Tema 2
El nivel de transporte

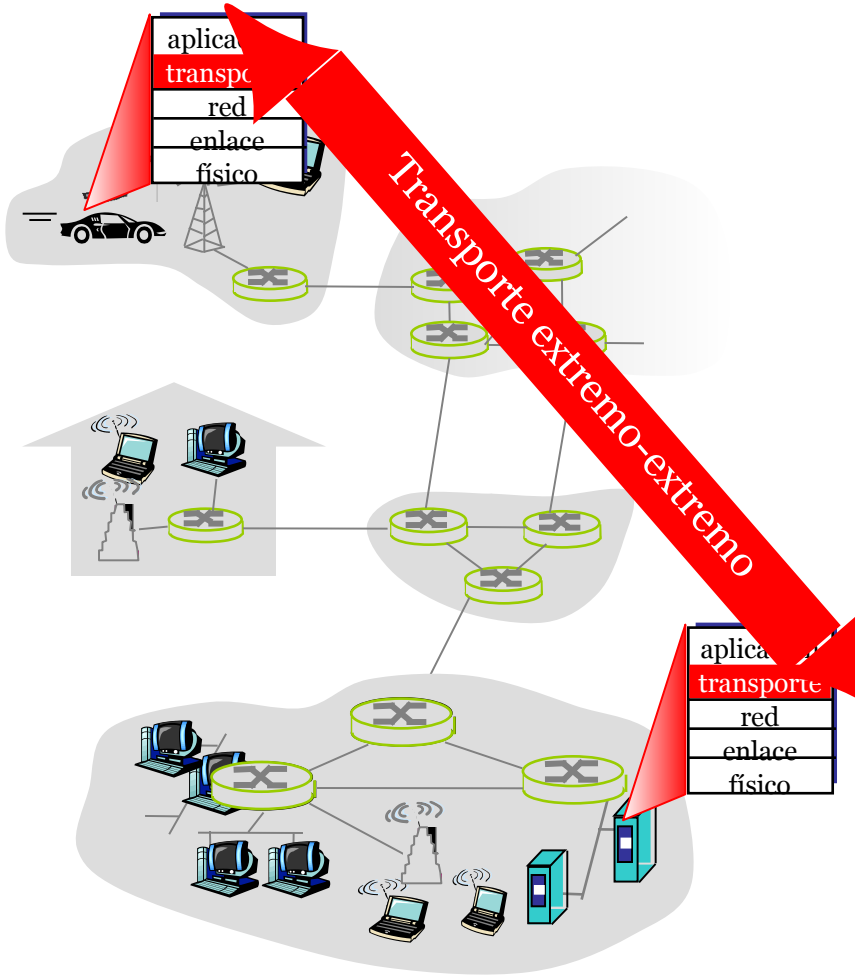
(1904) Redes de Comunicaciones

Contenidos

Contenidos

Objetivo del nivel de transporte

- Proporcionar un servicio de comunicación lógica entre procesos en hosts distintos a través de red
- Este nivel involucra sólo a los hosts
  - Emisor: fragmenta los mensajes de las aplicaciones en segmentos y se los pasa al nivel de red
  - Receptor: reensambla los segmentos en mensajes y se los entrega a las aplicaciones



Funciones del nivel de transporte

- Comunicación de extremo a extremo
  - Servicio de comunicación lógica entre procesos que se ejecutan en hosts distintos
- Multiplexión y demultiplexión
  - Identificar los procesos emisor y receptor sin ambigüedad cuando en el host origen y/o en el host destino hay múltiples procesos haciendo uso de varios sockets
- Detección de errores
  - Identificar y descartar segmentos erróneos
- Segmentación y reensamblaje
  - Fragmentar y reensamblar la información del nivel de aplicación para ajustarse al tamaño máximo de trama del nivel de enlace
- Control de flujo
  - Regular el tráfico para evitar que un emisor rápido inunde a un receptor lento
- Recuperación de errores
  - Resolver situaciones anómalas (segmentos erróneos, perdidos o duplicados)
- Control de la congestión
  - Evitar que se saturen enlaces o equipos de interconexión

Servicios y protocolos de nivel de transporte

- Los protocolos utilizan el servicio de encaminamiento de paquetes entre hosts proporcionado por el nivel de red
  - Entrega no garantizada y desordenada de paquetes (IP)
- Protocolos de transporte disponibles en Internet:
  - TCP
    - Entrega garantizada y en orden de los segmentos
    - Control de flujo y recuperación de errores
    - Control de la congestión
  - UDP
    - Entrega no garantizada y (posiblemente) desordenada de los segmentos
  - Ninguno ofrece garantías de latencia o de ancho banda
- Servicios proporcionados al nivel de aplicación:
  - Orientado a conexión, confiable (TCP)
  - Sin conexión, no confiable (UDP)

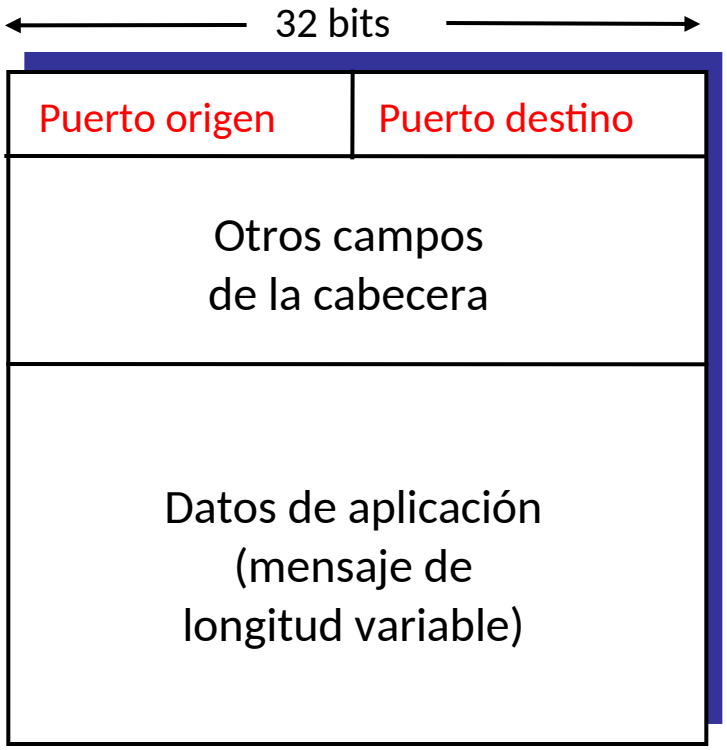
Contenidos

Multiplexión y demultiplexión

- Extensión del servicio de entrega de host a host que proporciona el nivel de red a un servicio de entrega de **proceso a proceso** para el nivel de aplicación
- El nivel de transporte tiene la responsabilidad de entregar los segmentos recibidos al proceso correcto
  - En realidad, se deben entregar los segmentos al socket correspondiente
- Multiplexión:
  - Obtener información de los distintos sockets, crear los segmentos y pasarlos al nivel de red
- Demultiplexión:
  - Entregar los segmentos procedentes del nivel de red a los sockets apropiados

Direccionamiento de procesos

- Cada paquete IP recibido por un host tiene direcciones IP de origen (S) y de destino (D)
- Cada paquete IP transporta un segmento TCP o UDP
- Cada segmento tiene números de puerto origen y de puerto destino
- El host utiliza las direcciones IP y los números de puerto para entregar el segmento al socket apropiado
  - UDP: (IP D, Port D)
  - TCP: (IP D, Port D, IP S, Port S)



