Que és?

- Protocol molt senzill de transferència de fitxers
  - S'utilitza en la carrega inicial del S.O. a través de la xarxa, gravat en PROM d'arrancada
- Es pot implementar sobre diferents serveis
  - Normalment sobre UDP
- Permet llegir/gravar fitxers del/al Servidor

TFTP sobre UDP unicast

- Client:
  - utilitza un port (TID) aleatori
    - canvia en cada sol·licitud
- Servidor:
  - espera sol·licituds en el port 69
    - El Servidor contesta per un port (TID) aleatori
- Està preparat pels tipus d'error d'UDP
  - pèrdua, duplicació o desordre

Tipus de paquets:

TFTP utilitza cinc tipus de paquets:

Tipus	Acrònim	Operació
1	RRQ	Sol·licitud de lectura
2	WRQ	Sol·licitud d'escriptura
3	DATA	Paquet de dades
4	ACK	Paquet d'assentiment
5	ERR	Paquet d'error

Formats dels missatges de TFTP:

Paquets RRQ i WRQ, sol·licitud de lectura/escriptura d'un fitxer

2 bytes	string	1 byte	string	1 byte
codi **RRQ=1** **WRQ=2**	**nom del fitxer**	**0**	**mode**	**0**

Paquets DATA, blocs de dades

2 bytes	2 bytes	n bytes
codi **DATA=3**	**número de bloc**	**dades**

Paquets ACK, assentiments

2 bytes	2 bytes
codi ACK=4	**número de bloc**

Paquets ERR, missatge d'error

2 bytes	2 bytes	String	1 byte
codi ERROR=5	**codi de l'error**	**missatge d'error**	**0**

Esdeveniments del model

- Arribada/enviament de trames

  Read Request: RRQ, <fich>, netascii

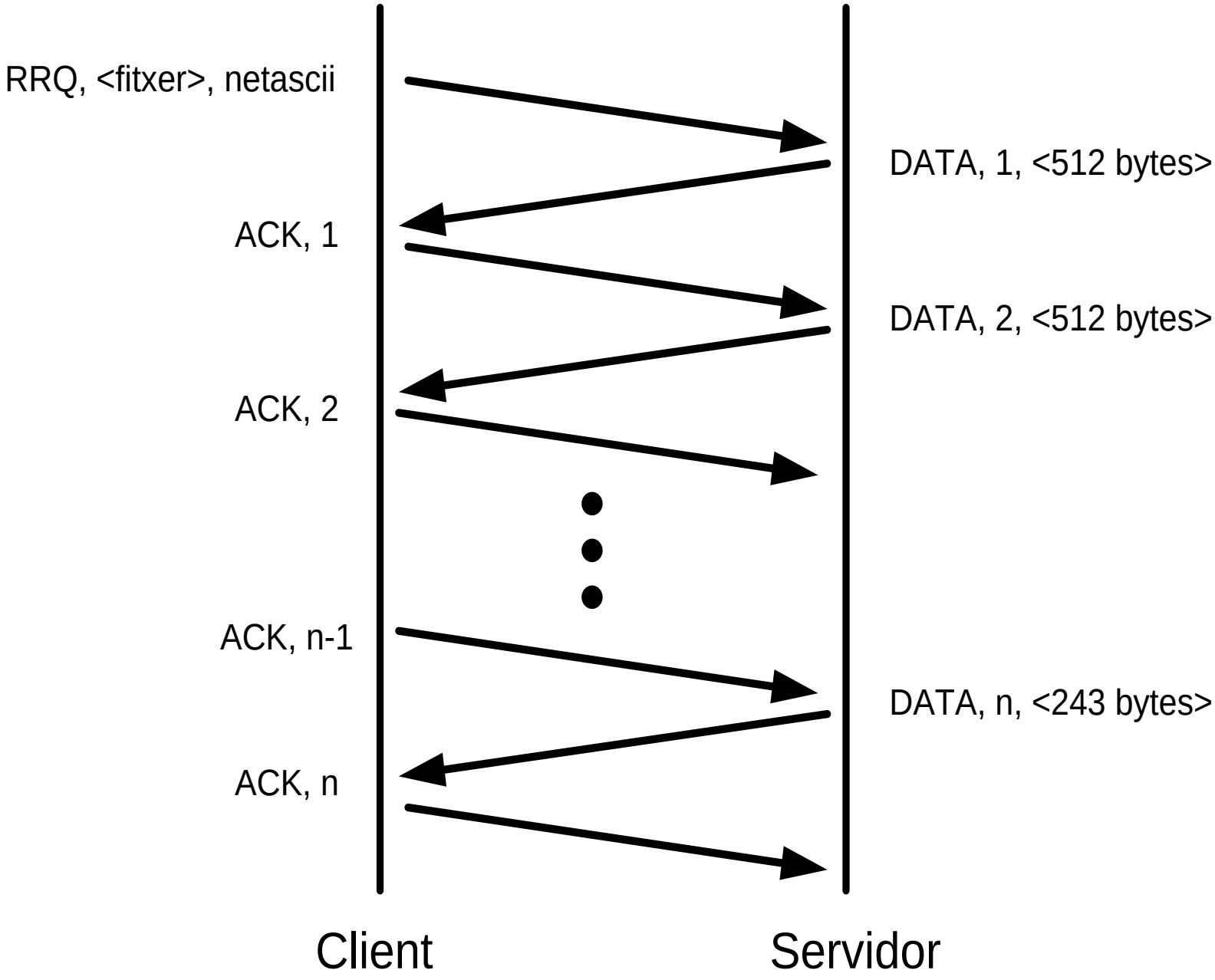
- Write Request: WRQ, <fich>, netascii

  Dades: DATA n (512)
  Dades (final): DATA n (<512)
  Assentiment: ACK n

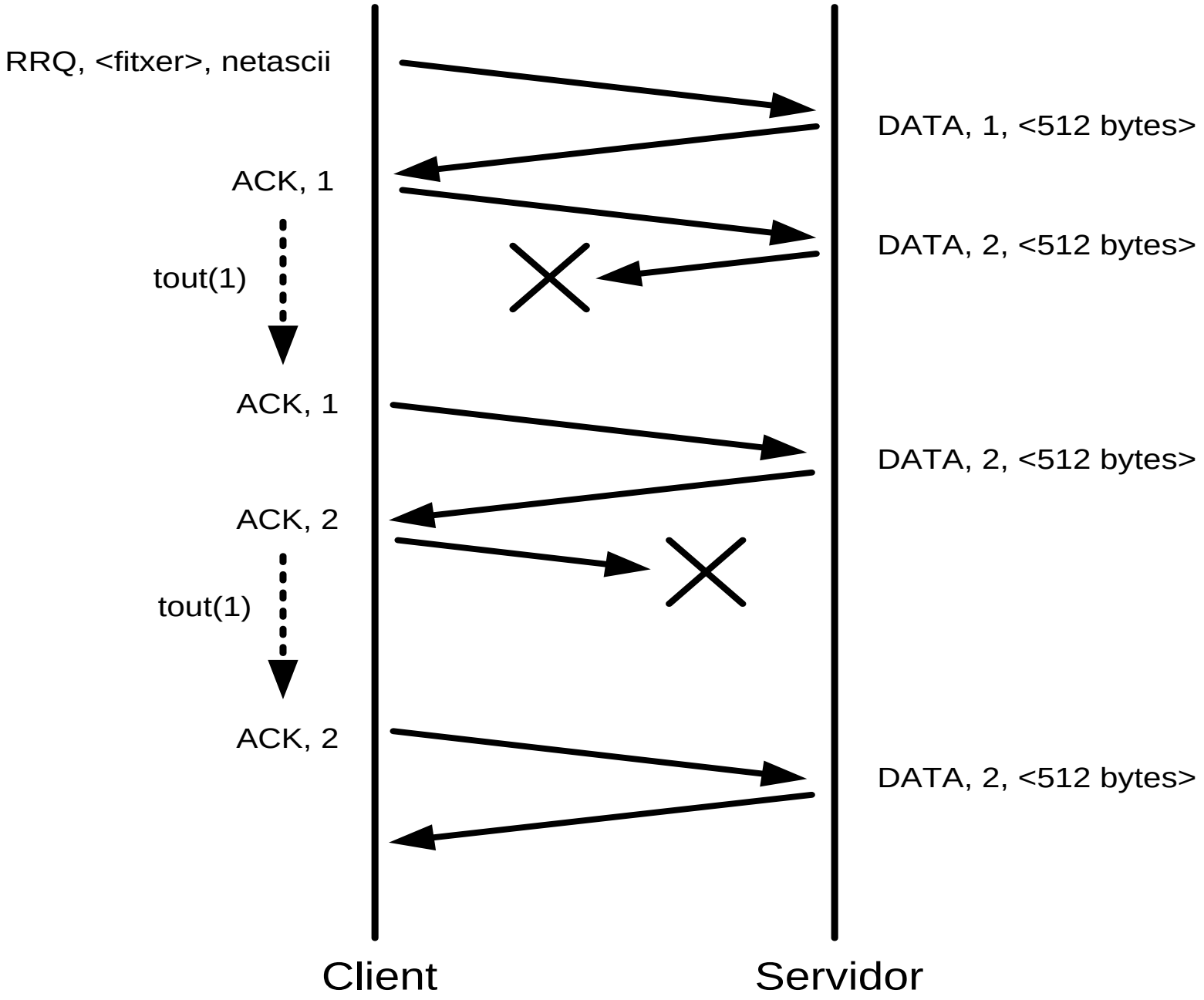
- Temporitzadors

  Retransmissió: time out

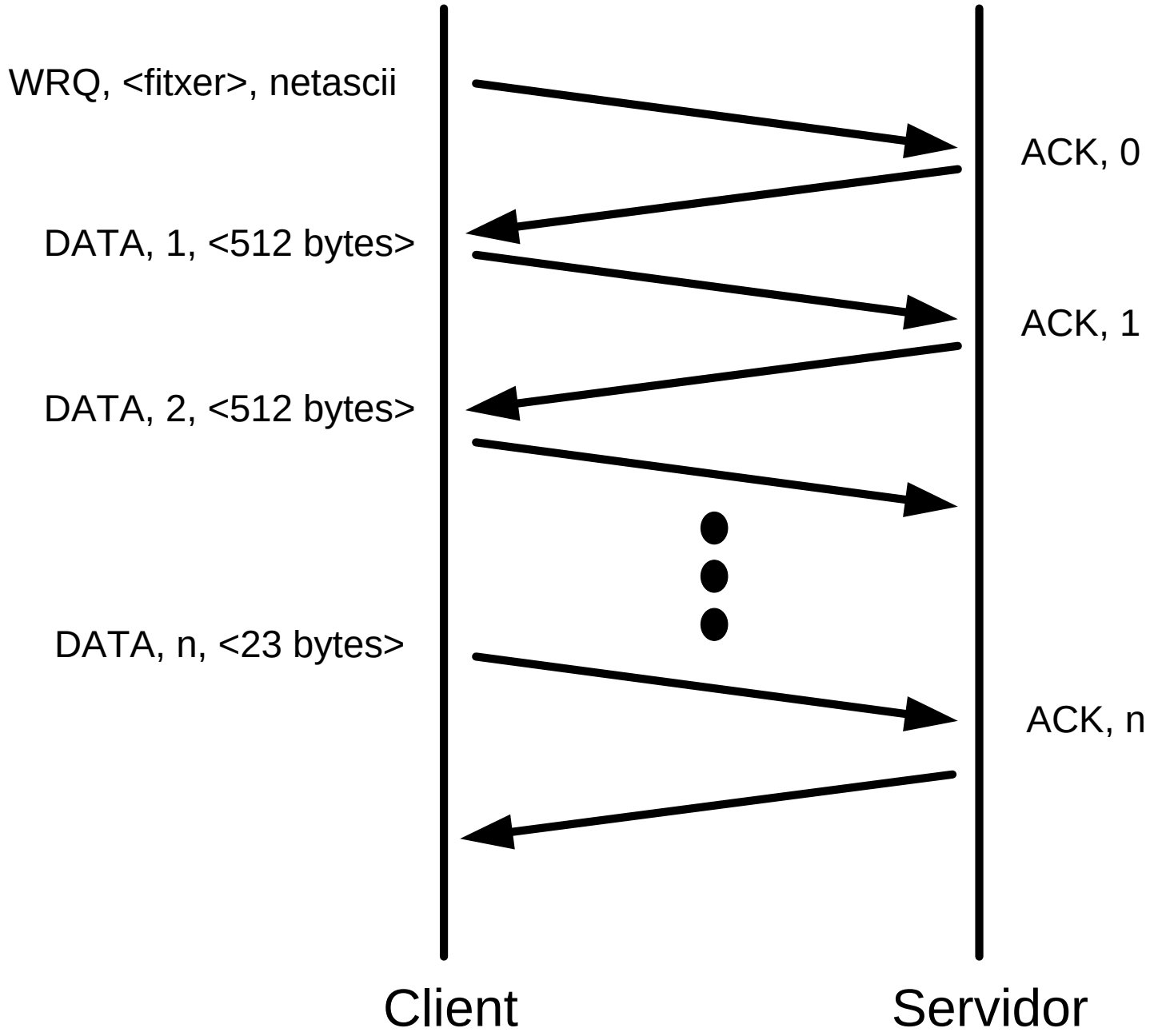
Lectura remota d'un fitxer:



Lectura remota d'un fitxer (errors):



Escriptura remota d'un fitxer:



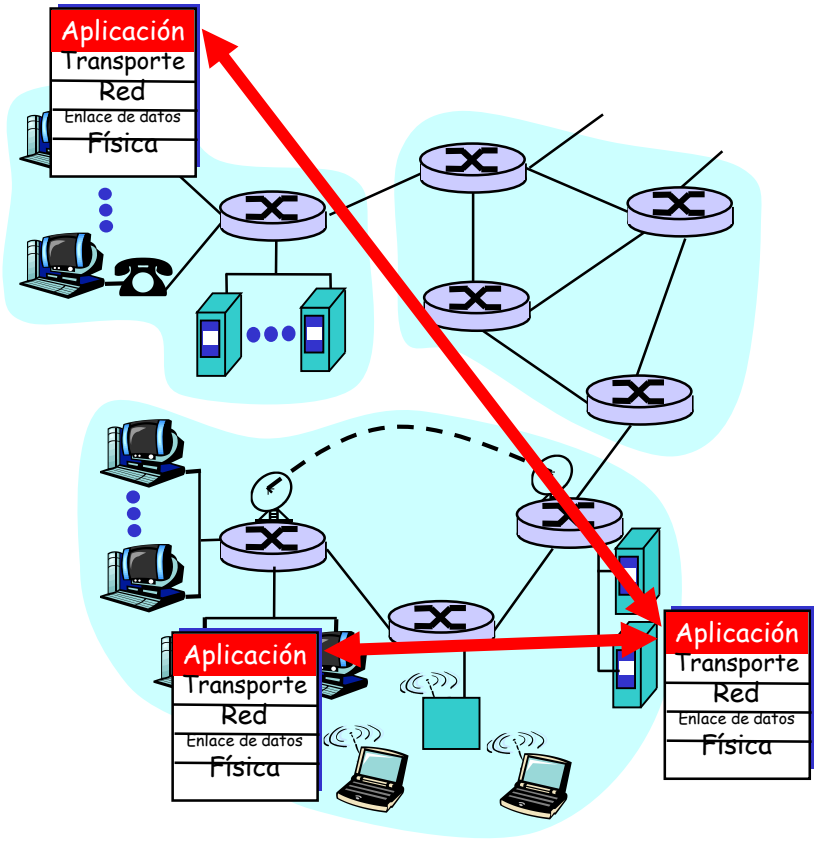
Aplicaciones y protocolos de la capa de aplicación

Aplicación: comunicación, procesos distributivos.

- Por ejemplo: correo electrónico, web, compartición de archivos entre iguales, mensajería instantánea.
- Funcionamiento en sistemas finales (hosts).
- Intercambio de mensajes para la implementación de aplicaciones.

Protocolos de capa de aplicación

- Una "parte" de la aplicación.
- Definición de mensajes intercambiados entre las aplicaciones y las acciones tomadas.
- Uso de servicios de comunicación proporcionados por los protocolos de capas inferiores (TCP, UDP).



Características de los protocolos de capa de aplicación

- Los tipos de mensajes intercambiados, por ejemplo, mensajes de petición y de respuesta.
- Sintaxis de los tipos de mensajes: qué campos hay en los mensajes y cómo están delineados estos campos.
- Semántica de los campos, por ejemplo, significado de la información en los campos.
- Reglas que determinan cuándo y cómo los procesos envían y responden a los mensajes.

