

Redes de Computadores
Nivel de Transporte:
TCP

Área de Ingeniería Telemática
Dpto. Automática y Computación
<http://www.tlm.unavarra.es/>

En clases anteriores...

- ▶ El nivel de transporte, principios
- ▶ UDP: nivel de transporte no orientado a conexión de Internet

En esta clase

- ▶ **TCP: nivel de transporte orientado a conexión y fiable de Internet**

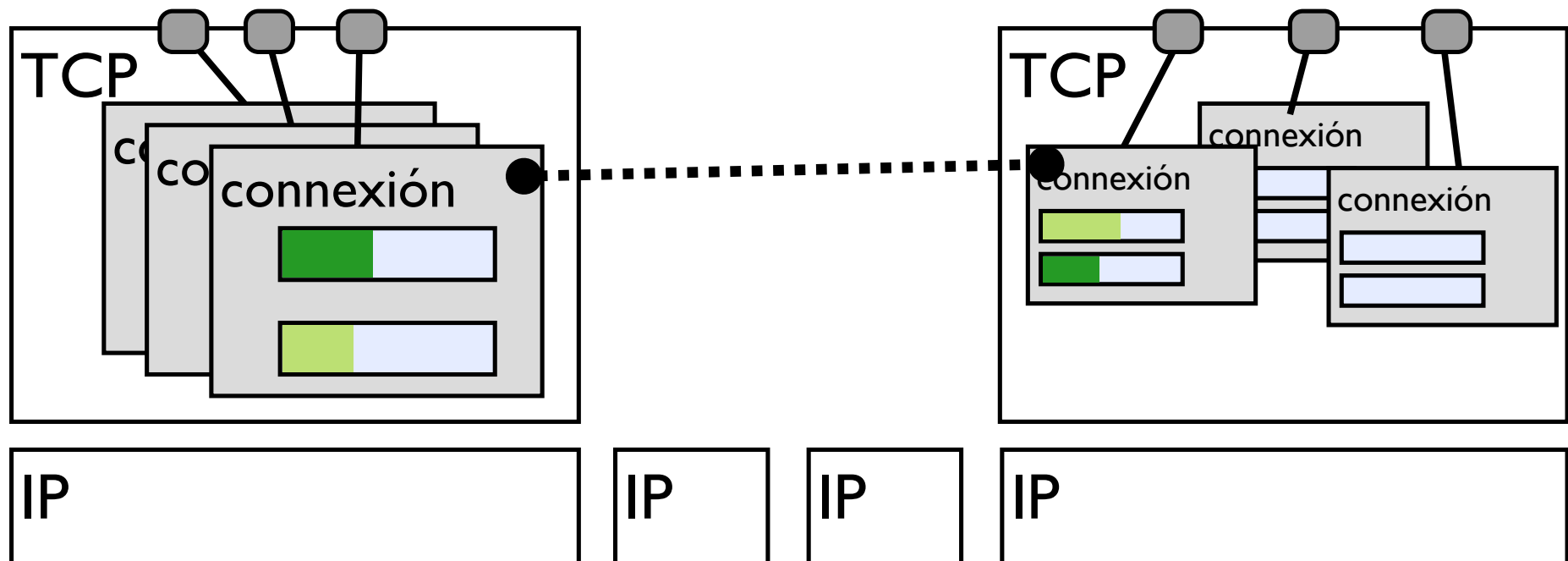
TCP

- ▶ **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

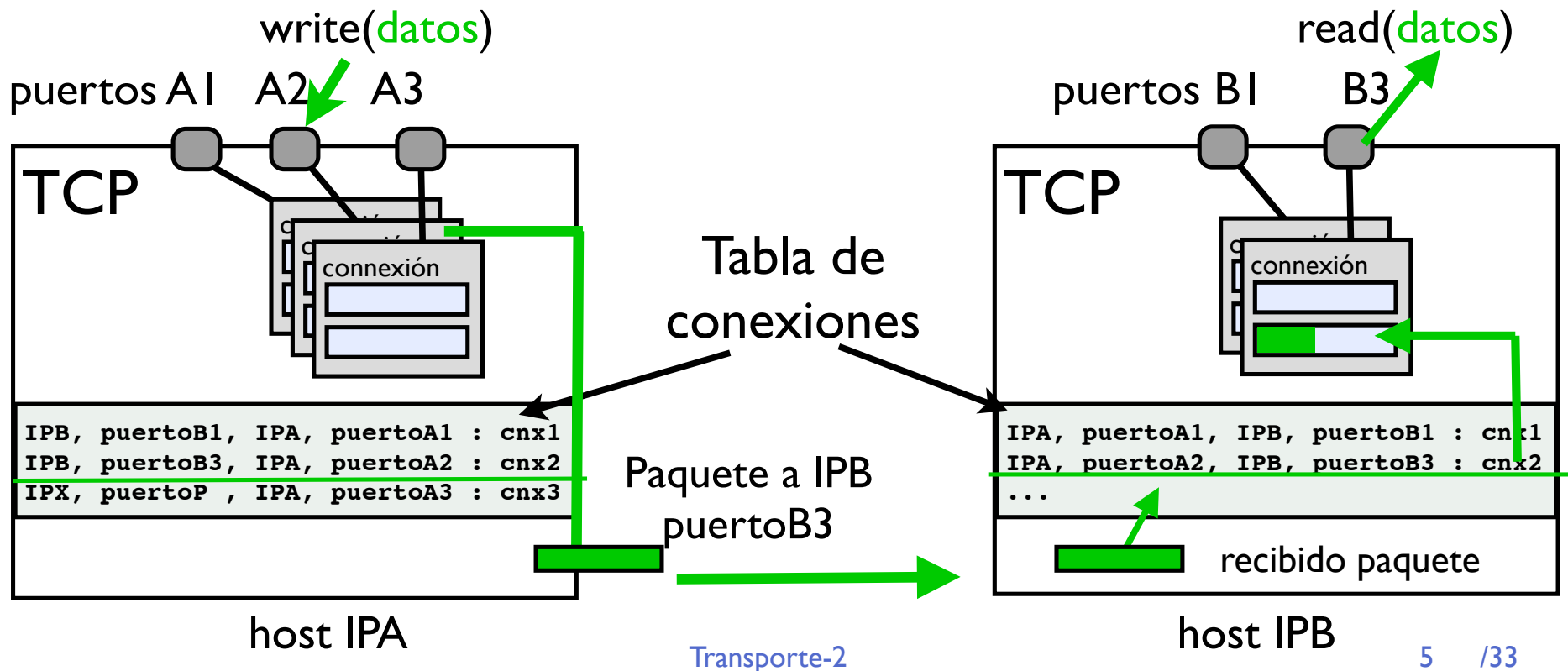
- ▶ Demultiplexación de datos que llegan a TCP:

- > 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

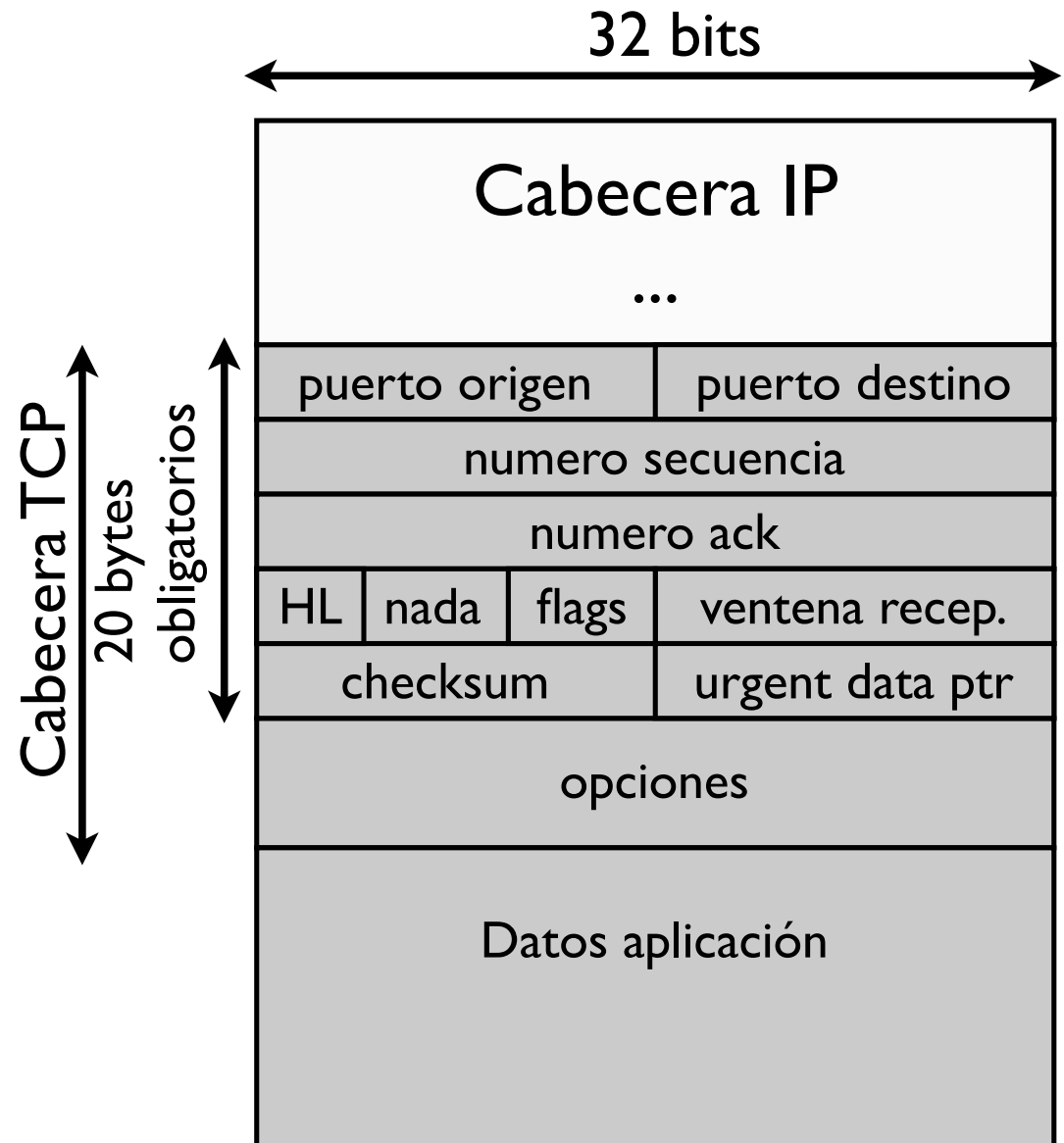


TCP

- ▶ **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 para confirmar los datos y obtener transporte fiable
- ▶ **Veamos los mensajes del protocolo TCP**

TCP

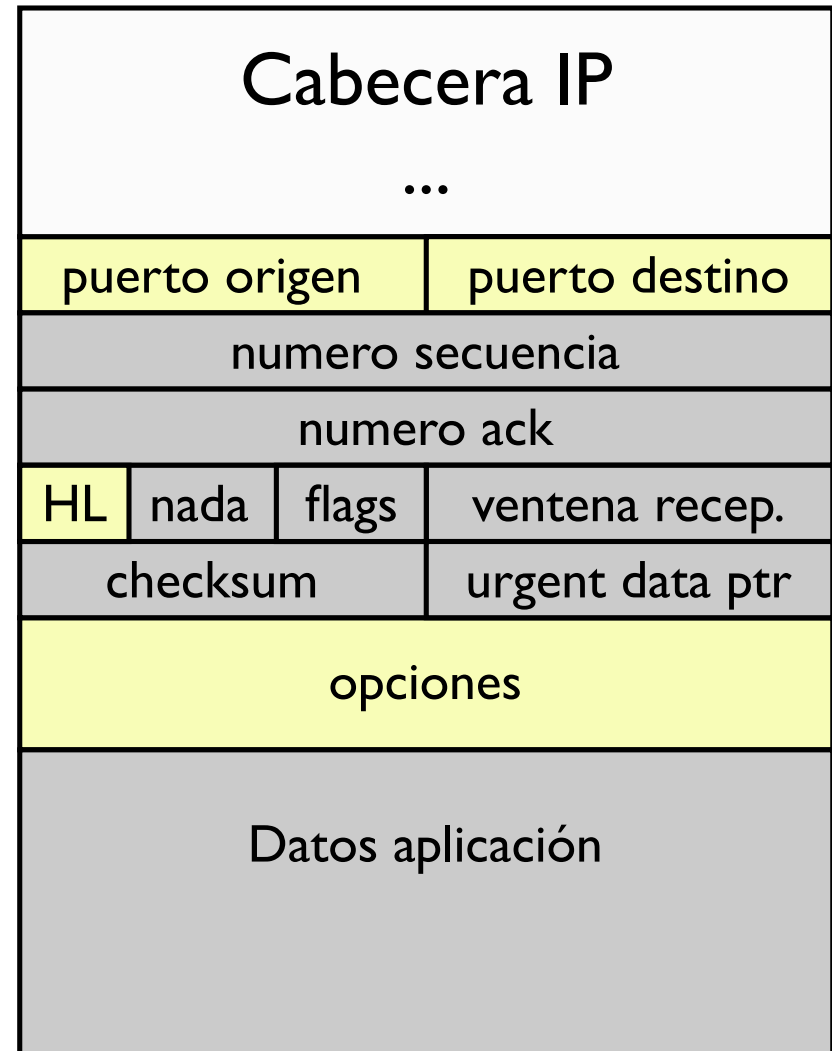
- ▶ Segmento TCP
- ▶ Cabecera de tamaño variable
 - > 20 hasta 60 bytes según las opciones
- ▶ Datos del nivel de aplicación



TCP

Contenido

- ▶ Datos de multiplexación
 - > Puerto origen
 - > Puerto destino
- ▶ 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

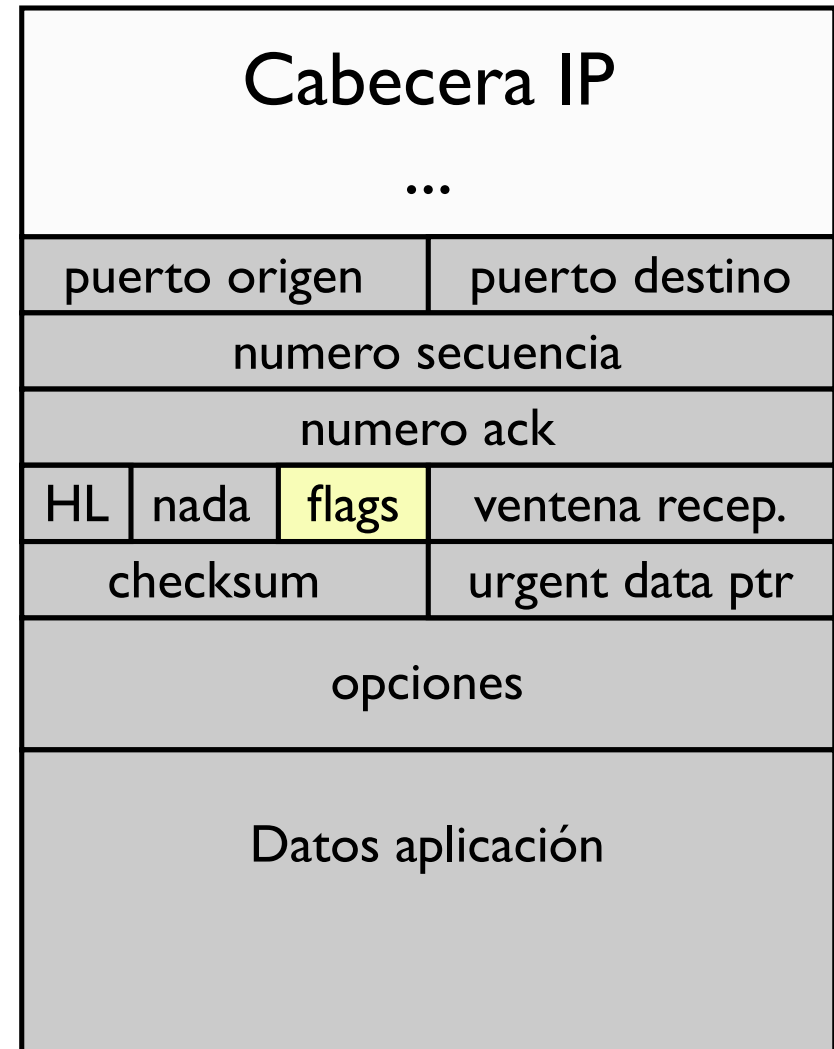


TCP

Contenido

▶ FLAGS: diferentes tipos de paquetes del protocolo

- > URG urgente
- > ACK acknowledgement
- > PSH push
- > RST reset
- > SYN syn
- > FIN fin



