Transporte fiable

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 transoporte en TCP/IP)
4. Conmutación de circuitos
5. Tecnologías
6. Control de acceso al medio en redes de área local
7. Servicios de Internet

Temario

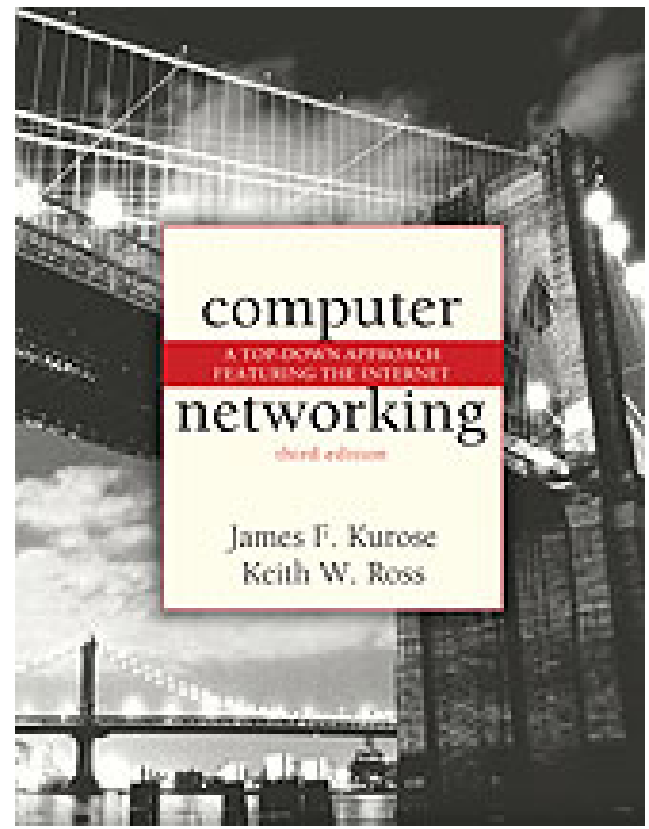
1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

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

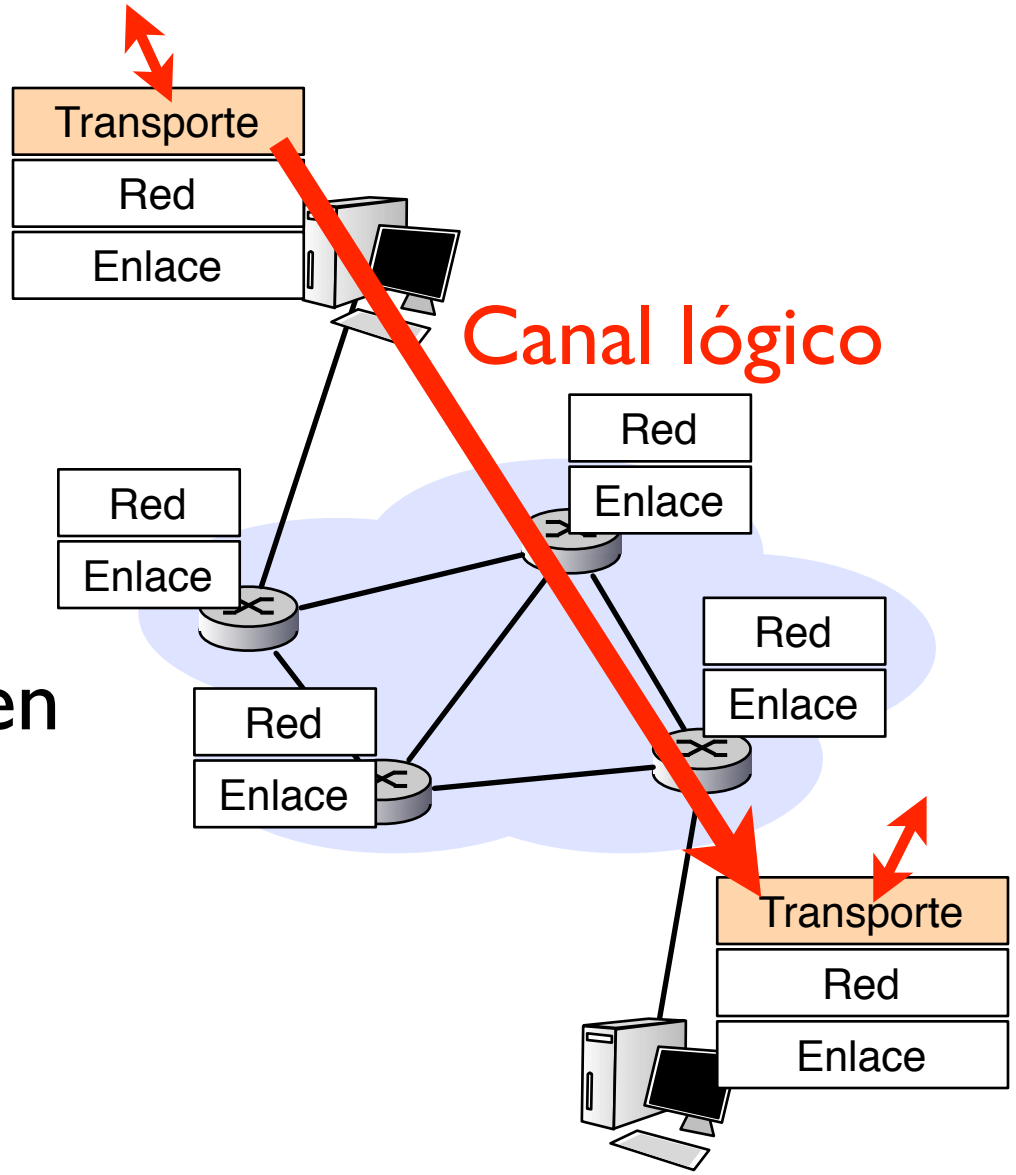


Objetivos

- Conceptos y principios del nivel de transporte
  - Multiplexación
  - Transporte fiable
  - Control de flujo
  - Control de congestion
- Estudio del nivel de transporte en Internet
  - Protocolos TCP y UDP

Funciones del nivel de transporte

- Comunicación lógica entre aplicaciones
- Protocolo en los extremos (end-to-end)
  - segmentación
  - reensamblado
- 2 niveles de transporte en Internet
  - TCP
  - UDP

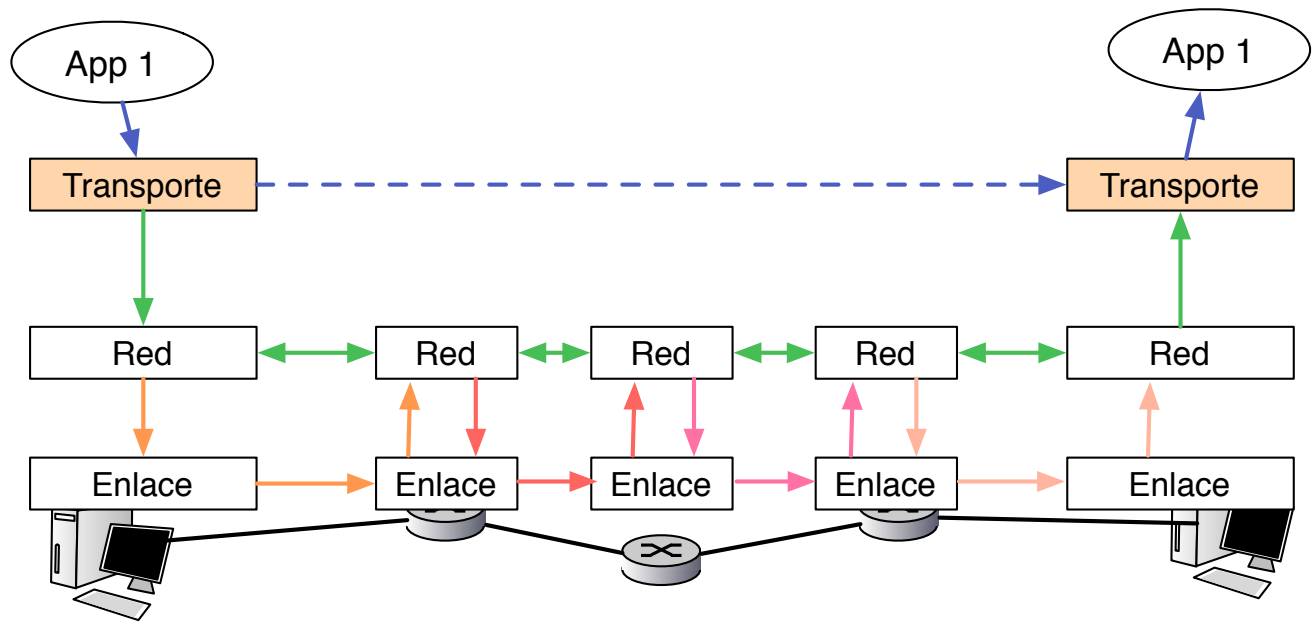


Red y transporte

- Nivel de red

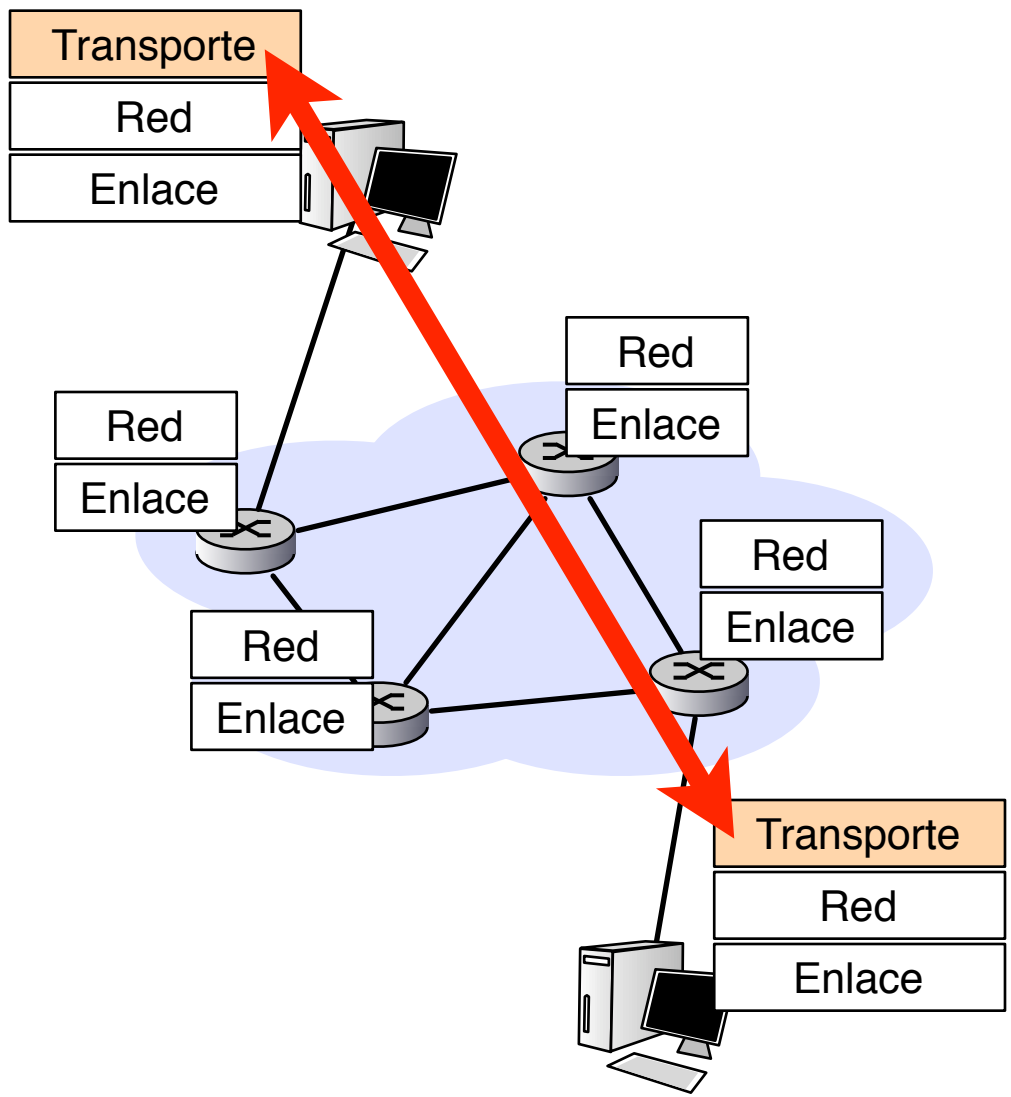
  Comunicación lógica entre **hosts**
- Nivel de transporte

  Comunicación lógica entre **procesos**
  - mejora y utiliza la comunicación entre hosts



Transporte en Internet

- Entrega fiable y en orden (TCP)
  - control de congestión
  - control de flujo
  - establecimiento de conexión
- Entrega no fiable, sin garantias de orden (UDP)
  - best-effort igual que IP
- En los dos casos
  - retardo no garantizado
  - ancho de banda no garantizado