Formato general del segmento TCP/UDP

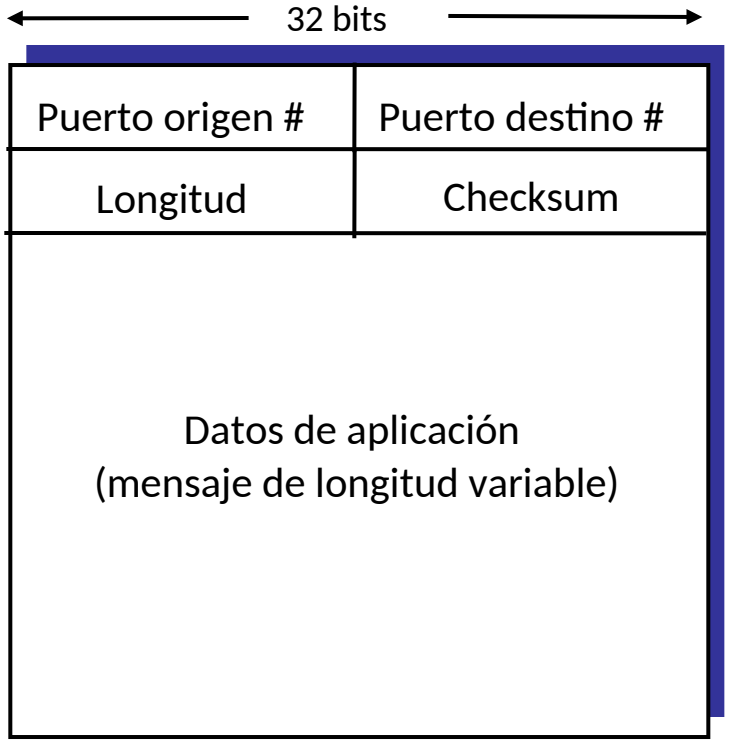
Contenidos

Introducción

- Protocolo del nivel de transporte que sólo incorpora las funciones de multiplexión/demultiplexión y detección de errores
- Servicio *best effort* (“se hace lo que se puede”) del nivel de transporte:
  - No garantizado (los segmentos pueden perderse)
  - No asegura la entrega en orden de los segmentos
- Protocolo sin conexión:
  - No hay negociación alguna entre emisor y receptor
  - Cada segmento UDP se trata de forma independiente
- Protocolo multipunto:
  - Un proceso puede enviar/recibir información a/desde otros procesos con el mismo socket:(IP D/S, Port D/S)
- RFC 768

Campos del segmento UDP

- Formado por dos partes:
  - Cabecera de control
  - Datos del nivel de aplicación (mensaje de longitud variable)
- Campos de la cabecera:
  - Puertos del proceso emisor y del proceso receptor del segmento
  - Longitud, en bytes, del segmento UDP incluyendo la cabecera
  - Checksum para controlar si el segmento ha cambiado durante su transmisión (errores)



Formato del segmento UDP

Detección de errores

- Cálculo del checksum:
  - Tratar los contenidos del segmento como una secuencia de enteros de 16 bits
  - El checksum es la suma en complemento a 1 de todos los enteros de la secuencia
  - El emisor incluye el checksum calculado en el campo correspondiente del segmento
- Comprobación del checksum:
  - El receptor calcula el checksum de la misma manera
  - Si ambos coinciden, no hubo errores durante su transmisión
  - En otro caso, el segmento se descarta

UDP frente a TCP

- Razones para usar UDP frente a TCP:
  - La aplicación tiene más control acerca de cómo se envía la información (beneficioso para aplicaciones multimedia)
    - No hay control de flujo ni recuperación de errores, lo cual evita la retransmisión de segmentos perdidos o erróneos
    - No hay control de la congestión, por lo que se puede enviar a la velocidad deseada
  - Se elimina el retraso asociado a establecer la conexión
  - No se necesita guardar el estado de la conexión
    - Normalmente, un servidor de aplicaciones podrá atender a más clientes si usa UDP en lugar de TCP
  - La cabecera del segmento es menor (la cabecera UDP tiene 8 bytes frente a la cabecera TCP que tiene 20 bytes) y, por tanto, la eficiencia es mayor

UDP en la práctica

- Protocolo de transporte habitual para aplicaciones multimedia en tiempo real o que usan *streaming*
  - Toleran la pérdida de información. . .
  - . . . pero son sensibles a la velocidad de transmisión
- Otros usos de UDP:
  - DNS (*Domain Name Service*)
  - DHCP (*Dynamic Host Configuration Protocol*)
  - SNMP (*Simple Network Management Protocol*)
  - RIP (*Routing Information Protocol*)
- En caso de que se requiera un servicio de transmisión confiable basado en UDP:
  - Los mecanismos para garantizarla correspondientes deben incluirse en el nivel de aplicación

Contenidos

Comunicación confiable

- El problema de la transmisión o comunicación confiable es una cuestión crítica en varios niveles de la arquitectura de red
- Un protocolo confiable debe proveer un servicio que
  - Garantice que ningún bit será modificado o se perderá
  - Asegure que todos los bits se entregarán en el destino en el mismo orden en que fueron enviados
- . . . considerando que la capa inferior es un canal punto a punto no confiable

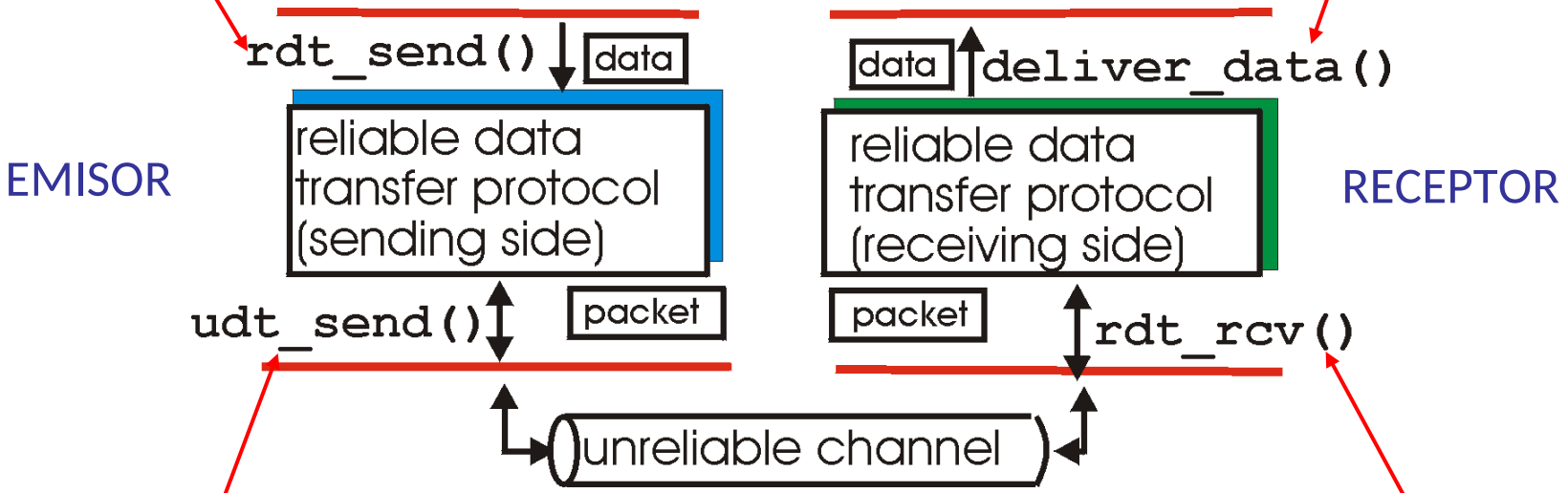
Enfoque

- Abordaremos la solución de forma general y más adelante veremos cómo se implementa en TCP
- Partiremos de un protocolo sencillo
  - rdt1.0 (*reliable data transfer*)
- Modificaremos el protocolo para contemplar las diferentes situaciones de error más comunes
- Describiremos el protocolo mediante máquinas de estado finitas
- Definiremos también las primitivas de servicio entre capas
- Asumiremos, por simplicidad, que se envían datos sólo en un sentido aunque haya flujo de información de control en ambos sentidos

