Transporte fiable

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

Arquitectura de Redes, Sistemas y Servicios
Grado en Ingeniería en Tecnologías de Telecomunicación, 2º

Temario

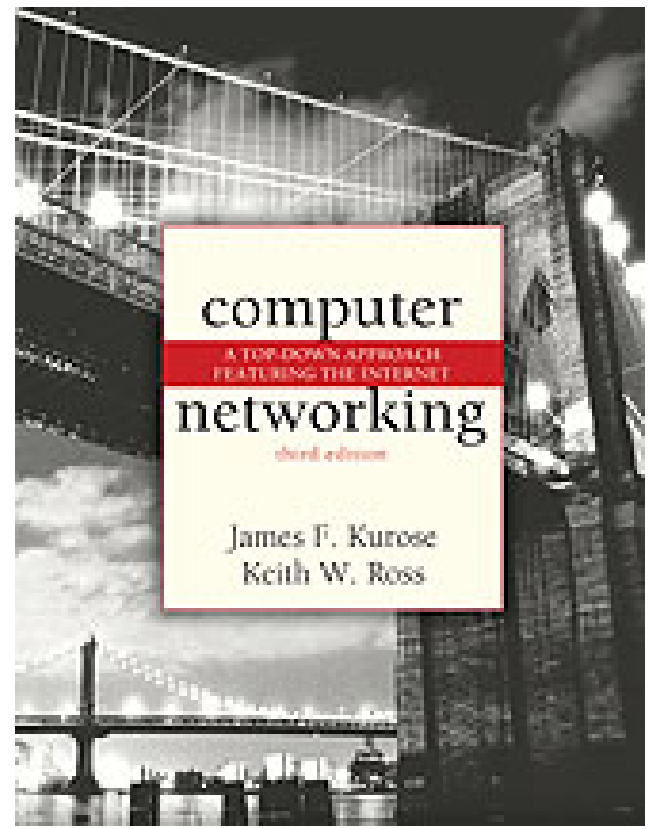
1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. Transporte fiable
7. Encaminamiento

Temario

1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. **Transporte fiable**
7. Encaminamiento

Material

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



El problema del transporte fiable

- Enviar datos y asegurarme de que llegan correctamente
  - No se pierde ninguno
  - No se cambia ninguno
  - No se generan más (no hay duplicados)
  - Llegan en el mismo orden
- ¿Qué velocidades se consiguen? ¿Qué limitaciones tiene?

Red y transporte

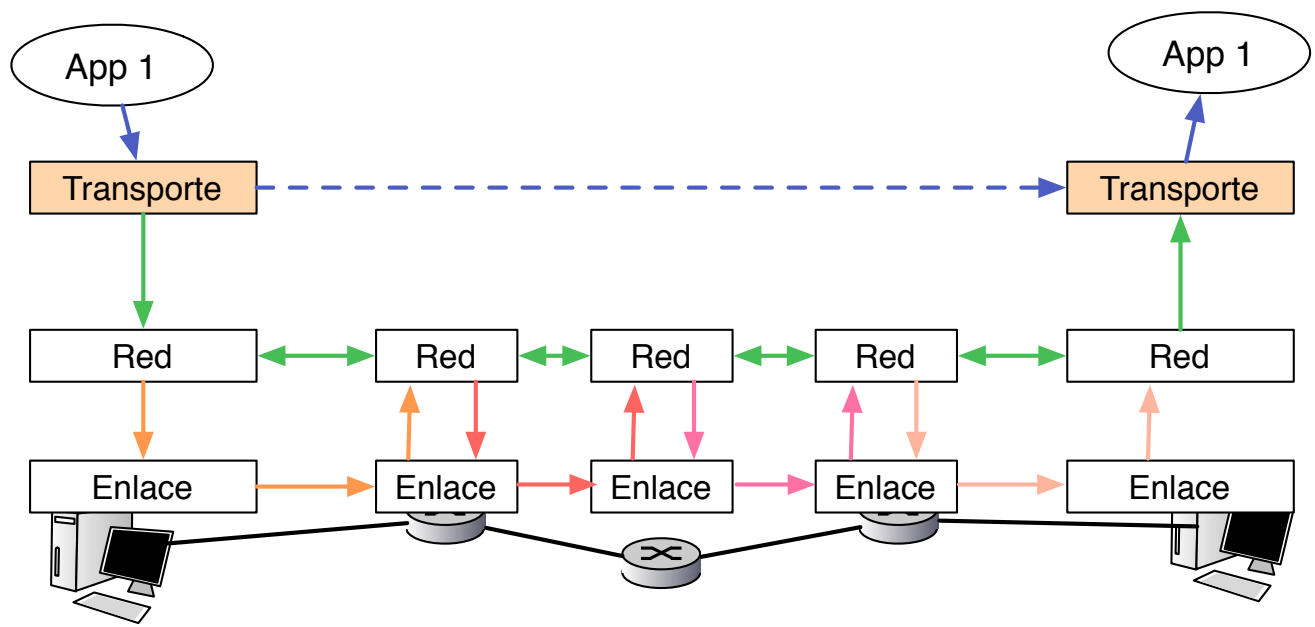
- Nivel de red: Comunicación lógica entre **hosts**