TCP

Contenido

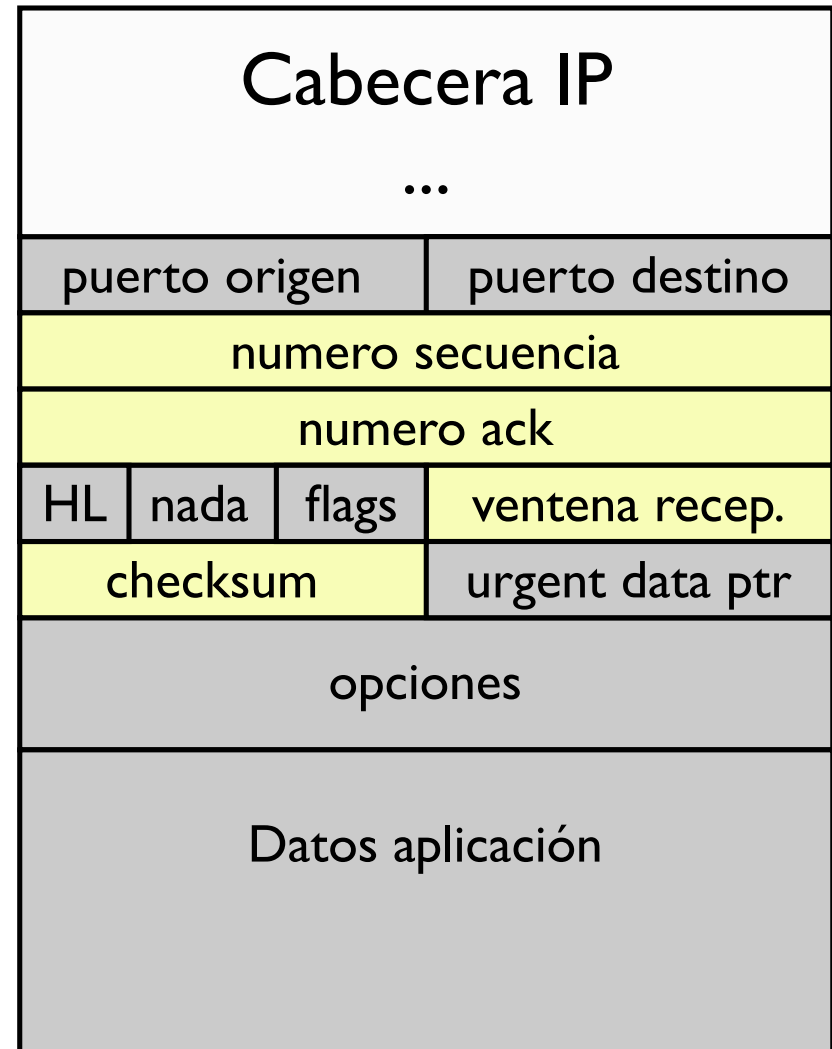
▶ Datos para transporte fiable

- > Número de secuencia
- > Número de ACK
- > Checksum

Cabecera + datos de aplicación + algunos datos de IP (pseudo cabecera como en UDP)

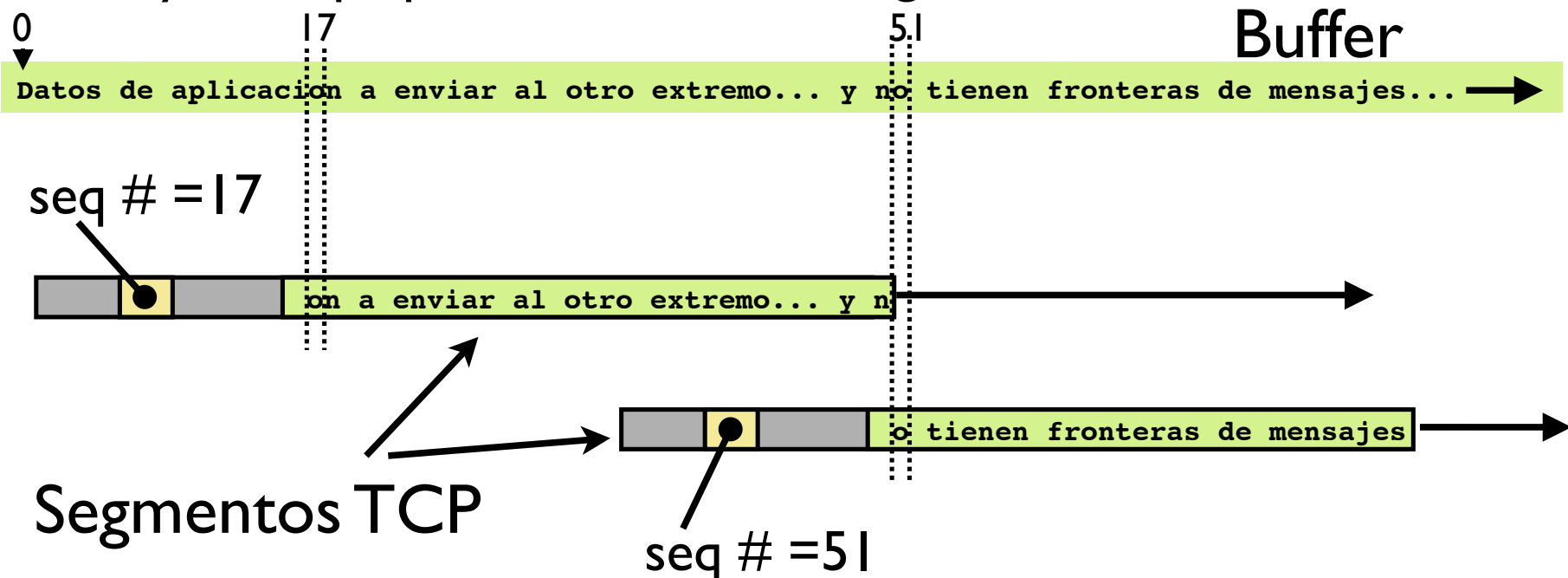
▶ Control de flujo

- > ventana de recepción
permite al otro extremo mandarme una cantidad de bytes sin recibir confirmación



TCP: envío de datos

- ▶ Los bytes a enviar se colocan en el buffer y forman una corriente de bytes sin fronteras de paquetes
- ▶ El número de secuencia (y el número de ACK) hacen referencia al byte concreto
 - > el número de secuencia indica cual es el número del primer byte del paquete en la secuencia global

