

# TCP

Area de Ingeniería Telemática  
<http://www.tlm.unavarra.es>

Arquitectura de Redes, Sistemas y Servicios  
3º Ingeniería de Telecomunicación

# Temario

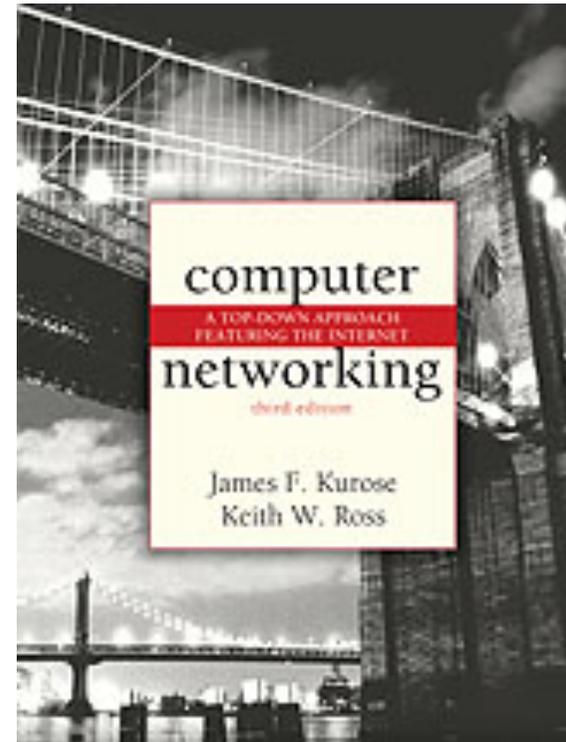
- Introducción
- Arquitecturas, protocolos y estándares
- Conmutación de paquetes
- Conmutación de circuitos
- Tecnologías
- Control de acceso al medio en redes de área local
- Servicios de Internet

# Temario

1. Introducción
2. Arquitecturas, protocolos y estándares
3. **Conmutación de paquetes**
  - Principios
  - **Problemas básicos**
    - Como funcionan los routers (Nivel de red)
    - Encaminamiento (Nivel de red)
    - **Transporte fiable (Nivel de transporte en TCP/IP)**
    - **Control de flujo (Nivel de transporte en TCP/IP)**
    - Control de congestión (Nivel de transporte en TCP/IP)
4. Conmutación de circuitos
5. Tecnologías
6. Control de acceso al medio en redes de área local
7. Servicios de Internet

# Material

Del Capítulo 3 de  
Kurose & Ross,  
**“Computer Networking a top-down approach  
featuring the Internet”**  
Addison Wesley

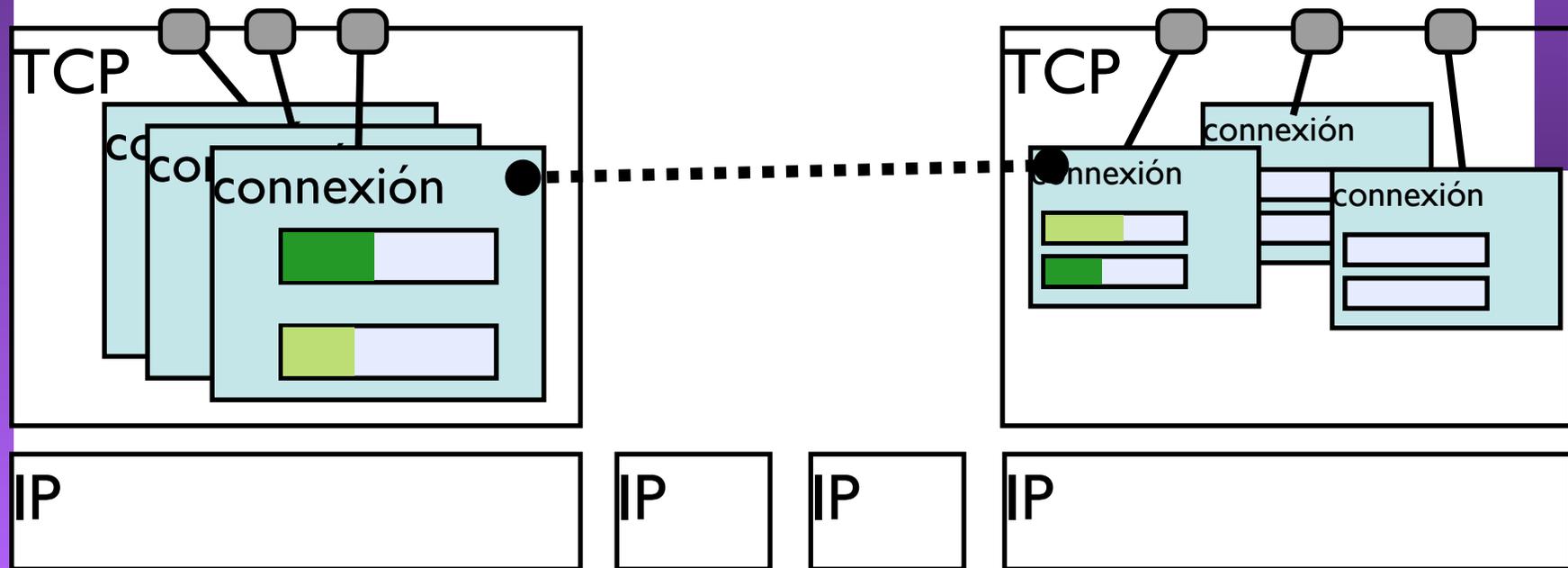


# TCP

- Protocolo de transporte de Internet (RFC 793)
- Transporte fiable
  - Entrega garantizada
  - Entrega en orden
- Orientado a conexión
  - Stream bidireccional (como si fuera un fichero) entre los dos extremos
  - No mantiene las fronteras de los mensajes
- Con control de flujo y congestión

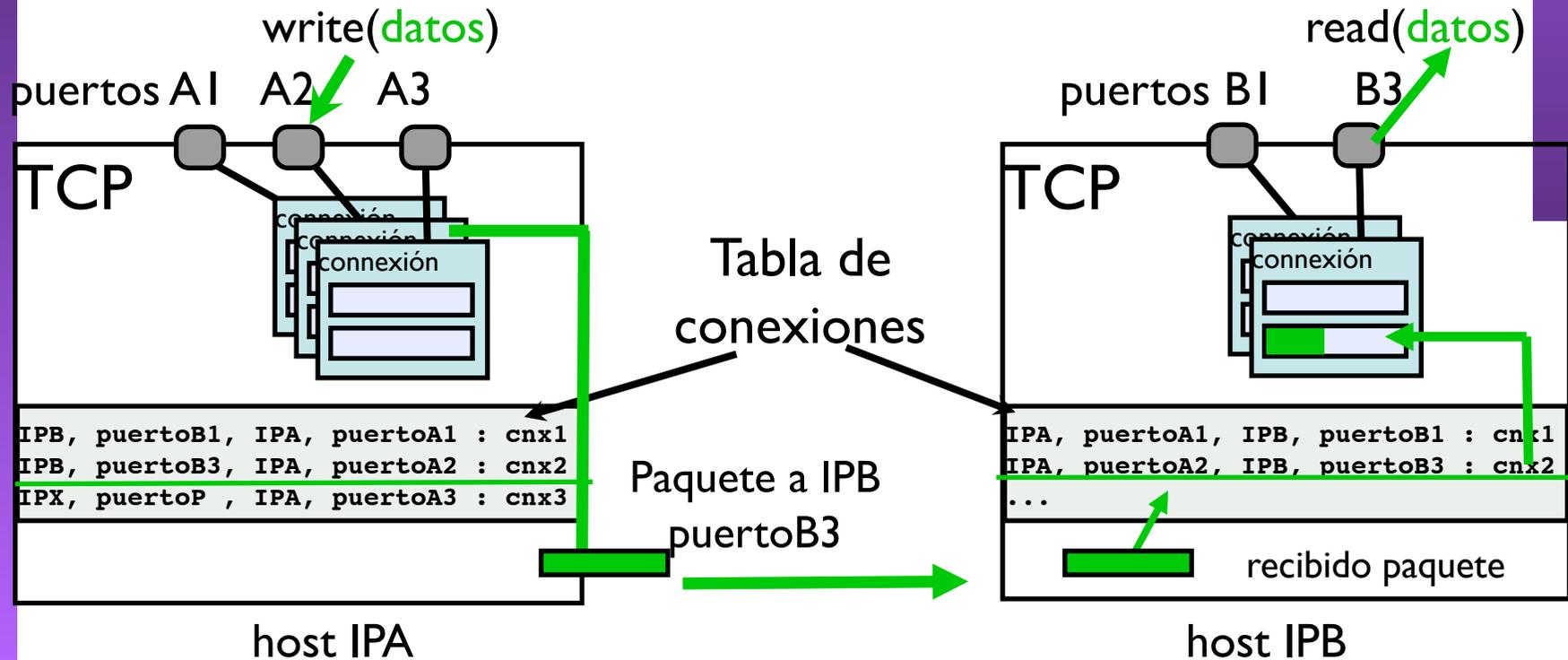
# TCP

- Interfaz con el nivel de aplicación
  - Tras establecer una conexión proporciona un stream bidireccional entre sockets
  - Sin fronteras entre mensajes
  - 2 buffers por conexión
    - Escribir en el socket pone los datos en buffer de envío
    - Buffer de recepción para esperar el read()