Primitivas de envío y recepción

rdt_send(): usada por la capa superior para que los datos sean entregados a la capa superior del receptor

deliver_data(): usada por **rdt** para entregar datos a la aplicación

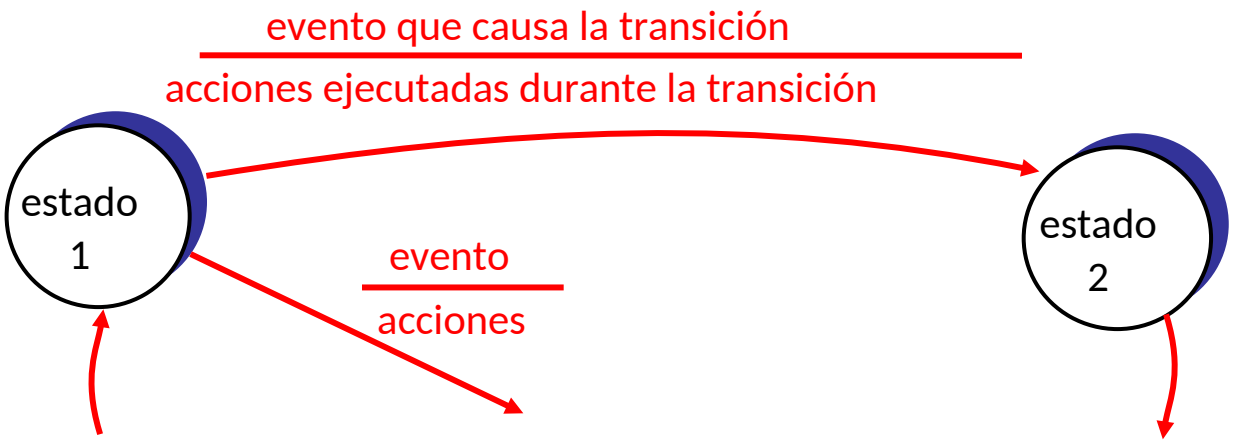


udt_send(): usada por **rdt** para enviar un segmento por el canal no confiable al receptor

rdt_rcv(): usada por la capa inferior cuando llega un segmento por el canal no confiable al receptor

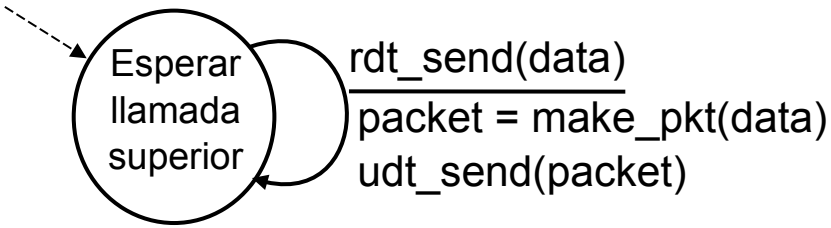
Máquinas de estado finitas

- Especificaremos el comportamiento del emisor y del receptor mediante máquinas de estado finitas

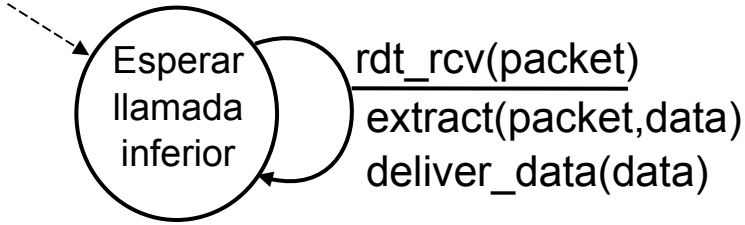


rdt1.0: Canal confiable

- El canal subyacente es confiable
  - No hay errores de bits
  - No hay pérdida de segmentos
- Dos máquinas de estados finitas distintas para emisor y receptor
  - El emisor envía datos a través del canal subyacente
  - El receptor lee datos del canal subyacente



Emisor

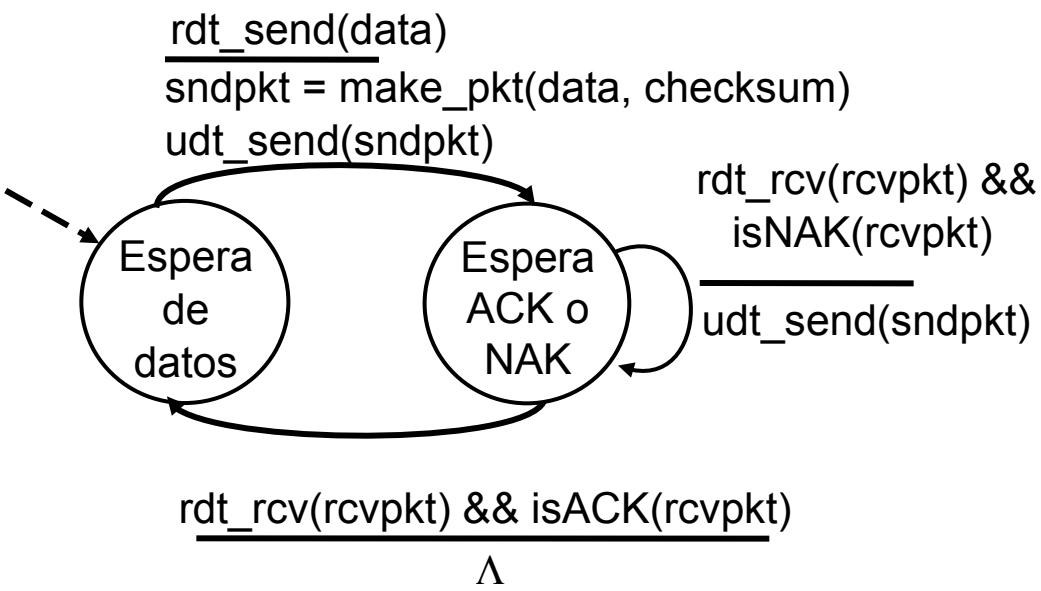


Receptor

rdt2.0: Canal con errores de bits

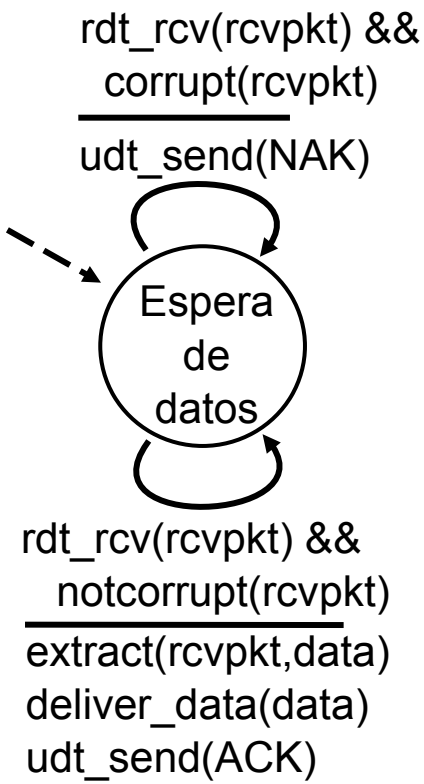
- El canal subyacente puede alterar los bits del segmento (pero no hay pérdida de segmentos)
  - Se utiliza un mecanismo de checksum para detectarlo
- Mecanismos básico para recuperarse de los errores:
  - Asentimientos (ACKs): el receptor indica explícitamente que el segmento se recibió correctamente
    - El emisor no envia otro segmento hasta que recibe un ACK
      - rdt2.0 es un protocolo de “parada y espera” (*stop-and-wait*)
  - Rechazos (NAKs): el receptor indica explícitamente que el segmento contenía errores
    - El emisor retransmite el mismo segmento si recibe un NAK
      - rdt2.0 es un protocolo con ARQ (*Automatic Repeat reQuest*)