  Envía este paquete al nivel de transporte de la dirección IP a.b.c.d

  He recibido este paquete de la dirección IP x.y.z.t

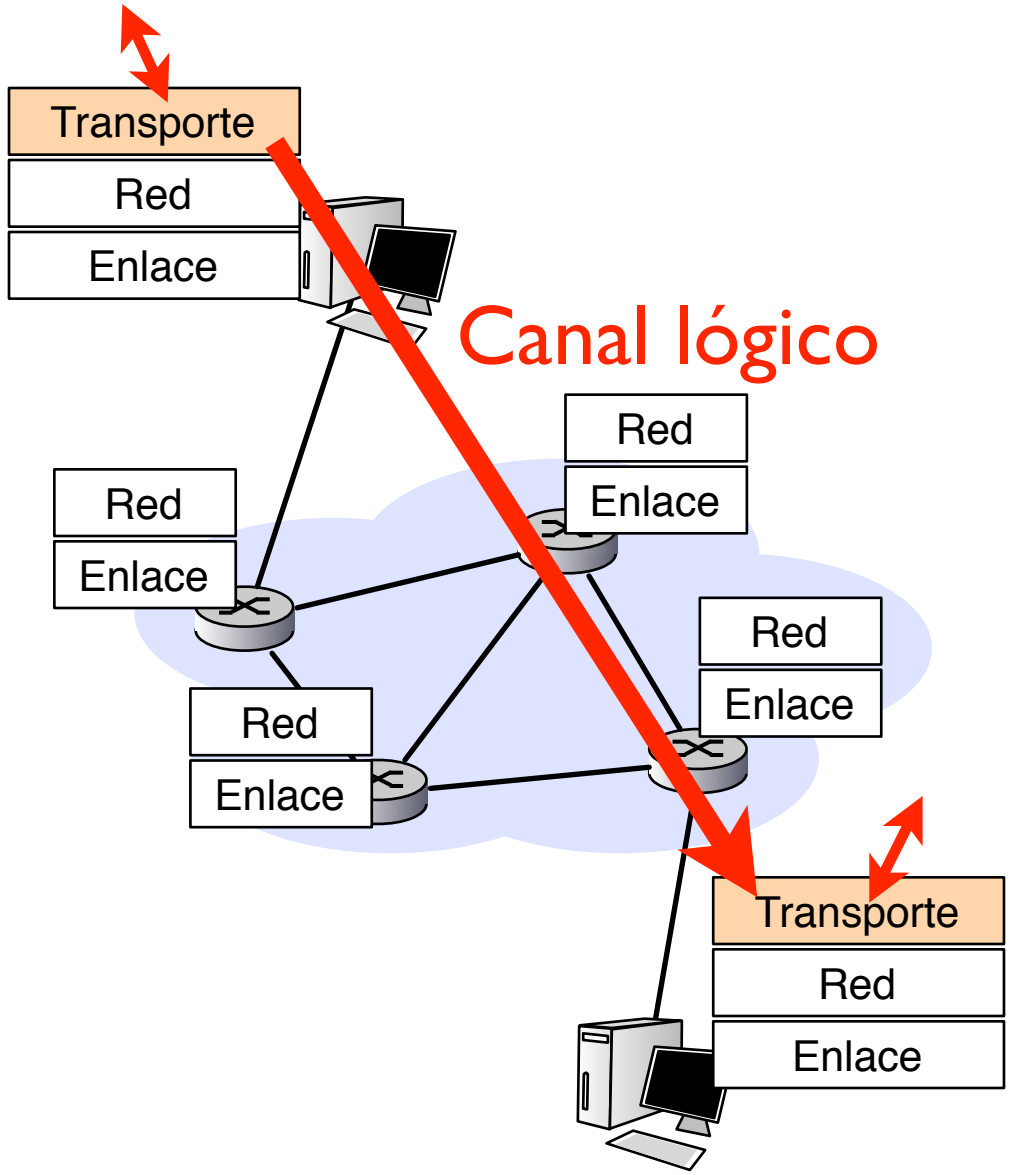
  No garantiza que todos los paquetes acaben llegando

- Nivel de transporte: Comunicación lógica entre **procesos**



Funciones del nivel de transporte

- Comunicación lógica entre aplicaciones
- Puede haber más de una aplicación en cada dirección IP
  - multiplexar aplicaciones
- Las aplicaciones quieren que todo lo que envían llegue
  - **Transporte fiable**
- Las aplicaciones ¿envían mensajes o establecen llamadas?
  - varios protocolos con interfaz de conexiones o mensajes
- No queremos saturar al receptor ni a la red
  - control de flujo y control de congestión