# TCP

- Demultiplexación de datos que llegan a TCP:
  - Se identifica al socket destino por la tupla ( IP origen, puerto origen, IP destino, puerto destino )
  - La tabla de tuplas (ip,puerto,ip,puerto) con sus sockets de un nivel TCP es la tabla de conexiones.
- La conexión sólo existe en los extremos TCP

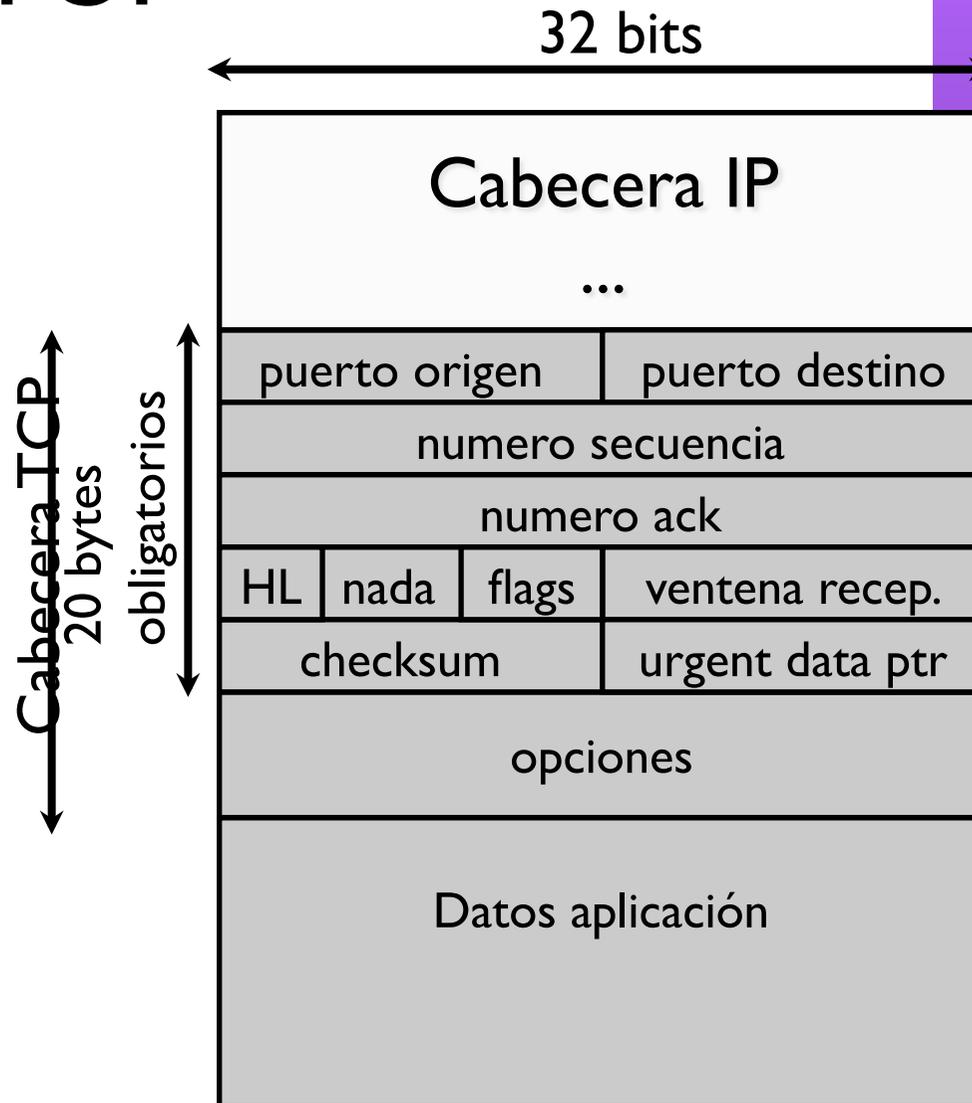


# TCP

- Los buffers aíslan a TCP de las operaciones del usuario.
  - TCP hará lo posible por enviar los datos cuando pueda
  - TCP colocara los datos en el buffer de recepción cuando lleguen
- Para realizar esto TCP necesitara un conjunto de mensajes para comunicarse con el TCP del otro lado
  - Mensajes de establecimiento y cierre de conexión
  - Mensajes de datos
  - Mensajes con ACKs
- Veamos los mensajes del protocolo TCP

- Segmento TCP
- Cabecera de tamaño variable
  - 20 hasta 60 bytes según las opciones
- Datos del nivel de aplicación

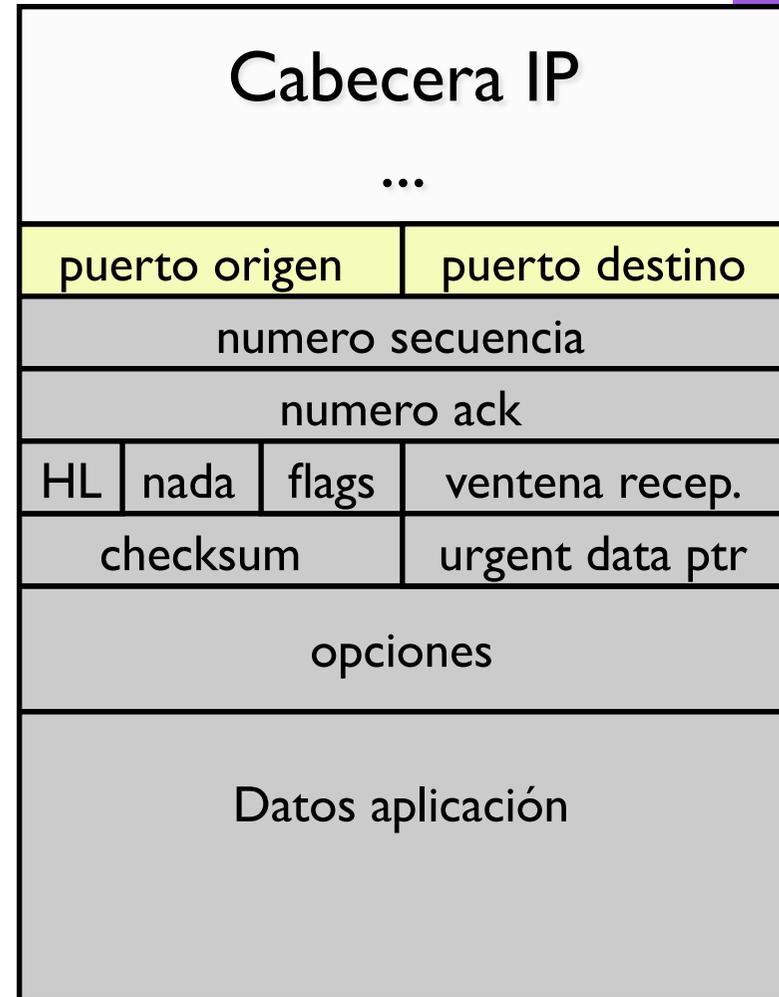
# TCP



# TCP

## Contenido

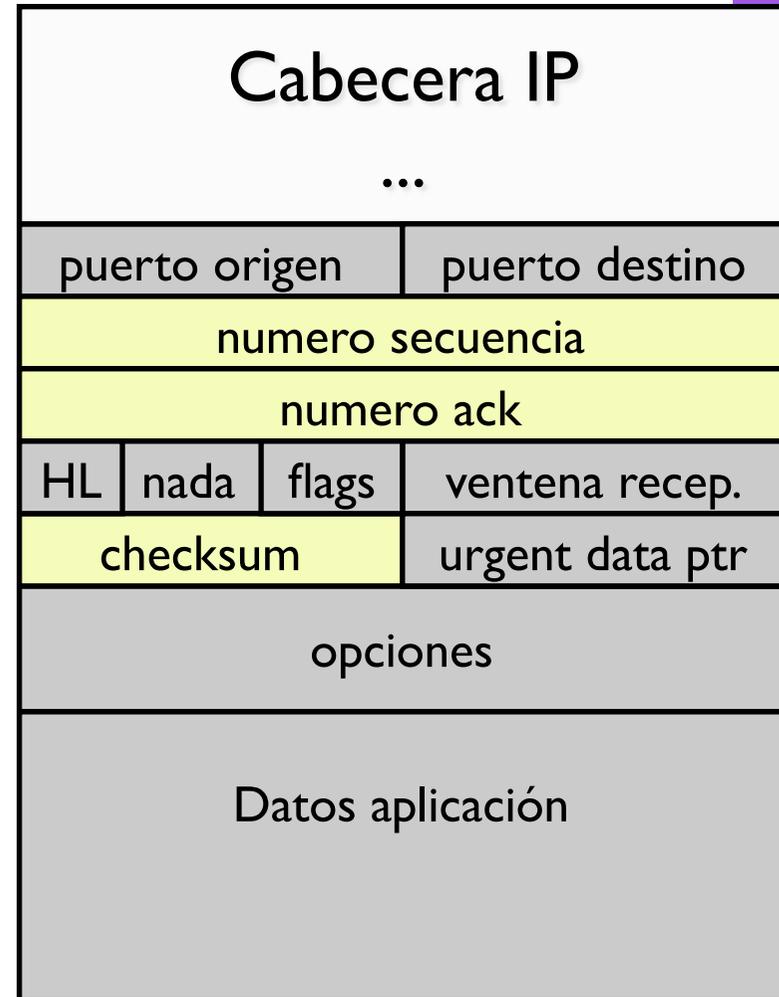
- Datos de multiplexación
  - Puerto origen
  - Puerto destino