rdt2.0: Canal con errores de bits

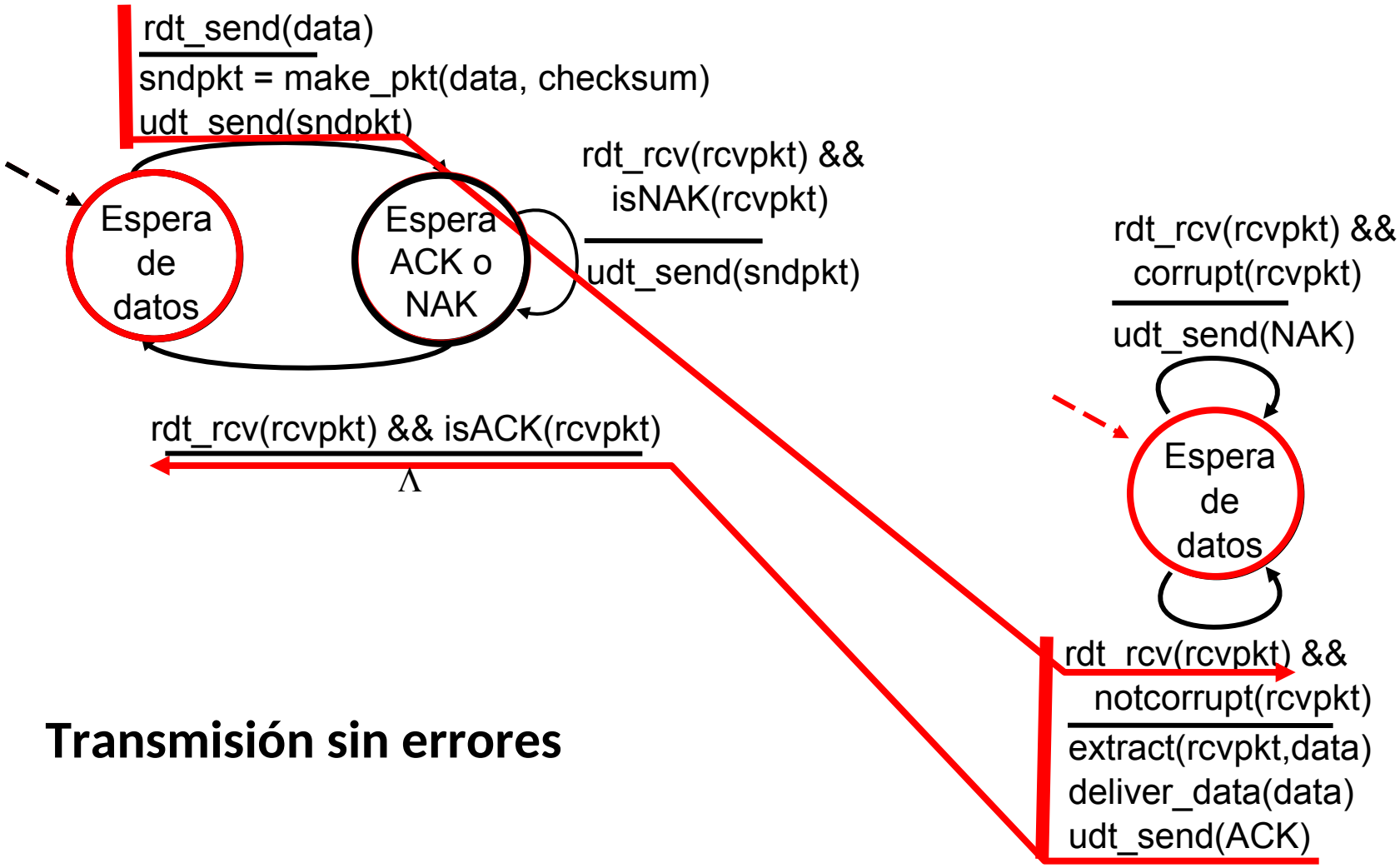


Emisor

Receptor

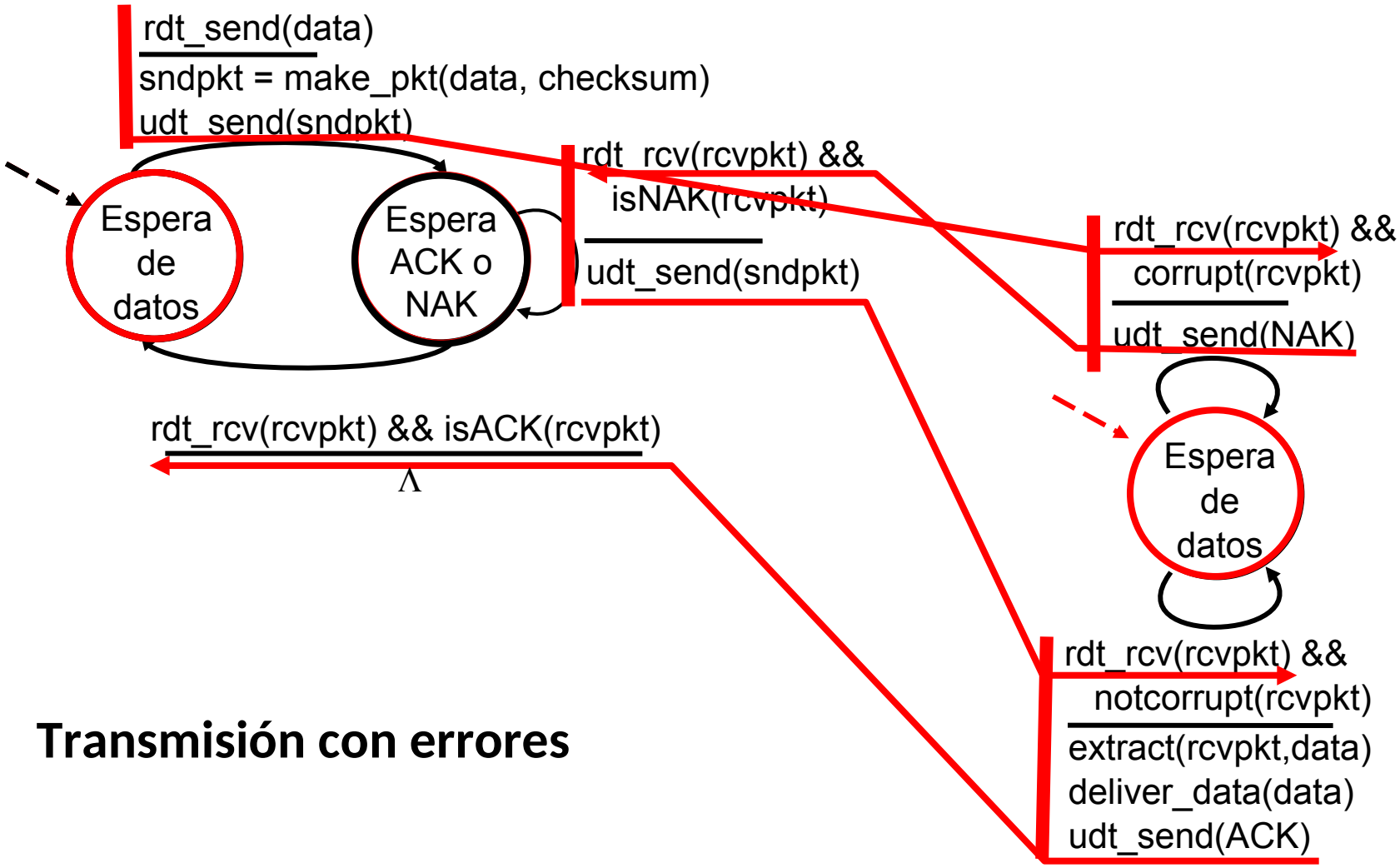


rdt2.0: Canal con errores de bits



Transmisión sin errores

rdt2.0: Canal con errores de bits



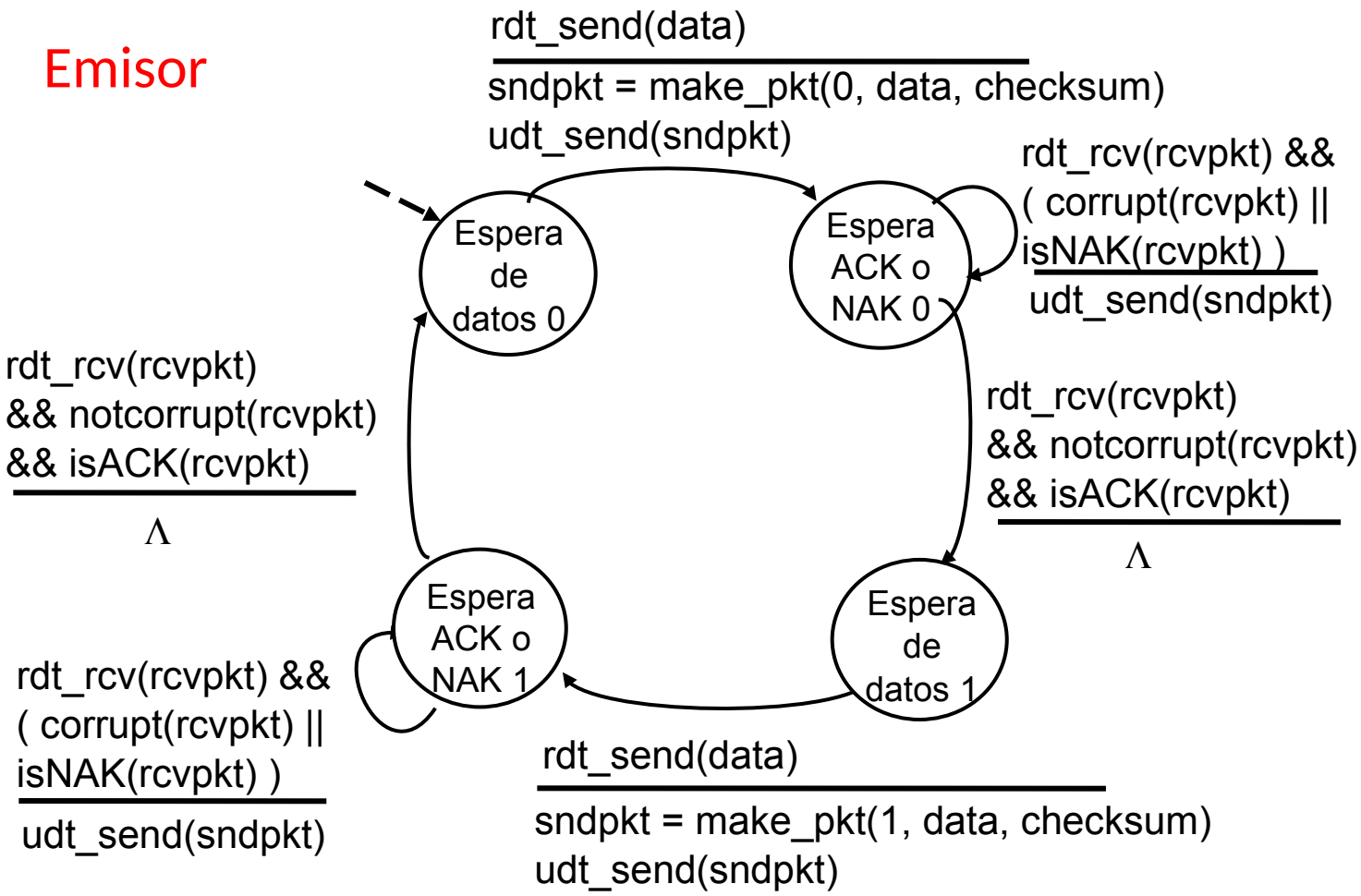
Transmisión con errores

rdt2.0 tiene un fallo bastante grave

- ¿Qué sucede si un ACK o un NAK están corruptos?
  - El emisor no sabrá realmente qué pasó en el receptor
  - El emisor no puede retransmitir el segmento porque el receptor lo podría procesar aún siendo un duplicado
- Mecanismo adicional para manejar los duplicados:
  - El emisor añadirá un número de secuencia a cada segmento
  - El emisor transmitirá de nuevo el segmento si el ACK o el NAK se han visto afectados por un error
  - El receptor descartará todos los segmentos duplicados, es decir, ya recibidos anteriormente