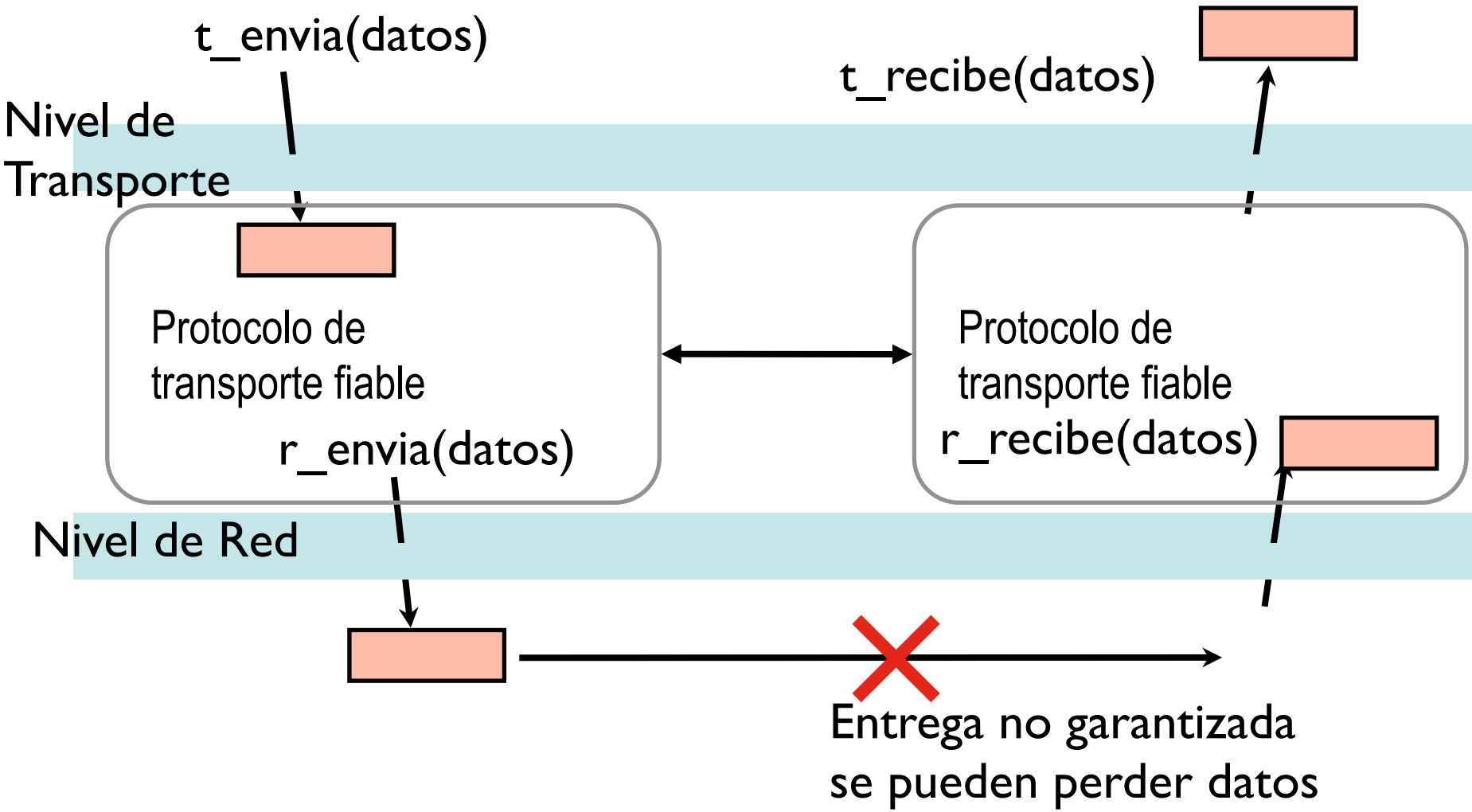
Transporte fiable

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

  Kurose

Transporte fiable

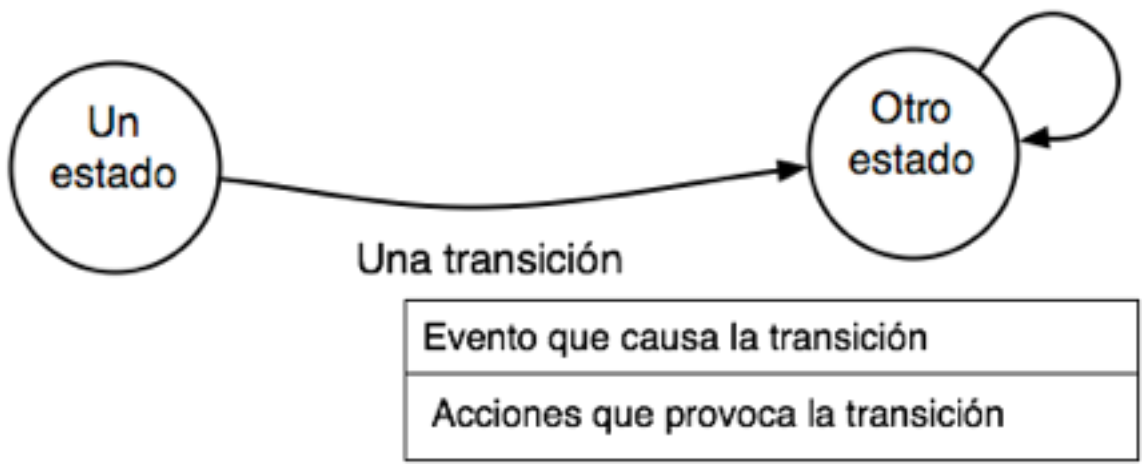
- ¿Se puede conseguir un transporte fiable sobre un nivel de datagramas de entrega no fiable?



Pensando en protocolos

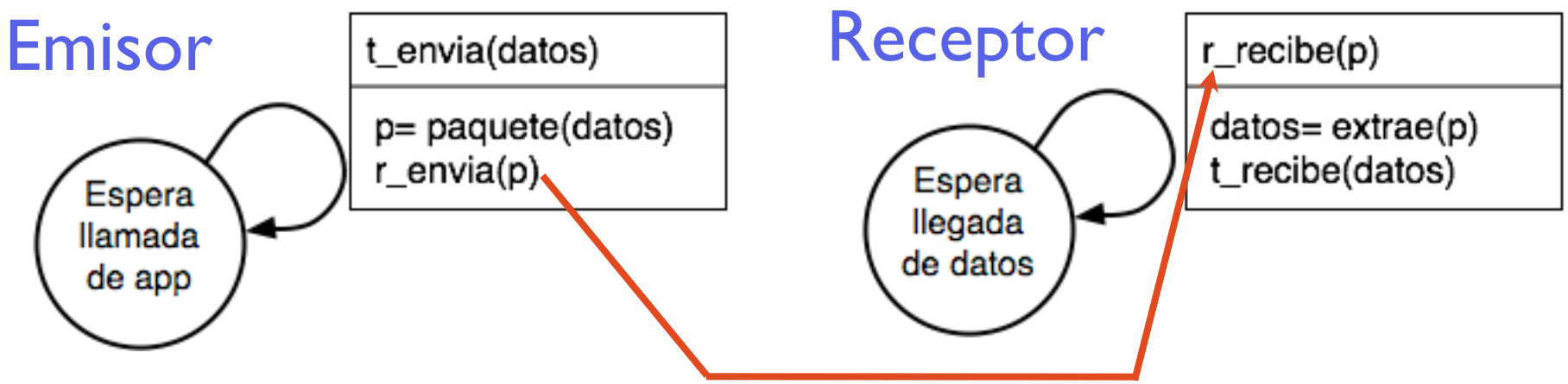
- Descripción protocolo (=programa)

  con máquinas de estados finitos
  - Eventos que pueden ocurrir
  - Acciones como resultado de esos eventos
  - El protocolo son las funciones para responder a eventos
- Emisor y receptor son diferentes programas y están 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