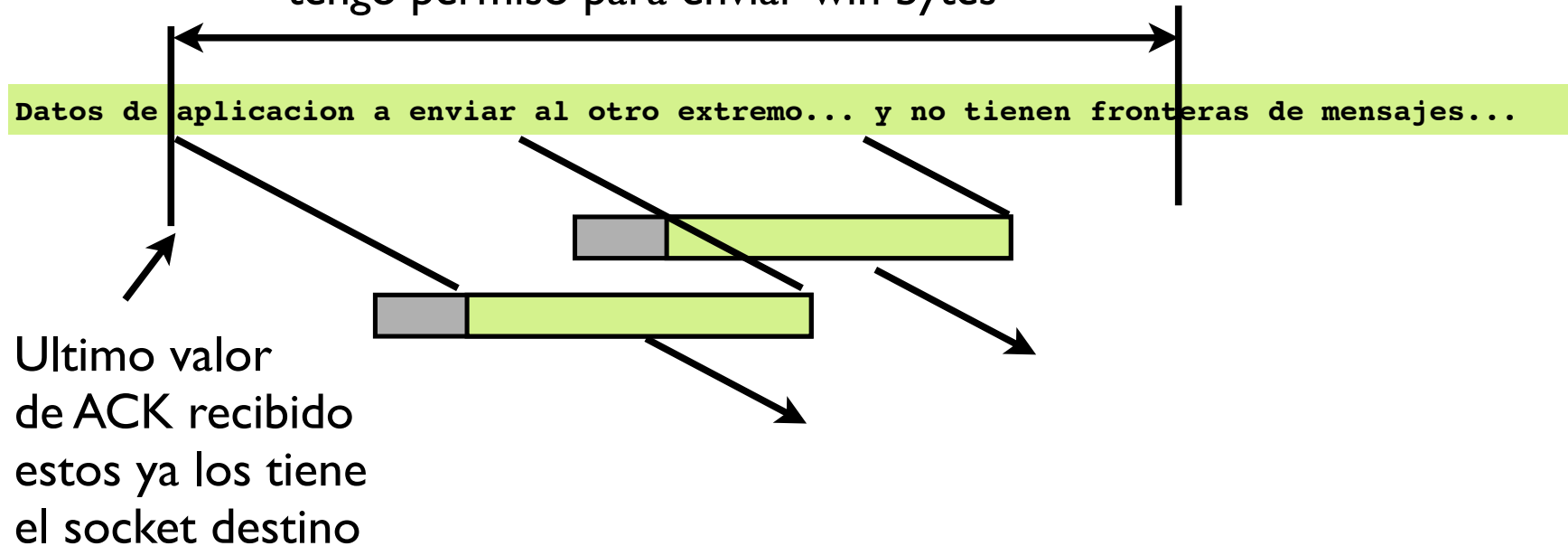


TCP: envío de datos

En cada momento en el receptor

win ventana recepcion

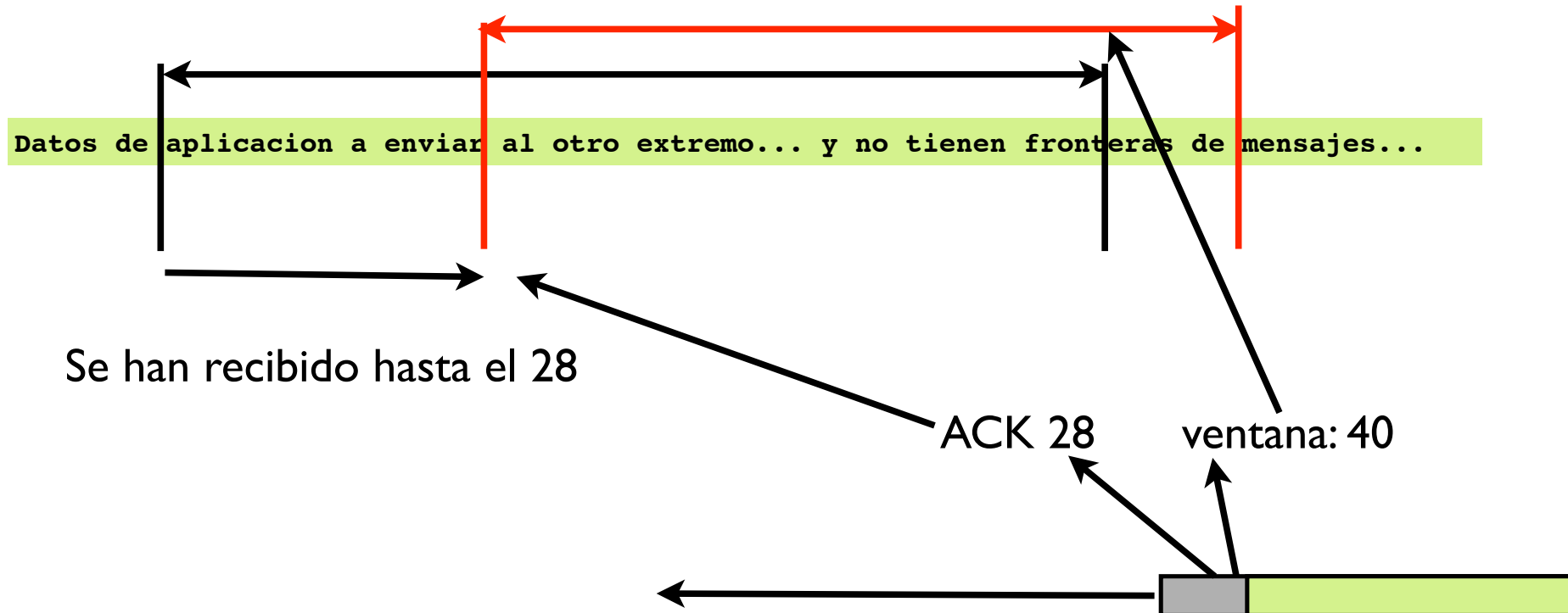
tengo permiso para enviar win bytes



Los bytes dentro de la ventana permitida se van colocando en paquetes y enviando

TCP: envío de datos

win ventana recepcion
tengo permiso para enviar estos



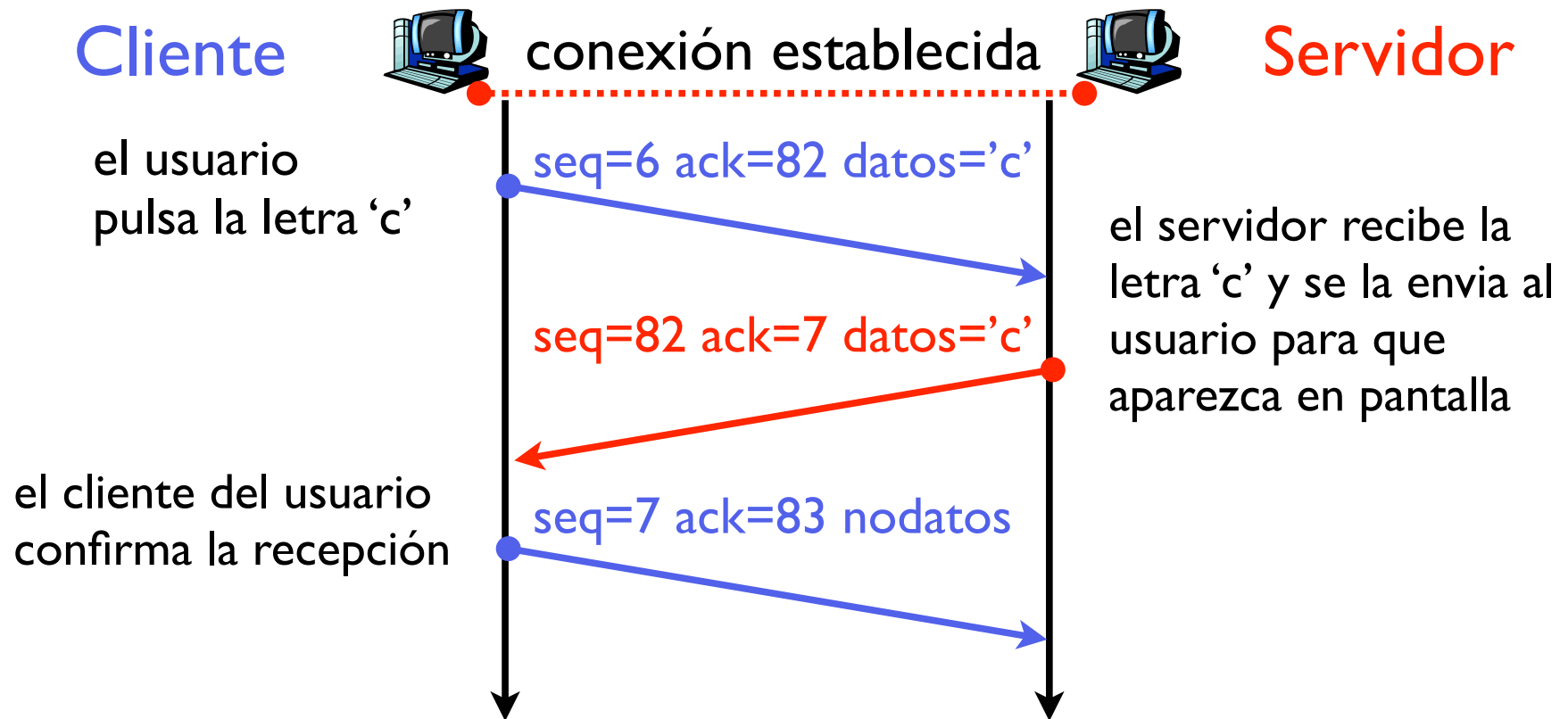
Los paquetes que llegan del otro extremo indican
Campo ACK: que bytes han sido ya recibidos
Campo win: cuantos bytes puedo enviar al receptor de momento
La ventana se desplaza hacia valores mayores

TCP: envío de datos

- ▶ **Secuencia y ACK: campos de 32 bits**
 - > 4 Gb de datos para dar la vuelta pero solo podemos mandar los que indica la ventana (ventana deslizante)
 - > 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
confirma todos los datos anteriores a ese (del sentido contrario)
Esto se llama: **cumulative ACK** o **Go back N**
- ▶ Si una conexión está transmitiendo en ambos sentidos los ACKs de un sentido van en los paquetes de datos del opuesto **piggyback**
- ▶ Si solo transmite en un sentido se enviaran paquetes sin datos solo para enviar el ACK

