TCP

Transporte fiable + control de flujo

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

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

Temario

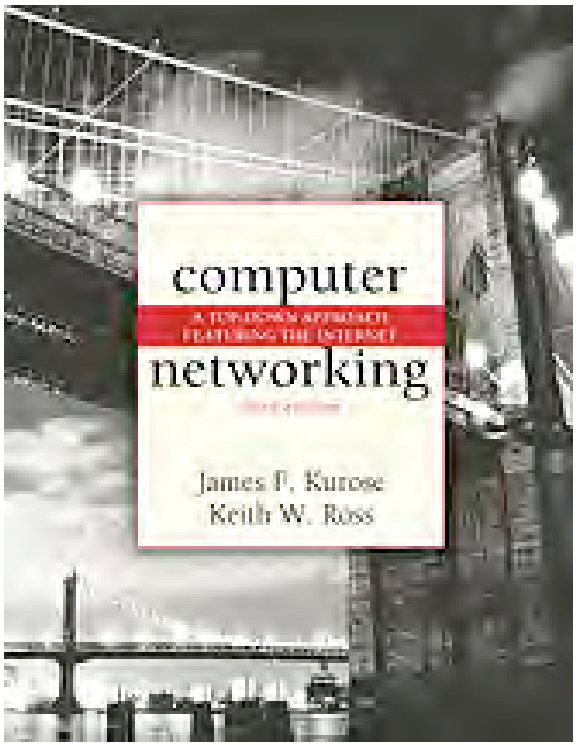
1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

Temario

1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
   - Principios
   - Problemas básicos
     - Encaminamiento
     - + arquitectura de routers
     - **Transporte fiable (hoy: en TCP)**
     - **Control de flujo (hoy: en TCP)**
     - Control de congestión
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

Material

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

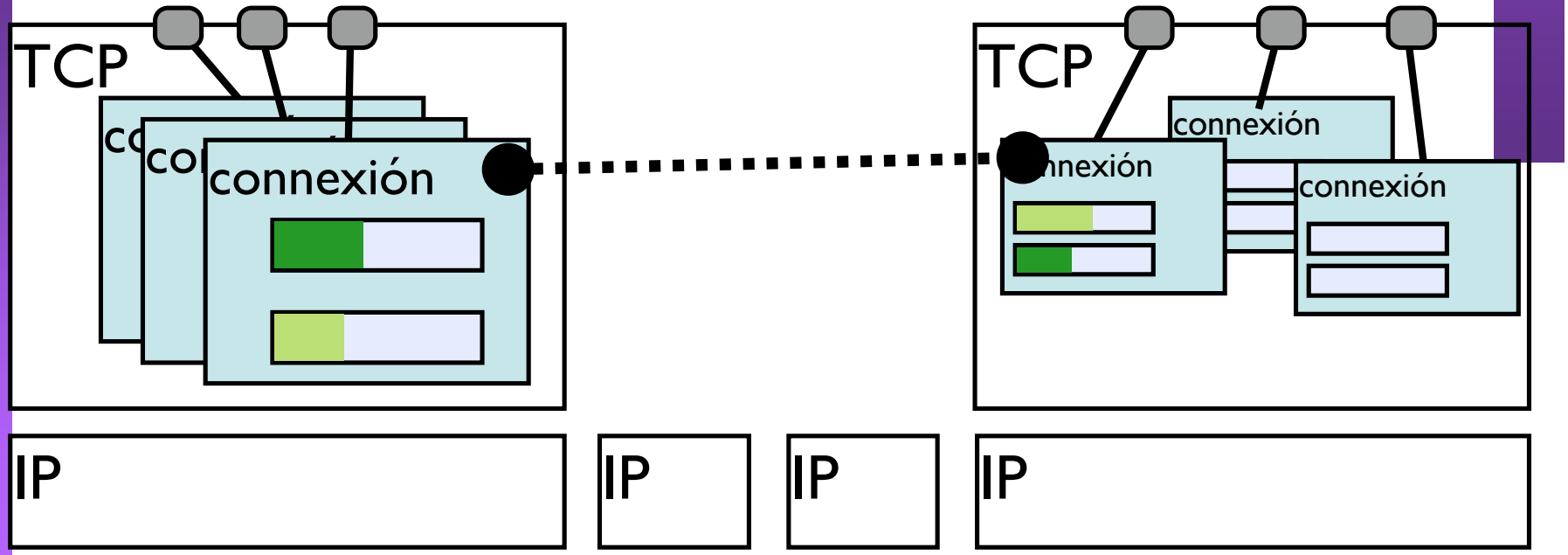


TCP

- 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

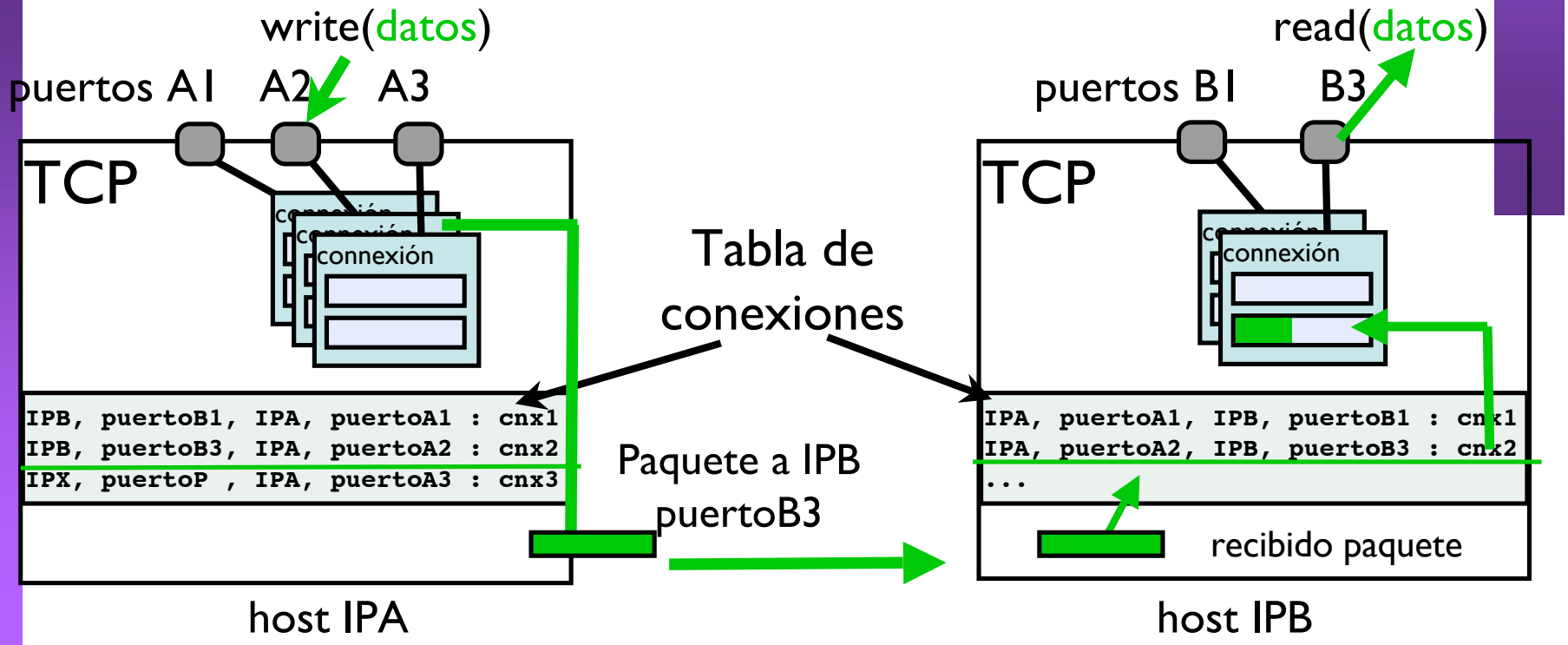
TCP

- 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 envio
    - 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 conexiónes.
- La conexión sólo existe en los extremos TCP