Protocolos de dominio público:

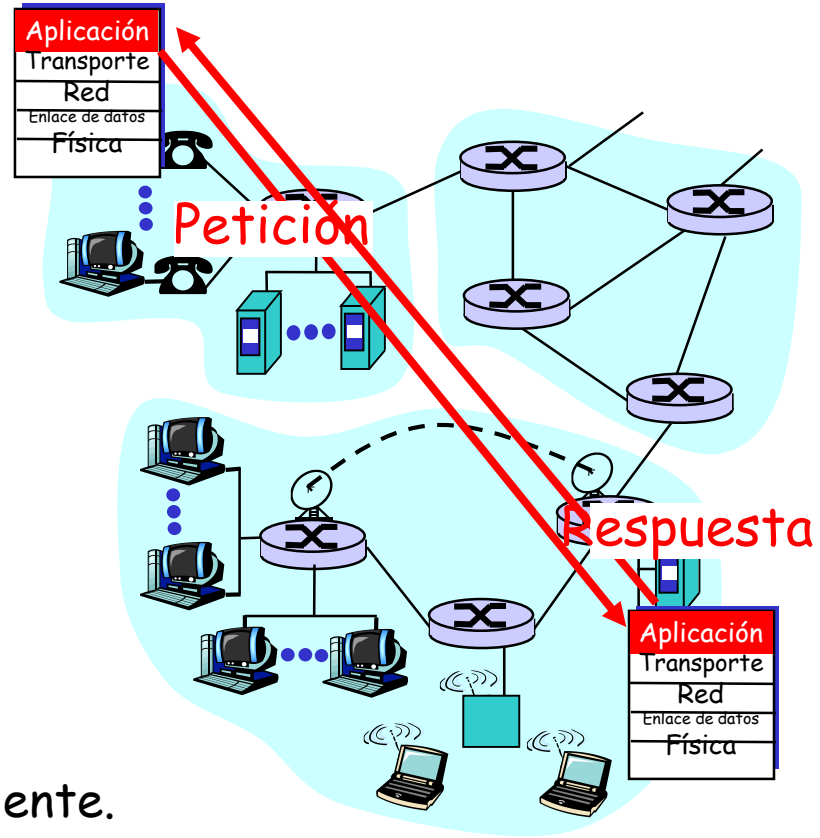
- Definidos en RFC.
- Permiten interoperabilidad.
- Por ejemplo: HTTP, SMTP, TFTP.

Protocolos de propietarios:

- Por ejemplo: KaZaA.

Paradigma del cliente-servidor

La aplicación de red típicamente tiene dos partes: *el cliente* y *el servidor*



Cliente:

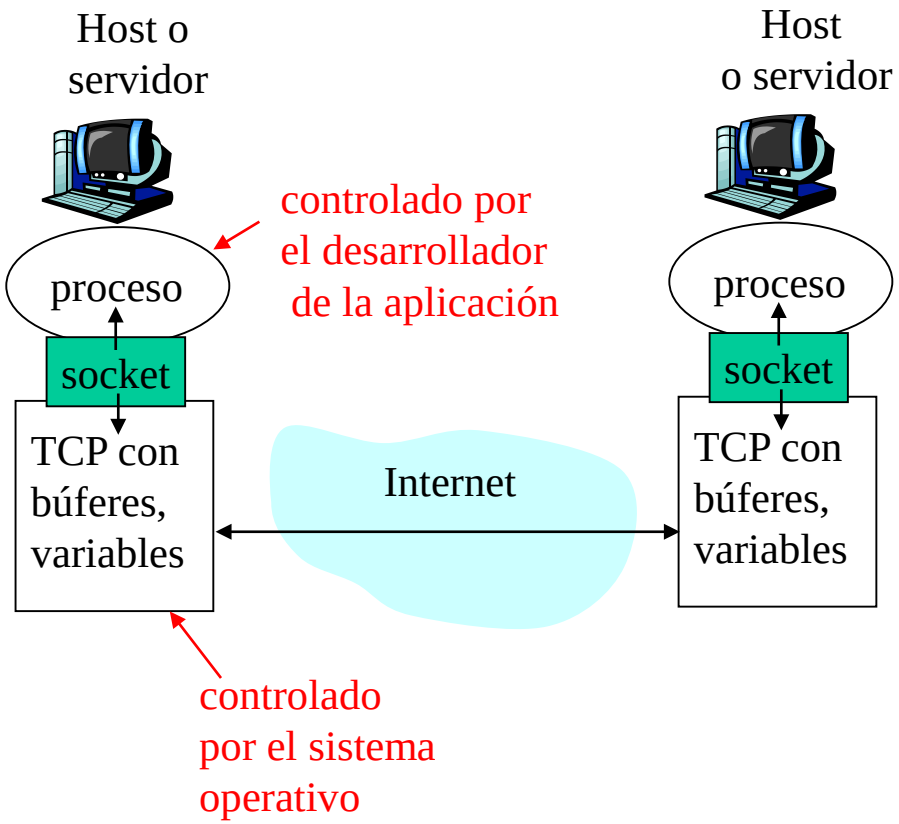
- Inicia el contacto con el servidor ("habla primero").
- Normalmente solicita un servicio del servidor.
- Web: cliente implementado en el navegador; correo electrónico: en el lector de correo.

Servidor:

- Proporciona el servicio solicitado al cliente.
- Por ejemplo, el servidor de Web envía la página Web solicitada, el servidor de correo entrega el correo electrónico.

Procesos que se comunican a través de la red

- El proceso envía/recibe mensajes a/de su socket.
- El socket es análogo a una puerta:
  - El proceso emisor empuja el mensaje por su puerta.
  - Este proceso asume la existencia de una infraestructura de transporte al otro lado de la puerta que transportará el mensaje al socket en el proceso receptor.



- API: (1) elección del protocolo de transporte; (2) posibilidad de fijar algunos parámetros (más adelante se tratará este tema con mucha más profundidad).

Direccionamiento de procesos:

- Para que un proceso reciba mensajes debe tener un identificador.
- Cada host tiene una única dirección IP de 32 bits.
- Pregunta: ¿Basta con la dirección IP del host, en el que el proceso se ejecuta, para identificar el proceso?
- Respuesta: No, ya que muchos procesos diferentes pueden estar ejecutándose en el mismo host.
- El identificador incluye tanto la dirección IP como los números de puerto asociados con el proceso del host.
- Ejemplos de números de puerto:
  - Servidor HTTP: 80.
  - Servidor de correo: 25.
- Más adelante se tratará este tema con más profundidad.

¿Qué servicio de transporte necesita una aplicación?

Pérdida de datos

- Ciertas aplicaciones (por ejemplo, audio) pueden tolerar algunas pérdidas.
- Otras aplicaciones (por ejemplo, transferencia de archivos, Telnet) requieren el 100 por ciento de transferencia fiable de datos.

Temporización

- Algunas aplicaciones (por ejemplo, telefonía de Internet, juegos interactivos) requieren un retardo lento para ser "efectivas".

Ancho de banda

- Algunas aplicaciones (por ejemplo, multimedia) requieren un mínimo de ancho de banda para ser "efectivas".
- Otras aplicaciones ("aplicaciones flexibles") hacen uso de cualquier ancho de banda que tengan a su disposición.

Requisitos de los servicios de transporte para aplicaciones comunes

Aplicación	Pérdida de datos	Ancho de banda	Sensible al tiempo
Transferencia de archivos	No pérdida	flexible	No
Correo electrónico	No pérdida	flexible	No
Documentos Web	No pérdida	flexible	No
Audio/vídeo de tiempo real	Tolerante	Audio: 5Kbps-1Mbps Vídeo:10Kbps-5Mbps	Sí, 100 mseg
Audio/vídeo almacenado	Tolerante	Igual que el anterior	Sí, pocos seg
Juegos interactivos	Tolerante	Pocos Kbps-10Kbps	Sí, 100 mseg
Mensajería instantánea	No pérdida	Flexible	Sí y no

Servicios de los protocolos de transporte de Internet

Servicio TCP:

- *Orientado a la conexión:* Sistema requerido entre el cliente y el servidor.
- *Transporte fiable* entre el proceso emisor y el receptor.
- *Control de flujo:* el emisor no debe sobrecargar al receptor.
- *Control de congestión:* regulación del emisor si la red se sobrecarga.
- *No proporciona:* temporización, garantías de un ancho de banda mínimo.

Servicio UDP:

- Transferencia de datos no fiable entre el proceso emisor y el receptor.
- No proporciona: sistema de conexión, fiabilidad, control de flujo, control de congestión, temporización y garantía de ancho de banda.

PREGUNTA: ¿Por qué tomarse la molestia? ¿Por qué existe un UDP?

Programación de sockets

Objetivo: aprender cómo se construye una aplicación cliente/servidor que se comunique utilizando sockets.

Socket API

- Introducido por BSD4.1 UNIX, 1981
- Creado, utilizado y puesto en circulación explícitamente por aplicaciones.
- Paradigma cliente/servidor.
- Dos tipos de transporte de servicio vía socket API:
  - Datagrama poco fiable.
  - Orientación del flujo de bytes fiable.

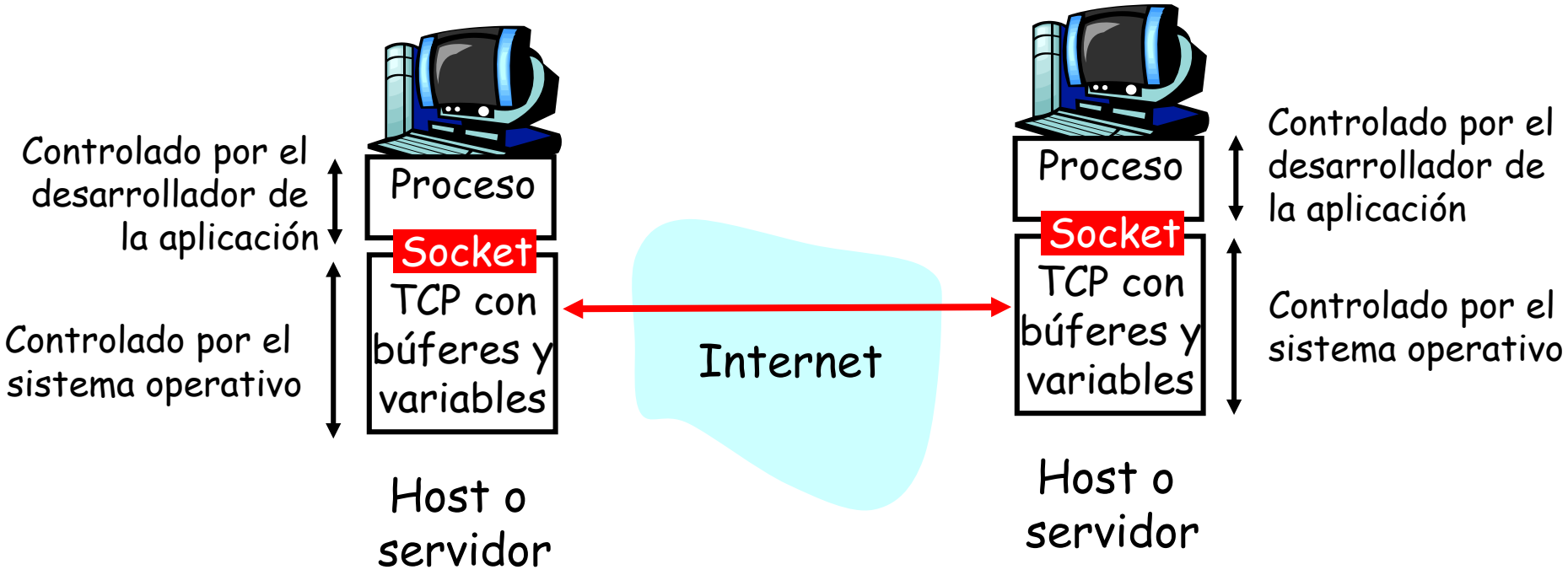
socket

Una interfaz *controlada por un sistema operativo, creada por una aplicación y con un host local*, (una "puerta") por la cual el proceso de aplicación puede tanto enviar como recibir mensajes de otro proceso de aplicación.

Programación de sockets usando TCP

Socket: una puerta entre el proceso de aplicación y el protocolo de transporte final (UCP o TCP).

Servicio TCP: transferencia fiable de **bytes** de un proceso a otro.



Programación de sockets *con TCP*

El cliente debe contactar con el servidor:

- El proceso del servidor en principio debe estar funcionando.
- El servidor debe haber creado un socket (puerta) para dar la bienvenida al contacto del cliente.

El cliente contacta con el servidor:

- Creando un socket TCP local del cliente.
- Especificando la dirección IP y el número de puerto del proceso del servidor.
- Cuando el cliente crea un socket: el cliente establece una conexión TCP con el servidor.

- Cuando el cliente contacta con él, el servidor TCP crea un nuevo socket para que el proceso servidor se comunique con el cliente.
  - Permite al servidor hablar con múltiples clientes.
  - Fuente de los números de puerto utilizados para distinguir a los clientes (más en el Capítulo 3).

Punto de vista de la aplicación

TCP propociona una transferencia de bytes en orden y fiable ("tubería") entre cliente y servidor.

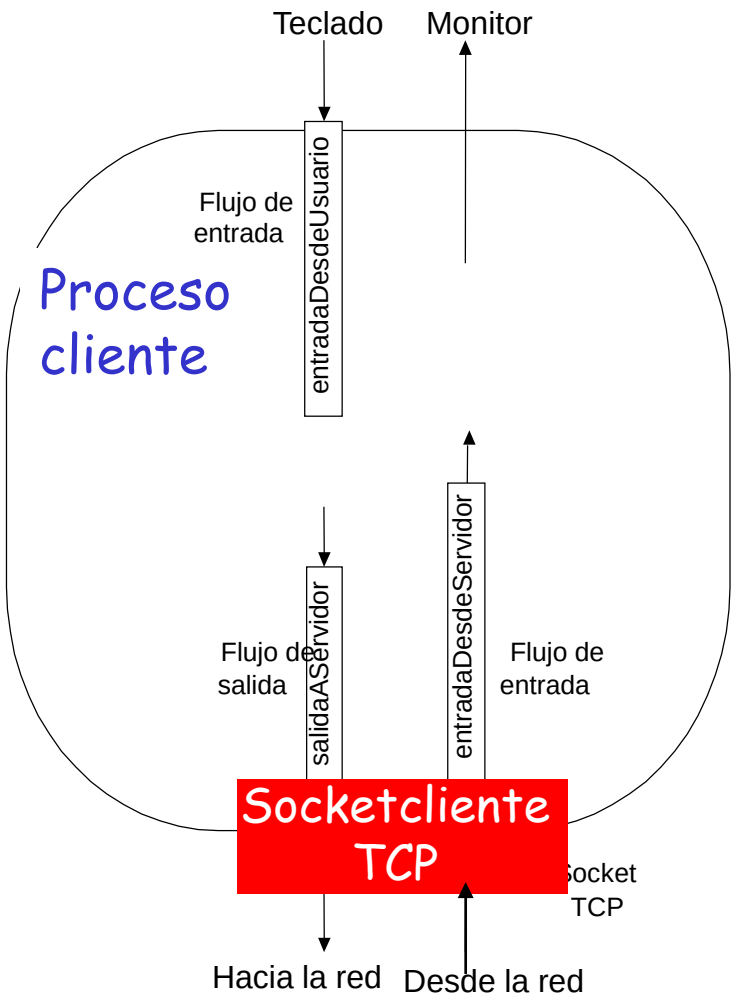
Jerga relacionada con el flujo

- Un flujo es una secuencia de caracteres que fluye hacia o desde un proceso.
- Un flujo de entrada está relacionado con una fuente de entrada del proceso, por ejemplo, el teclado o un socket.
- Un flujo de salida está relacionado con una fuente de salida, por ejemplo, el monitor o un socket.

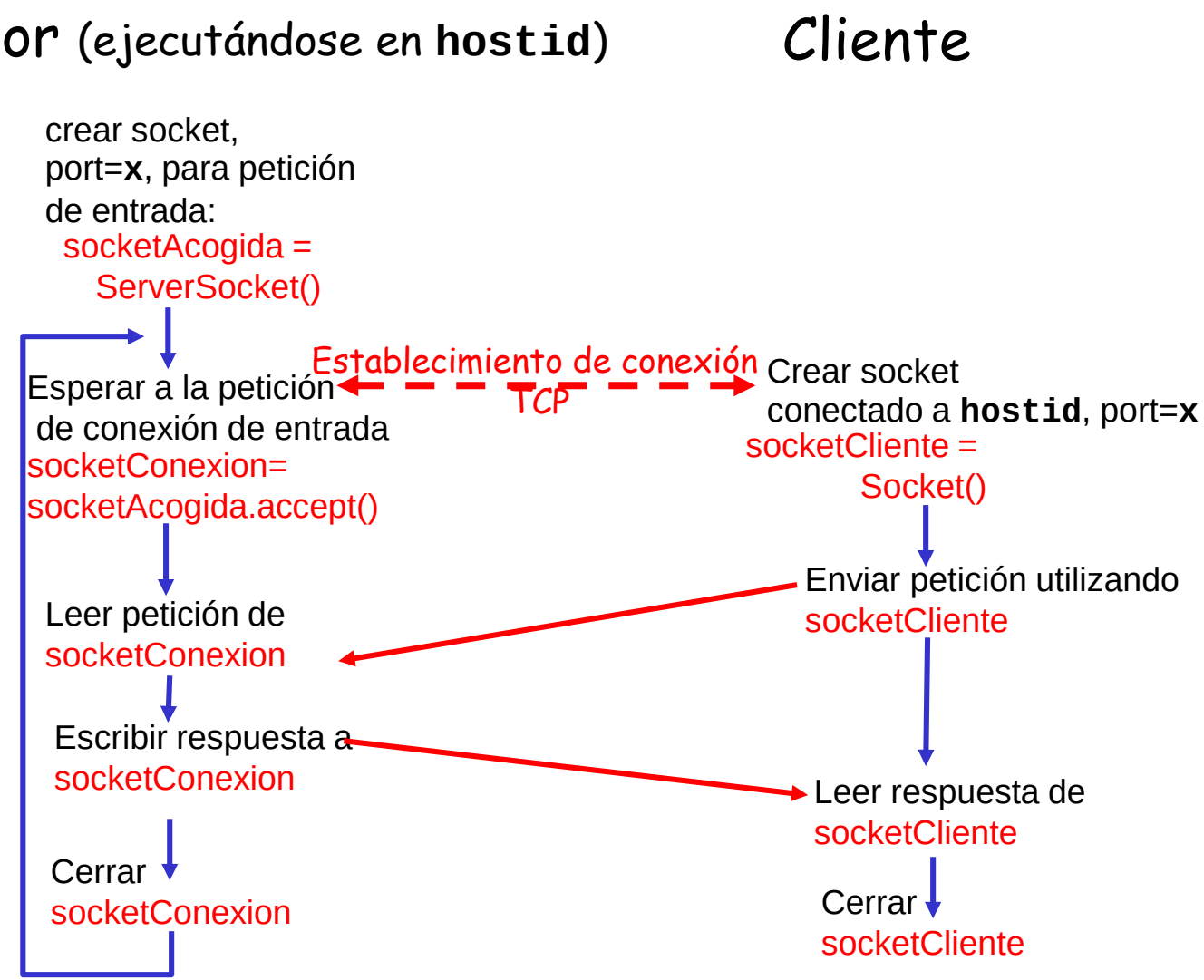
Programación de sockets con TCP

Ejemplo de aplicación cliente-servidor:

1) El cliente lee una línea de su entrada estándar (flujo **`inFromUser`**) , y se la envía al servidor por su socket (flujo **`outToServer`**).
2) El servidor lee una línea en su socket.
3) El servidor convierte la línea a mayúsculas y se la envía de vuelta al cliente.
4) El cliente lee e imprime la línea modificada de su socket (flujo **`inFromServer`**).