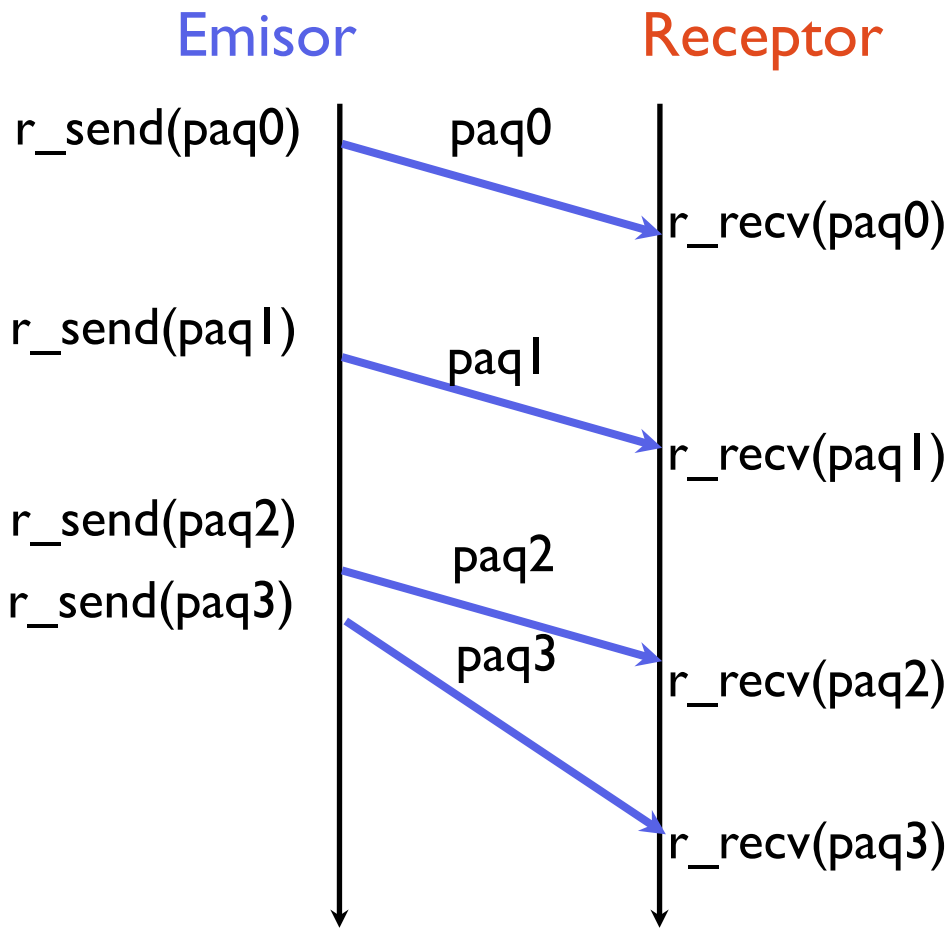


red fiable:

r_envia(p) siempre causa un r_recibe(p)

Ejemplo

- Todo lo que envío llega
- La red/el canal puede retrasar los paquetes pero acaba entregándolos todos
- Si la red es compleja puede que algunos tarden más que otros
- Algún problema si los entrega siempre pero en desorden??
  - Si queremos entrega en orden el protocolo debe construirlo



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 o CRC

      información redundante que depende los datos (por ejemplo bits de paridad)

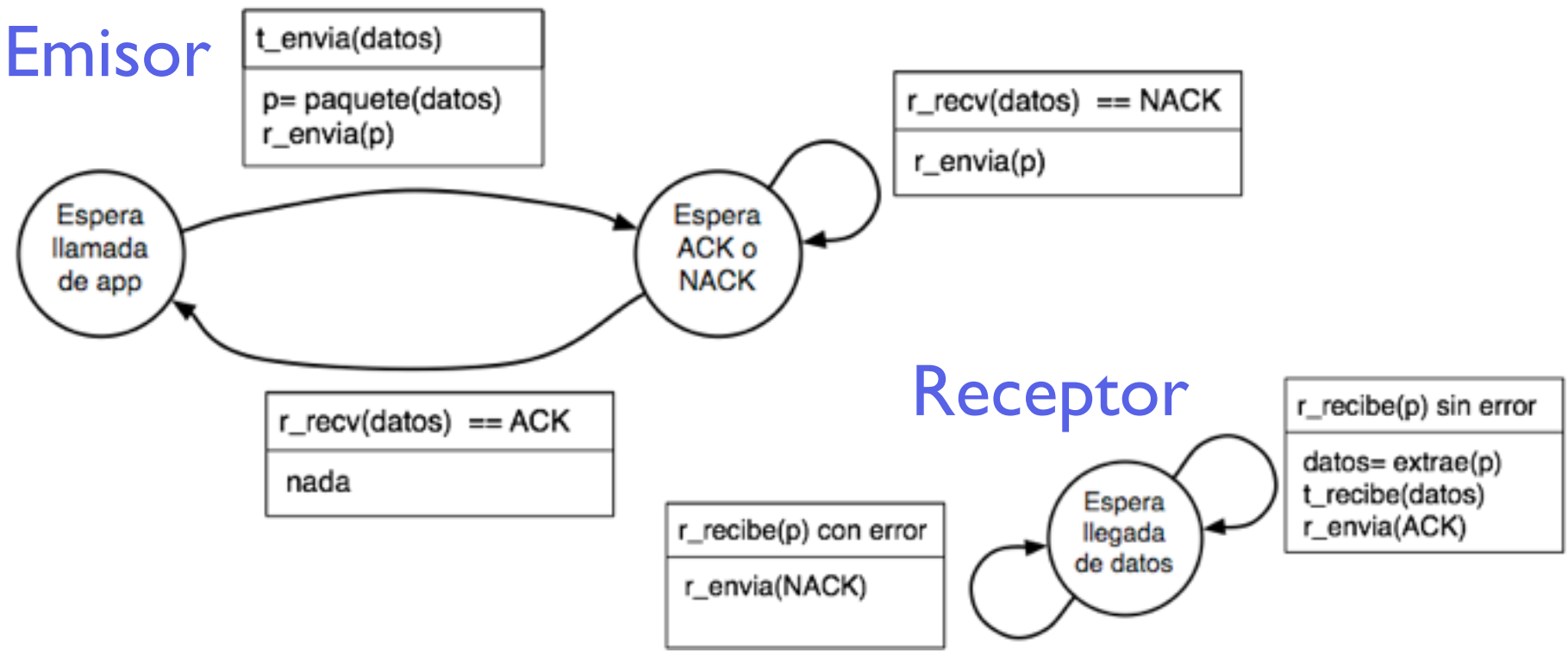
      si cambian bits es improbable que se siga cumpliendo la condición

      detecto en recepción que el paquete no es el mismo que se envió
  - **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 correctamente
  - **Reenvío de paquetes**
- El protocolo de transporte fiable debe retransmitir automaticamente los errores