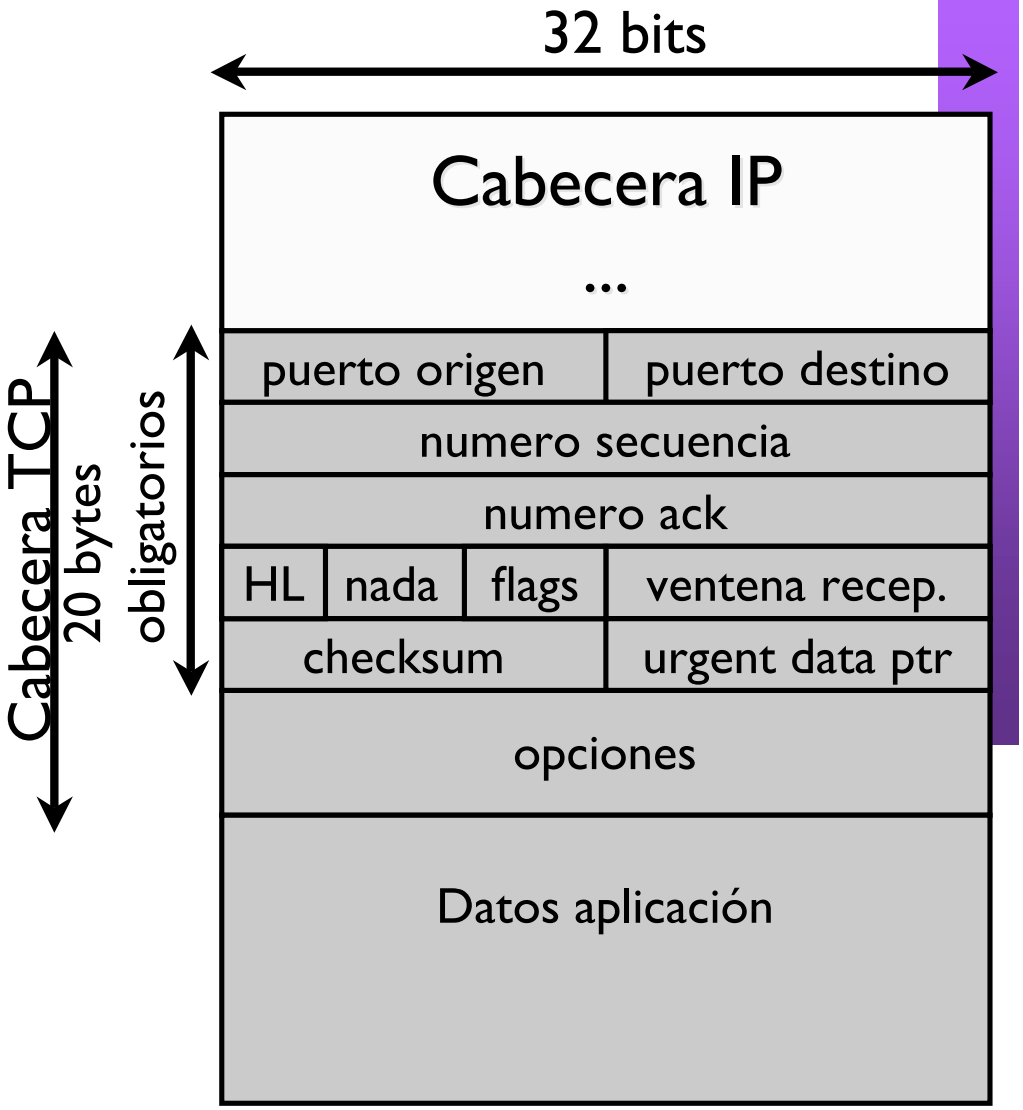


TCP

- Los buffers aislan a TCP de las operaciónes 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

TCP

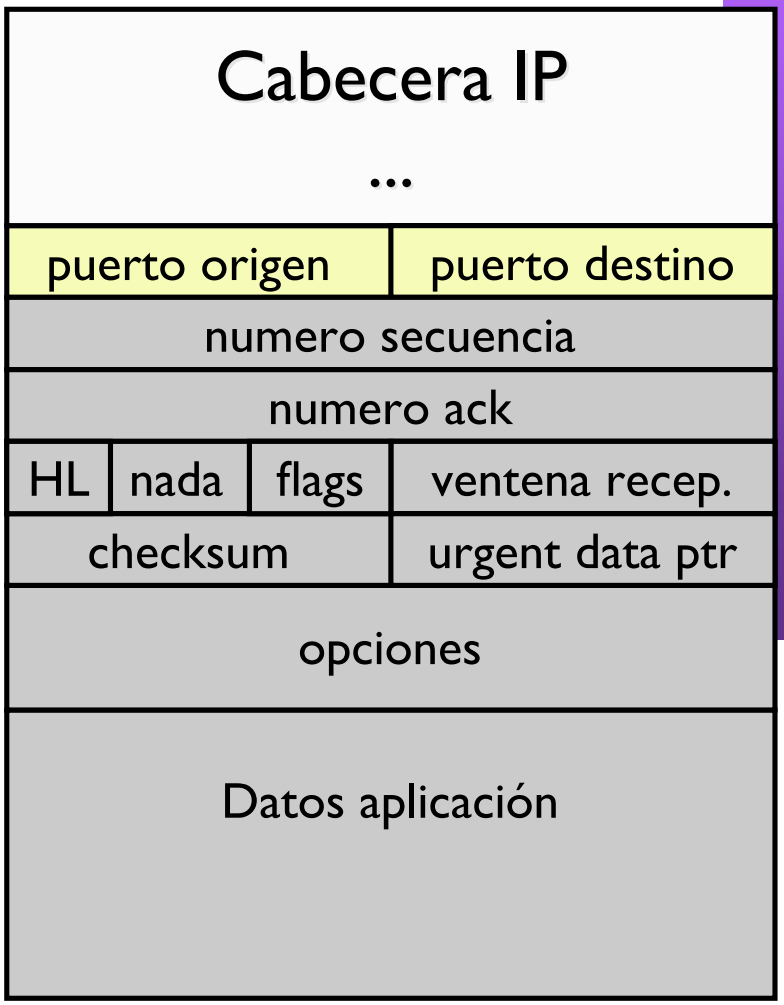
- Segmento TCP
- Cabecera de tamaño variable
  - 20 hasta 60 bytes según las opciones
- Datos del nivel de aplicación



TCP

Contenido

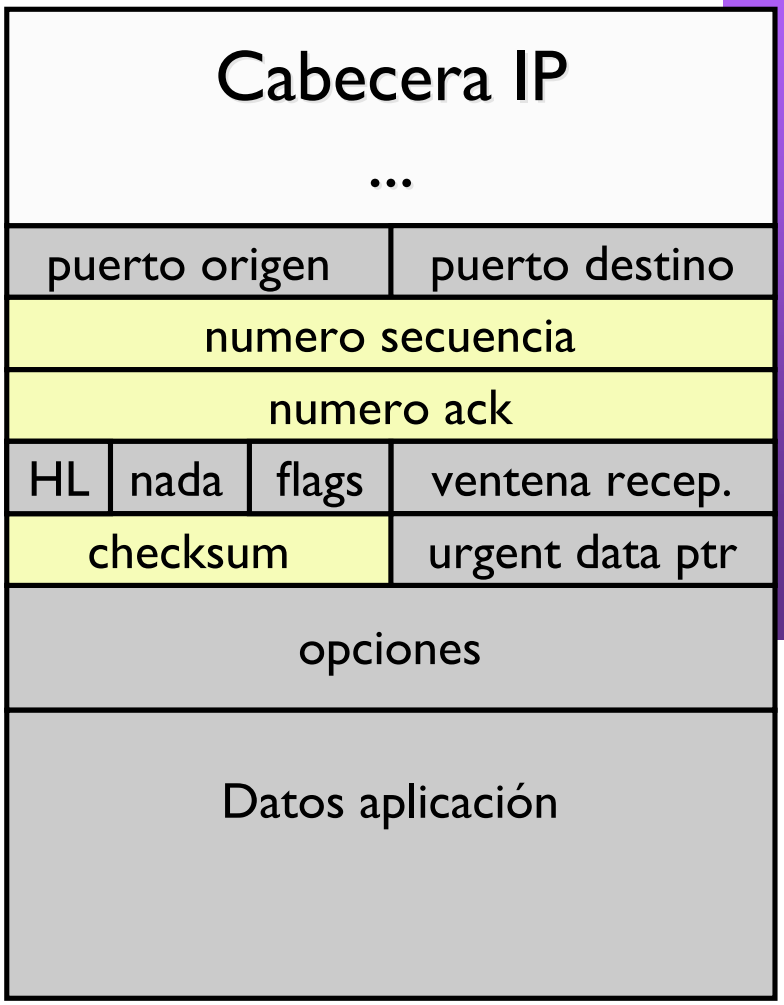
- Datos de multiplexación
  - Puerto origen
  - Puerto destino