Interacción socket cliente/servidor: TCP



Programación de sockets *con UDP*

UDP: no hay "conexión" entre el cliente y el servidor.

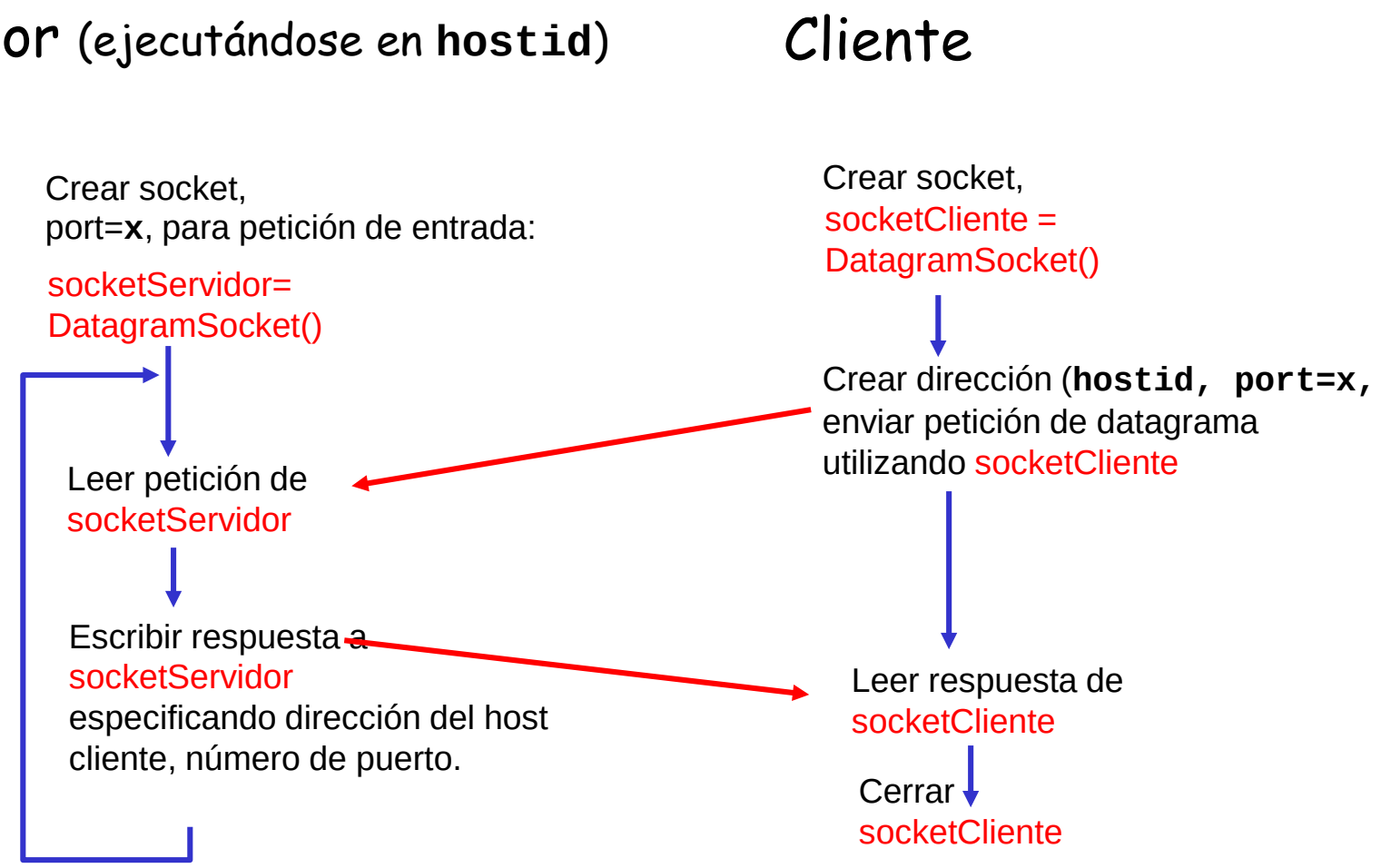
- No hay sincronización.
- El remitente añade explícitamente la dirección IP y el puerto de destino a cada paquete.
- El servidor debe extraer la dirección IP y el puerto del remitente del paquete recibido.

UDP: Los datos transmitidos puede que se reciban desordenados o que se pierdan.

Punto de vista de la aplicación

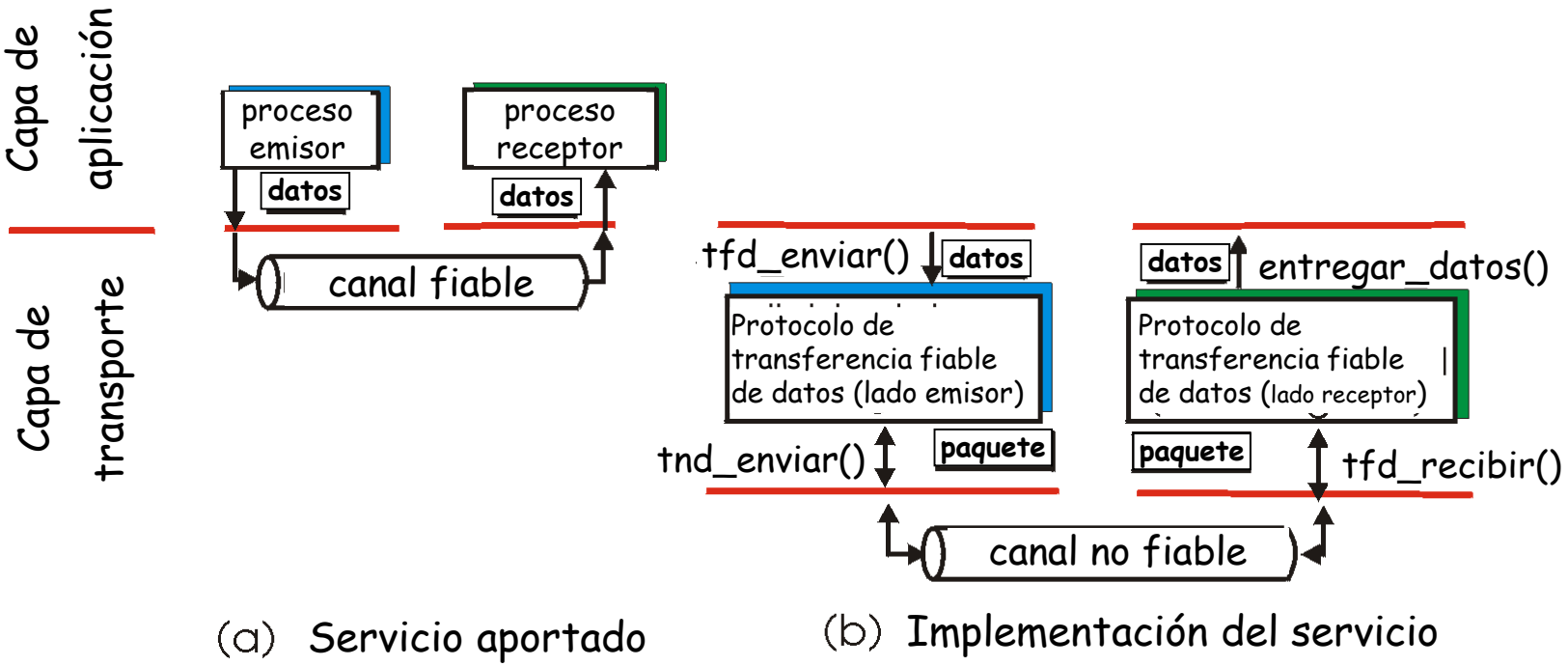
UDP proporciona transferencia poco fiable de grupos de bytes ("datagramas") entre cliente y servidor

Interacción socket cliente/servidor: UDP



Fundamentos de la transferencia fiable de datos

- Importante en capas de aplicación, de transporte y de enlace.
- ¡Lista "top-ten" de las cuestiones más importantes sobre redes de computadores !



(a) Servicio aportado

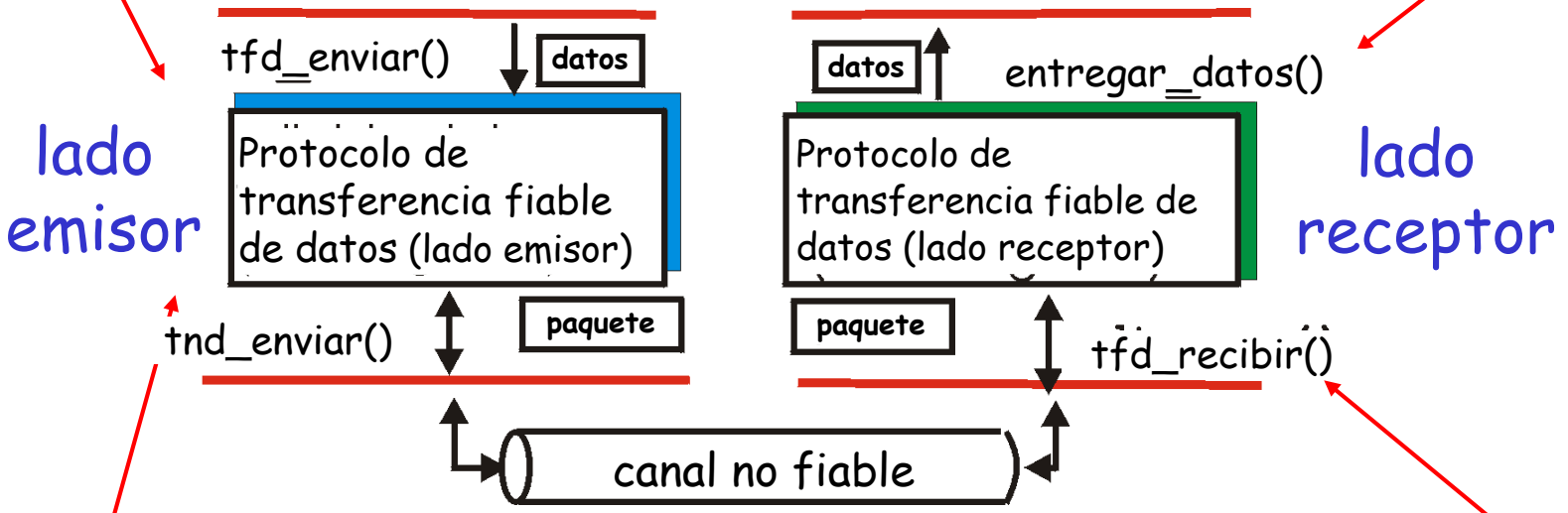
(b) Implementación del servicio

- Las características del canal poco fiable determinarán la complejidad del protocolo de transferencia fiable de datos (tfd).

Transferencia fiable de datos : introducción

Tfd_enviar(): llamada desde arriba, (por ejemplo, desde la aplicación). Pasa a los datos a enviar del receptor de la capa inferior.

entregar_datos(): llamada de la **tfd** para entregar los datos a la capa superior.



tnd_enviar(): llamada de la tfd para transferir el paquete por el canal no fiable al receptor.

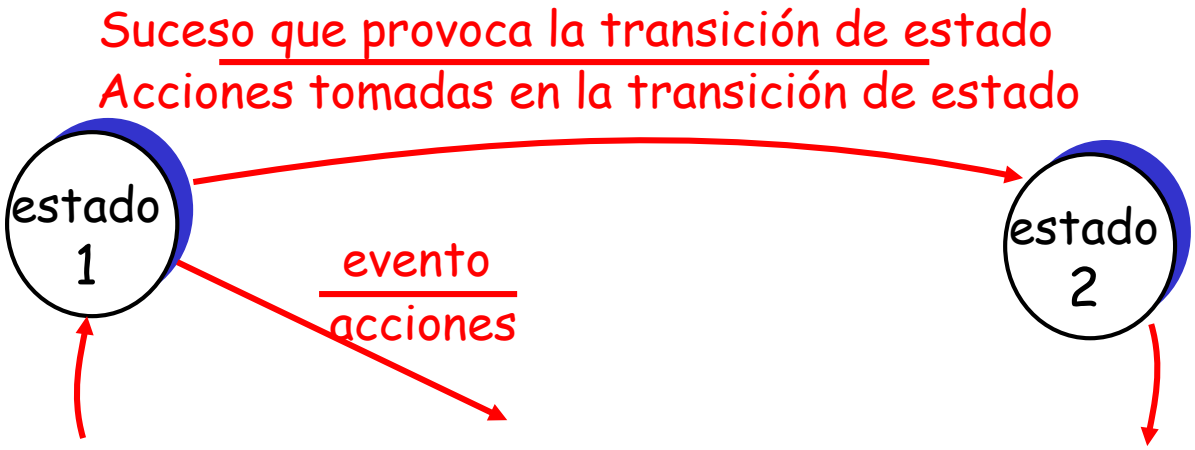
tfd_recibir(): llamada cuando el paquete llega al lado receptor del canal.

Transferencia fiable de datos :introducción

Lo que haremos, será:

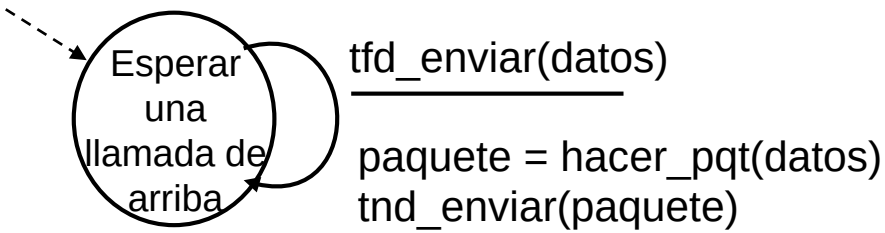
- Incrementar el desarrollo de los lados emisor y receptor del protocolo fiable de transferencia de datos (pft).
- Considerar solamente la transferencia de datos unidireccional.
  - ¡Pero la información de control fluirá en ambas direcciones!
- Uso de máquinas de estados finitos (FSM, Finite State Machine) para especificar el emisor y el receptor.

Estado: encontrándose en un "estado", el siguiente estado sólo lo determina el siguiente suceso.

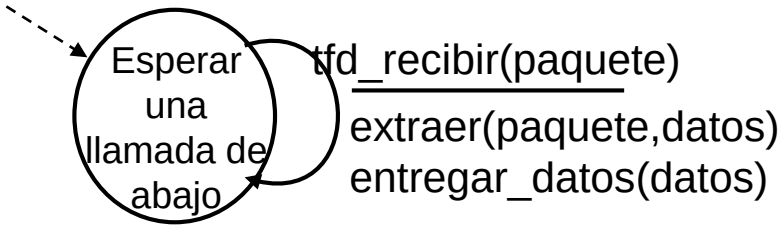


Tnd1.0: transferencia fiable por un canal fiable

- Canal subyacente perfectamente fiable:
  - Sin errores en los bits.
  - Sin pérdida de los paquetes.
- FSMs separadas para el emisor y el receptor:
  - El emisor envía los datos al canal subyacente.
  - El receptor lee los datos del canal subyacente.



