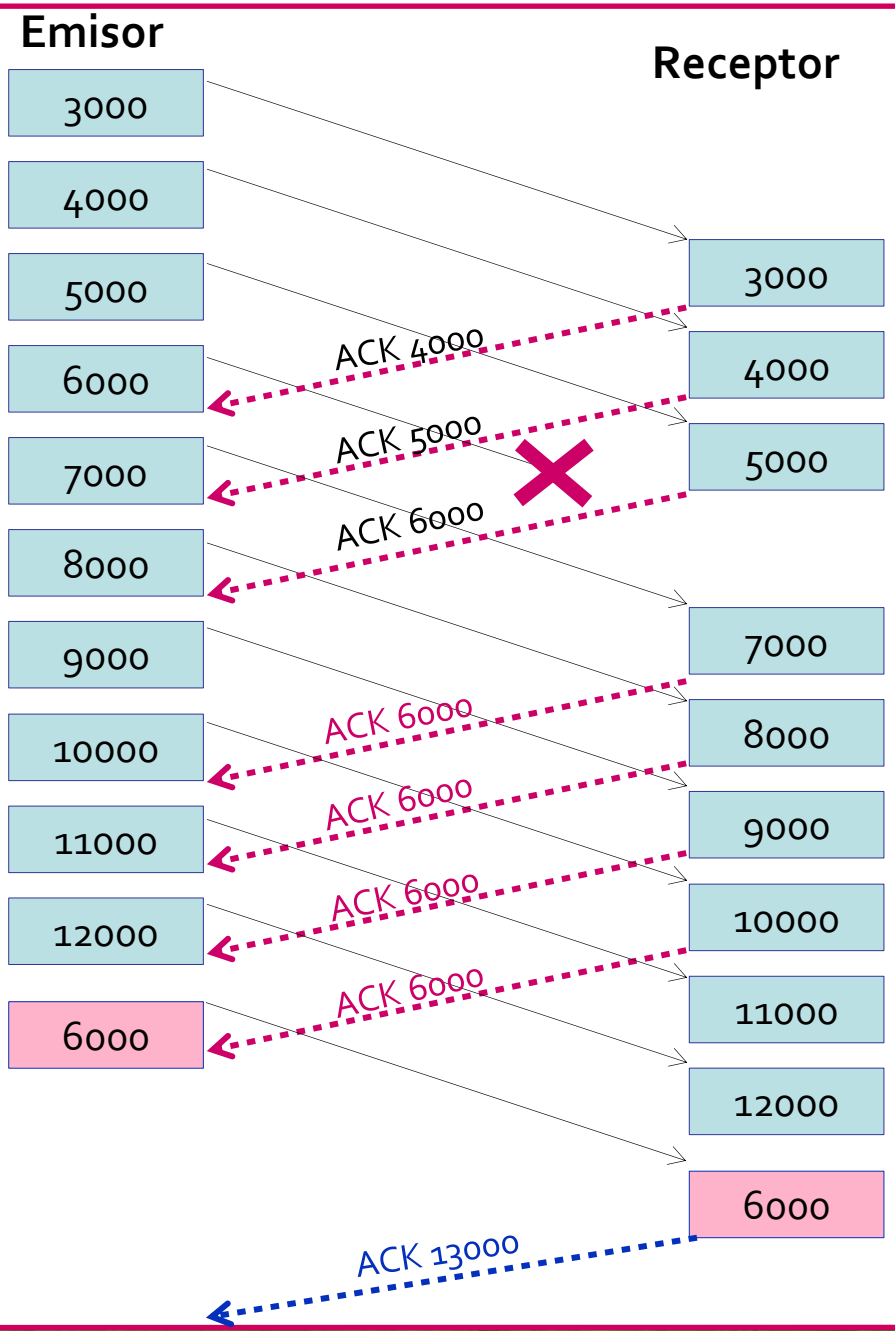
Reconocimientos duplicados

- Se deben a la llegada de un segmento fuera de orden
  - Debido a entrega desordenada o pérdida
- Se envía un ACK por segmento recibido fuera de orden
  - ¡¡Inmediatamente!!



Retransmisión rápida

- El intervalo de timeout suele ser grande
- Se puede acelerar las retransmisiones si cuando se reciben tres o más **ACKs duplicados**…
  - se retransmite el segmento perdido sin esperar a que venza su temporizador (***retransmisión rápida***, RFC 2581).



Índice

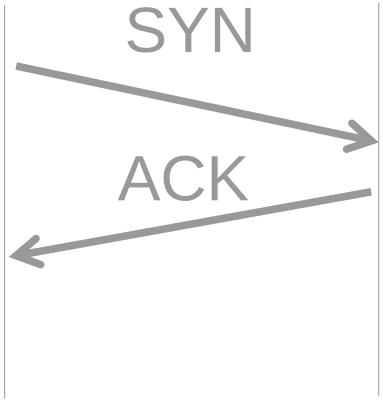
A11

4.4. Gestión de la conexión TCP

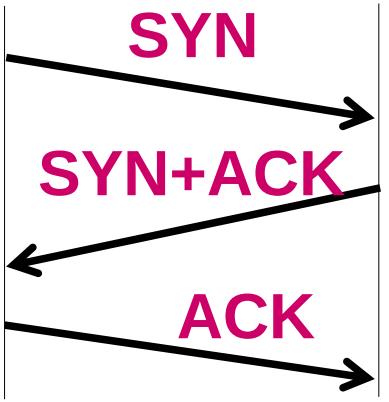
- Establecimiento de conexión
  - `s = new Socket("1.2.3.4",80);`
  - intercambio de mensajes
- Transmisión de datos
  - bidireccional
  - `entrada.read(b)y salida.write(b)`
- Cierre de la conexión
  - `s.close();`