Funciones TCP/UDP

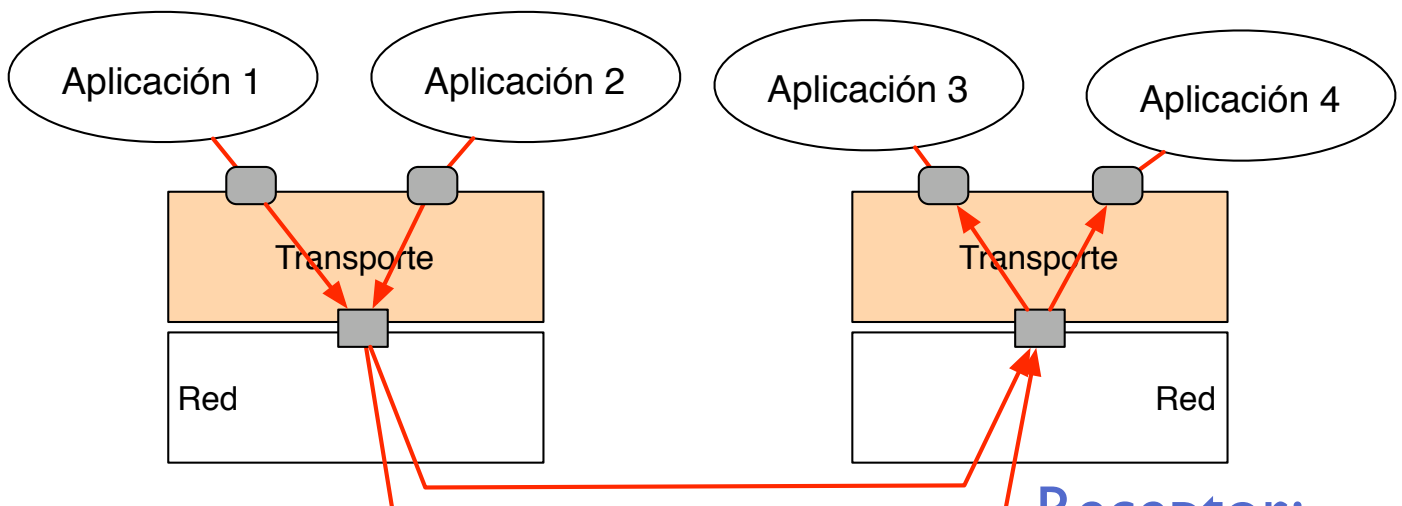
- Función común

  Multiplexación/demultiplexación de aplicaciones

- Funciones sólo UDP
  - Envio no orientado a conexión
- Funciones sólo TCP
  - Manejo de conexiones
  - Transporte fiable de datos
  - Control de flujo y de congestión

Multiplexación y demultiplexación

- Un host con varias aplicaciones/programas/ procesos corriendo
  - El nivel de red envia los paquetes al nivel de red del ordenador destino
  - El nivel de transporte arbitra la comunicación entre diferentes aplicaciones
  - **un** nivel de red, **un** nivel de transporte, **varias** aplicaciones



Emisor:

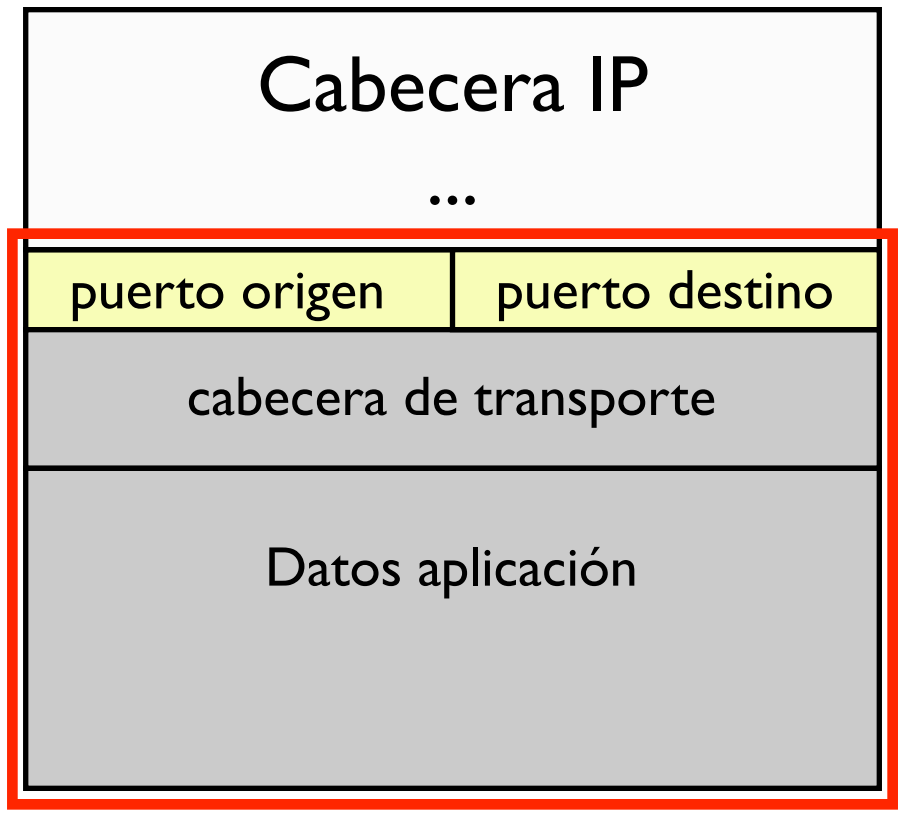
- recoger datos de distintas fuentes
- encapsular con información del origen

Receptor:

- entregar a la aplicación (socket) correcta

Multiplexación y demultiplexación

- El host recibe datagramas IP
  - Cada datagrama IP lleva una dirección IP de origen y de destino
  - Cada datagrama IP lleva un segmento del nivel de transporte
  - Cada segmento del niel de transporte lleva un puerto origen y destino (puerto: identificador de proceso/aplicación)
- El nivel de tranporte usa las direcciónes IP y los puertos para decidir a quien entrega los datos

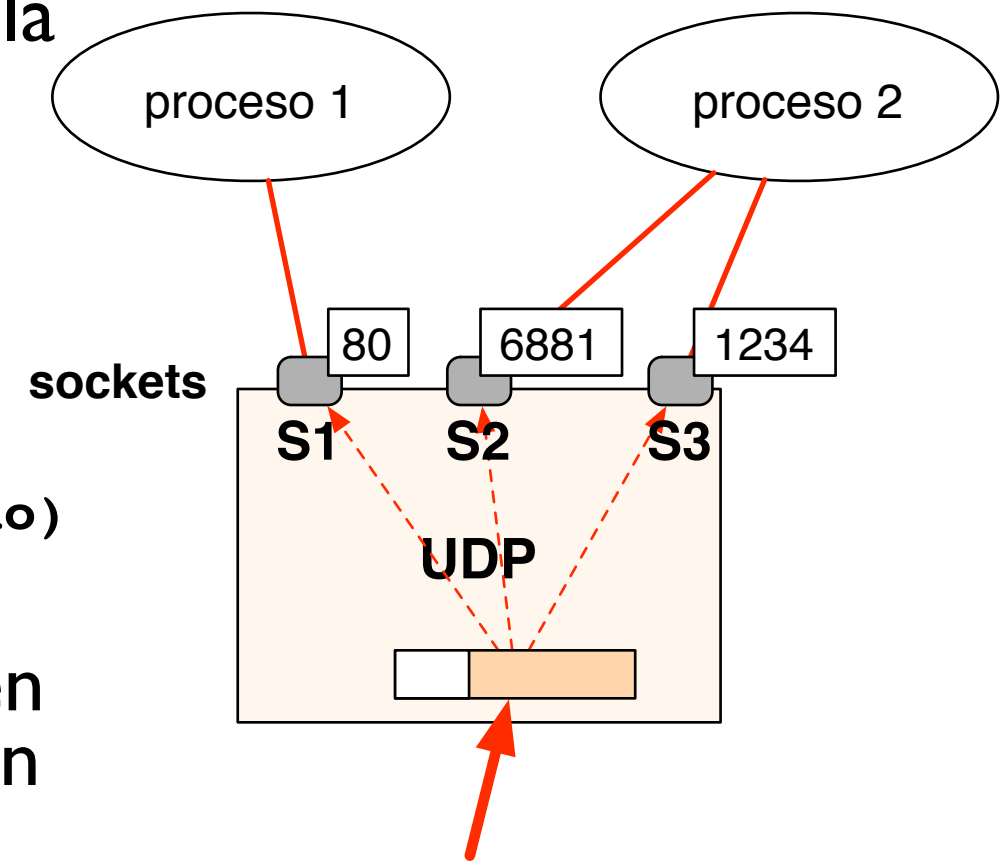


Segmento del nivel de transporte

Demultiplexación no orientada a conexión

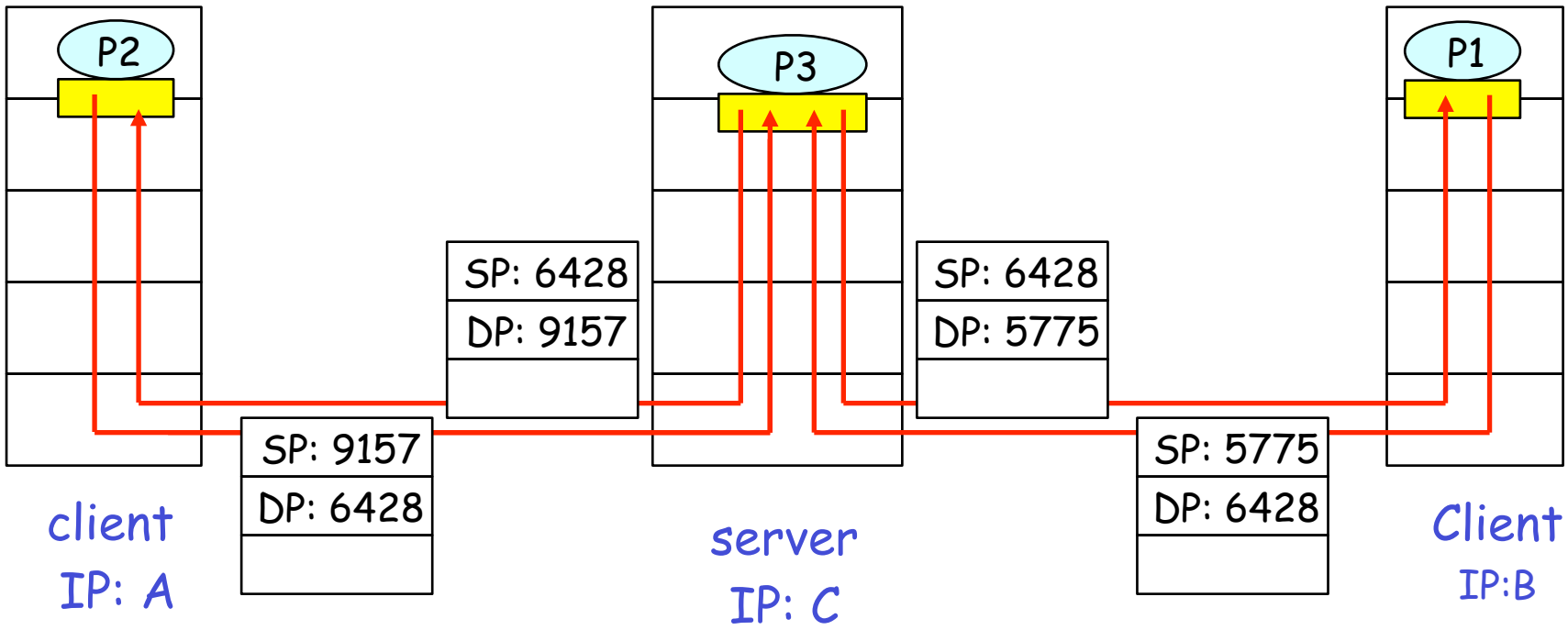
- Se extraen de la cabecera IP la dirección IP destino y de la cabecera UDP el puerto destino
- Se entregan los datos al socket identificado por la tupla

  `(dirIP destino, puerto destino)`
- Paquetes provenientes de diferentes direcciónes origen y puertos origen se entregan al mismo socket



Demultiplexación no orientada a conexión

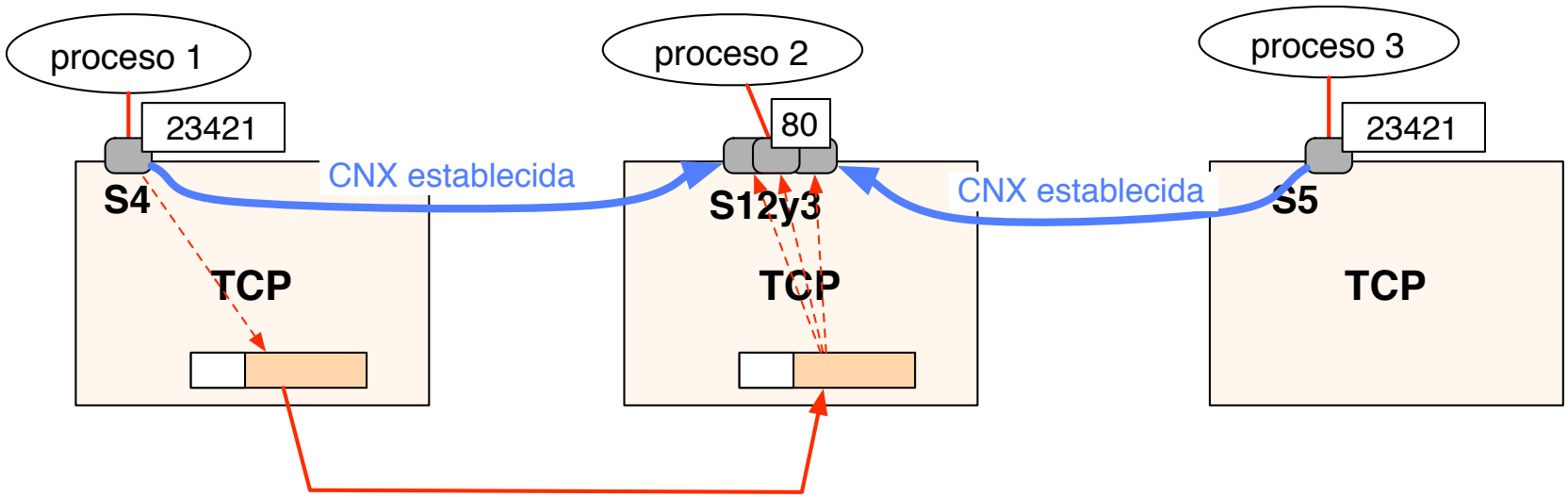
- Ejemplo



- La dirección origen y puerto origen permiten responder al cliente

Demultiplexación orientada a conexión

- Cuando TCP recibe un paquete puede pertenecer a varias conexiones establecidas
- Varios sockets con el mismo puerto...