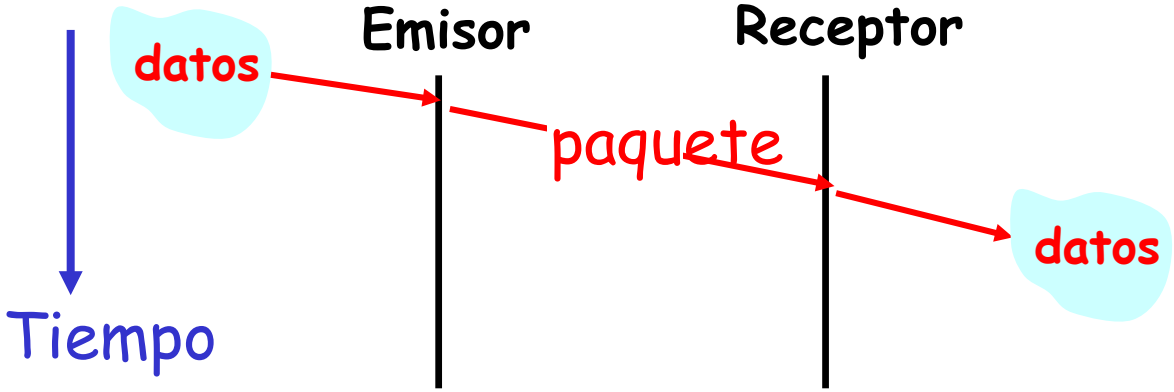
Emisor



Receptor

Tnd1.0: transferencia fiable por un canal fiable

- Canal Ideal: sin errores, no necesita confirmaciones (SIMPLEX)



Tnd2.0: canal con errores en bits

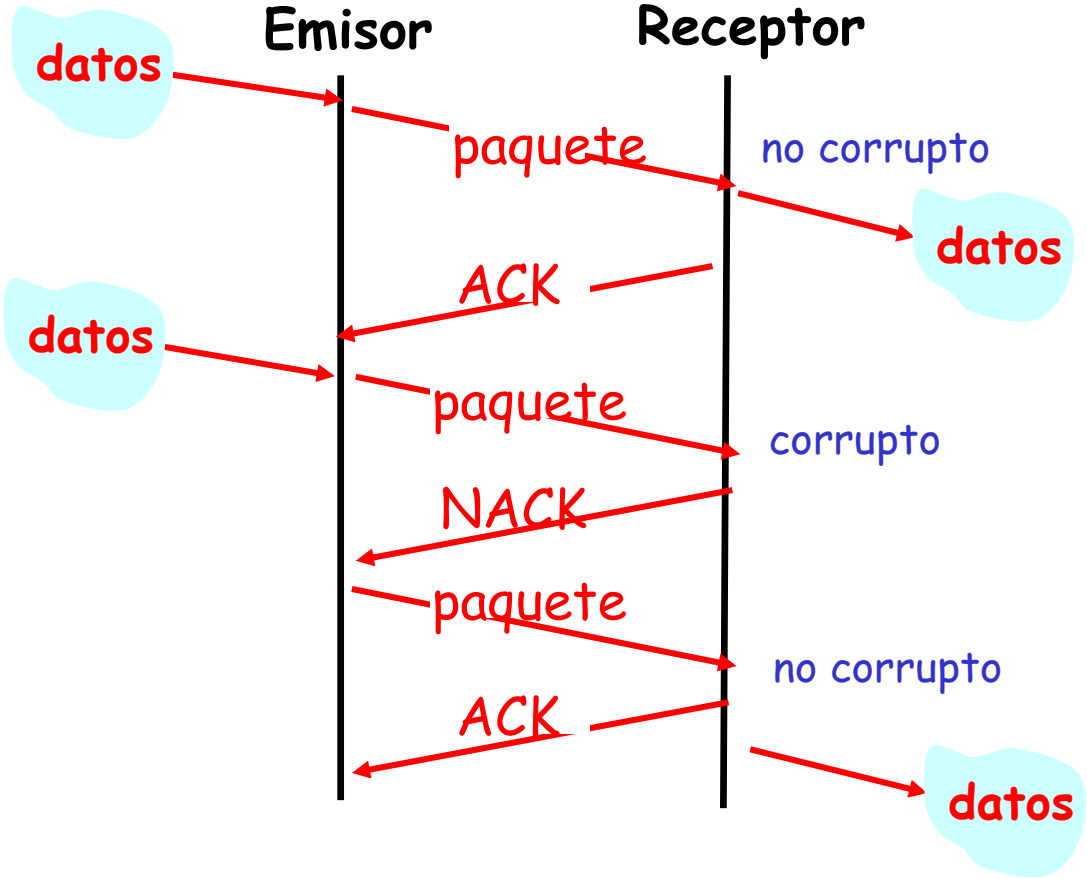
- Protocolos ARQ (Automatic Repeat reQuest):
  necesita confirmaciones postitivas y/o negativas (HALF DUPLEX)

Características:
- Detección de errores
- Realimentación del receptor
- retransmisión

Tnd2.0: canal con errores en bits

- Puede que el canal subyacente altere los bits del paquete.
  - Recuerde la suma de comprobación UDP para detectar errores en bits.
- *La pregunta es:* ¿Cómo recuperarse de los errores?
  - *Reconocimientos positivos (ACKs):* El receptor le dice explícitamente al emisor que el paquete ha sido bien recibido (OK).
  - *Reconocimientos negativos (NACKs):* El receptor le dice explícitamente al emisor que el paquete contiene errores.
  - El emisor retransmite el paquete cuando recibe un NACK.
  - ¿Ejemplos humanos utilizando ACKs y NACKs?
- Nuevos mecanismos en **`tnd2.0`** (más allá del **`tnd1.0`**):
  - Detección de errores.
  - Realimentación del receptor: mensajes de control (ACK,NACK) receptor -> emisor.

Tnd2.0: canal con errores en bits

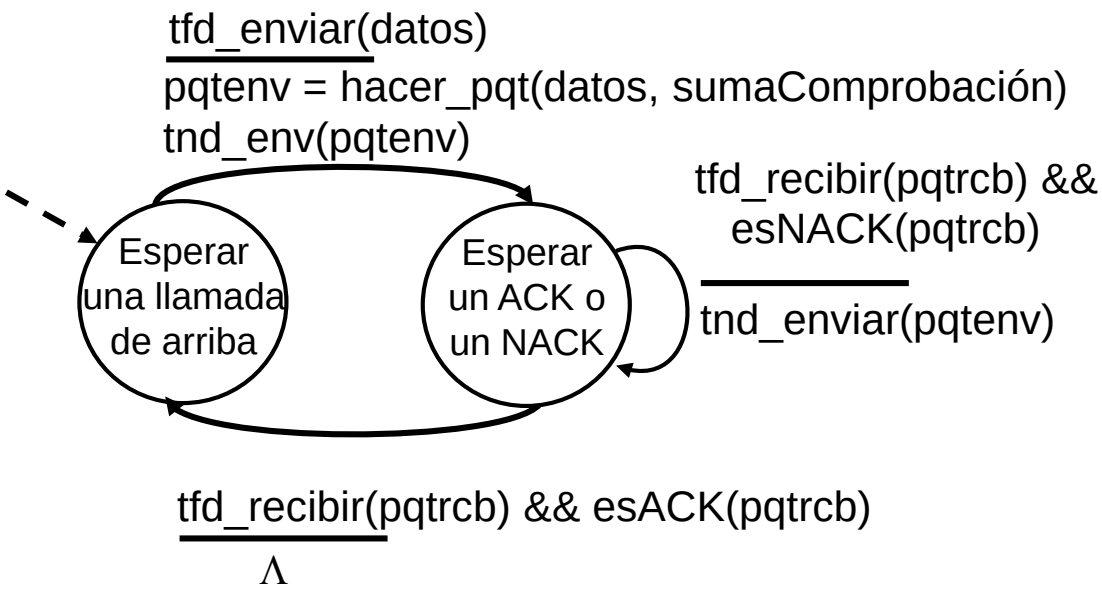


El Emisor no enviará más datos hasta que este seguro de que el Receptor ha recibido el paquete correctamente

¿Qué pasa si los paquetes de reconocimiento tienen errores?

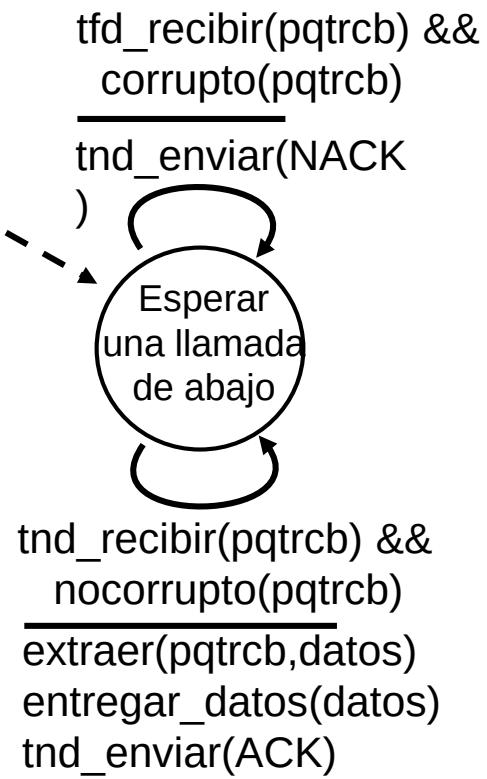
El Emisor no tiene forma de saber si el Receptor ha recibido o no correctamente el último fragmento de datos transmitidos.

Tnd2.0: especificaciones de FSM

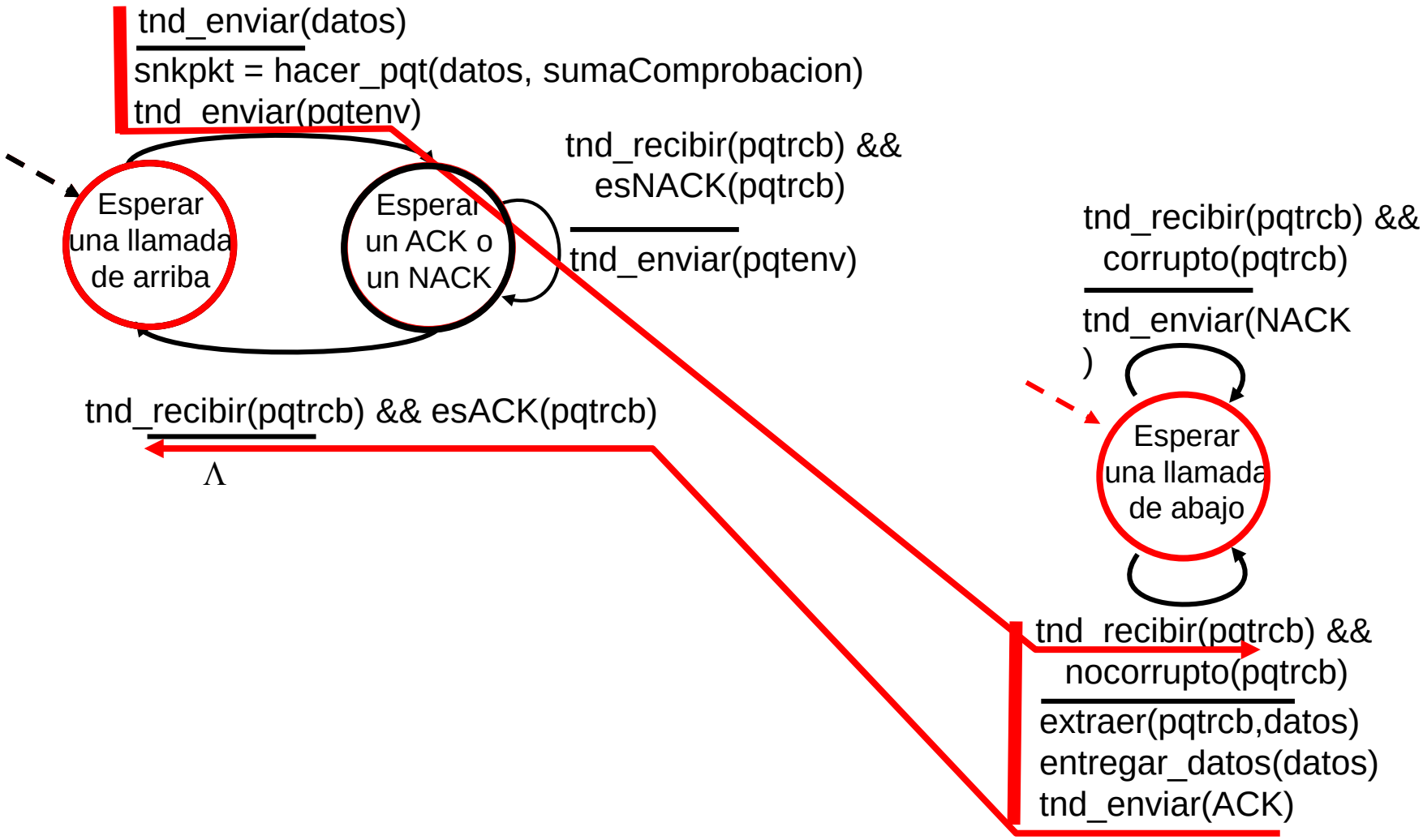


emisor

receptor



Tnd2.0: operación sin ningún error



¡Tnd2.0 tiene un defecto!

¿Qué pasa si ACK/NACK están corruptos?

- ¡El emisor no sabe qué le ha pasado al receptor!
- Simplemente, el emisor no puede retransmitir un posible duplicado.

¿Qué hacer?

- ¿Los ACKs/NACKs emisores son el ACK/NACK del receptor? ¿Qué pasa si el ACK/NACK emisor se pierde?
- El emisor retransmite, ¡pero esto puede causar una retransmisión de un paquete ya recibido!

Gestión de duplicados:

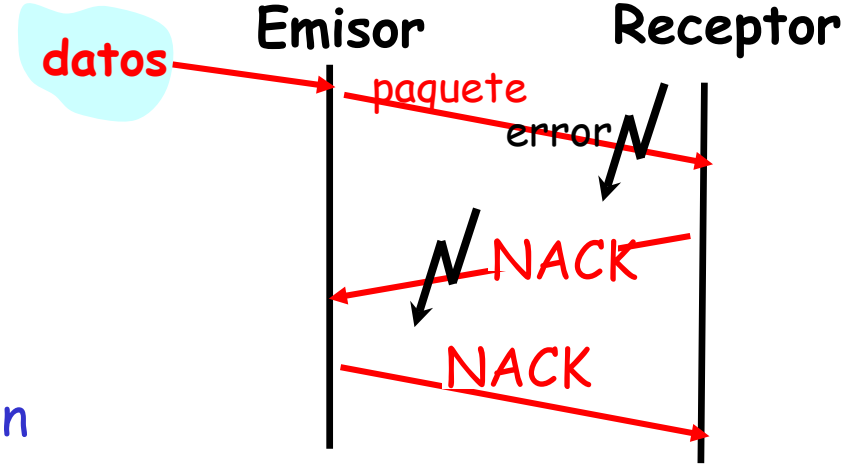
- El emisor añade un *número de secuencia* a cada paquete.
- El emisor retransmite el paquete en curso si ACK/NACK es falso.
- El receptor se desembaraza del paquete duplicado.

Parada y espera

El emisor envía un paquete y espera la respuesta del receptor .

Posibles soluciones:

1. No se puede continuar la comunicación (abrazo mortal de S.O.)
2. Utilizar códigos correctores de errores, no solo detectores (añaden bastantes más bits, TINF)
3. Reenviar el paquete de datos, siempre que se recibe un reconocimiento (ACK/NACK) con errores