# TCP

## Contenido

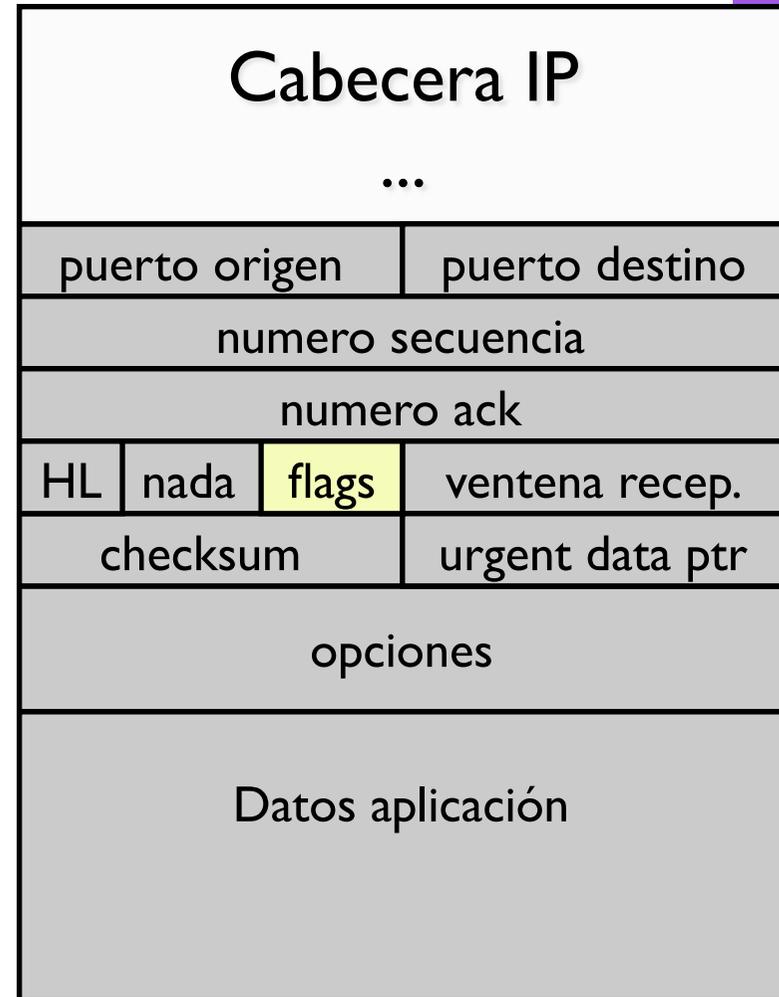
- Datos para transporte fiable
  - Número de secuencia
  - Número de ACK
  - Checksum
 Cabecera + datos de aplicación + algunos datos de IP (pseudo cabecera como en UDP)
- En un mismo paquete podemos mandar datos y confirmar datos del sentido contrario



# TCP

## Contenido

- FLAGs: diferentes tipos de paquetes del protocolo
  - URG urgente
  - ACK acknowledgement
  - PSH push
  - RST reset
  - SYN syn
  - FIN fin

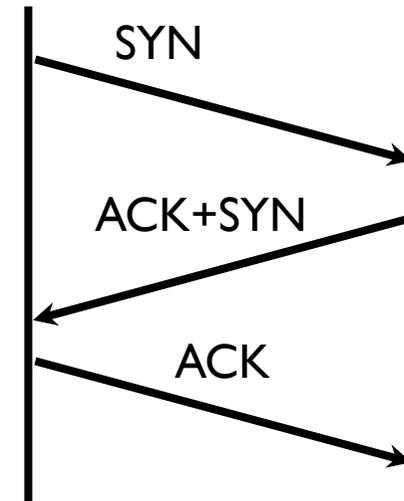


# TCP: conexiones

- TCP es orientado a conexión
- Previamente a comunicarse datos entre un emisor y un receptor deben negociar un establecimiento de conexión.
  - TCP inicializa sus variables para la conexión y crea los buffers
  - Esto se hace mediante los paquetes que utilizan los flags SYN, FIN y RST
  - Protocolo para establecer la conexión
  - Protocolo para liberar la conexión

# TCP: establecimiento de conexión

- Mecanismo: Three way handshake
  - Lado cliente (socket que hace connect)
  - envía un paquete sin datos con el flag **SYN**
  - Establece el numero de secuencia inicial
  - Lado servidor (socket que hace accept)
  - responde con un paquete sin datos con **ACK y SYN**
  - Establece el numero de secuencia inicial
  - Lado cliente confirma este paquete con un **ACK**
  - Este paquete ya puede llevar datos
  - Al recibir el ACK el servidor puede enviar ya datos
  - Los SYNs gastan un número de secuencia para poder confirmarse con ACKs



# Ejemplo

- Observando una conexión web...

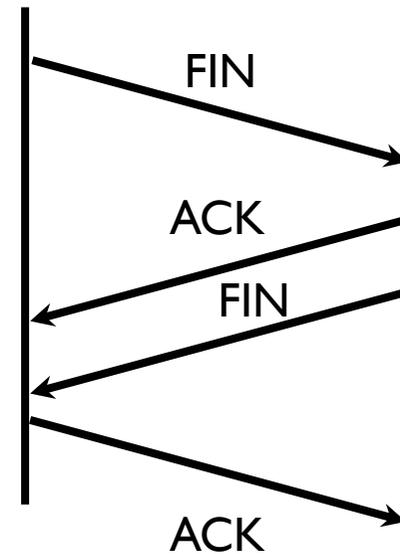
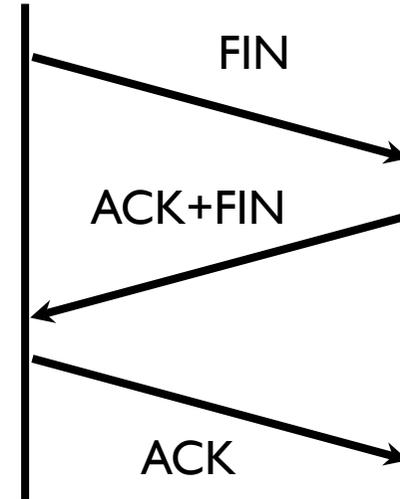
Los SYNs usan un número de secuencia para poder ser confirmados

```
IP ...177.53656 > ...105.80: S 3482203897:3482203897(0) win 65535 SYN
IP ...105.80 > ...177.53656: S 3356369201:3356369201(0) ack 3482203898 win 24616 SYN+ACK
IP ...177.53656 > ...105.80: . ack 3356369202 win 65535 ACK
IP ...177.53656 > ...105.80: P 3482203898:3482204138(240) ack 3356369202 win 65535
IP ...105.80 > ...177.53656: . ack 3482204138 win 24616
IP ...105.80 > ...177.53656: P 3356369202:3356369502(300) ack 3482204138 win 24616
IP ...105.80 > ...177.53656: . 3356369502:3356370950(1448) ack 3482204138 win 24616
IP ...105.80 > ...177.53656: P 3356370950:3356372398(1448) ack 3482204138 win 24616
```

Aqui empieza la transferencia  
Paquete 4

# Cierre de la conexión

- Cualquiera de los dos extremos puede iniciarlo
  - Envía un paquete sin datos con el flag **FIN**. Consume también un número de secuencia
  - El otro extremo, confirma enviando un **ACK** e indica que cierra también con otro **FIN**. Este segundo **FIN** puede ir en el mismo paquete o en otro.
  - El extremo original confirma con un **ACK**



# Ejemplo

- El final de una conexión web...

El servidor está enviando datos

El cliente decide cerrar y manda un FIN

...