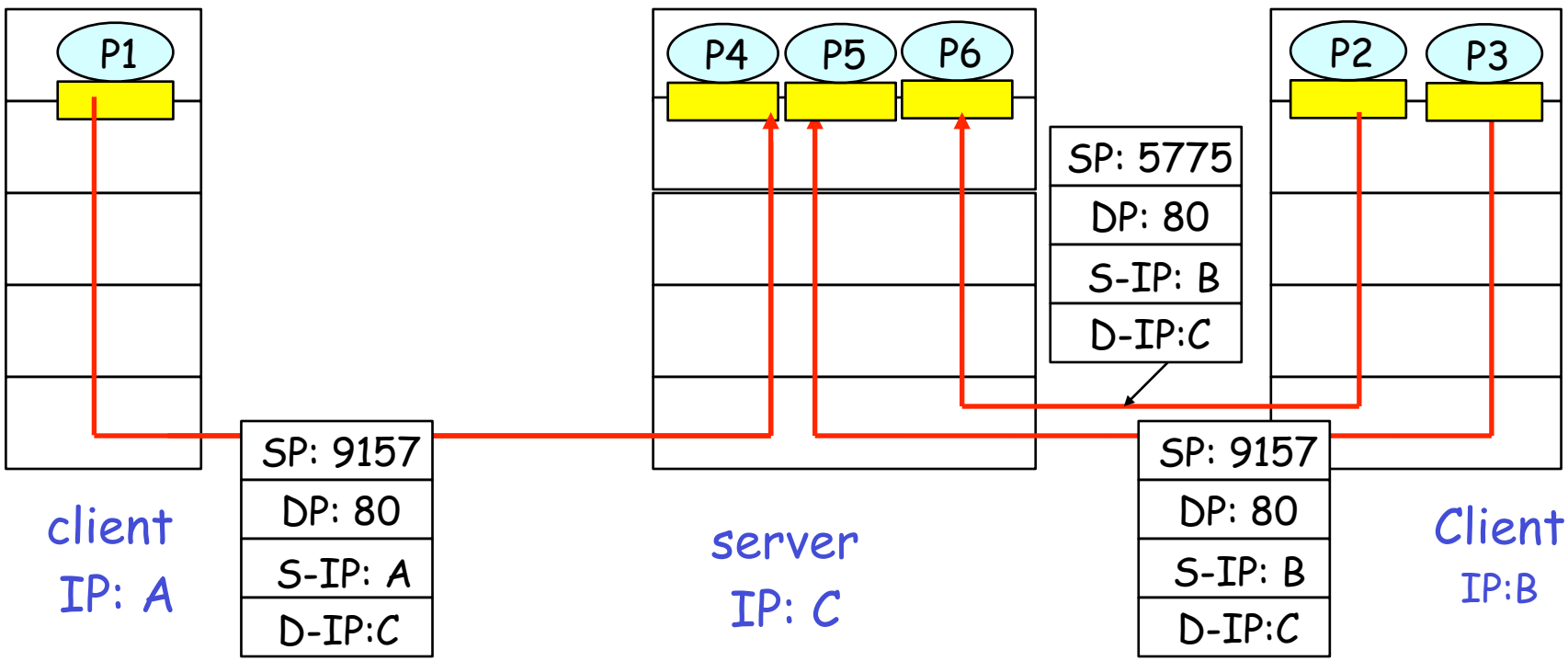


- Se entrega al socket identificado por la 4-tupla

  `(dirIP origen, puerto origen, dirIP destino, puerto destino)`

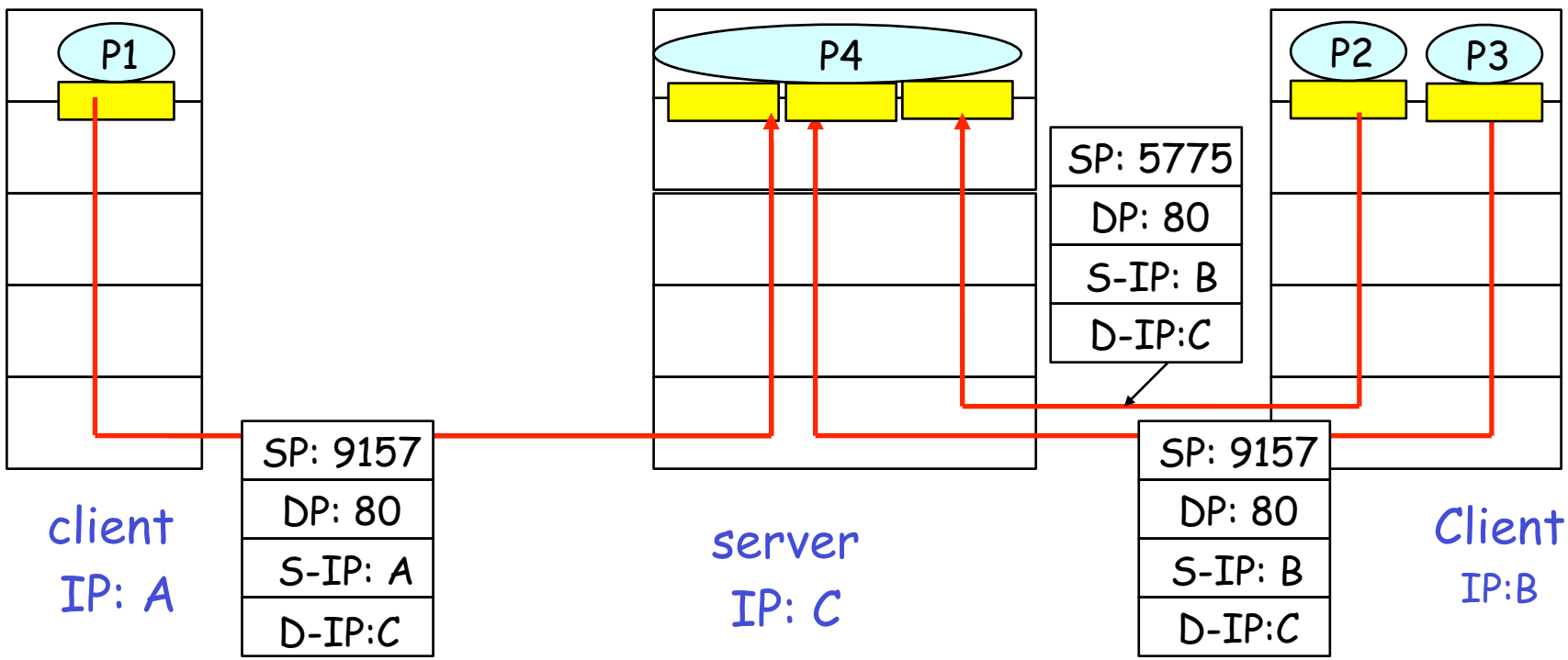
Demultiplexación orientada a conexión

- Ejemplo



Demultiplexación orientada a conexión

- Otro ejemplo



UDP: User Datagram Protocol

- UDP: User Datagram Protocol (RFC-768)
- Proporciona un servicio de transporte para aplicaciones sobre IP de tipo **Best-Effort**
  - Sin garantizar la entrega
  - Sin garantizar el orden de entrega
  - Mucho menos con tiempo o con ancho de banda garantizado
  - Lo unico que añade a IP es la multiplexación de aplicaciones y deteccion de errores ??
- No orientado a conexión
  - No hay establecimiento
  - Cada datagrama se trata independientemente (protocolo sin estado)

UDP

¿Por qué un protocolo como UDP?

- Es rapido: no hay establecimiento

  aunque no garantize el retardo no añade retardos innecesarios
- Es simple: no hay estado de conexión

  ni en el emisor ni en el receptor
- Poco overhead

  la cabecera UDP ocupa lo minimo posible
- Es eficiente: no hay control de congestión

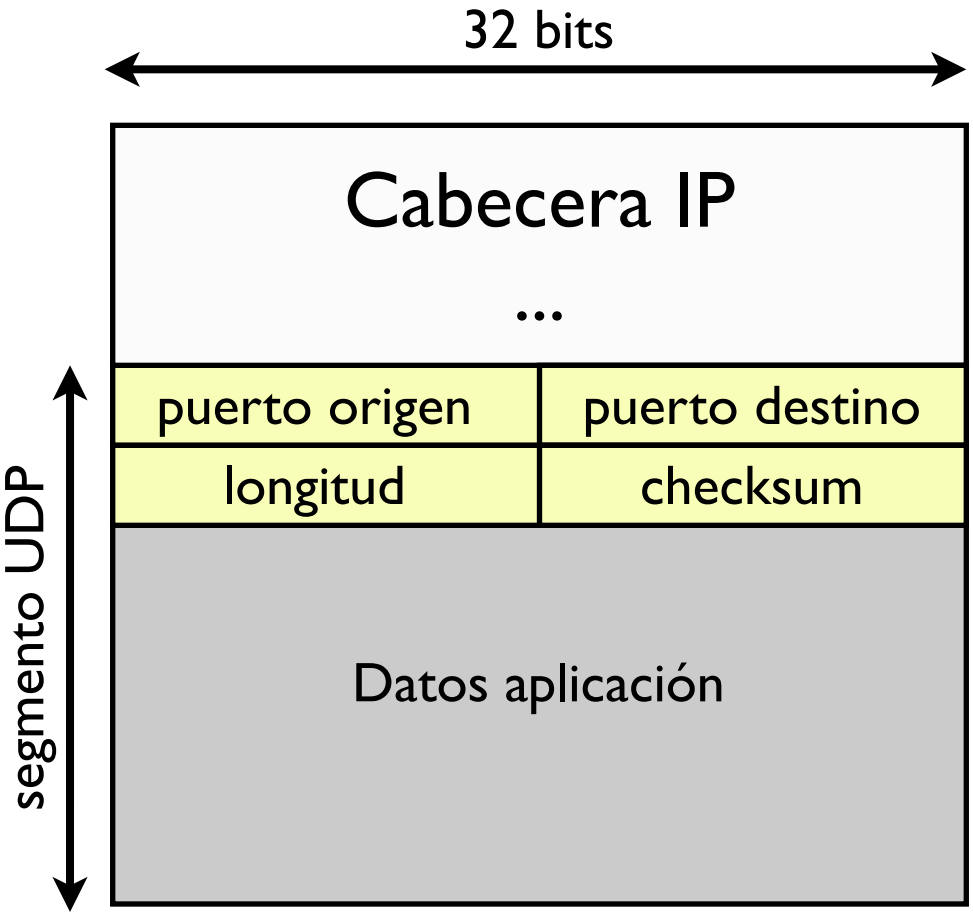
  puede usar todo el ancho de banda que consigas

UDP: detalles

Formato del paquete

- puerto origen
- puerto destino
- longitud del segmento UDP
  - 8 - 65535 bytes
- checksum

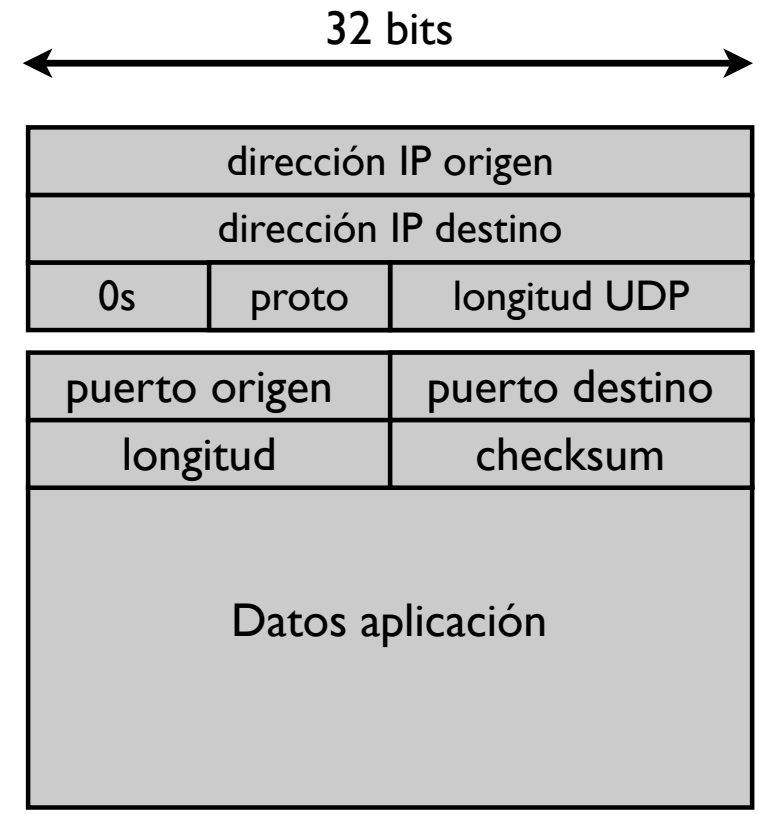
sólo 8 bytes de cabecera !