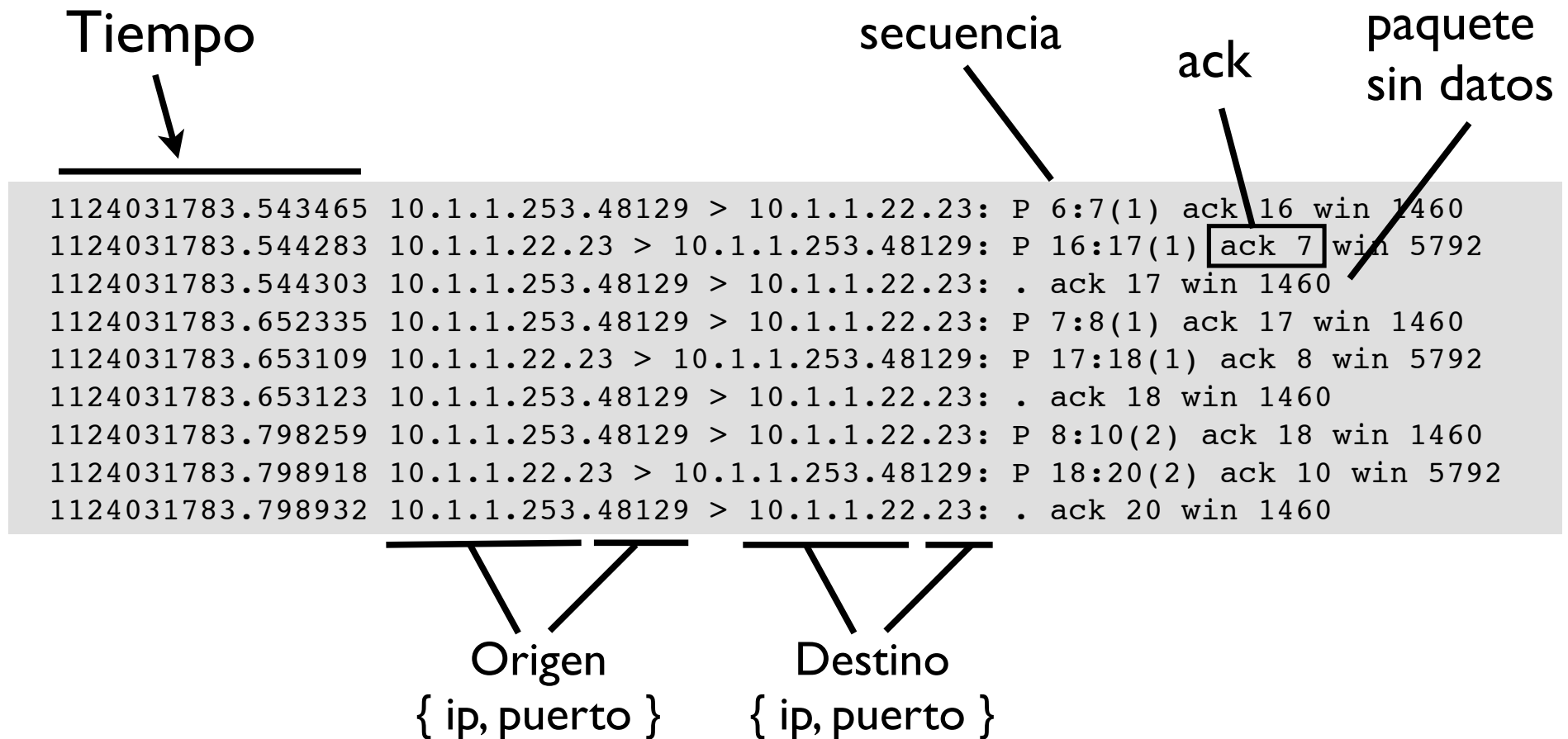
Ejemplo

- ▶ Paquetes de un telnet desde 10.1.1.253 a 10.1.1.22



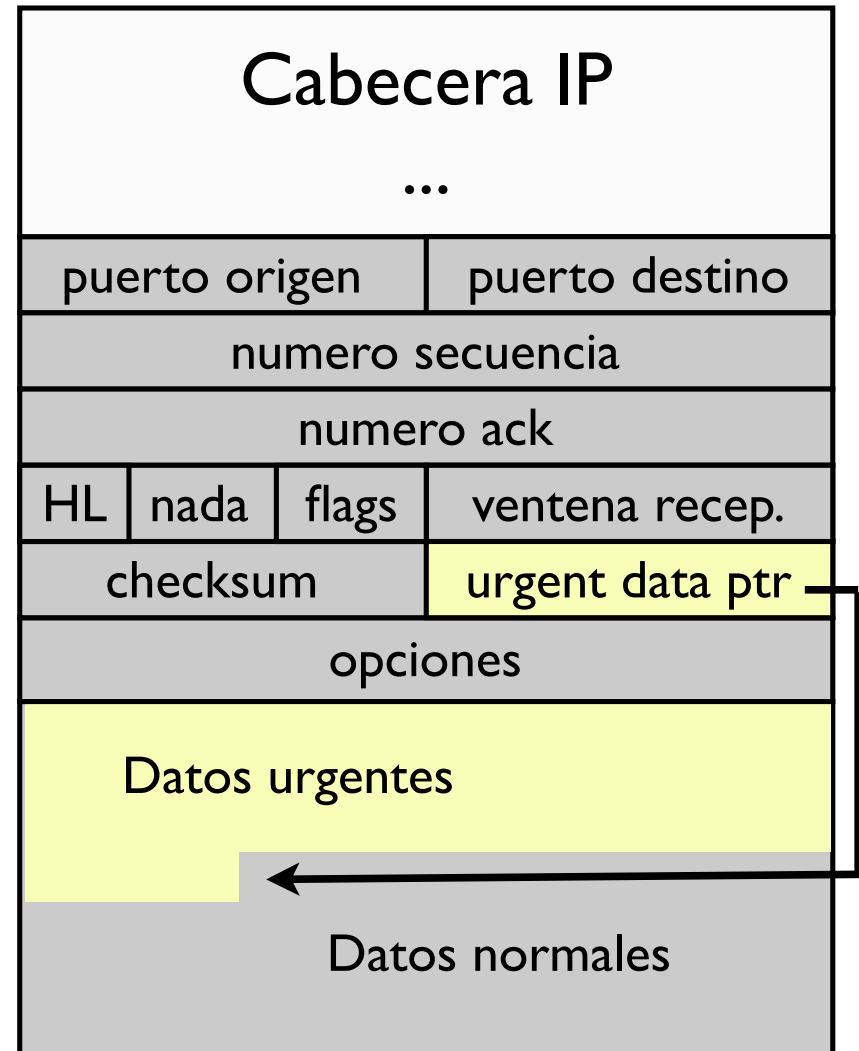
Ejemplo

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



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
- En sockets los datos urgentes hay que pedirlos con `setsockopt`



Resumen

- ▶ TCP es el protocolo de transporte fiable de Internet
 - > Multiplexación de procesos usando puertos
 - > Más complicado y con mas overhead que UDP
 - > Varios tipos de mensajes en un mismo formato de paquete
 - > Fiabilidad mediante ventana deslizante, numero de secuencia y ACKs

- ▶ ¿Como se establecen las conexiones TCP?

TCP: conexiones

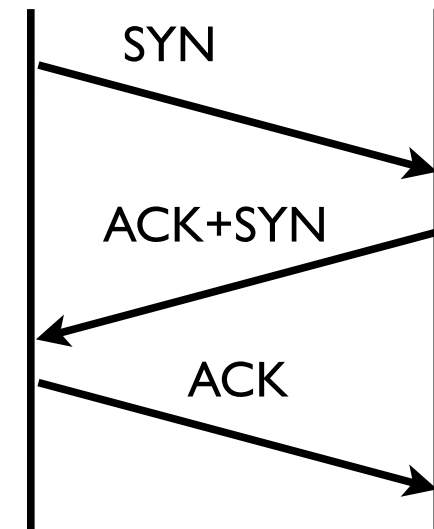
- ▶ TCP es orientado a conexión
- ▶ Previamente a comunicarse datos entre un emisor y un receptor deben negociar un establecimiento de conexión.
 - > TCP inicializa sus variables para la conexión y crea los buffers
 - > Esto se hace mediante los paquetes que utilizan los flags SYN, FIN y RST
 - > Protocolo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

▶ Otra conexión web...