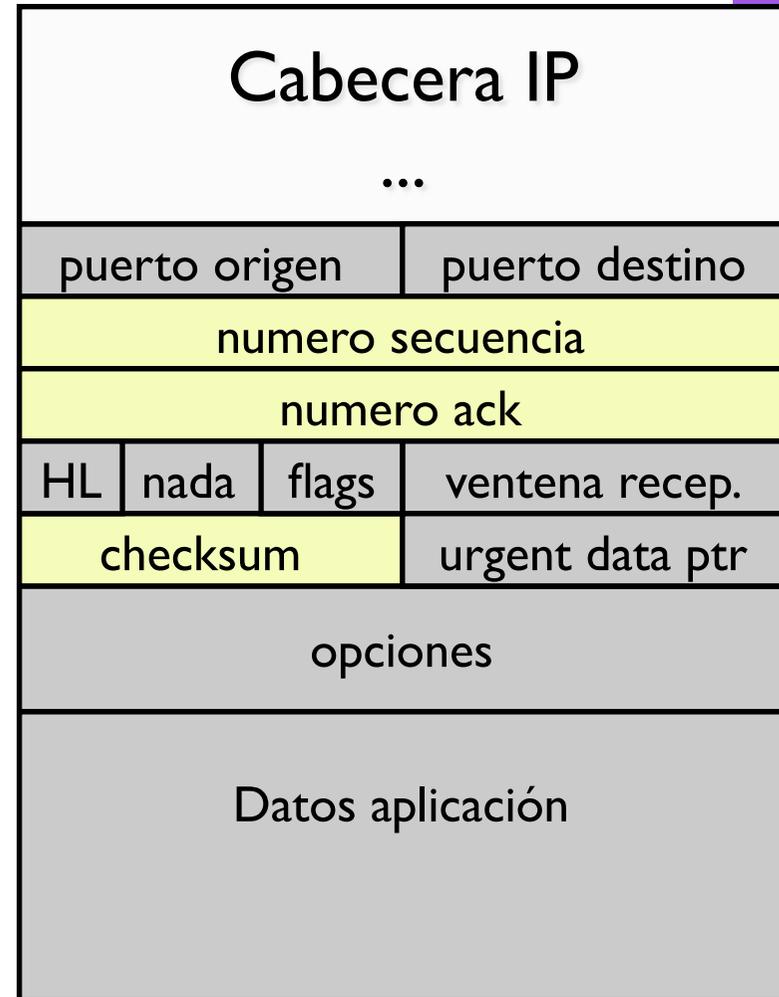
```
IP 130.206.166.105.80 > 130.206.169.177.53701: P 80314174:80315622(1448) ack 4067364561 win 24616
IP 130.206.166.105.80 > 130.206.169.177.53701: P 80315622:80316551(929) ack 4067364561 win 24616
IP 130.206.169.177.53701 > 130.206.166.105.80: . ack 80316551 win 65535
IP 130.206.169.177.53701 > 130.206.166.105.80: F 4067364561:4067364561(0) ack 80316551 win 65535
IP 130.206.166.105.80 > 130.206.169.177.53701: . ack 4067364562 win 24616
IP 130.206.166.105.80 > 130.206.169.177.53701: F 80316551:80316551(0) ack 4067364562 win 24616
IP 130.206.169.177.53701 > 130.206.166.105.80: . ack 80316552 win 65535
```

El servidor cierra su sentido

# TCP

## Contenido

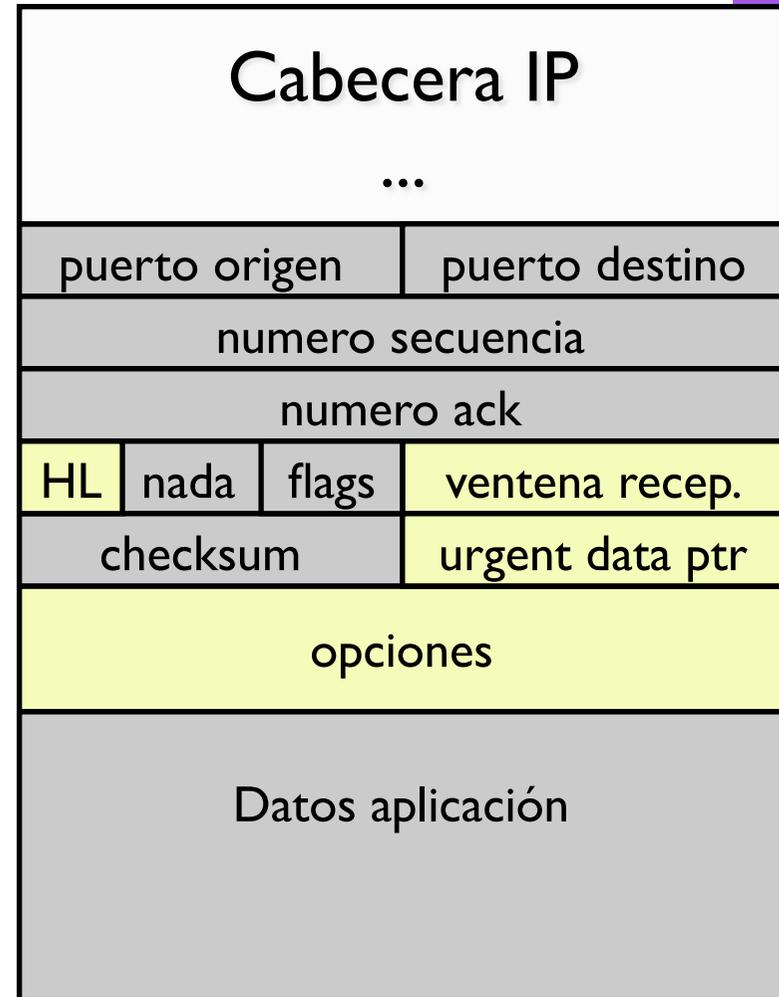
- Datos para transporte fiable
  - Número de secuencia
  - Número de ACK
  - Checksum
 Cabecera + datos de aplicación + algunos datos de IP (pseudo cabecera como en UDP)
- En un mismo paquete podemos mandar datos y confirmar datos del sentido contrario



# TCP

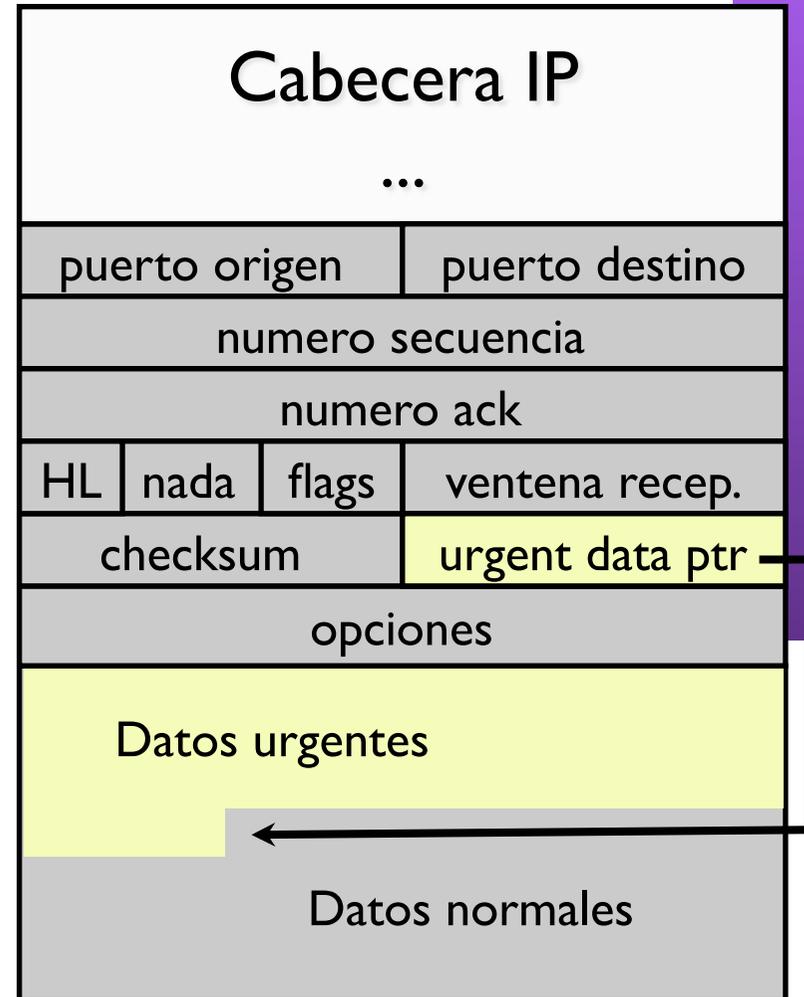
## Contenido

- Ventana de recepción
- Datos urgentes
- HL (header length)
  - Tamaño de la cabecera (en palabras de 4 bytes)
  - 4 bits de de 5 a 15 palabras de 20 a 60 bytes
- Opciones extras



