Transporte fiable

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

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

Temario

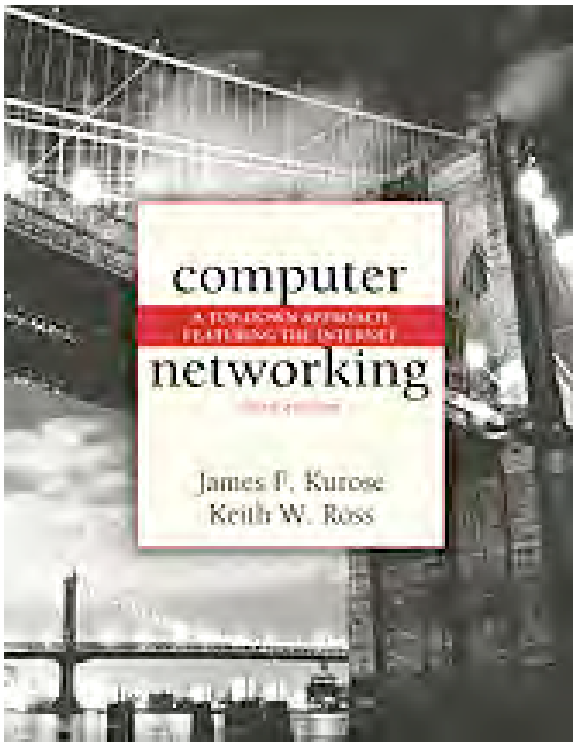
1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

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: principios básicos)**
     - Control de flujo
     - 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



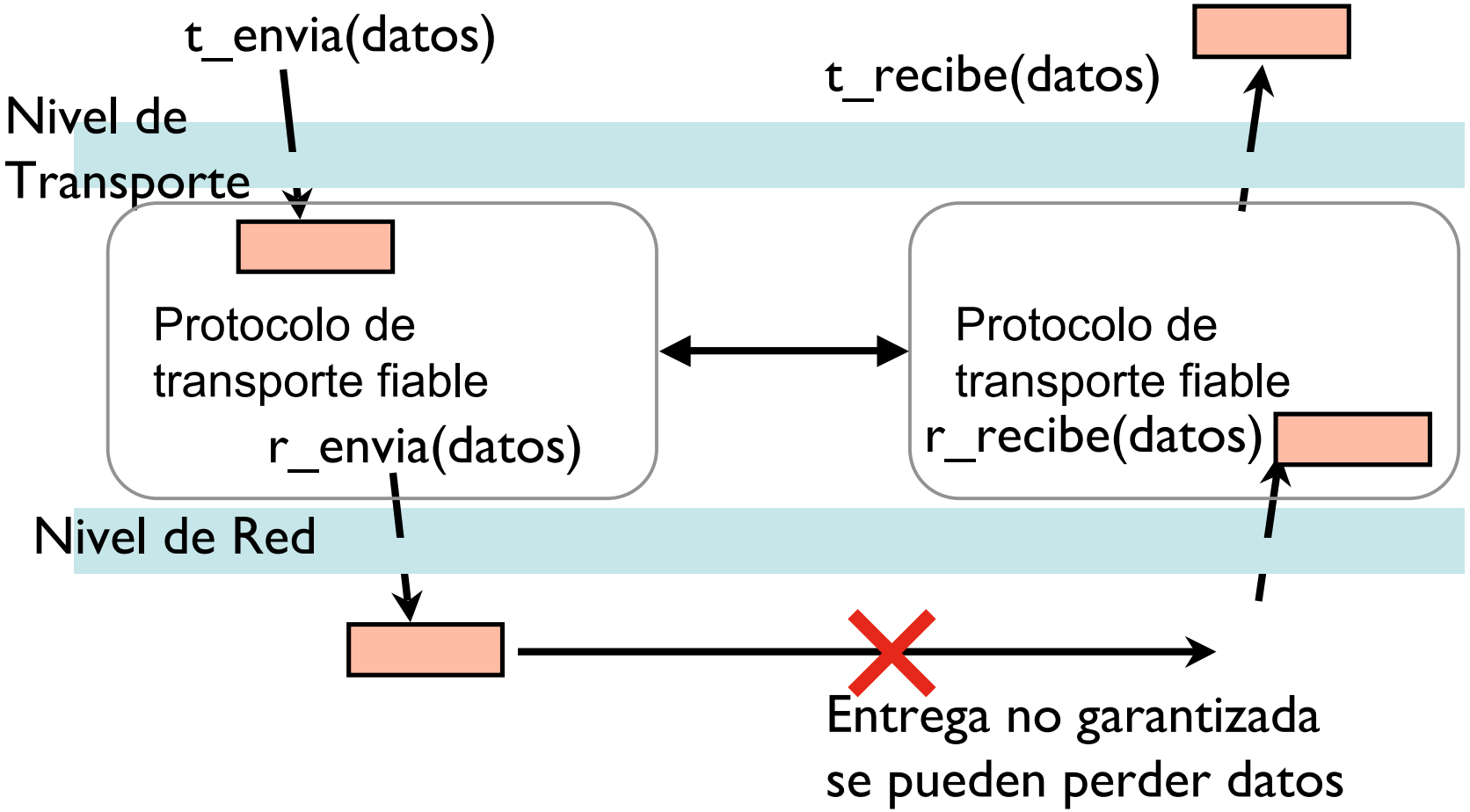
Transporte fiable

- Si hubiera un “Top ten” problemas de redes el transporte fiable sería un buen candidato para el primer puesto

  Kurose

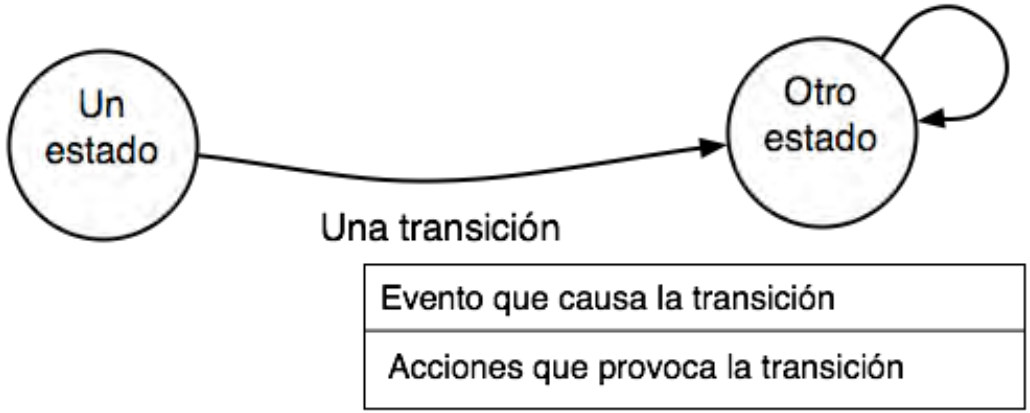
Transporte fiable

- ¿Se puede conseguir un transporte fiable sobre un nivel de datagramas de entrega no fiable?



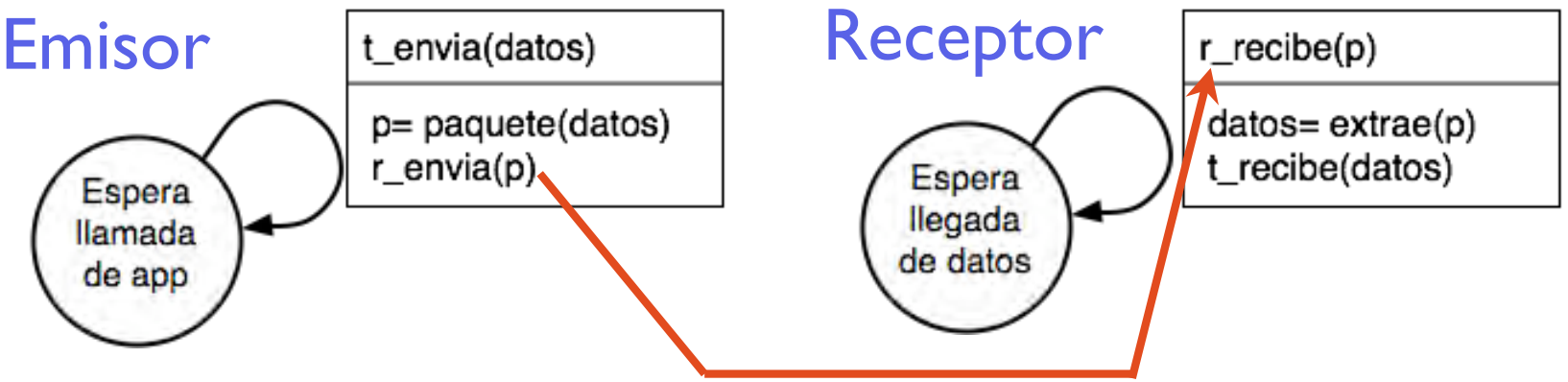
Protocolo de transporte fiable

- Descripción protocolo (=programa) con máquinas de estados finitos
  - Eventos que pueden ocurrir
  - Acciones como resultado de esos eventos
- Emisor y receptor son diferentes programas y estan en general en distinto estado (normalmente ni siquiera su conjunto de estados es el mismo)



Protocolo de transporte fiable

- Ejemplo: protocolo de transporte sobre un nivel de red fiable
- Diagrama de estados de emisor y receptor

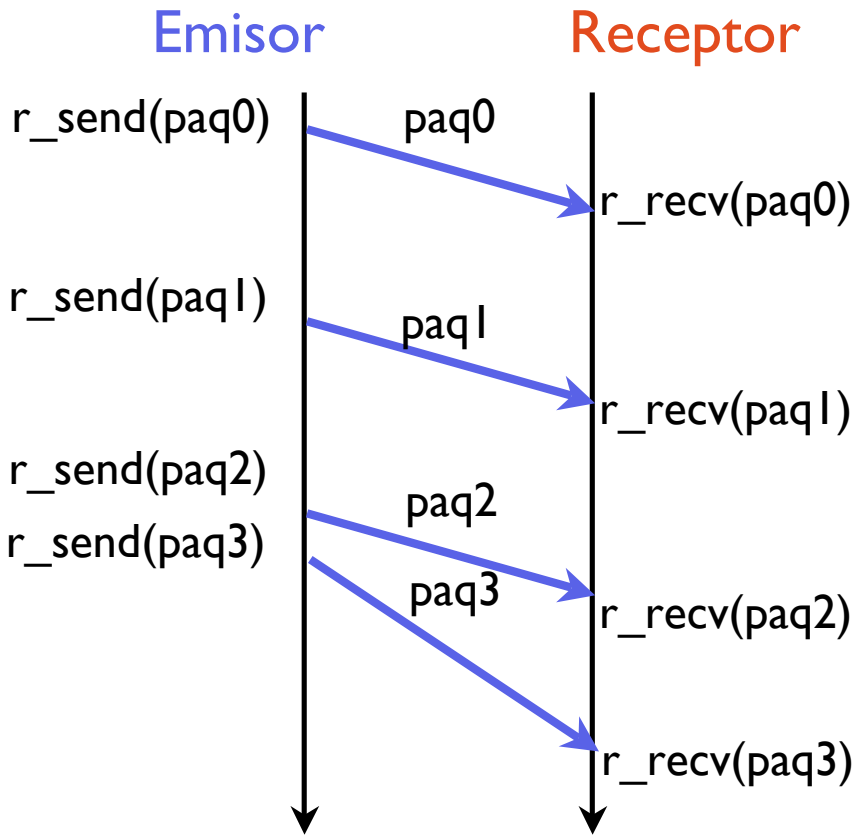


red fiable:

r_envia(p) siempre causa un r_recibe(p)

Ejemplo

- Todo lo que envío llega
- La red puede retrasar los paquetes pero acaba entregandolos todos
- Algun problema si los entrega siempre pero en desorden??