TCP

Contenido

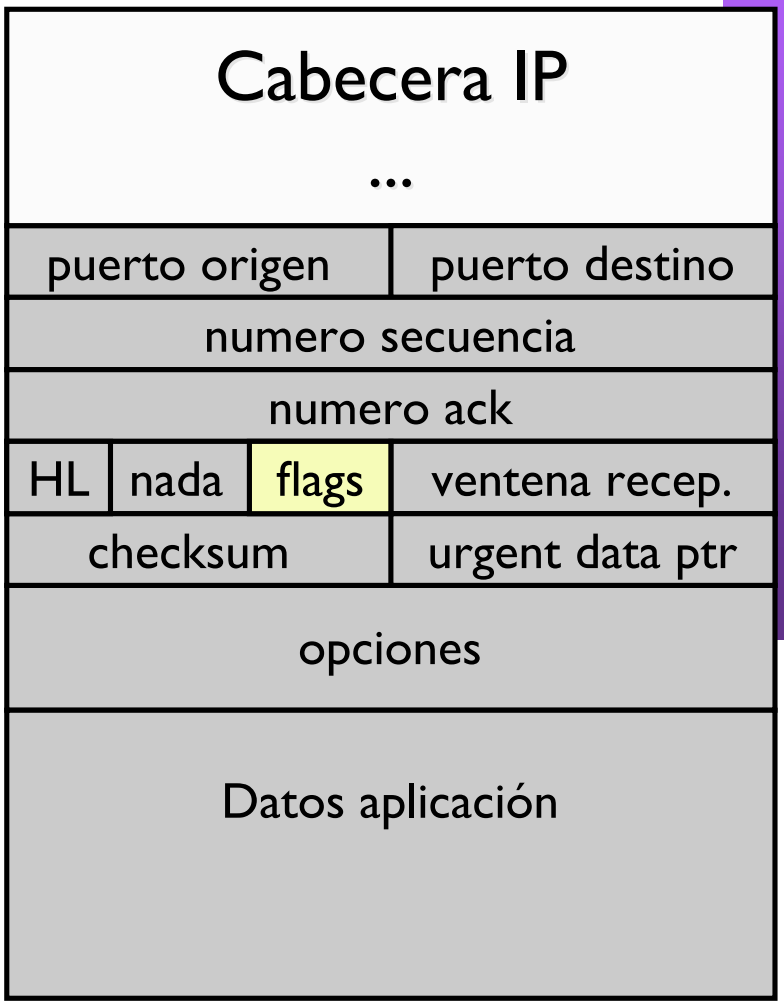
- Datos para transporte fiable
  - Número de secuencia
  - Número de ACK
  - Checksum
    Cabecera + datos de application + algunos datos de IP (pseudo cabecera como en UDP)
- En un mismo paquete podemos mandar datos y confirmar datos del sentido contrario



TCP

Contenido

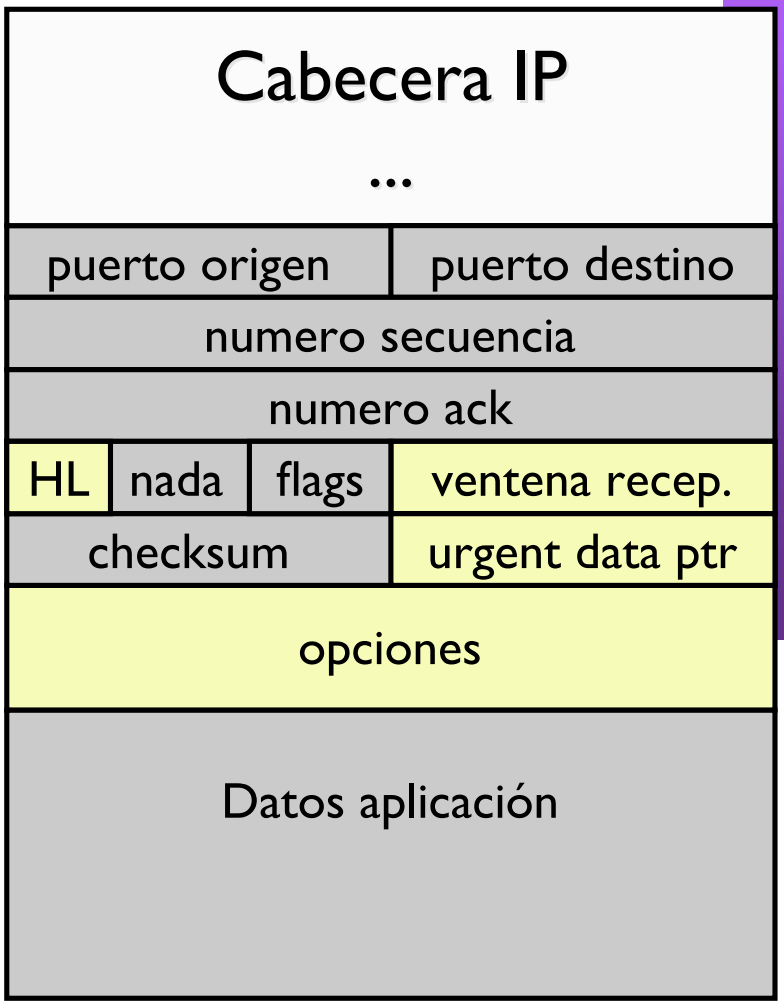
- FLAGs: diferentes tipos de paquetes del protocolo
  - URG urgente
  - ACK acknowledgement
  - PSH push
  - RST reset
  - SYN syn
  - FIN fin



