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

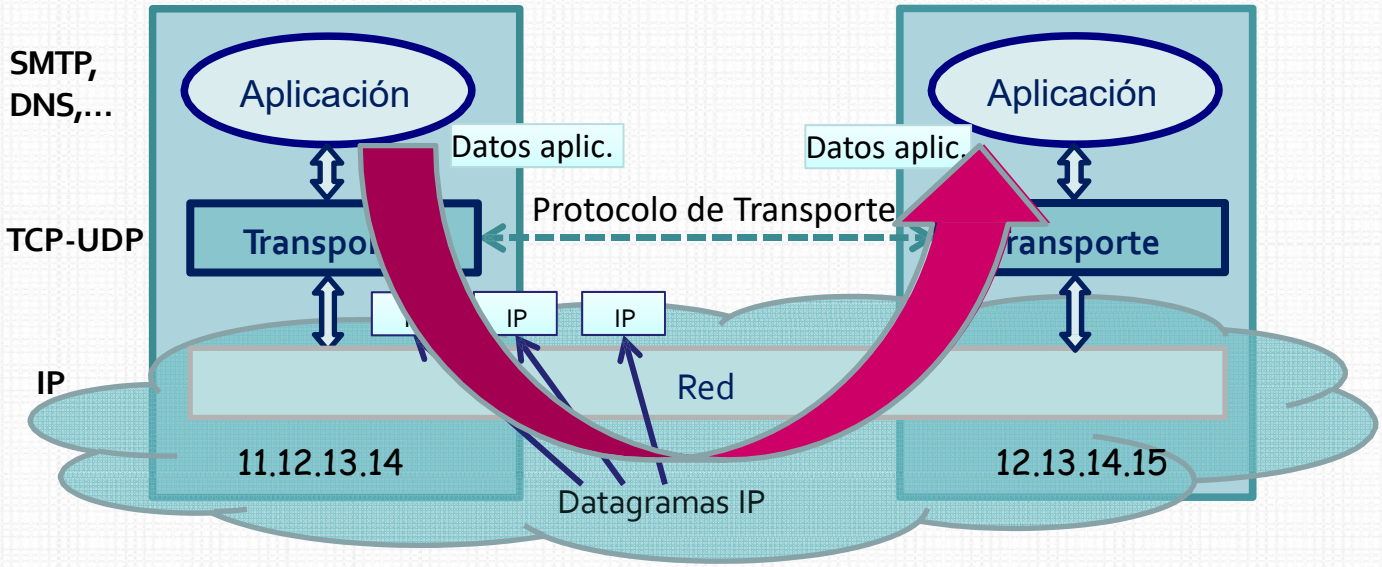
1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP

Sesión 1

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. El control de la congestión TCP
    6. Opciones TCP

1. Servicios del nivel de transporte

- Proporciona **comunicación lógica** entre dos **procesos** de aplicación que se ejecutan en hosts diferentes.
- Los protocolos de transporte se implementan en los sistemas terminales
  - Extremo emisor: fragmenta los mensajes de aplicación en segmentos y los pasa al nivel de red
  - Extremo receptor: ensambla los segmentos en mensajes y los pasa al nivel de aplicación.

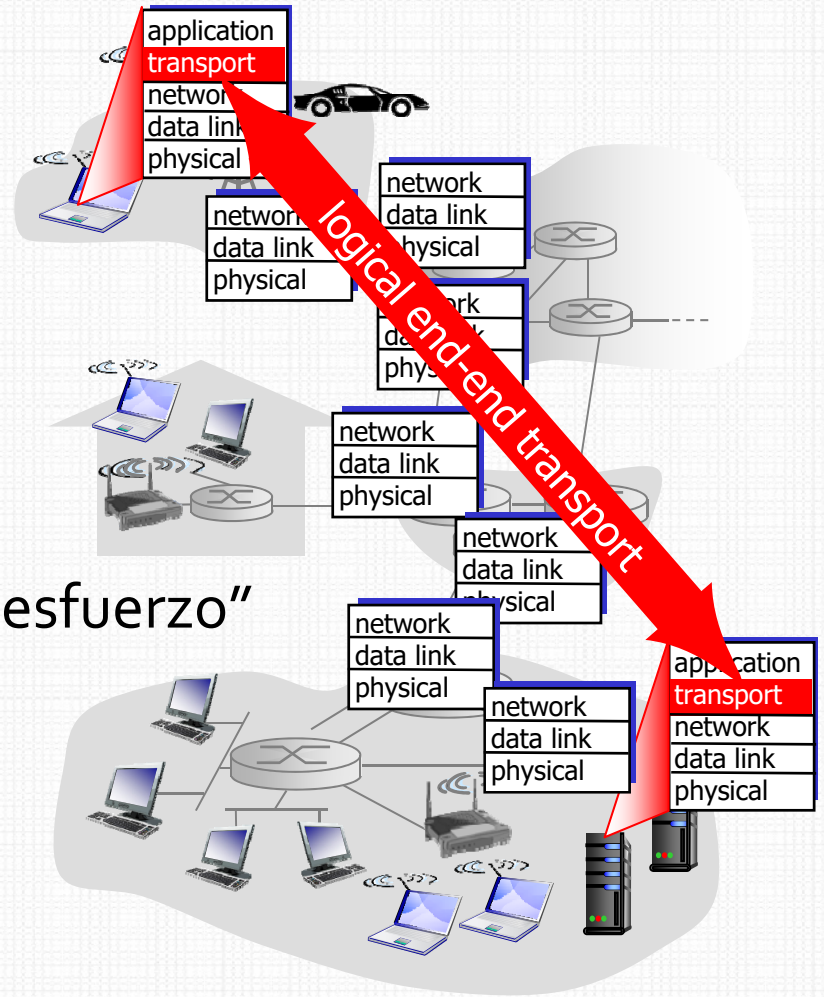


- En una arquitectura de red puede haber varios protocolos de transporte disponibles
  - En Internet: TCP y UDP

El nivel de transporte

- Nivel de Transporte vs. Nivel de Red
  - Nivel de Red
    - Protocolo IP *(Internet Protocol)*
    - Comunicación lógica entre **hosts**
    - Servicio de entrega de "mejor esfuerzo" *(best effort)*
  - Nivel de Transporte:
    - Comunicación lógica entre **procesos** que se ejecutan en hosts diferentes
      - Facilita la implementación de múltiples servicios de aplicación en el mismo host.
    - Servicio de entrega según protocolo (TCP o UDP)

- Protocolos
  - TCP (Transmission Control Protocol)
    - Fiable, entrega en orden
    - Establecimiento de conexión
    - Control de flujo
    - Control de congestión
  - UDP (User Datagram Protocol)
    - No fiable, entrega desordenada
    - Sencillo, extensión del servicio IP "mejor esfuerzo"
  - Servicios **no disponibles**:
    - Garantía de retardos
    - Garantía de ancho de banda

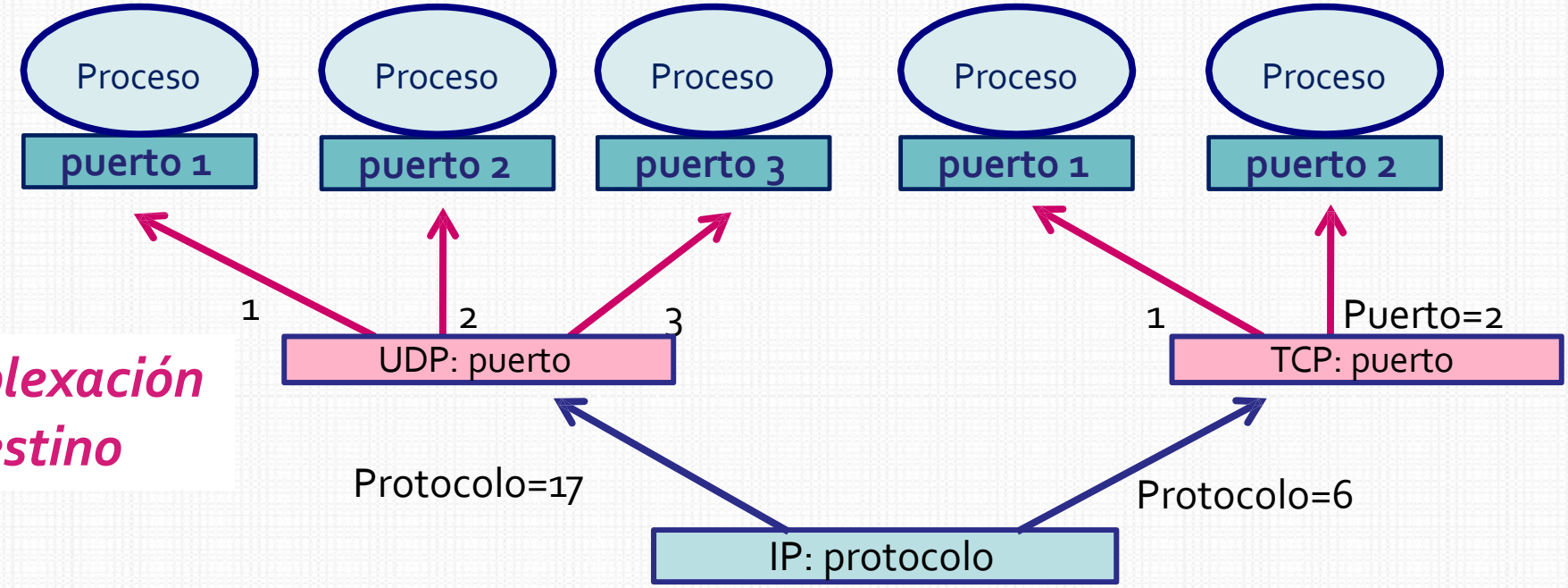


Direccionamiento de transporte

- Direccionamiento de red (IP):
  - Comunicación lógica entre hosts
  - Dirección IP
- Direccionamiento de transporte:
  - Comunicación lógica entre procesos
  - Amplía el servicio de entrega de IP entre dos hosts a un servicio de entrega entre dos procesos.
    - Multiplexación y demultiplexación del nivel de transporte
  - Se apoya en los servicios del nivel de red, puede mejorarlos
  - Identificación de los procesos
    - El identificador debe incluir la "IP" y el "puerto"

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



Demultiplexación en destino

Demultiplexación

- El host recibe un datagrama IP
  - Cada datagrama tiene una dirección IP fuente y una dirección IP destino
  - Cada datagrama contiene un segmento del nivel de transporte
  - Cada segmento contiene un puerto fuente y un puerto destino
- El nivel de transporte usa las direcciones IP y los números de puerto para dirigir el segmento al socket apropiado

Formato segmento TCP/UDP

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);
```

La capa de transporte asigna automáticamente un nº de puerto al socket

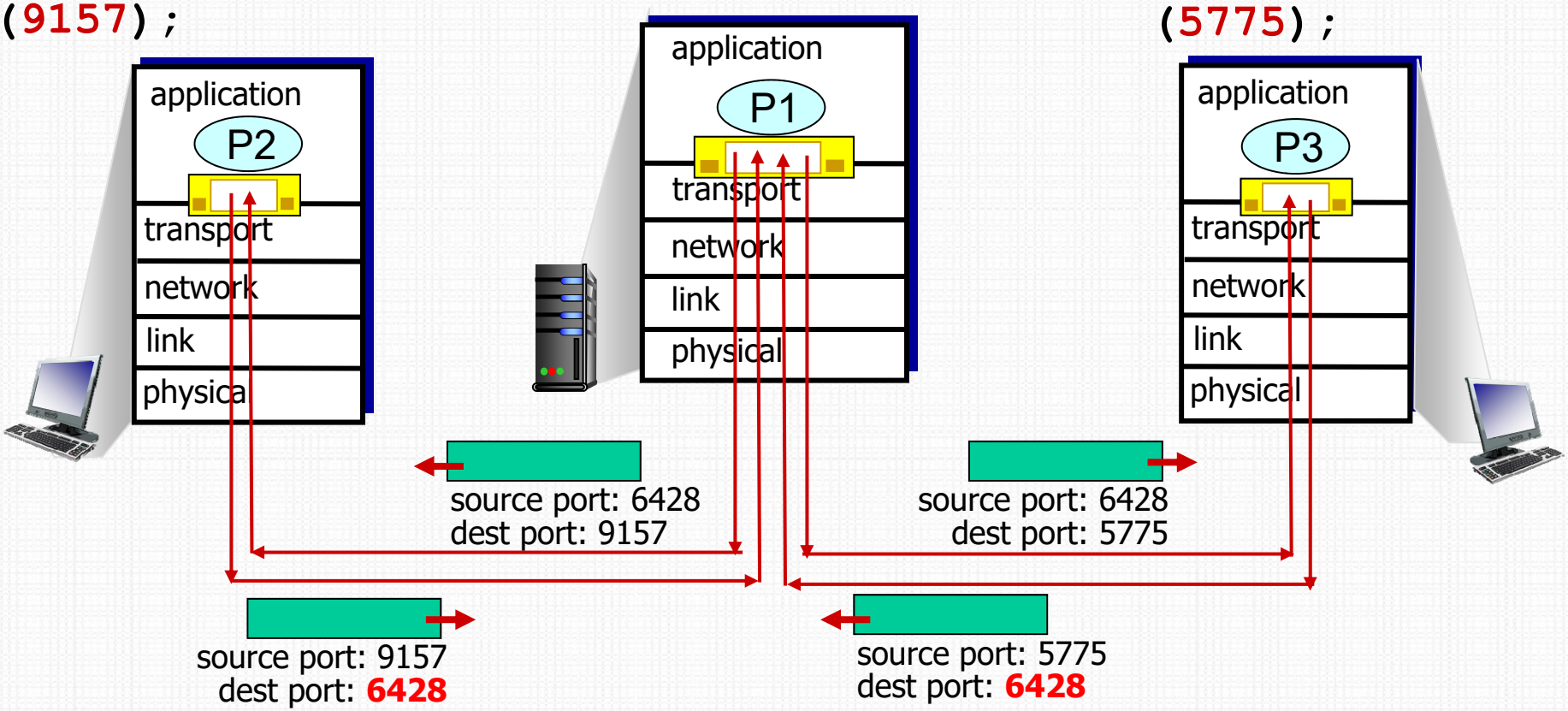
- Un *socket* UDP se identifica por dos valores:
  - (**Dirección IP destino, número de puerto destino**)
- Dos segmentos UDP con diferentes direcciones IP y/o nº de puerto de origen, pero la misma dir. IP destino y el mismo nº de puerto destino, se enviarán al mismo proceso de destino a través del mismo socket de destino.

Demultiplexación en UDP

```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

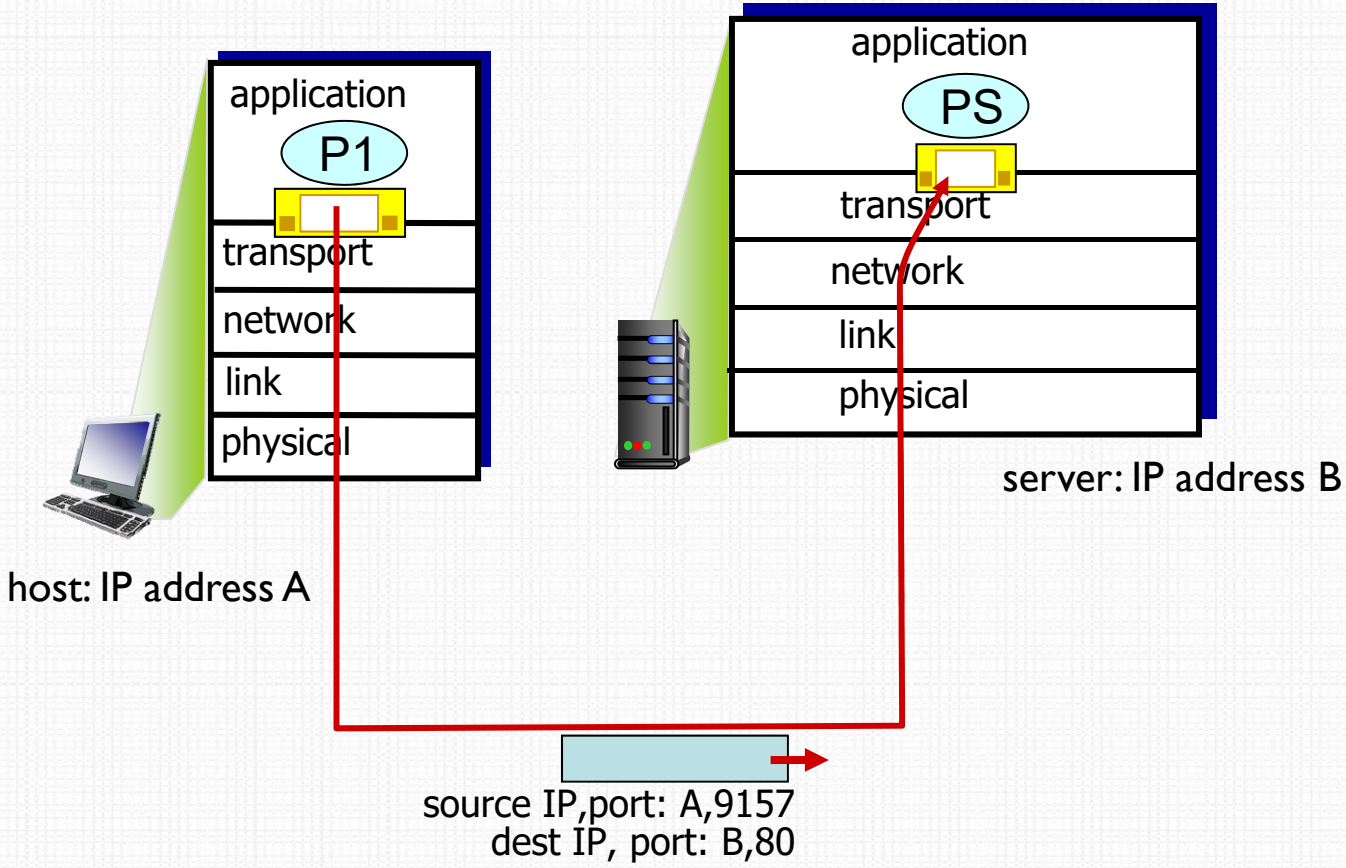
```
DatagramSocket
 mySocket3 = new
 DatagramSocket
 (5775);
```



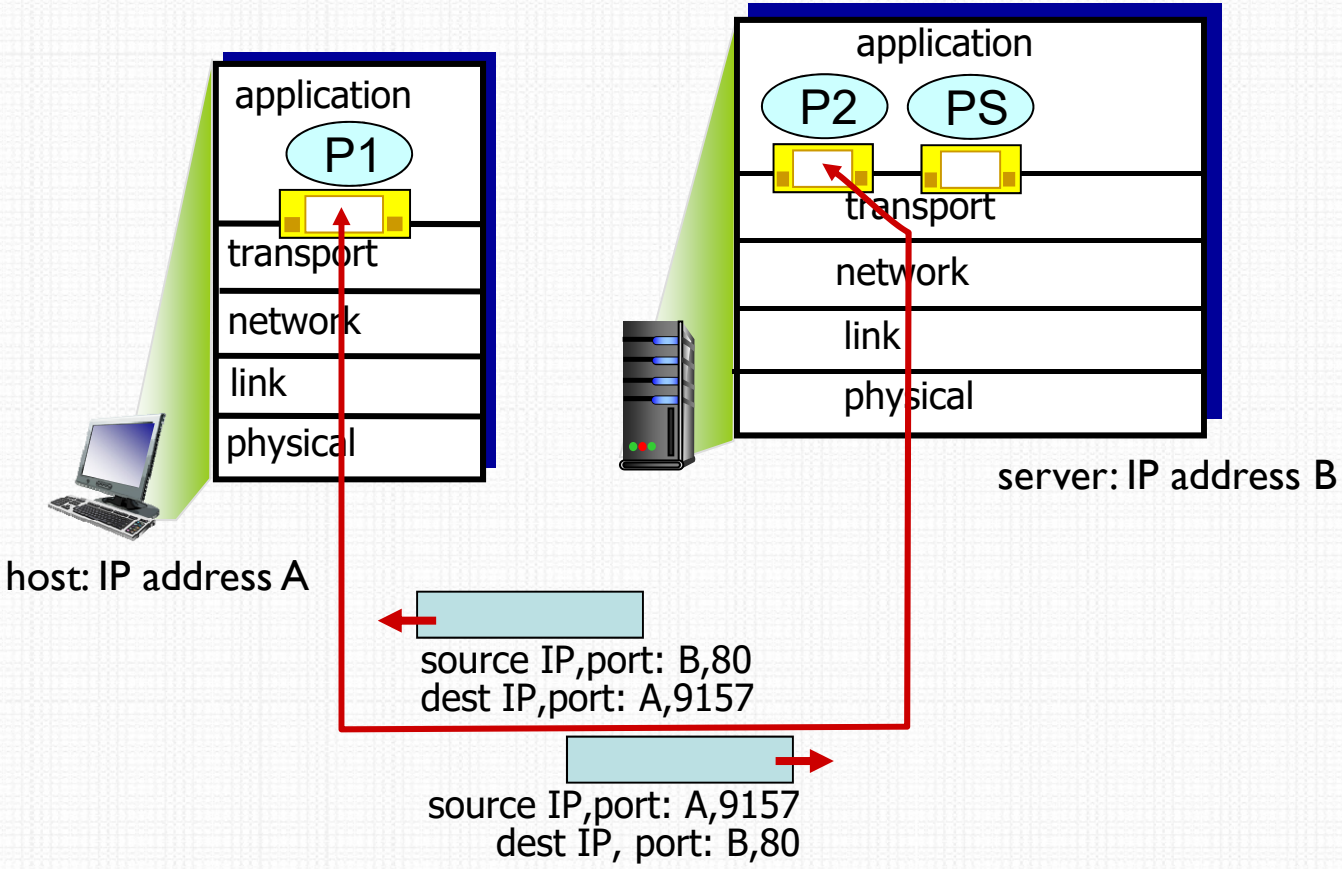
Puertos en TCP

- Creación de *sockets* TCP
  - `Socket cliente = new Socket("www.serv.com",80);`
  - `ServerSocket servidor = new ServerSocket(80);` — Socket de acogida
  - `Socket servidorconectado = servidor.accept();` — Socket de conexión
- Un *socket* TCP se identifica por una 4-tupla:
  - **IP fuente, puerto fuente, IP destino, puerto destino**
- El receptor usa los cuatro valores para dirigir el segmento al socket apropiado
- El host servidor puede soportar varios sockets TCP simultáneos:
  - Cada socket identificado por su 4-tupla