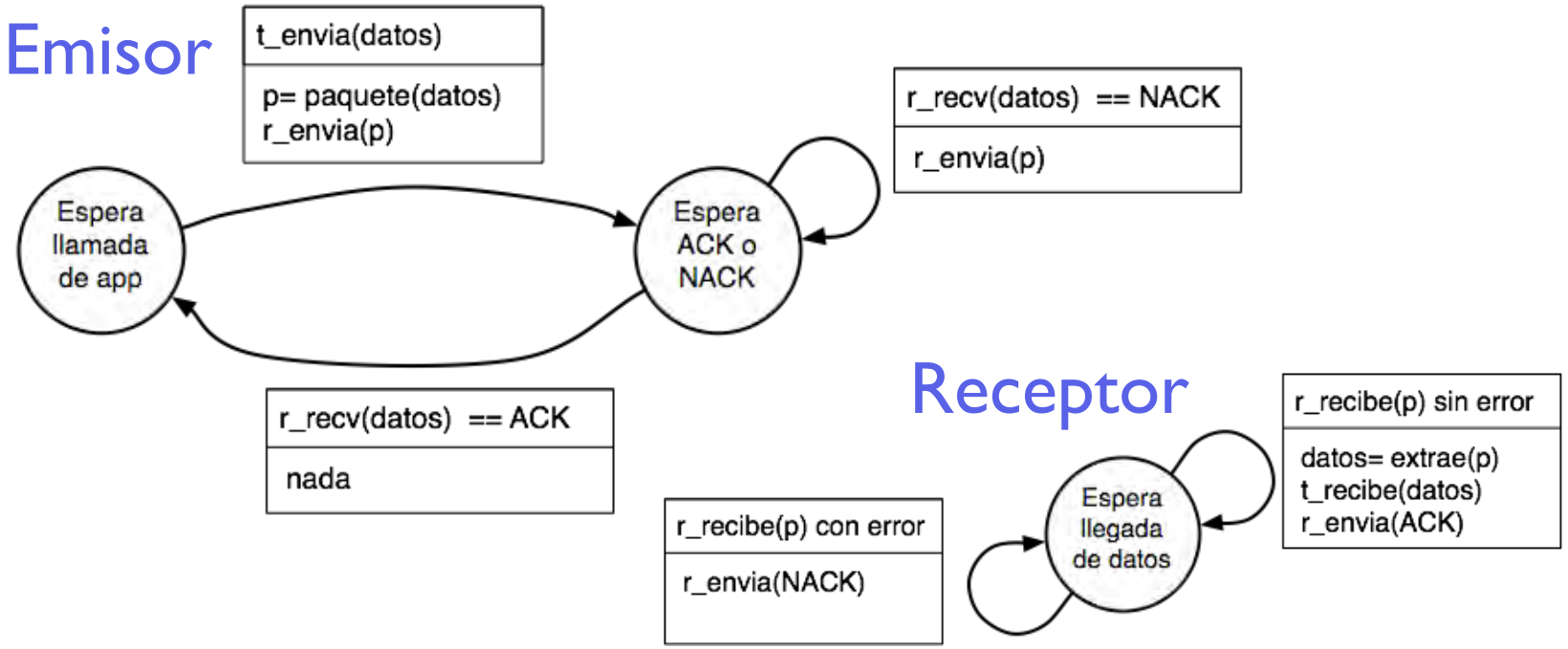
Operación normal

Errores de bit

- Pero... el nivel de red puede cambiar bits (probabilidad de error)
- Cambios necesarios en el protocolo de transporte
  - **Detección de errores**
    - Uso de checksum
  - **Comunicación de fallos al emisor**
    - **ACK** (acknowledgement): avisar al emisor de los paquetes que recibimos.
    - **NACK** (negative acknowledgement): avisar al emisor de los paquetes que no recibimos
  - **Reenvío de paquetes**
- El protocolo de transporte fiable debe retransmitir automaticamente los errores. Esto se conoce tipicamente como ARQ (Automatic Reqpeat reQuest)

Protocolo de transporte fiable

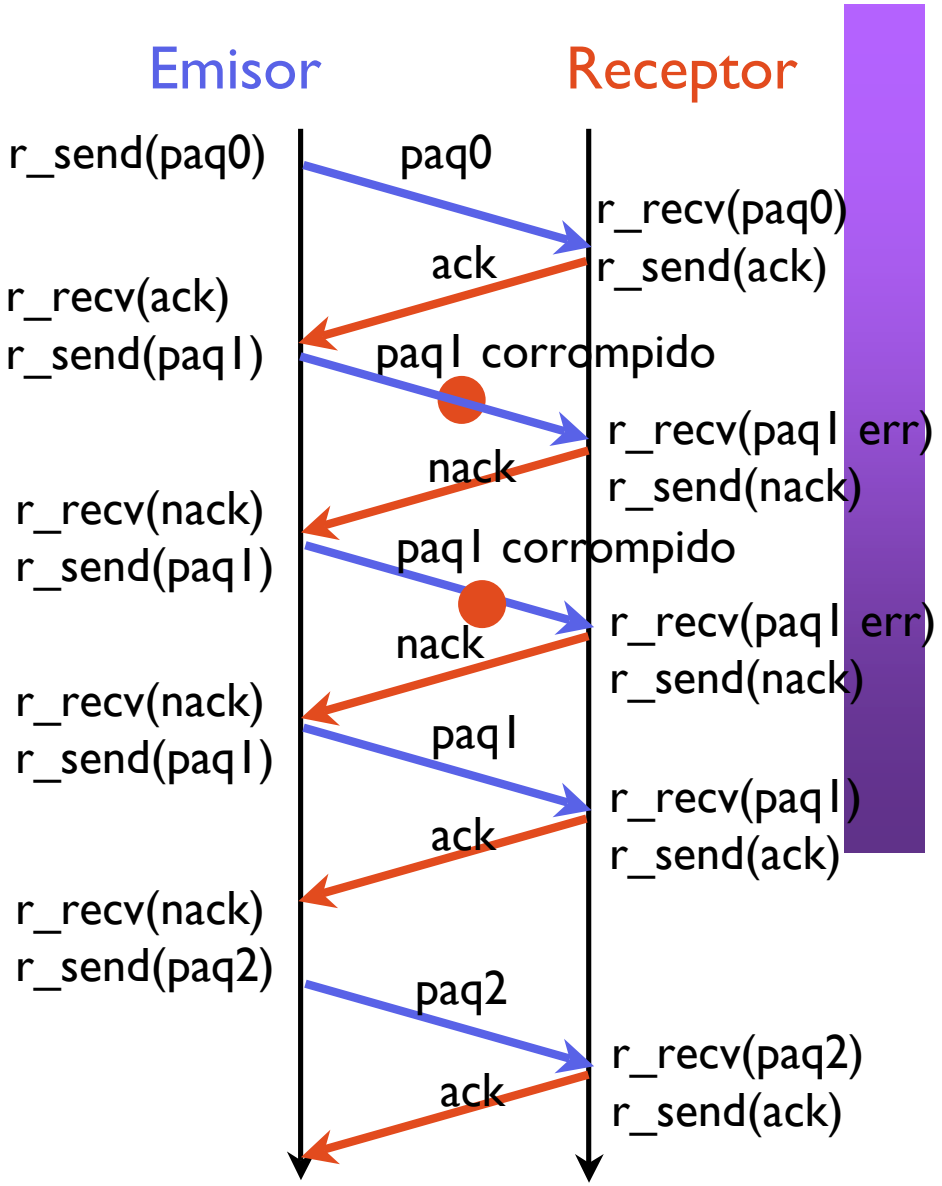
- Para un canal con errores de bits
  - Usamos detección de errores con checksum/CRC
  - Informamos al emisor de si llegan o no



- Más conocido como **Stop-and-Wait**

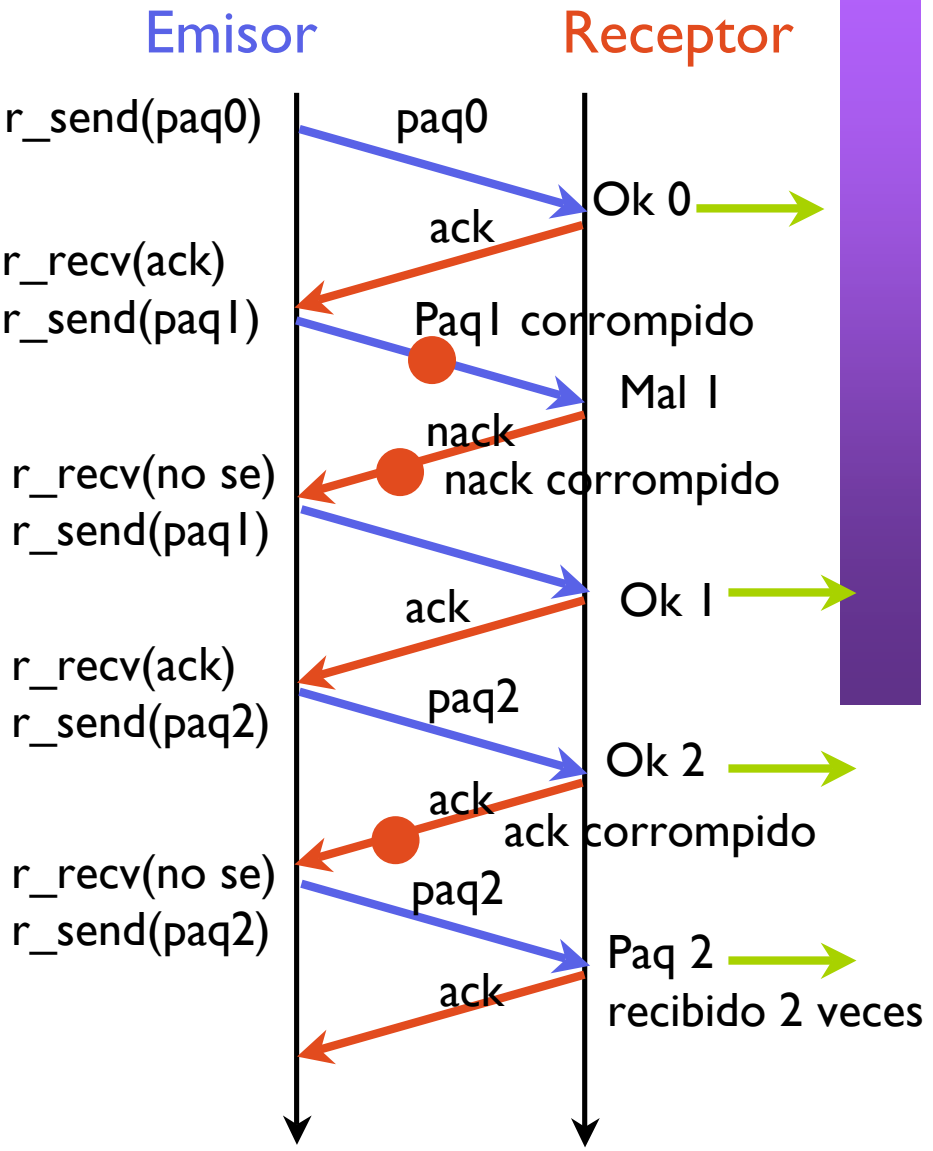
Stop-and-wait

- El emisor controlado por el receptor
  - ACK (recibido OK manda otro)
  - NACK (recibido mal manda otra vez el mismo)
  - Mientras no me dice nada no envío
- De hecho esto puede considerarse también control de flujo (el emisor envia cuando el receptor le da permiso) = regulación de flujo por el receptor



Problemas con stop-and-wait

- ¿Qué pasa si hay un error en la transmisión del ACK o NACK?
- Soluciones complican el protocolo
  - Detección de errores para ACK y NACK?
  - Checksums que permitan no solo detectar sino corregir errores?
  - Reenviar los datos si no entiendo el ACK/NACK ??
    - **Nuevo problema: paquetes duplicados**