Establecimiento de la conexión TCP

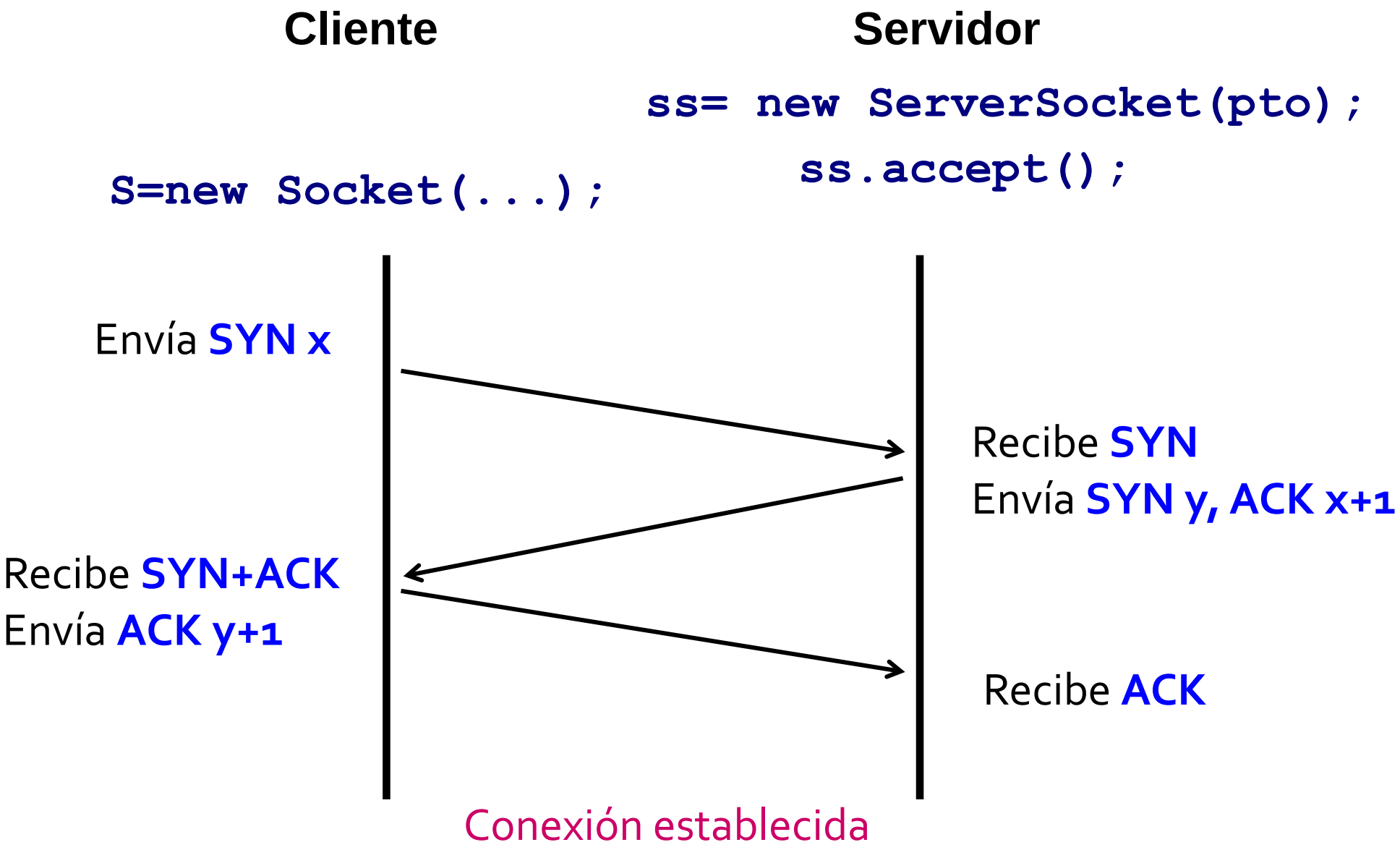
- Protocolo en dos fases
  1) Petición
  2) Aceptación
  - Problema: duplicados retrasados



- Protocolo en tres fases
  1) Petición
  2) Aceptación
  3) Reconocimiento
  - Cada extremo debe pedir y reconocer la petición del otro