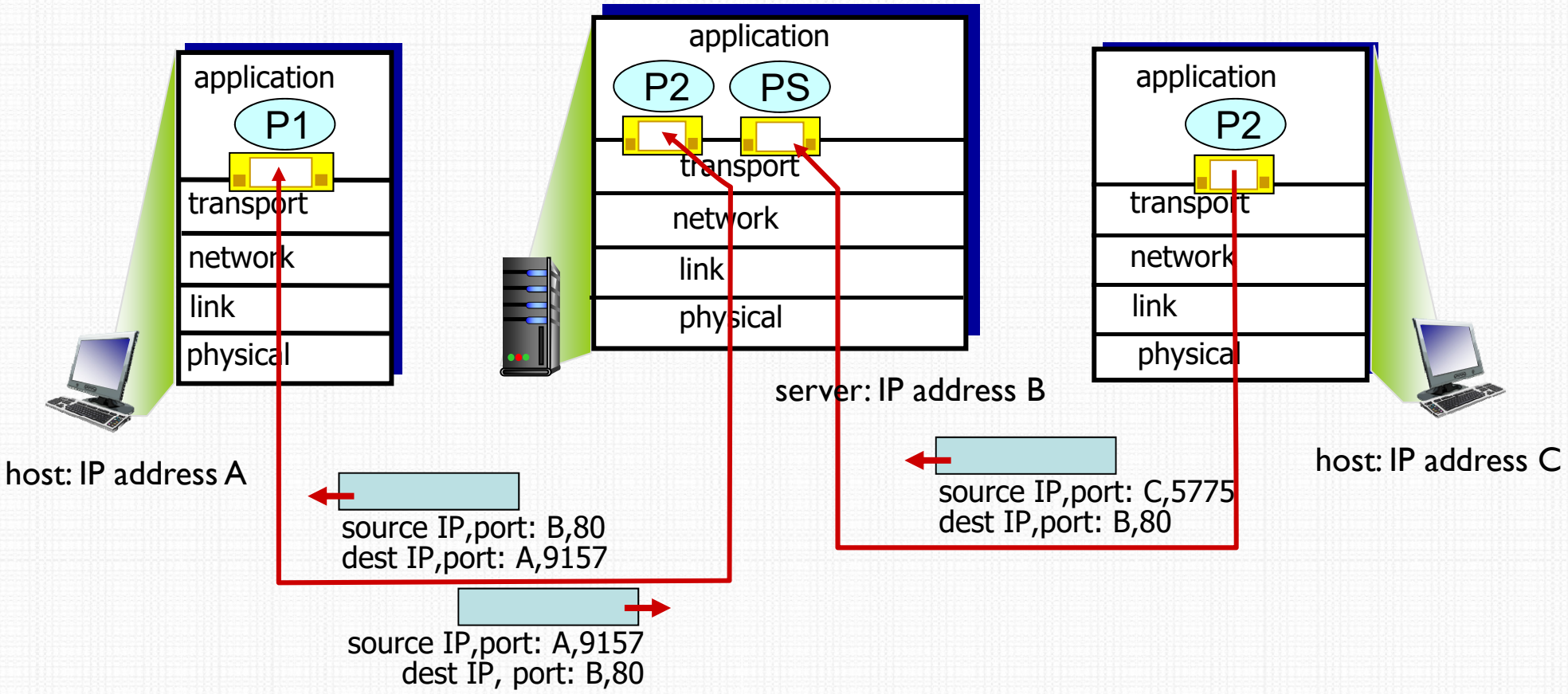
- Demultiplexación en TCP



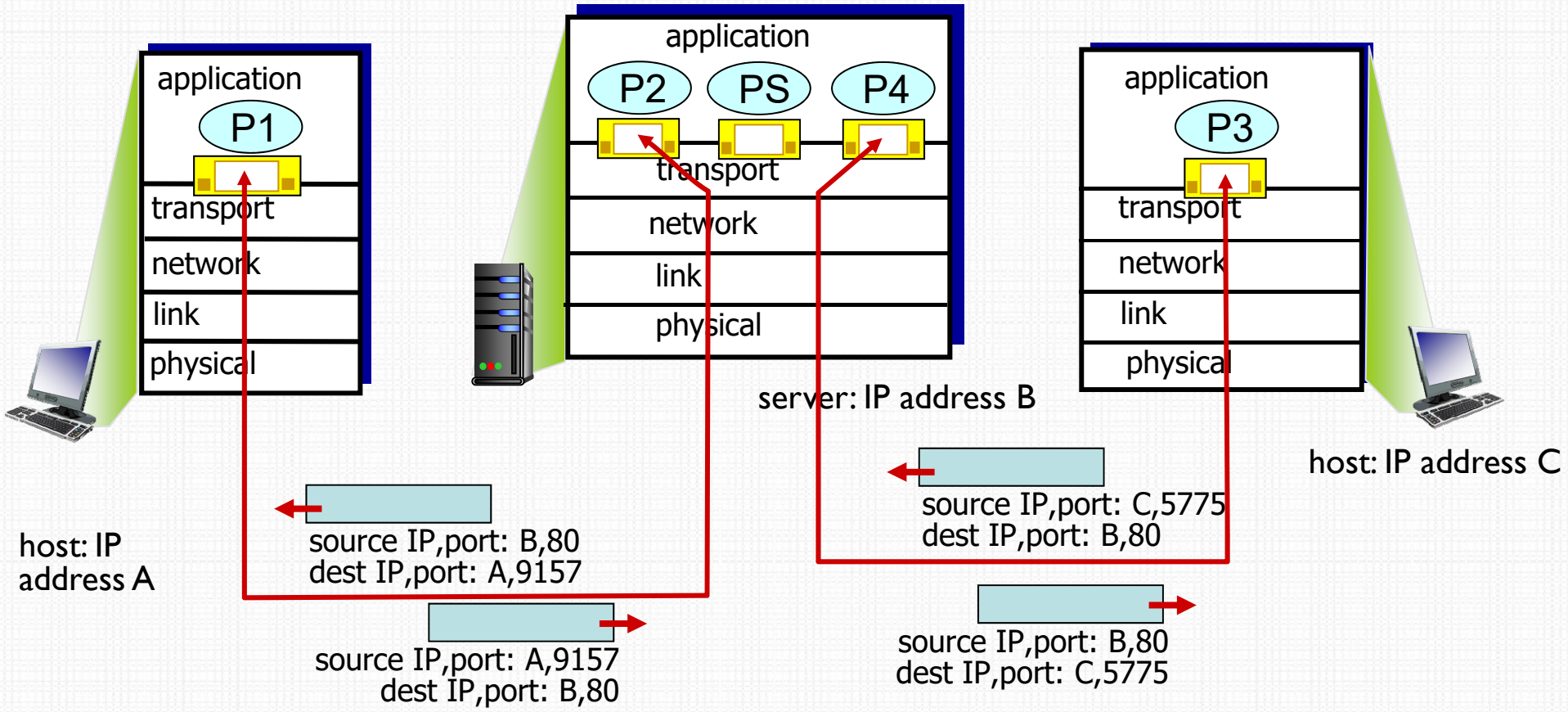
- Demultiplexación en TCP



- Demultiplexación en TCP

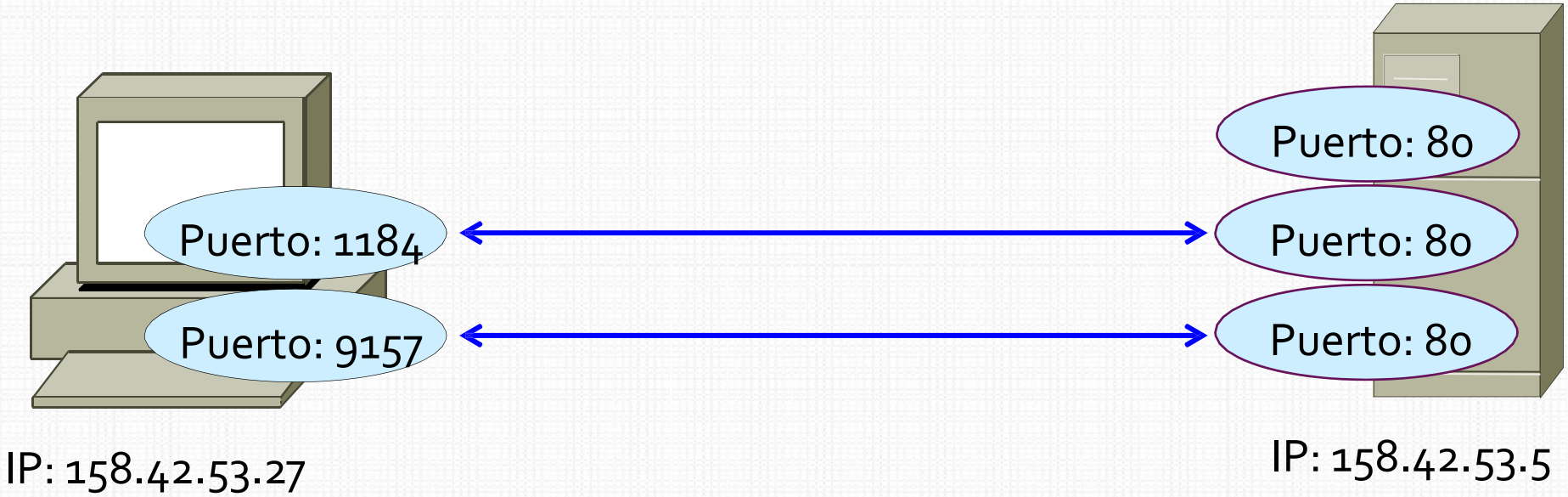


- Demultiplexación en TCP



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

1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP

Sesión 1

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. El control de la congestión TCP
   6. Opciones TCP

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
- Bloques de hasta 64 KBytes
- Permite difusiones
- Unidad de transferencia de información: datagrama
- Aplicaciones UDP:
  - Flujos multimedia (tolerantes a pérdidas, sensibles a la tasa de transferencia)
  - DNS, RIP,…
- Transferencia fiable sobre UDP:
  - Debe implementarla la propia aplicación

Transporte sin conexión: UDP

- **¿Por qué existe UDP?**
  - No hay establecimiento de conexión (que puede añadir un retardo)
  - Sencillo: no hay estado de la conexión ni en el emisor ni en el receptor
  - Cabecera del segmento pequeña (8 bytes en UDP/20 bytes en TCP)
  - No hay control de congestión: UDP sale del emisor tan rápido como se desee.

Encapsulado de segmentos UDP

Hasta 64 kB

APLICACIÓN: Mensaje

Mensaje aplicación

UDP:

Añade cabecera

Pasa a IP

SEGMENTO UDP o DATAGRAMA UDP

Cabecera UDP | Mensaje aplicación

IP:

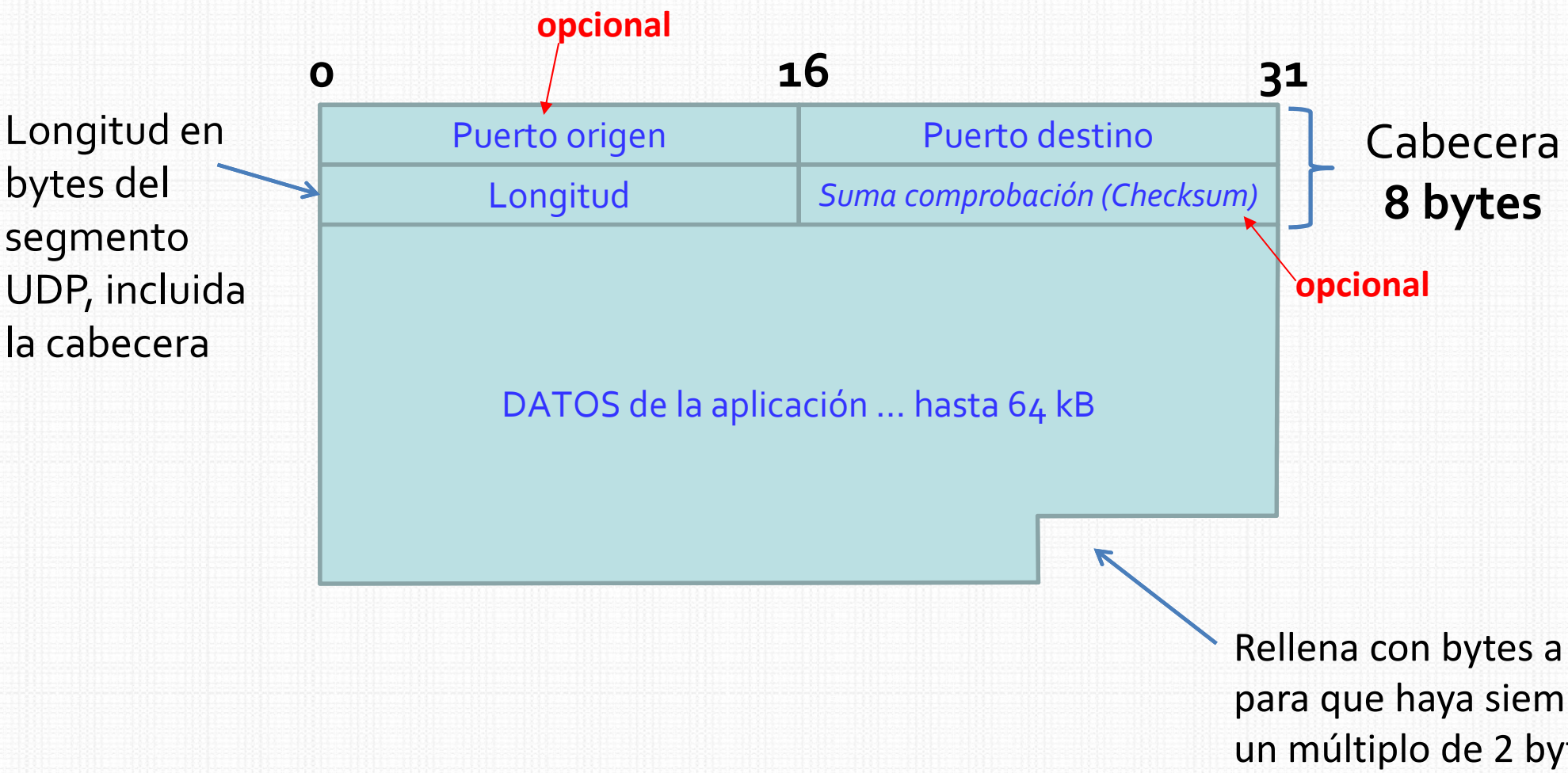
Añade cabecera

Envía a la red

DATAGRAMA **IP**

Cabecera IP | Cabecera UDP | Mensaje aplicación

Formato de un segmento UDP



Checksum UDP

Objetivo: detectar errores en los segmentos transmitidos

- **Emisor:**
  - Considera el contenido del segmento (cabecera y datos) como una secuencia de 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** (Ca1) del resultado y lo emplea como *checksum*

- **Receptor:**
  - Realiza nuevamente la suma (incluido el Checksum)
  - El resultado debe ser 0 en Ca1 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 (aritmética en Ca1)
- Ejemplo: suma de 2 enteros de 16 bits

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
acarreo	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
suma		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Ca1

Índice

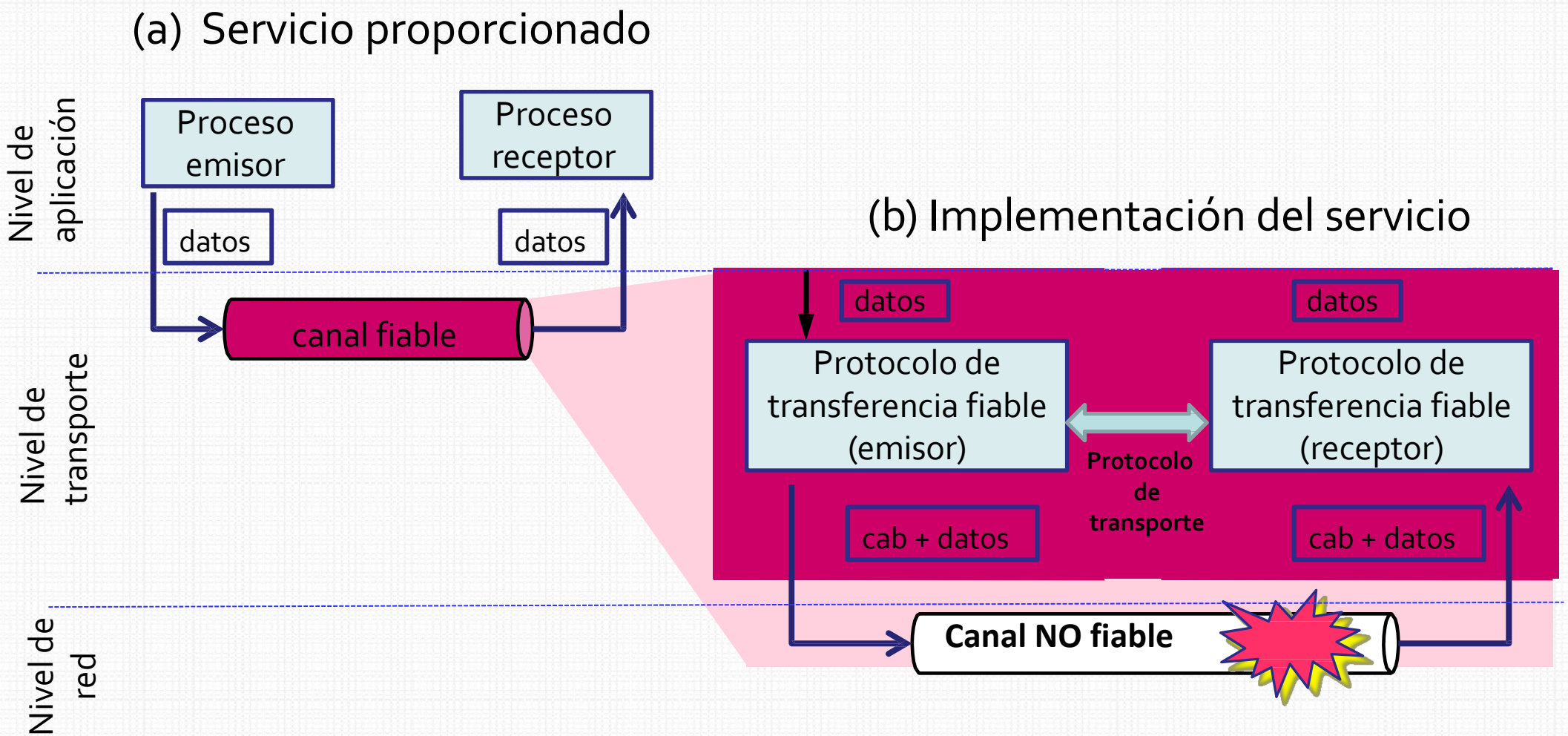
1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP
3. Fundamentos de la transferencia fiable de datos
4. Transporte orientado a la conexión: TCP
   1. Concepto
   2. Formato de un segmento
   3. Control de flujo y de error
   4. Gestión de una conexión TCP
   5. El control de la congestión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

Sesión 2

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

Protocolos para transferencia fiable

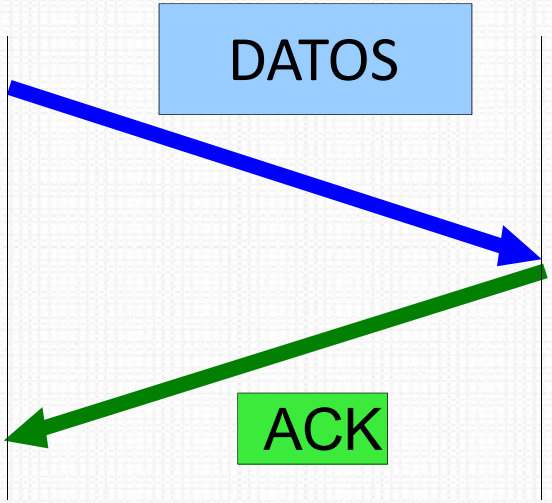
- Cuestiones básicas a resolver:
  - Emisor
    - ¿Se ha recibido correctamente el segmento?
  - Receptor
    - ¿El segmento recibido es correcto?
      - Evaluación de un código redundante (*checksum* o similar)
    - ¿Qué hacer en caso de que no lo sea?
- Solución:

Protocolo ARQ (*Automatic Repeat reQuest*),

- Solicitud automática de repetición (retransmisión)

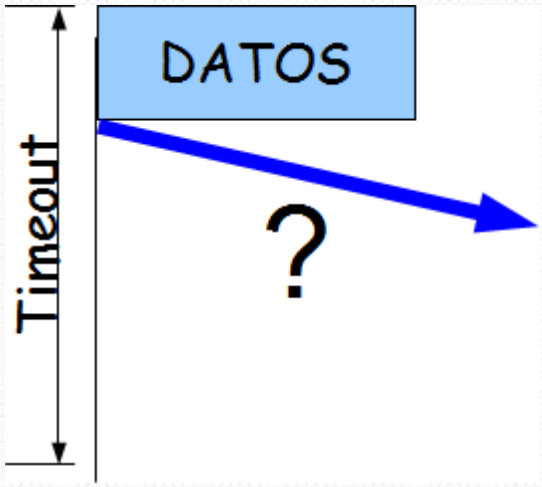
ARQ (*Automatic Repeat reQuest*)

- La recuperación de errores se basa en dos mecanismos:
  - Reconocimientos dentro de un plazo de tiempo (*Acknowledgements*, ACKs)
    - El problema puede presentarse también en el ACK (la transmisión de los ACKs tampoco es fiable)
  - Retransmisión en caso de fallo
    - Si no llega el reconocimiento en un intervalo de tiempo acotado (*RTO*, *retransmission timeout*) se reenvía el paquete



ACKs

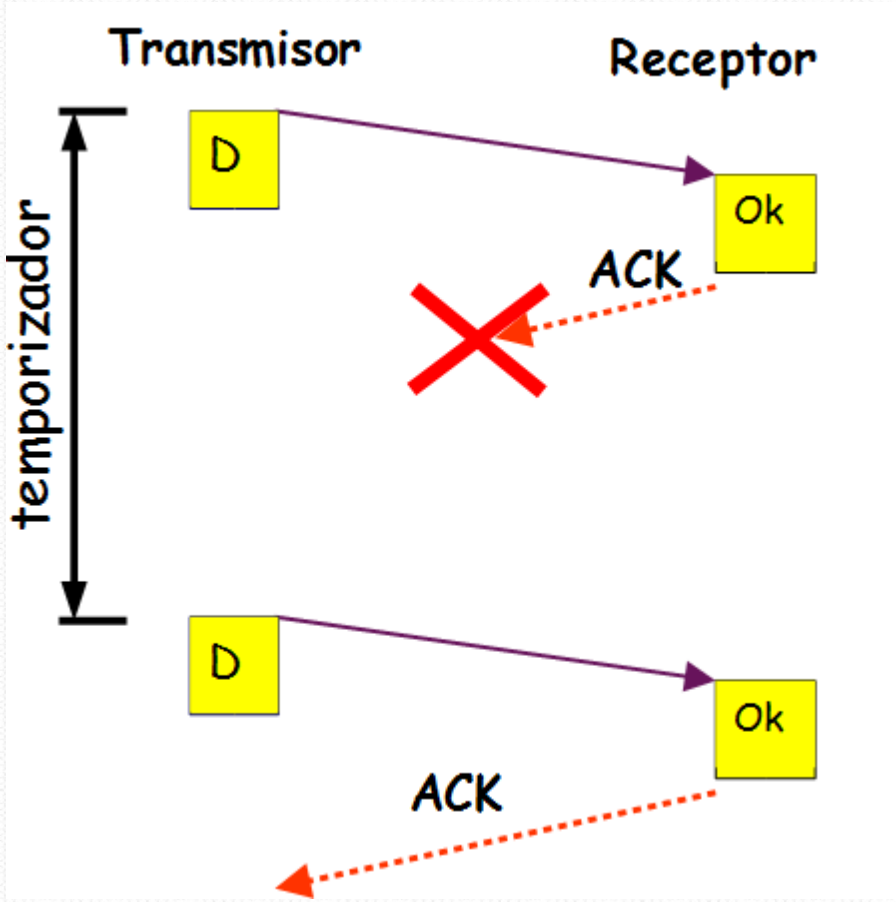
- ¿Por qué no llega el ACK?
  - Se pierde el paquete
  - Se pierde el ACK
  - El paquete llega con errores



- El receptor debe poder comprobar la presencia o ausencia de errores en la información recibida: detección de errores
- La detección se realiza mediante códigos detectores de errores (por ejemplo, Checksum)