Solución

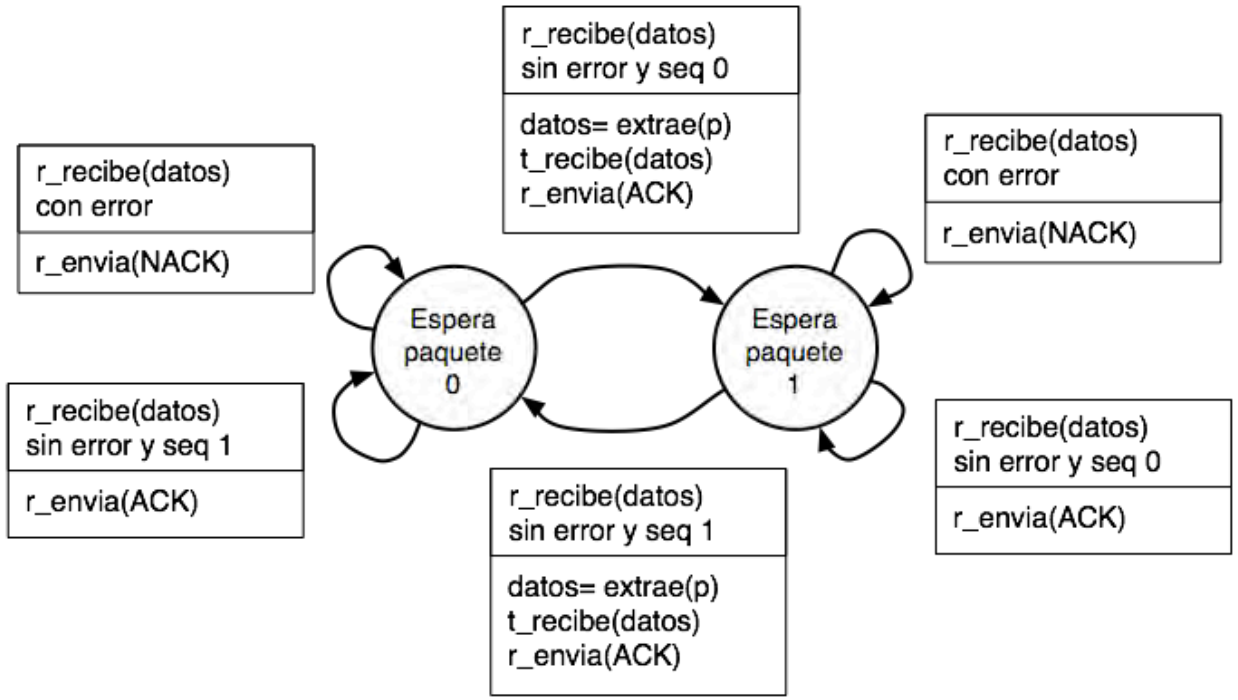
- Los protocolos más usados utilizan contra esto numeros de secuencia del paquete
- El paquete va etiquetado con un numero de secuencia que permite confirmralo/rechazarlo indicando cual
- El numero de secuencia es un campo del paquete por lo que podrá tener una serie finita de valores
- Aunque es fácil asignar bits para que el numero de secuencia pueda crecer mucho antes de dar la vuelta, veamos primero las bases con numeros de secuencia en rangos limitados

Protocolo con número de secuencia

- 1 bit para número de secuencia

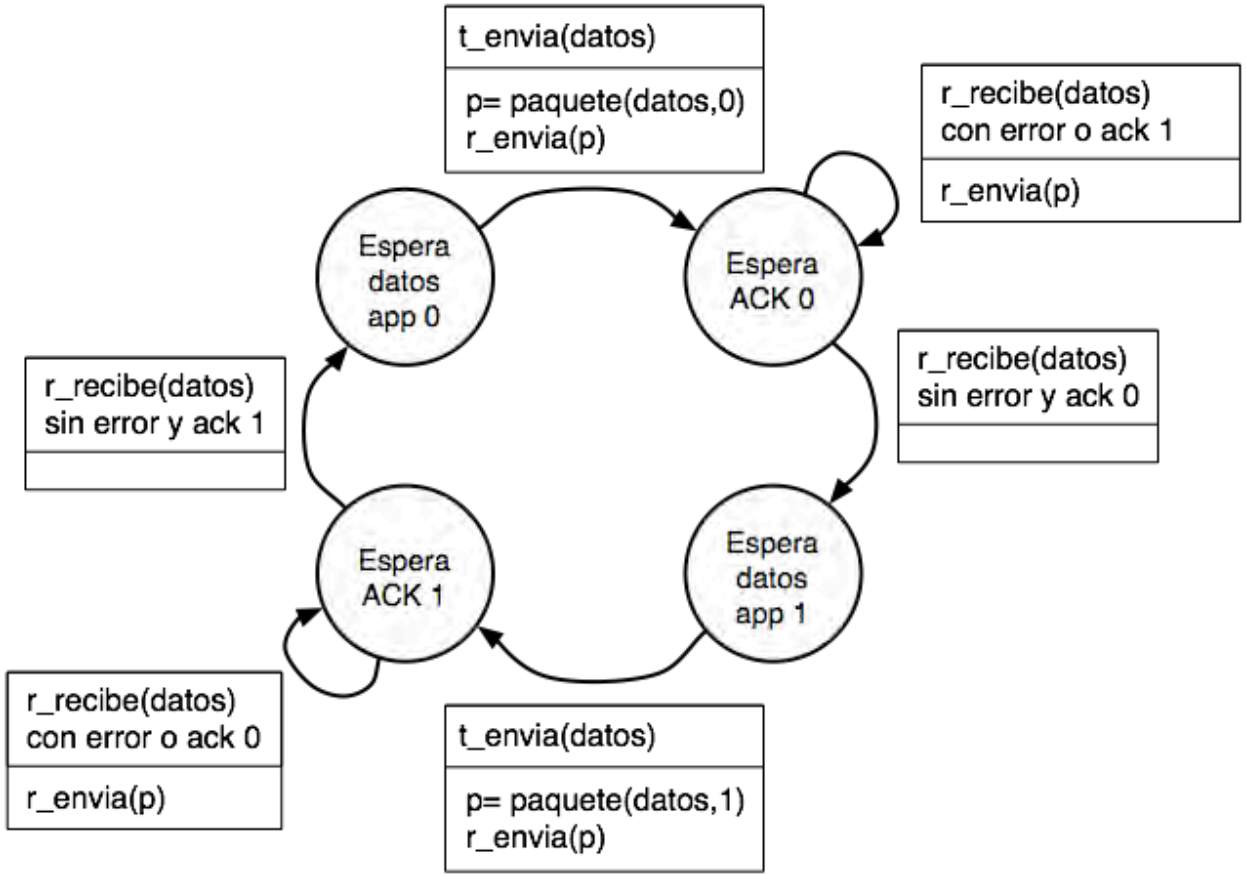
  Cada paquete de datos es secuencia 0 o 1
  - Si llega el que esperamos mandamos ACK y lo entregamos
  - Si llega el que no esperamos?

    mandamos ACK pero no son datos nuevos



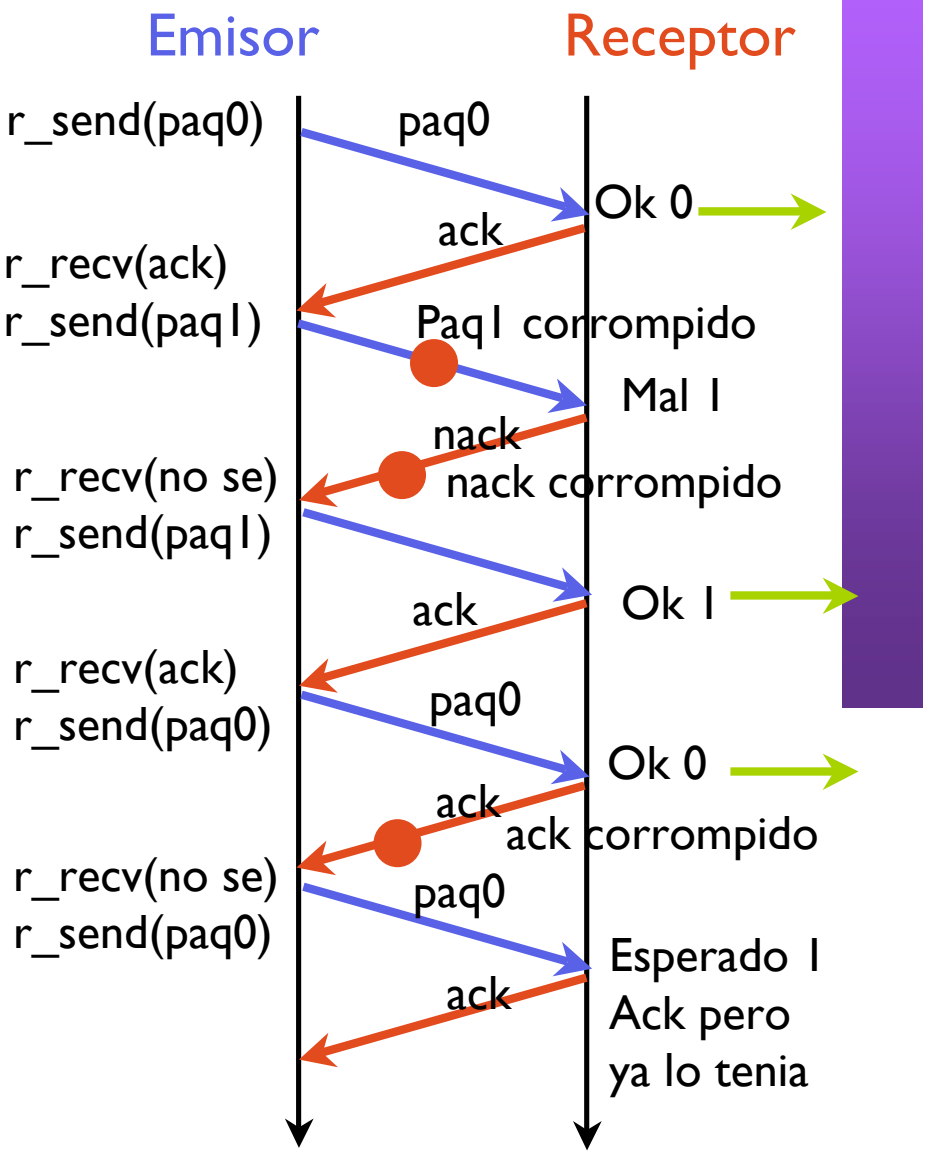
Protocolo con número de secuencia

- Estados del emisor



Ejemplo

- El receptor solo entrega a la aplicación el paquete correcto
- En lugar de enviar ACK o NACK podria enviar cual espero
  - ACK + 0 = estoy esperando el 0
    (igual lo llamabais RR0
    Ready to receive 0)
  - ACK + 1 = estoy esperando el 1



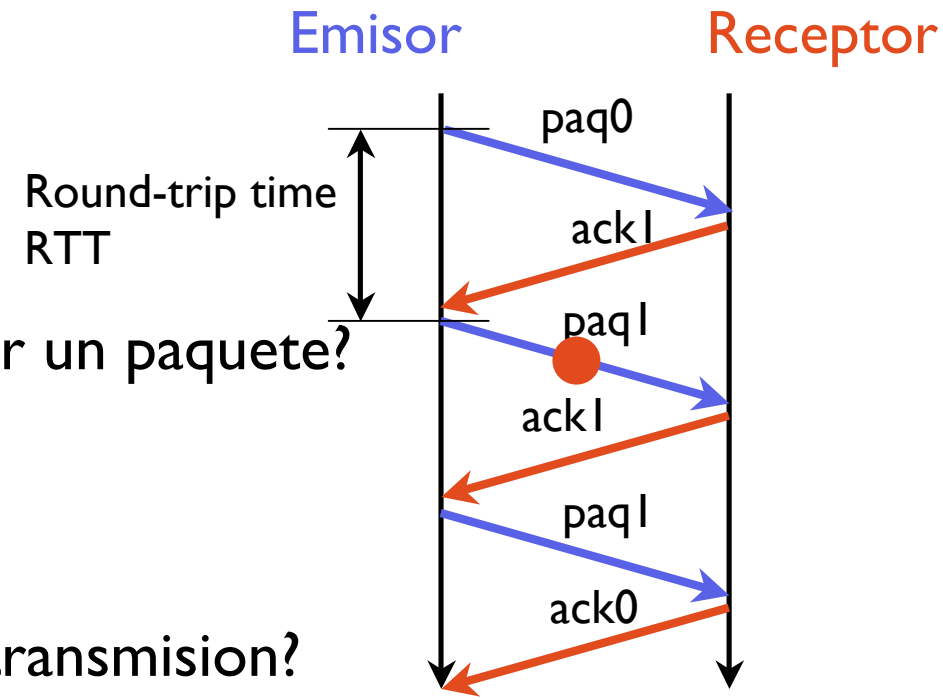
Hasta ahora

- Protocolo
  - Stop and wait
  - Con numeros de secuencia para no entregar duplicados
  - Con ACK que indica cual es el dato que espero
- Garantiza fiabilidad sobre un canal con errores de bits
- Problemas
  - ¿Y si se pueden perder paquetes?
  - Cómo de rápido es el protocolo

Eficiencia

- Cuanto se tardan en transferir s bytes con un protocolo de este tipo?
  - Dividimos en paquetes de tamaño c

s/c paquetes



Cuanto se tarda en enviar un paquete?
Si no hay errores 1 RTT
Si hay errores 2 RTTs?