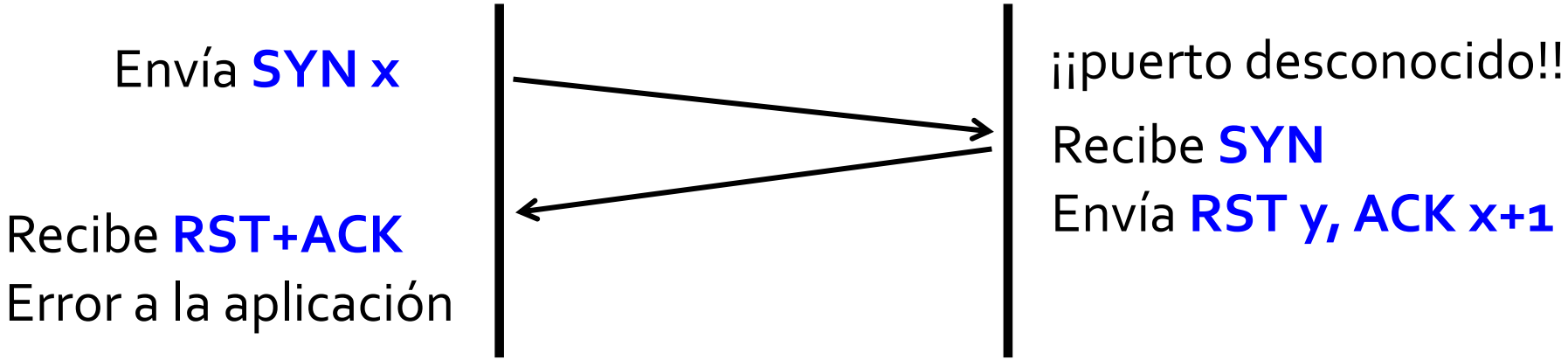


Establecimiento de la conexión TCP (II)



El bit RST

- RESET: aborta la conexión TCP
- Causas:
  - Número de secuencia imposible
  - El puerto destino no está en uso (no **`ServerSocket`**)

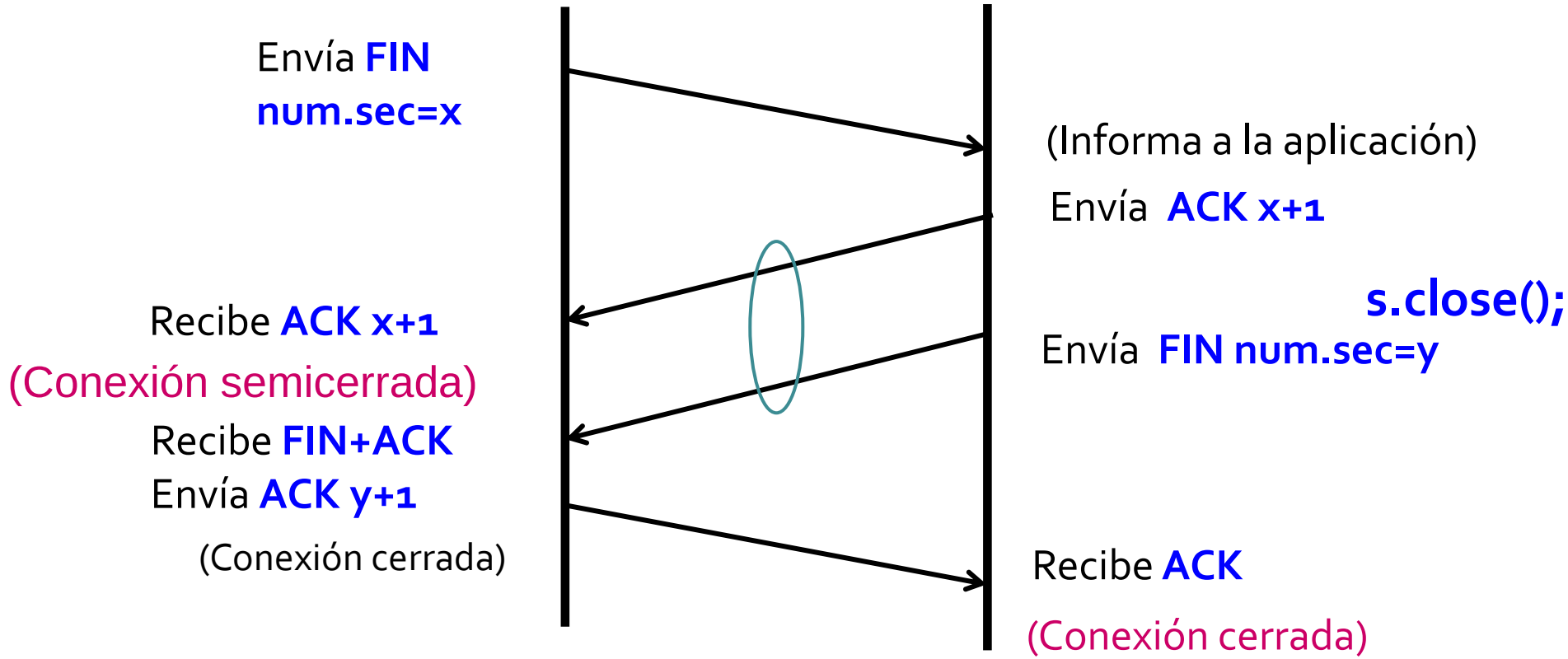


Conexión **NO** establecida

Cierre de la conexión TCP

- Cierre de los dos flujos de datos de forma independiente

```
s.close();
```



Ejemplo: Conexión-Desconexión

- A.48209 > B.pop3: S 3237421429:3237421429(0) win 5840 <mss 1460>
  - B.pop3 > A.48209: S 3331773798:3331773798(0) ack 3237421430 win 5792 <mss 1460>
- A.48209 > B.pop3: . ack 3331773799 win 5840

  - B.pop3 > A.48209: P 3331773799:3331773850(52) ack 3237421430 win 5792
- A.48209 > B.pop3: . ack 3331773851 win 5840