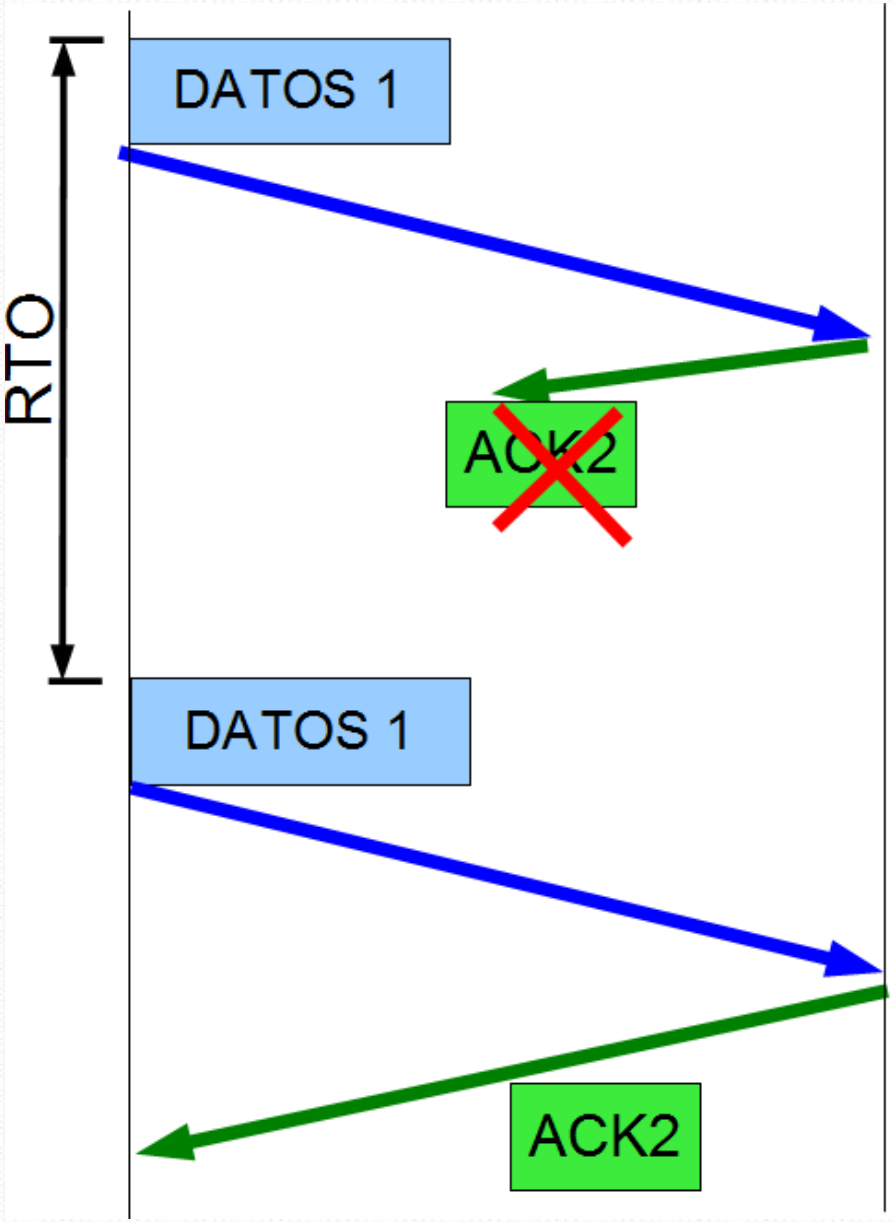
Detección de pérdida de segmento

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



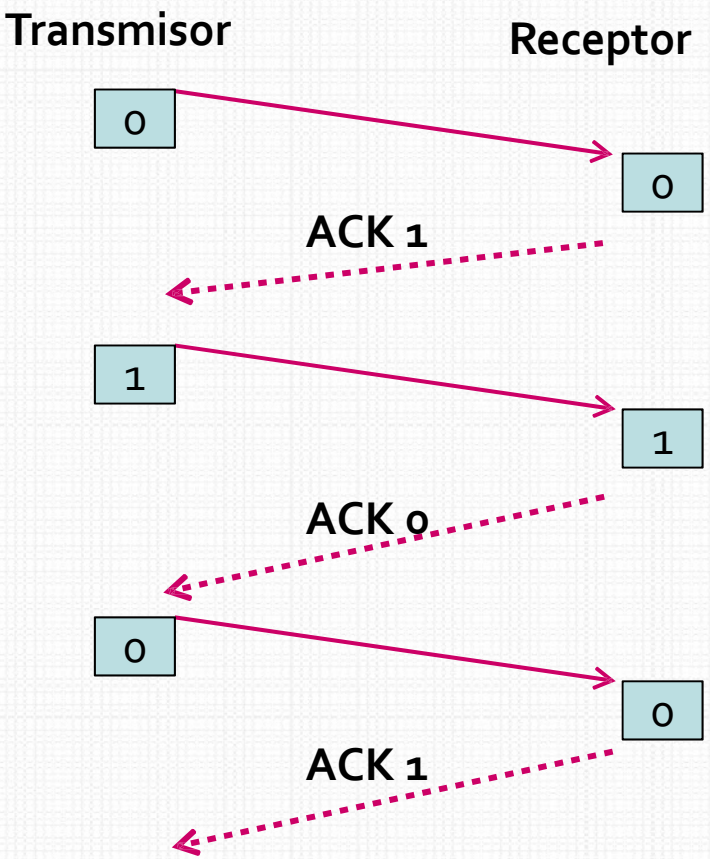
Numeración de los paquetes

- Se numeran los paquetes
  - También los ACK
- El número del ACK indica el **paquete que se espera**, no el recibido
- En general los ACK son **acumulativos**: ACK4 quiere decir que se han recibido correctamente y en orden todos los paquetes hasta el 3



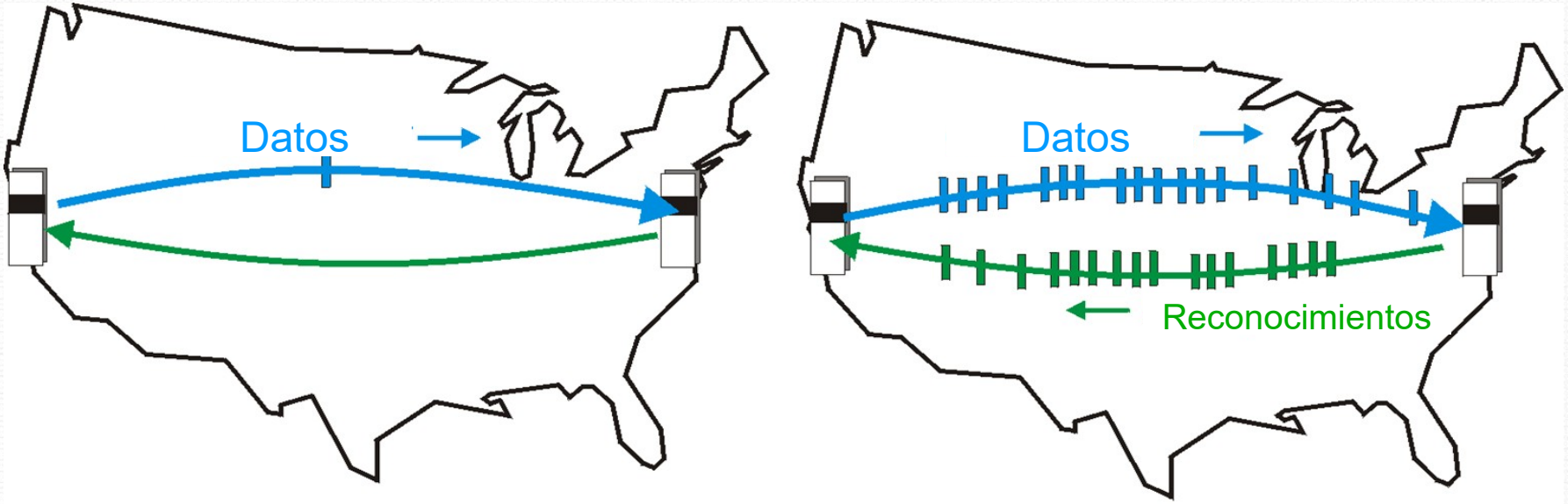
Stop and wait (parada y espera)

- Se identifican los segmentos (nº de secuencia de 1 bit)
- El transmisor sólo puede tener un segmento pendiente de reconocimiento
  - **¡¡Simple pero ineficiente!!**
- Se pierde mucho ancho de banda



Ventana deslizante

- Mejora
  - Varios segmentos por RTT



Parada y espera Ventana deslizante

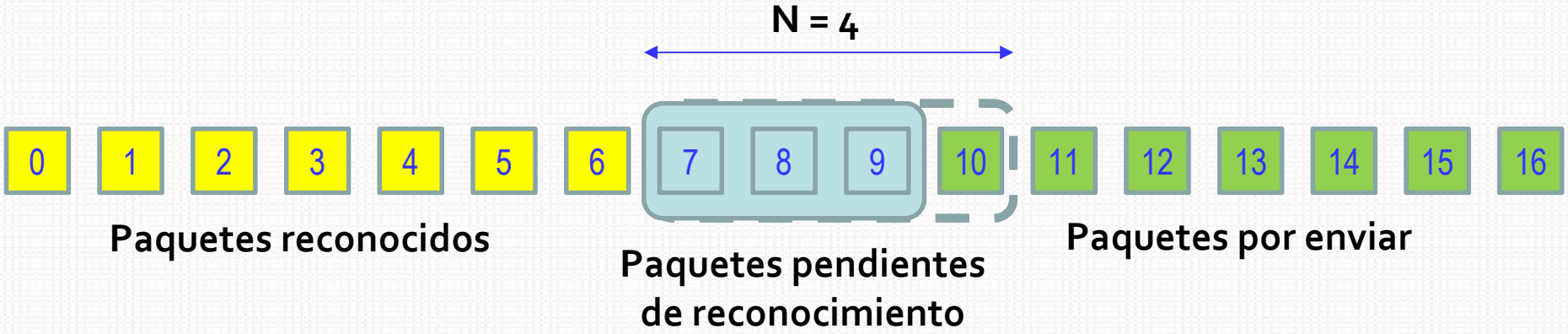
Retardo propagación ida y vuelta = 30 ms
V_trans = 1 Gbps
L= 8000 bits
T_trans = 0.008 ms
Despreciando T_ack , RTT = 30,008 ms

Ventana deslizante

- Consiste en enviar varios segmentos por RTT (antes del primer reconocimiento) (**pipelining o procesamiento en cadena**)
- Hay que incrementar el rango de los nº de secuencia
- El emisor debe almacenar los segmentos enviados pendientes de reconocimiento
  - Para ello se emplea la **ventana de transmisión**

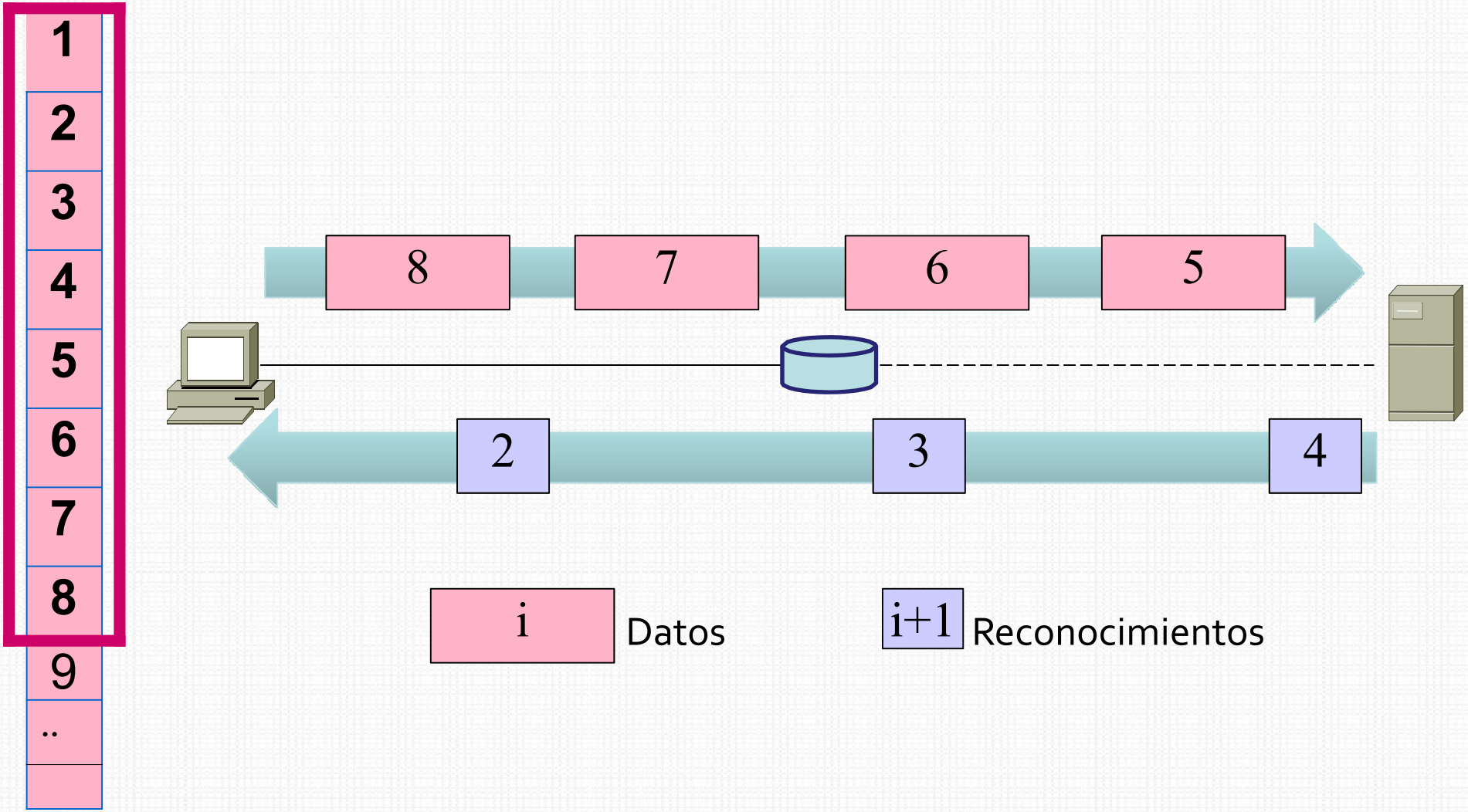
Ventana de transmisión

- Contiene los paquetes enviados y pendientes de reconocimiento
- De longitud variable, máximo N
- La ventana incrementa su límite inferior al recibir reconocimientos y su límite superior con nuevas transmisiones