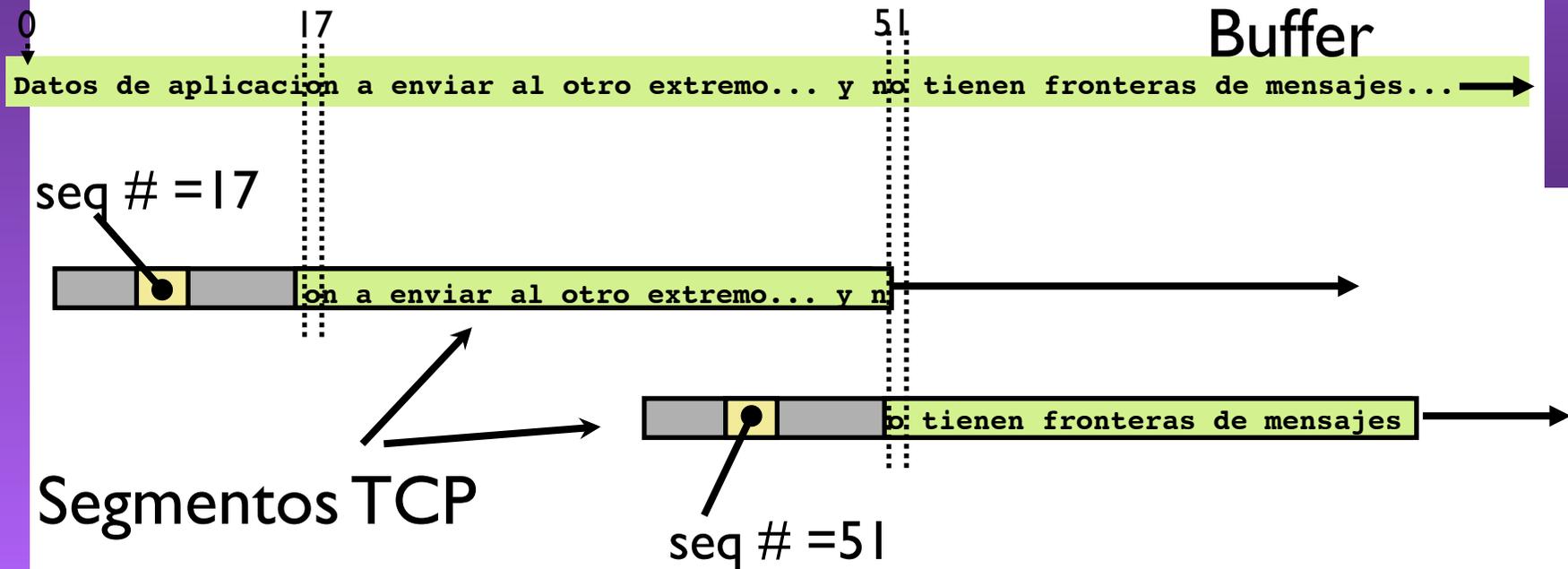
# Datos urgentes

- Si URG está activado.
  - El paquete lleva datos urgentes.  
Canal de datos Out-of-band
  - El puntero urgente indica donde acaban los datos urgentes
  - Los datos normales se entregan normalmente en el buffer para la aplicación
  - Los datos urgentes se entregan aparte
- No se usa mucho



# TCP: envío de datos

- Los bytes a enviar se colocan en el buffer y forman una corriente de bytes sin fronteras de paquetes
- TCP envía los datos en paquetes de un tamaño determinado por la variable MSS (Maximum Segment Size) que se negocia en el establecimiento de conexión
- El número de secuencia (y el número de ACK) hacen referencia al primer byte del paquete en la secuencia global

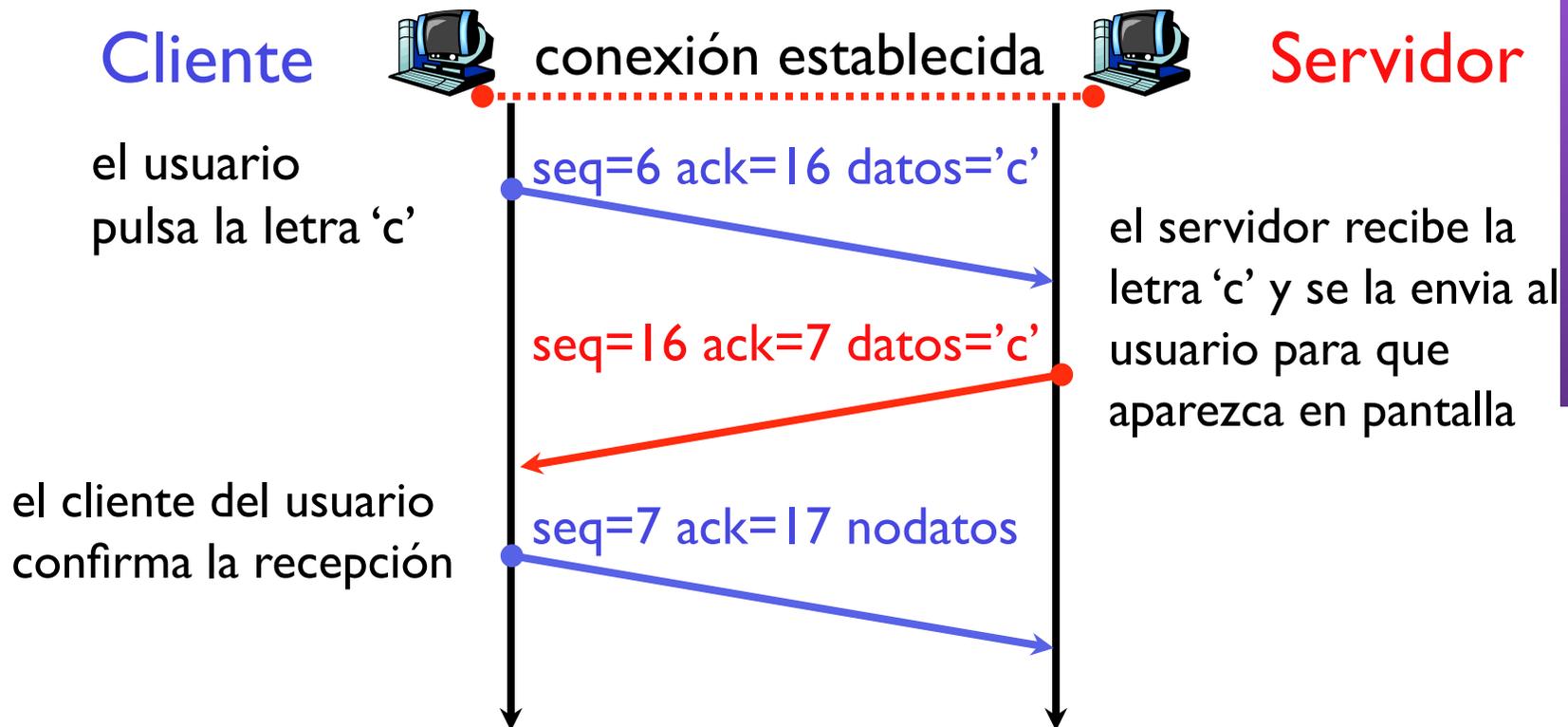


# TCP: envío de datos

- Secuencia y ACK: campos de 32 bits
  - 4 GB de datos antes de dar la vuelta
  - La secuencia no empieza de 0 sino que se genera al azar al principio de cada conexión y para cada sentido
- El campo ACK
  - es valido si esta activado el flag ACK
  - indica la próxima secuencia que el receptor espera recibir
  - cumulative ACK: tipo Go back N a nivel de byte
- Si una conexión está transmitiendo en ambos sentidos los ACKs de un sentido van en los paquetes de datos del opuesto piggyback