  Esto se conoce tipicamente como **ARQ (Automatic Repeat reQuest)**

Protocolo de transporte fiable

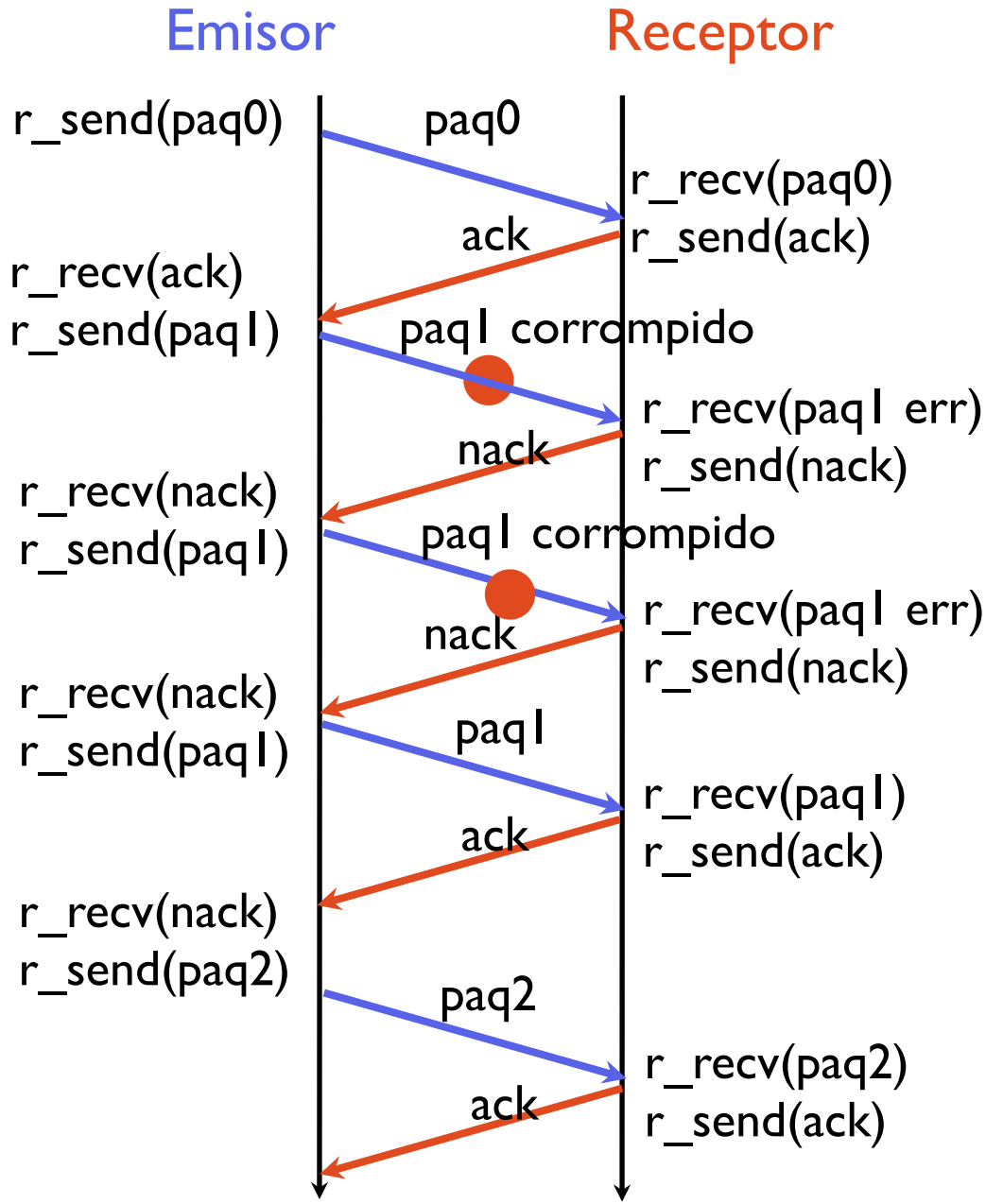
- Para un canal con errores de bits
  - Usamos detección de errores con checksum/CRC
  - Informamos al emisor de si llegan o no
  - El emisor tiene dos estados: esta esperando datos de la aplicación o bien está esperando una confirmación
- EL protocolo es conocido como **Stop-and-Wait**



Stop-and-wait

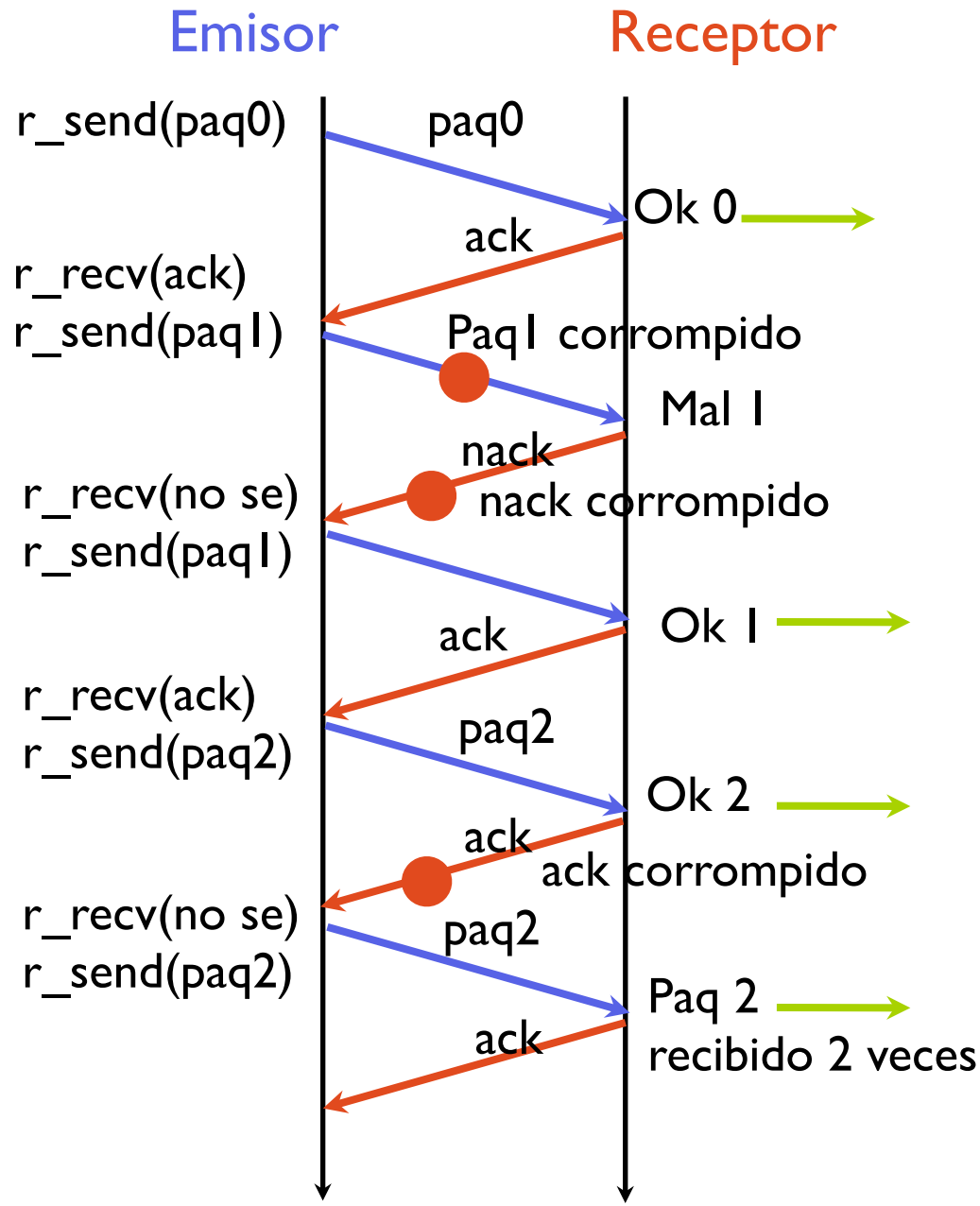
- El emisor controlado por el receptor
  - ACK (recibido OK manda otro)
  - NACK (recibido mal manda otra vez el mismo)
  - Mientras no me dice nada no envío