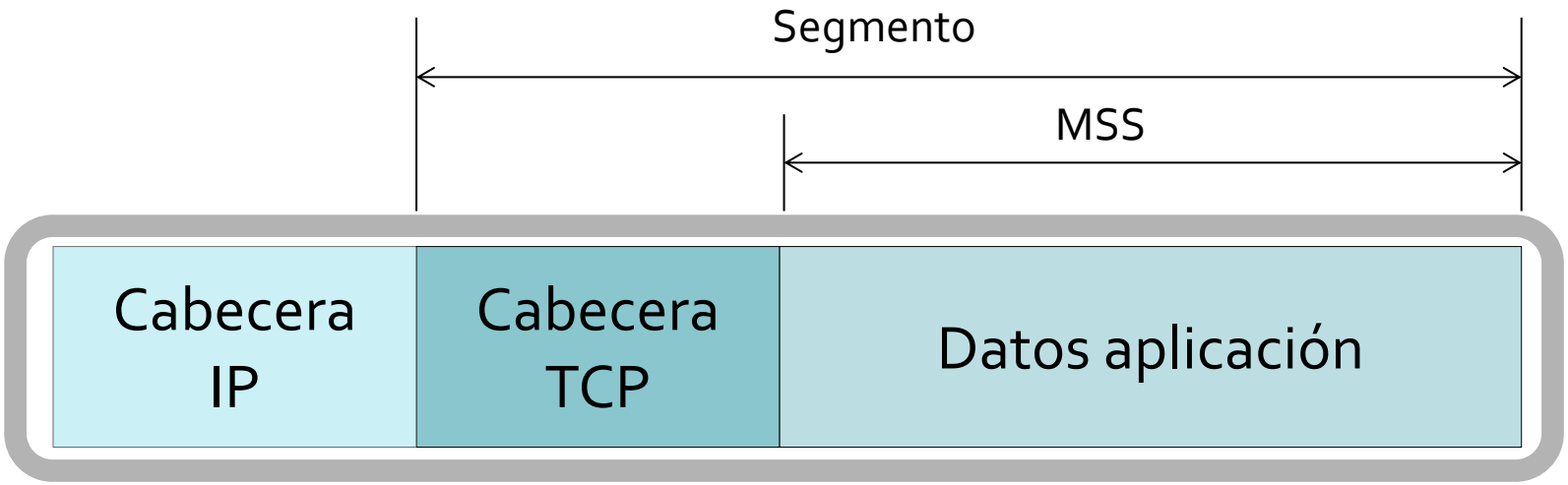
- A.48209 > B.pop3: F 3237421430:3237421430(0) ack 3331773851 win 5840
  - B.pop3 > A.48209: F 3331773851:3331773851(0) ack 3237421431 win 5792
- A.48209 > B.pop3: . ack 3331773852 win 5840

Índice

Opciones TCP

- Principales opciones TCP:
  - Tamaño máximo de segmento (MSS)
  - Escala de ventana
  - Marcas de tiempo (*timestamp*)
  - Reconocimientos selectivos

Opciones TCP: *Maximum Segment Size* (MSS)



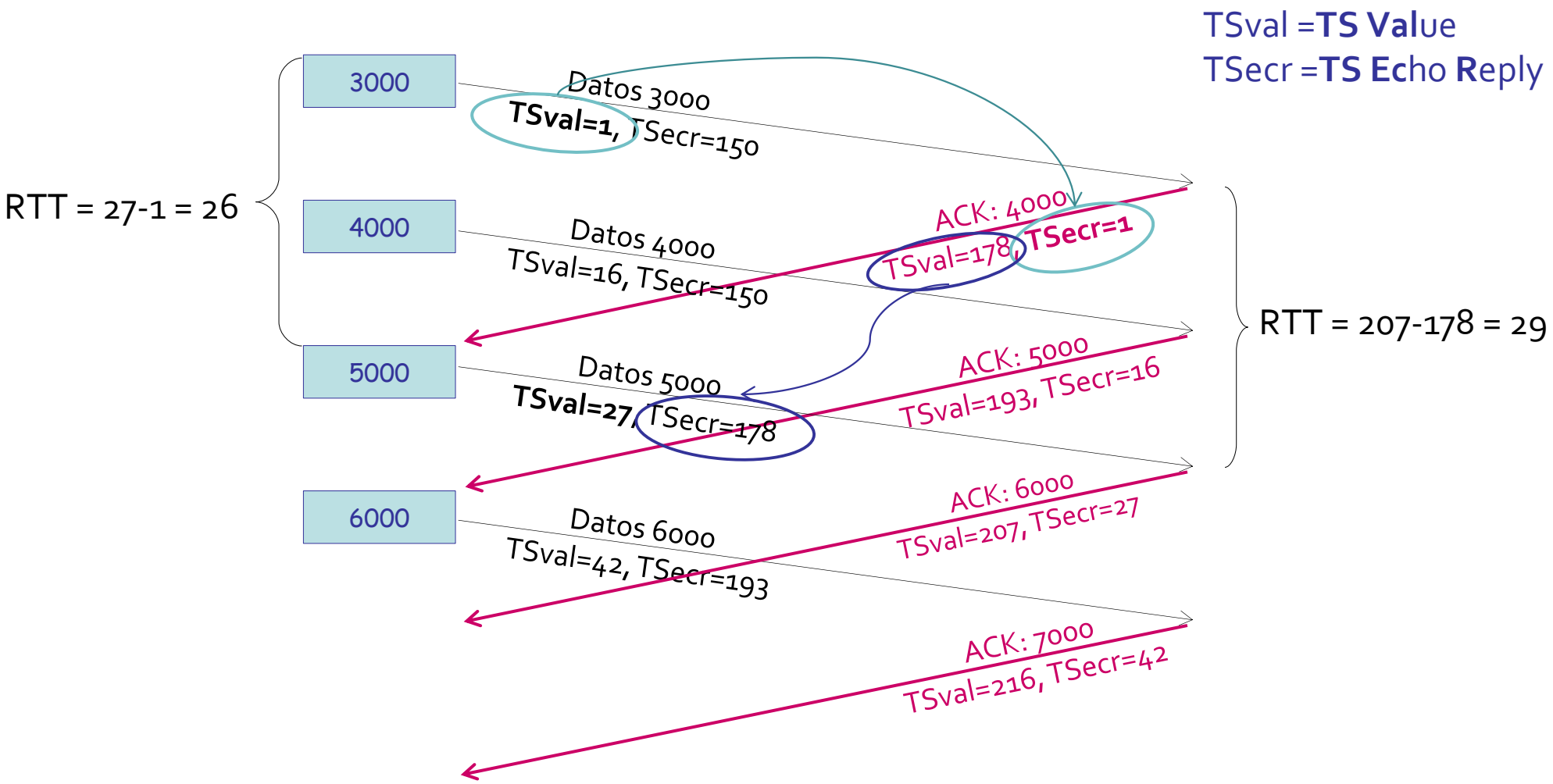
- Intenta evitar la fragmentación de IP
- Cada extremo anuncia su MSS al establecer la conexión
  - No se le pueden enviar segmentos mayores que MSS
- Por defecto MSS = 536 octetos