Y si hay errores en la retransmision?

Tiempo transmisión de un paquete

- Tiempo para transferir 1 paquete

  si la probabilidad de que un paquete se pierda es p

  1 RTT con probabilidad (1-p)

  2 RTT con probabilidad (1-p)*p

  3 RTT con probabilidad (1-p)*p^2

  n RTT con probabilidad (1-p)*p^{n-1}

  (v.a. Distribucion geométrica)

  el numero medio de RTTs se deja como ejercicio

  pero es 1/(1-p) RTTs

Prestaciones

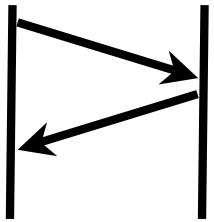
- Luego s bytes se transfieren en
  t = s/c * 1/(1-p) * RTT
  y la velocidad de transferencia es
  s/t = c * (1-p) /RTT
- Y cuanto es eso en un caso real?
- Si elegimos tamaños de paquetes muy grandes hay que mandar menos, pero la probabilidad de perdida de un paquete es mayor
- Si elegimos paquetes pequeños hay que esperar un RTT al menos para mandar cada paquete

Ejemplo

- Ejemplo:
  Enlace de 1Gbps con un retardo de 15ms (4500Km), paquetes de 1000 bytes
  A que velocidad puedo enviar?

  Suponiendo sin errores

RoundTripTime =30ms



$$v = \frac{tam}{RoundTripTime} = \frac{8000bits}{.03s} = 266Kbps$$

0.026% !! :-(

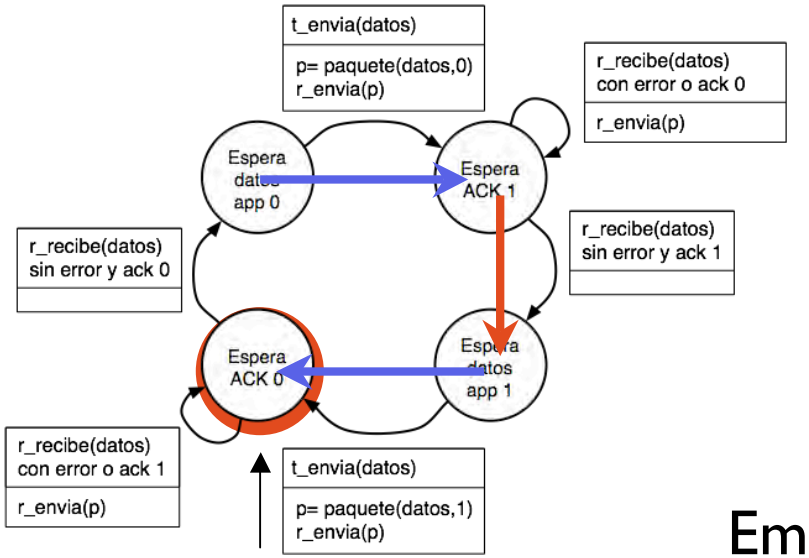
Pérdidas de paquetes

- El nivel de transporte recibe los paquetes que entrega el nivel de red
  - El nivel de red puede no garantizar la entrega de paquetes.
- Puede ser que un paquete entregado en el nivel de red del emisor nunca se entregue en el nivel de red del receptor
- Cómo afecta esto al protocolo anterior?

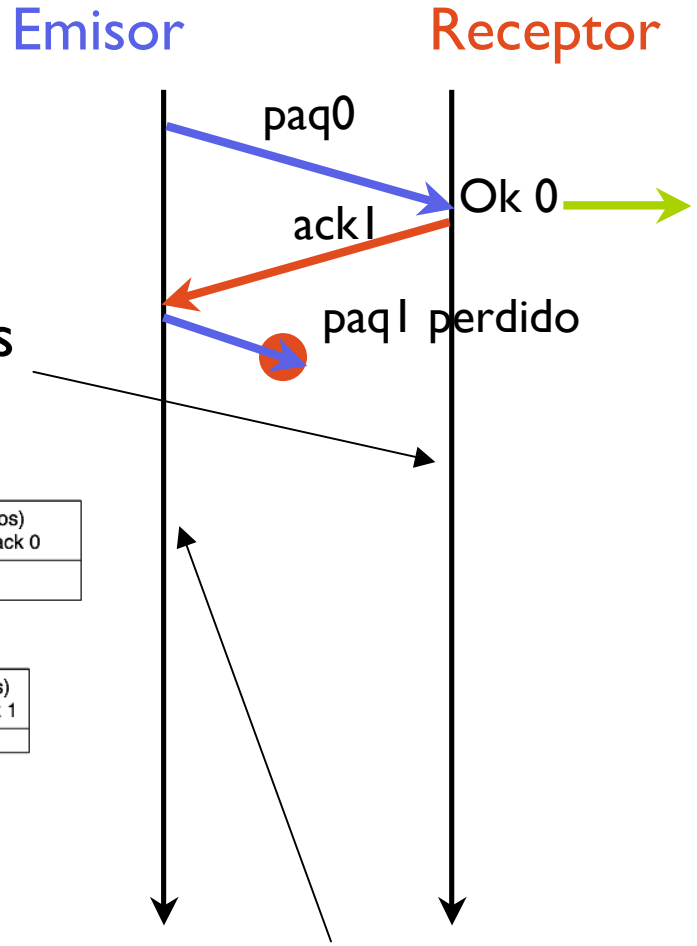
Pérdidas de paquetes

- Si se pierde un paquete el emisor se queda bloqueado en un estado

Receptor sólo manda ACKs cuando le llega algo



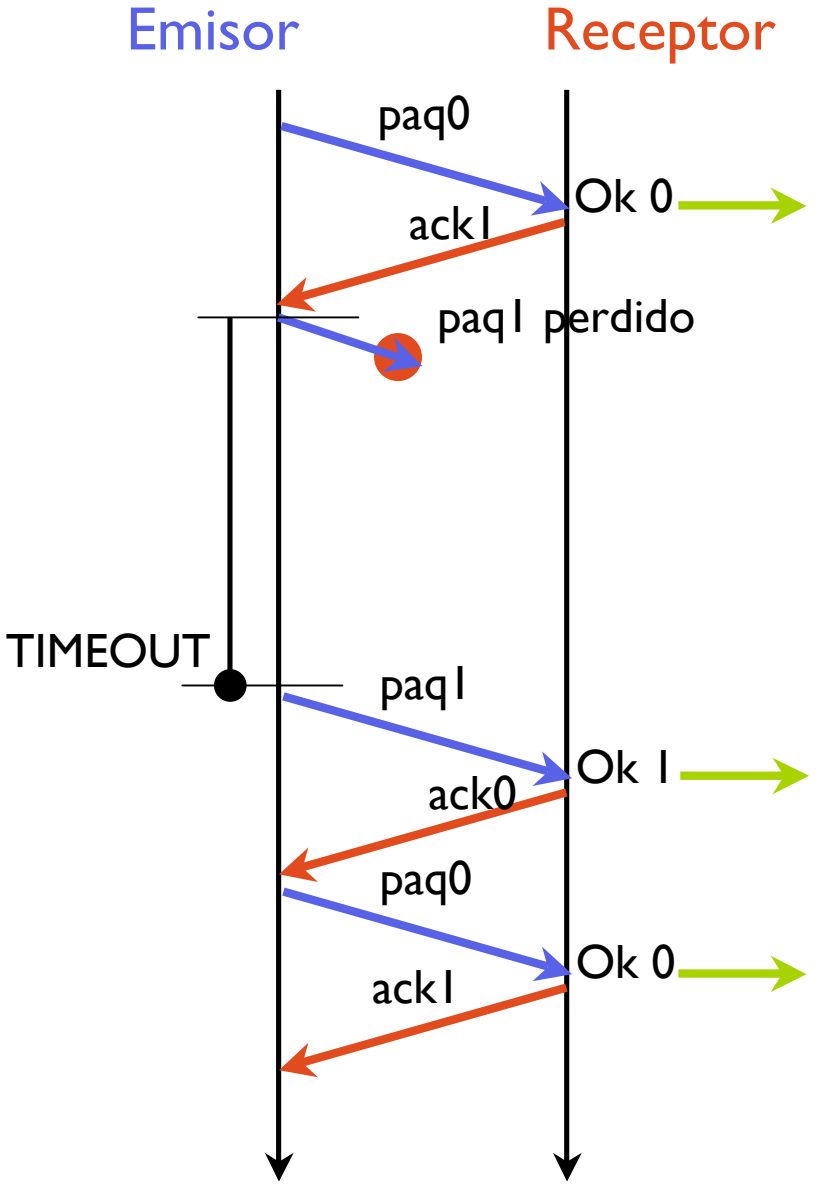
Sin recibir ACKs no puedo salir de este estado