TCP

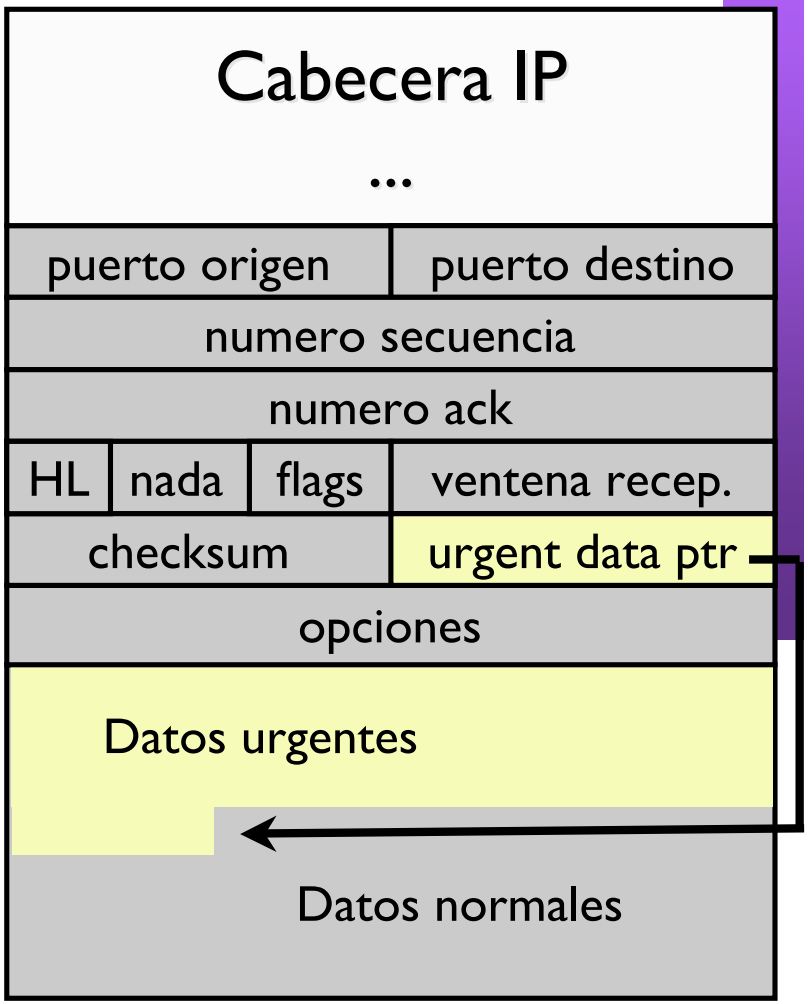
Contenido

- 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



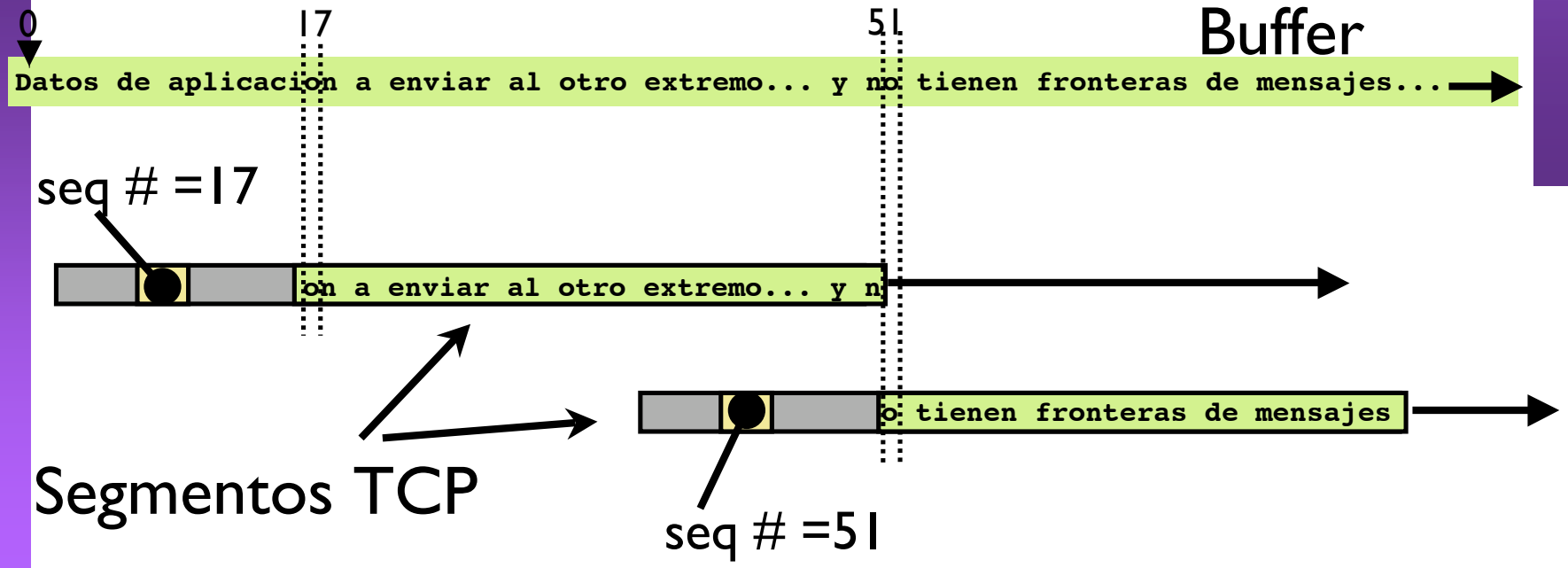
Datos urgentes

- 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
  En sockets los datos urgentes hay que pedirlos con setsockopt



TCP: envío de datos

- Los bytes a enviar se colocan en el buffer y forman una corriente de bytes sin fronteras de paquetes
- TCP envia 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 numero de ACK) hacen referencia al primer byte del paquete en la secuencia global

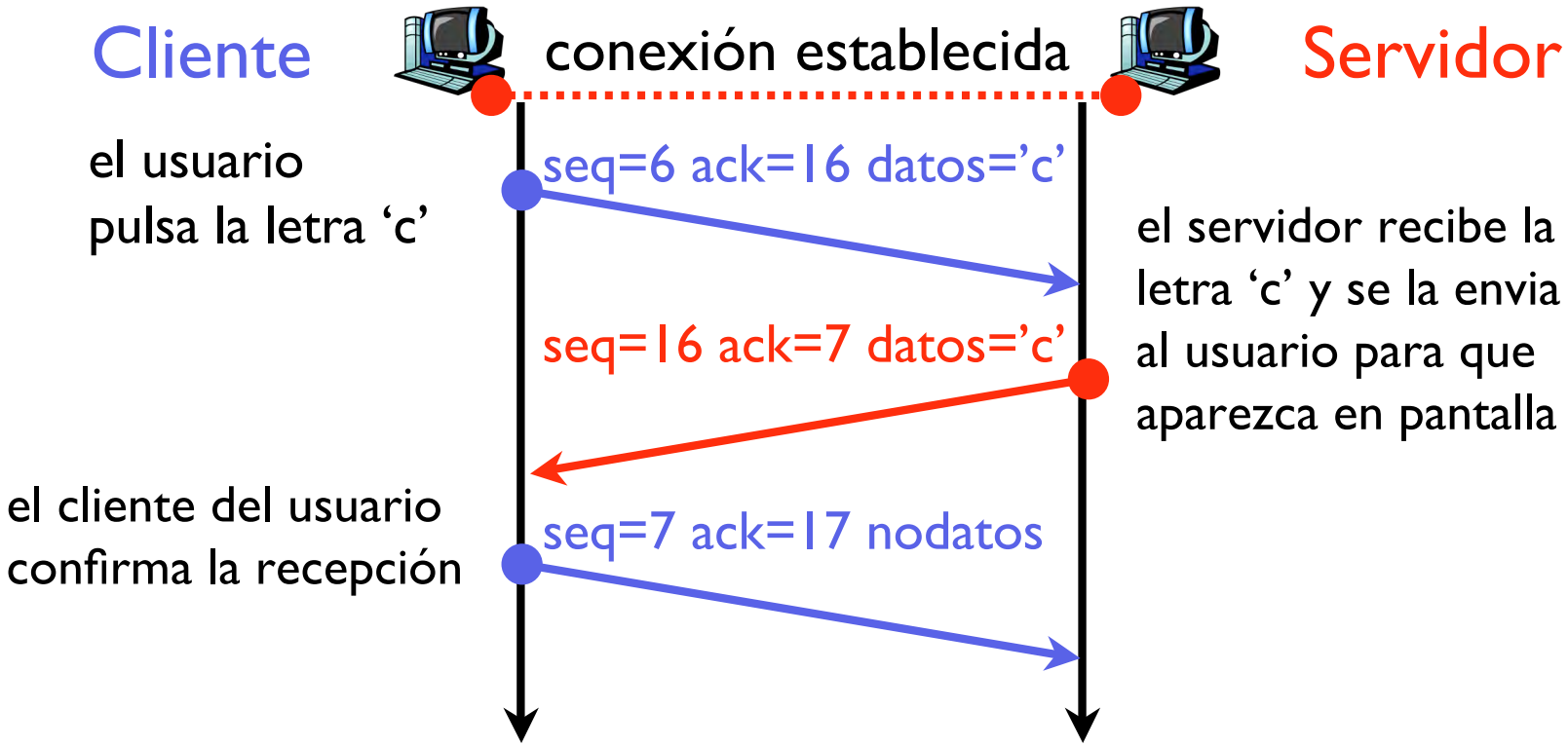


TCP: envío de datos

- Secuencia y ACK: campos de 32 bits
  - 4 Gb de datos para 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