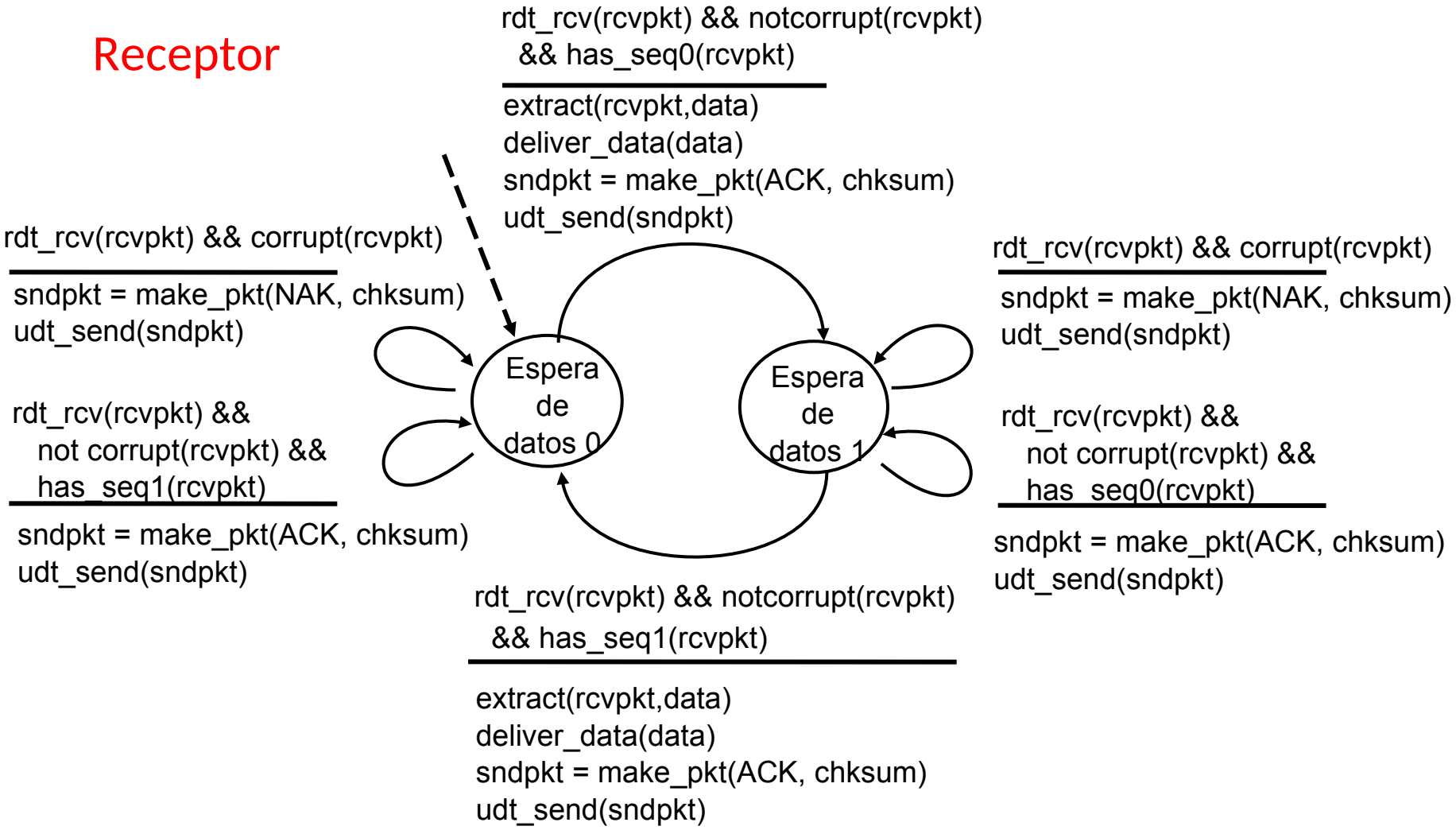
rdt2.1: Manejo de ACK/NAK erróneos

Emisor



rdt2.1: Manejo de ACK/NAK erróneos

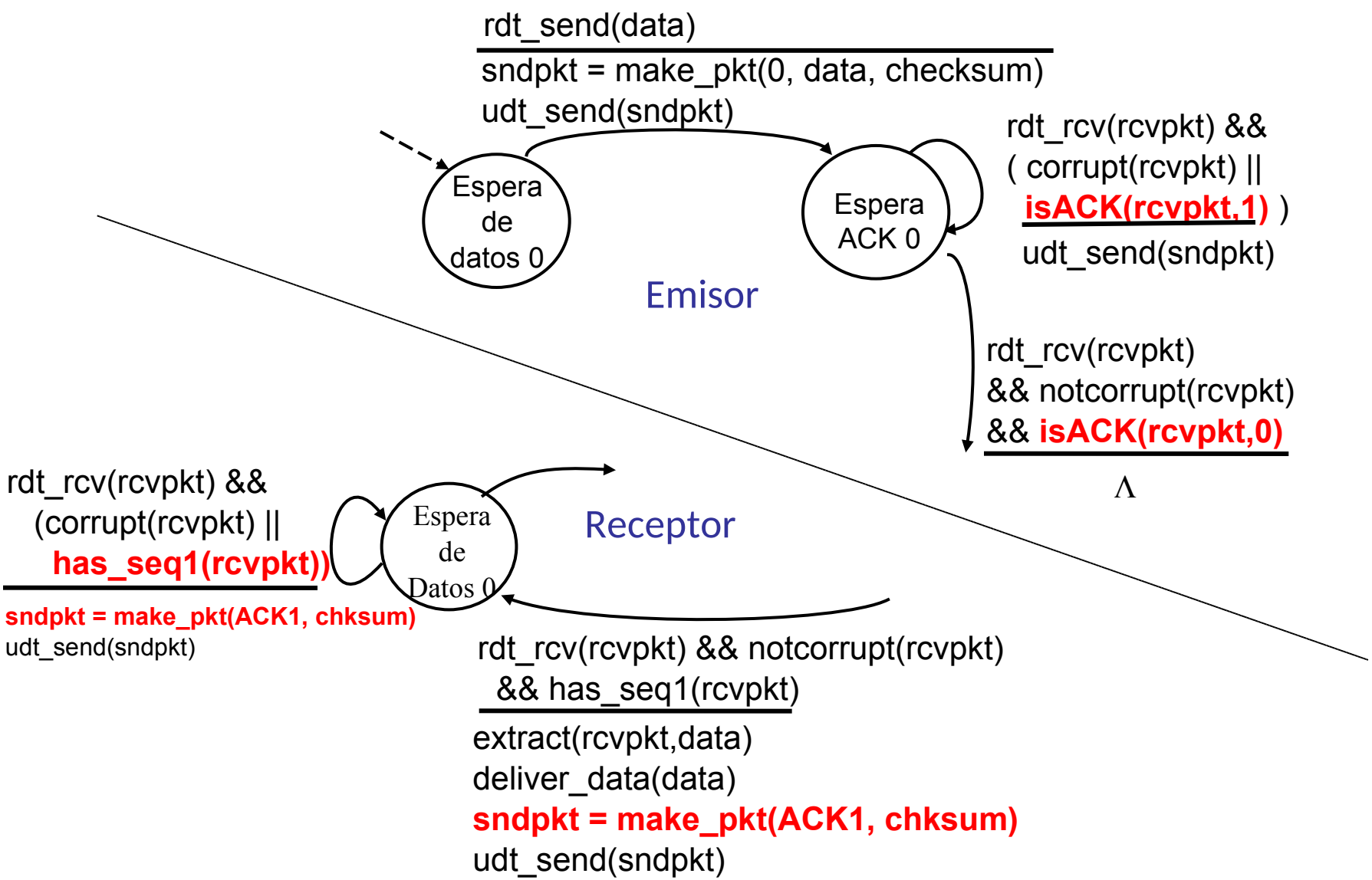
Receptor



rdt2.2: Protocolo sin NAKs. Ejercicio.

- Desarrolla un protocolo (máquinas de estado finitas) con la misma funcionalidad que rdt2.1, pero usando sólo ACKs, es decir, sin usar NAKs