Emisor no puede enviar hasta recibir el ACK

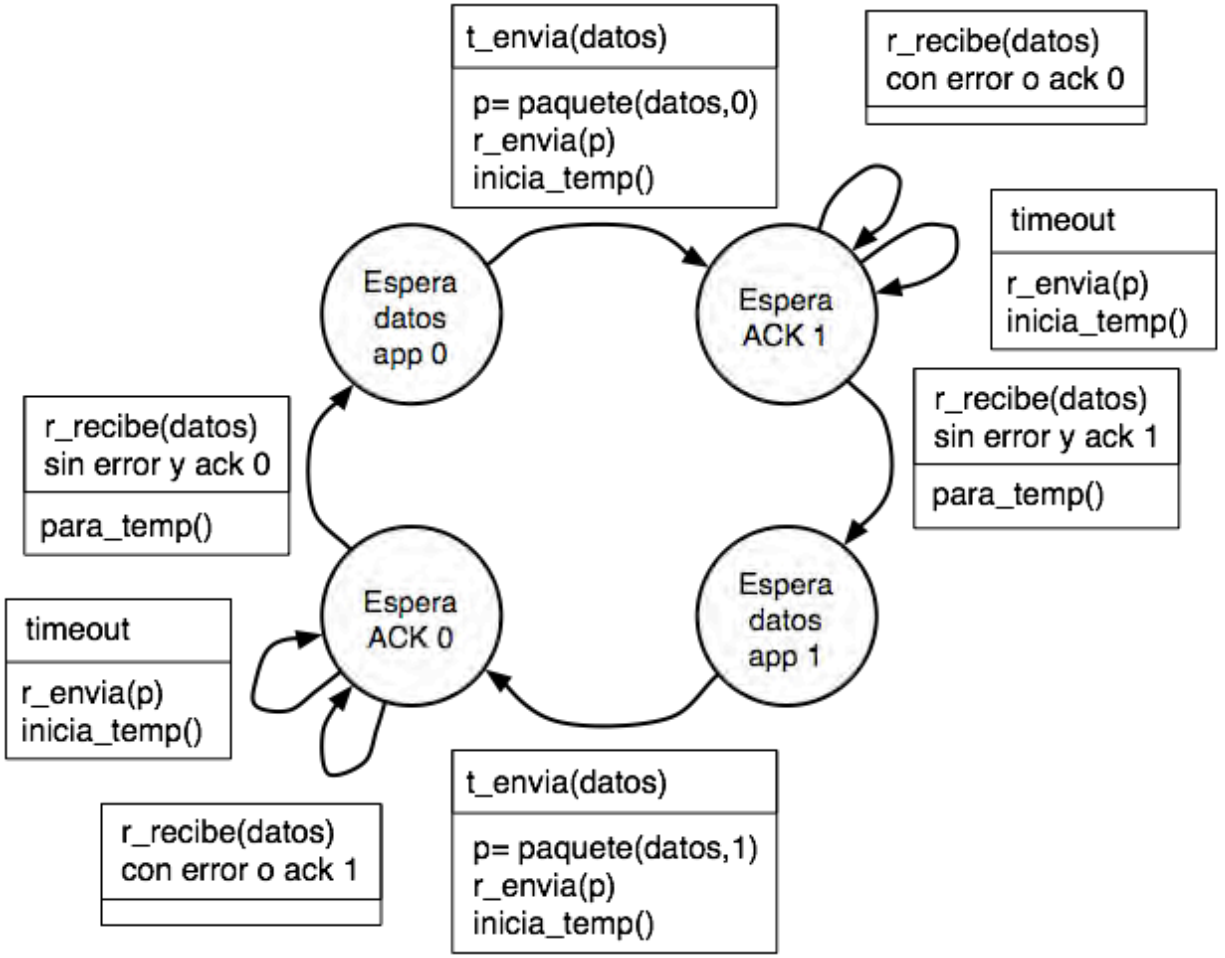
Pérdidas de paquetes

- Si se pierde un paquete el emisor se queda bloqueado en un estado
- Para romper el bloqueo usamos un temporizador en el emisor
  - Al enviar un paquete de datos ponemos en marcha un temporizador
  - Si transcurrido un tiempo, no se ha recibido ACK (TIMEOUT), reenviamos el paquete
- El receptor no se modifica