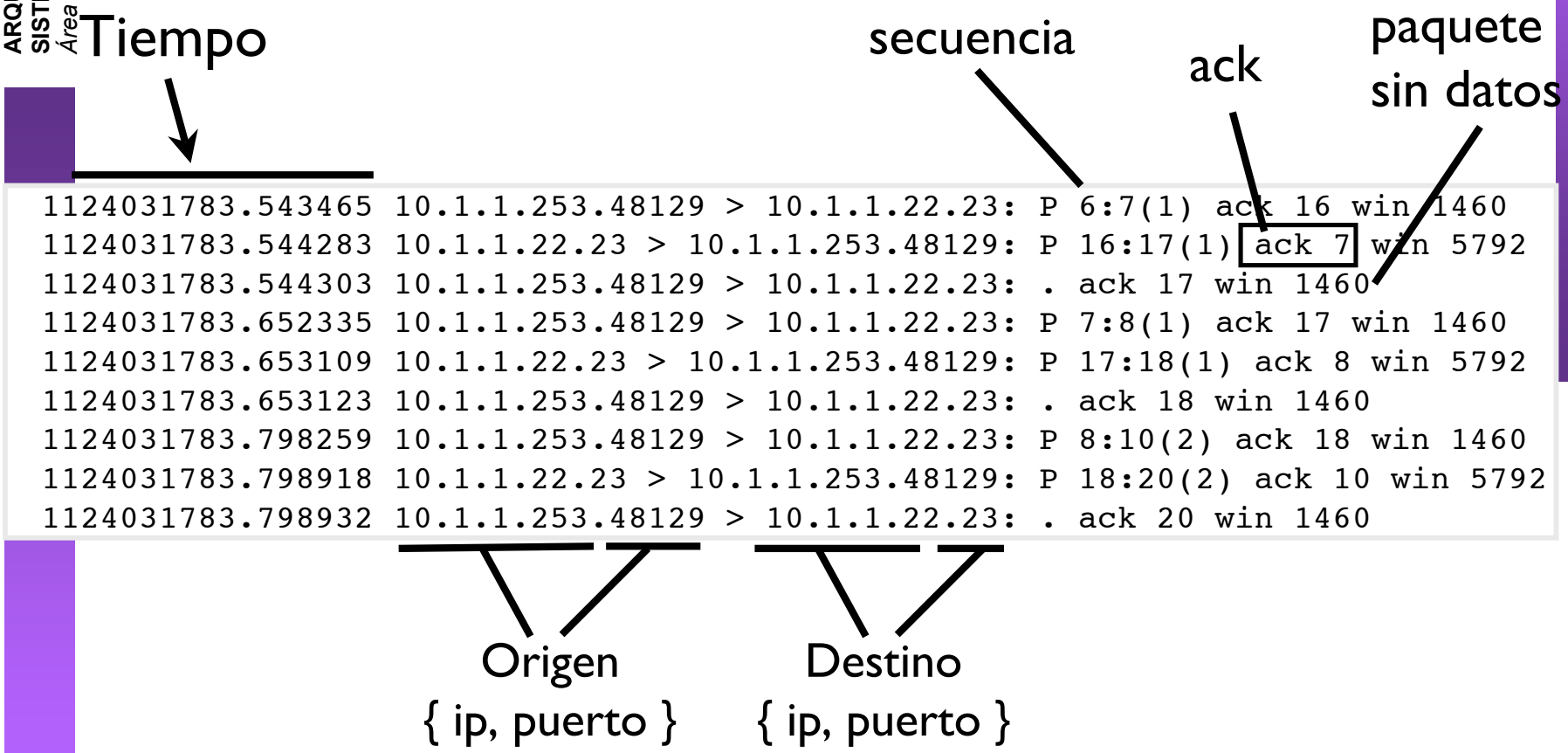
Ejemplo

- 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 rcibir ACK)
- Los paquetes TCP llevan
  - Número de secuencia de los datos. Si no llevan datos, el campo número de secuencia indica el proximo numero 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 aplicacion

- 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

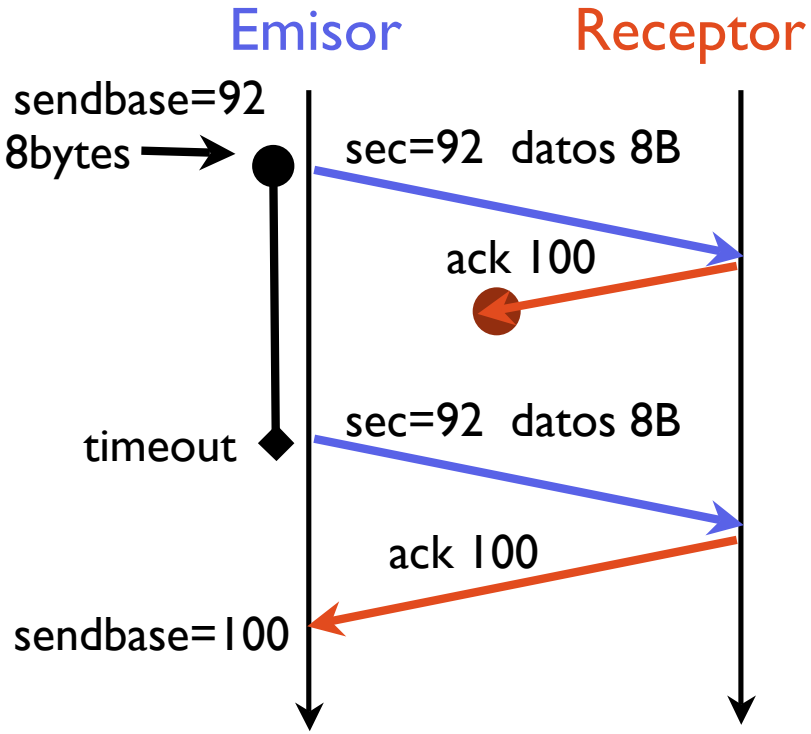
- Si confirme un segmento nuevo

> actualizar ventana cumulative ACK

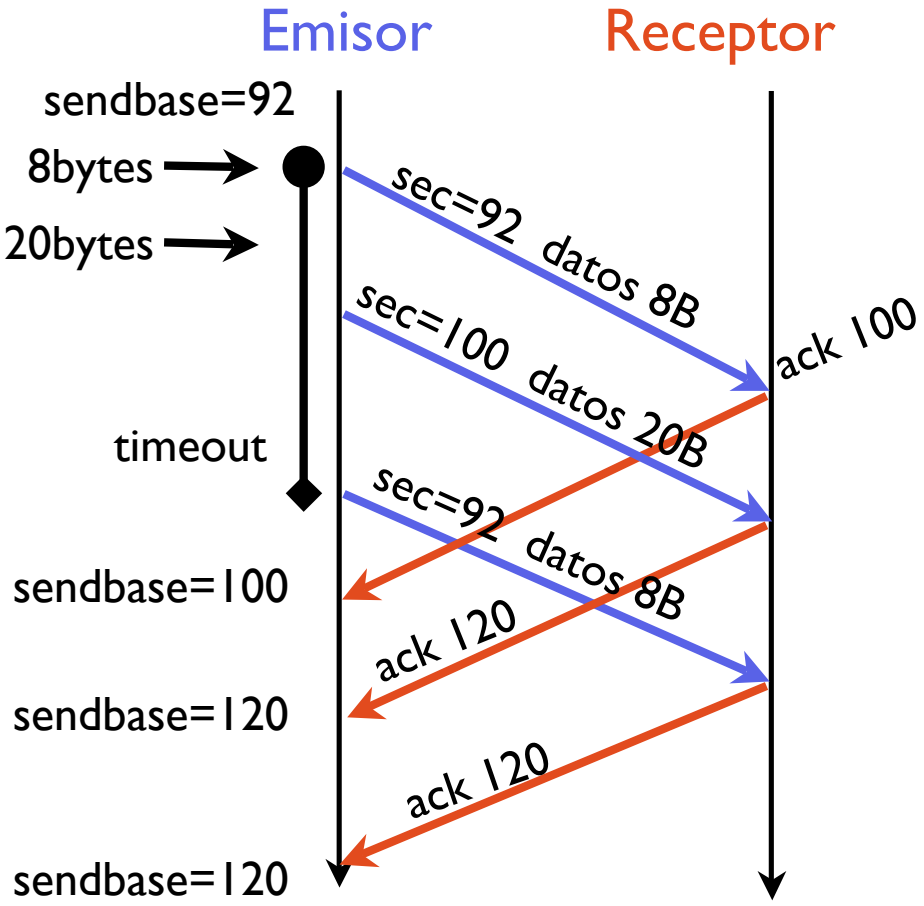
> reiniciar timeout si quedan segmentos por confirmar