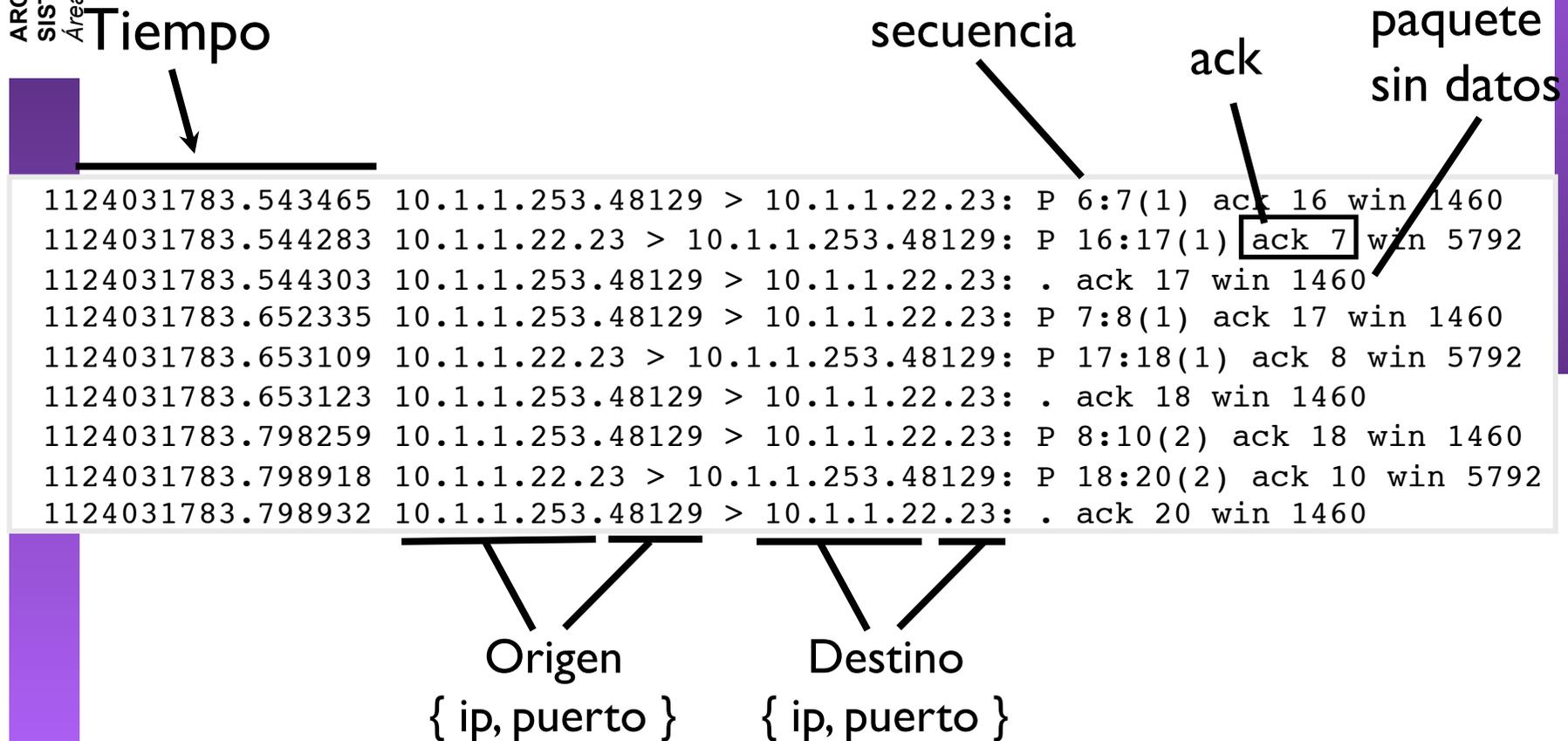
# Ejemplo

- Paquetes de un telnet desde 10.1.1.253 a 10.1.1.22



# Ejemplo

- Paquetes de un telnet desde 10.1.1.253 a 10.1.1.22
- Usando tcpdump para ver los paquetes



# TCP: transporte fiable

- TCP utiliza una ventana deslizante
  - Número de secuencia: el primer byte enviado en el segmento
  - ACK: el próximo byte que espera recibir el receptor
  - Máximo de la ventana permitida de recepción indicada en el campo window (cuantos bytes puedo enviar sin recibir ACK)
- Los paquetes TCP llevan
  - Número de secuencia de los datos. Si no llevan datos, el campo número de secuencia indica el próximo número de secuencia que se enviará
  - Próximo número de secuencia que espera recibir su emisor. Es válido si el byte ACK está activado (o sea todos salvo en el SYN inicial)
  - Los números de secuencia son independientes en ambos sentidos
- Transmisiones simultáneas en los dos sentidos  
Cada extremo funciona como un emisor y un receptor independientes

# TCP: emisor

## Eventos en el emisor

Llegan datos desde el nivel de aplicación

- ▶ crear segmento nuevo
- ▶ sec # el siguiente de la stream
- ▶ iniciar temporizador si no hay uno iniciado

timeout

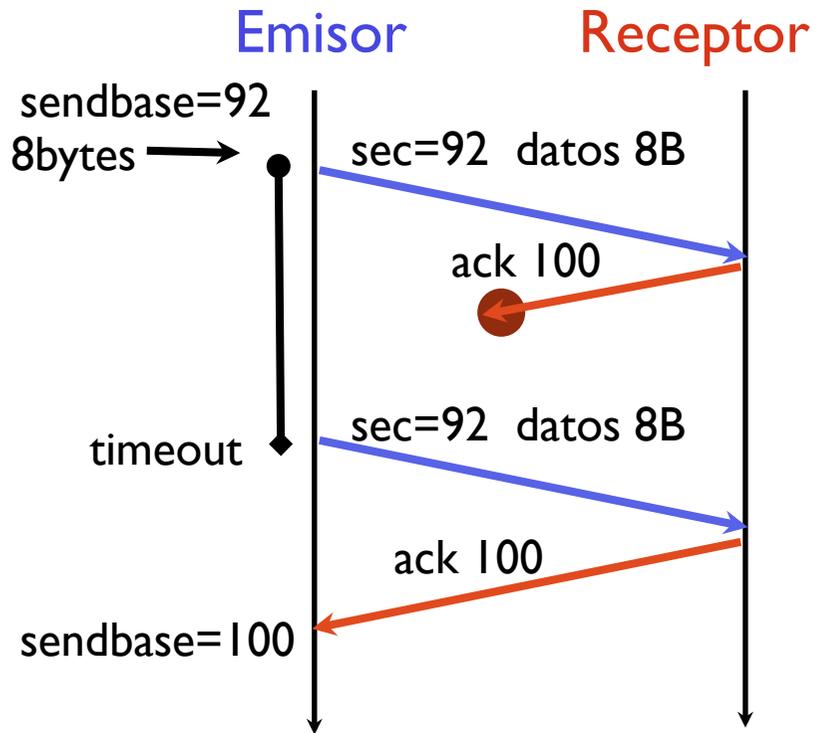
- ▶ retransmitir el segmento que causó el timeout
- ▶ reiniciar timeout

recibido ACK

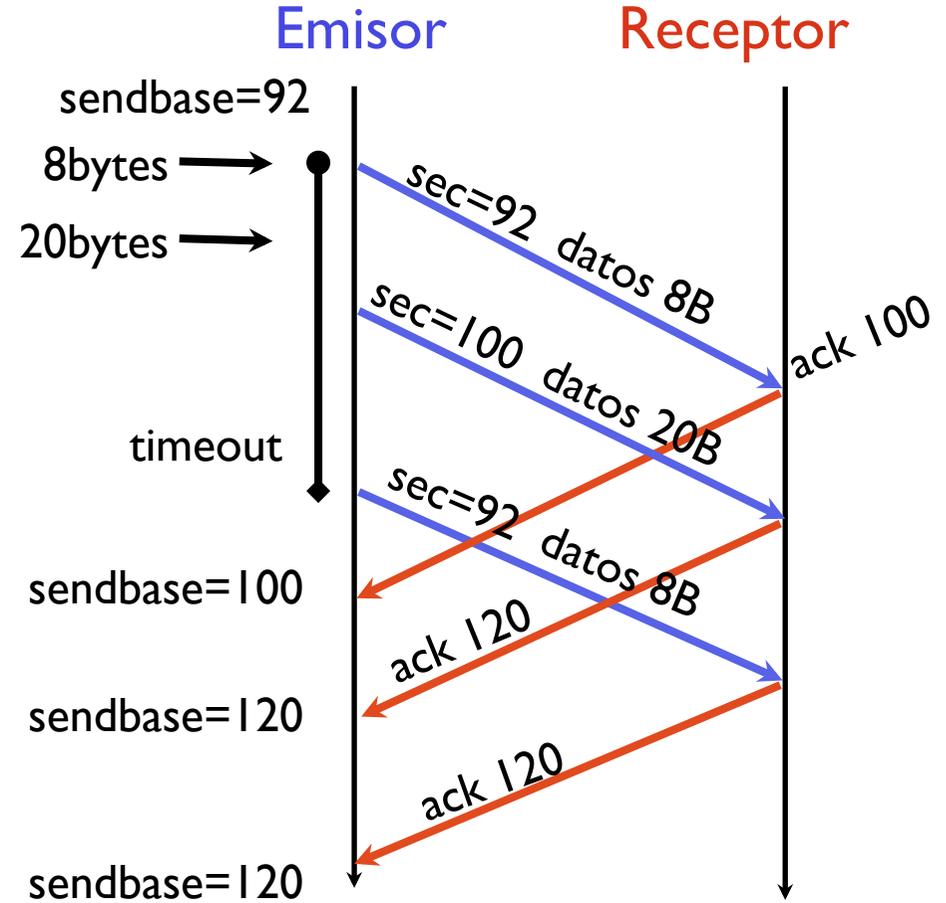
- ▶ Si confirme un segmento nuevo
- > actualizar ventana cumulative ACK
- > reiniciar timeout si quedan segmentos por confirmar