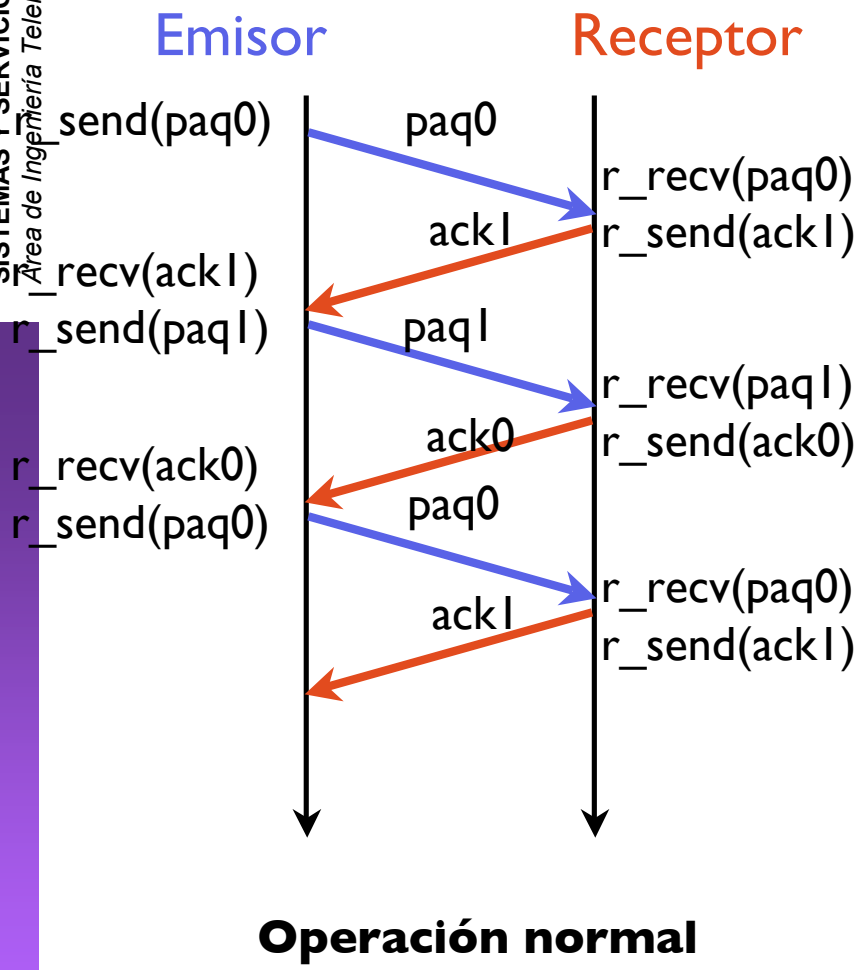


Protocolo con timeout

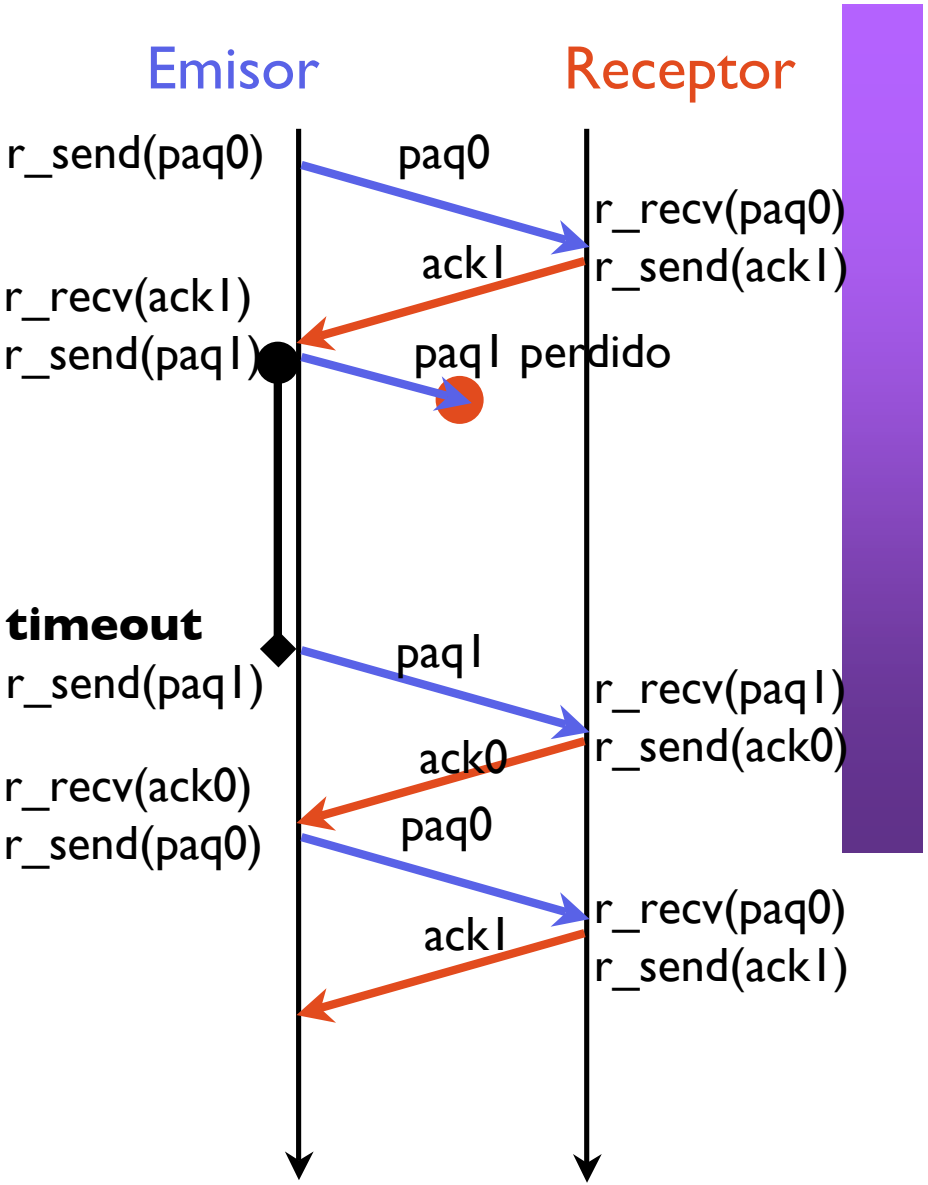
- Emisor con retransmisión por timeout



Ejemplos

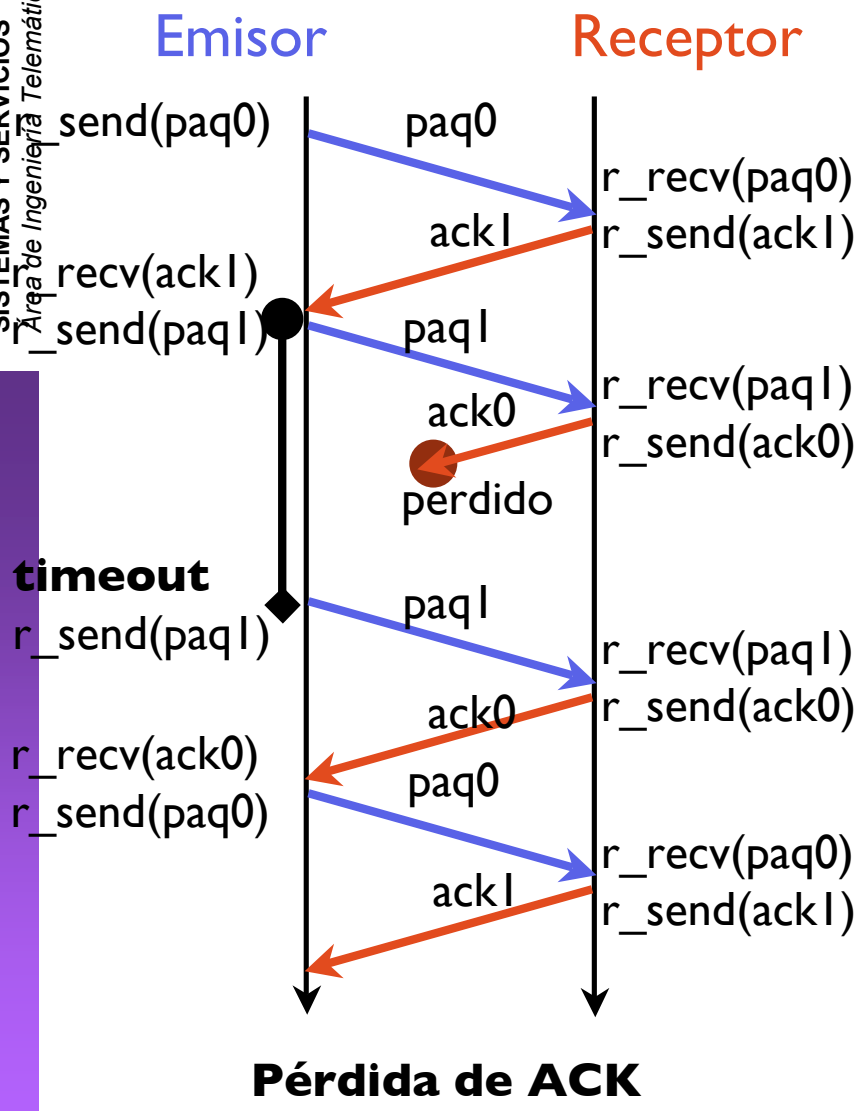


Operación normal

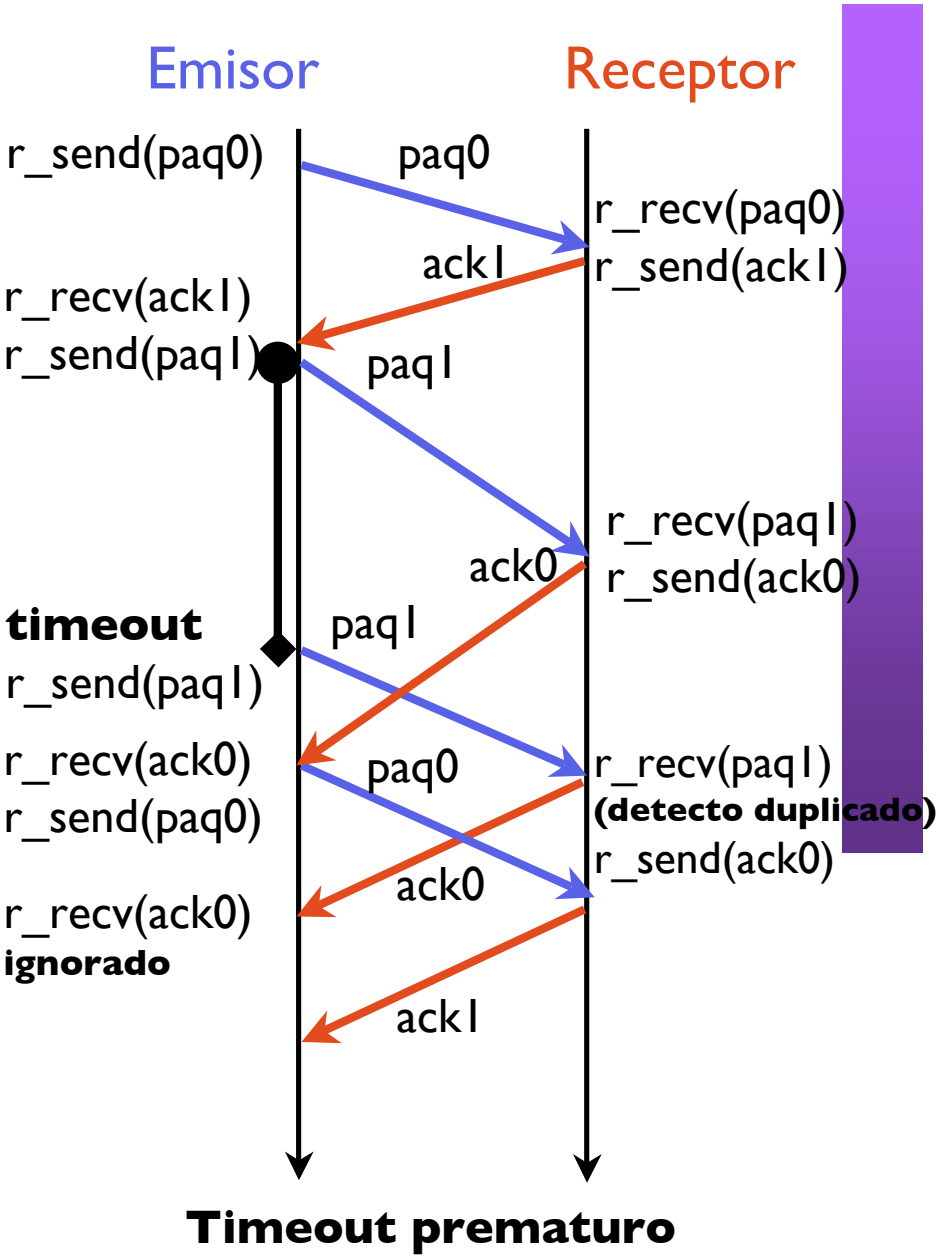


Pérdida de paquete

Ejemplos



Pérdida de ACK



Timeout prematuro

Prestaciones

- El protocolo anterior es fiable sigue siendo muy poco eficiente
- Ejemplo:
  Enlace de 1Gbps con un retardo de 15ms (4500Km), paquetes de 1000 bytes
  A que velocidad puedo enviar?
- Si los paquetes se pierden con probabilidad p

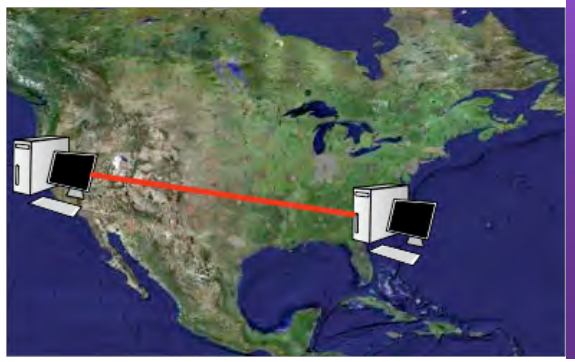
  RTT con probabilidad (1-p)

  RTT+TO con probabilidad (1-p)*p

  RTT+2TO con probabilidad (1-p)*p^2

  …

  RTT+n*TO con probabilidad (1-p)*p^{n-1}



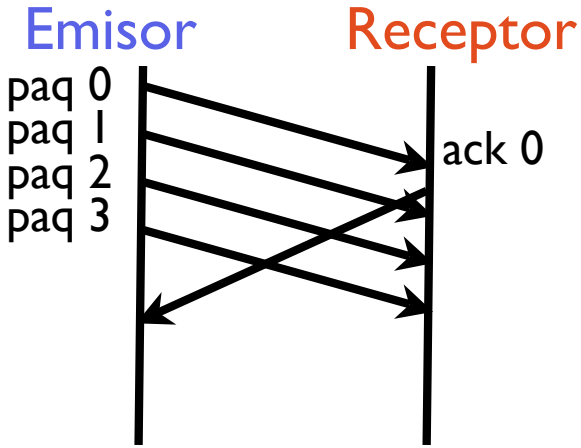
- El timeout se procura elegir del orden del RTT
  - Mayor implica que reaccionamos despacio a los errores
  - Menor implica que se retransmiten paquetes que no hacia falta
- Las prestaciones son parecidas al anterior, pero con p probabilidad de perdida del paquete. Normalmente en Internet

  P(perdidadelpaqete) >> P(corrupciondelpaquete)

  Y normlamente la modificación la detecta el nivel de enlace y lo descarta = pérdida

Protocolos más eficientes

- Para aumentar la eficiencia, se envían varios paquetes (ventana de paquetes) mientras llega el ACK
  Varios paquetes en la red por confirmar
  - Se usan más números de secuencia que 0 y 1
  - Emisor y receptor necesitarán buffer para varios paquetes
  - Varias políticas para reaccionar a los errores
    - Go-Back N
    - Selective repeat



Go back-N

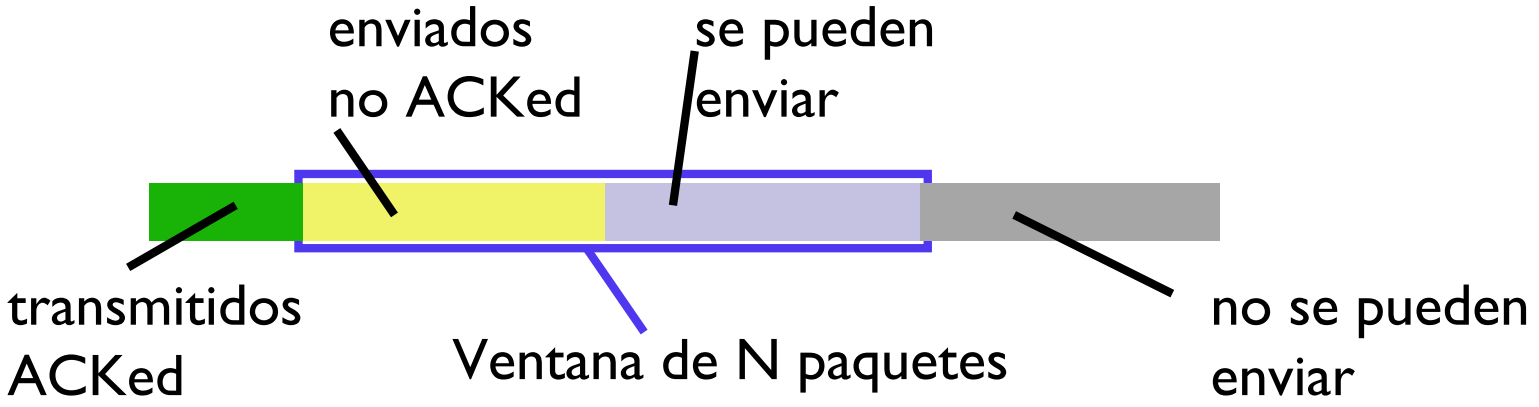
- Se utiliza número de secuencia en el paquete
- Se permite una ventana de N paquetes sin confirmar
- **Cada ACK confirma todos los paquetes anteriores (cumulative ACK)**
- Timeout al iniciar la ventana
- **Si caduca el timeout se retransmite la ventana**

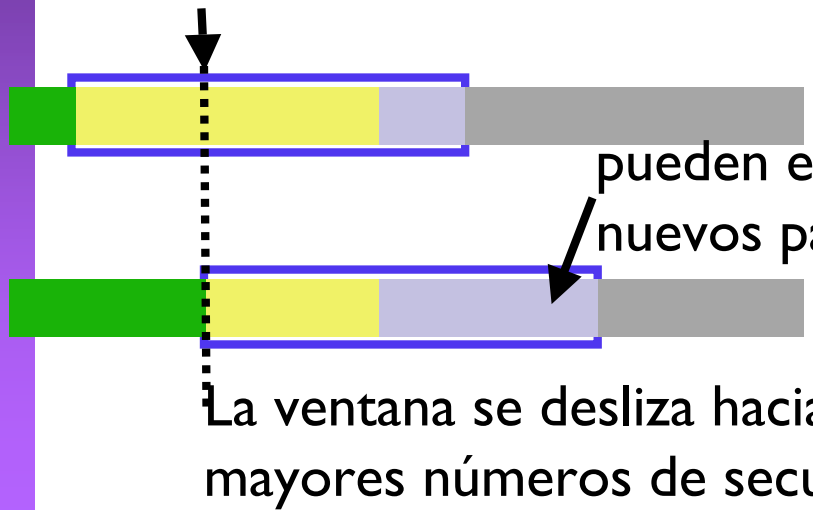
Eventos en el emisor (Go back-N)

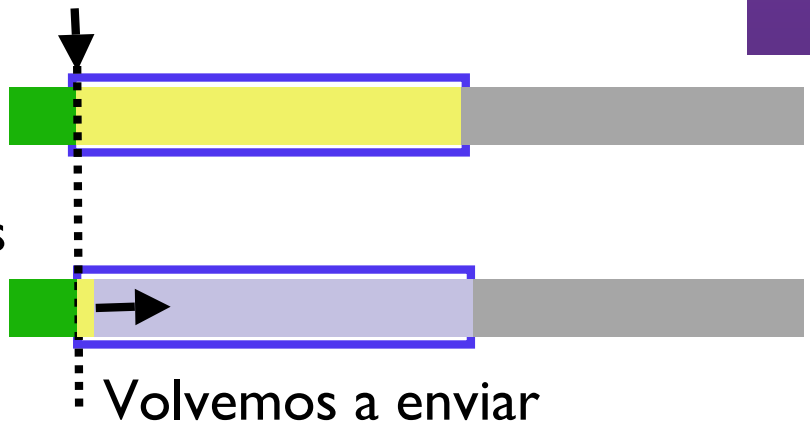
- **Recibo un ACK**
  - Avanza la ventana hasta la posición confirmada
  - Envía los siguientes paquetes hasta llenar la ventana
  - Reinicia el timeout si envías paquetes nuevos
- **Caduca el timeout del primer paquete de la ventana**
  - Reenvía todos los paquetes de la ventana
  - Reinicia el timeout