UDP: checksum

- Cálculo del checksum
  - Se trata el segmento como serie de valores de 16 bits
  - Suma binaria

    en complemento a 1
    - **pseudo-cabecera** con algunos campos de la cabecera IP (para proteger errores en las direcciones y el protocolo)
    - **segmento UDP**



- Es el complemento a uno de la suma binaria en complemento a 1 en 16 bits del bloque de arriba
- De esta forma al hacer la suma binaria en complemento a 1 de un paquete correcto debe salir 0xFFFF
- Detecta errores en todo el segmento UDP (aunque no todos)
- Si UDP detecta errores en un paquete recibido no lo entrega

UDP

En que se usa UDP ?

- Aplicaciones de streaming multimedia (audio/video en tiempo real o audio/video-conferencia)
  - tolerantes a las pédidas y sensibles al ancho de banda
- Mensajes de DNS
  - bajo retardo y poco overhead
- SNMP (monitorización de red), RIP, DHCP...
  - poco overhead y protocolo sencillo
- Juegos en red
  - mínimo retardo posible

En resumen

- El nivel de transporte proporciona comunicación lógica entre aplicaciones mejorando los servicios de IP
- Dos protocolos de transporte
  - TCP orientado a conexión y comunicación fiable con muchas prestaciones
  - UDP orientado a datagramas y con no demasiadas prestaciones pero gracias a ello
    - Simple de implementar, entrega rápida y más eficiente
- Pero ¿cómo se hace transporte fiable si IP puede perder mensajes?

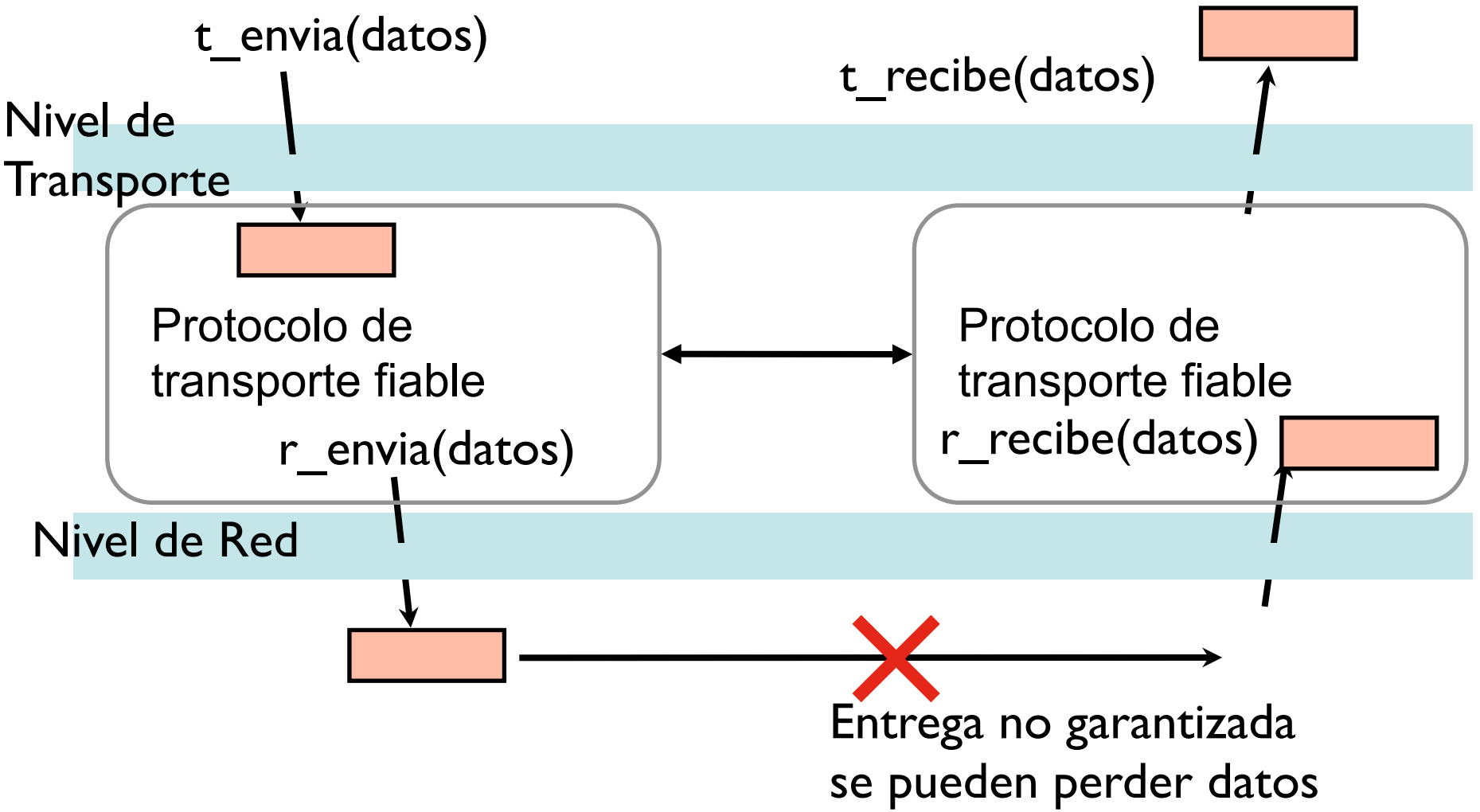
Transporte fiable

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

  Kurose

Transporte fiable

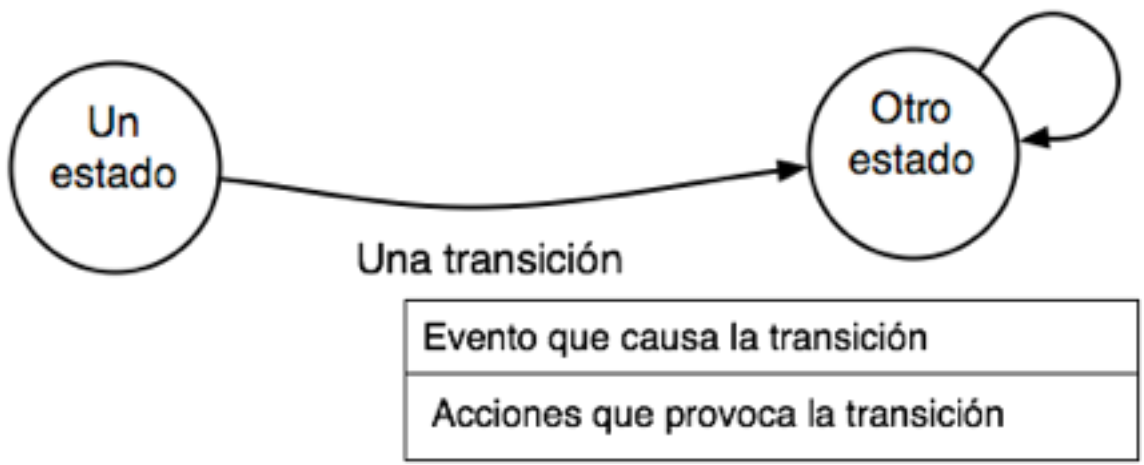
- ¿Se puede conseguir un transporte fiable sobre un nivel de datagramas de entrega no fiable?



Protocolo de transporte fiable

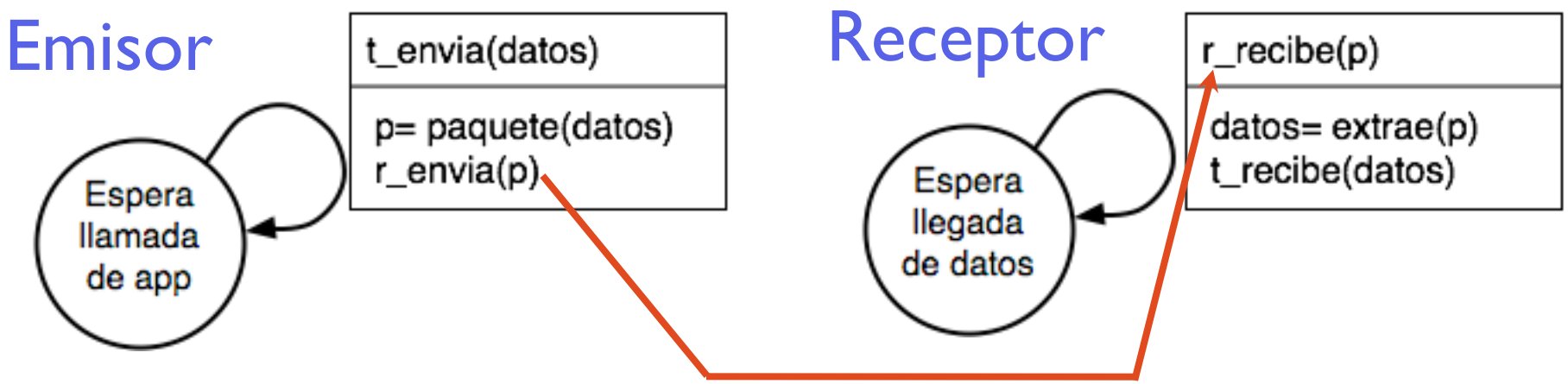
- Descripción protocolo (=programa)

  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

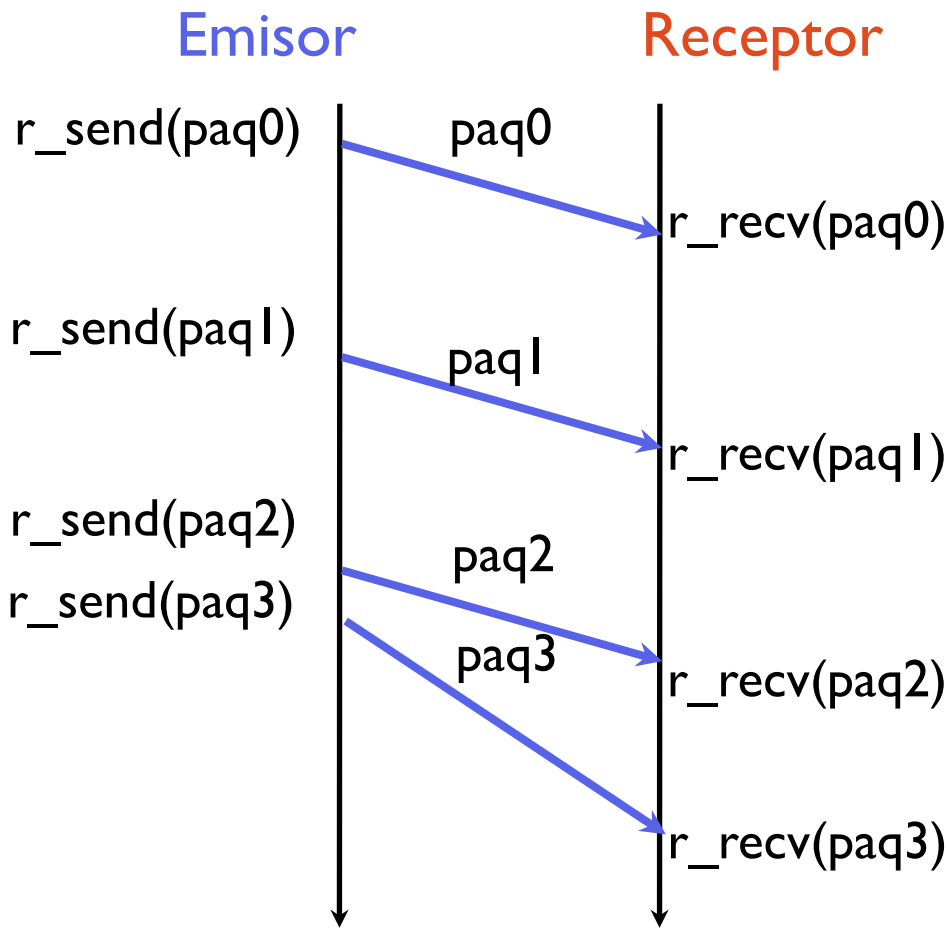


red fiable:

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
- Algún problema si los entrega siempre pero en desorden??