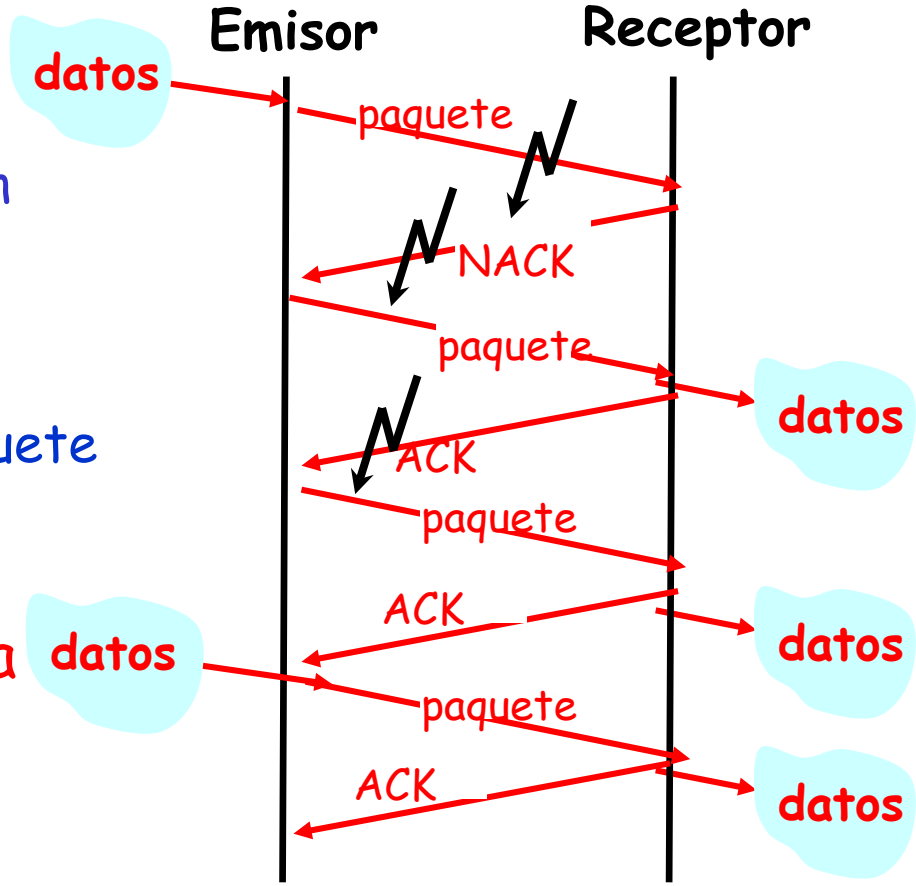


Problema: Paquetes duplicados

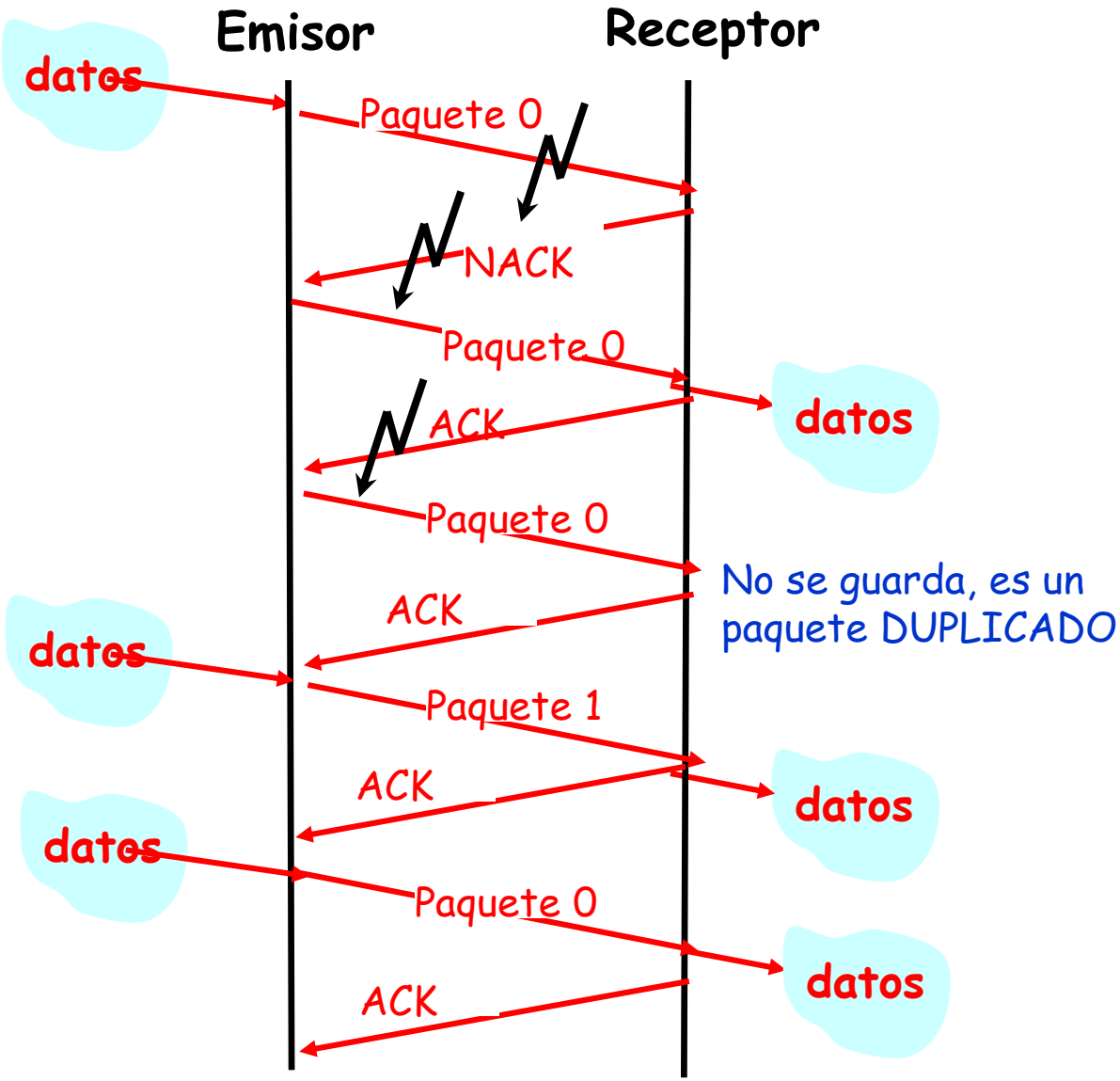
No sabemos cuando se recibe un paquete nuevo o una retransmisión

Solución: Números de secuencia

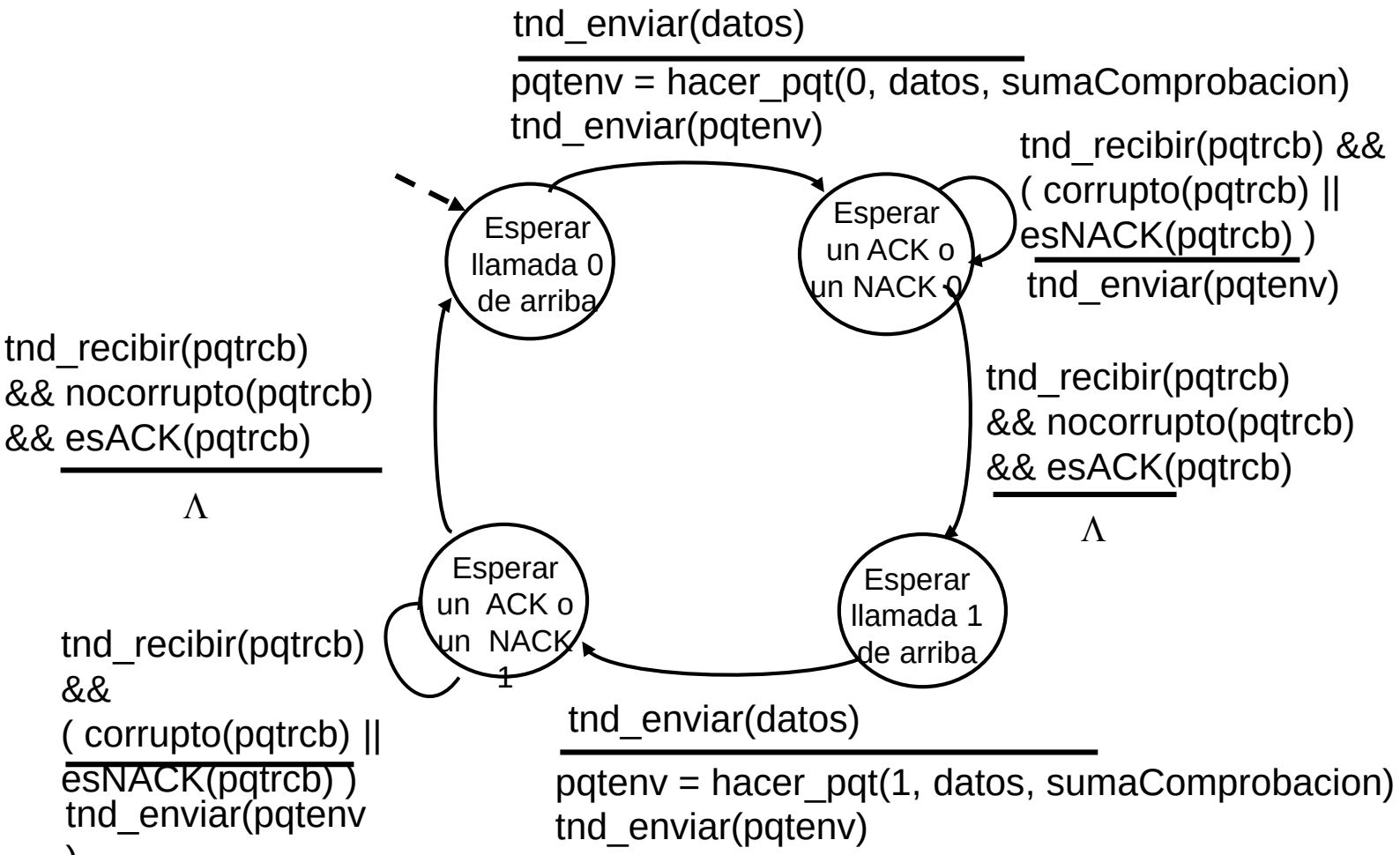
Modulo 2



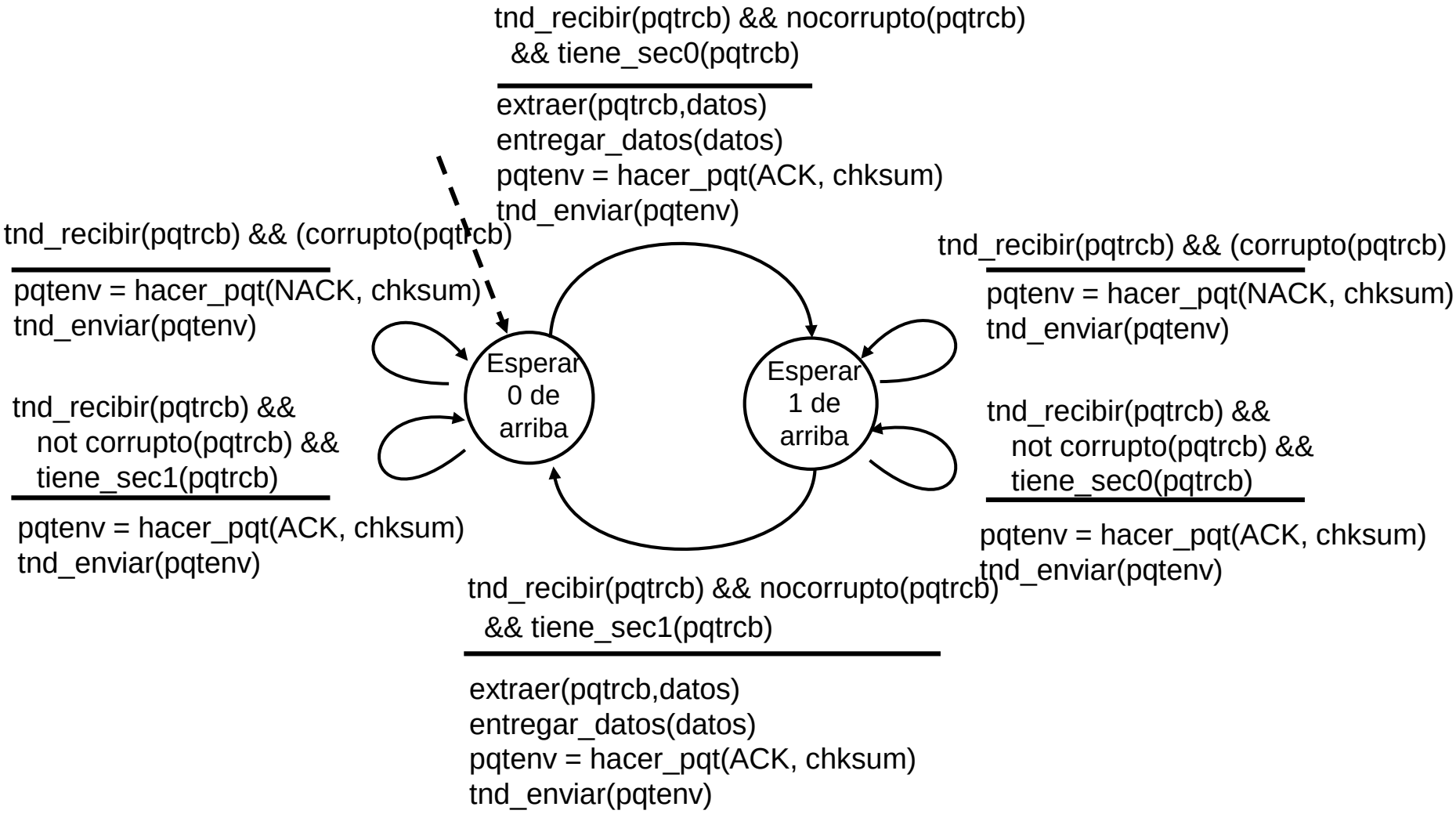
Tnd2.1: receptor, gestión de falsos ACK/NACK's



Tnd2.1: emisor, gestión de falsos ACK/NACK's



Tnd2.1: receptor, gestión de falsos ACK/NACKs



Tnd2.1: debate

Emisor:

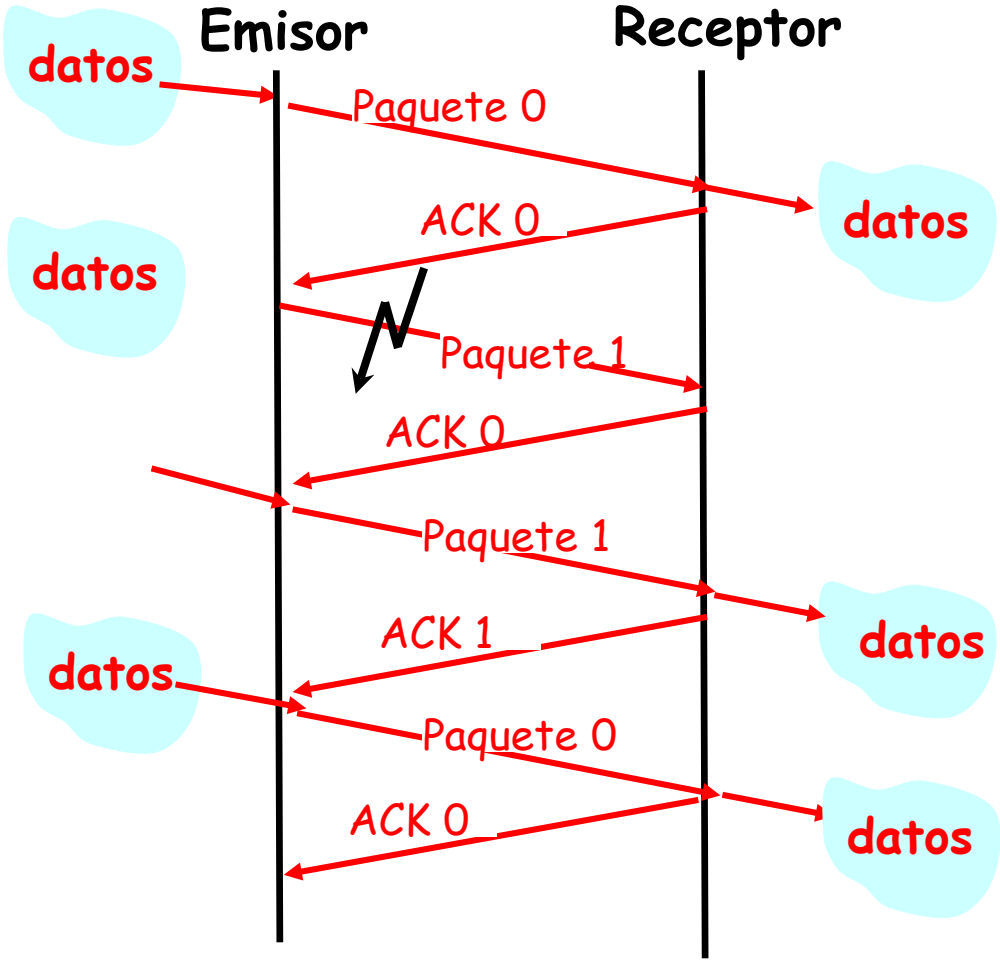
- Secuencia de números añadida al paquete.
- Dos números para la secuencia (0,1) bastarán. ¿Por qué?
- El emisor debe comprobar si los ACK/NACK están corruptos.
- El doble de los estados que haya:
  - El estado debe "recordar" si el paquete "en curso" tiene un número de secuencia 0 o 1.

Receptor:

- Debe comprobar si el paquete recibido es un duplicado:
  - El estado indica si se espera un 0 o un 1 en el número de secuencia del paquete.
- Nota: El receptor *no* puede saber si el emisor recibió correctamente su último ACK/NACK.

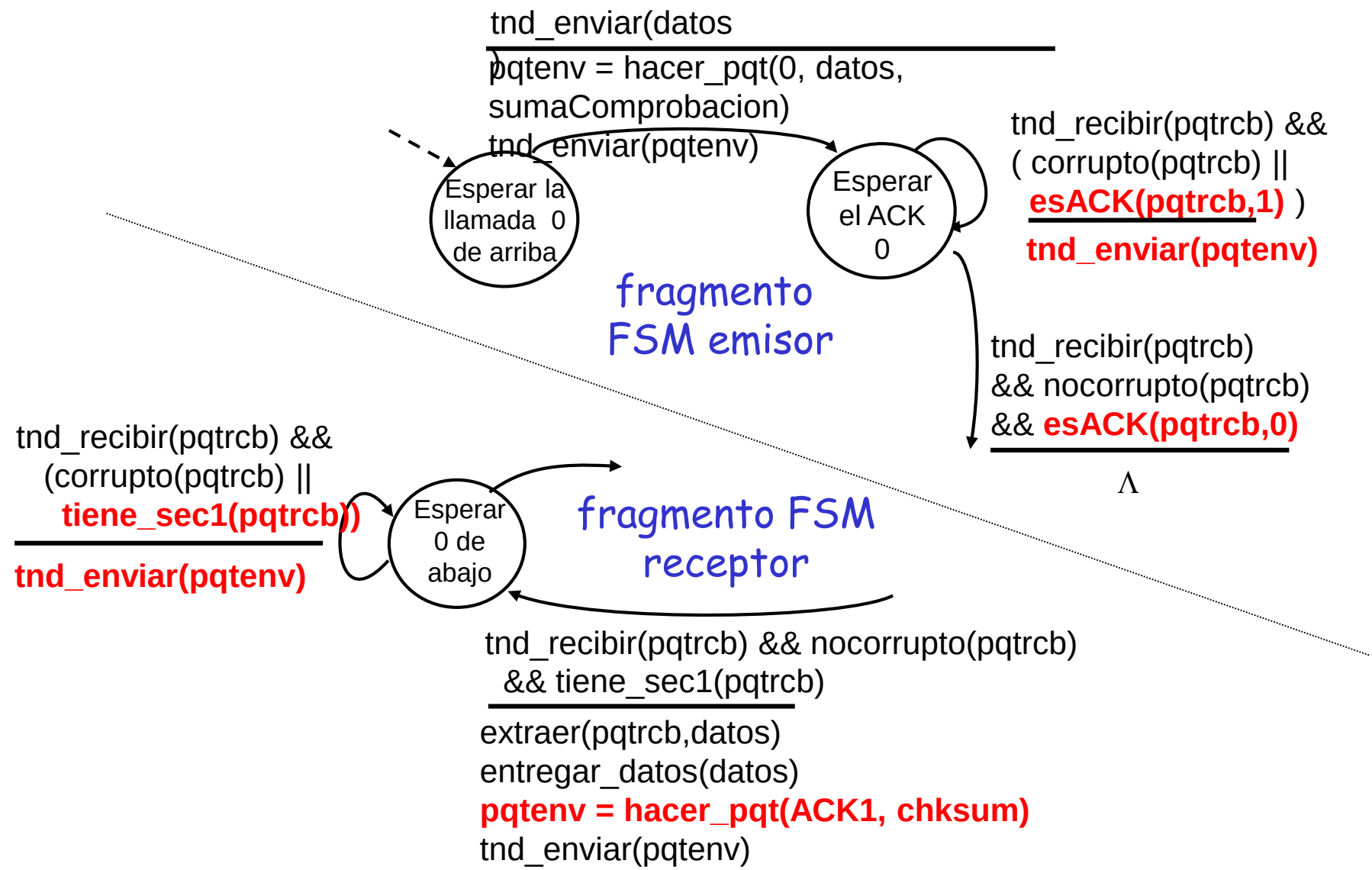
Tnd2.2: un protocolo sin NACK's

- En lugar de NACK, el receptor envía ACK por cada paquete recibido correctamente:
  - El receptor debe incluir *explícitamente* el número de secuencia del paquete al que se refiere el ACK.
- Un duplicado de ACK recibido por el emisor tiene el mismo efecto que un NACK: *retransmitir el paquete en curso.*



Un Emisor que recibe dos ACK's para el mismo paquete (ACK duplicado) es consciente de que el Receptor no recibió correctamente el paquete que sigue al reconocido por el duplicado

Tnd2.2: fragmentos del emisor y el receptor



Tnd3.0: canales con errores y pérdidas

Nuevo supuesto: El canal subyacente puede perder también los paquetes (datos o ACKs).