Opciones TCP: escala de ventana

- En TCP, la longitud máxima de ventana = 65.536 bytes
- El envío continuo requiere ventana >= RTT*Vtrans
- El tamaño máximo de la ventana puede ser una limitación importante en redes de alta velocidad:
  - Si RTT=1ms y Vtrans=1 Gbps
    - RTT*Vtrans=10^6 bits=125.000 bytes
- Con esta opción se especifica un k, ventana=2^k x ventana
  - K <=14 → Máxima ventana escalada = 1 GB

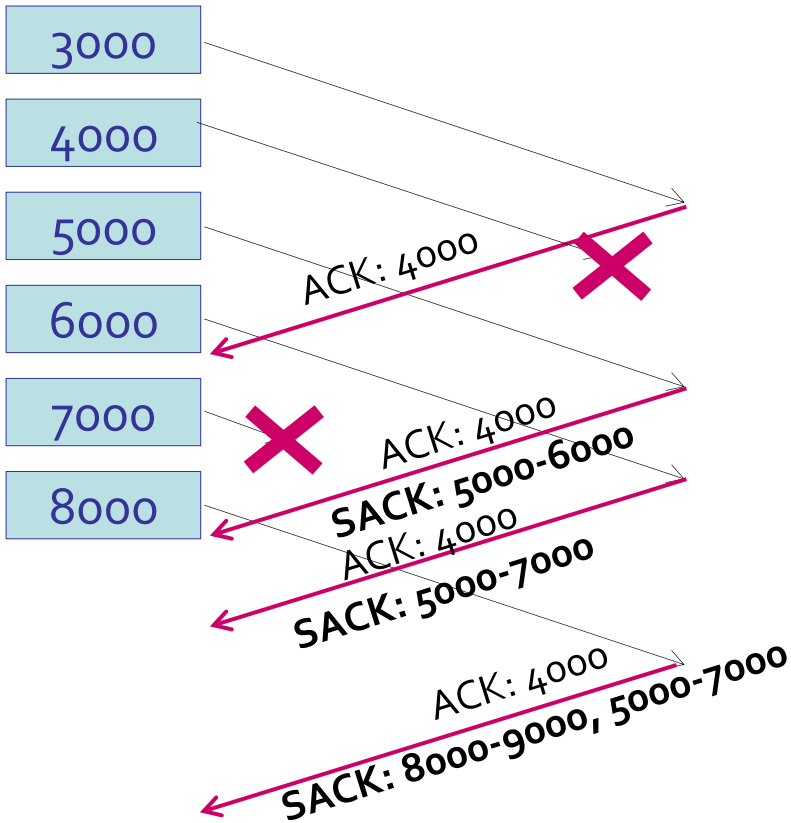
Opciones TCP: *TimeStamp*

- Permite calcular el RTT de forma más precisa (RFC 1323)



Opciones TCP: ACKs selectivos

- Permite reconocer bloques de datos no contiguos (RFC 2018)
- Información SACKs:
  - Aparece primero el último bloque recibido (max. 4)
  - El límite superior del bloque SACK indica el siguiente octeto de datos que se espera