- De hecho esto puede considerarse también control de flujo (el emisor envía cuando el receptor le da permiso) = regulación de flujo por el receptor



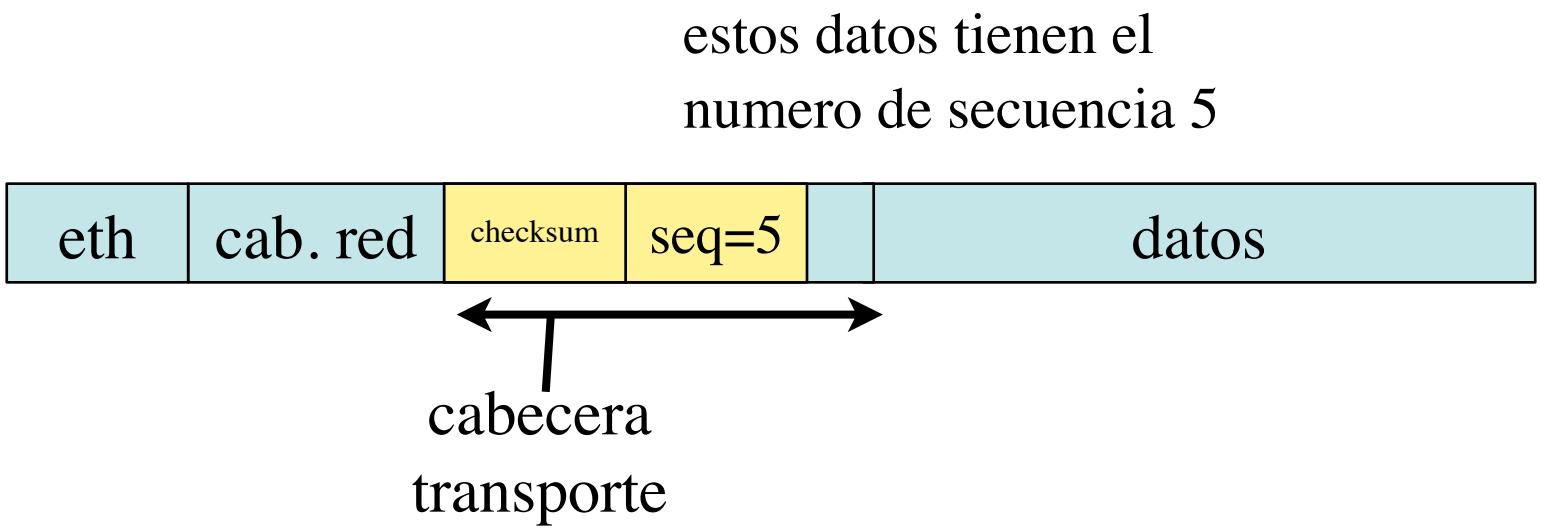
Problemas con stop-and-wait

- ¿Qué pasa si hay un error en la transmisión del ACK o NACK?
- Soluciones complican el protocolo
  - Detección de errores para ACK y NACK?
    - y que pasa si se pierden las confirmaciones del ACK/NACK
  - Checksums que permitan no solo detectar sino corregir errores?
    - Mucha información
  - 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 confirmarlo/ 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 números de secuencia en rangos limitados