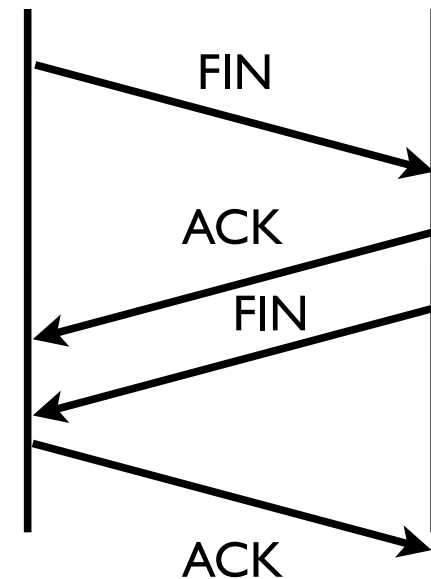
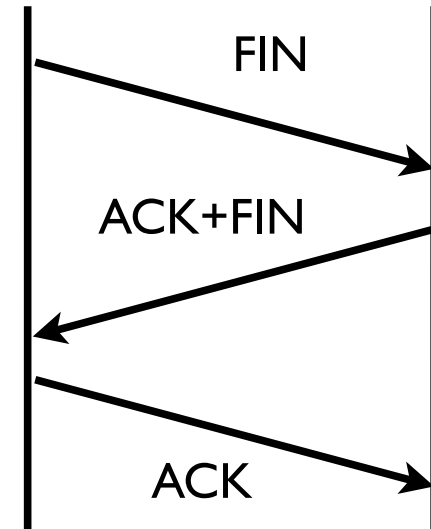
Los SYNs usan un número de secuencia para poder ser confirmados

```
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**



Cierre de la conexión

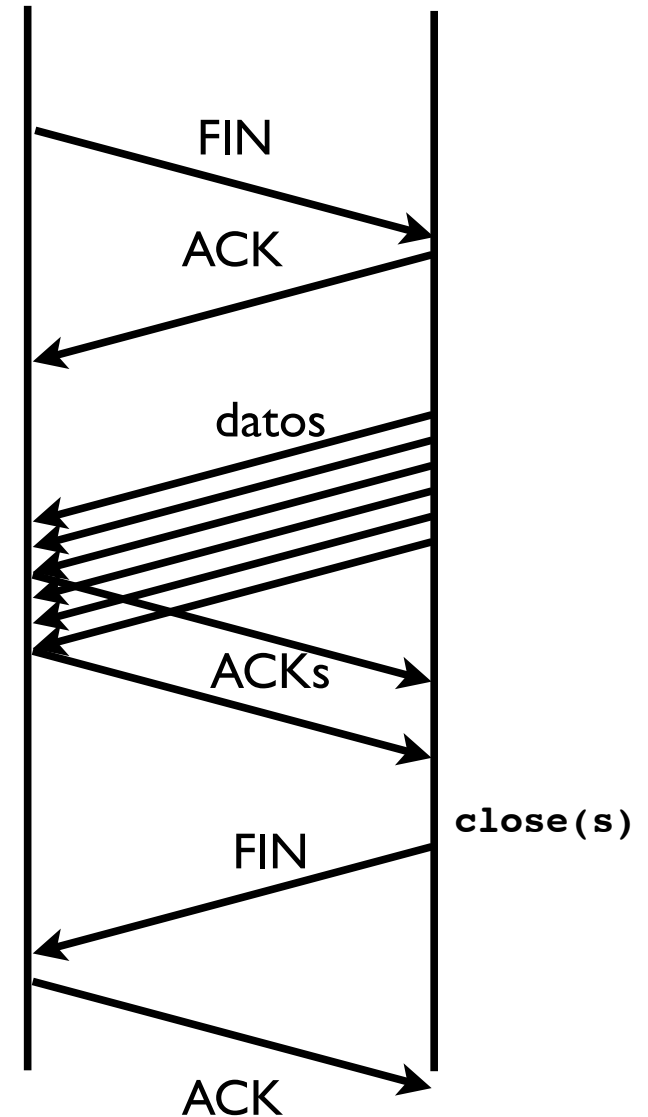
► Medio cierre

- > El cierre de cada sentido de la conexión es en realidad independiente
- > Un extremo puede cerrar la conexión indicando que no enviara mas datos pero puede seguir recibiendo lo que le envian

Recuérdese la funcion:

```
shutdown(socket, SHUT_WR)
```

```
shutdown(s, SHUT_WR)
```



Ejemplo

- ▶ El final de una conexión web...

El servidor está enviando datos

El cliente decide cerrar y manda un FIN

...

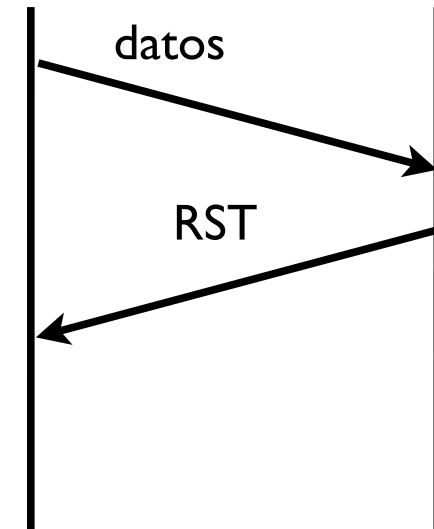
```
IP 130.206.166.105.80 > 130.206.169.177.53701: P 80314174:80315622(1448) ack 4067364561 win 24616
IP 130.206.166.105.80 > 130.206.169.177.53701: P 80315622:80316551(929) ack 4067364561 win 24616
IP 130.206.169.177.53701 > 130.206.166.105.80: . ack 80316551 win 65535
IP 130.206.169.177.53701 > 130.206.166.105.80: 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: abortar una conexión

▶ Paquete Reset (RST)

- > Se envía cuando TCP recibe un paquete que es inconsistente con su estado de la conexión
 - recibir datos sin tener conexión abierta
- > Le dice al otro extremo que esa conexión no existe y que destruya toda la información de ese estado de conexión
- > También se usa para decir que no hay nadie escuchando un puerto
- > También se puede usar por el nivel de aplicación para cerrar una conexión de forma rápida (setsockopt con la opción SO_LINGER)
- > El otro extremo no hace falta que conteste nada