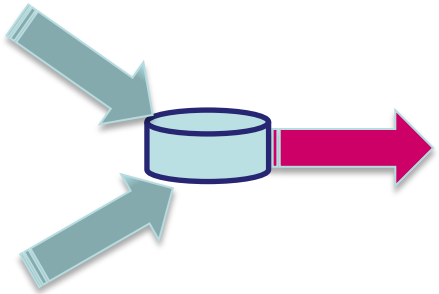
Índice

A_{12}

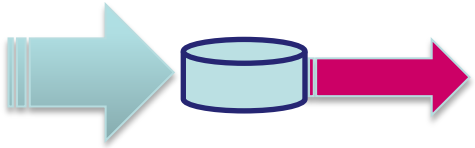
La congestión en la red

Causas básicas

- Enlaces compartidos en los routers



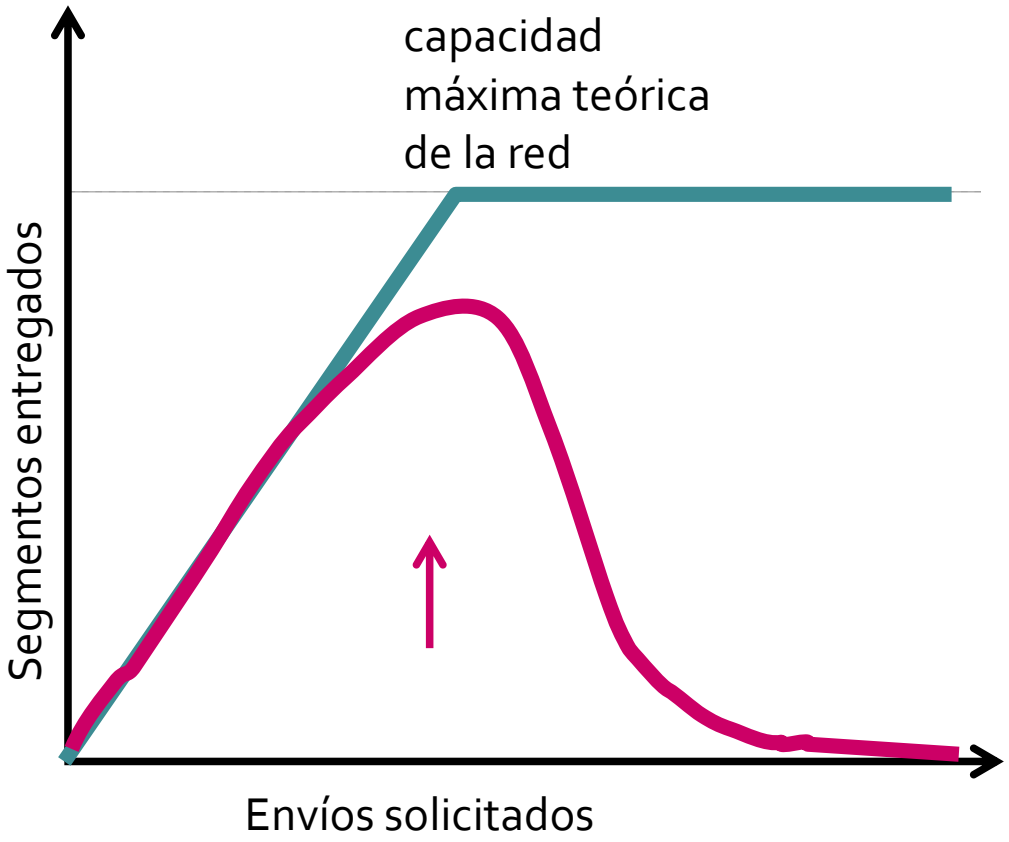
- Tránsito a enlaces con menor ancho de banda que los precedentes



Consecuencias

- Grandes retardos en las colas cuando la tasa de llegada se aproxima a la capacidad del enlace
  - El incremento del RTT puede generar retransmisiones innecesarias
- Descarte de paquetes
  - Desperdicio de los recursos ya empleados
- Retransmisiones:
  - Incrementan aún más el tráfico de la red

¿Qué sucede con la congestión?



A más tráfico, más segmentos se pierden, y más **retransmisiones** se suman al tráfico de los nuevos segmentos a enviar por primera vez.

Control de la congestión en TCP

- El comportamiento de TCP no sólo garantiza la fiabilidad de la comunicación
- Además pretende:
  - Reducir la congestión
    - Sin disponer de información proporcionada por IP
    - Sólo con la información que se detecta en el terminal
  - Propiciar un reparto equitativo de la capacidad disponible entre todos los flujos TCP presentes

Cuestiones para el emisor

- El emisor deber resolver tres problemas:
  1. Cómo limitar el envío de tráfico
  2. Cómo detectar que existe congestión
  3. Cómo variar la tasa de transmisión en función de la congestión percibida

1. Cómo limitar el envío de tráfico

- Se basa en un límite adicional:
  - Ventana de congestión (VentanaCong)
    - Corresponde a una estimación de cuánto tráfico puede transmitir la red para cada conexión
    - La ventana de congestión depende de las condiciones de la red
    - La ventana de congestión limita el tamaño máximo de la ventana de transmisión (bytes de datos que se pueden enviar)
    - **VentanaTransMax = mínimo(V_rec,VentanaCong)**

Los dos problemas restantes

2. Cómo detectar que existe congestión

- TCP supone que la pérdida de un segmento se debe siempre a la congestión
  - Dos posibilidades:
    - Vencimiento de un temporizador (*timeout*)
    - Recepción de tres ACKs duplicados
  - ¿Debe reaccionarse igual?¿Tienen la misma gravedad?