Parece de tipo Go back-N

Ejemplos

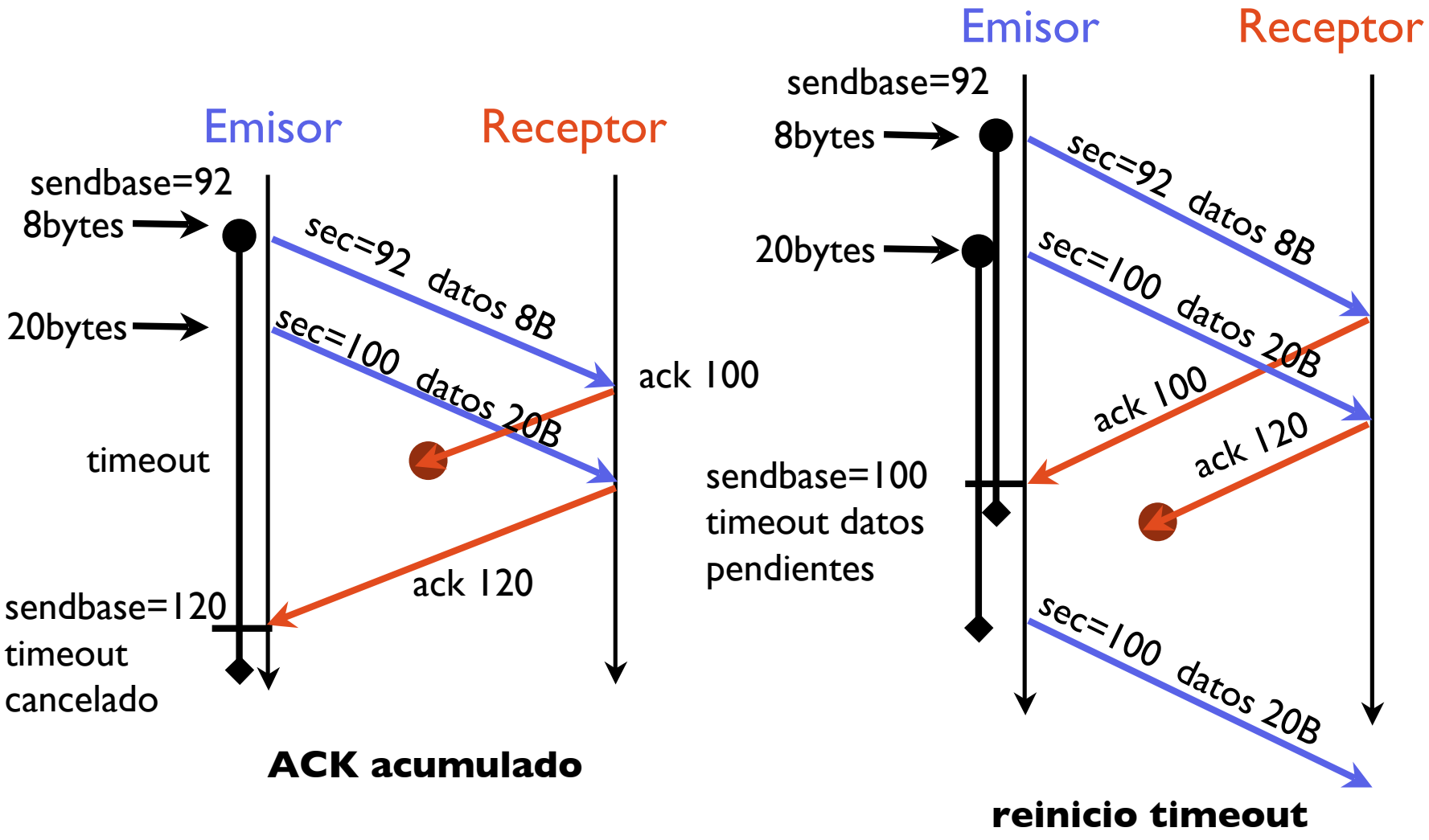


pérdida de ACK



timeout prematuro

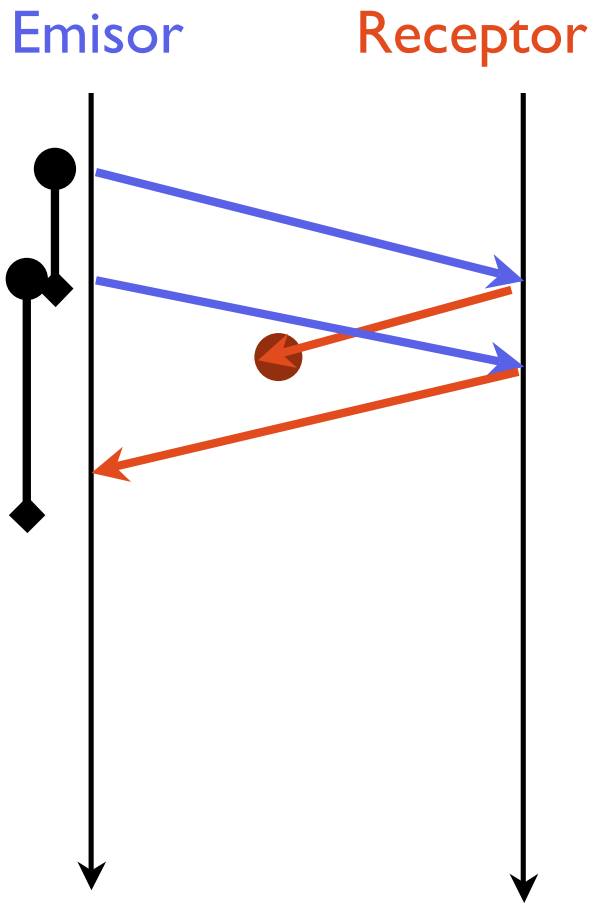
Ejemplos


Emisor
Receptor
sendbase=92
8bytes
sec=92 datos 8B
20bytes
sec=100 datos 20B
ack 100
timeout
ack 120
sendbase=120
timeout
cancelado
ACK acumulado
Emisor
Receptor
sendbase=92
8bytes
sec=92 datos 8B
20bytes
sec=100 datos 20B
ack 100
ack 120
sendbase=100
timeout datos
pendientes
sec=100 datos 20B
reinicio timeout

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



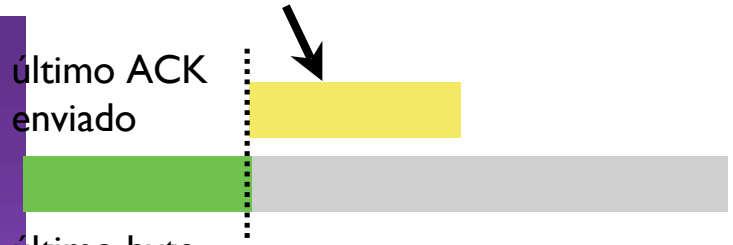
TCP: transporte fiable

- 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 tambien más complicado en realidad

TCP: receptor

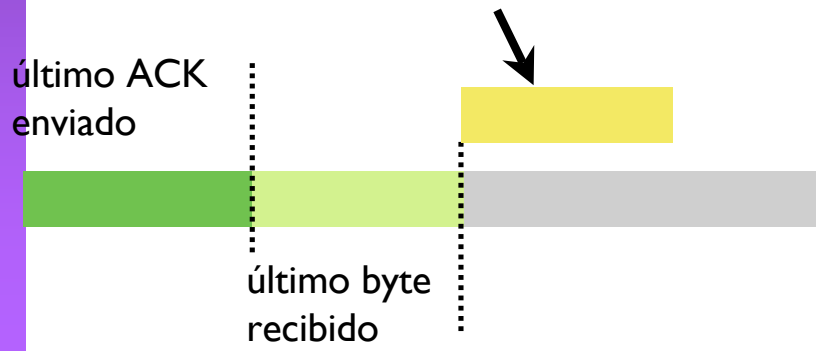
Eventos del receptor

- **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

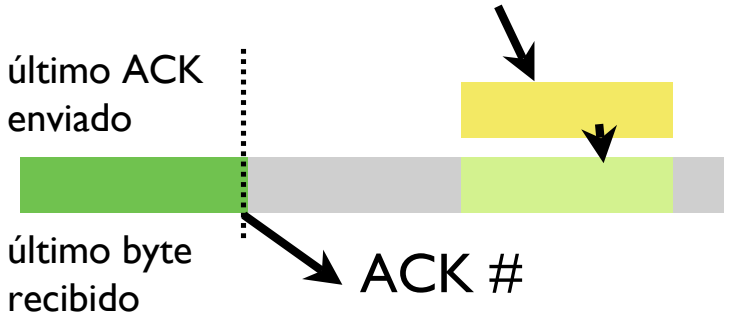


Acción: envía inmediatamente ACK (al ser acumulado confirma los dos)