- sumaComprobacion, número de secuencia , ACKs, las retransmisiones serán de ayuda, pero no bastarán.

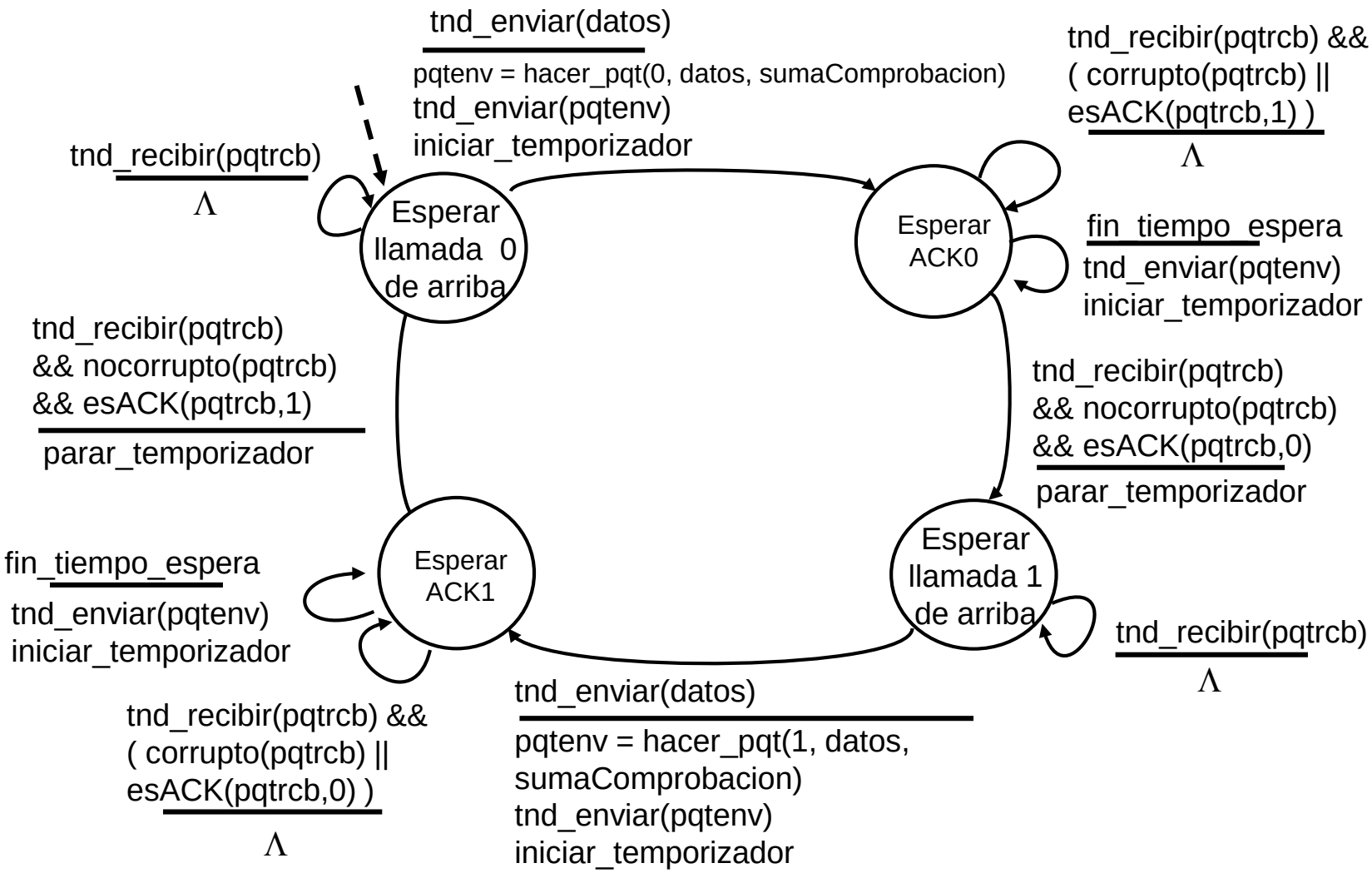
Pregunta: ¿Qué hacer cuando hay una pérdida?

- El emisor espera hasta que cierta cantidad de datos o ACK's se pierden, y entonces retransmite.

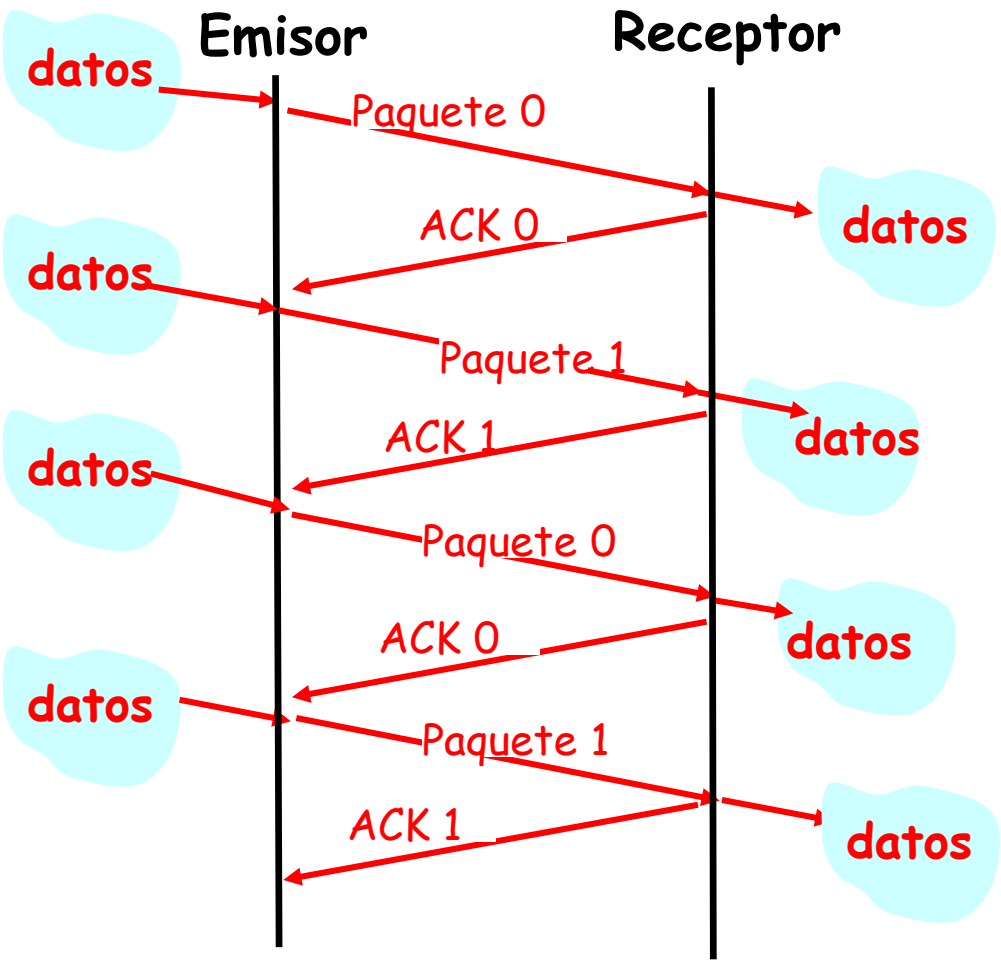
Aproximación: el emisor espera un tiempo "razonable" el ACK.

- El emisor retransmite si no recibe ACK en este tiempo.
- Si el paquete (o ACK) sólo se ha retardado (y no perdido):
  - La retransmisión se habrá hecho por duplicado, pero el uso del número de secuencia resuelve de antemano esto.
  - El receptor debe especificar el número de secuencia del paquete al que se refiere el ACK.
- Requiere un temporizador de cuenta atrás.