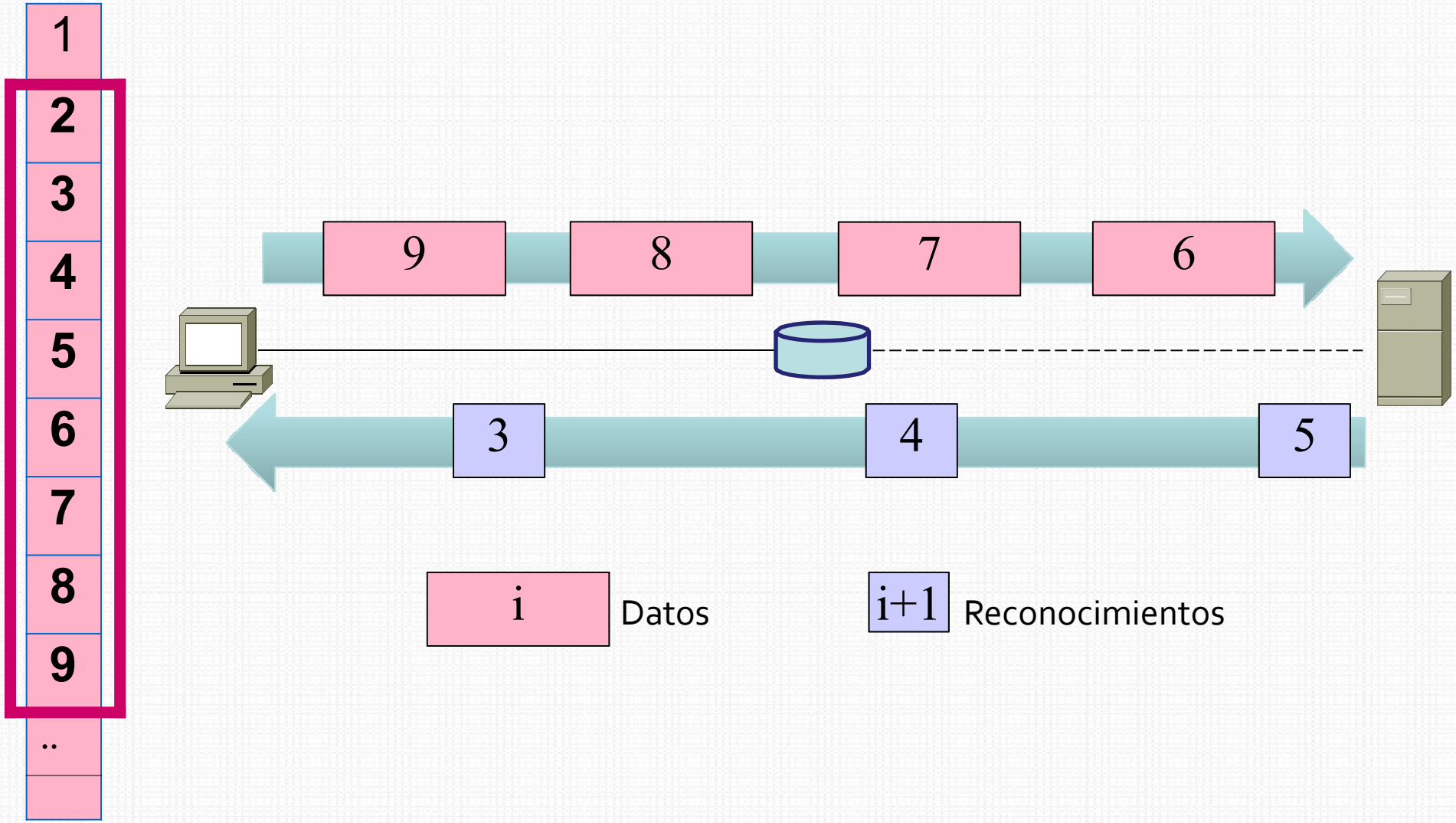
Funcionamiento ventana deslizante

Ventana máxima = 8



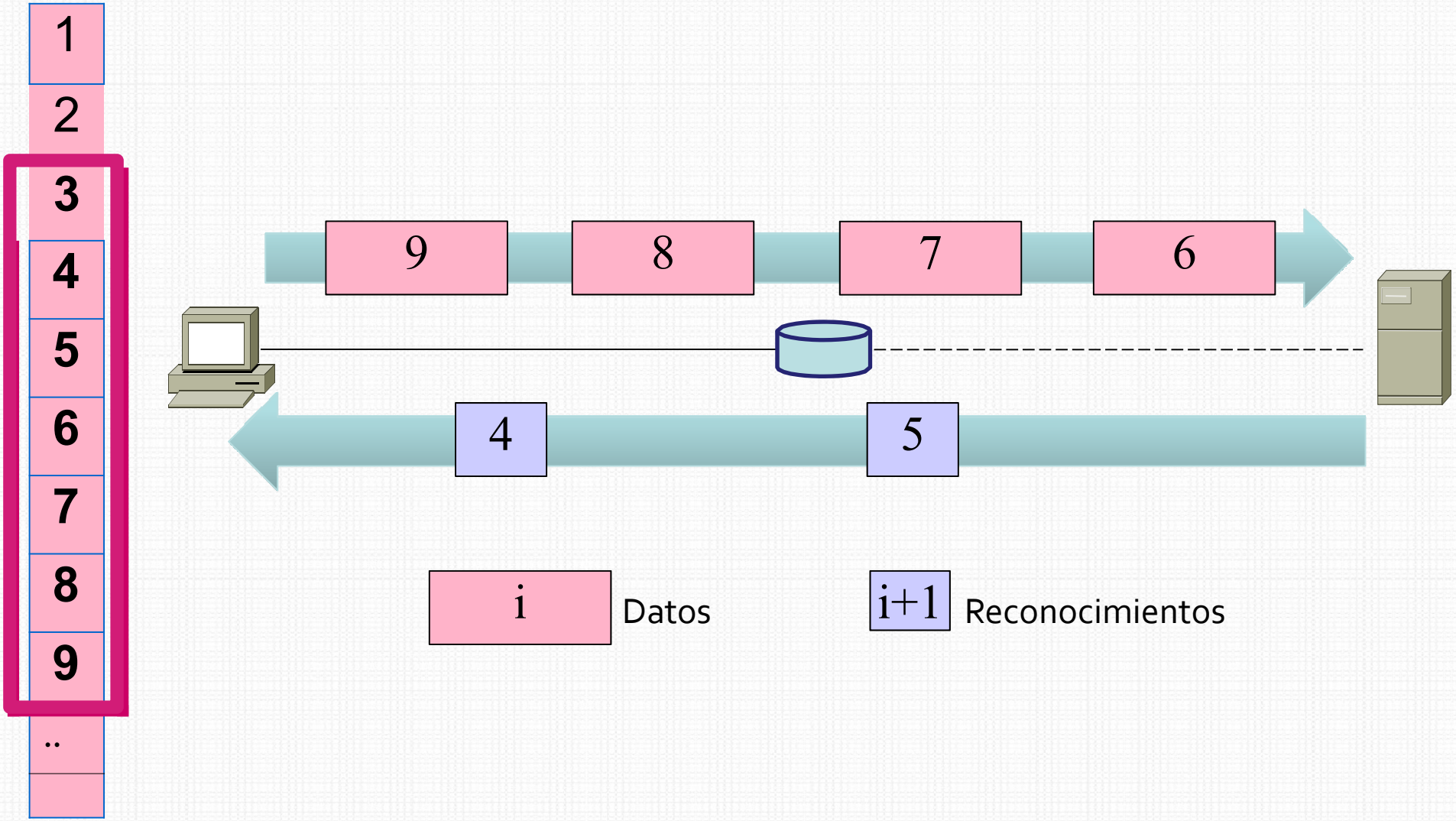
Funcionamiento ventana deslizante

Ventana máxima = 8



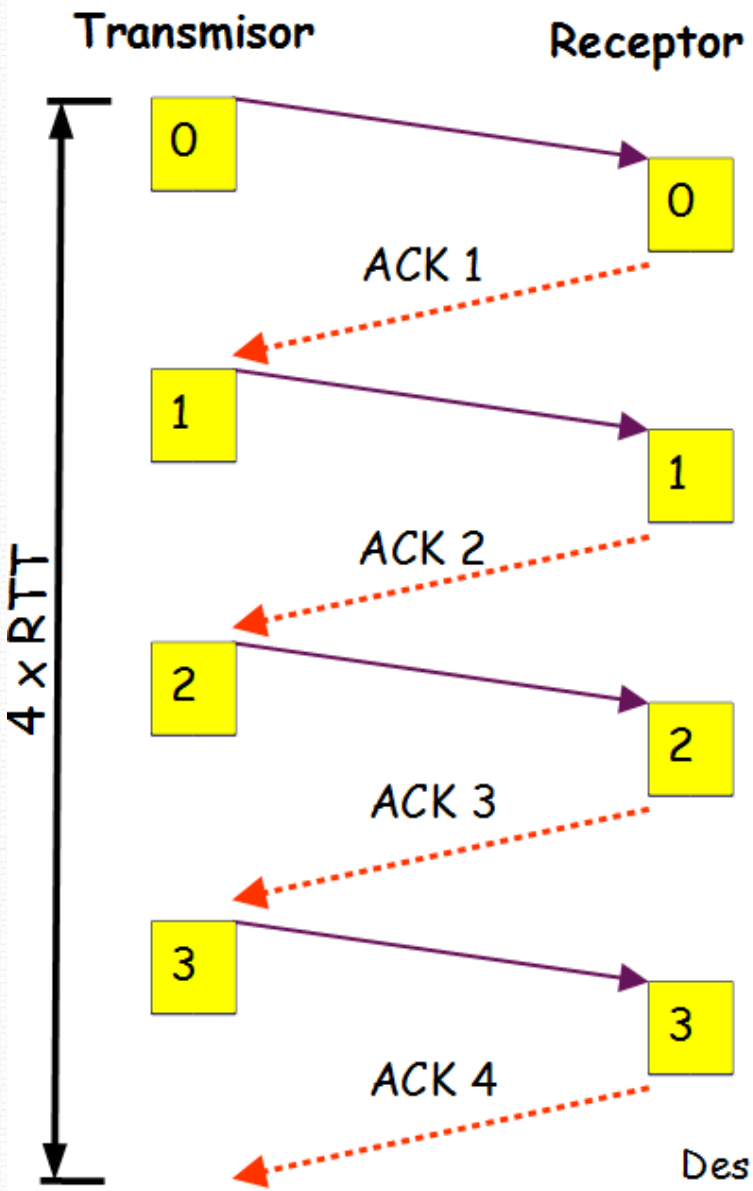
Funcionamiento ventana deslizante

Ventana máxima = 8

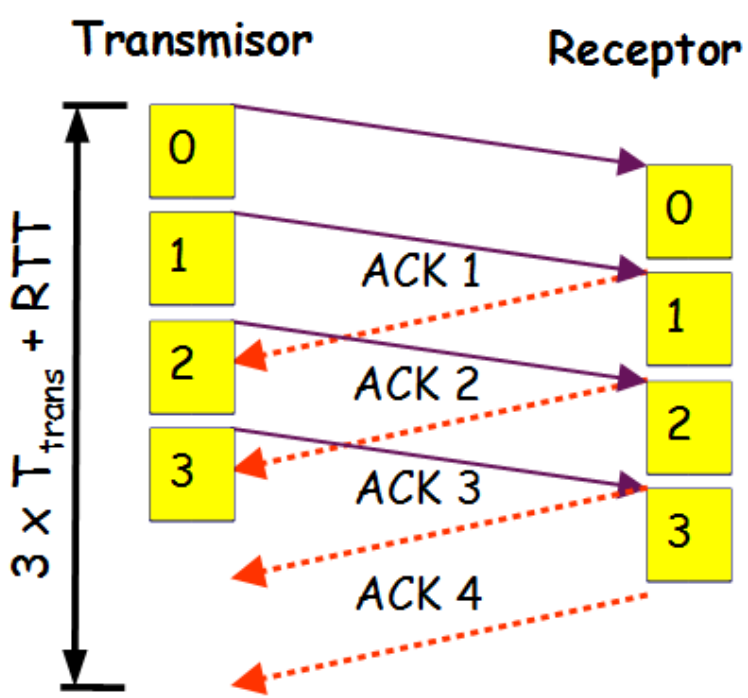


Comparación de prestaciones

Parada y espera



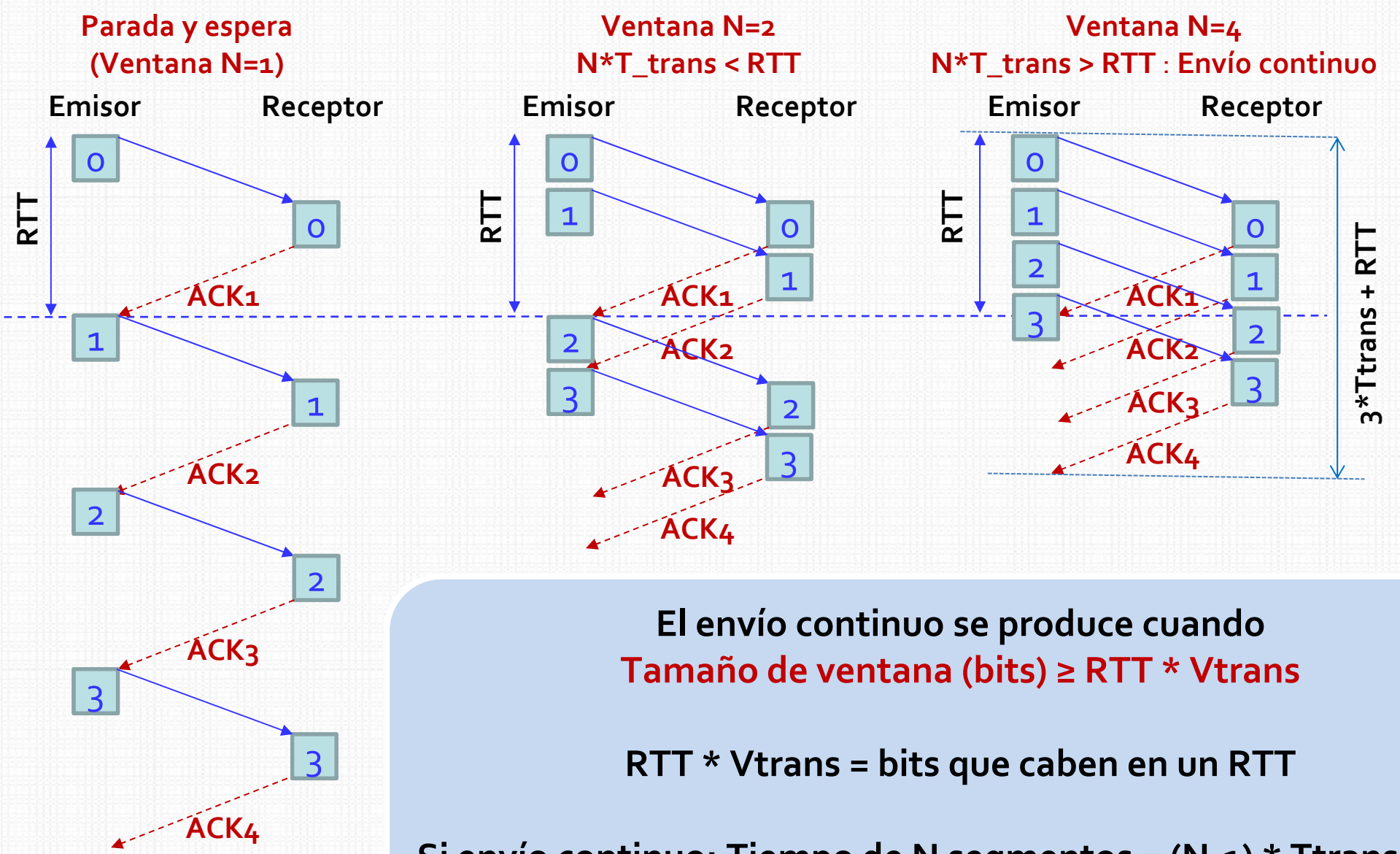
Ventana deslizante



Ventana N=4

Despreciamos tiempos de proceso en emisor y receptor

Dimensionamiento de ventana



El envío continuo se produce cuando

Tamaño de ventana (bits) ≥ RTT * Vtrans

RTT * Vtrans = bits que caben en un RTT

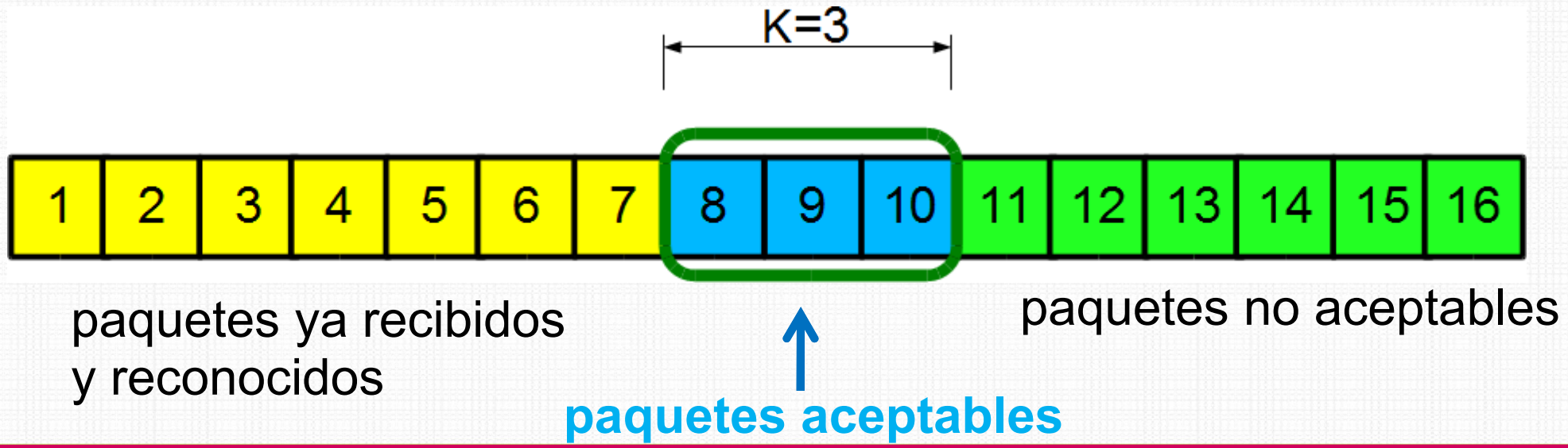
<u>Si envío continuo</u>: Tiempo de N segmentos = (N-1) * Ttrans + RTT

Recepción en ventana deslizante

- Los reconocimientos son acumulativos
- 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

Ventana de recepción

- Contiene los nº de secuencia de los segmentos que el receptor espera recibir
- Tiene una longitud fija de K
  - Vuelta atrás: K=1
  - Repetición selectiva: 1 < K <= N



Resumen: Tamaño de ventanas

Ventana de Transmisión

- **Tamaño variable** (límite máximo a N)
- Segmentos con temporizador de retransmisión asociado
- Dimensionamiento para envío continuo:
  - Nº bits ventana > RTT*V_{Trans}

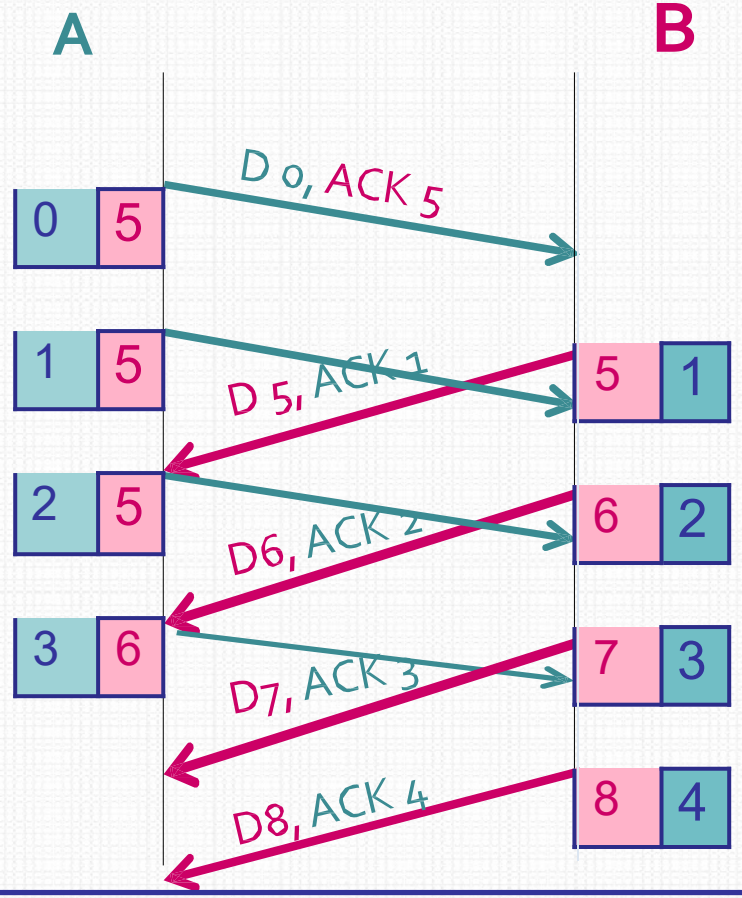
Ventana de Recepción

- **Tamaño fijo**
- **Vuelta atrás**
  - Recepción ordenada (tamaño K=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. El control de la congestión TCP
   6. Opciones TCP

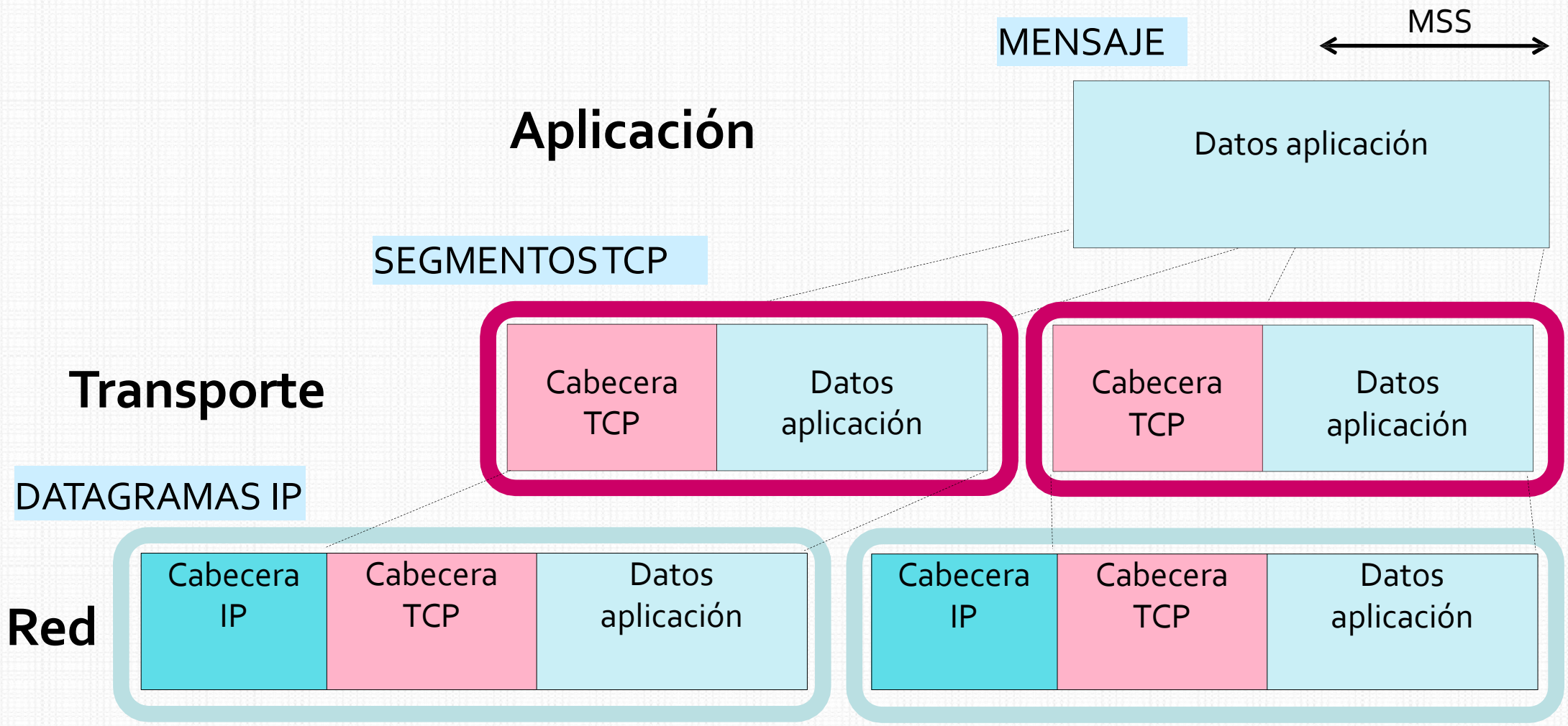
Sesión 3

TCP (*Transmission Control Protocol*)

RFC's 793, 1122, 1323, 2018, 2581

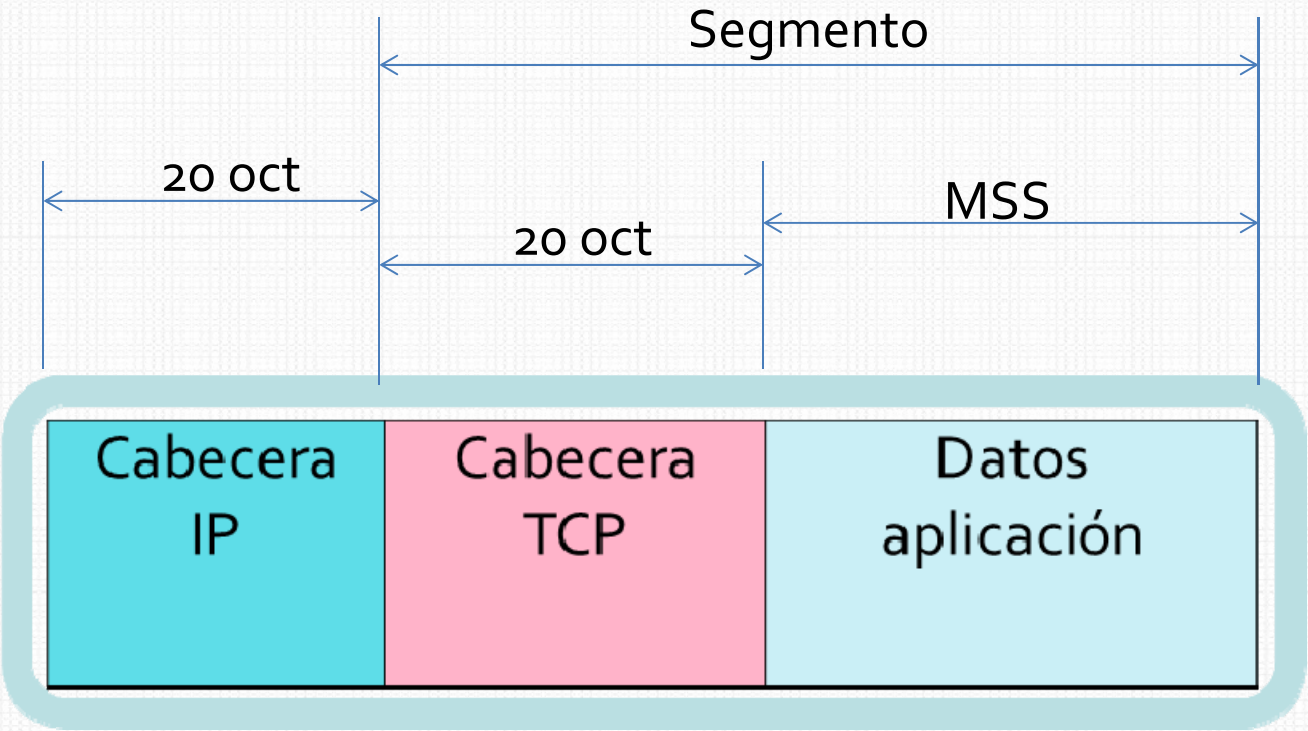
- Servicio orientado a la conexión:
  - Punto a punto
  - Control de flujo
  - Servicio fiable (retransmisiones)
- Flujo de bytes ordenado entre aplicaciones
  - No tiene en cuenta las fronteras entre mensajes
- Datos full dúplex
  - Flujo de datos bidireccional en la misma conexión
  - MSS: *maximum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

Encapsulado de TCP



MSS=Maximum Segment Size

Maximum Segment Size (MSS)



- Cada extremo anuncia su MSS al establecer la conexión
- Por defecto MSS = 536 octetos

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

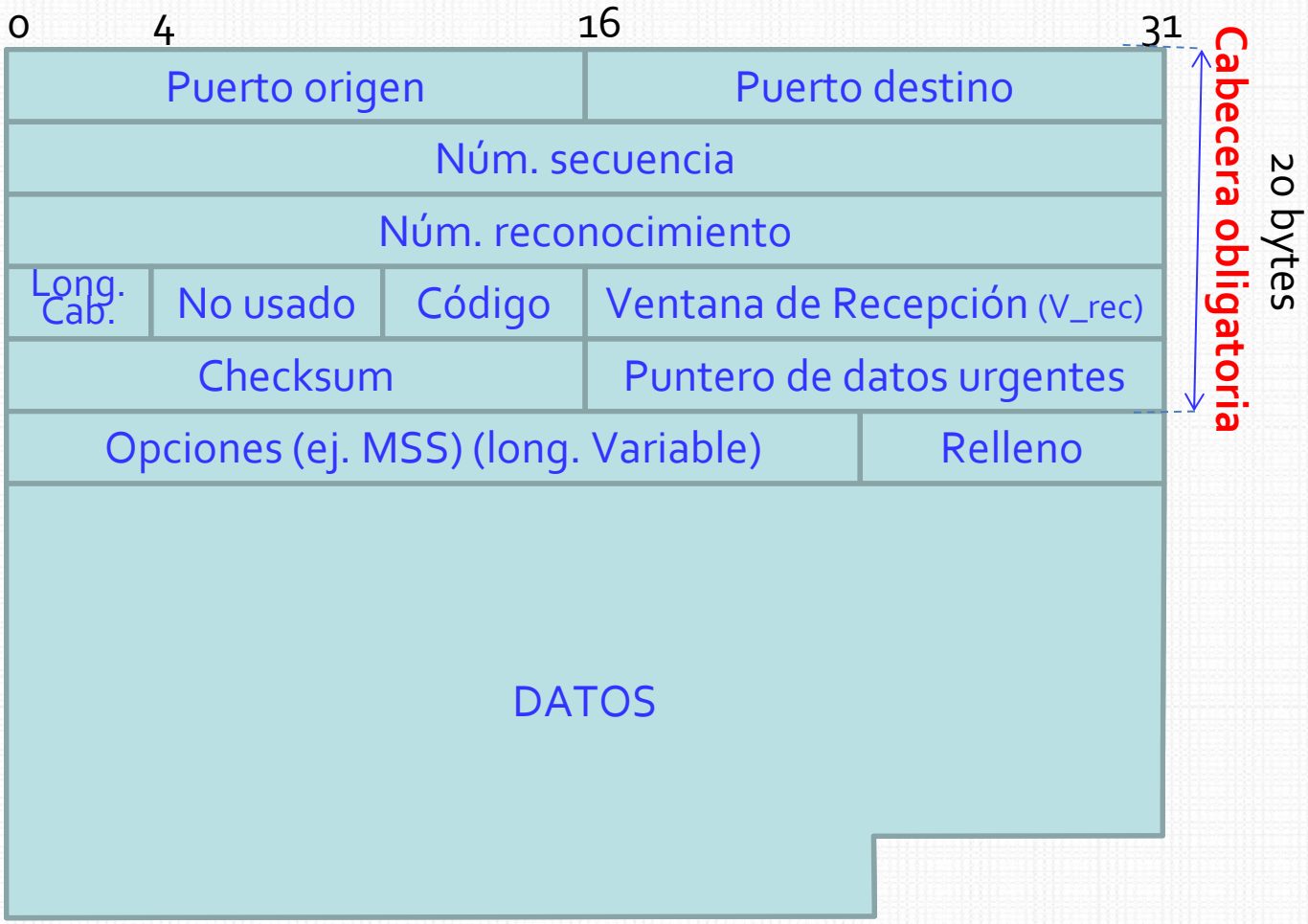
TCP

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

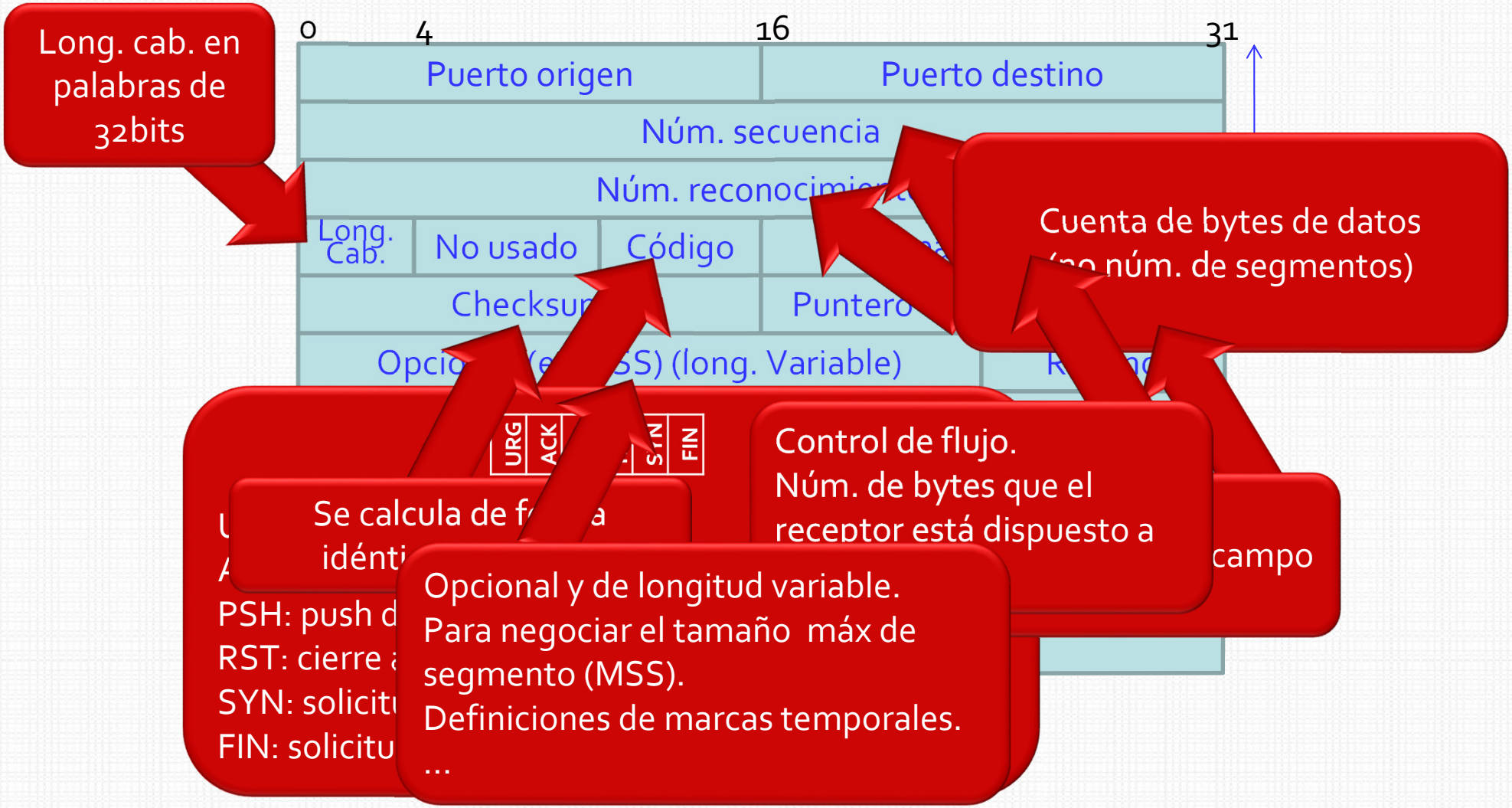
4.2. Formato de un segmento TCP

La cuenta de datos se lleva en bytes:

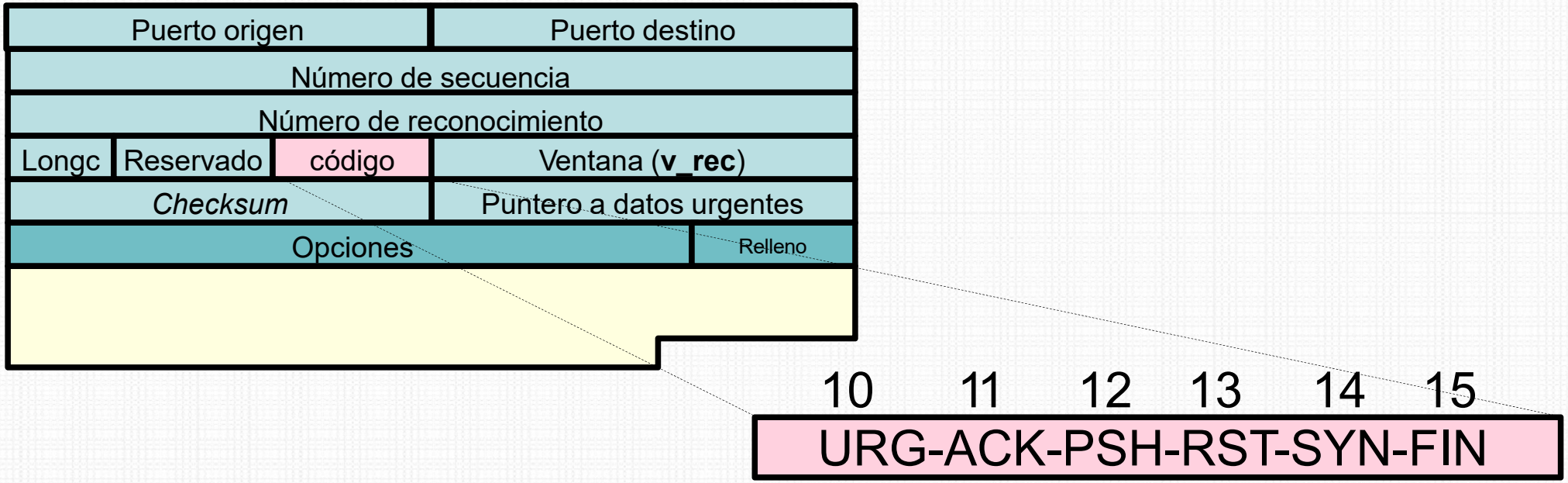
- Todos los campos que se refieran a los datos contarán en bytes



4.2. Formato de un segmento TCP



Campo de código



URG	Puntero a datos urgentes es válido
ACK	El campo de reconocimiento es válido
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

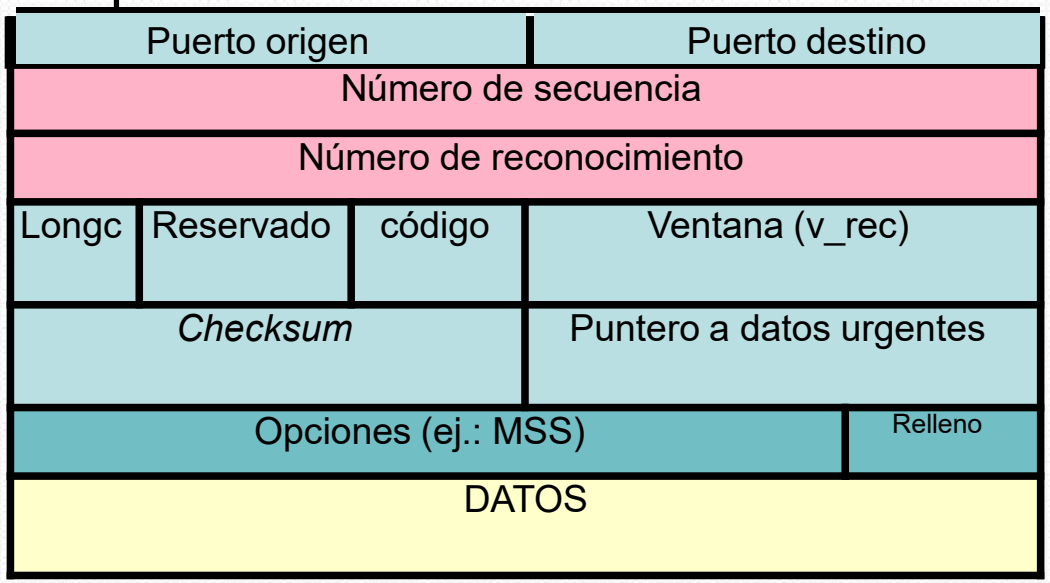
Poco empleado

Checksum en TCP

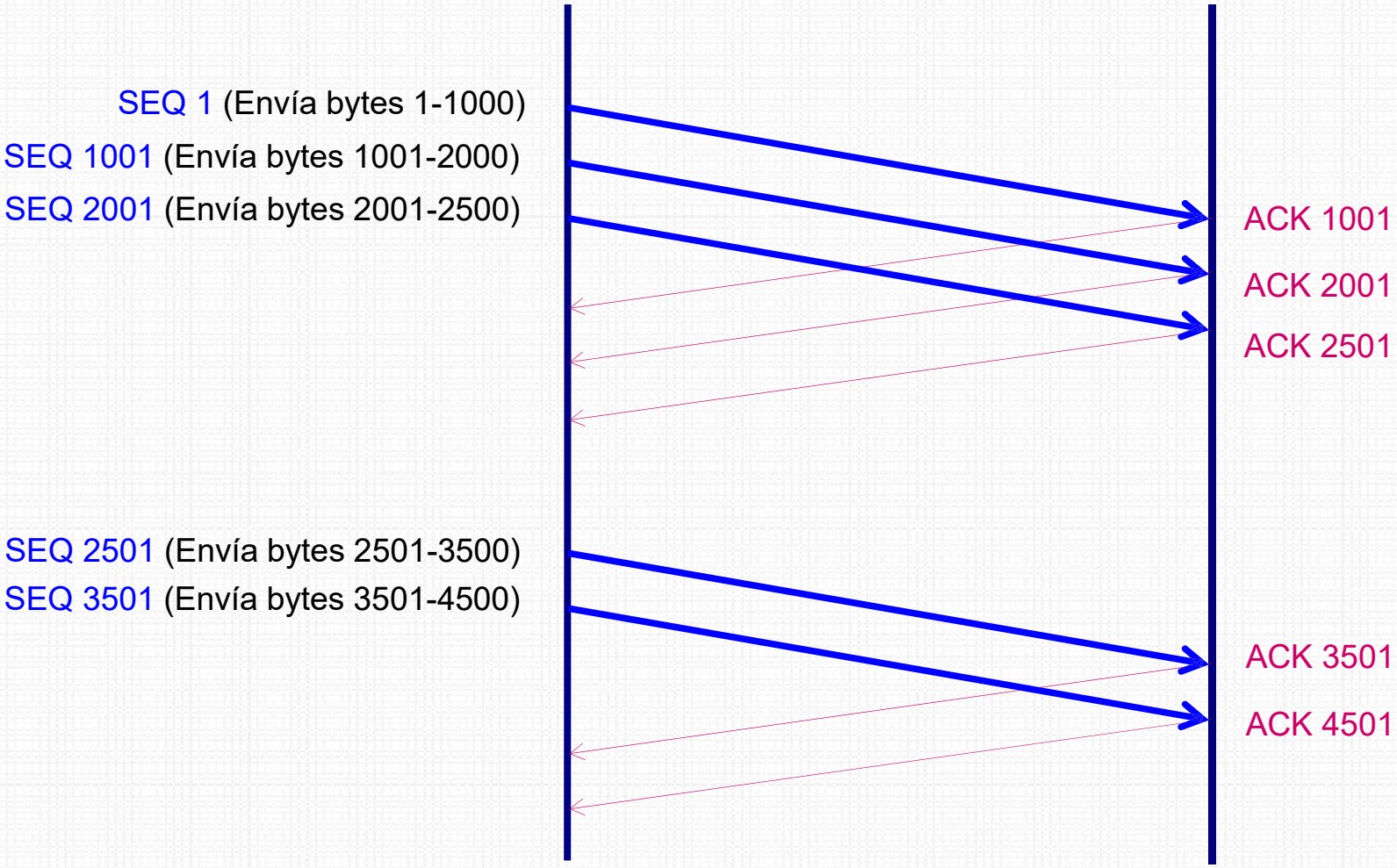
- Asegura que los segmentos no han sido modificados en tránsito
- Cálculo similar al *checksum* UDP
  - Se aplica a todo el segmento (cabecera+datos)
- Un segmento dañado se descarta
  - Tendrá que ser retransmitido

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



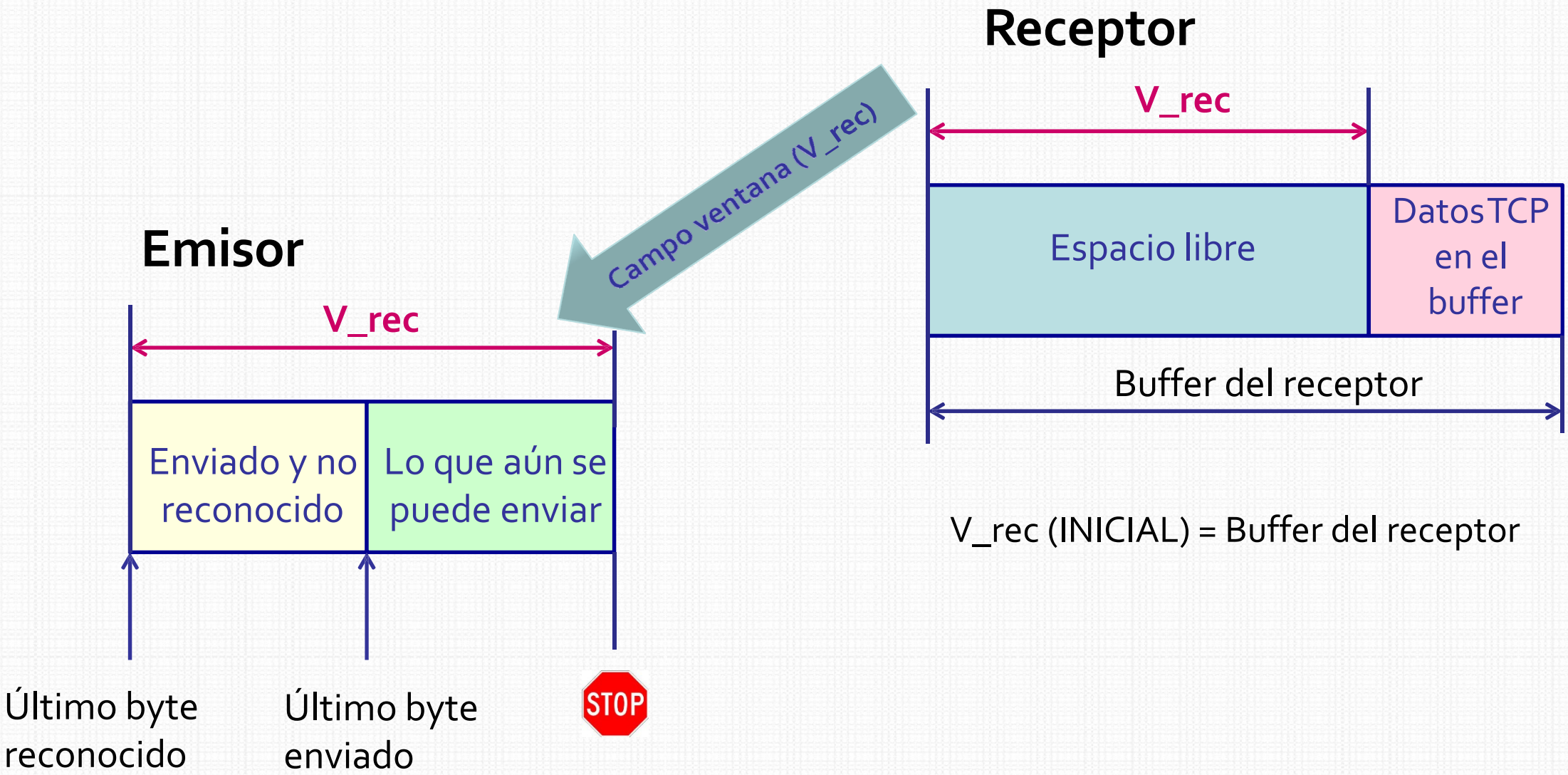
Como el tránsito de datos es bidireccional* los reconocimientos pueden viajar **superpuestos** (*piggybacking)*

**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

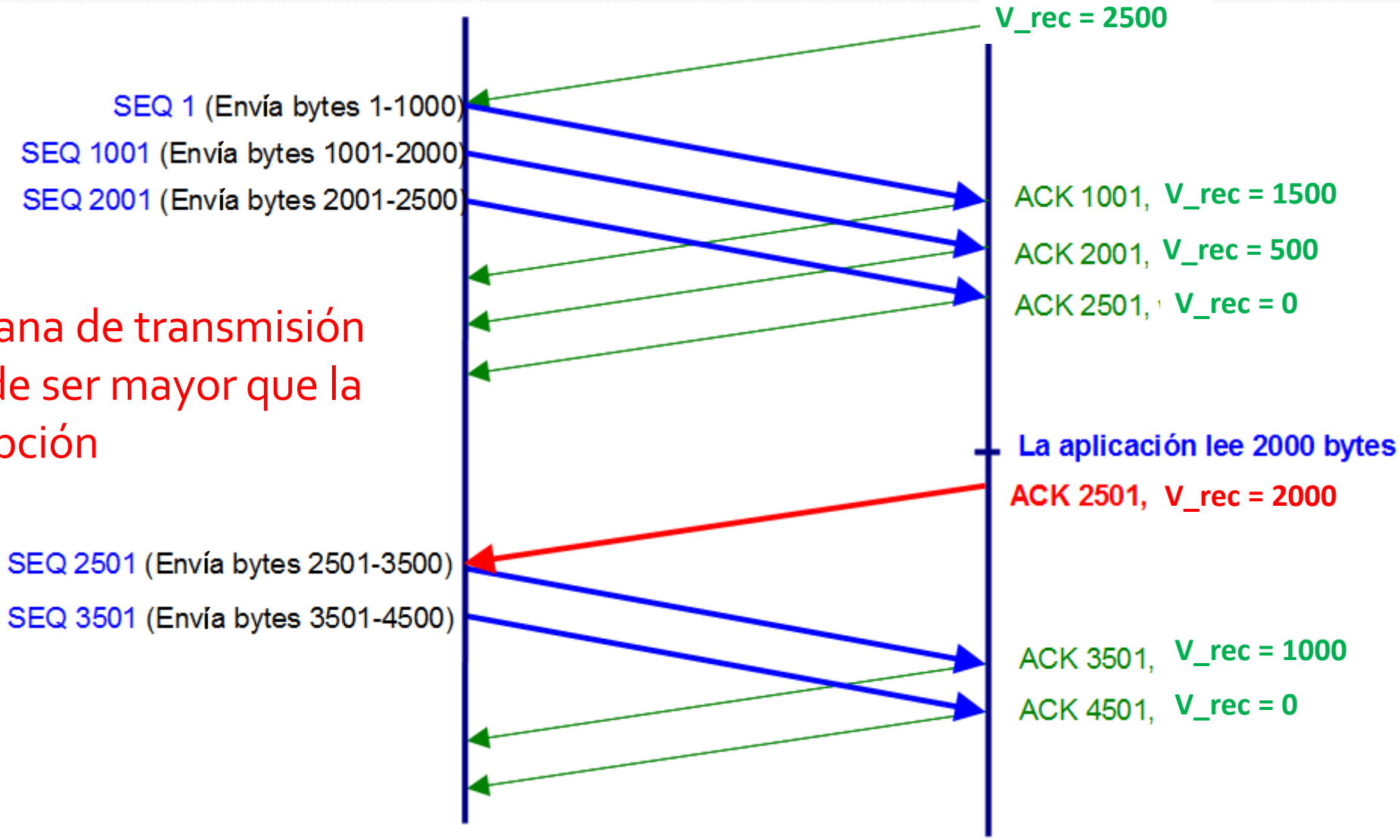
Control de flujo TCP



V_rec (INICIAL) = Buffer del receptor

Emisor se asegura que:
Último Byte enviado – Último Byte reconocido ≤ V_rec

Control de flujo TCP



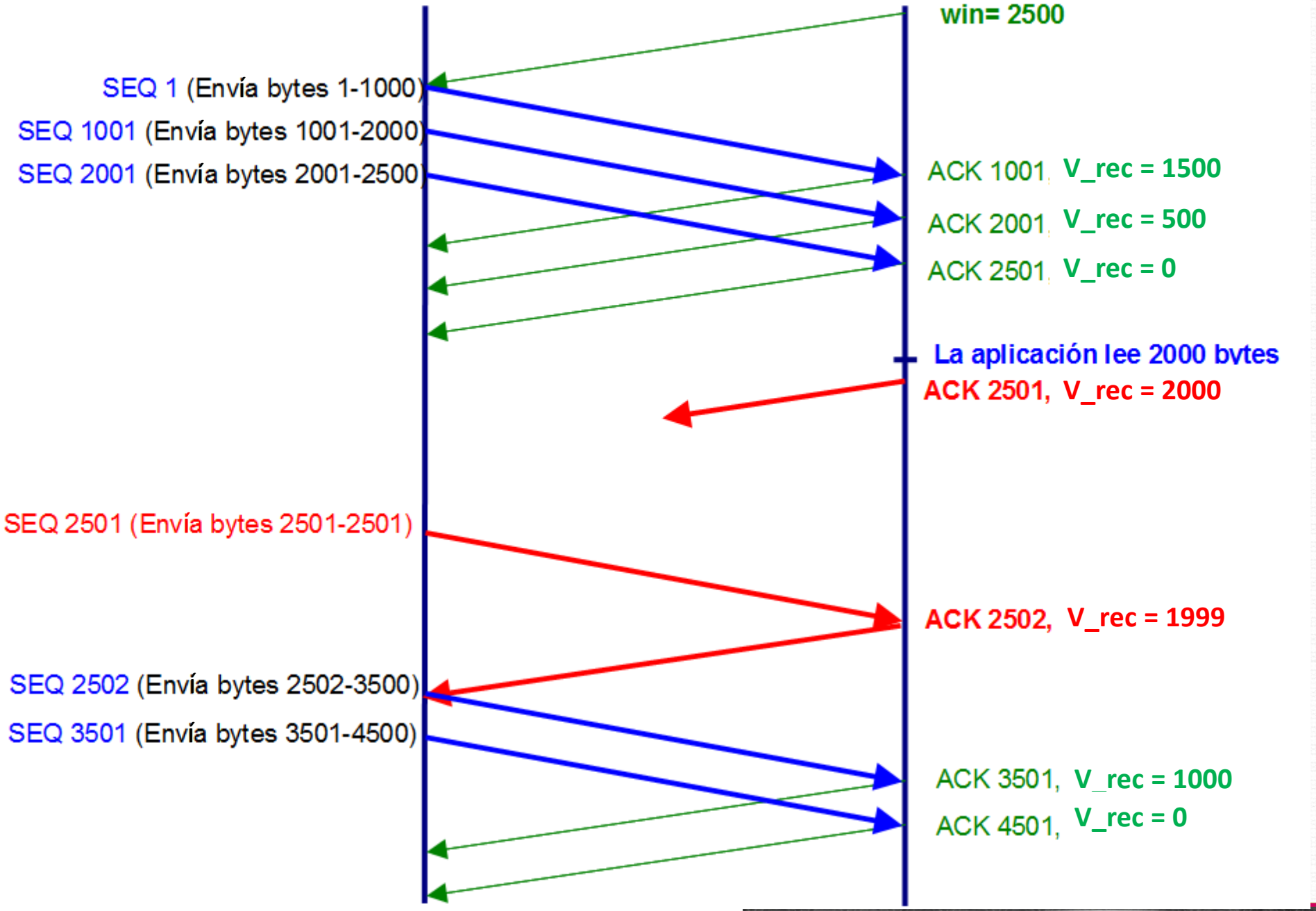
La ventana de transmisión no puede ser mayor que la de recepción

– No se envía ningún dato que no "quepa" en el receptor

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 aunque V_rec=0. El receptor enviará reconocimientos de estos segmentos informando del nuevo valor de su V_rec.
- Comparación con UDP
  - No proporciona control de flujo
  - ¡¡Posible desbordamiento!!