TCP: receptor

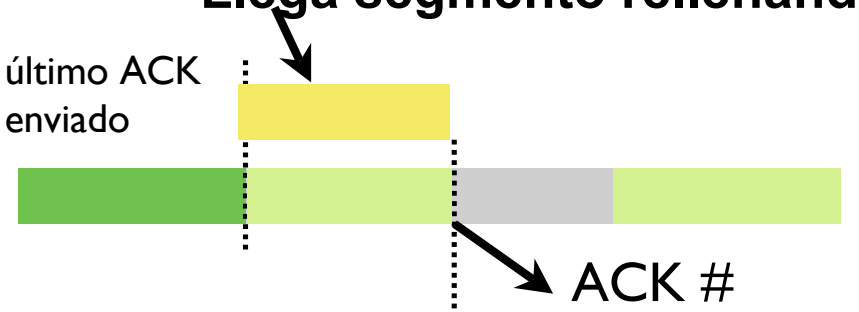
Eventos del receptor

- **Llega segmento fuera de orden generando hueco**



Acción: envía inmediatamente ACK causando ACK duplicado

- **Llega segmento rellenando hueco**



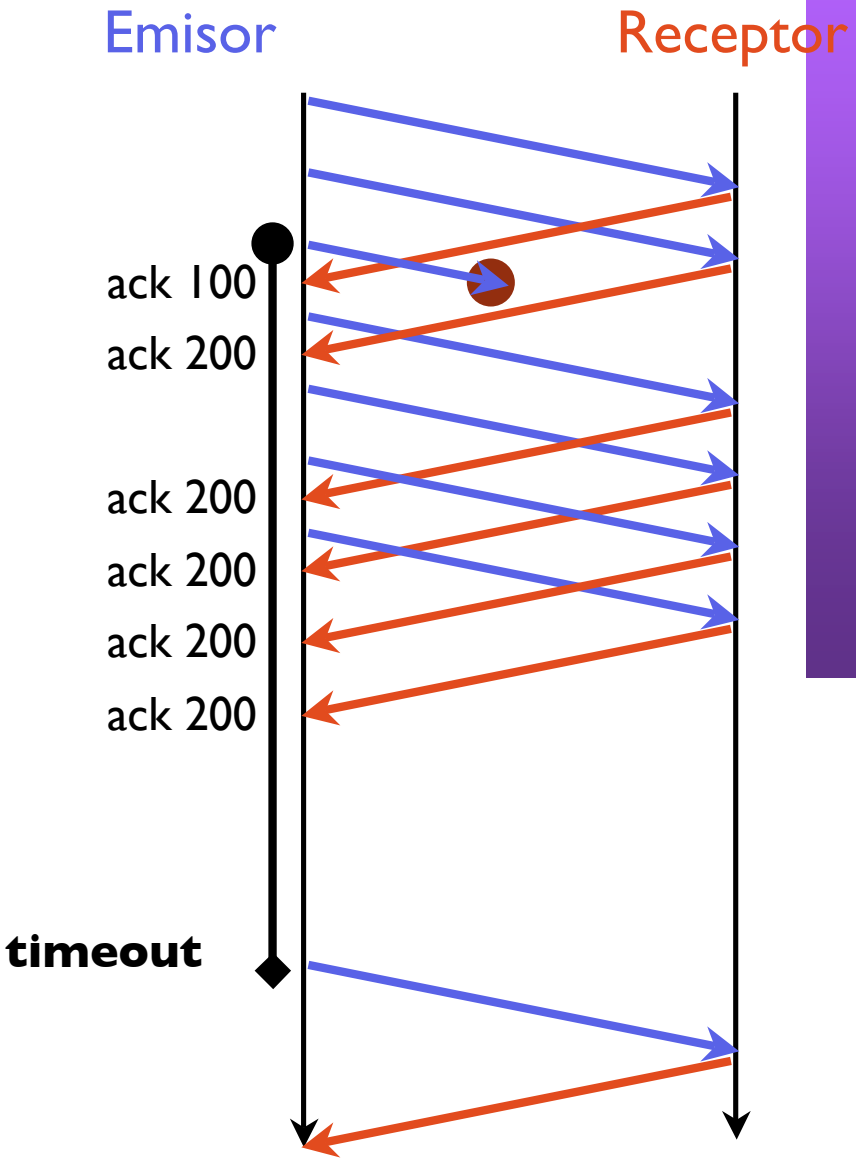
Acción: envía inmediatamente ACK

- **Llega segmento anterior a la ventana**

Acción: envía ACK

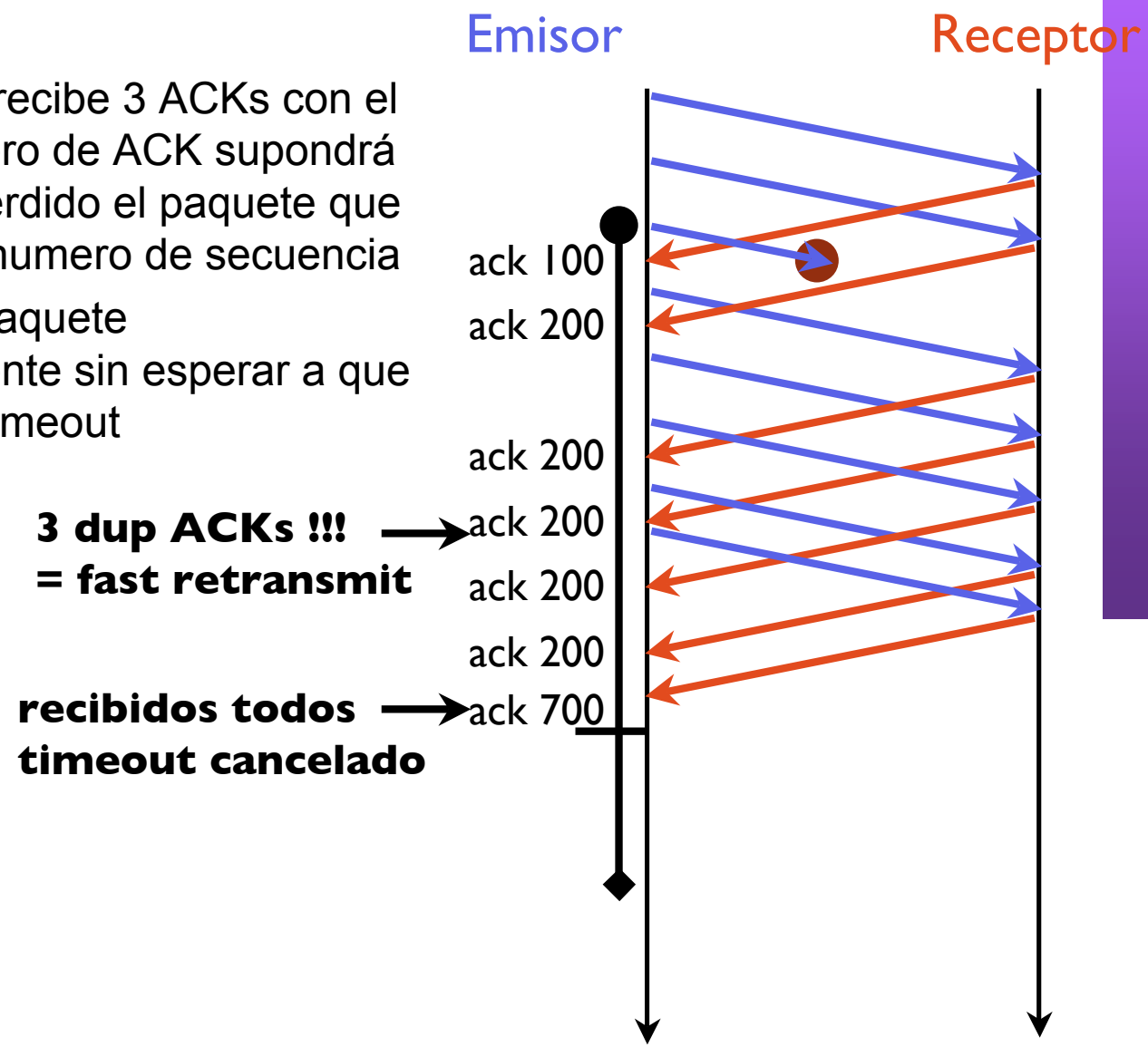
TCP: Fast retransmit

- 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



TCP: transporte fiable

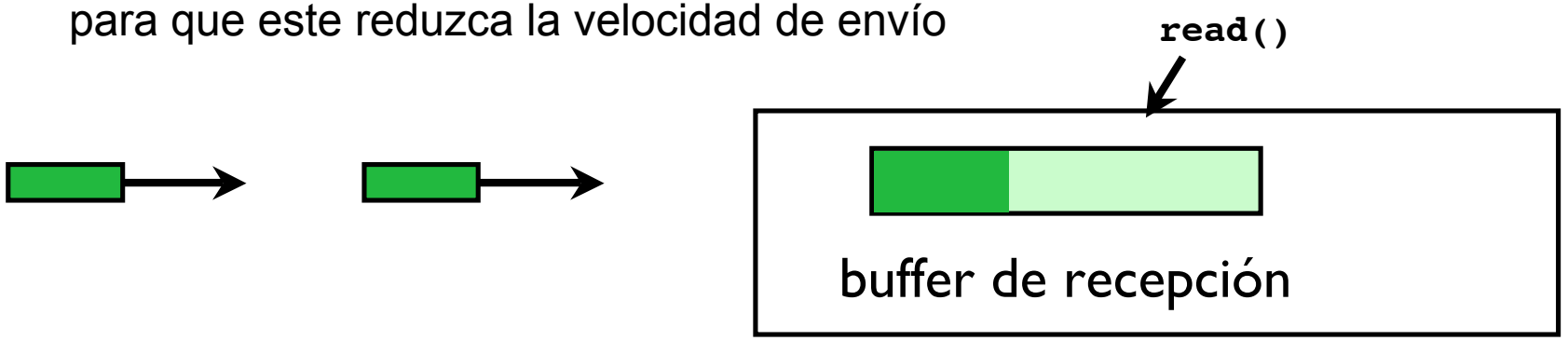
- Carácterísticas de Go back-N y SR
  - ACKs acumulados (y por byte individual)
  - Pero almacena paquetes fuera de secuencia en recepción

    Aunque no envía ACKs por paquete
  - Eso permite técnicas más sofisticadas de retransmisión al detectar duplicados (Fast retransmit)

- Y qué pasa con el control de flujo?