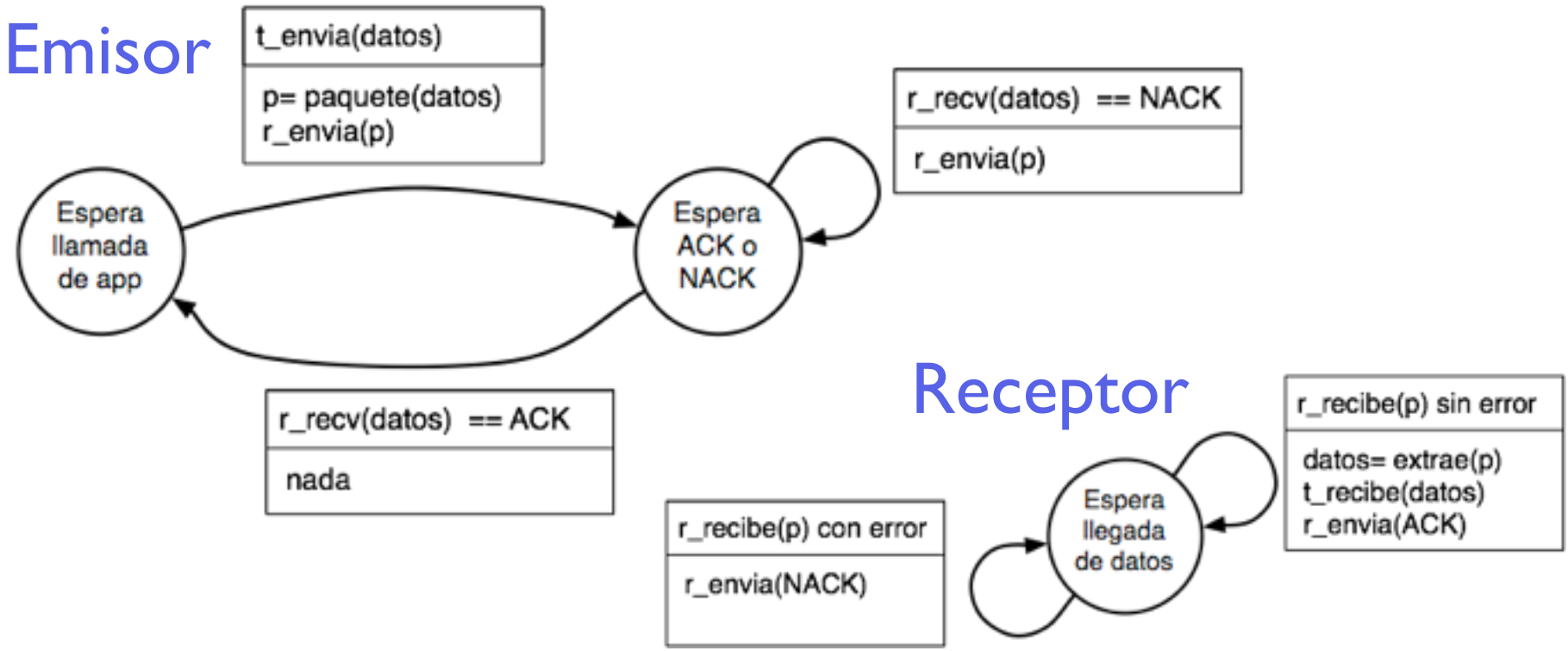
Operación normal

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 Repeat reQuest)

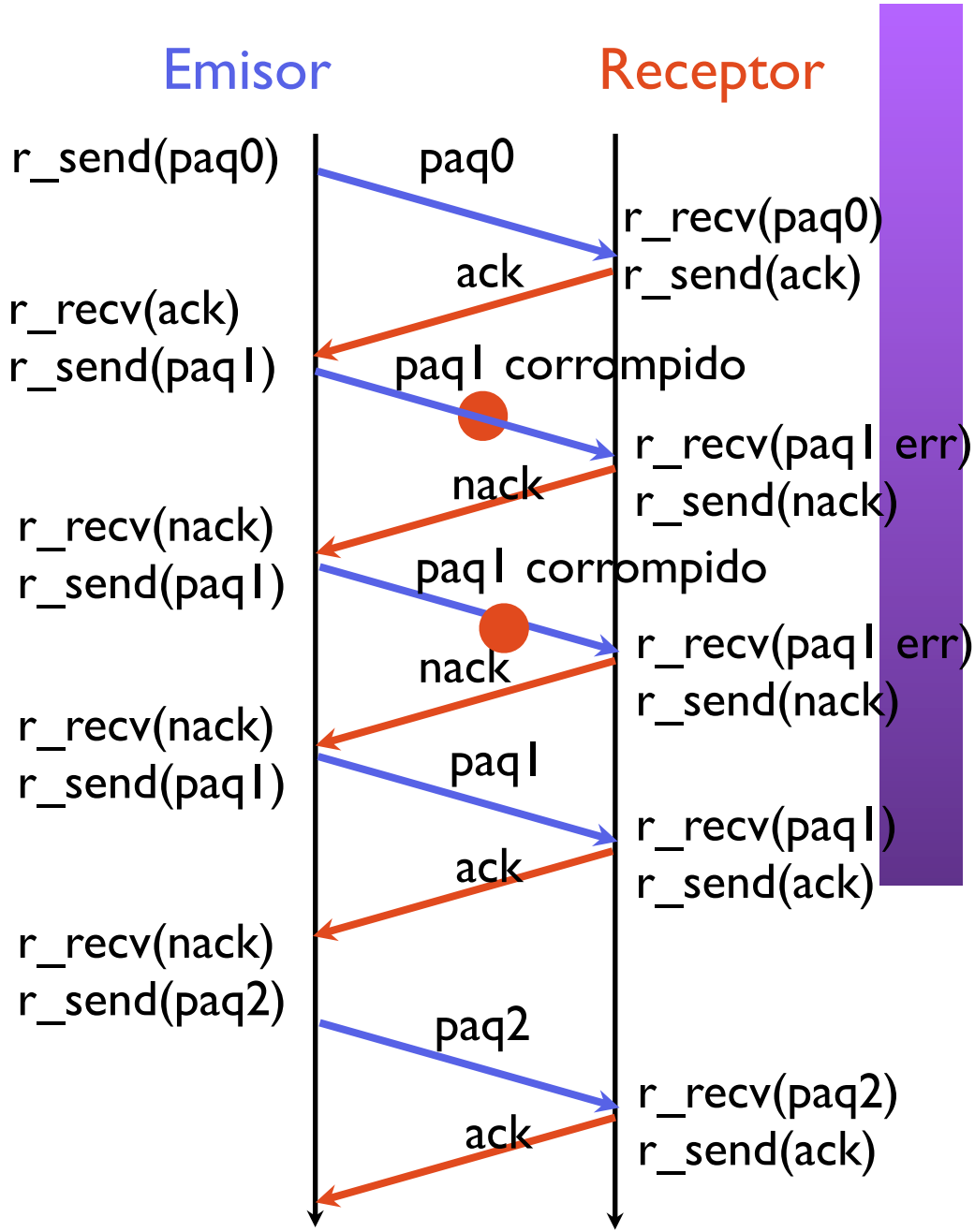
Protocolo de transporte fiable

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



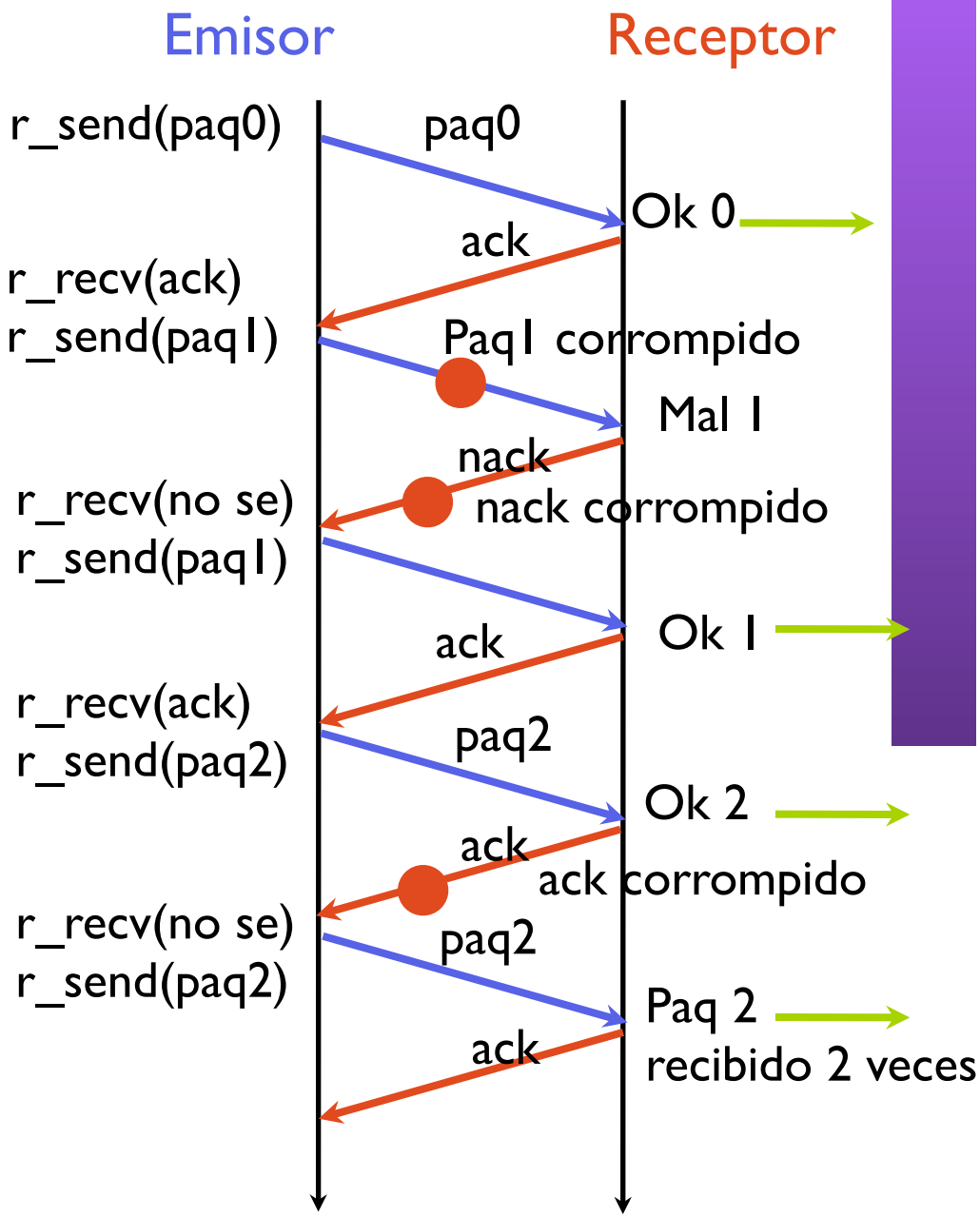
Stop-and-wait

- 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

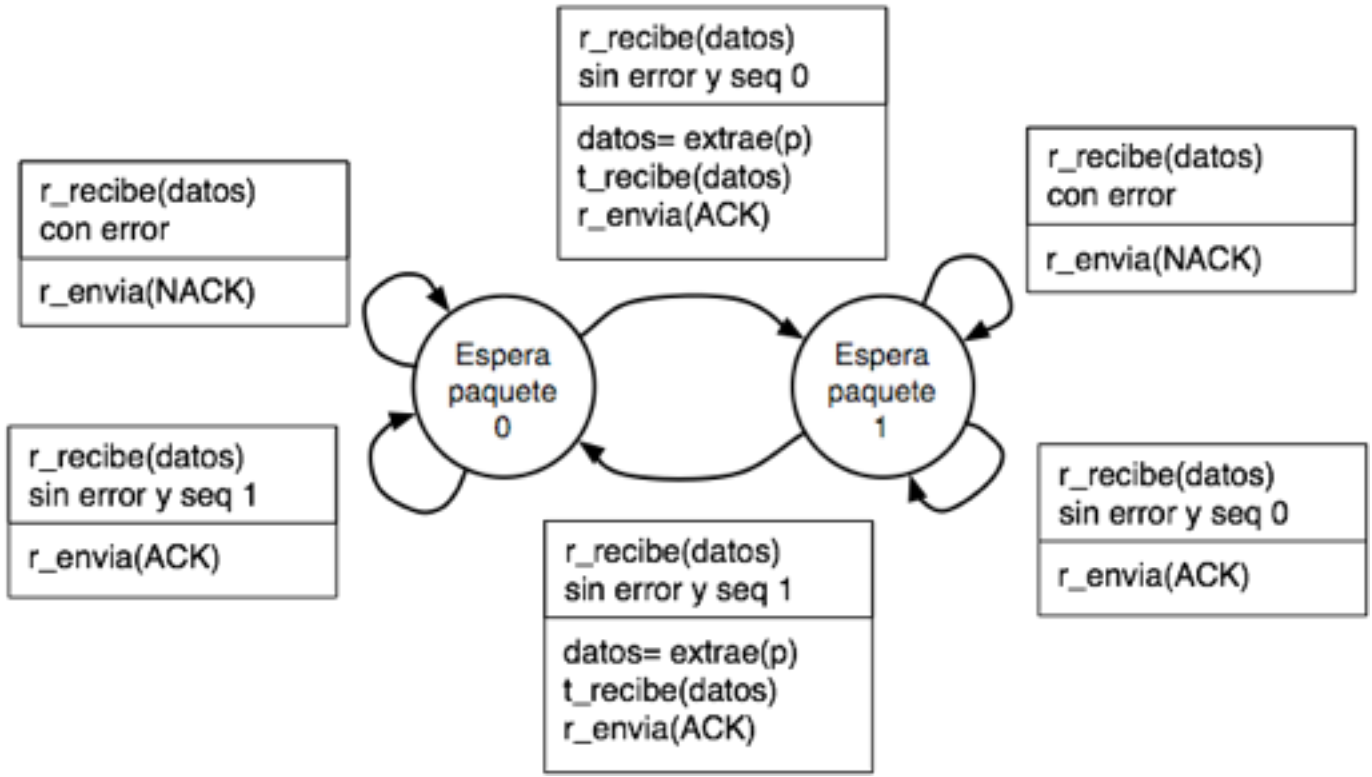
- Los protocolos más usados utilizan contra esto numeros de secuencia del paquete
- El paquete va etiquetado con un numero de secuencia que permite confirmralo/rechazarlo indicando cual
- El numero de secuencia es un campo del paquete por lo que podrá tener una serie finita de valores
- Aunque es fácil 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