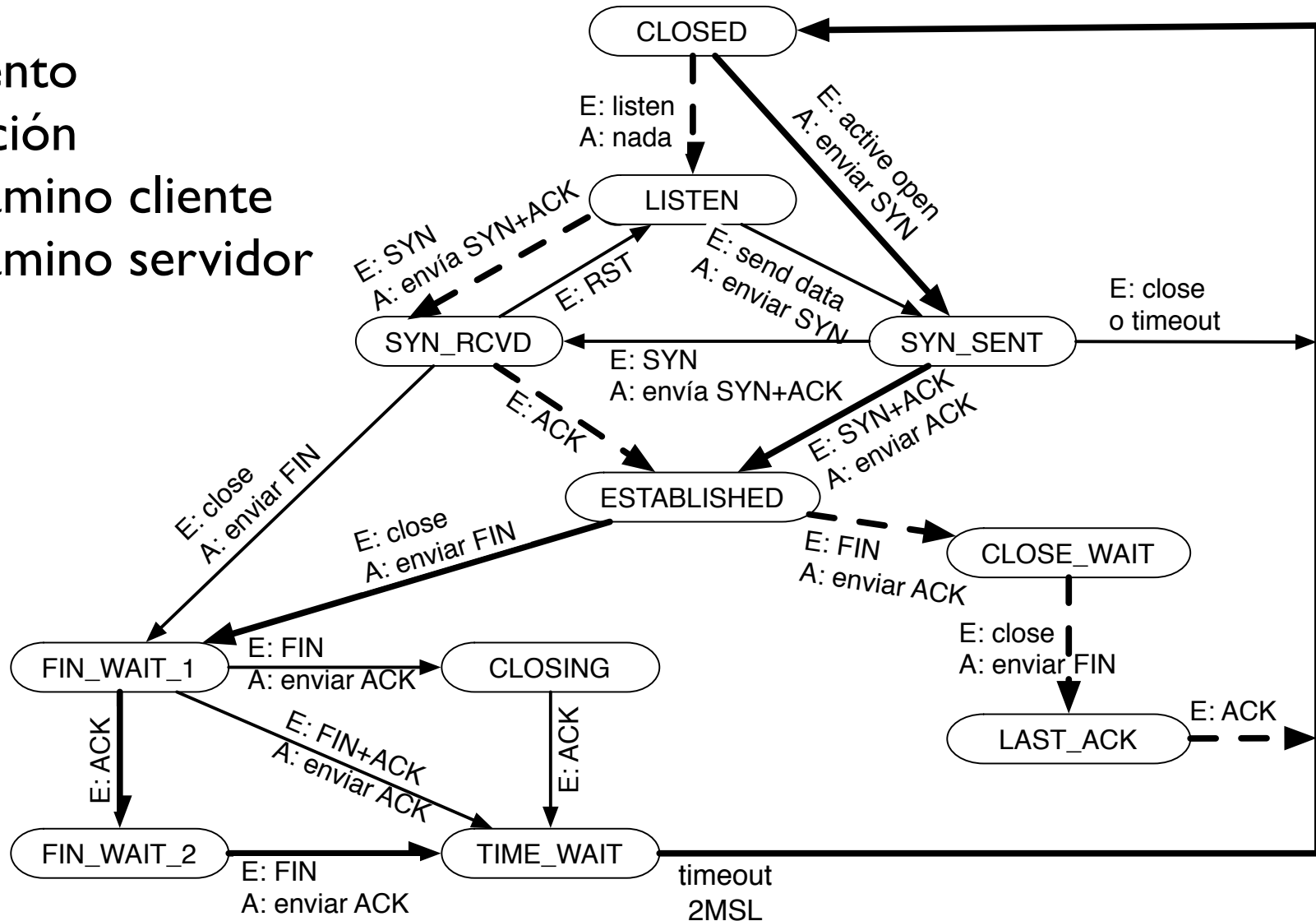
TCP: establecimiento de conexión

- ▶ ¿Qué pasa si se pierden paquetes del establecimiento o del cierre?
 - > Tienen un número de secuencia así que se pueden retransmitir. Es como si fueran un byte de datos.
 - > Se utiliza retransmisión por timeout
- ▶ Retransmision de SYNs
 - > **Problema:** normalmente el tiempo de espera depende de los tiempos de ida y vuelta (RTT) que se observan. Al comenzar la conexión no ha habido tiempo de hacer estimaciones del RTT.

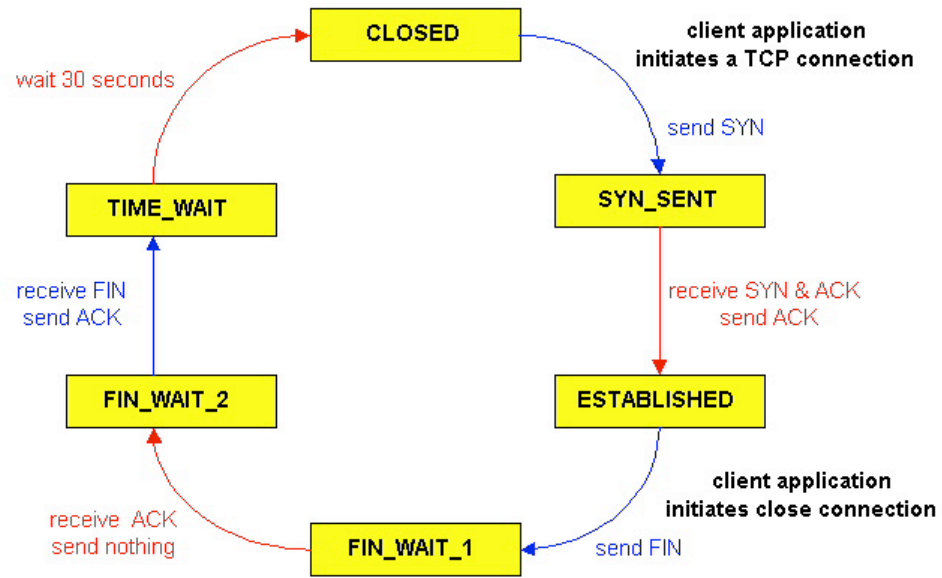
La mayoría de las implementaciones utilizan un timeout inicial de 6 segundos. Si falla el timeout se pone a 24 segundos y se va doblando
- ▶ Retransmision de FINs
 - > **Problema:** el sistema operativo no puede deshacerse del estado de la conexión inmediatamente. Tiene que mantenerlo un tiempo por si acaso hacen falta retransmisiones