Protocolo con número de secuencia

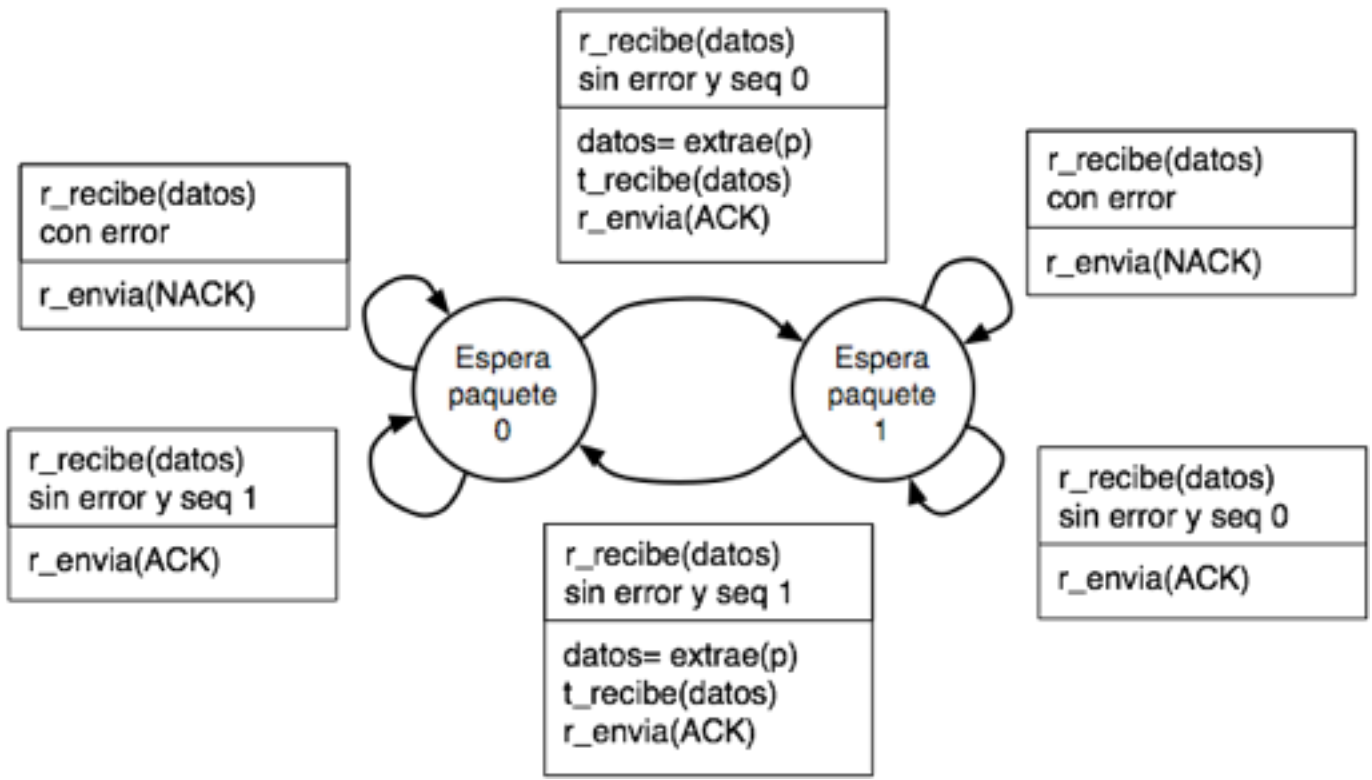
- 1 bit para número de secuencia

  Cada paquete de datos es secuencia 0 o 1

  – Si llega el que esperamos mandamos ACK y lo entregamos

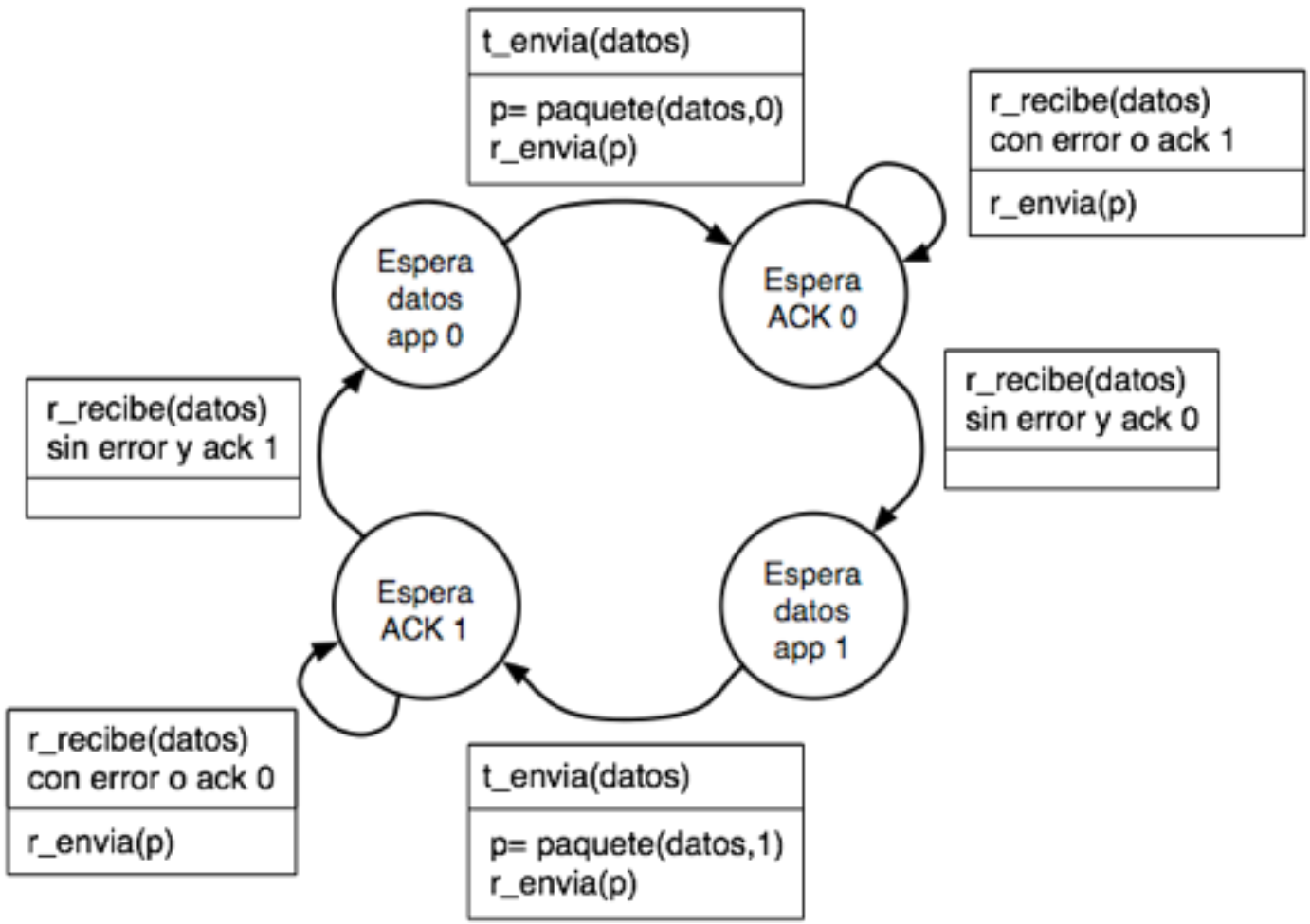
  – Si llega el que no esperamos?