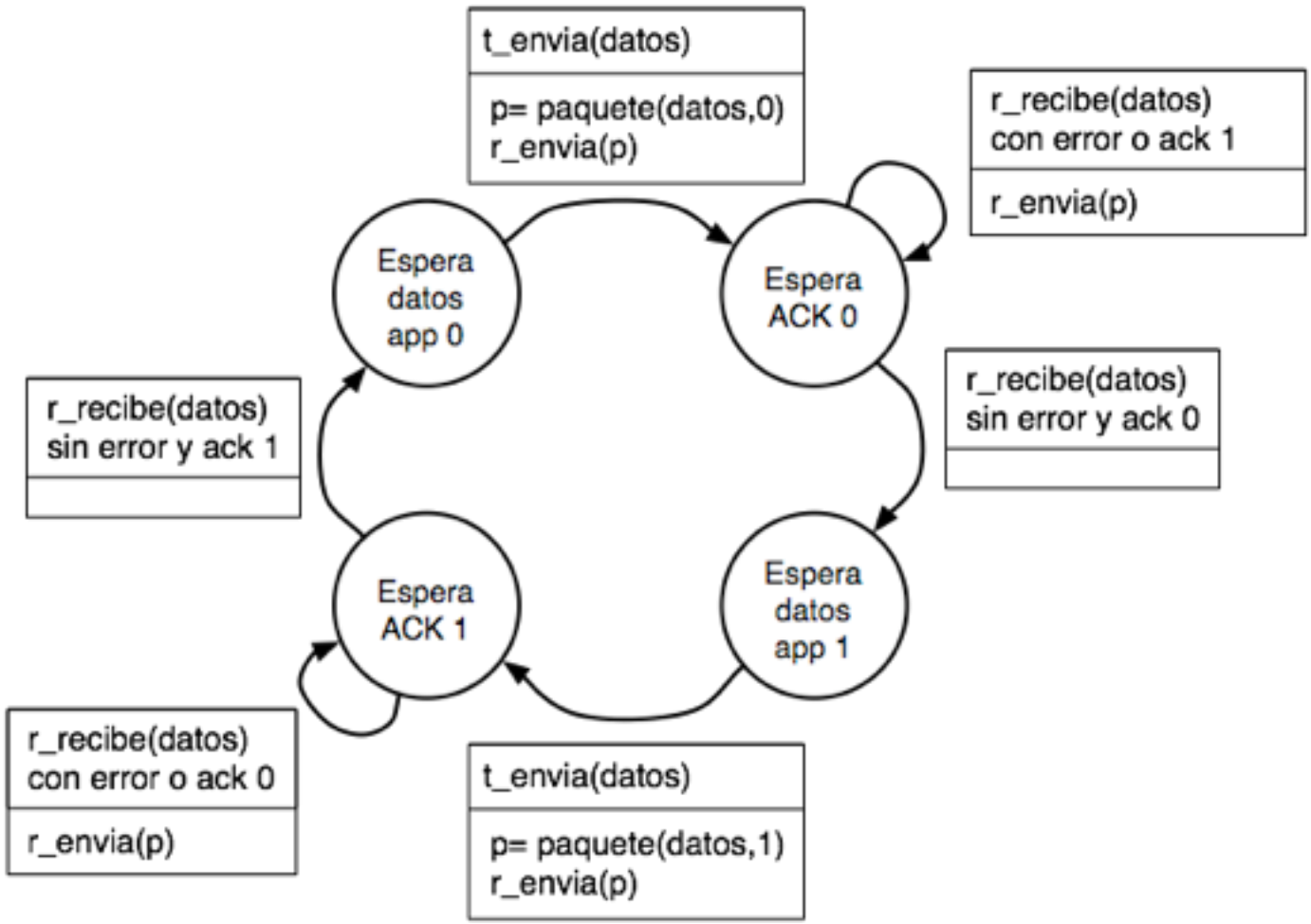
  Cada paquete de datos es secuencia 0 o 1
  - Si llega el que esperamos mandamos ACK y lo entregamos
  - Si llega el que no esperamos?