TCP : Diagrama de estados

- E: evento
- A: acción
- camino cliente
- camino servidor

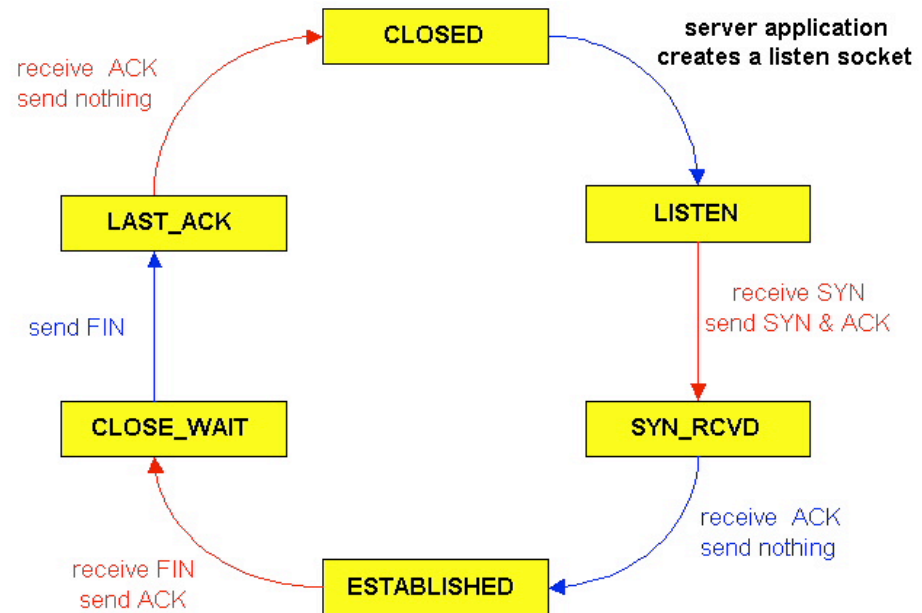


TCP estados



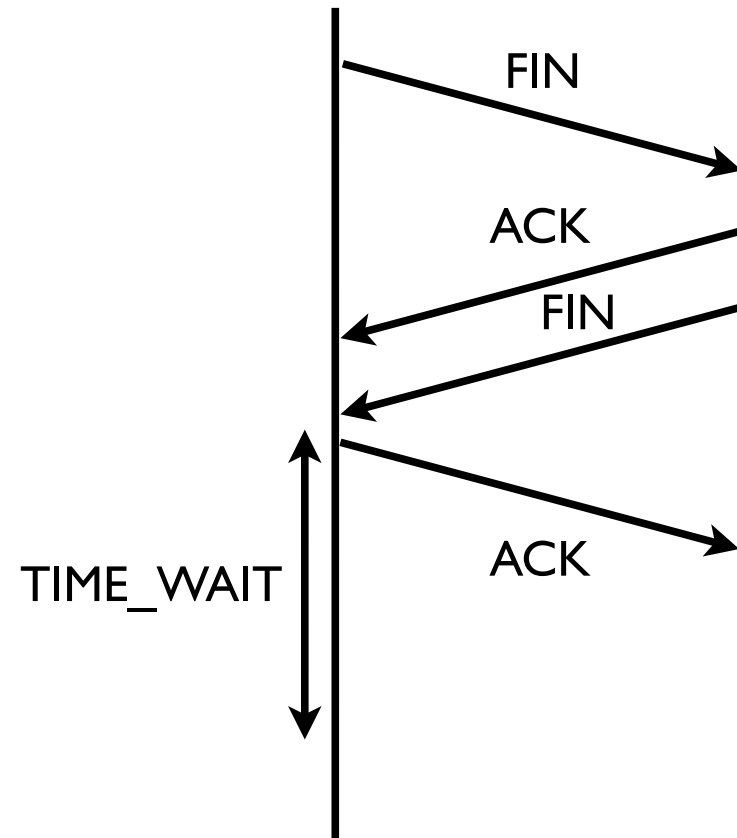
TCP Estados del cliente

TCP Estados del servidor



TIME_WAIT

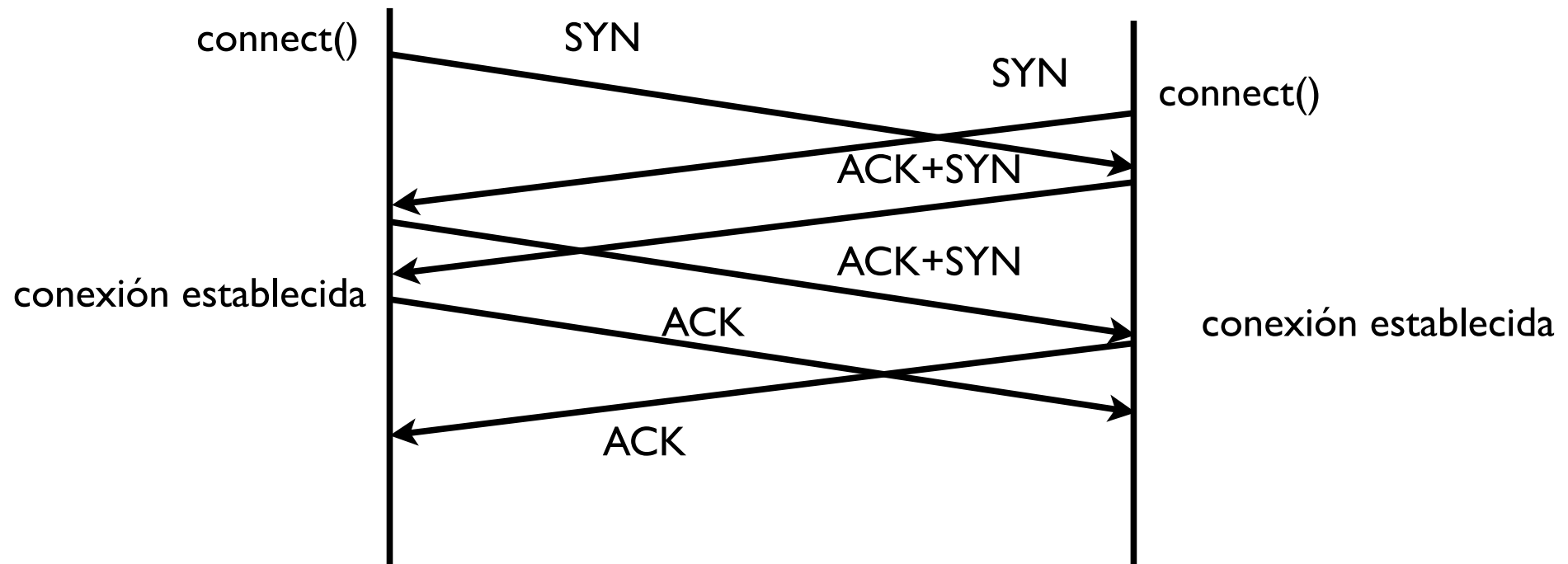
- ▶ El extremo que ha enviado el primer FIN
 - > Debe mantener el estado de la conexión un tiempo (TIME_WAIT)
 - > No puede estar seguro que el otro extremo ha recibido su ACK y podría retransmitir el FIN
 - > Cuanto tiempo esperar?
2MSL (maximum segment lifetime)
 - > Qué pasa si llega un segmento de FIN después de eso??



TCP: establecimiento de conexión

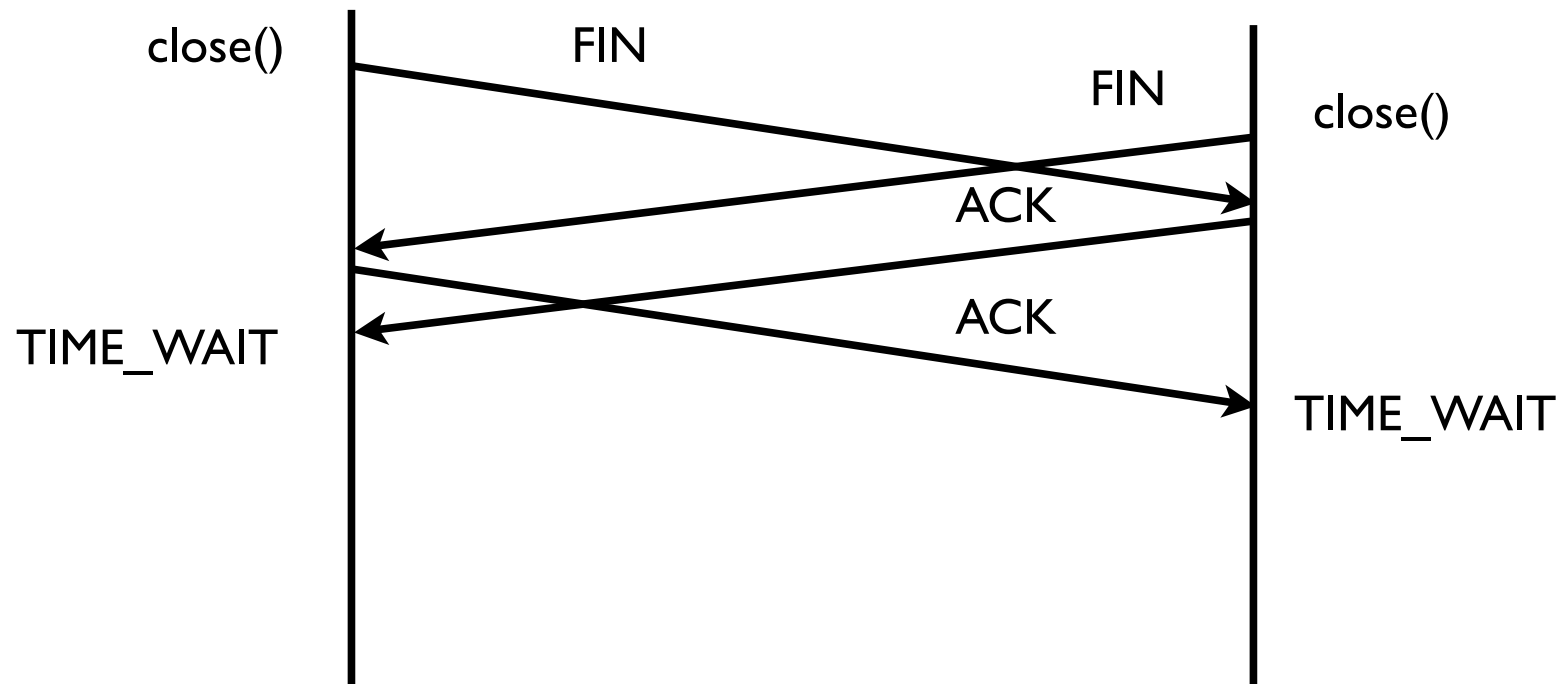
▶ Casos especiales: apertura simultánea

- > Si dos clientes inician simultáneamente una conexión entre ellos
- > Nótese que los dos tienen que usar puertos conocidos por el otro
- > Se usa muy poco



TCP: establecimiento de conexión

- ▶ Casos especiales: cierre simultáneo
 - > Los dos extremos deciden cerrar a la vez
 - > Los dos extremos deben esperar en TIME_WAIT para poder retransmitir el FIN



Control de flujo y congestión

▶ Control de flujo

- > No quiero enviar tan rápido que cause problemas al receptor [tendra buffer suficiente para guardar lo que le envío?]
- > La **ventana de recepcion** sirve para eso solo mando lo que el receptor me ha autorizado

▶ Control de congestión

- > No quiero enviar tan rápido que cause problemas a la red
- > Dificil no podemos saber lo que están enviando los demás
- > Podemos observar las perdidas y reaccionar bajando la velocidad cuando hay problemas... [pero esto causa mas problemas]
- > El control de congestion es un problema muy dificl de redes
- > TCP lo hace con ventanas que dependen de las perdidas pero hay mas ideas por ahi...

Conclusiones

- ▶ TCP es el protocolo de transporte fiable de Internet
 - > Multiplexación de procesos usando puertos
 - > Más complicado y con mas overhead que UDP
 - > Varios tipos de mensajes en un mismo formato de paquete
 - > Fiabilidad mediante ventana deslizante, numero de secuencia y ACKs
- ▶ Establecimiento y liberación de conexiones TCP
- ▶ Estados de una conexión

Próxima clase:

- ▶ Problemas
- ▶ Nivel de red