Eventos en el emisor (Go back n)

Ventana deslizante





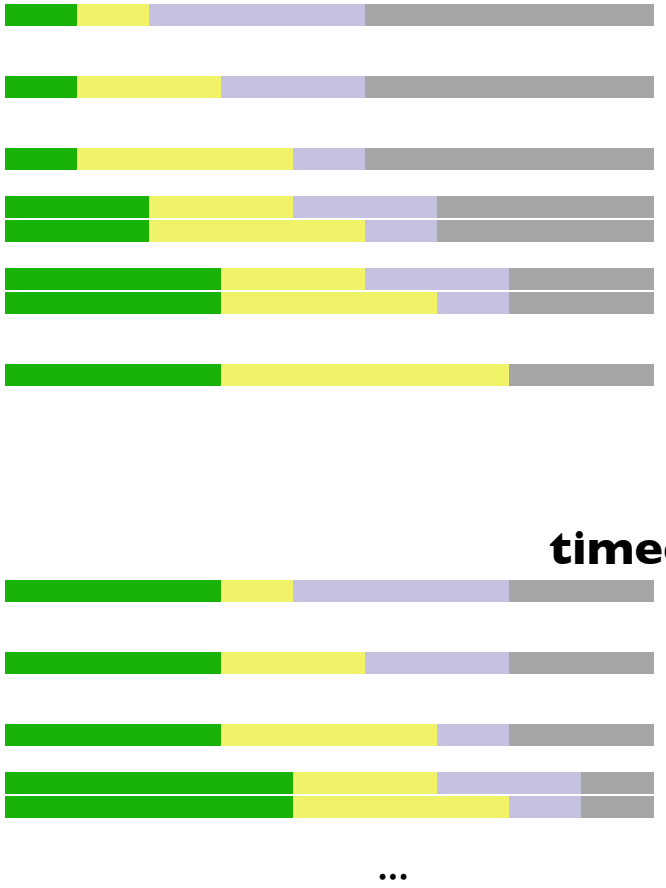


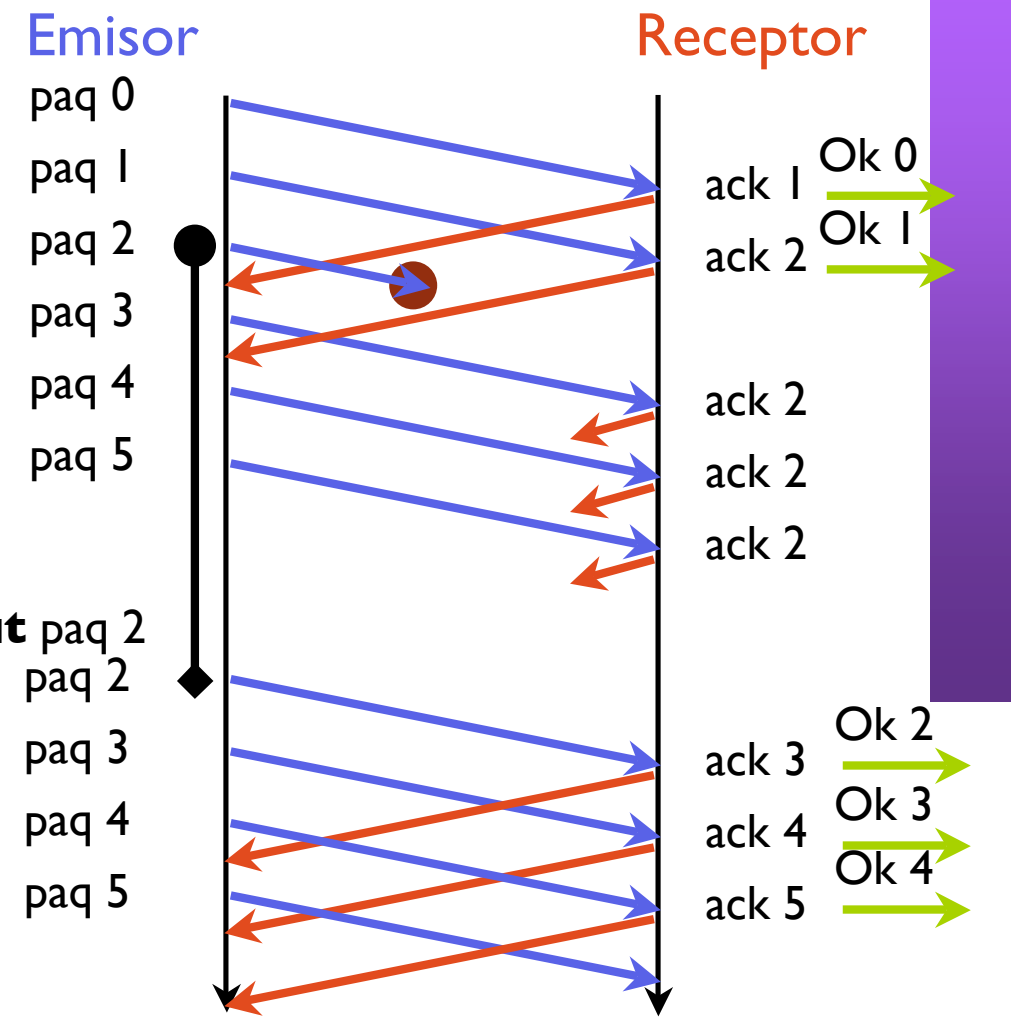
Eventos en el receptor (Go back-N)

- **Llega el paquete esperado**
  - Envía un ACK indicando el siguiente esperado
  - Entrega datos al nivel superior
- **Llega otra paquete**
  - Envía un ACK indicando el paquete que estoy esperando
  - Descarta los datos

Go back-N

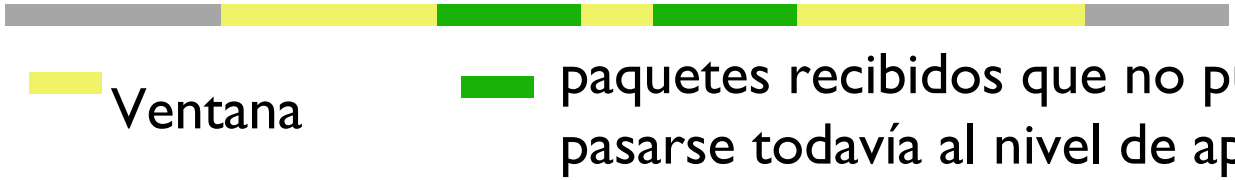
Ventana de 4 paquetes





Selective Repeat

- El receptor confirma (ACK) individualmente cada paquete
  - Mantiene en buffer los paquetes recibidos a la espera de reconstruir la secuencia y pasarlos al nivel de aplicación



- Se reenvian los paquetes no confirmados por timeout
  - **Timeout individual por cada paquete**
- Ventana de N paquetes que pueden enviarse sin recibir ACK

Eventos en el emisor (SR)

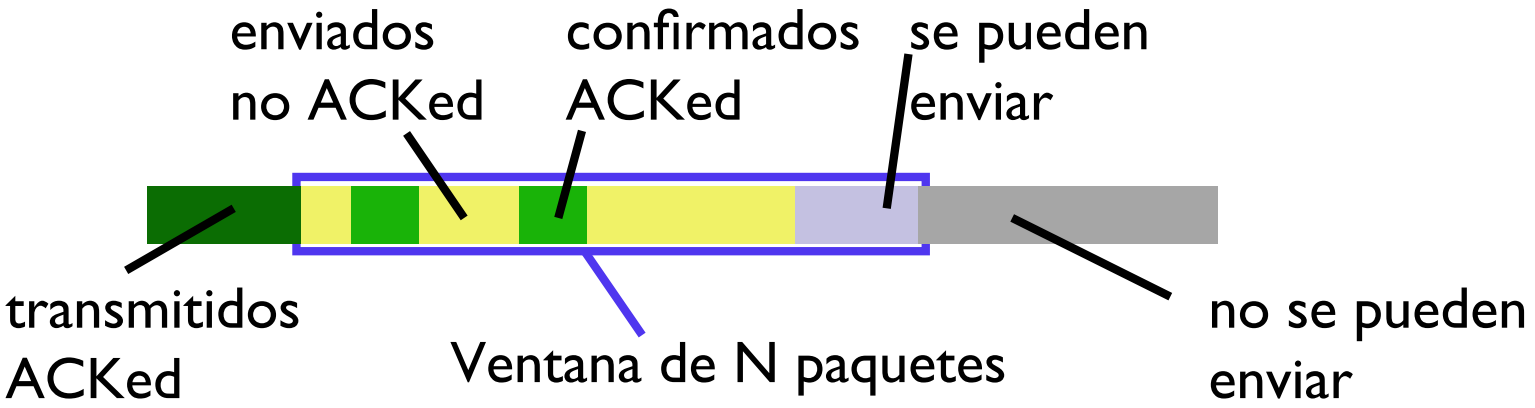
- **ACK recibido**
  - Se cancela el timeout de ese paquete
  - Si se puede avanzar la ventana se avanza hasta donde se pueda
  - Si la ventana avanza se envían paquetes nuevos si hay disponibles y se inician sus timeouts
- **Timeout de un paquete**
  - El paquete se reenvía
  - Se reinicia el timeout de ese paquete

Eventos en el receptor (SR)

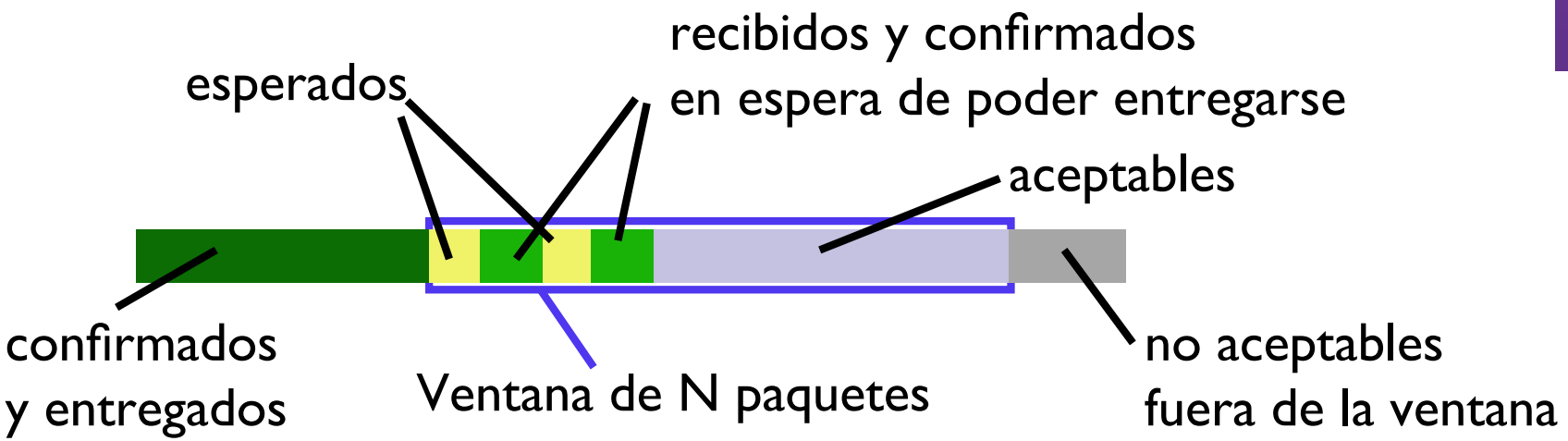
- El receptor tiene un buffer (y por tanto ventana) limitada
- **Recibidos datos en la ventana**
  - Se envía ACK de ese paquete
  - Se guarda el paquete en su posicion del buffer
  - Si el paquete es el esperado se entregan todos los paquetes continuos disponibles en la ventana al nivel superior y se avanza hasta donde se pueda
- **Recibidos datos anteriores a la ventana**
  - Se ignoran los datos
  - Se envía ACK de ese paquete
- **Recibidos datos posteriores a la ventana**
  - Se ignoran y no se envía nada