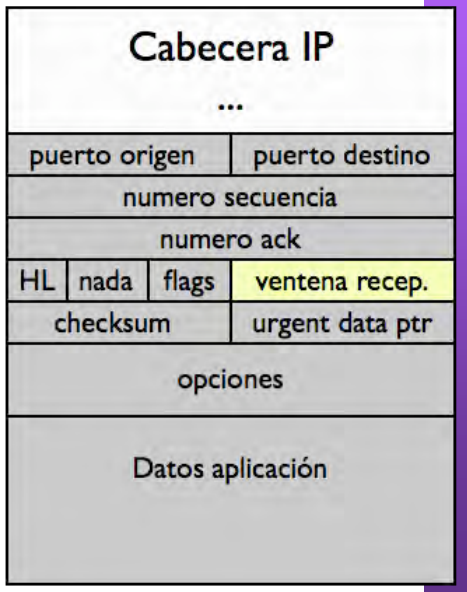
TCP: Control de flujo

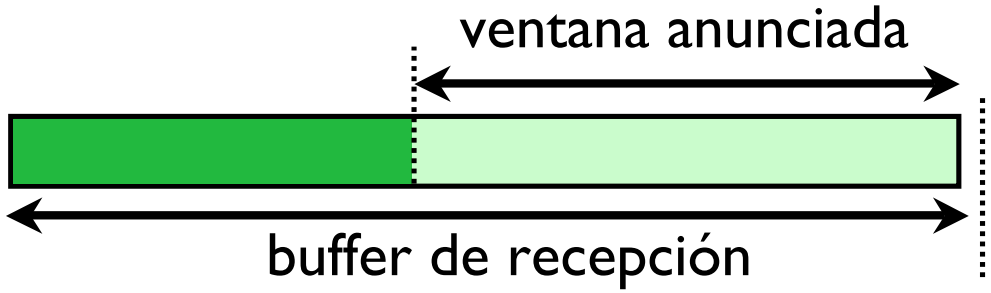
- El receptor de TCP tiene un buffer en el que TCP va colocando los datos que llegan.
  - Estos datos se le entregan al nivel de aplicación al hacer un read() sobre el socket
  - La aplicación puede ser lenta al leer los datos. Qué pasa si los datos llegan y no hay buffer?
  - Hace falta un mecanismo que ajuste la velocidad de los datos que llegan a la velocidad a la que lee la aplicación
- Este es el problema del **control de flujo**.
  - Es un problema general de los protocolos de comunicaciones
  - Normalmente se resuelve haciendo que el receptor sea capaz de enviar indicaciones al emisor de que su buffer se esta llenando para que este reduzca la velocidad de envío



TCP: Control de flujo

- TCP informa al emisor de cuanto buffer tiene libre en cada paquete que le envía !!
  - Esa es la función del campo ventana de recepción de la cabecera
  - En cada paquete el receptor anuncia cuantos datos es capaz de recibir
  - Este valor se utiliza como máximo numero de bytes que se pueden tener en la red sin recibir ACK. Máximo de la ventana deslizante





Ejemplo

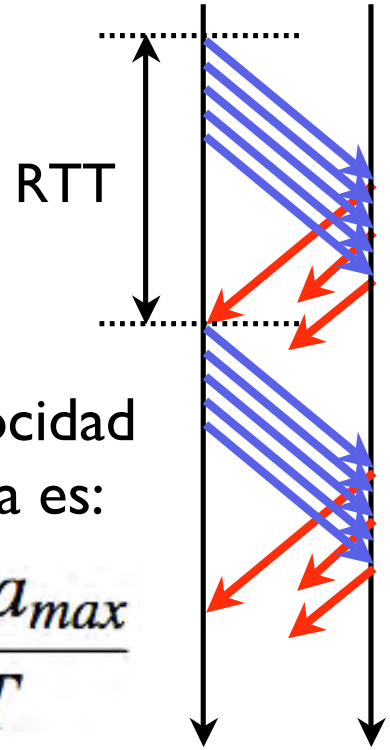
- De una transferencia de página web

```
1124207801.184011 IP 130.206.169.177.53611 > 130.206.166.105.80: . ack 1 win 65535
1124207825.463815 IP 130.206.169.177.53611 > 130.206.166.105.80: P 1:39(38) ack 1 win 65535
1124207825.464062 IP 130.206.166.105.80 > 130.206.169.177.53611: . ack 39 win 24616
1124207825.466289 IP 130.206.166.105.80 > 130.206.169.177.53611: P 1:291(290) ack 39 win 24616
1124207825.466784 IP 130.206.166.105.80 > 130.206.169.177.53611: . 291:1739(1448) ack 39 win 24616
1124207825.466915 IP 130.206.166.105.80 > 130.206.169.177.53611: P 1739:3187(1448) ack 39 win 24616
1124207825.511610 IP 130.206.169.177.53611 > 130.206.166.105.80: . ack 3187 win 63422
1124207825.512278 IP 130.206.166.105.80 > 130.206.169.177.53611: . 3187:4635(1448) ack 39 win 24616
1124207825.512382 IP 130.206.166.105.80 > 130.206.169.177.53611: . 4635:6083(1448) ack 39 win 24616
1124207825.512503 IP 130.206.166.105.80 > 130.206.169.177.53611: . 6083:7531(1448) ack 39 win 24616
1124207825.512626 IP 130.206.166.105.80 > 130.206.169.177.53611: P 7531:8979(1448) ack 39 win 24616
1124207825.711709 IP 130.206.169.177.53611 > 130.206.166.105.80: . ack 8979 win 57630
1124207825.712371 IP 130.206.166.105.80 > 130.206.169.177.53611: . 8979:10427(1448) ack 39 win 24616
1124207825.712474 IP 130.206.166.105.80 > 130.206.169.177.53611: . 10427:11875(1448) ack 39 win 24616
1124207825.712595 IP 130.206.166.105.80 > 130.206.169.177.53611: . 11875:13323(1448) ack 39 win 24616
1124207825.712723 IP 130.206.166.105.80 > 130.206.169.177.53611: . 13323:14771(1448) ack 39 win 24616
1124207825.712842 IP 130.206.166.105.80 > 130.206.169.177.53611: P 14771:16219(1448) ack 39 win 24616
1124207825.911783 IP 130.206.169.177.53611 > 130.206.166.105.80: . ack 16219 win 50390
```

- Conforme recibo datos se va llenando el buffer

TCP: Control de flujo

- El campo para anunciar ventana sólo tiene 16 bits
  - Solo puede anunciar 65KBytes !!! (el numero de secuencia direcciona 4GB)
  - Podían ser suficientes en los primeros tiempos de TCP...
- Consecuencias
  - no hay que preocuparse del overlflow de numero de secuencia
  - si hay que preocuparse por las velocidad de transferencia



La máxima velocidad de transferencia es:

$$v = \frac{ventana_{max}}{RTT}$$

En el ejemplo de 1Gbps y 30 ms

$$v = \frac{65535bytes}{30ms} \approx 17.5Mbps$$

Al menos llegamos al 17%

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 retransmidion más sofisticados con delayed ACKs y Fast Retransmit
  - Control de flujo anunciando el tamaño máximo de la ventana deslizante
  - Control de congestión que lo dejamos para cursos superiores

Próxima clase:

- Otra red de conmutación de paquetes que no es TCP/IP: Redes ATM