    mandamos ACK pero no son datos nuevos



Protocolo con número de secuencia

- Estados del emisor

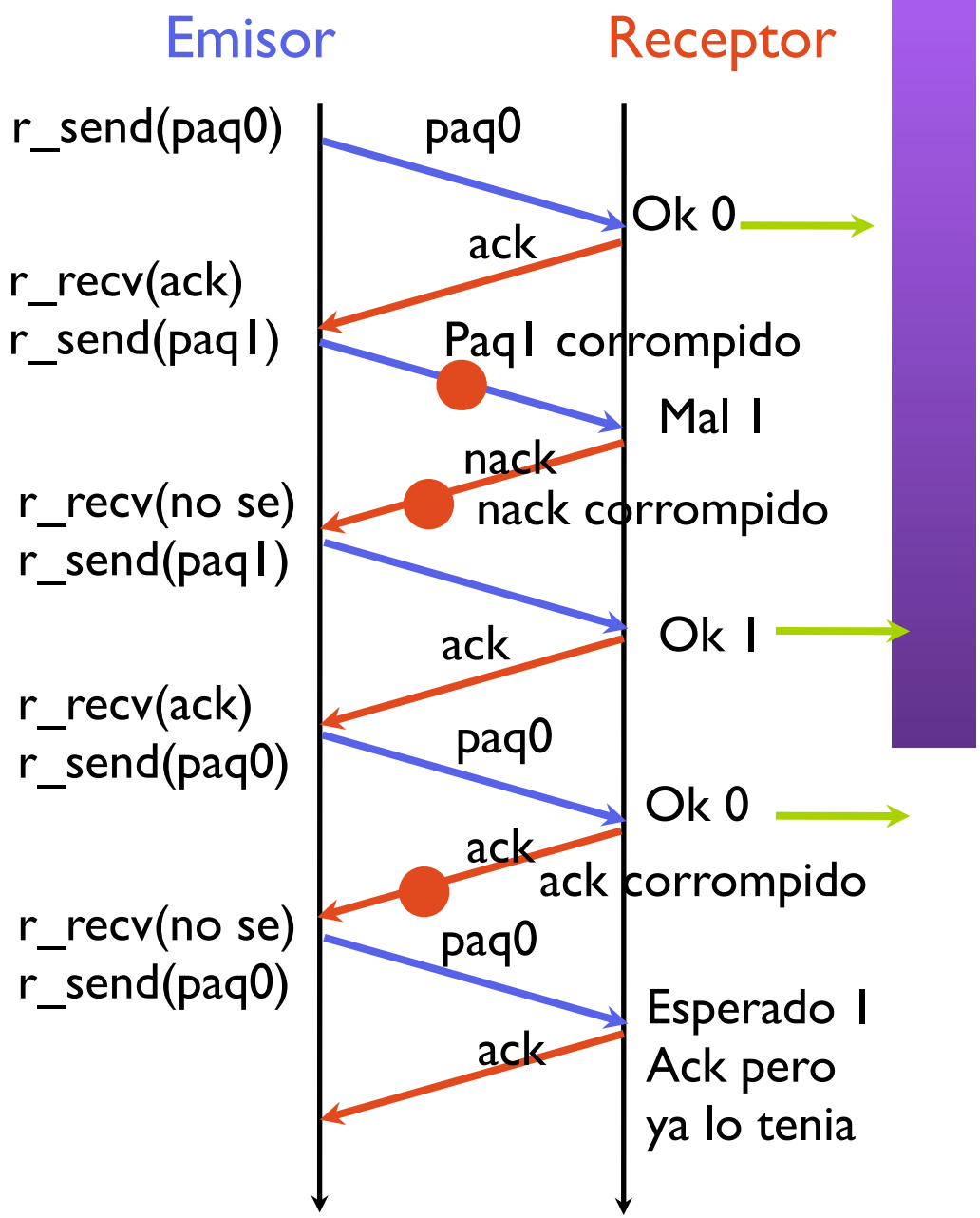


Ejemplo

- 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



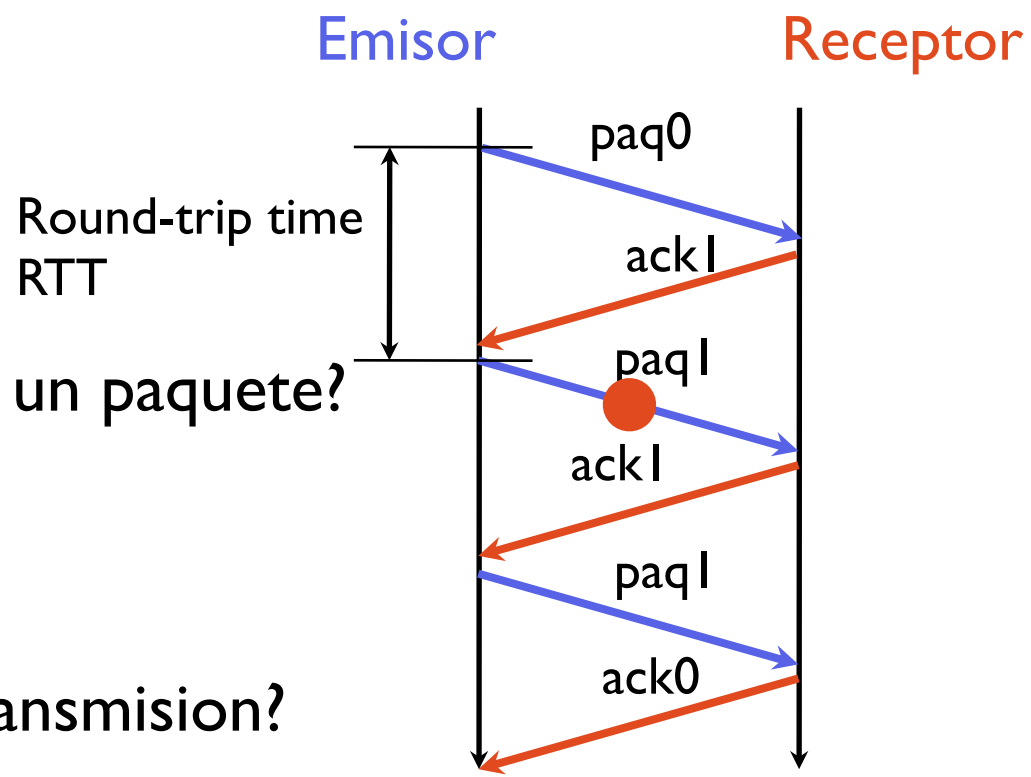
Hasta ahora

- 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

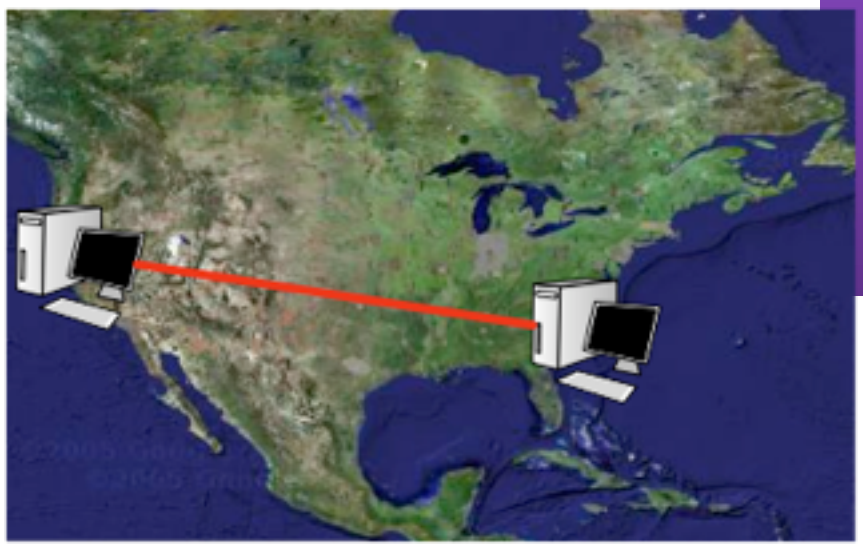
Prestaciones

- 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

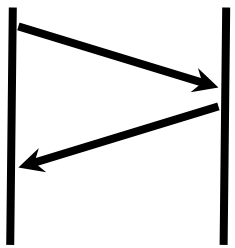
Ejemplo

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

  Suponiendo sin errores



RoundTripTime =30ms



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

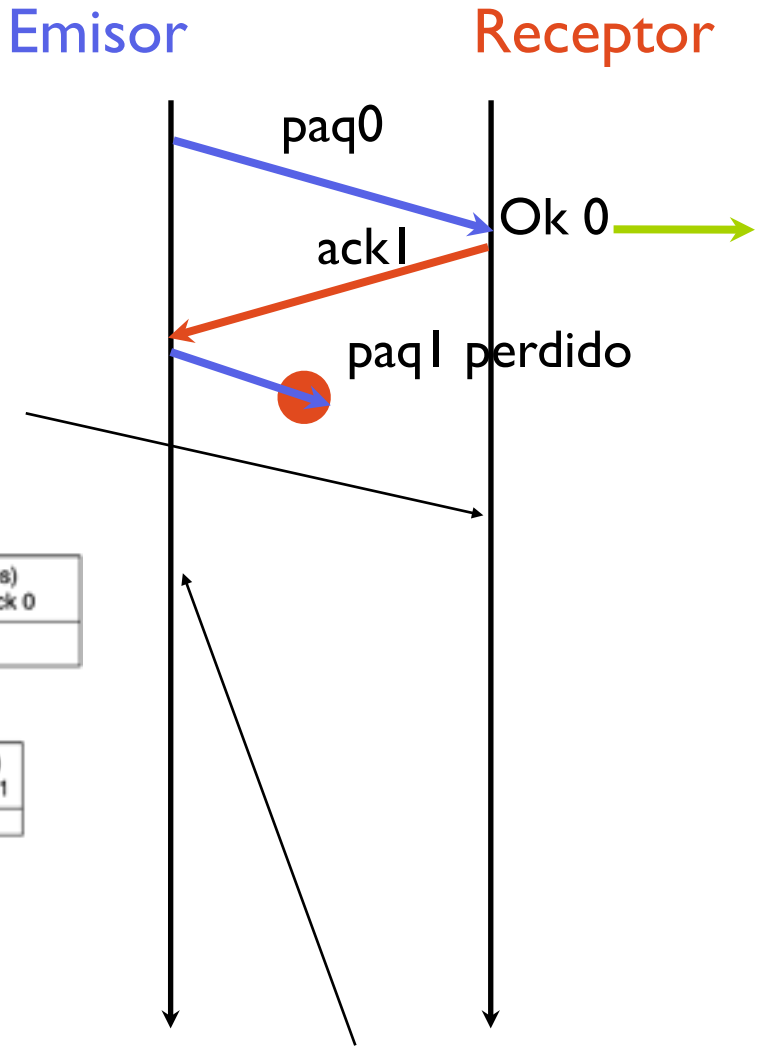
0.026% !! :-(

Pérdidas de paquetes

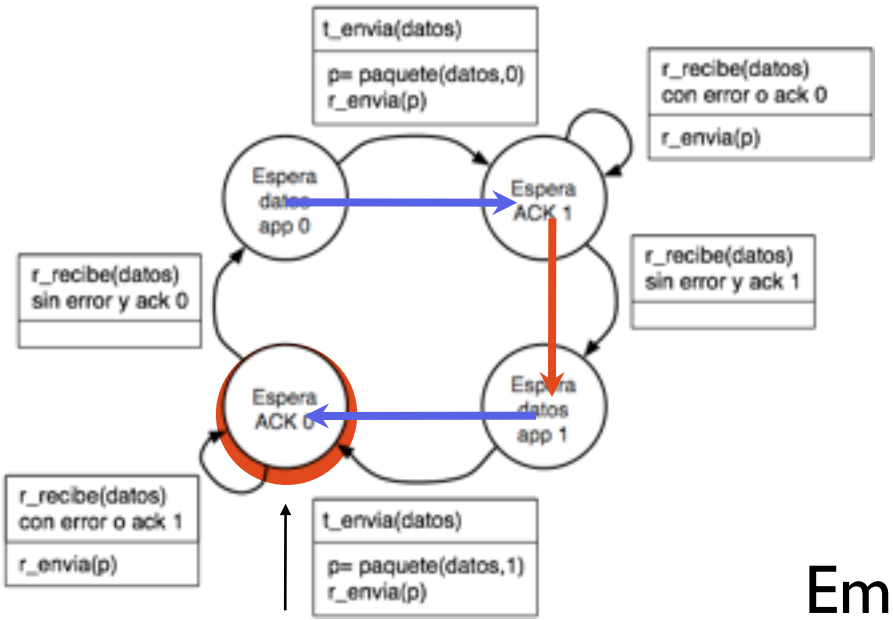
- El nivel de transporte recibe los paquetes que entrega el nivel de red
  - El nivel de red puede no garantizar la entrega de paquetes.
- Puede ser que un paquete entregado en el nivel de red del emisor nunca se entregue en el nivel de red del receptor
- Cómo afecta esto al protocolo anterior?