Control de flujo TCP

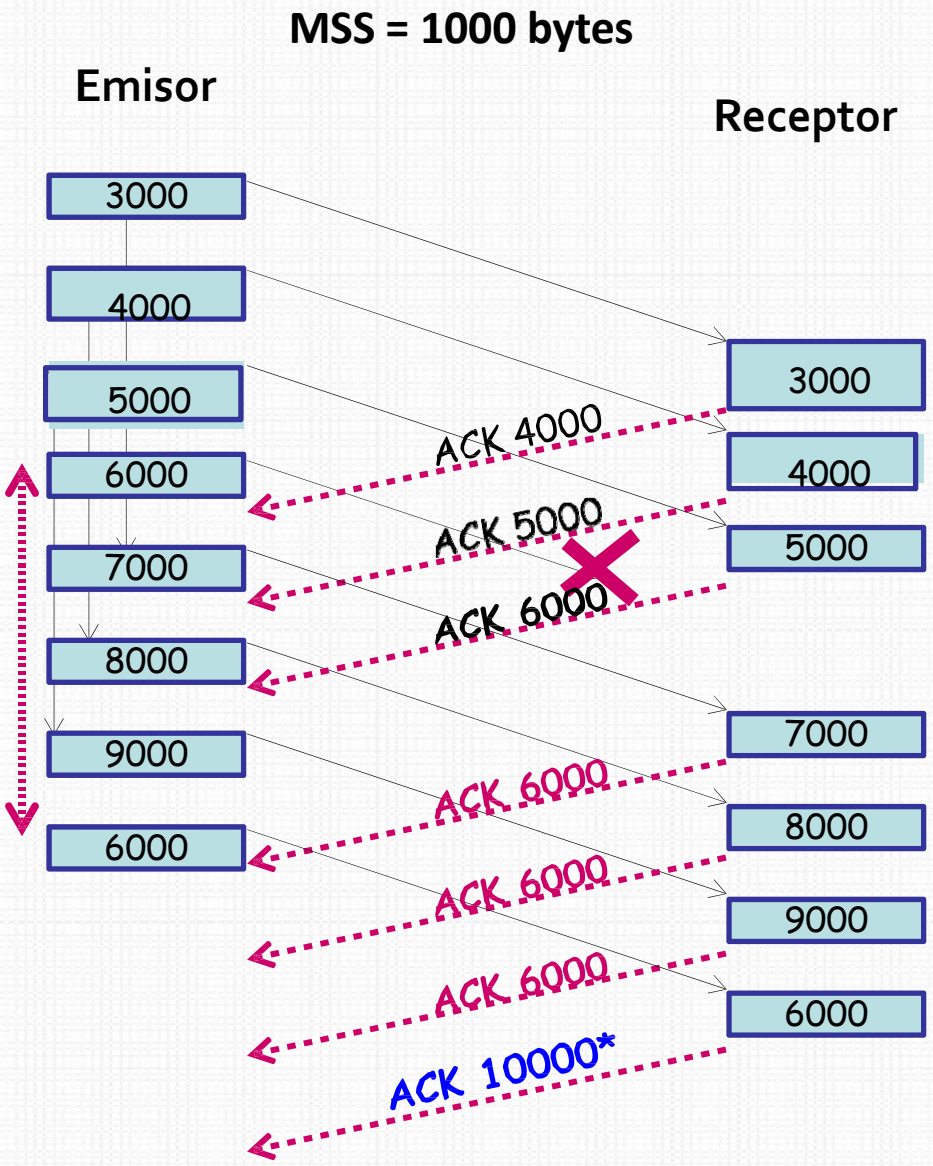


Control de error en TCP

- Utiliza **ARQ** (*Automatic Repeat reQuest*) :
  - 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 recepción, V_rec**)
  - Si los datos no llegan en orden (no definido en la RFC)
    - Se pueden descartar: **vuelta atrás**
    - Se pueden aceptar: **retransmisión selectiva**

Reconocimientos en caso de pérdidas

- Reconocimientos acumulativos:
  - Cuando se recibe un segmento dañado o se pierde, 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



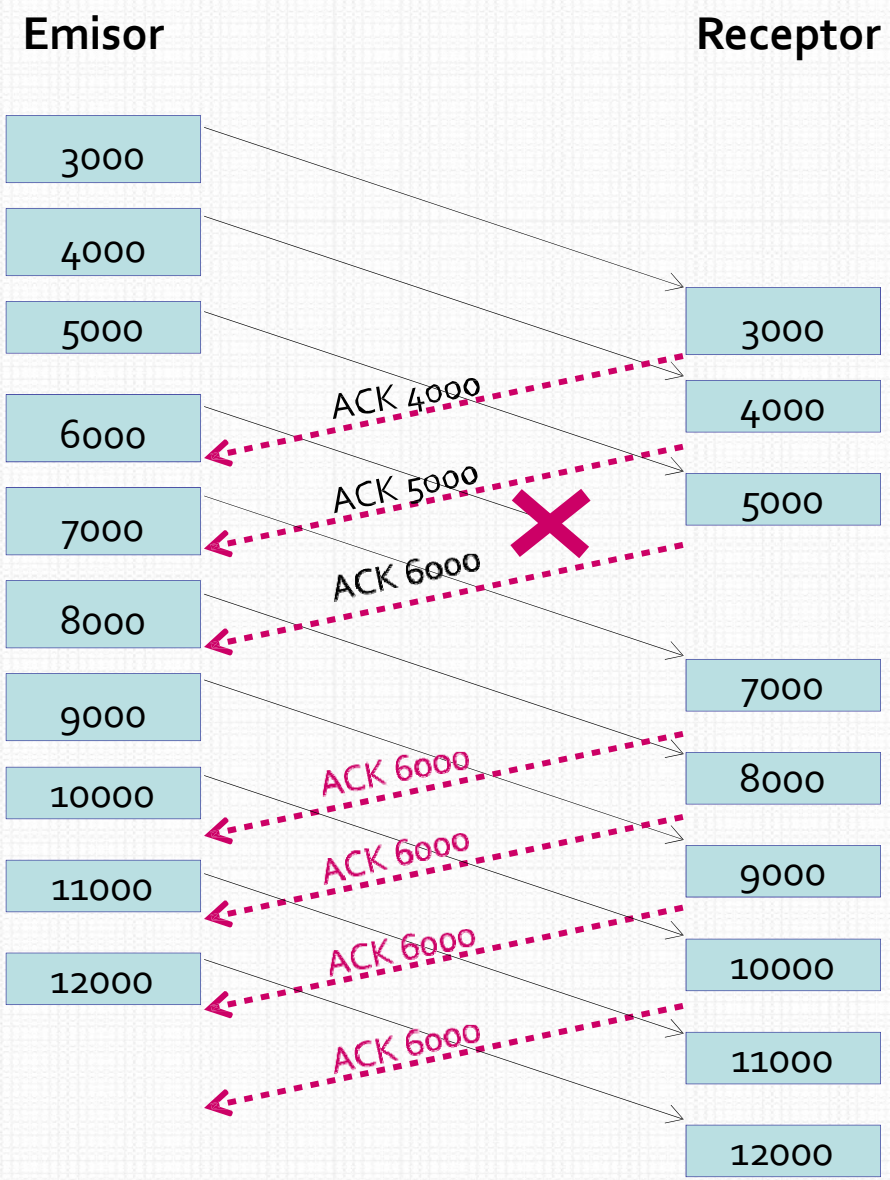
* Suponiendo **retransmisión selectiva**

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)
- Objetivo: reducción del tráfico de reconocimientos

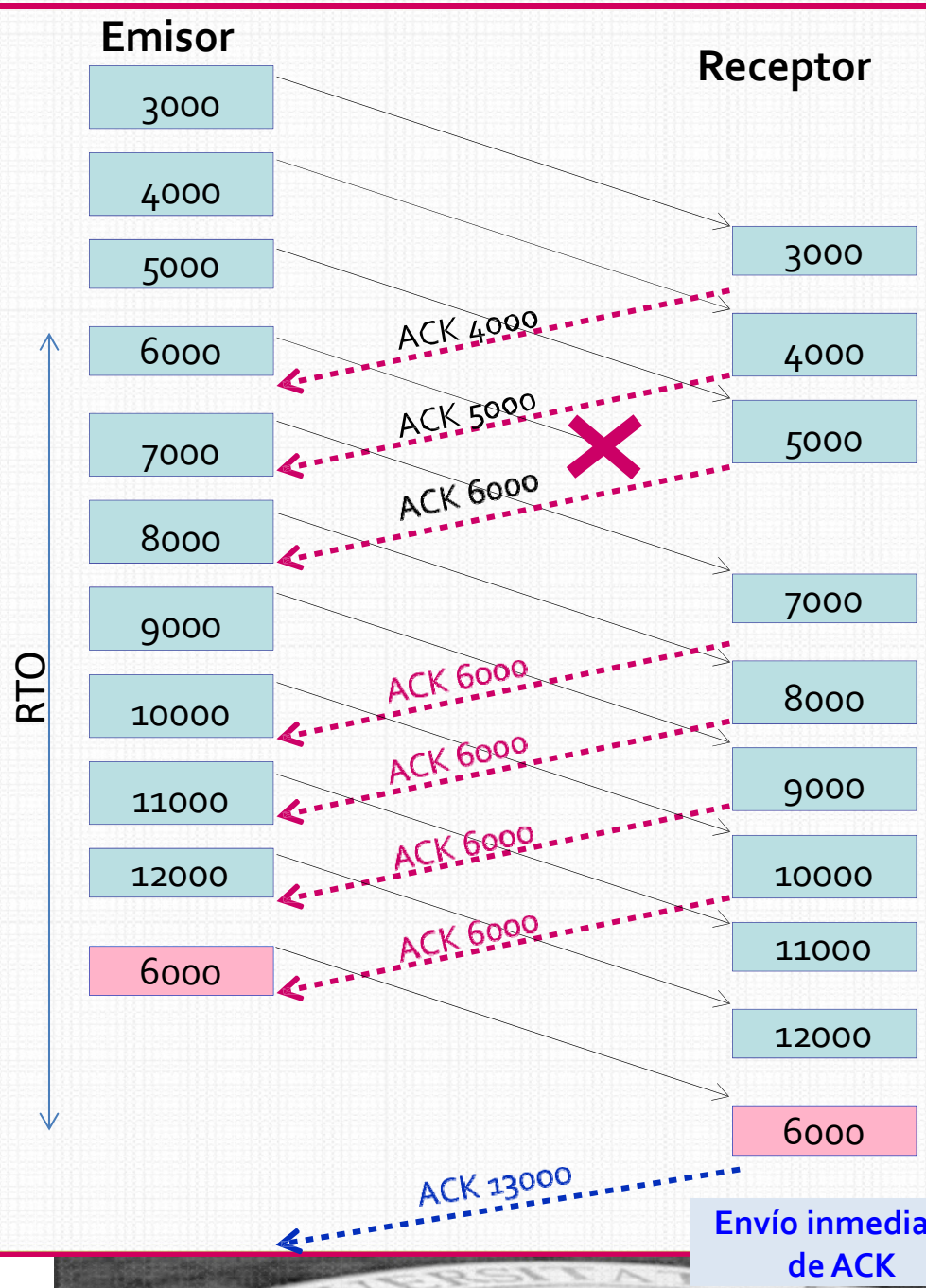
Reconocimientos duplicados

- 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 (RTO) 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).



Generación de ACK en TCP [RFC 1122, RFC 2581]

Evento en el receptor	Acción del receptor TCP
Llegada de segmento en orden con el nº sec. esperado. Todos los datos hasta el nº sec. ya con ACK	**ACK retardado**. Esperar 500ms al siguiente segmento. Si no llega el próximo segmento, mandar ACK
Llegada de segmento en orden con nº sec. esperado. Otro segmento tiene el ACK pendiente	Enviar **inmediatamente** un **ACK** acumulativo reconociendo los dos segmentos ordenados
Llegada de un segmento desordenado con un nº sec. mayor que el esperado. Detección de hueco	Enviar **inmediatamente** un ***ACK duplicado***, indicando el nº sec. del siguiente byte esperado (el límite inferior del hueco)
Llegada de segmento que completa parcial o completamente el hueco existente en los datos recibidos	Enviar **inmediatamente** un **ACK**, siempre que el segmento empiece en el límite inferior del hueco

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 de retransmisión.
  - Si quedan segmentos pendientes de reconocimiento, se reinicia el temporizador
- Si vence el temporizador se produce el reenvío de al menos un segmento (el primero sin reconocer de la ventana) y se reinicia el temporizador
  - En vuelta atrás, se reenvían todos

Temporizador RTO y RTT

- ¿Cuál es el tiempo adecuado de espera para el temporizador de retransmisión **RTO (Retransmission TimeOut)**?
  - Si es demasiado corto: retransmisiones innecesarias
  - Si demasiado largo: poco eficiente
  - Debería ser 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
  - <u>No se emplea directamente</u> 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

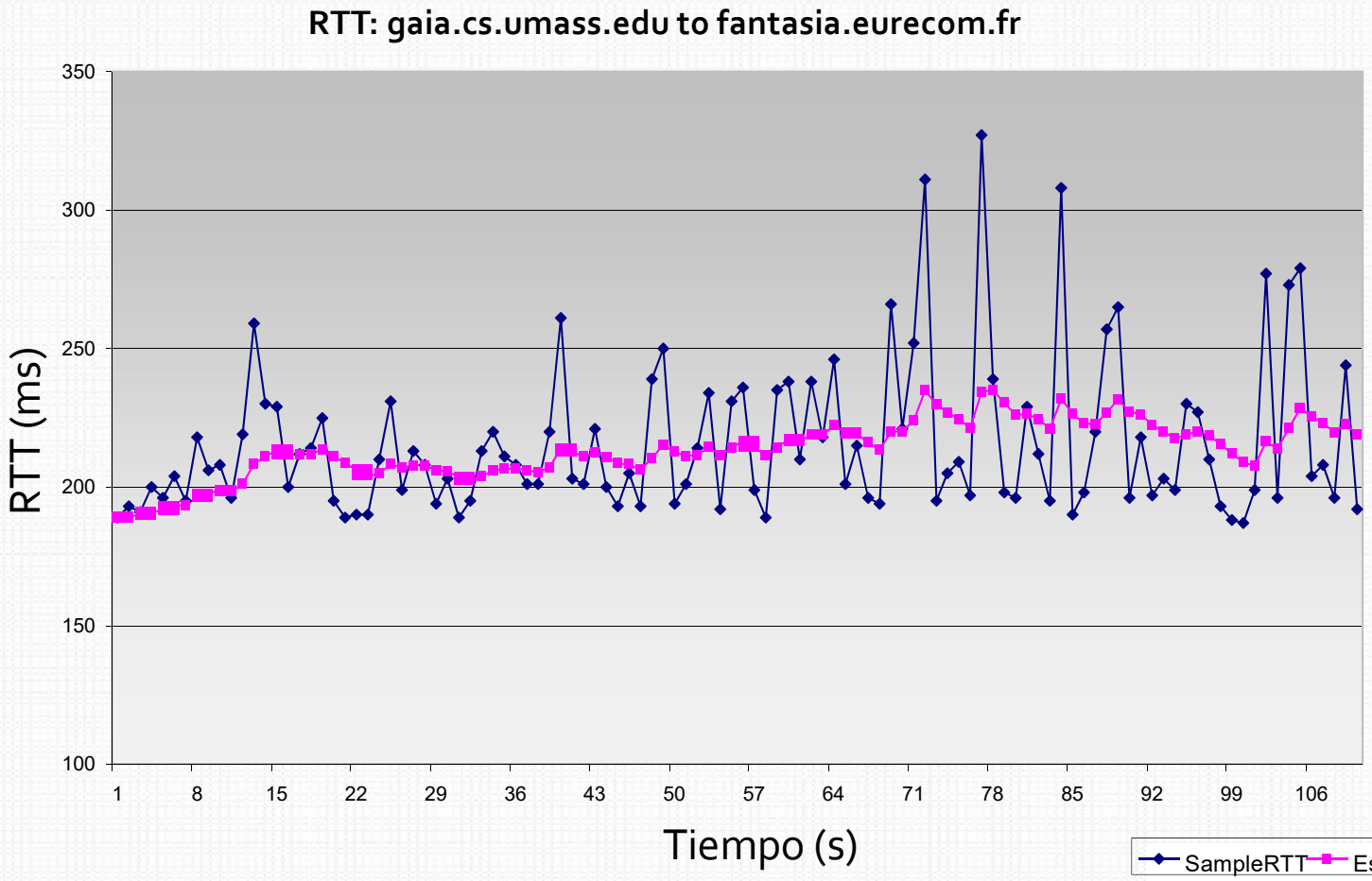
Ejemplo: estimación del RTT

- Cálculo (*algoritmo de van Jacobson*, *RFC 2988*):

EstimaciónRTT = (1 – α) EstimaciónRTT + α MuestraRTT

0 < α < 1; normalmente α = 1/8=0,125

EWMA: Exponential Weigthed Moving Average (Media Móvil Exponencialmente Ponderada) ➔ asigna más peso a las muestras más recientes



Estimación del *Timeout* (RFC 6298)

- RTO (Intervalo de timeout): ***EstimaciónRTT*** más un margen de seguridad (para evitar retransmisiones innecesarias)

- DevRTT nos da una medida de la variabilidad de la muestra

$$DevRTT = (1-\beta) * DevRTT + \beta * |MuestraRTT - EstimaciónRTT|$$

Media EWMA

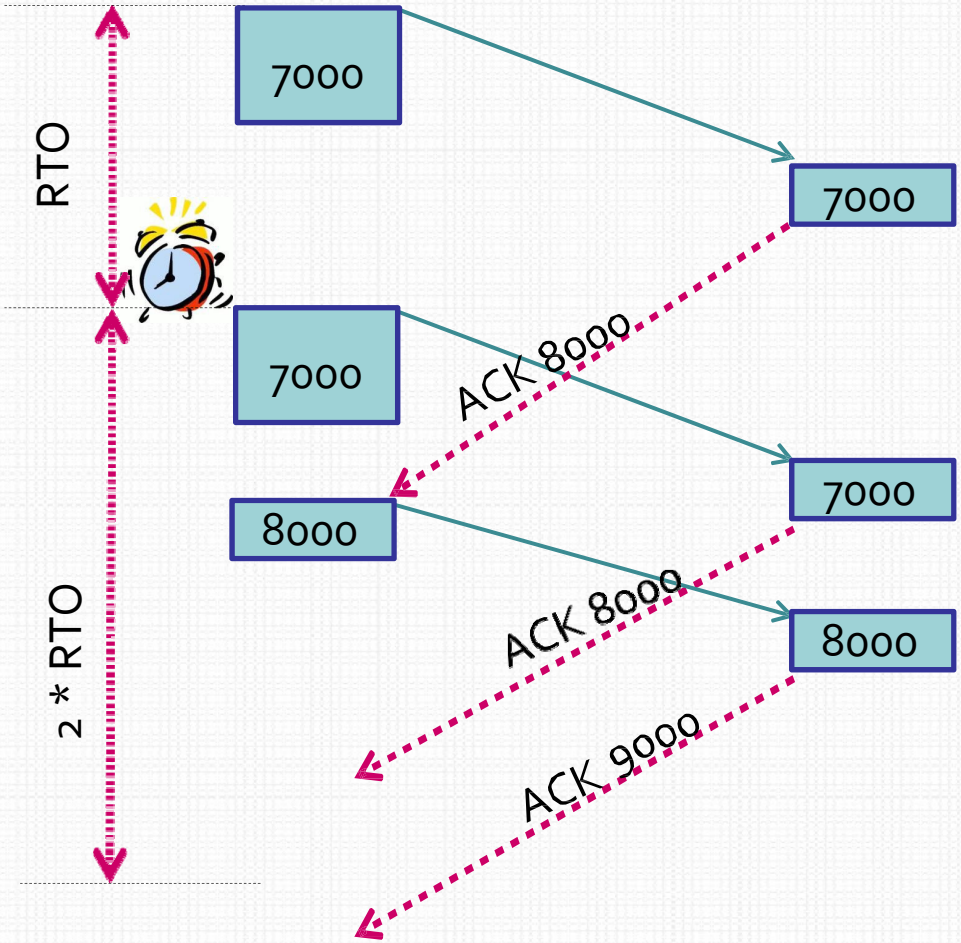
  – Valor recomendado $\beta=1/4$

"Margen de seguridad"

$$RTO = EstimaciónRTT + 4 * DevRTT$$

Duplicación del *Timeout* (RFC 6298)

- Problema:
  - Ambigüedad en el RTT cuando hay retransmisiones
  - Se puede confundir el ACK del segmento original con el de la retransmisión
- Solución (algoritmo de Karn):
  - No tener en cuenta las medidas del RTT de los segmentos retransmitidos
  - **Al retransmitir doblar el valor del temporizador**
    - ***Exponential Backoff***



Algoritmo de Karn

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. El control de la congestión TCP
   6. Opciones TCP

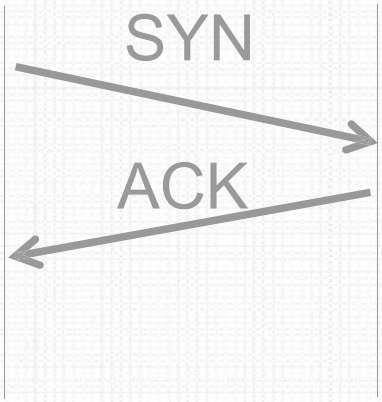
Sesión 4

4.4. Gestión de la conexión TCP

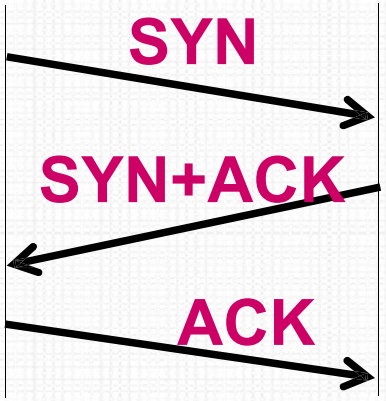
1. Establecimiento de conexión
   - `s = new Socket("NombreHost",NumPuerto);`
   - Intercambio de mensajes
2. Transmisión de datos bidireccional
   - `entrada.read(b);`
   - `salida.write(b)`
3. Cierre de la conexión
   - `s.close();`

Establecimiento de la conexión TCP

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



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



Establecimiento de la conexión TCP

(Apertura activa) **Cliente**

Servidor *(Apertura pasiva)*

```
ss=new ServerSocket(pto);
```

```
S=new Socket(...);
```

```
ss.accept();
```

Envía **SYN x**

Nº secuencia inicial cliente

SYN

Recibe **SYN x**

Envía **SYN y, ACK x+1**

SYNACK

Recibe **SYN y,ACK x+1**

Envía **x+1,ACK y+1**

Nº secuencia inicial servidor

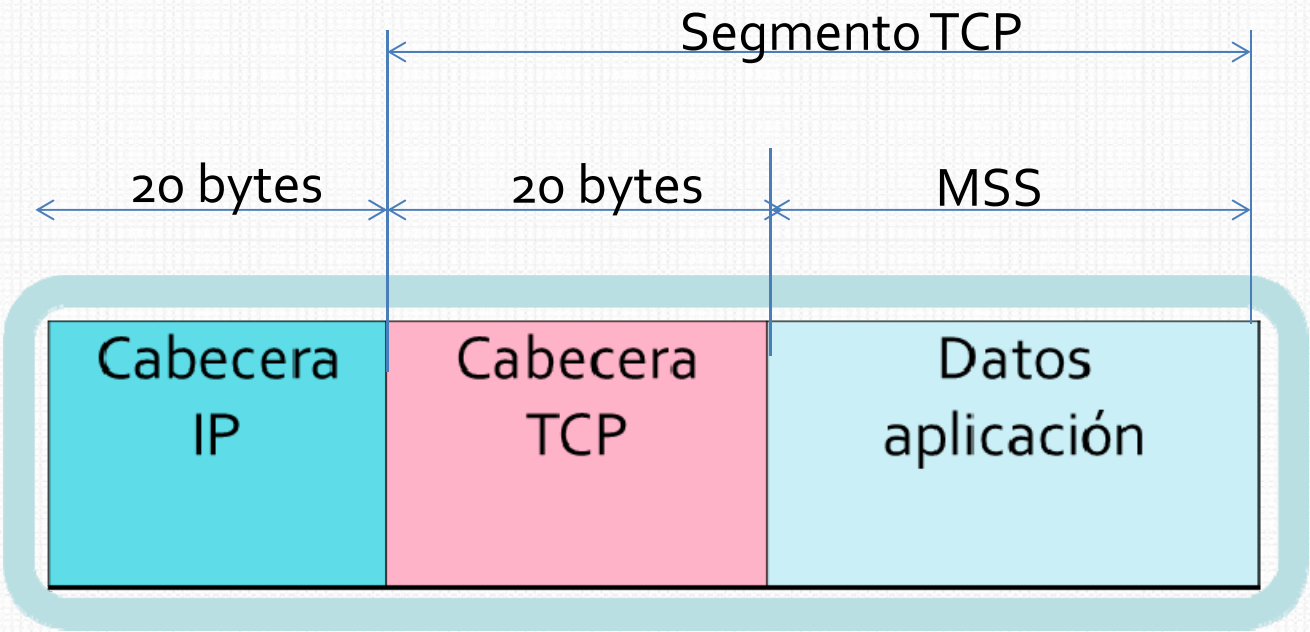
ACK

Recibe **ACK y+1**

Puede transportar datos del cliente

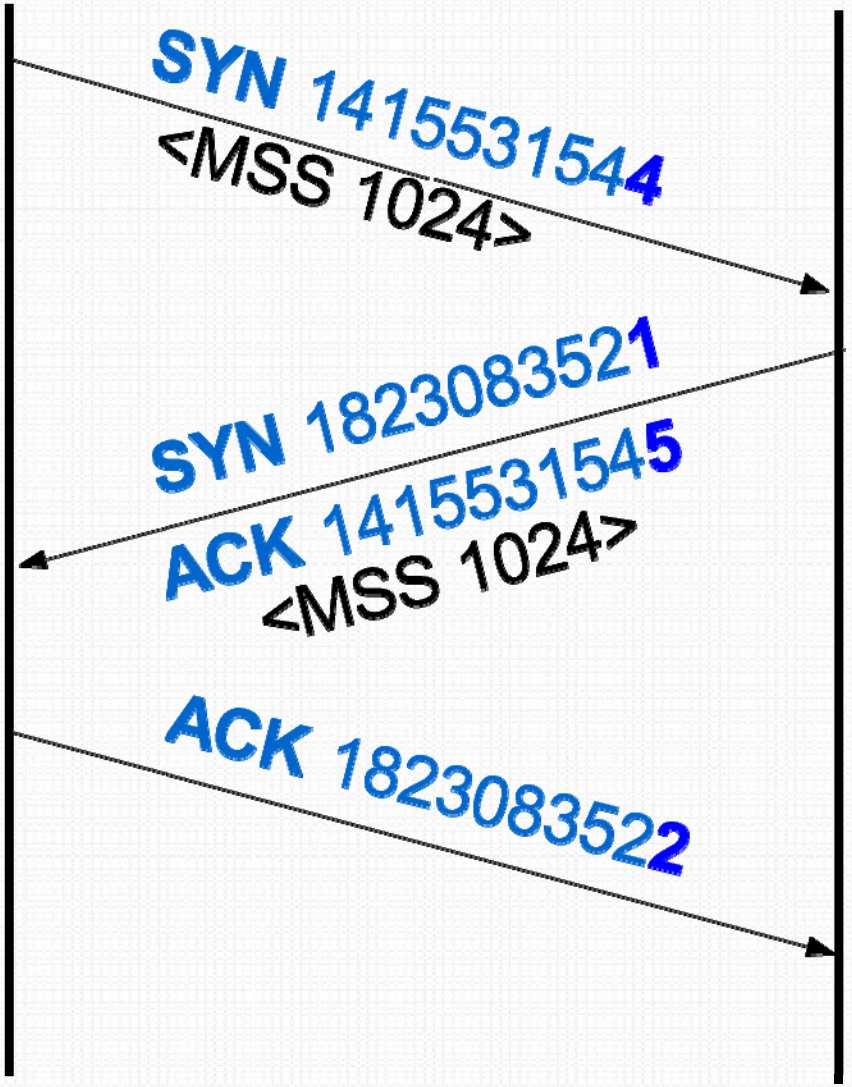
Conexión establecida

Maximum Segment Size (MSS)