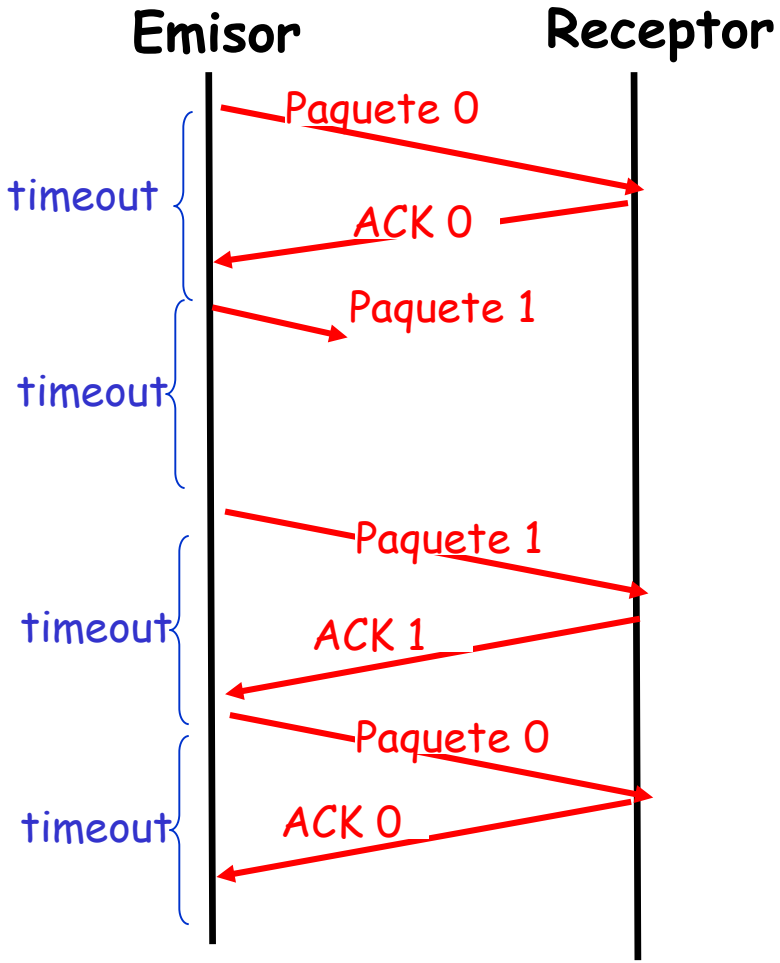
Tnd3.0 emisor



Tnd3.0 en acción

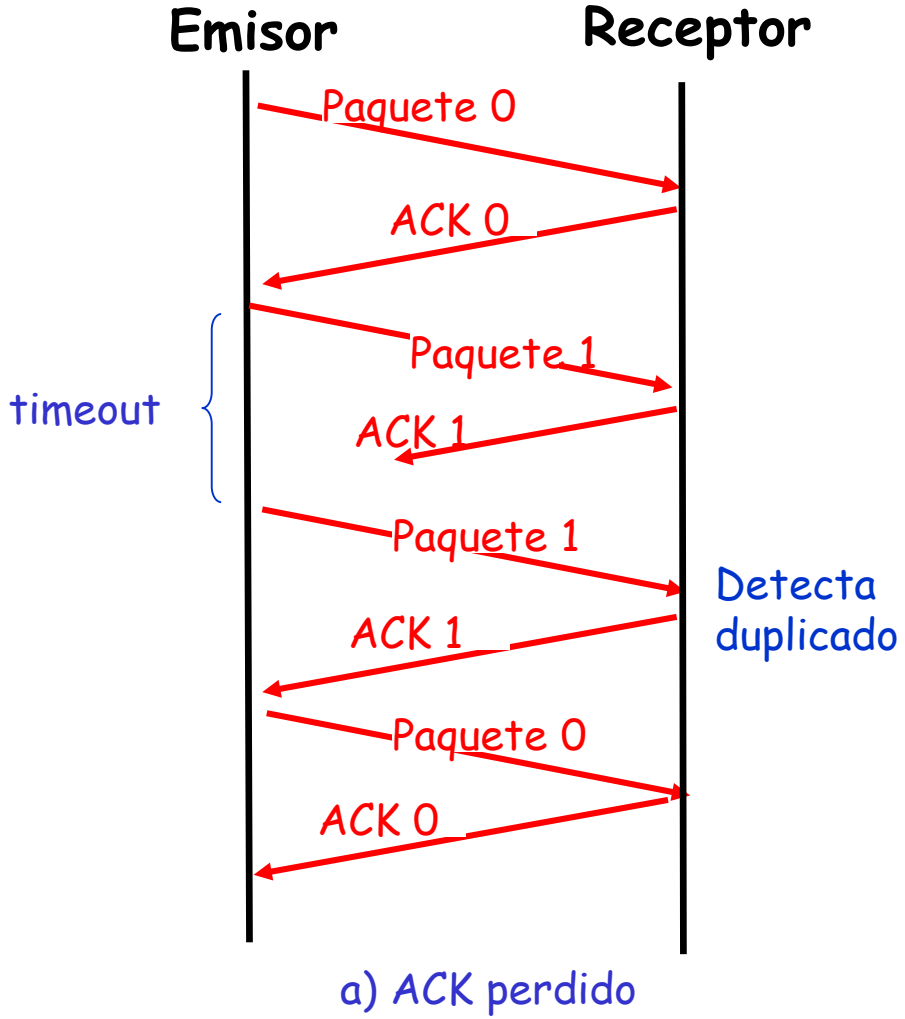


a) Funcionamiento sin pérdidas

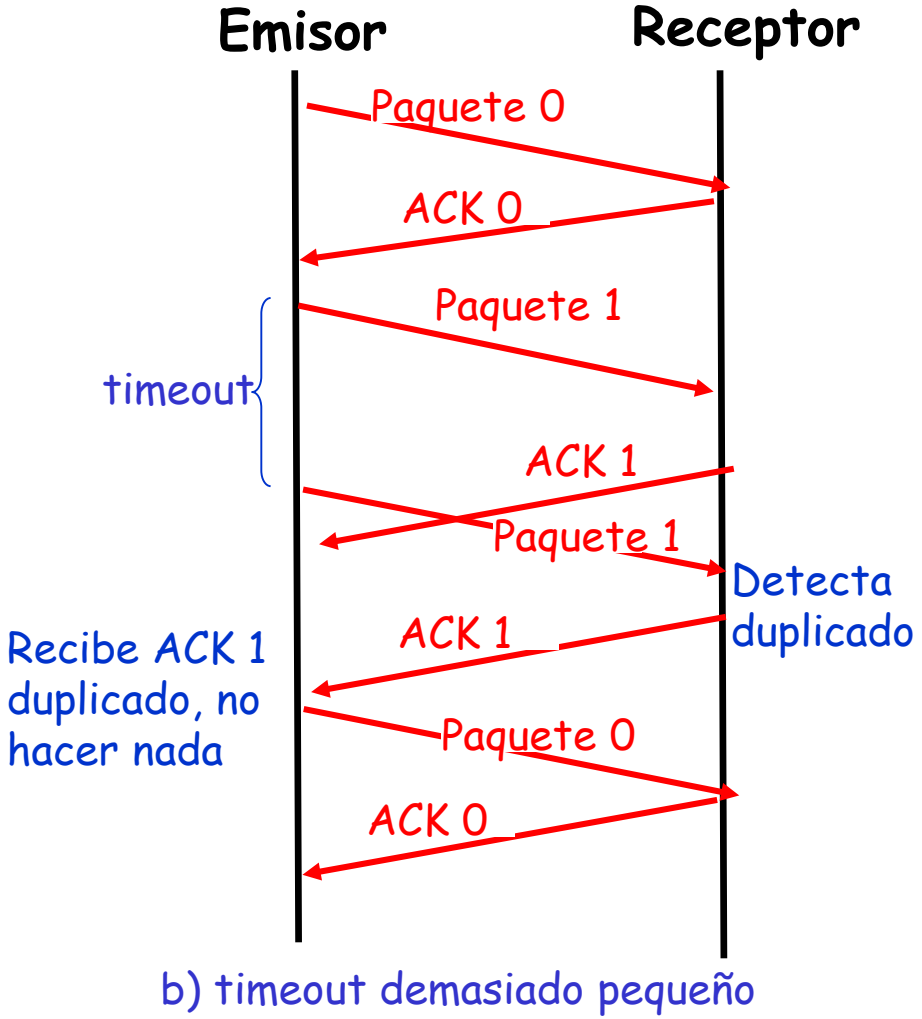


b) Paquete pérdido

Tnd3.0 en acción



a) ACK perdido

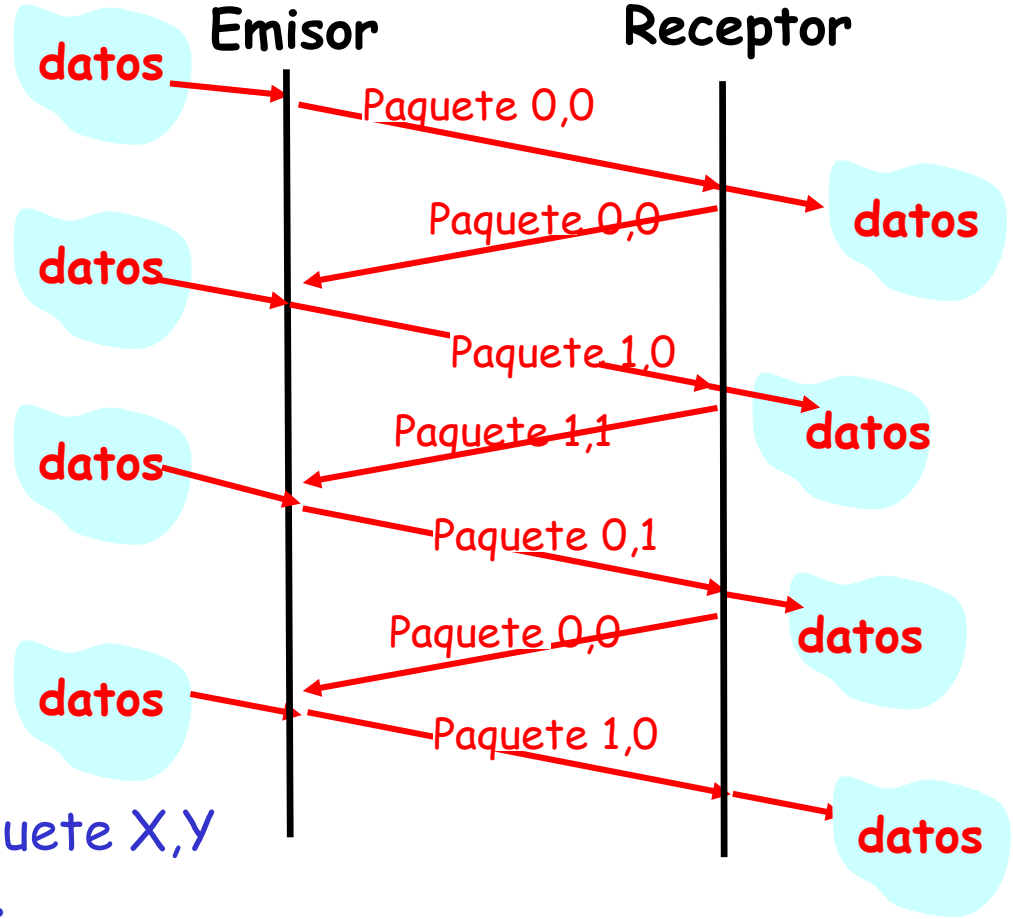


b) timeout demasiado pequeño

Este protocolo se denomina STOP & WAIT Bit Alternante

Concepto de piggybacking:

Consiste en la utilización de paquetes de datos para realizar reconocimientos, siempre y cuando el receptor tenga datos a enviar.



Formato: Paquete X,Y

X: nº de paquete

Y: nº de paquete reconocido

Presentación de tnd3.0

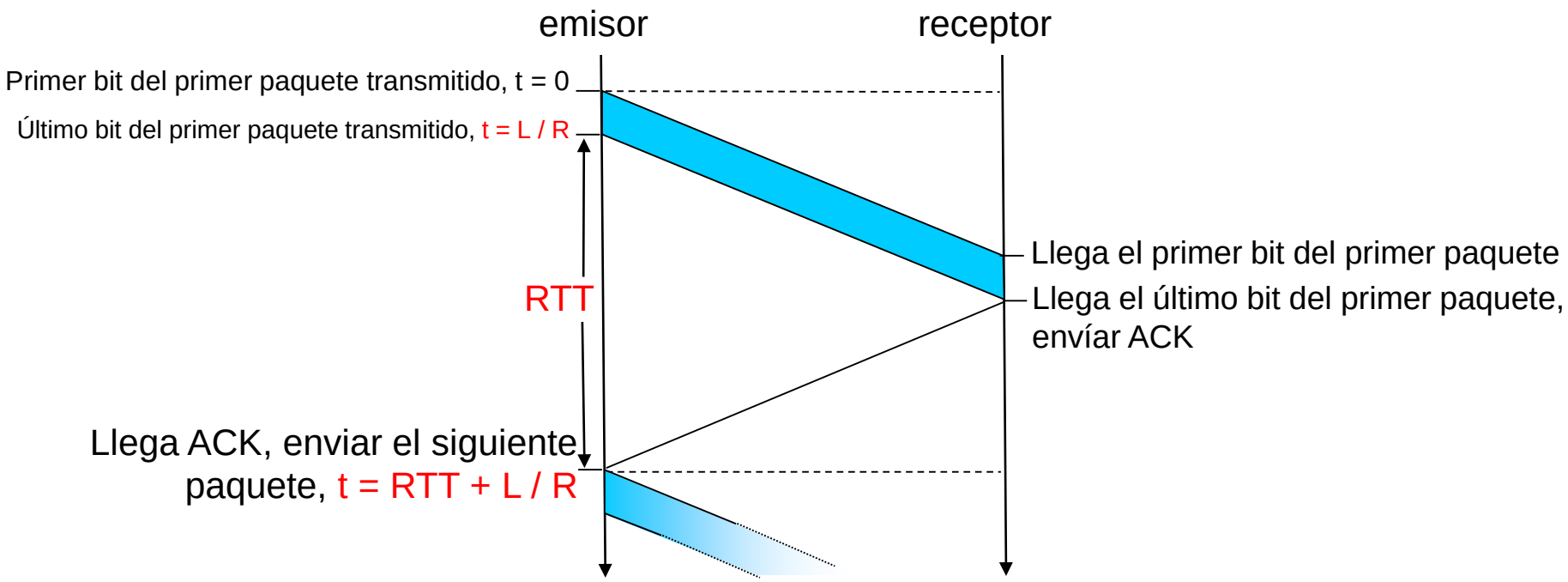
- Tnd3.0 funciona, pero la presentación es pésima.
- Ejemplo: enlace 1 Gbps, 15 ms de retardo de terminal a terminal e-e prop. delay, paquete 1KB:

$$T_{transmitir} = \frac{L \text{ (longitud paquete en bits)}}{R \text{ (tasa de transmisión, bps)}} = \frac{8kb/pqt}{10^9 \text{ b/seg}} = 8 \text{ microseg}$$

$$U_{emisor} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

- U_{emisor}: utilización – fracción de tiempo en la que el emisor está enviando.
- 1KB paquete cada 30mseg -> 33kB/seg producción por un enlace de 1 Gbps.
- ¡El protocolo de red limita el uso de recursos físicos!

Tnd3.0: operación de parada y espera

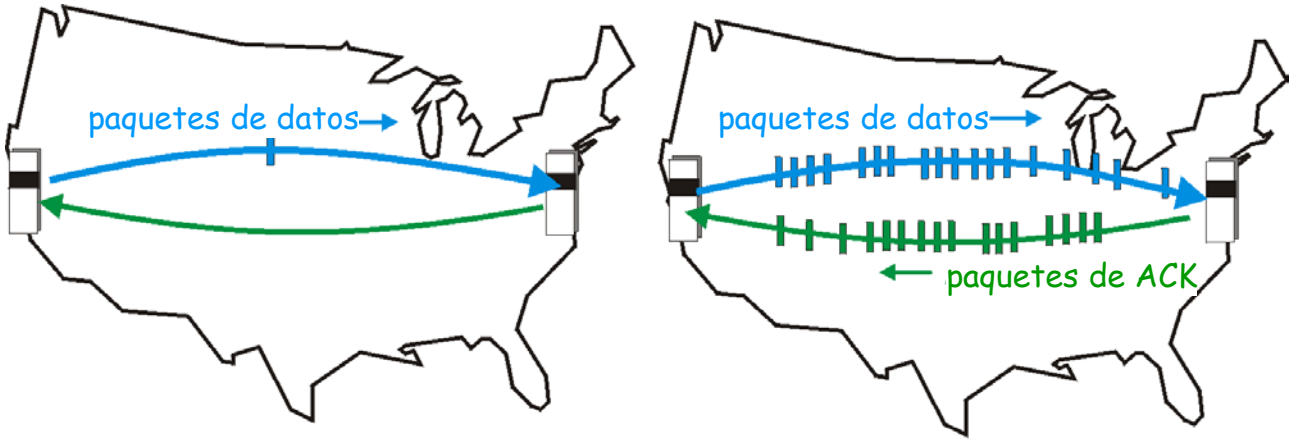


$$U_{emisor} = \frac{L / R}{RTT + L / R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

Protocolos de transmisión continua

Entubamiento: el emisor puede envíar varios paquetes, "en vuelo", en proceso de ser reconocidos.

- El rango de números de secuencia debe incrementarse.
- Almacenamiento del búfer del emisor y/o del receptor.

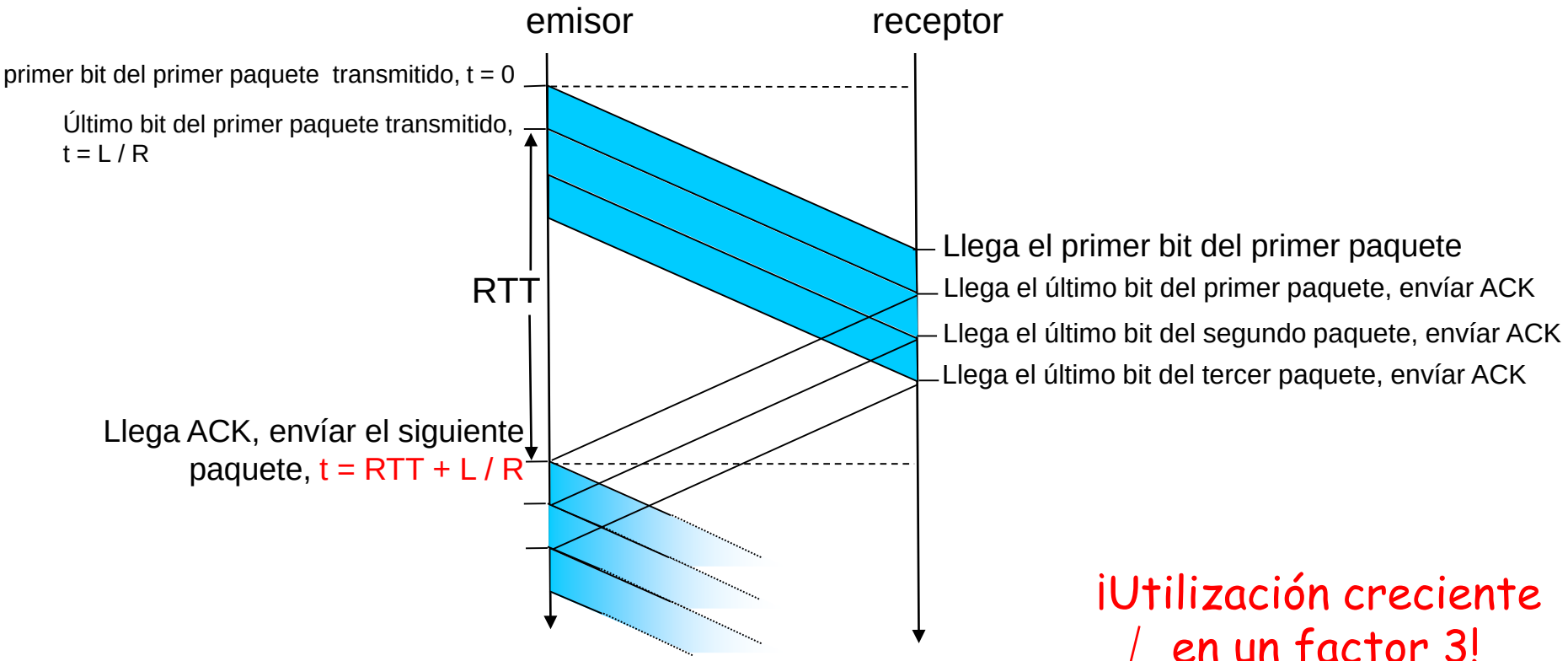


(a) Un protocolo de parada y espera en operación. (b) Un protocolo entubado en operación.

- Dos formas genéricas del protocolos :

Retroceder N (GBN), Repetición selectiva (SR).

Transmisión Continua: utilización creciente



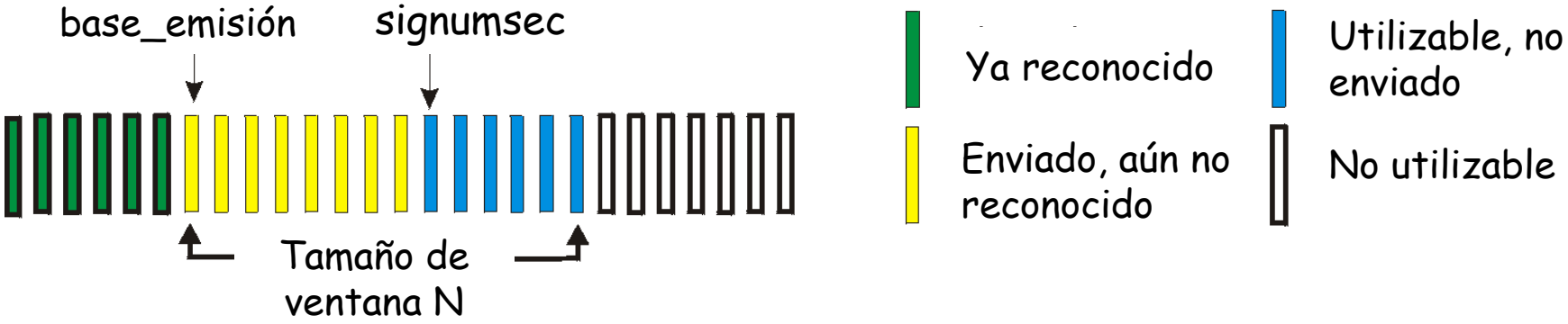
¡Utilización creciente en un factor 3!

$$U_{emisor} = \frac{3 * L / R}{RTT + L / R} = \frac{0{,}024}{30{,}008} = 0{,}0008$$

Retroceder N (GBN)

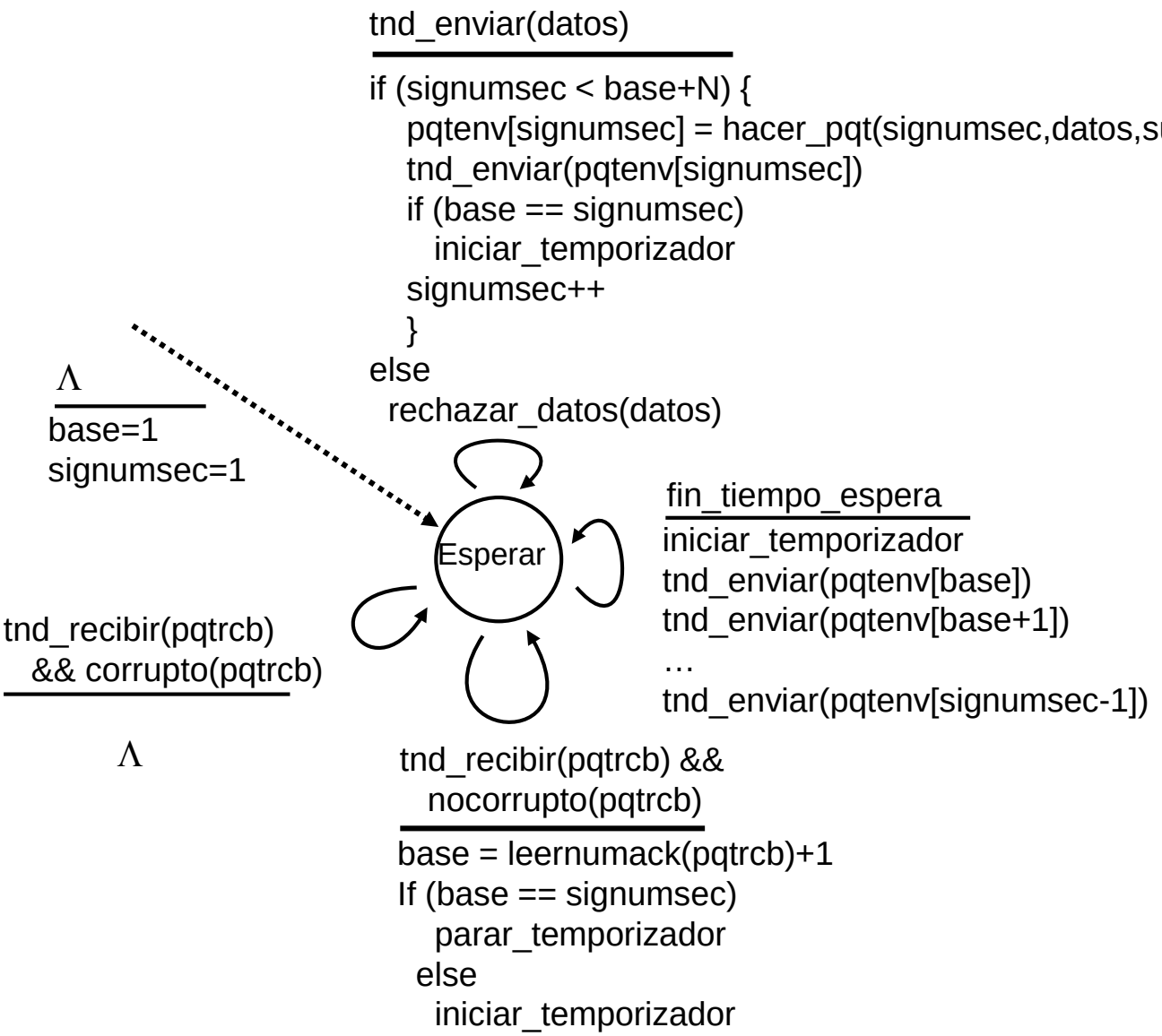
Emisor:

- Número de secuencia k-bit en la cabecera del paquete.
- "Ventana" de hasta N, se permiten paquetes consecutivos no reconocidos.