Parece de tipo Go back-N

# Ejemplos

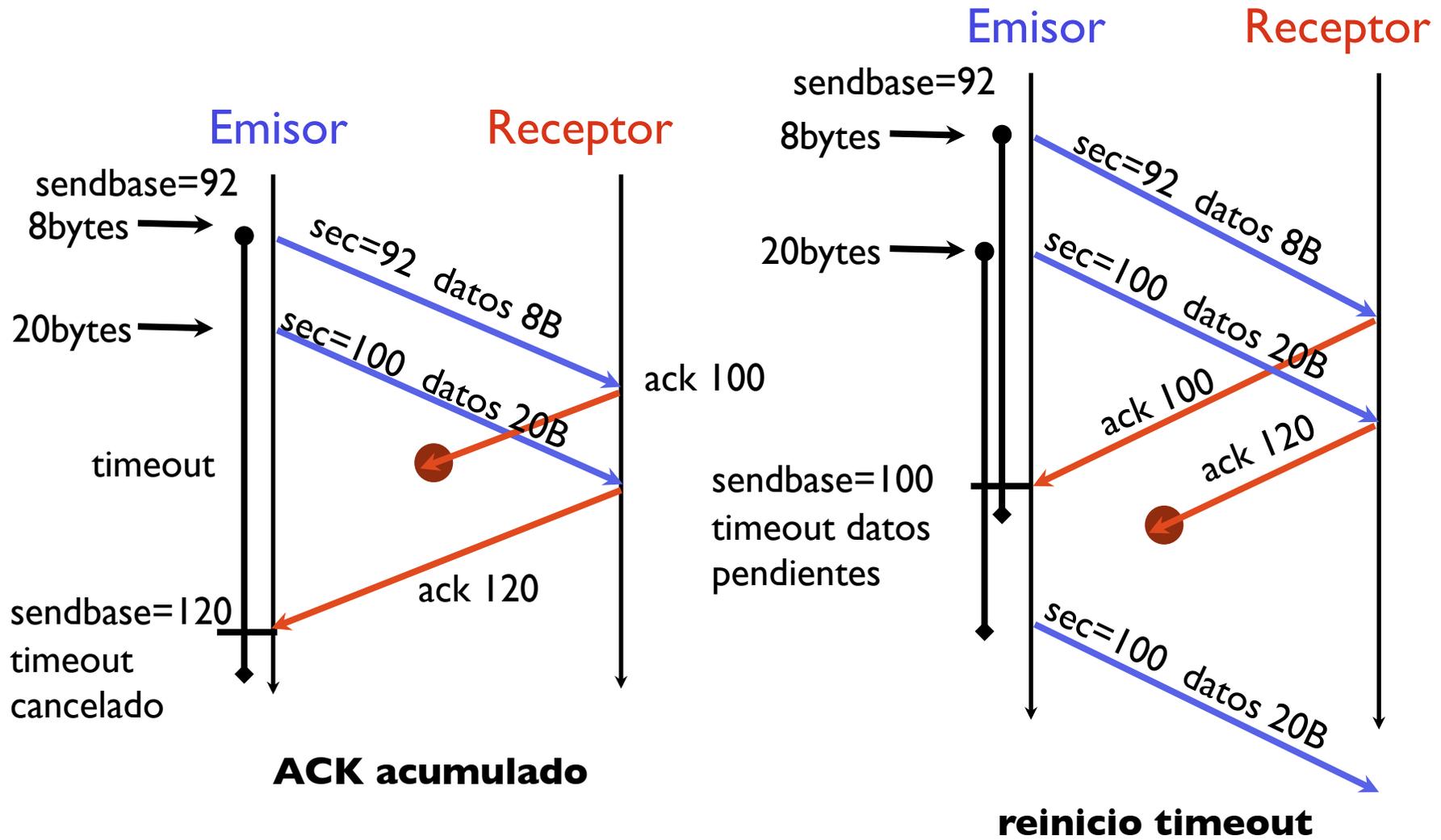


**pérdida de ACK**



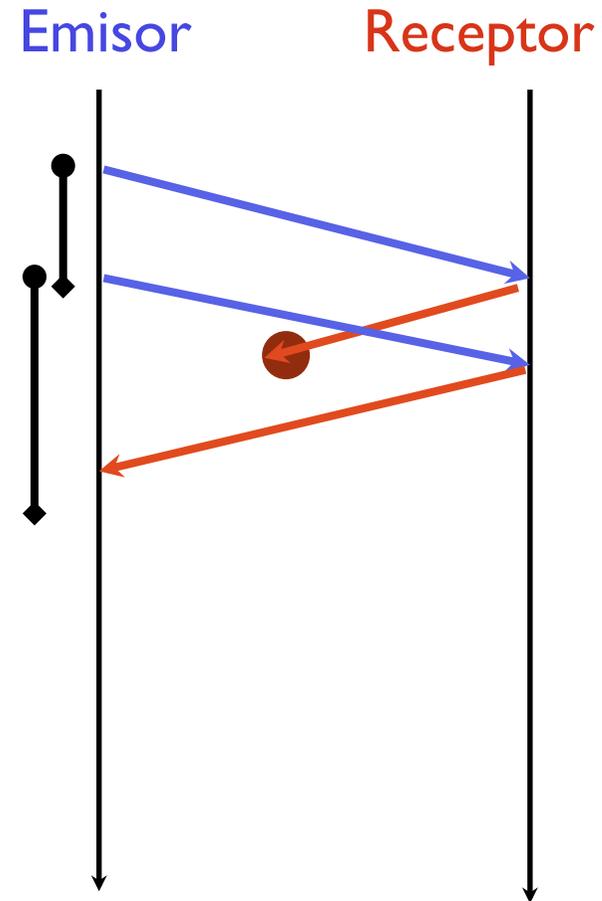
**timeout prematuro**

# Ejemplos



# TCP: timeout

- El RTT se va estimando observando los tiempos que tardan en llegar los ACKs y se elige un timeout mayor que el RTT estimado
- El timeout se dobla cada vez que caduca y se envía de nuevo el paquete
- El timeout que usamos va acercandose al RTT que observamos, y si hay errores sube bruscamente



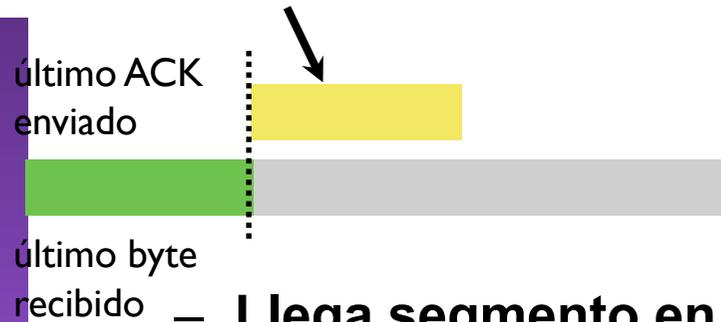
# TCP: transporte fiable (resumen)

- Emisor de tipo Go back-N
  - ACKs acumulados por bytes individuales
  - Timeout adaptativo estimando el RTT
  - Ventana indicada por el receptor en cada paquete, en lugar de ser un N fijo
- El receptor es un poco más complicado
- El emisor es también más complicado en realidad

# TCP: receptor

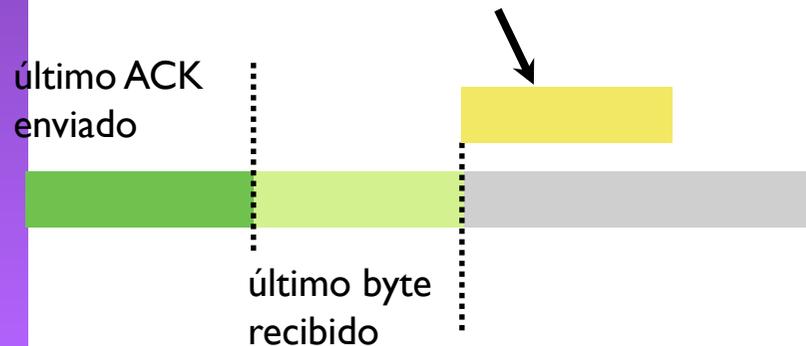
## Eventos del receptor

- **Llega segmento en orden con el numero de secuencia esperado**  
No hay ACKs pendientes de enviar



Acción: **Delayed ACK**, espera hasta 500ms al siguiente paquete, si no llega manda ACK

- **Llega segmento en orden con el numero de secuencia esperado**  
Hay un delayed ACK pendiente

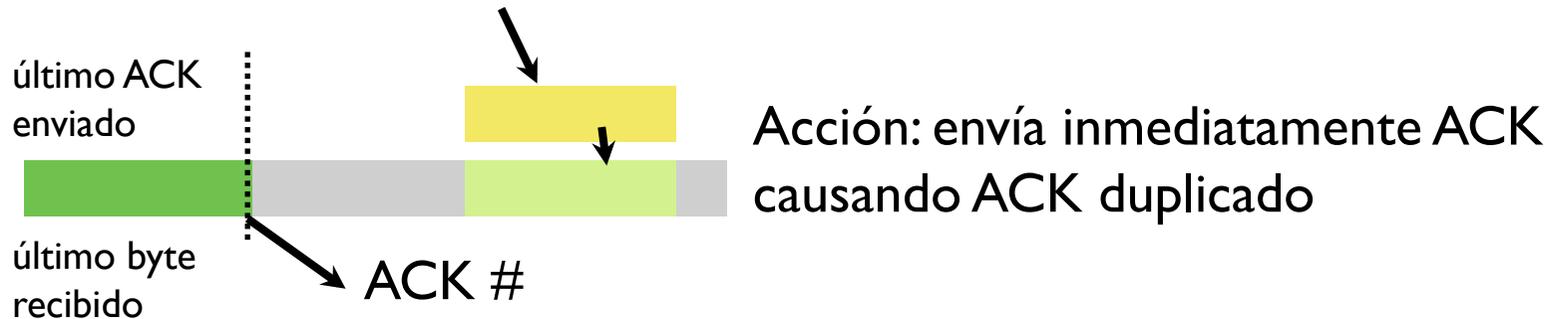


Acción: envía inmediatamente ACK (al ser acumulado confirma los dos)