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

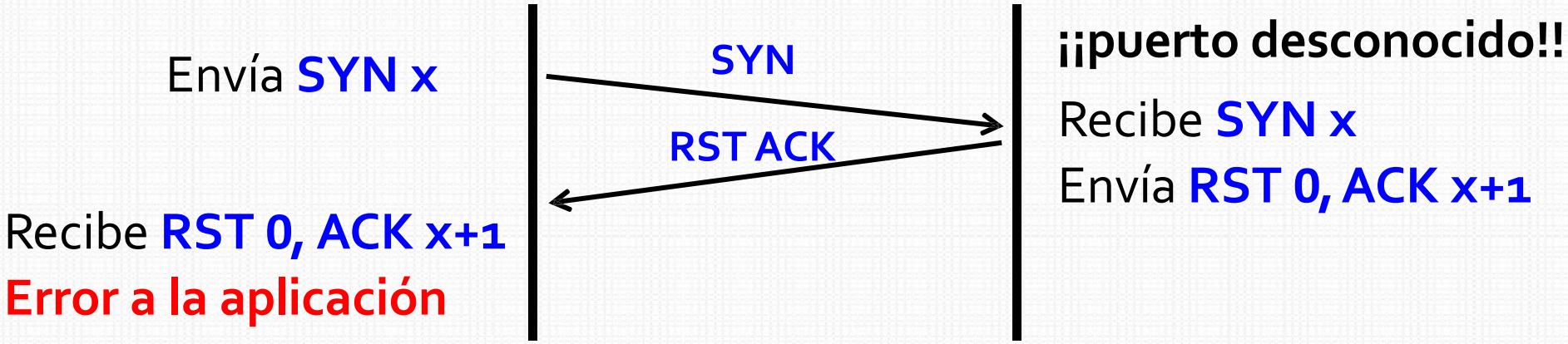
Traza de un establecimiento de conexión TCP



El bit RST

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

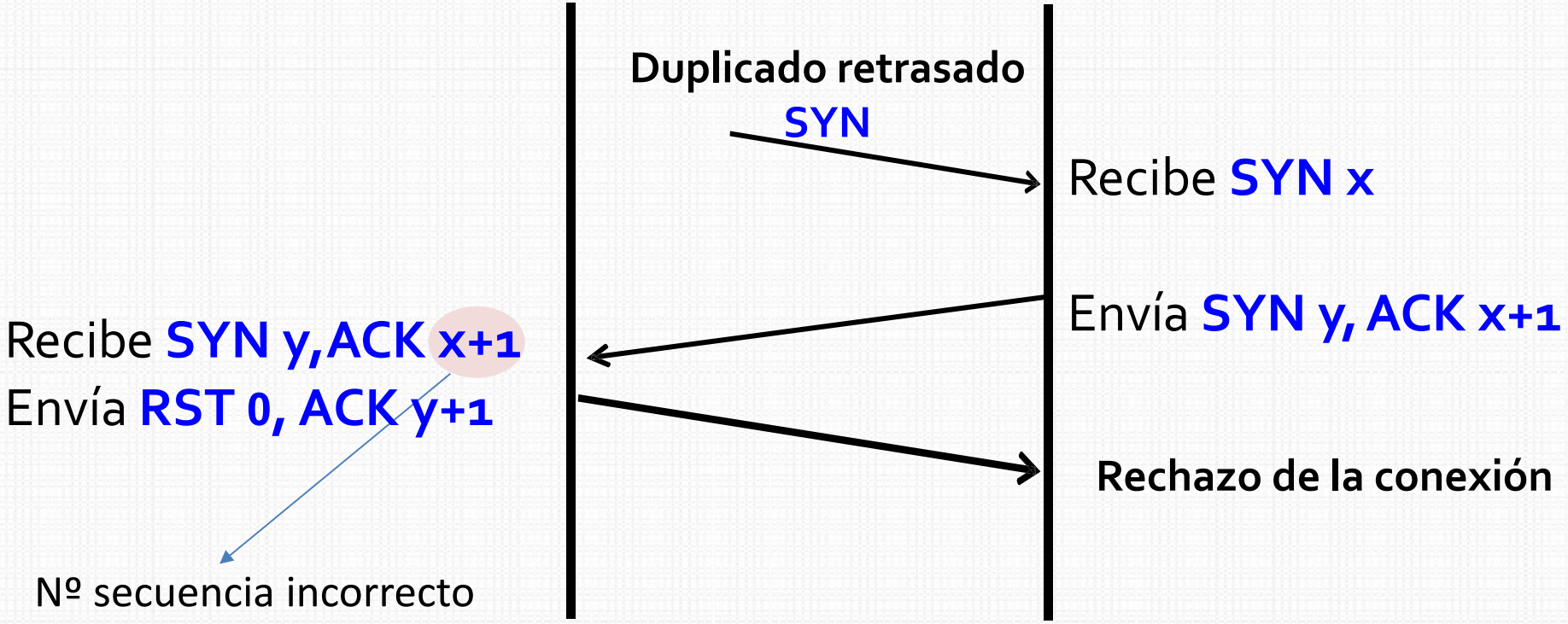
```
S=new Socket(...);
```



(Conexión NO establecida)

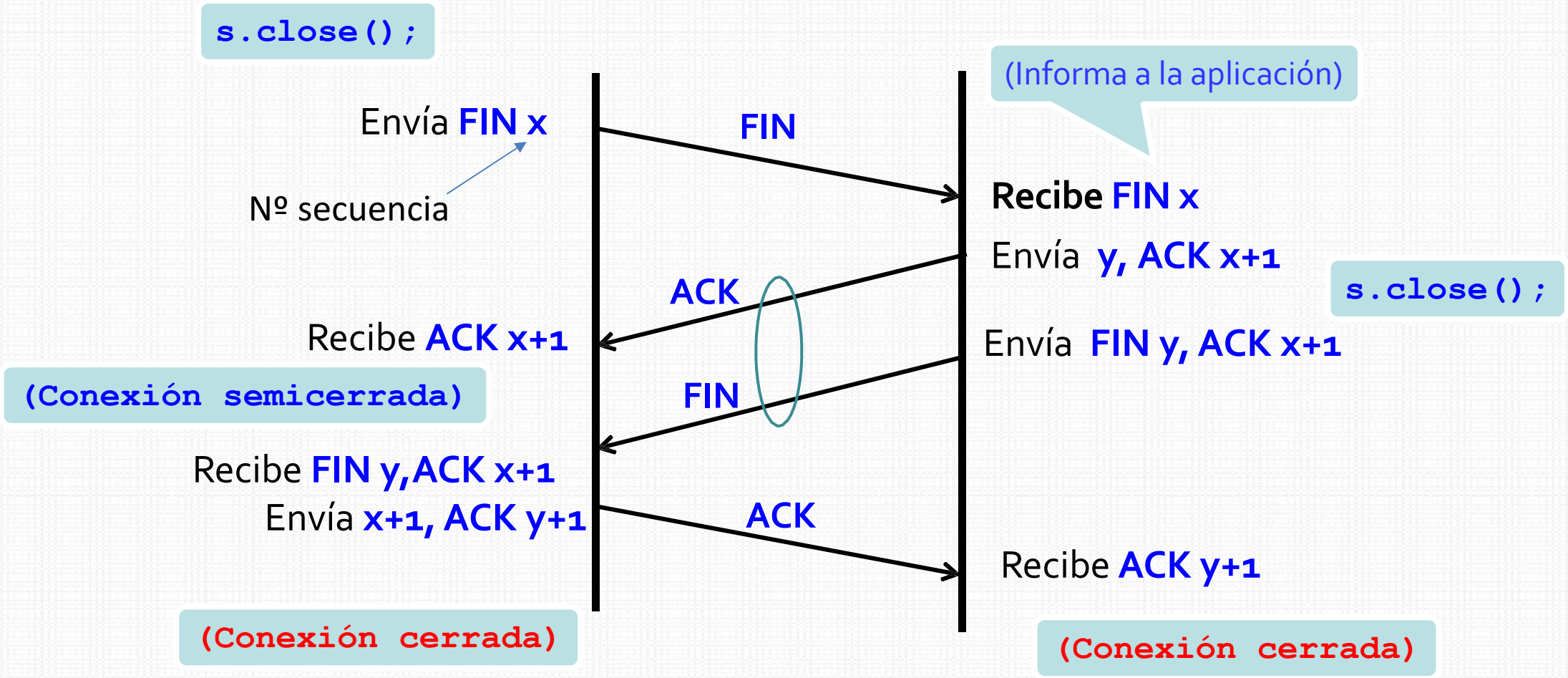
El bit RST

- RST con duplicado retrasado en el establecimiento

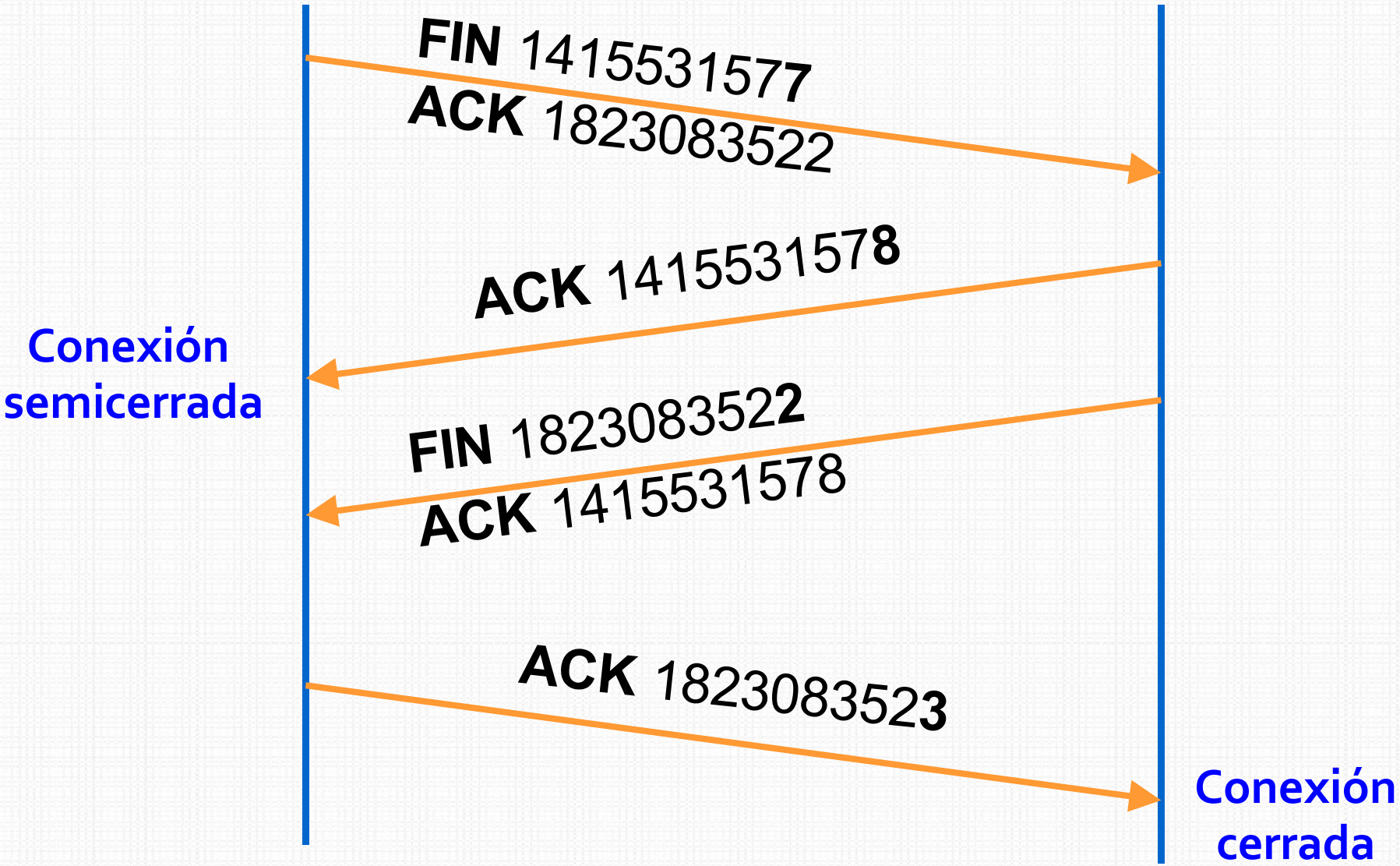


Cierre de la conexión TCP

- Cualquiera de los dos procesos puede finalizar la conexión
- Cierre de los dos flujos de datos de forma independiente



Traza del cierre de conexión TCP

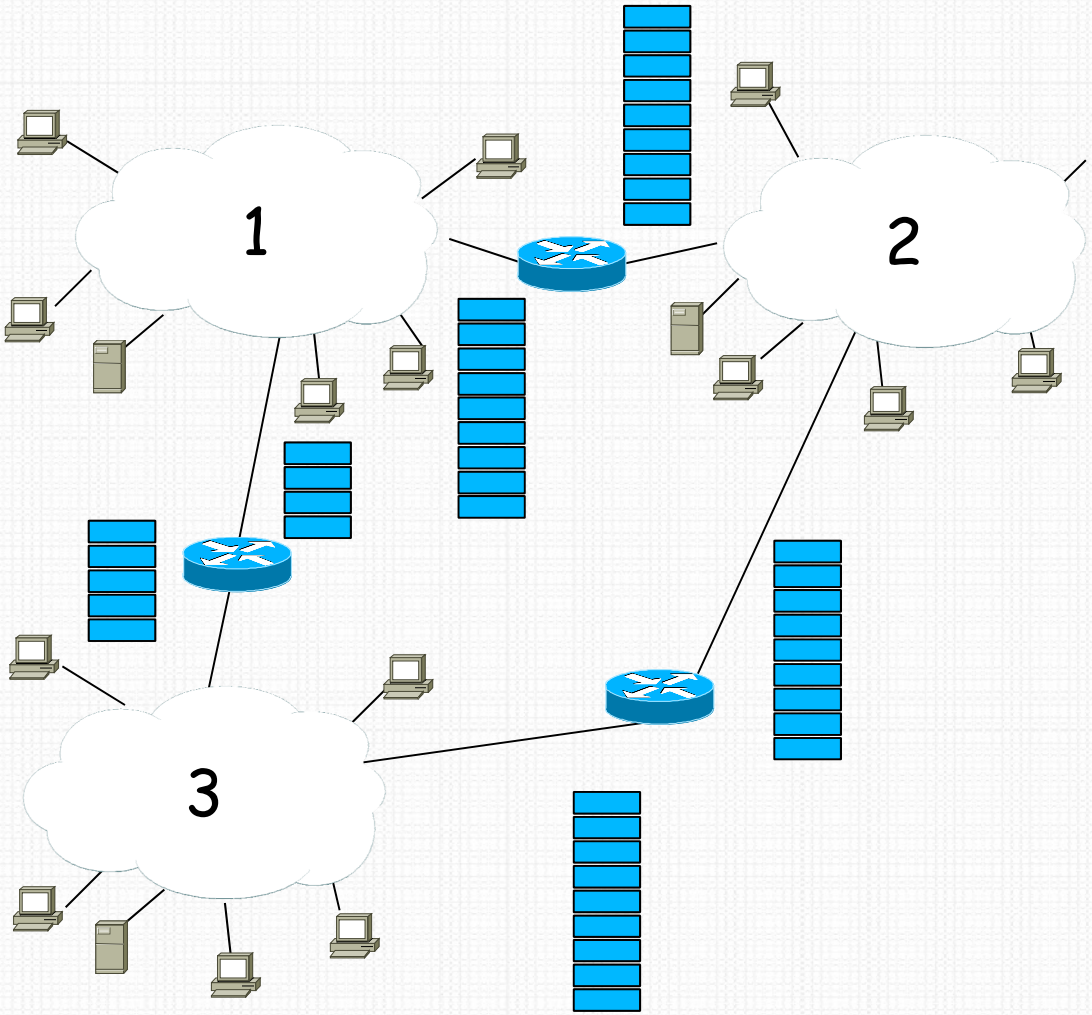


Índice

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. El control de la congestión TCP
    6. Opciones TCP

La congestión en la red

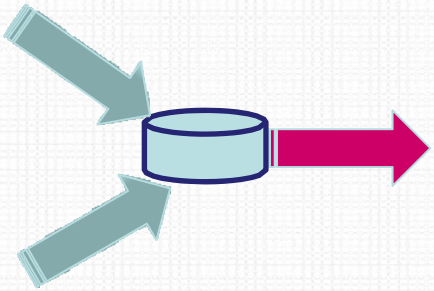
- Los routers tienen capacidad limitada
  - colas (buffers)
- Si las colas se llenan, los routers descartan paquetes



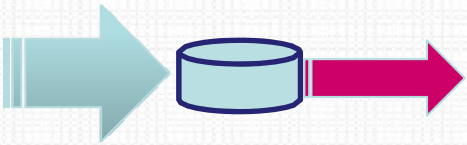
La congestión en la red

Causas básicas

- Enlaces compartidos en los routers



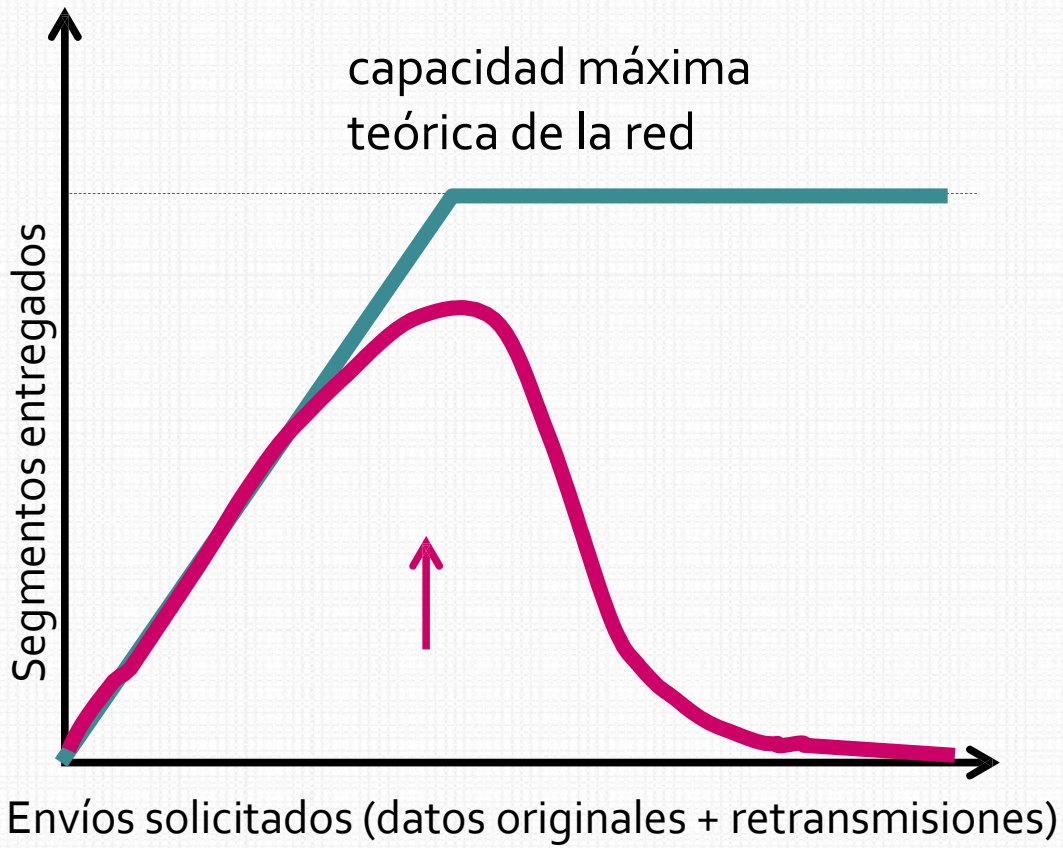
- Tránsito a enlaces con menor ancho de banda que los precedentes



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
- **Método**: limitar el envío de tráfico de cada emisor en función de la congestión de red percibida

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 ¿qué algoritmo emplear?

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
    - El valor de 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)

VentanaTransMáx = <u>mínimo</u> (V_rec,VentanaCong)

¿Cómo se mide la VentanaCong?