3. Principios de funcionamiento

- Indicio negativo: pérdida de segmento
  - Reducir la tasa de envío
- Indicio positivo: recepción de un ACK que reconoce datos no reconocidos previamente
  - Incrementar la tasa de envío

Gestión de la ventana de congestión

- Tres algoritmos distintos para el emisor que intentan evitar la congestión de la red:
  - **Arranque lento** (*Slow Start*):
    - Cómo iniciar el flujo de datos a través de una conexión
  - **Evitación de la congestión** (*Congestion Avoidance*)
    - Aumento paulatino en caso de ventanas "grandes"
  - **Recuperación rápida** (*Fast Recovery*)
    - Cómo actuar cuando se pierde un paquete
    - Pérdidas por errores de transmisión < 1%
- En la práctica se implementan juntos

Arranque lento *(Slow Start)*

- Al comenzar la conexión se podría transmitir todos los segmentos de V_rec
- Sin embargo se comienza "lentamente":
  - Inicialmente, VentanaCong = 2 MSS
- Crecimiento exponencial, por cada reconocimiento recibido (que reconozca nuevos datos):
  - VentanaCong = VentanaCong + 1 MSS

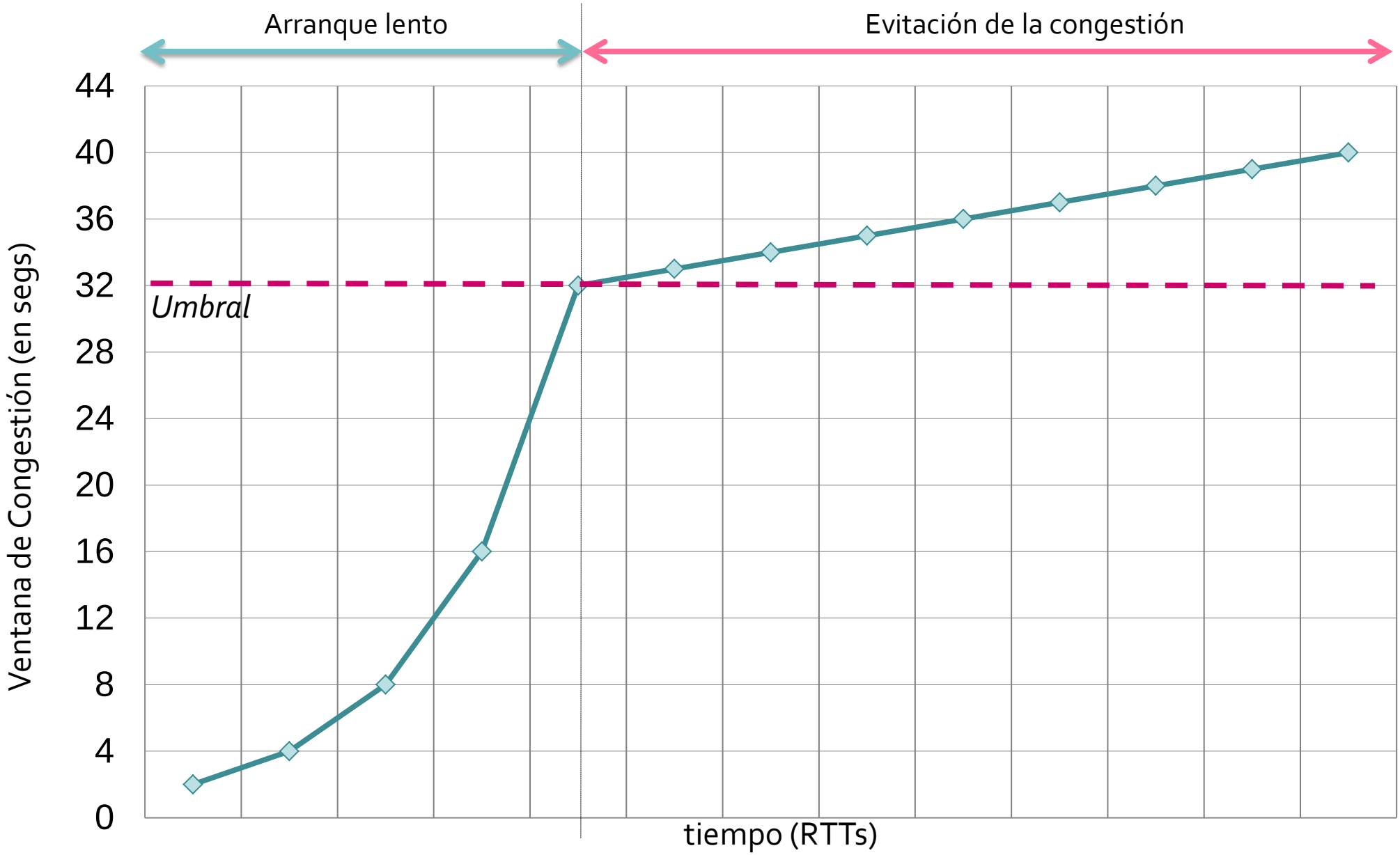
Evitación de la congestión

- Un límite adicional, Umbral
  - Inicialmente puede ser igual a V_rec
- A partir de Umbral el crecimiento es lineal
  - Un MSS cuando se reciben todos los ACKs de la ventana de congestión
  - Muchas implementaciones realizan incrementos parciales, por cada ACK recibido
- El crecimiento se mantiene hasta que se detectan problemas:
  - Vence un temporizador
  - Se reciben 3 ACKs duplicados
- Entonces, se aplica recuperación rápida