  - El receptor incluirá el número de secuencia del segmento que se está confirmando en el ACK
  - En lugar de un NAK, el receptor enviará el ACK del último segmento recibido correctamente si el último segmento recibido contenía errores
  - La recepción de un ACK duplicado implica la misma acción que un NAK, es decir, retransmitir el último segmento

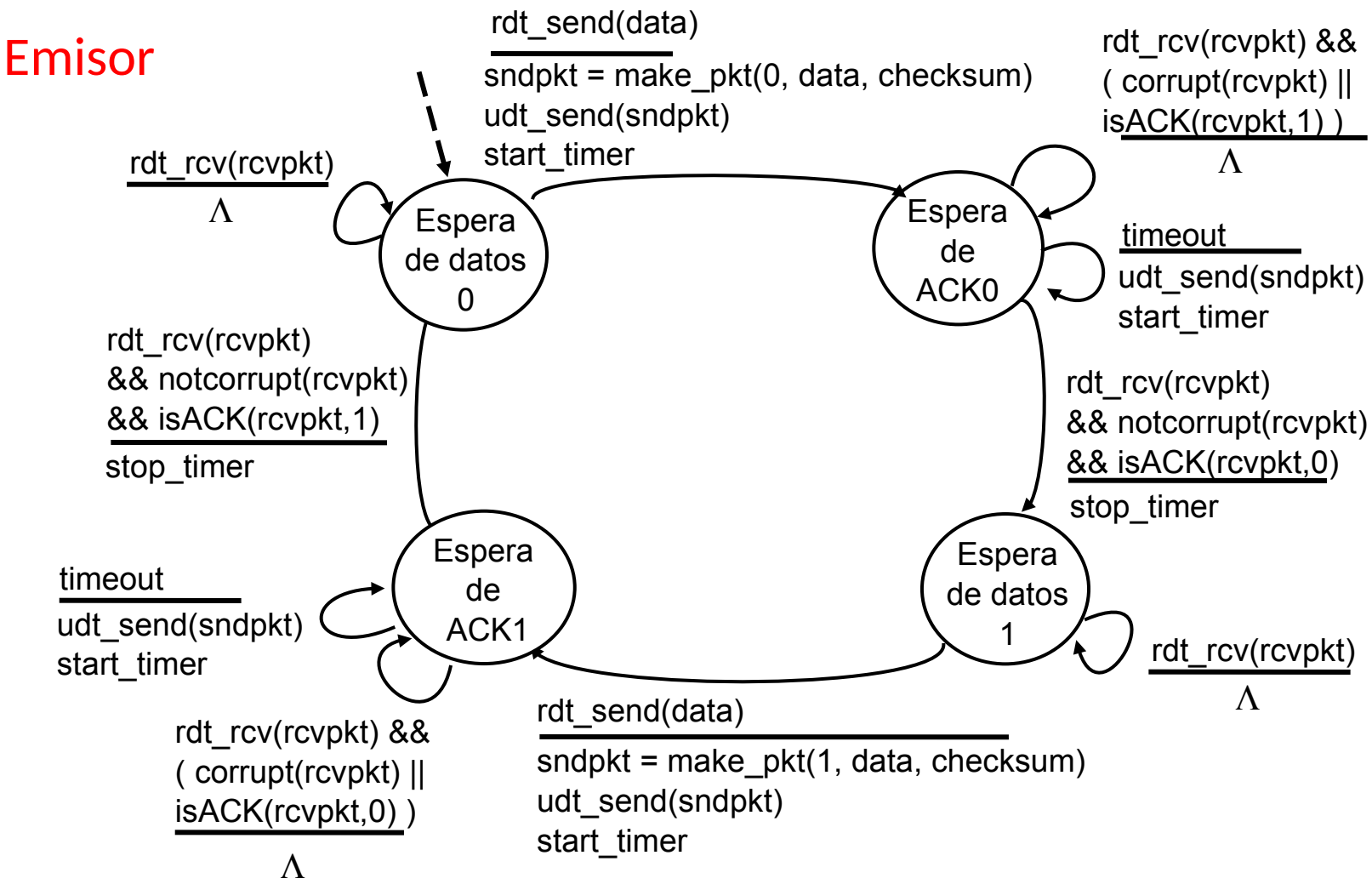
rdt2.2: Protocolo sin NAKs. Solución.