  mandamos ACK pero no son datos nuevos



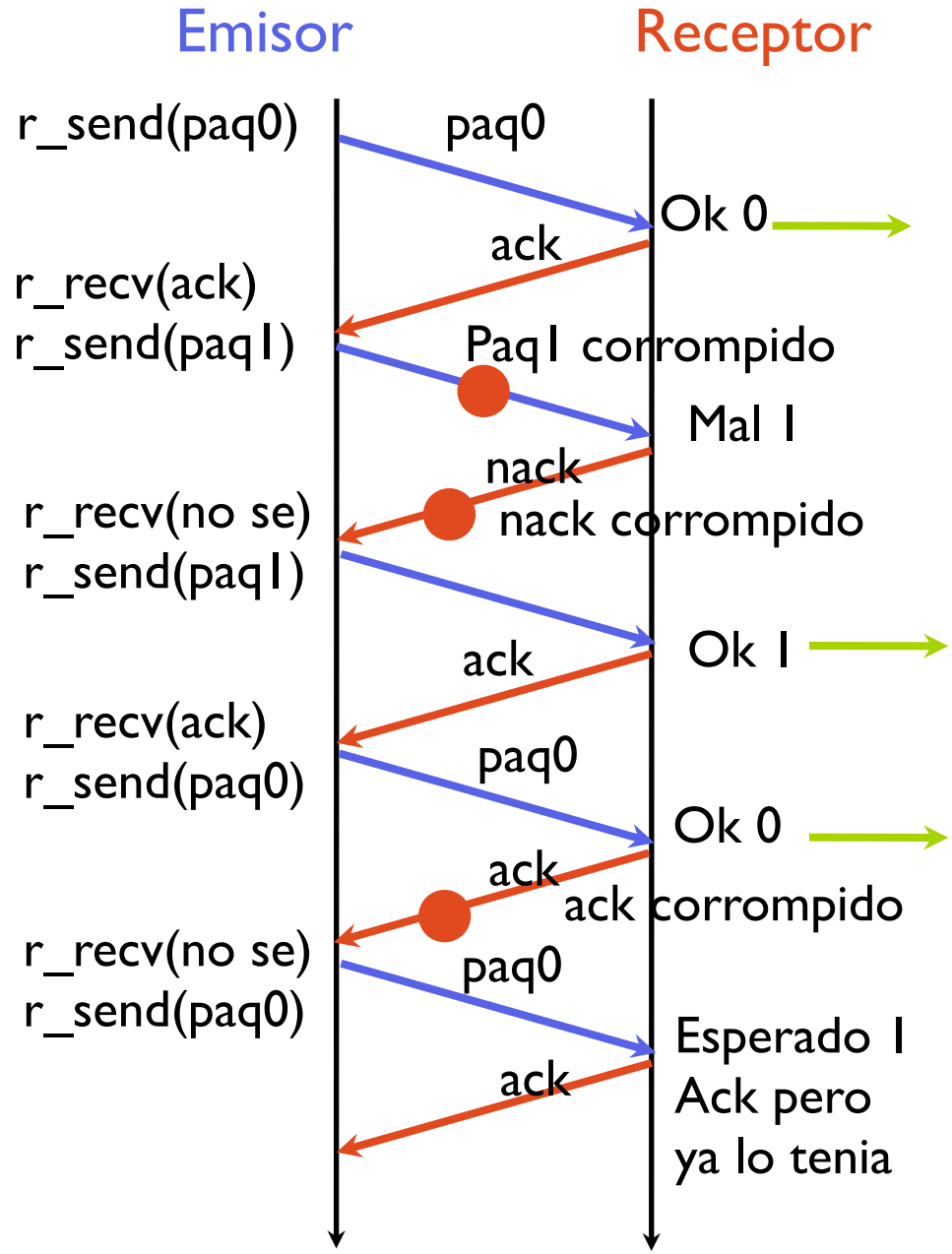
Protocolo con número de secuencia

- Estados del emisor



Ejemplo

- El receptor solo entrega a la aplicación el paquete correcto
- Mejora de nombre
  - Para el receptor
    ACK seq=0 y NACK seq=1
    significan lo mismo
    diremos
    ACK 1 = esperando el 1
    (algunos llaman RR1
    Ready to receive 1)
  - **ACK 0 = esperando el 0**
    (en vez de ACK seq=1 y NACK seq=0)



Ejemplo

- El receptor solo entrega a la aplicación el paquete correcto
- Mejora de nombre
  - Para el receptor

    ACK seq=0 y NACK seq=1

    significan lo mismo

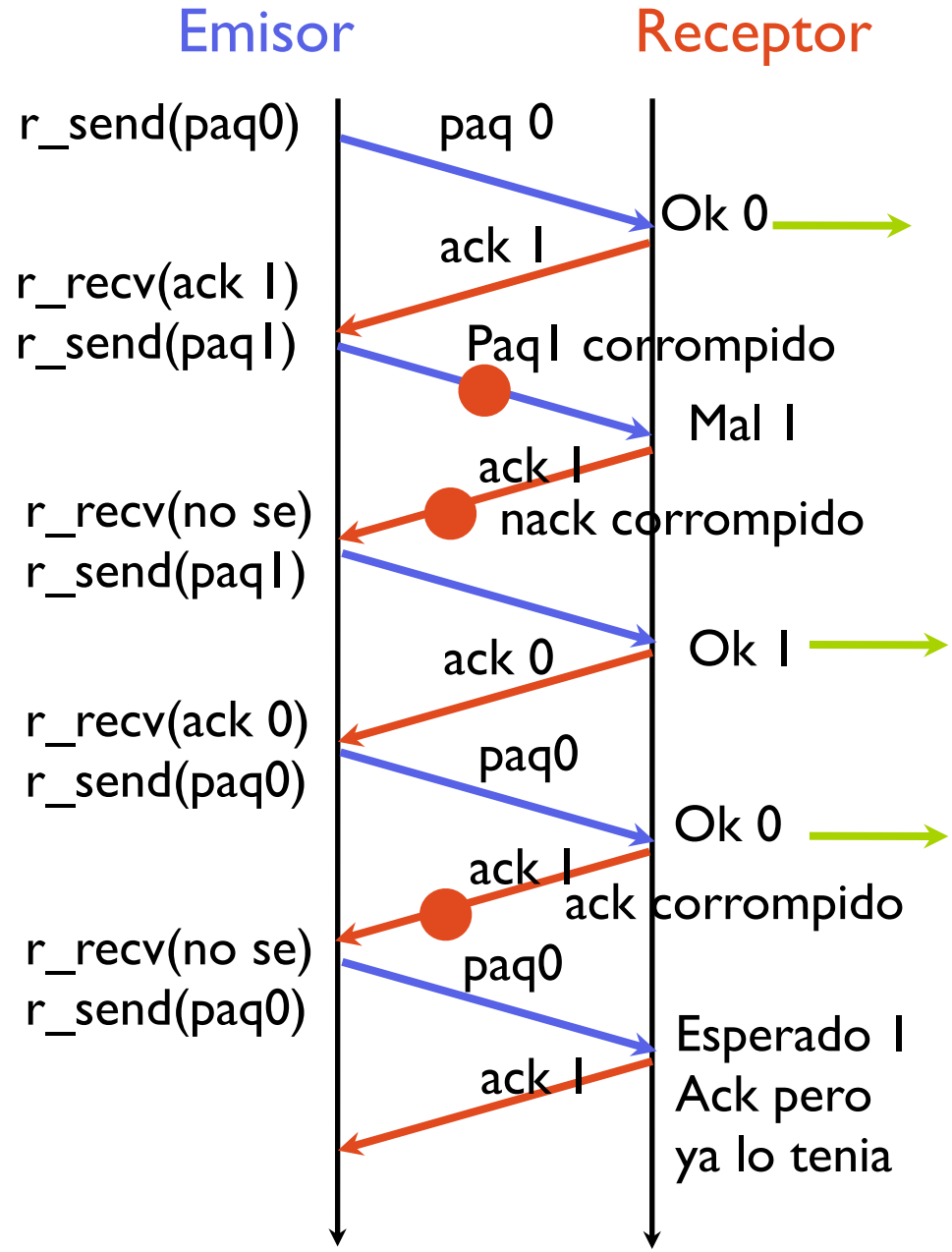
    diremos

    ACK 1 = esperando el 1

    (algunos llaman RR1

    Ready to receive 1)
  - **ACK 0 = esperando el 0**

    (en vez de ACK seq=1 y NACK seq=0)



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