Pérdidas de paquetes

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



Receptor sólo manda ACKs cuando le llega algo

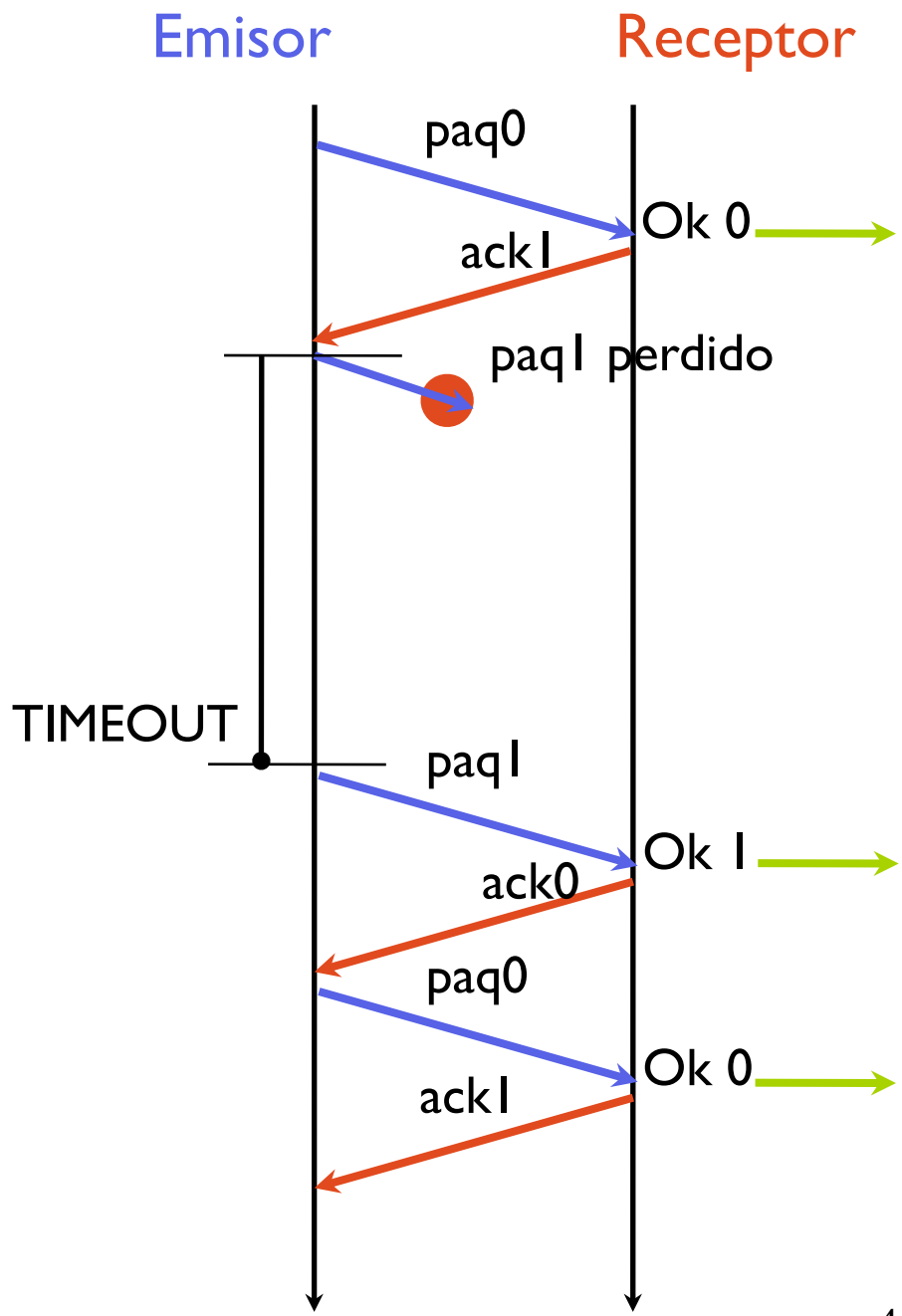


Sin recibir ACKs no puedo salir de este estado

Emisor no puede enviar hasta recibir el ACK

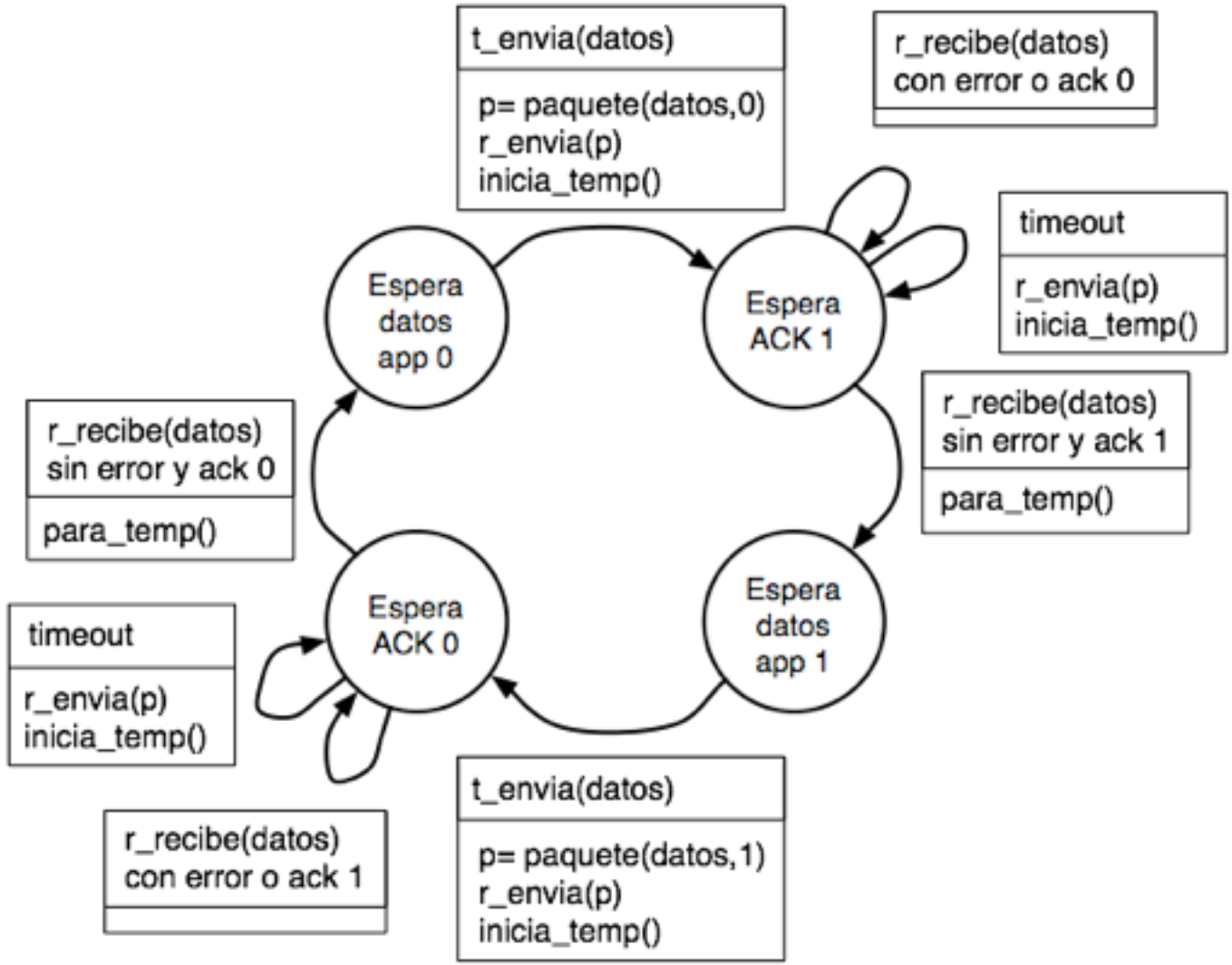
Pérdidas de paquetes

- 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

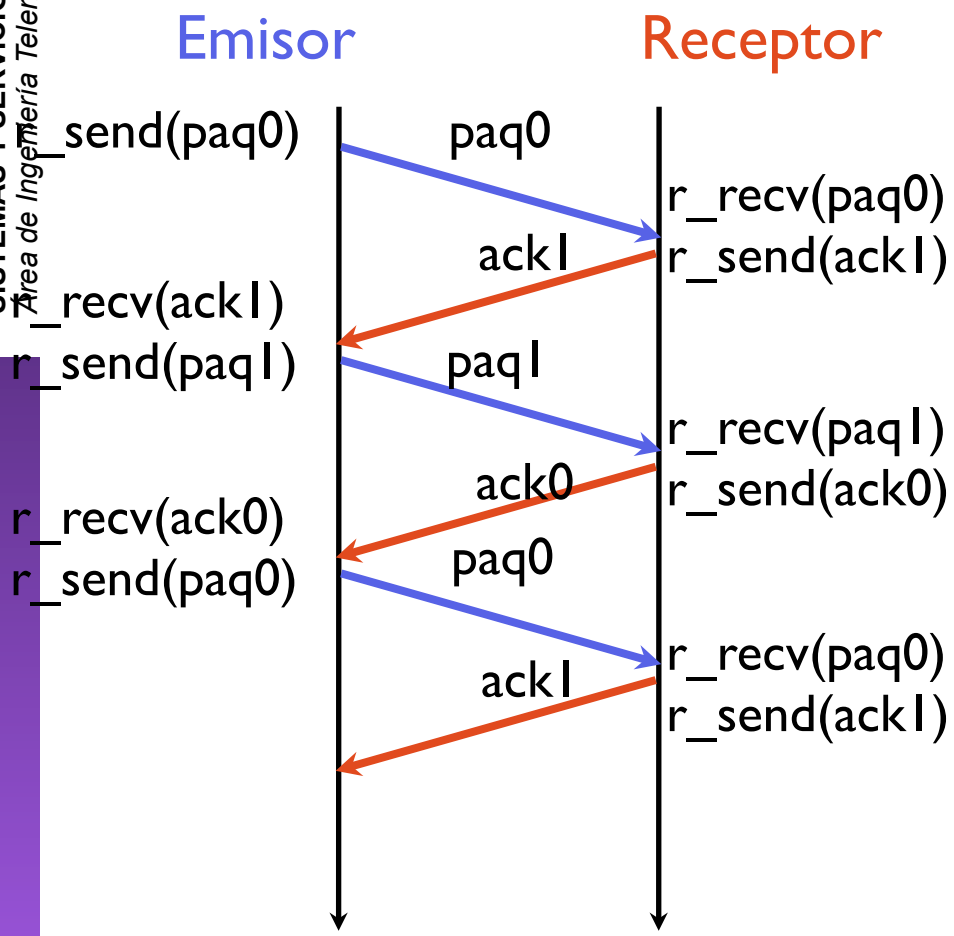


Protocolo con timeout

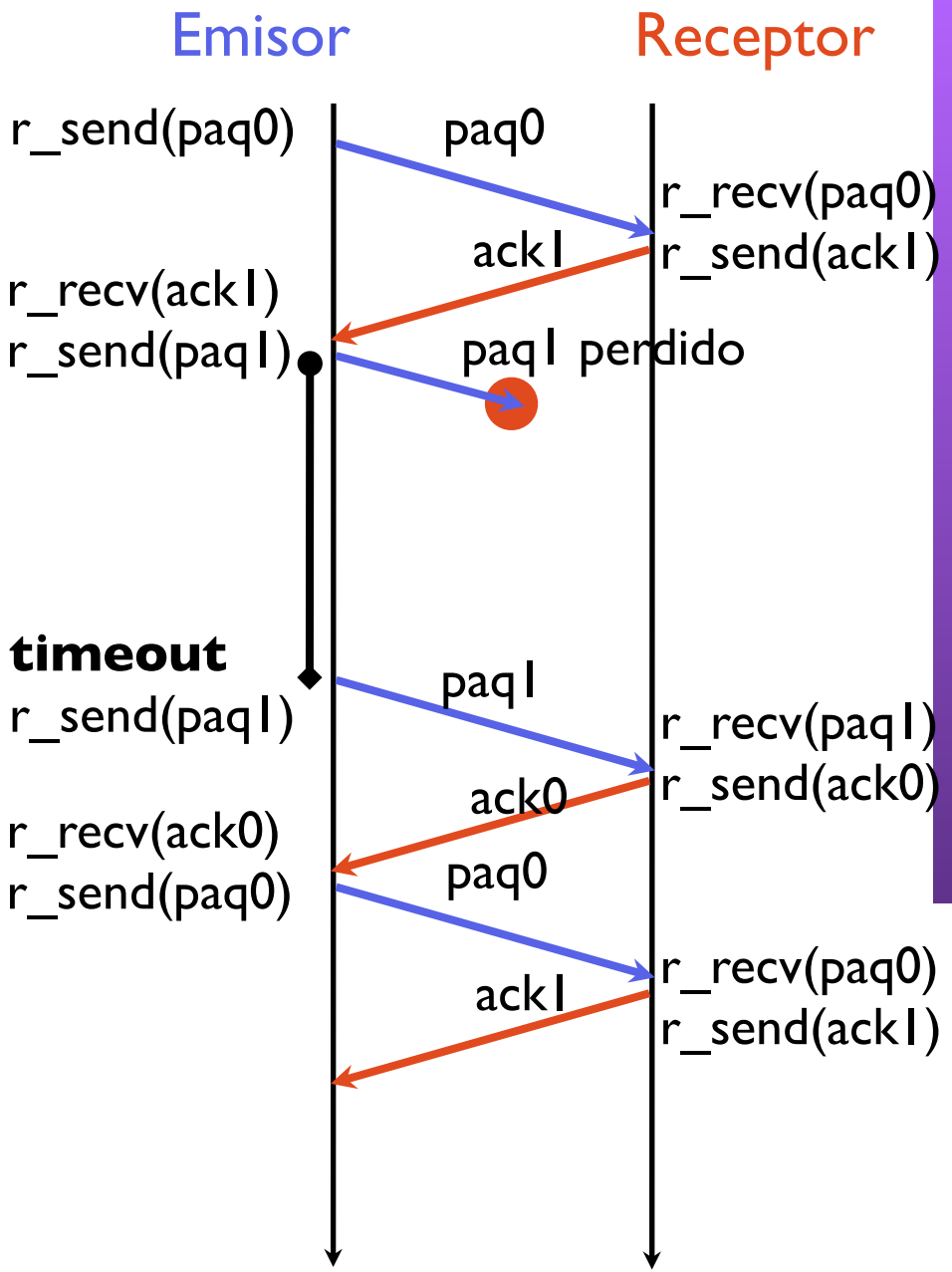
- Emisor con retransmisión por timeout



Ejemplos

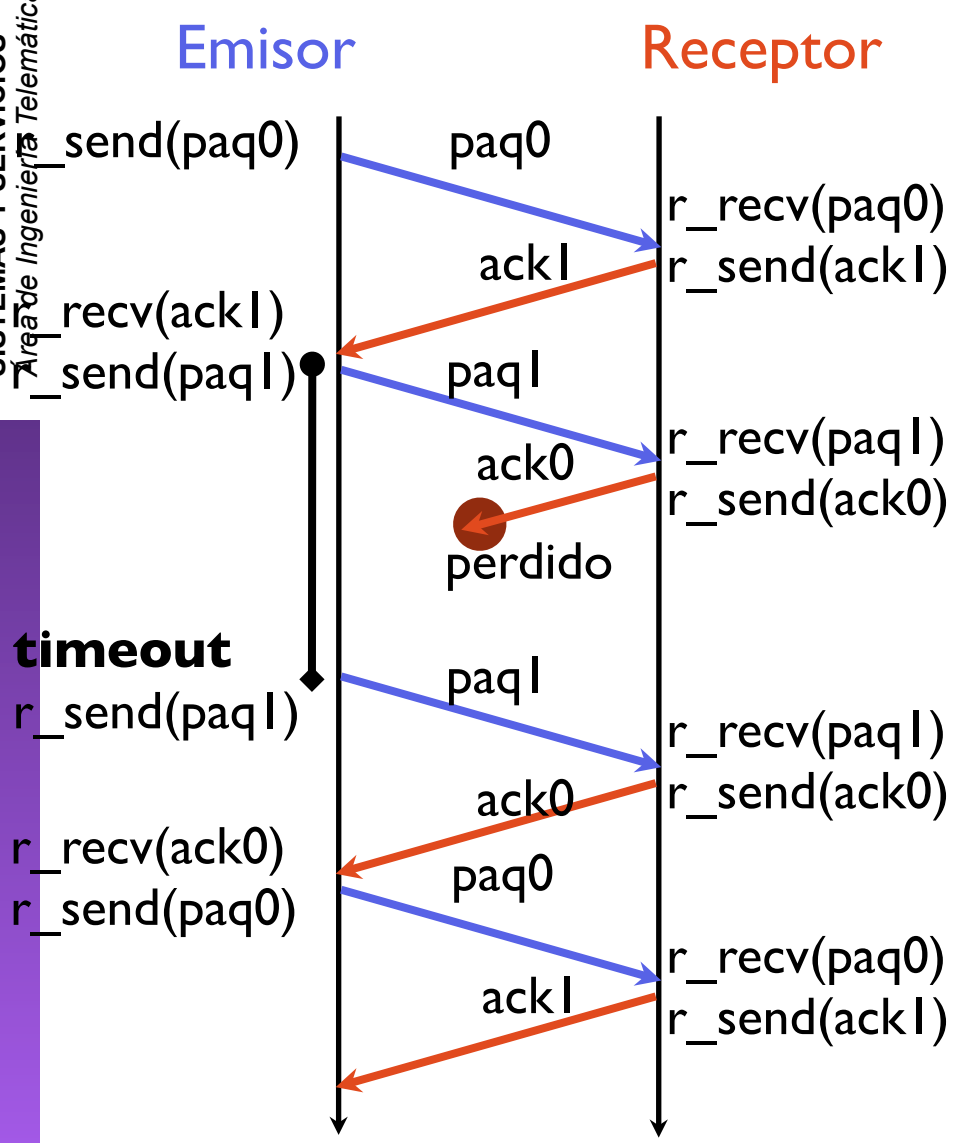


Operación normal

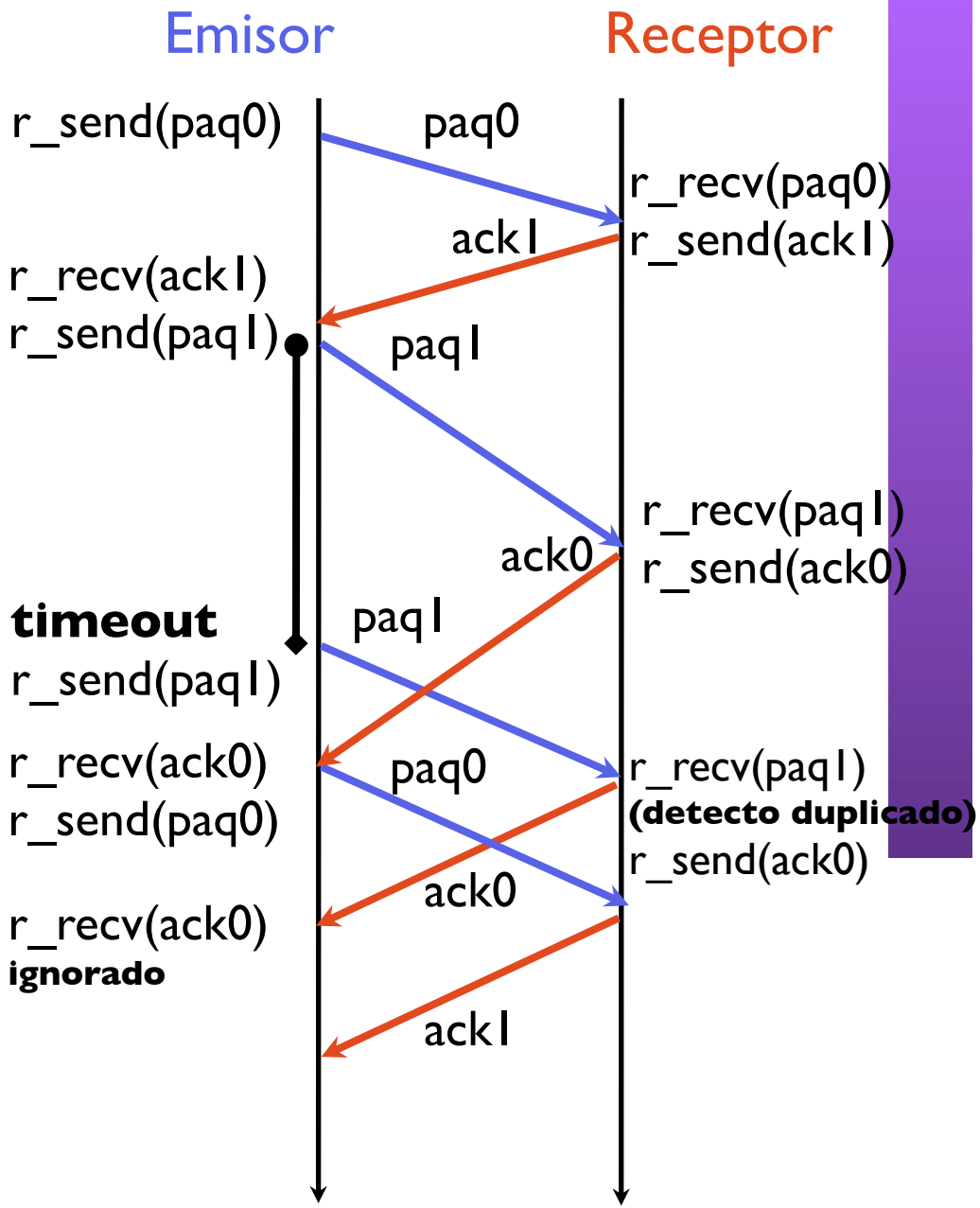


Pérdida de paquete

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

- 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)

Conclusiones

- Hay mecanismos y protocolos que permiten conseguir un transporte fiable sobre una red no fiable
- Pero y las prestaciones?

  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?

Próxima clase:

- transporte fiable con mejores prestaciones (+problemas)
- protocolo de transporte fiable de Internet TCP (+problemas)