# TCP: receptor

## Eventos del receptor

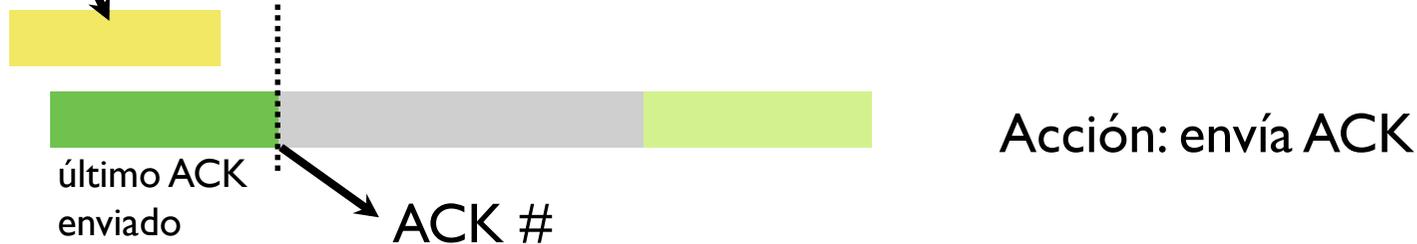
- Llega segmento fuera de orden generando hueco



- Llega segmento rellenando hueco

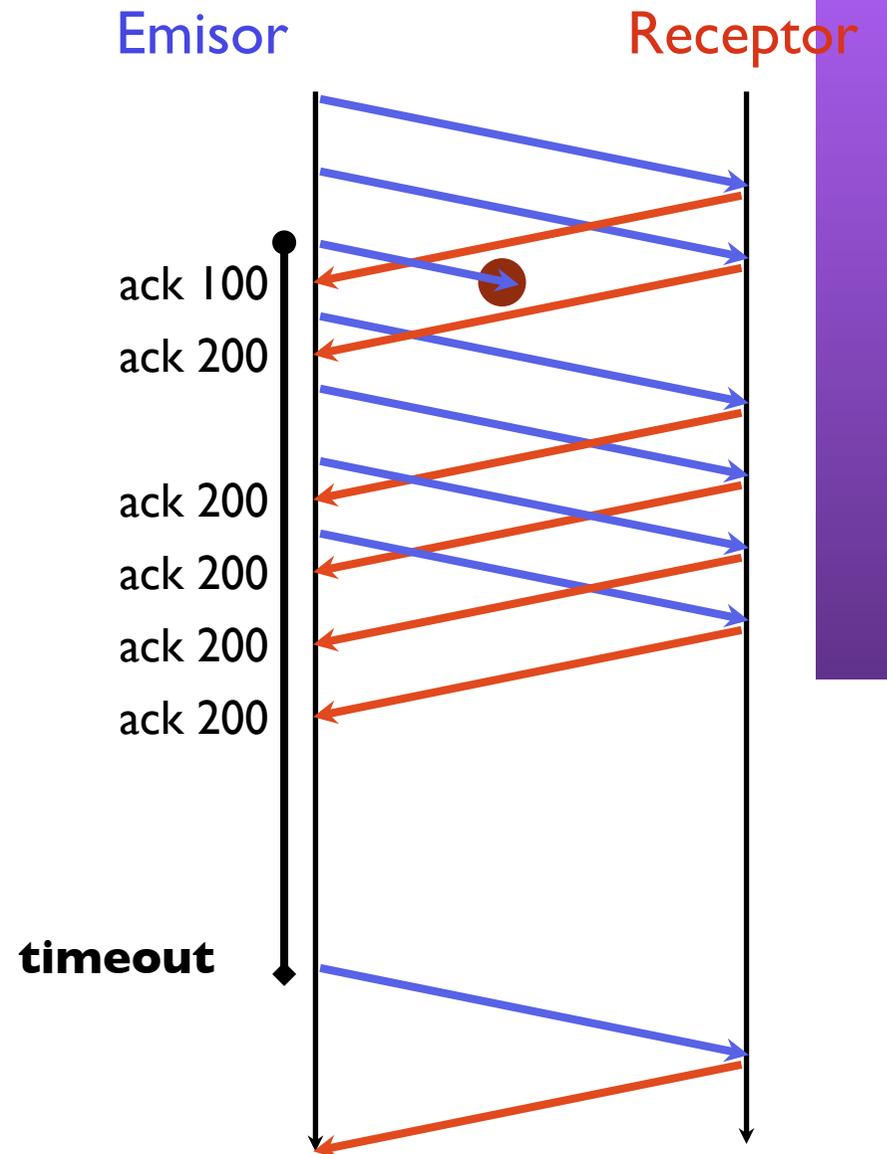


- Llega segmento anterior a la ventana



# TCP: Fast retransmit

- El timeout normalmente es relativamente largo
  - Si se pierde un paquete de datos se genera hueco y se detendrá la transmisión durante un timeout
  - Normalmente el emisor envía varios paquetes seguidos
- El receptor no puede hacer un NACK pero está generando ACKs duplicados !!



# TCP: Fast retransmit

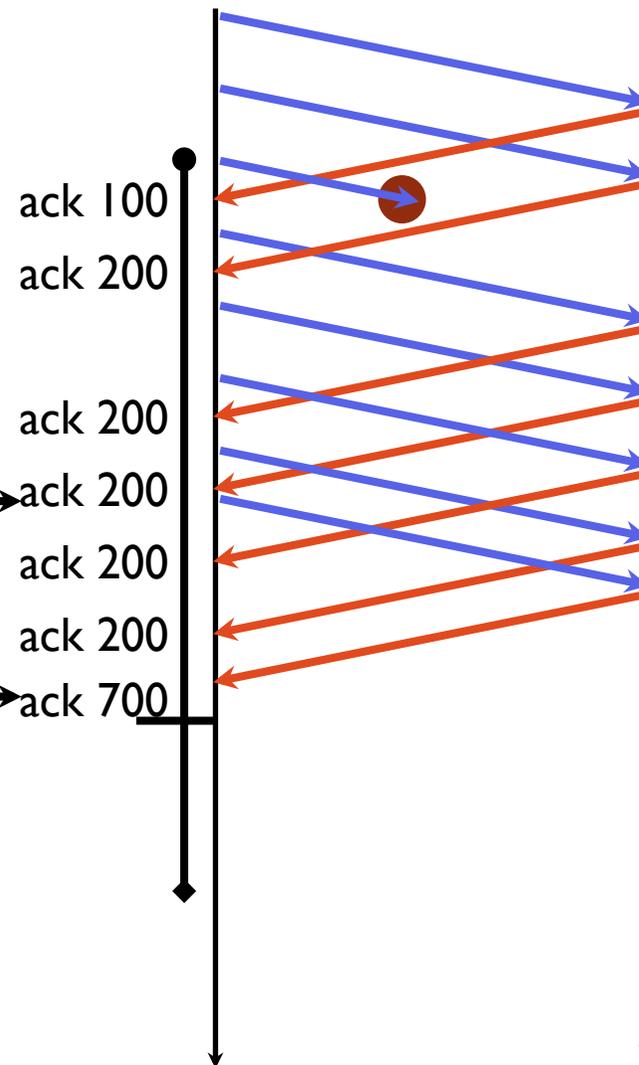
- Fast retransmit
  - Si el emisor recibe 3 ACKs con el mismo numero de ACK supondrá que se ha perdido el paquete que llevaba ese numero de secuencia
  - Reenvia el paquete inmediatamente sin esperar a que caduque el timeout

**3 dup ACKs !!!** →  
**= fast retransmit**

**recibidos todos** →  
**timeout cancelado**

Emisor

Receptor



# TCP: transporte fiable (resumen)

- Características de Go back-N y SR
  - ACKs acumulados (y por byte individual)
  - Pero almacena paquetes fuera de secuencia en recepción
- Aunque no envía ACKs por paquete
- Eso permite técnicas más sofisticadas de retransmisión al detectar duplicados (Fast retransmit)

# Control de flujo y congestión

- Pero no siempre interesa enviar lo mas rápido posible...  
Por que?
  - Porque no queremos saturar al receptor. Solución enviar feedback hacia el emisor controlando su velocidad  
El problema del **control de flujo**
  - Porque no queremos saturar la red que está siendo usada por otros programas/usuarios también. Solución adaptarse al estado de la red... pero adaptarse de forma justa (hay otros emisores, quien tiene más derecho a usar la red? **Network neutrality?**)  
El problema del **control de congestión**
- El control de flujo es un problema sencillo. El control de congestión es un problema difícil y es uno de los problemas fundamentales de redes
- TCP incluye mecanismos de control de flujo y congestión
  - Estos mecanismos disminuyen la velocidad de envío de TCP según las condiciones del destino y la red
  - Esta es la razón por la que a veces preferimos UDP

# Conclusiones

- TCP es el protocolo de transporte fiable de Internet
- El transporte fiable de TCP se basa en:
  - Ventana deslizante con ACKs acumulados
  - Retransmisiones por timeout con timeout adaptativo basado en estimación de RTT
    - Mas mecanismos de retransmision más sofisticados con delayed ACKs y Fast Retransmit
- Falta
  - Control de flujo: asegurandose de que no enviamos tan rapido como para perjudicar al receptor (lecturas para la proxima clase)
  - Control de congestión: asegurándose de que no enviamos tan rápido que perjudiquemos a otros usuarios de la red

## Próxima clase:

- Control de flujo y prestaciones de TCP
- Problemas