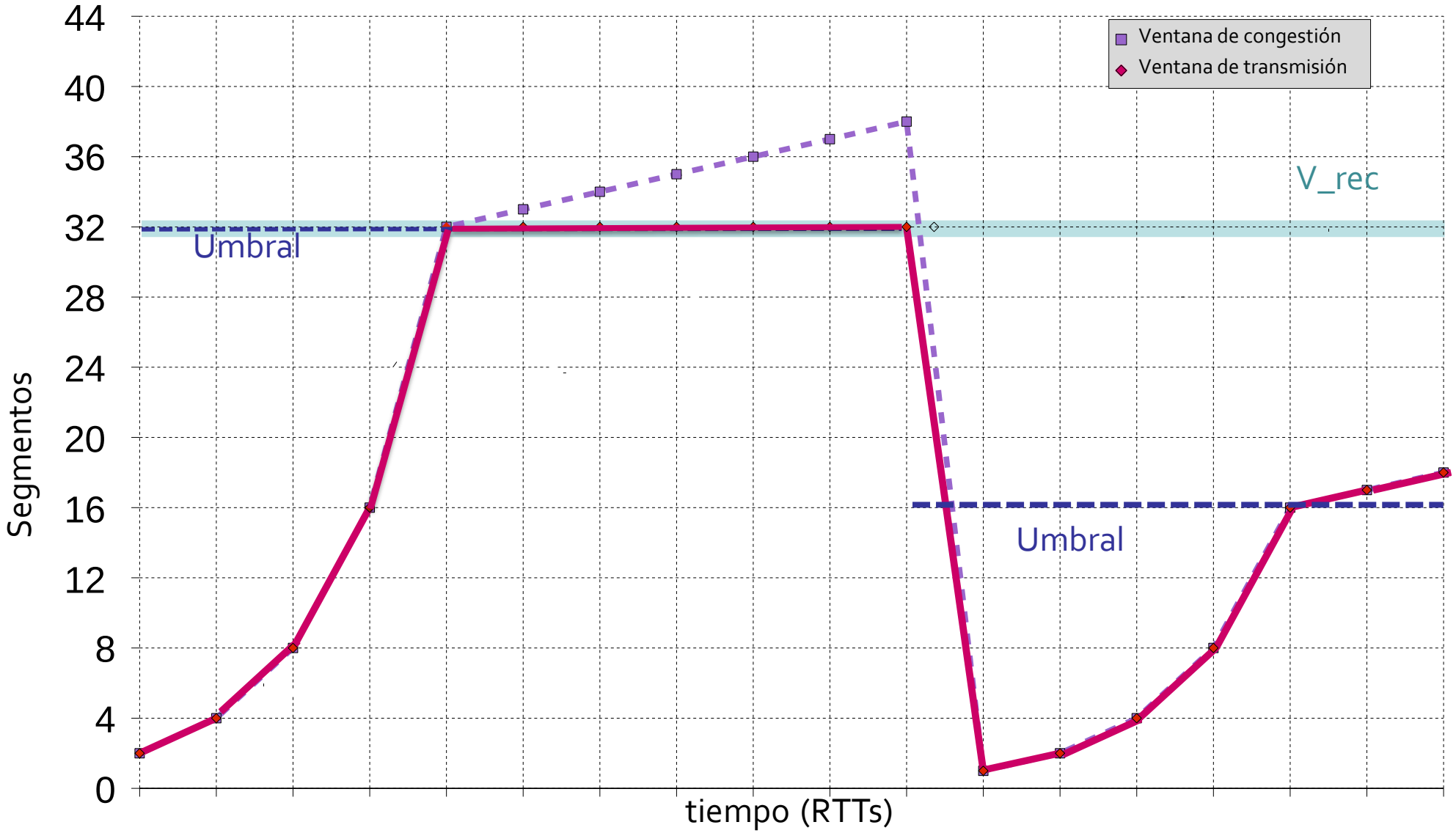
Recuperación rápida

- Si congestión (pérdida o 3 ACK´s duplicados):
  - Umbral = max(ventana_tx/2, 2 segmentos)
  - Si vence temporizador (pérdida)
    - VentanaCong = 1 segmento
  - Si 3 ACKs duplicados
    - VentanaCong = Umbral

Evitación de congestión: ejemplo



Ajuste de Umbral (pérdida)



Control de la congestión: resumen

- Cada conexión está limitada por:
  - El receptor: V_rec
  - La estimación del emisor: VentanaCong
- La capacidad de una conexión se estima por:
  - Inicio progresivo: arranque lento y evitación de la congestión
  - Reducción al producirse una retransmisión:
    - Si vence el temporizador, se reduce el Umbral y se aplica arranque lento
    - Si llegan reconocimientos duplicados, se reduce el Umbral y la VentanaCong

Conclusión

- La capacidad de una conexión es desconocida
- TCP descubre esa capacidad mediante varios mecanismos:
  - Arranque lento
  - Evitación de la congestión
- TCP intenta conseguir el mejor rendimiento