Selective Repeat

- Ventana deslizante del emisor

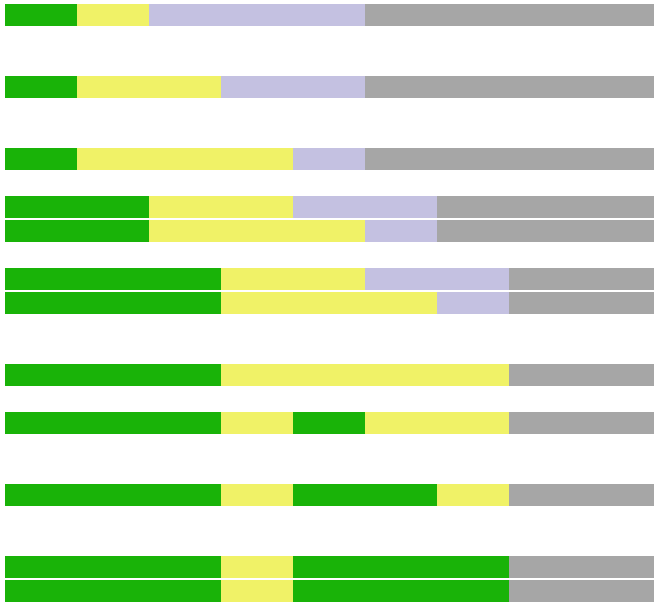


- Ventana deslizante del receptor

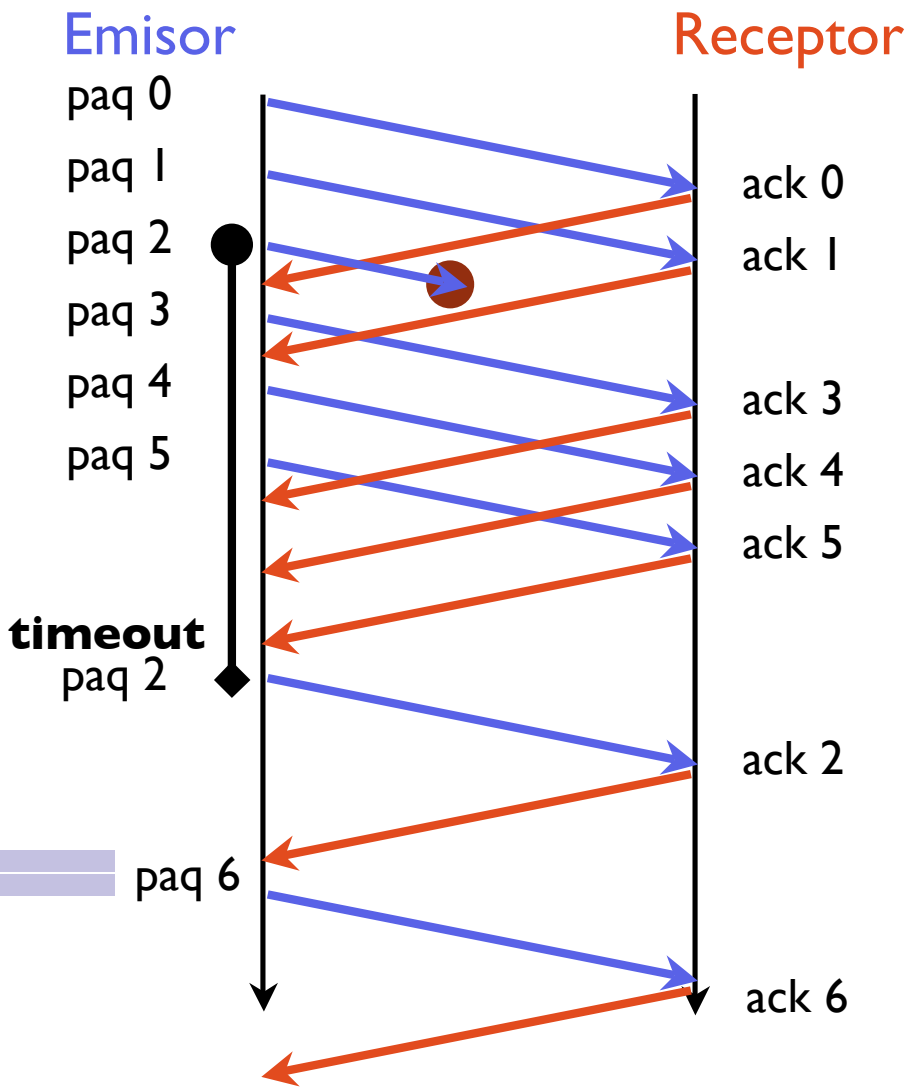


Selective repeat

Ventana de 4 paquetes



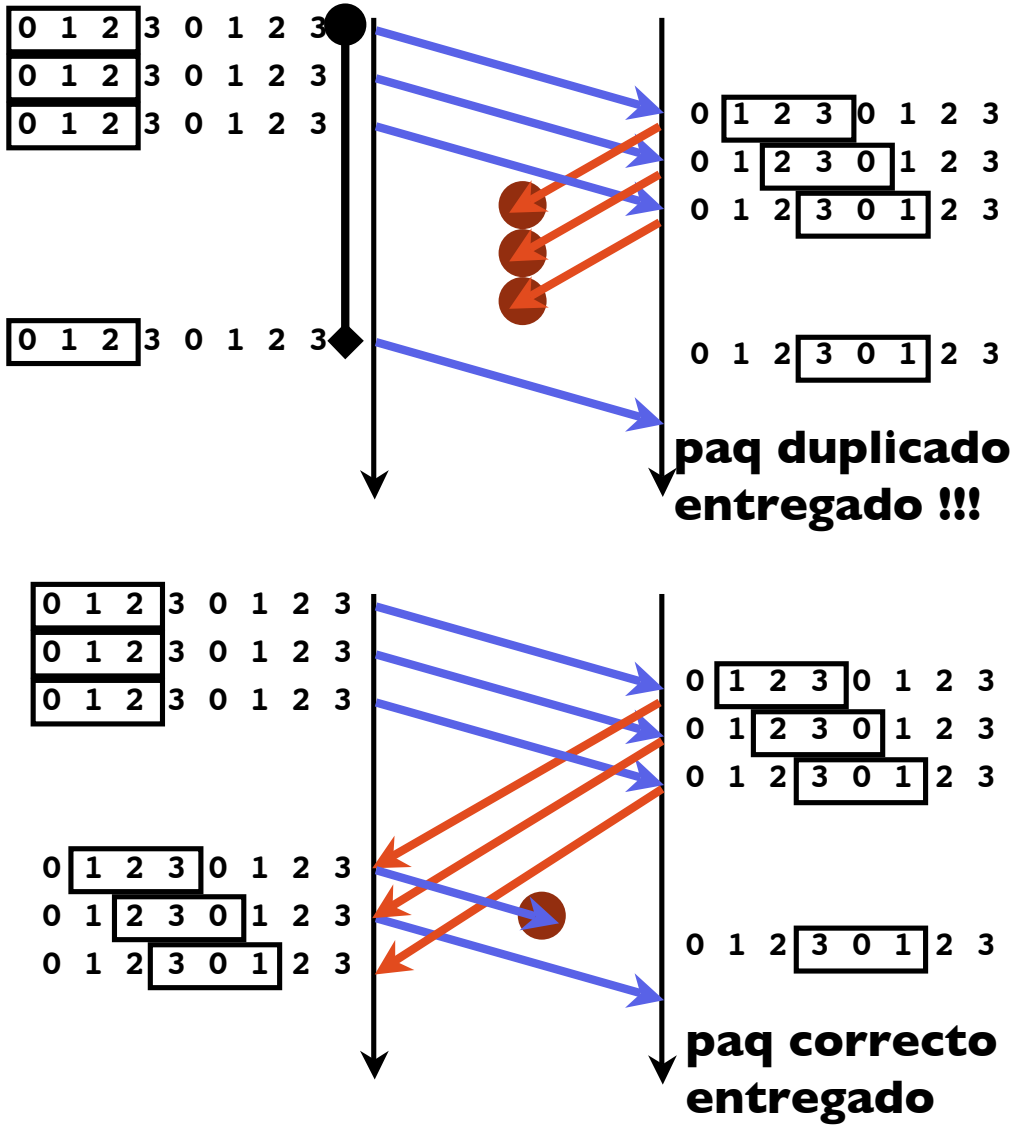
...



Aquí ACK no indica
el que espero recibir
sino indica el que **confirmo**

El problema del selective repeat

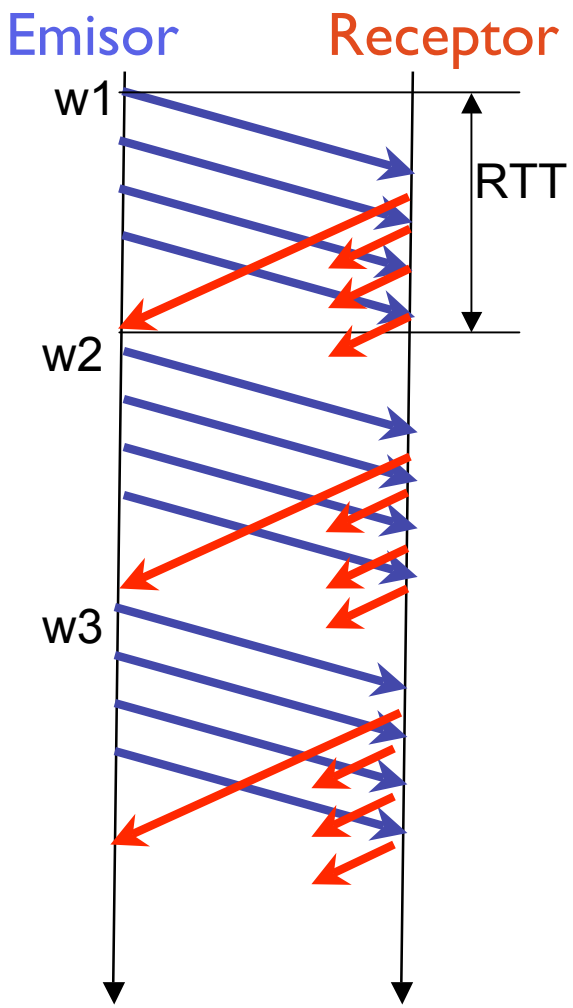
- Número de secuencia finito
- Ejemplo
  - seq= {0,1,2,3}
  - N=3
- El receptor no puede notar la diferencia entre los dos escenarios
- Y entrega datos duplicados como buenos
- Que relación debe haber entre #secuencia y N?



Go back-N y Selective Repeat

- Transporte fiable garantizado (en un escenario en el que se pierdan paquetes)
- Eficiencia mejor que stop-and-wait
- Cuanto mejor?
  - Análisis exacto más difícil.
    Aproximación tipica para caso mejor
    1 ventana cada RTT

    v=tamañoventana/RTT
- También proporcionan control de flujo dado que permiten al emisor enviar datos limitados por la ventana



Conclusiones

- Hay mecanismos y protocolos que permiten conseguir un transporte fiable sobre una red no fiable
- Incluso hay mecanismos que permiten conseguir un transporte fiable y razonablemente eficiente, utilizando números de secuencia y ventanas deslizantes

  Si me bajo un fichero de 900MB por HTTP desde un servidor. El ping a ese servidor es de 60ms. Y mi acceso a Internet es de empresa a 100Mbps. Cuánto tardare como mínimo? Estoy limitado por el acceso?
- Estos principios serán las bases para los protocolos de transporte de Internet

Próxima clase:

- protocolo de transporte de Internet TCP