- Tanto V_rec como VentanaCong se miden en bytes
  - Longitud de la mayoría de segmentos = MSS bytes
  - La relación es:

número de bytes/MSS = número de segmentos

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

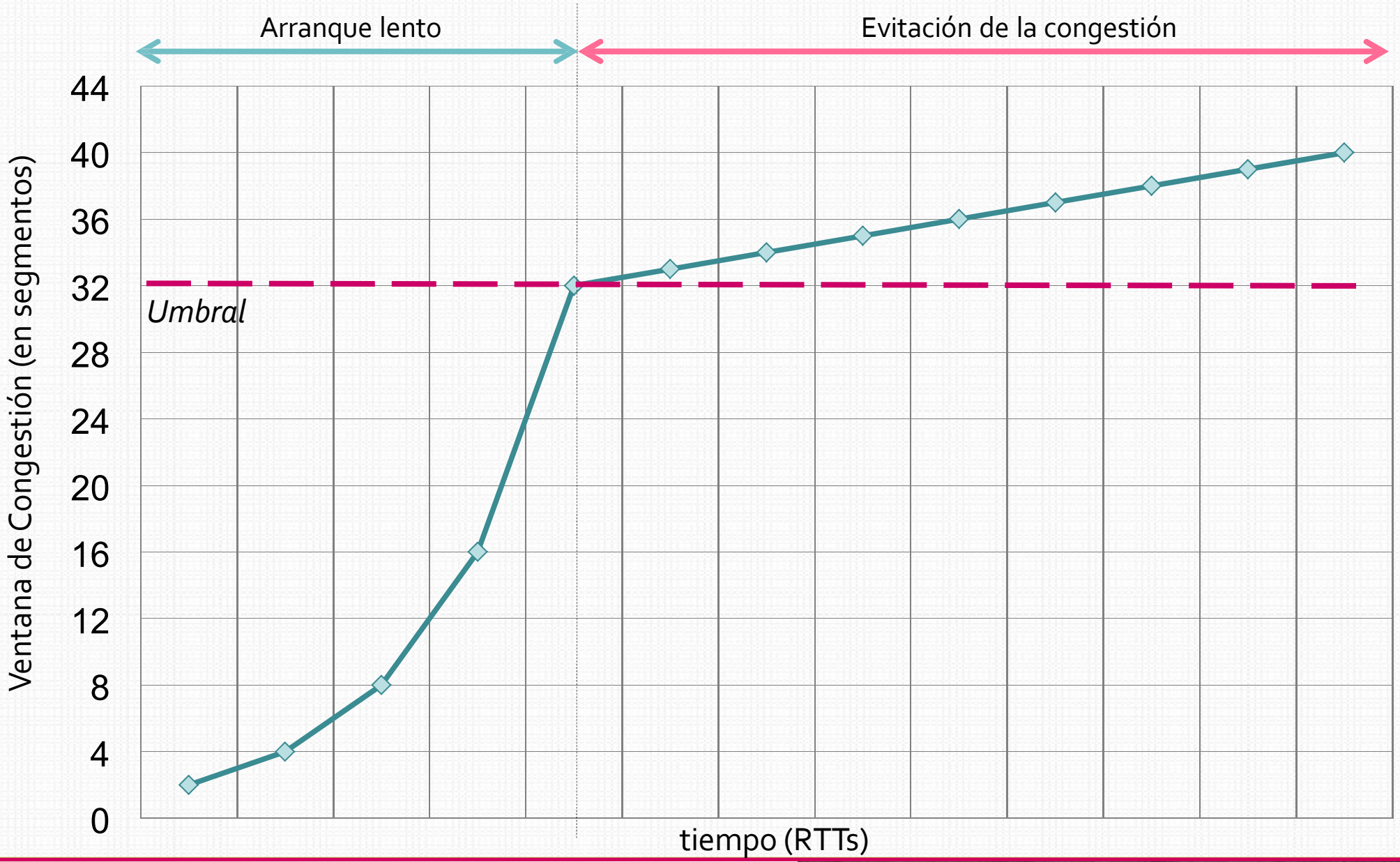
Arranque lento *(Slow Start)*

- Al comenzar la conexión se podrían 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

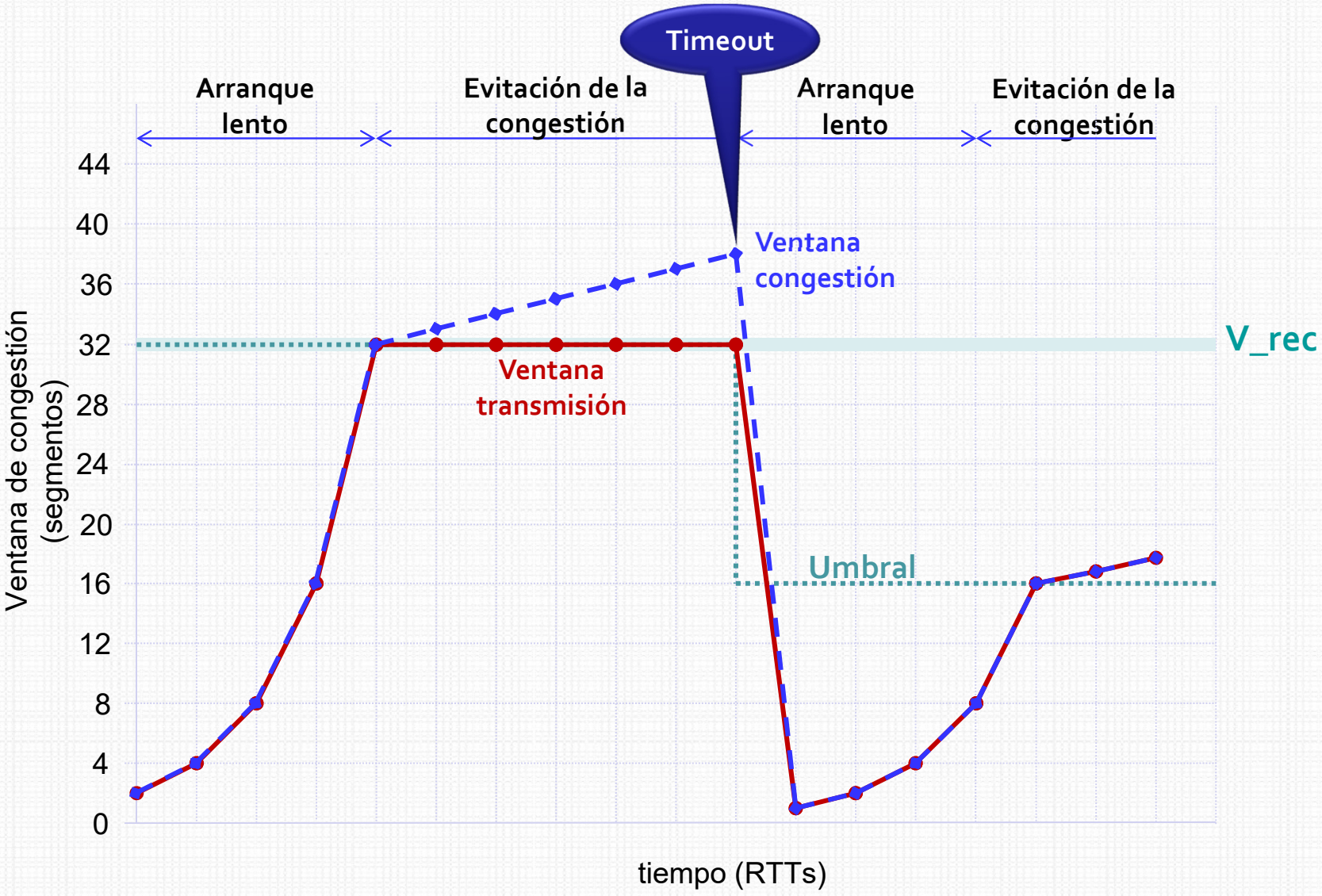
Evitación de congestión: ejemplo



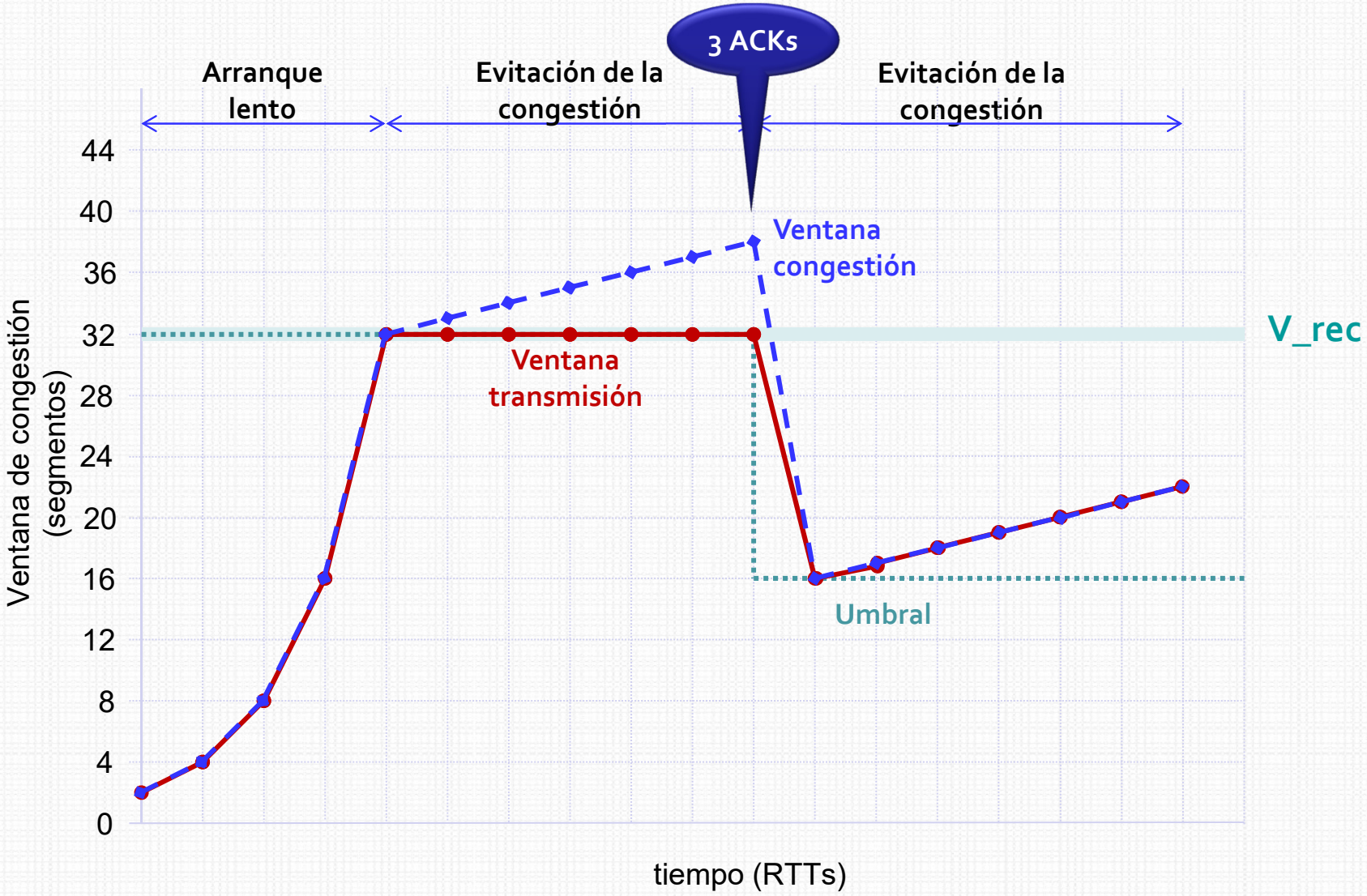
Recuperación rápida

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

Recuperación rápida (pérdida)



Recuperación rápida (3 ACKs)

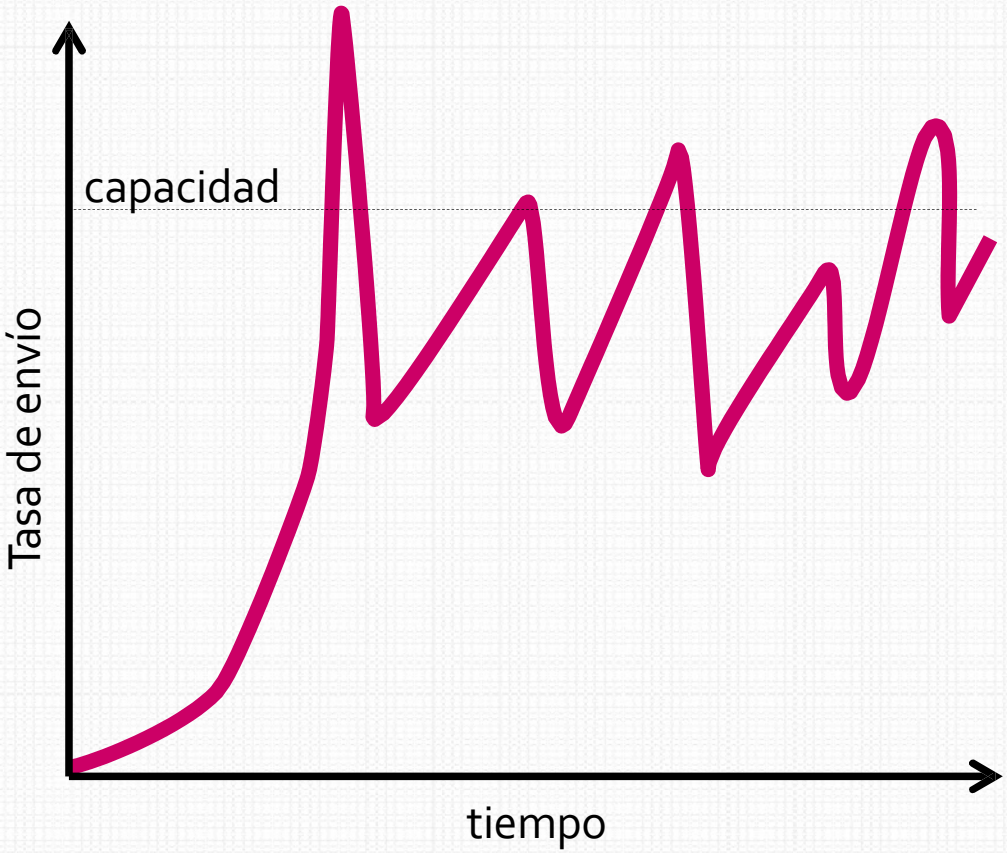


Control de la congestión: resumen

- 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 la VentanaCong y se aplica *Arranque lento*
    - Si llegan reconocimientos duplicados, se reduce el Umbral y la VentanaCong y se aplica *Evitación de la congestión*

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



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. El control de la congestión TCP
    6. Opciones TCP

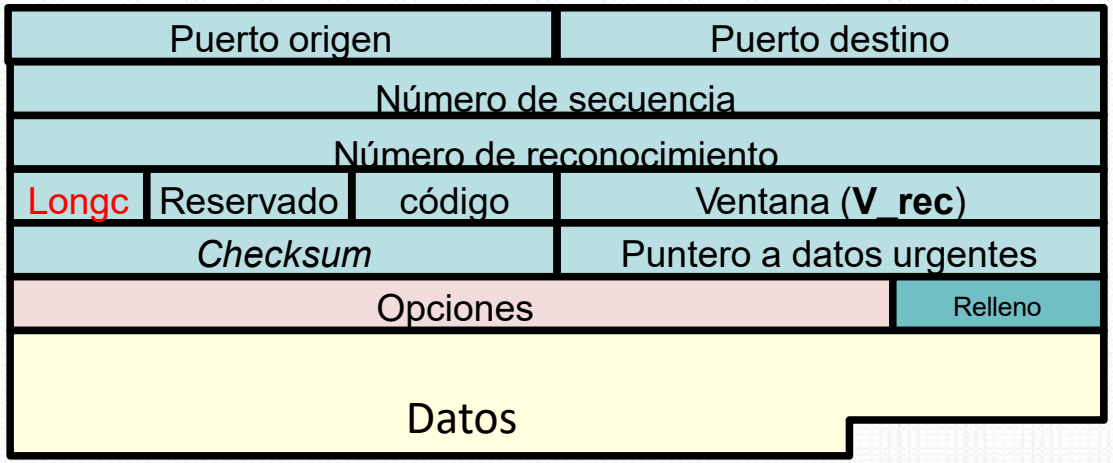
Sesión 5

Opciones TCP

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

Opciones TCP

- Campo de Opciones
  - Variable 0-320 bits (40 bytes máximo), divisible por 32
  - Longitud determinada por el campo *Longc* de la cabecera (4 bits)
  - Cada opción tiene hasta tres campos:
    - Código (1 byte), Longitud (1 byte), Datos (variable)



Campo de Opciones

Nota: 0x02, 0x03 y 0x04 solo se emplean en SYN y SYN_ACK

Código	Opción	Longitud	Datos	Utilización
0x00	Fin opciones	n=2, 3 ó 4	Hasta (n-2) bytes a 0x00	Completa hasta el final con múltiplos de 32 bits
0x01	No operación			Alinea opciones a palabras de 32 bits
0x02	Tamaño máximo de segmento	0x04	MSS (2 bytes)	Negociación del tamaño máximo de segmento
0x03	Escala de ventana TCP	0x03	Despl. (1 byte)	Permite aumentar la ventana de transmisión
0x04	Reconocimientos selectivos	0x02		Activa reconocimiento selectivo
0x05	SACK	n=10, 18, 26 ó 34	Izq. Bloque 1 (4 bytes) Der. Bloque 1 (4 bytes) Izq. Bloque 2 (4 bytes) Der. Bloque 2 (4 bytes) …	Permite reconocimientos explícitos para bloques de datos no consecutivos
0x08	Marcas de tiempo	0x0A	Valor TS (4 bytes) Respuesta de Eco TS (4 bytes)	Permite etiquetar los segmentos con tiempos locales para mejorar el cálculo del RTT

Opciones TCP: *TimeStamp*

- Permite calcular el RTT de forma más precisa (RFC 1323)
  - Dos campos de 4 bytes

Código = 0x08

10 bytes

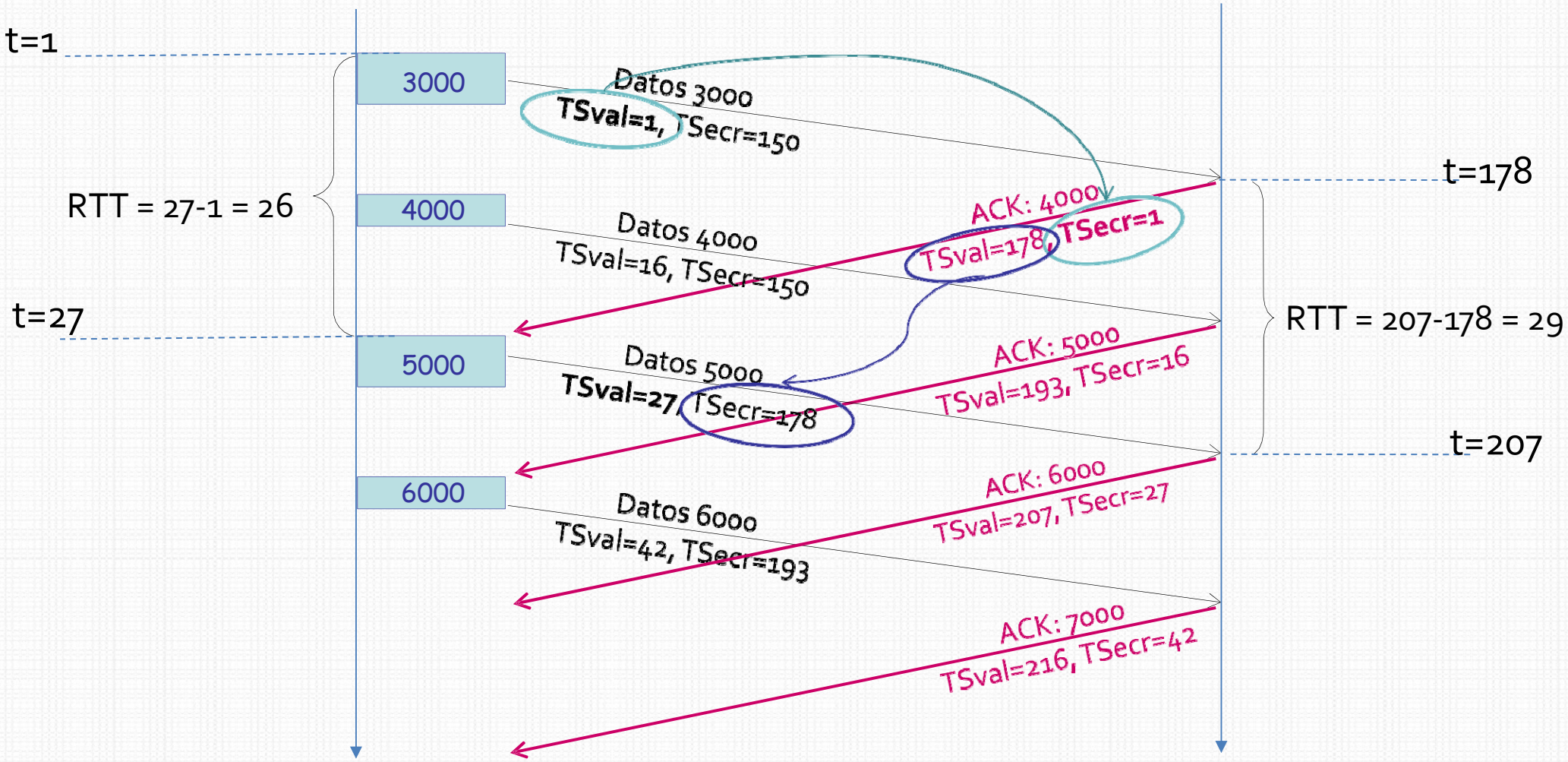
Tiempo actual en el emisor (TSval)

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Kind								Length							
Timestamp Value -															
Timestamp Echo Reply -															

Devuelve el valor TSval que venía con los datos reconocidos. Válido si ACK=1

Opciones TCP: *TimeStamp*

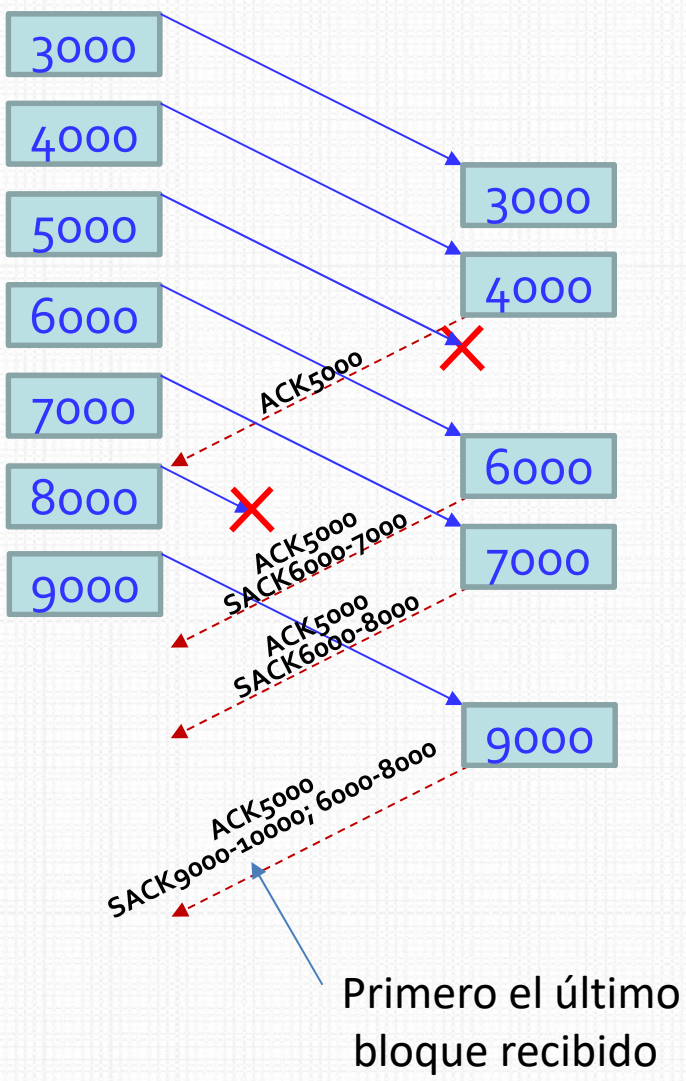
TSval =**Timestamp Val**ue
TSecr =**Timestamp Ec**ho **R**eply



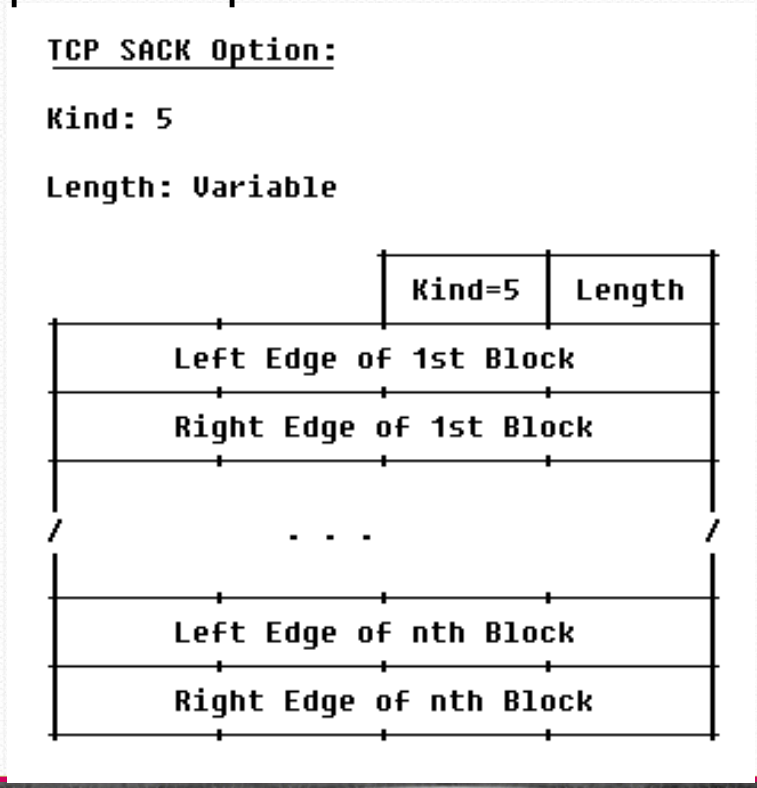
Opciones TCP: ACKs selectivos

- Con ACKs acumulativos no se puede confirmar la recepción de segmentos fuera de orden
  - Puede provocar retransmisiones innecesarias
- Los ACKs selectivos (SACK) permiten reconocer bloques de datos no contiguos (RFC 2018)
- Opción *SACK-permitted* en segmentos SYN (Código=0x04) para activar ACKs selectivos

Opciones TCP: ACKs selectivos



- Información SACKs:
  - Aparece primero el último bloque recibido (máx. 4 bloques)
  - El límite superior del bloque SACK indica el siguiente octeto de datos que se espera



Opciones TCP: escala de ventana

- En TCP, la longitud máxima de ventana = 65.535 bytes
  - Ventana = $2^{16} - 1$
- El envío continuo requiere:

  Tamaño Ventana (en bits) >= RTT x Vtrans
- El tamaño máximo de la ventana puede ser una limitación importante en redes de alta velocidad:
  - Si RTT = 1 ms y Vtrans = 1 Gbps
    - RTT*Vtrans = 10^6 bits= 125.000 bytes
- Con esta opción se especifica un K,

  nueva Ventana = 2^K x Ventana
  - 0 <= K <= 14 → Máxima ventana escalada = 1 GiB (2^{30})
  - **Opción escala de ventana solo permitida en segmentos SYN** (Código = 0x03)