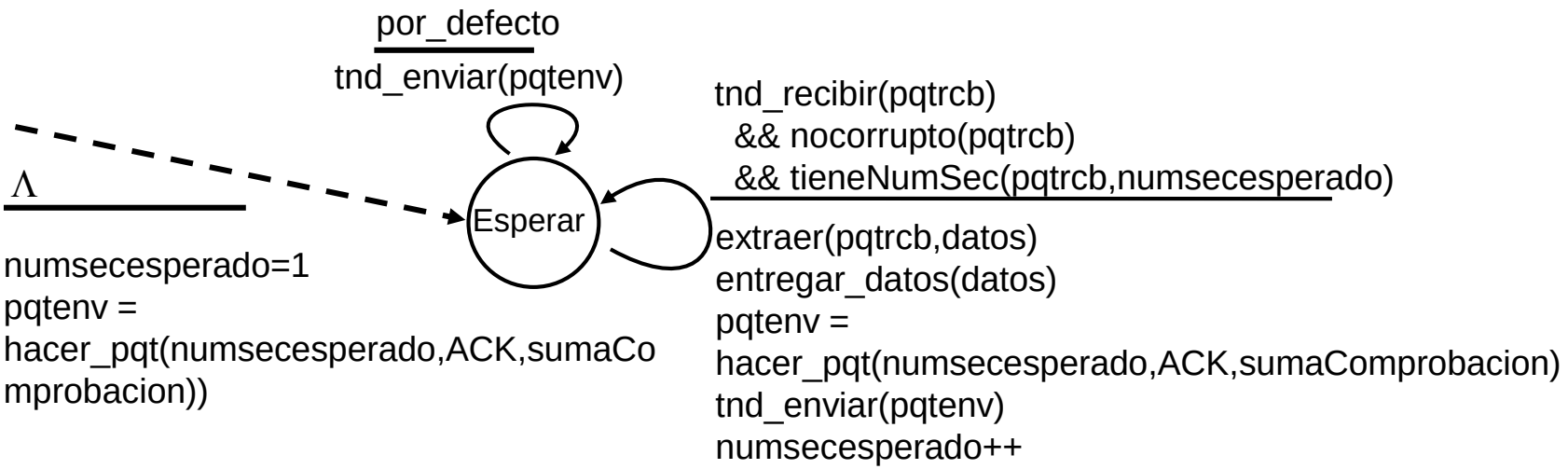


- ACK(n): reconoce todos los paquetes que incluyan el número de secuencia hasta n - "ACK acumulativo".
  - Puede que engañe a los ACKs duplicados (ver receptor).
- Temporizador por cada paquete en vuelo.
- *Fin del tiempo de espera(n):* retransmite el paquete n y todos los paquetes de números de secuencia mayores por ventana.

GBN: emisor del FSM extendido



GBN: receptor del FSM extendido

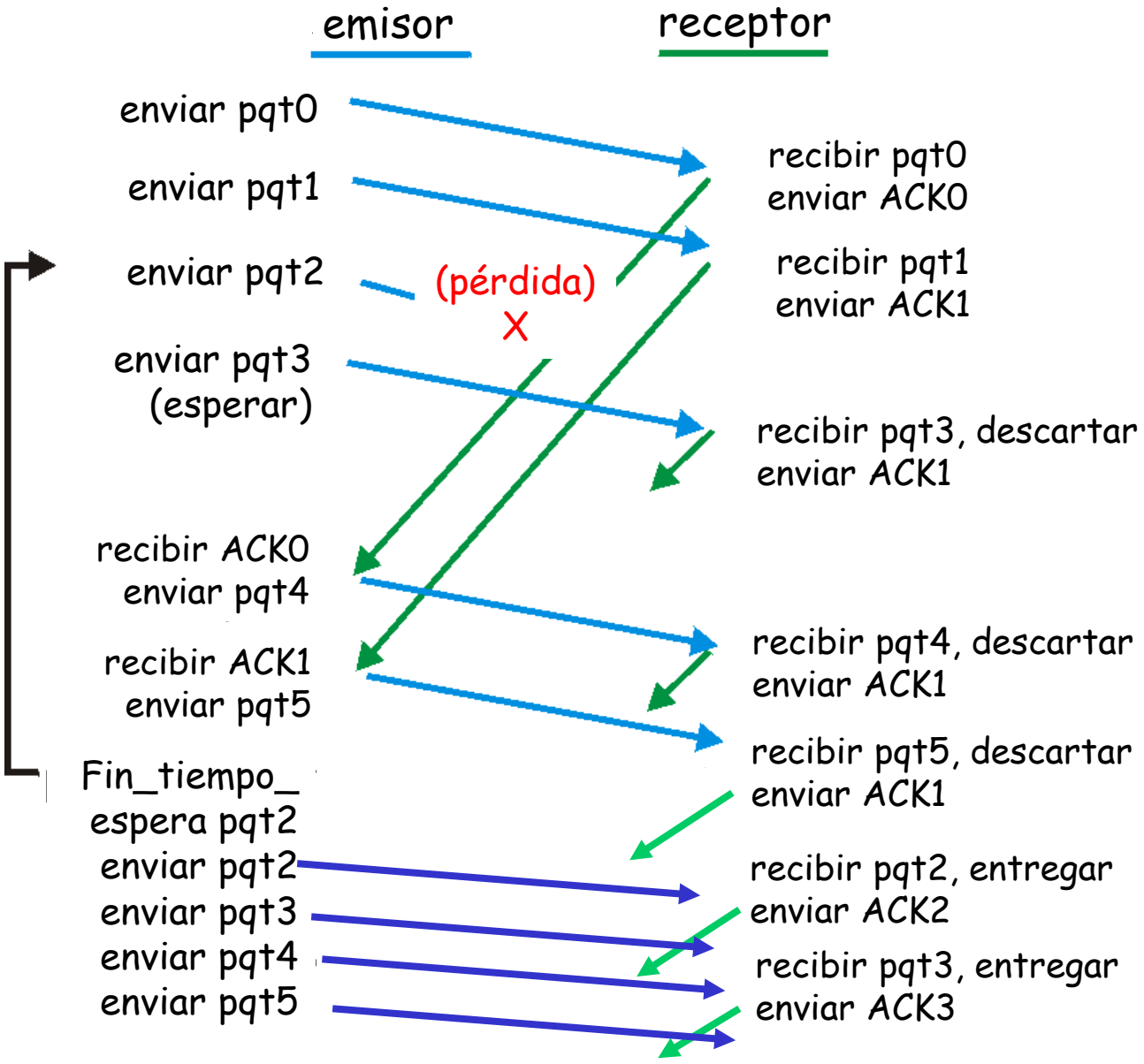


Sólo ACK: siempre envía ACK por cada paquete recibido correctamente con el mayor número de secuencia *en orden*.

- Puede que genere ACKs duplicados.
- Solamente necesita recordar **`numsecesperado`.**

- Paquete sin orden:
  - Descartar (sin almacenar en el búfer) -> ¡sin almacenamiento en el búfer del receptor!
  - Paquete de ACK duplicado con el mayor número de secuencia en orden.

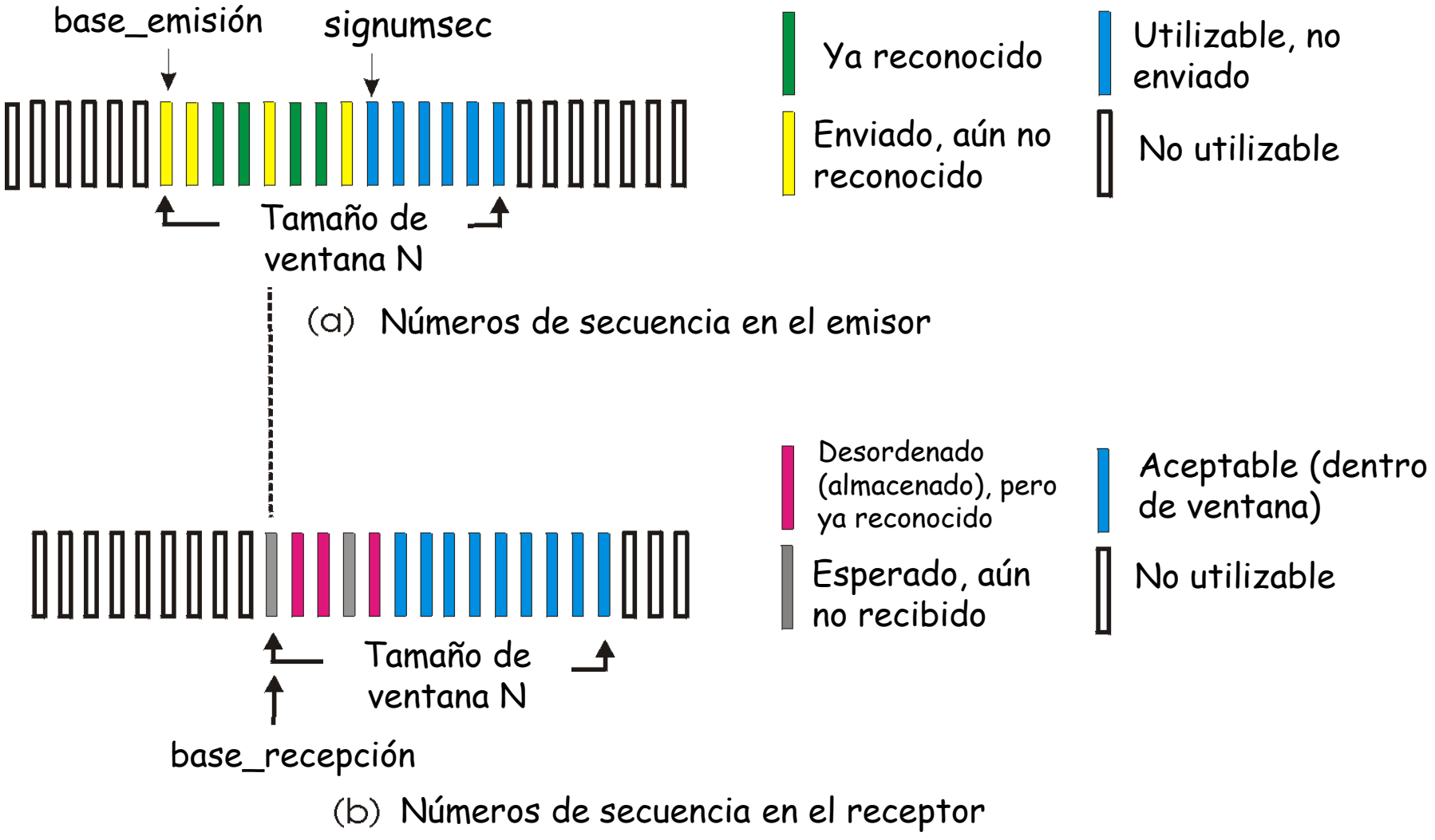
GBN en acción



Repetición selectiva

- El receptor *individualmente* reconoce que todos los paquetes se han recibido correctamente:
  - Paquetes almacenados en el búfer, si es necesario para la entrega final en orden a la capa superior.
- El emisor reenvía solamente los paquetes para los cuales no ha recibido ACK:
  - Temporizador del emisor por cada paquete reconocido.
- Ventana del emisor:
  - N números de secuencia consecutivos.
  - De nuevo se limitan los números de secuencia de los paquetes enviados y reconocidos.

Repetición selectiva: ventanas del emisor y el receptor



Repetición selectiva

emisor

Datos desde arriba :

- Si la siguiente secuencia de números esta disponible en pantalla, enviar paquete.

Fin del tiempo de espera(n):

- Reenvíar paquete n, reiniciar en temporizador.

ACK(n) en [BaseEnvío,BaseEnvío+N]:

- Marca paquete n como recibido.
- Si n es el paquete más pequeño no reconocido, la ventana base avanza hasta la siguiente secuencia de números.

receptor

Paquete n en [rcvbase, rcvbase+N-1]

- Envíar ACK(n).
- Desordenado: almacena en el búfer.
- Ordenado: entrega (y también almacena en el búfer, en paquetes ordenados), la ventana avanza hasta el siguiente paquete no recibido.

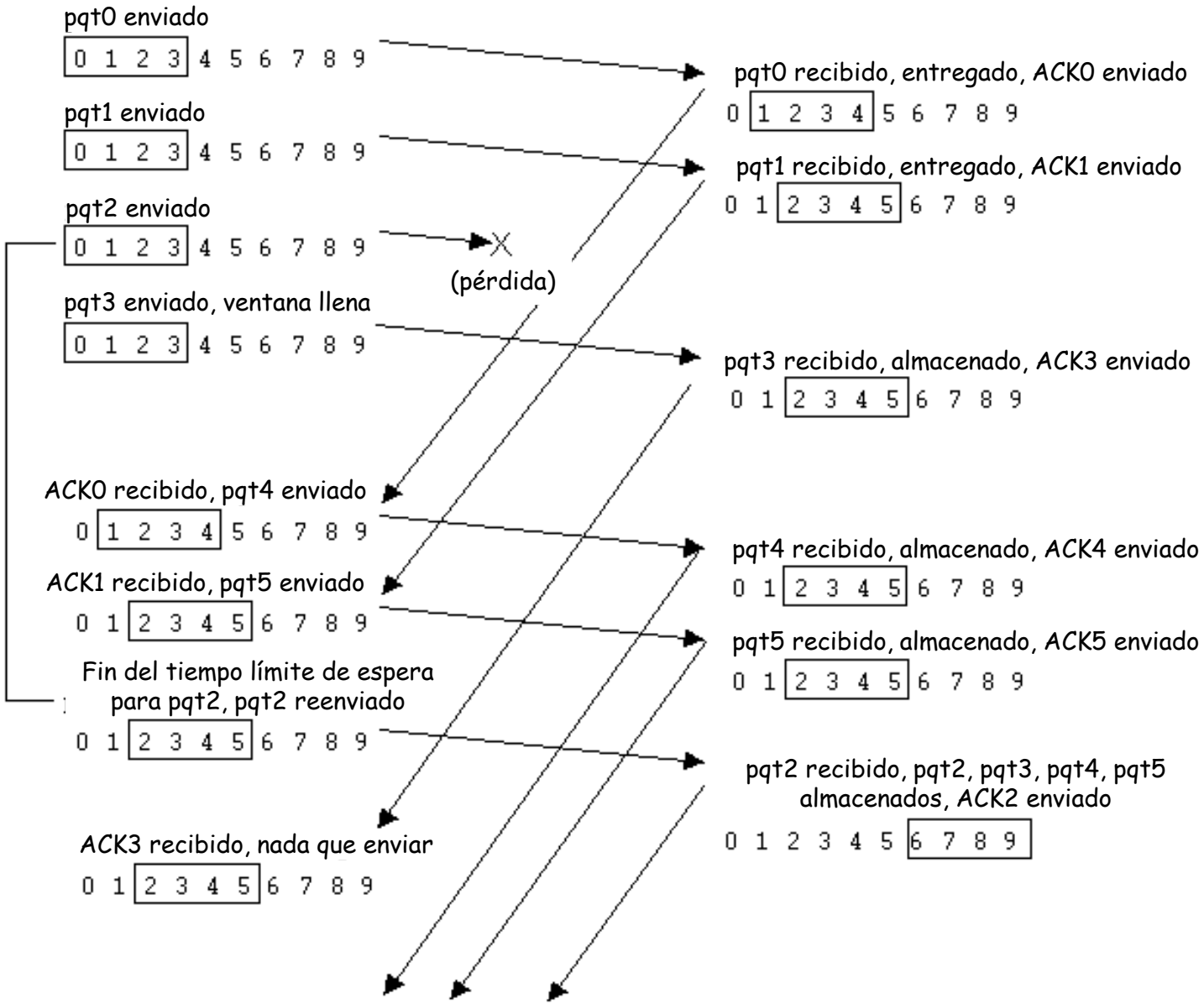
Paquete n en [rcvbase-N,rcvbase-1]

- ACK(n)

en otro caso:

- ignorar.

Repetición selectiva en acción



Repetición selectiva: un dilema

Ejemplo:

- Secuencia de números: 0, 1, 2, 3
- Tamaño de la ventana=3
- ¡El receptor no ve la diferencia entre los dos ejemplos!
- Incorrectamente, los datos duplicados pasan por nuevos en (a).