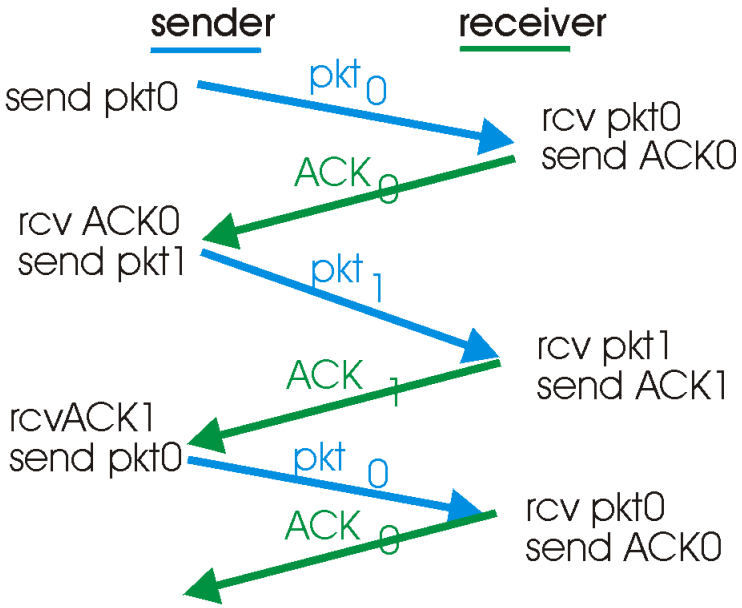
rdt3.0: Canal con errores y pérdidas de bits

- El canal subyacente puede también perder segmentos (tanto de datos como ACKs)
  - En ese caso, el checksum, los números de secuencia, los ACKs y las retransmisiones no son suficientes
- Mecanismo adicional para recuperarse de las pérdidas (tanto de datos como de ACKs):
  - El emisor espera durante un tiempo *razonable* el ACK
    - Temporizador de cuenta atrás (*timeout*)
  - El segmento se retransmite si no llega el ACK durante ese tiempo
  - Si los datos o el ACK sólo se retrasan (no se pierden):
    - La retransmisión generará duplicados que pueden detectarse (y descartarse) mediante los números de secuencia de los segmentos

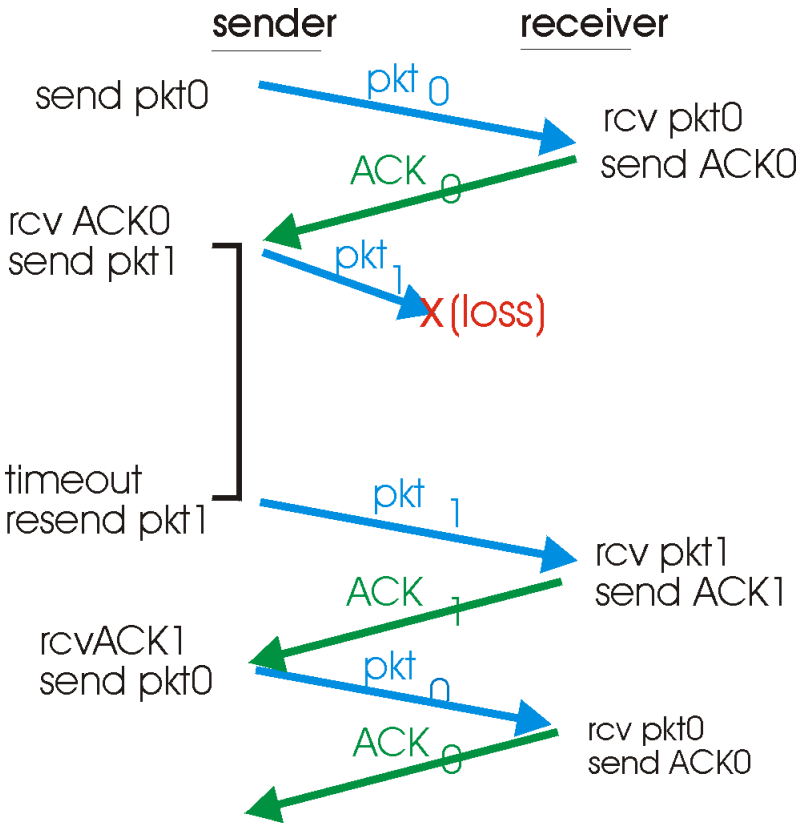
rdt3.0: Canales con errores y pérdidas de bits



rdt3.0: Diagramas de uso

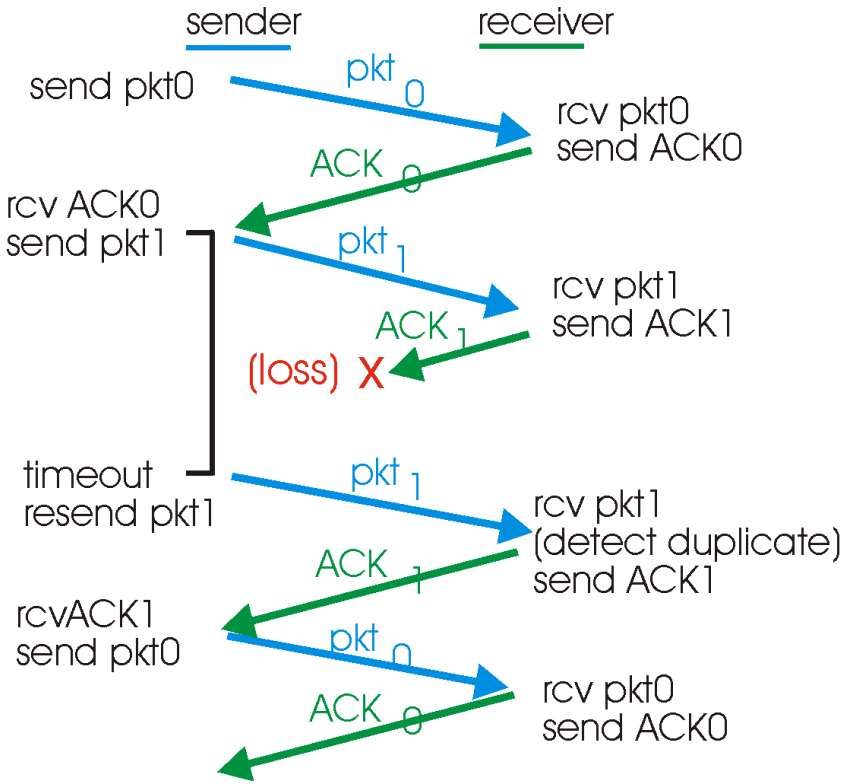


(a) operation with no loss

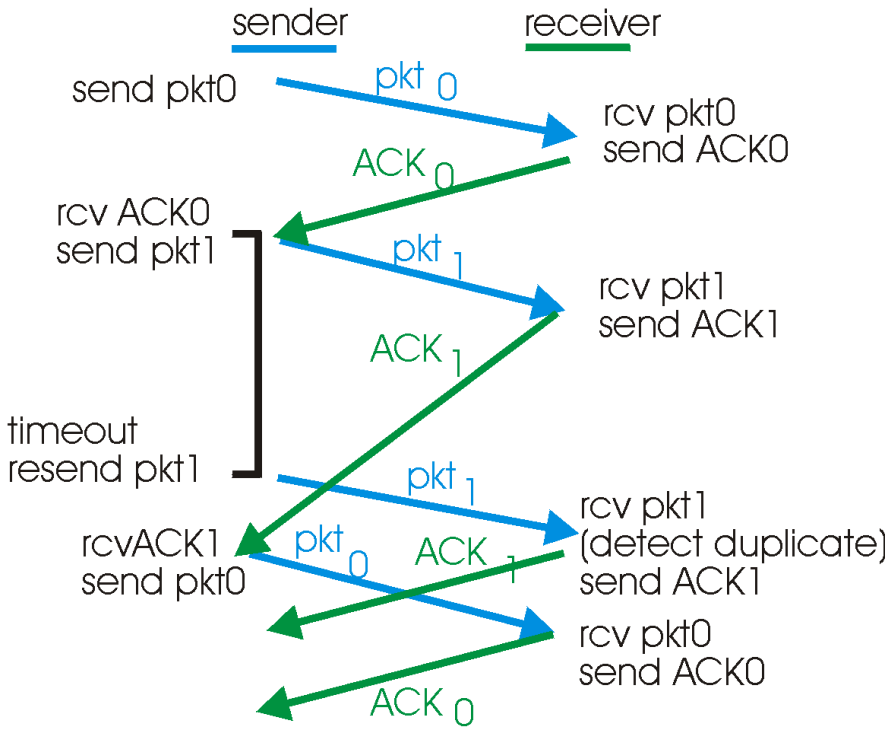


(b) lost packet

rdt3.0: Diagramas de uso



(c) lost ACK



(d) premature timeout

Eficiencia de rdt3.0 sin errores

- Protocolo de parada y espera (*stop-and-wait*)
  - Si el emisor espera una confirmación por cada segmento transmitido y el siguiente segmento se envía sólo cuando se recibe su confirmación
  - El tiempo total para transmitir un segmento y su confirmación es:

$$T = t_{prop} + t_{seg} + t_{proc} + t_{prop} + t_{ack} + t_{proc}$$

t_{prop}: *tiempo de propagación*

t_{seg}: *tiempo de transmisión de un segmento*

t_{proc}: *tiempo de procesamiento de un segmento o ACK*

t_{ack}: *tiempo de transmisión de un ACK*

  - Ventaja: simplicidad (cada segmento se transmite y se confirma individualmente)
  - Desventaja: ineficiencia (cada segmento y su confirmación recorren todo el camino entre emisor y receptor antes de poder enviar el siguiente)

Eficiencia de rdt3.0 sin errores

- Protocolo de parada y espera (*stop-and-wait*)
  - La utilización o eficiencia del protocolo se define como:

$$U = \frac{t_{información}}{t_{total}} = \frac{t_{útil}}{2t_{prop} + t_{seg} + t_{ack} + 2t_{proc}}$$

  - donde

$$t_{prop} = \frac{d}{V_{prop}}$$

d = distancia del enlace
V_{prop} = velocidad de propagación

$$t_{seg} = \frac{l}{V_{trans}}$$

l = longitud total del segmento
V_{Trans} = velocidad de transmisión

  - Si se conoce el tamaño de la cabecera del protocolo (h) entonces $t_{útil}$ es el tiempo necesario para transmitir sólo los datos del segmento (l-h), sin la cabecera
  - Si no se conoce, entonces $t_{útil}$ es igual a t_{seg}

Problemas

- **Problema 1. Se desea transmitir una serie de segmentos de 1000 bits con un protocolo de parada y espera. Calcula la eficiencia máxima del protocolo (en ausencia de errores) suponiendo una velocidad de transmisión de 1Kbps y 1 Mbps en los siguientes supuestos:**
  - **(a) par trenzado de 1 Km**
  - **(b) cable coaxial de 200 Km**
  - **(c) conexión por satélite de 50.000 Km**

  La V_{prop} es de $2x10^8$m/s en un medio cableado y de $3x10^8$m/s en un medio inalámbrico.

Contenidos

Técnicas de ventana deslizante

- Protocolo de ventana deslizante (*sliding-window*)
  - El emisor puede enviar varios segmentos antes de recibir una confirmación
    - Permite aprovechar más eficientemente el canal
  - El emisor guarda copias de los segmentos enviados hasta que recibe el ACK correspondiente
  - El receptor puede confirmar la recepción de uno o más segmentos en cualquier momento mediante un ACK
  - Mecanismo de control de flujo:
    - El número máximo de segmentos enviados pendientes de confirmación se conoce como ventana (*buffer*)
    - Esta limitación evita que un emisor rápido inunde a un receptor lento

Técnicas de ventana deslizante

- Protocolo de ventana deslizante (*sliding-window*)
  - Esta técnica también requiere un esquema de numeración de los datos y los ACKs para realizar un seguimiento de los segmentos enviados y recibidos
    - Cada segmento de datos incluye un número de secuencia
    - Cada ACK debe incluir información acerca de cuántos segmentos está confirmando
  - El emisor también dispone de uno o más temporizadores para recuperarse de las pérdidas de segmentos
  - Hay múltiples variantes en función del tipo de rechazo que se produzca y del uso de los temporizadores:
    - **Vuelta atrás N (*Go-Back-N*)**
    - **Rechazo selectivo (*Selective Reject*)**

https://media.pearsoncmg.com/ph/esm/ecs_kurose_compnetwork_8/cw/#interactiveanimations

Eficiencia de la ventana deslizante sin errores

- Protocolo de ventana deslizante (*sliding-window*)
  - La utilización o eficiencia de la línea se define como:

$$U = \frac{N * t_{útil}}{2t_{prop} + t_{seg} + t_{ack} + 2t_{proc}}$$

donde N es el número de segmentos que se pueden enviar sin esperar confirmación.

Problemas

- **Problema 2. Se desea transmitir una serie de segmentos de 1000 bits con un protocolo de ventana deslizante. Calcula la eficiencia máxima del protocolo (en ausencia de errores) suponiendo una velocidad de transmisión de 1 Mbps en los siguientes supuestos:**
  - **(a) par trenzado de 1 Km con N=1**
  - **(b) cable coaxial de 200 Km con N=7**
  - **(c) conexión por satélite de 50.000 Km con N=127**

  La V_{prop} es de $2x10^8$m/s en un medio cableado y de $3x10^8$m/s en un medio inalámbrico.

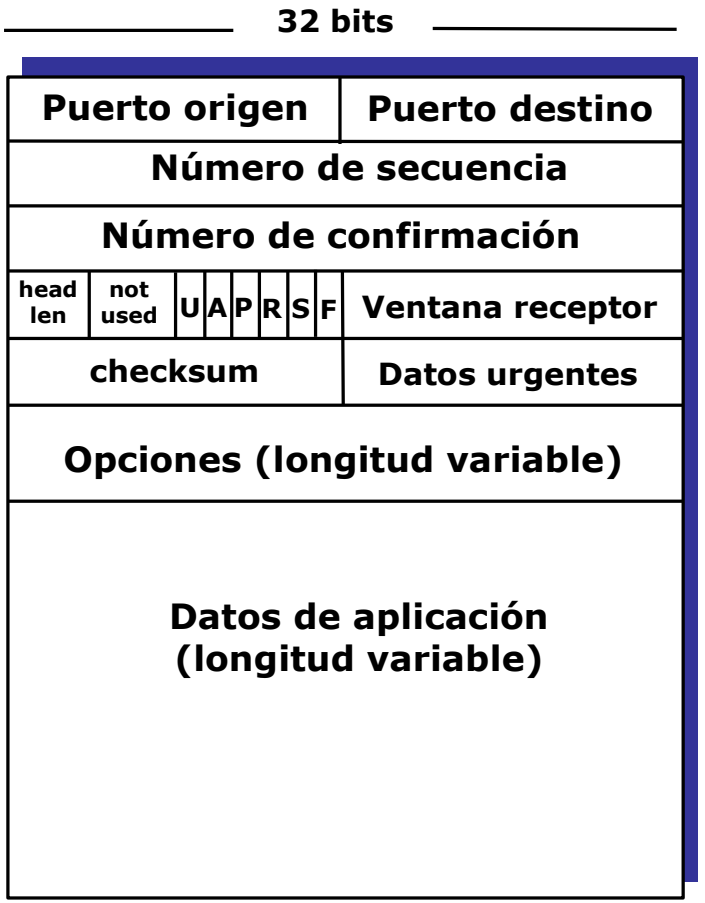
Contenidos

Introducción

- Servicio confiable del nivel de transporte:
  - La información no puede perderse y siempre se entrega en orden al nivel de aplicación
- Protocolo orientado a la conexión:
  - El emisor inicia una negociación (intercambio de mensajes de control) con el receptor para establecer la conexión
- Protocolo punto a punto con transmisión full-dúplex
  - Sólo un emisor y un receptor: (IP D, Port D, IP S, Port S)
  - El flujo de información es bidireccional en cada conexión
- Control de flujo y recuperación de errores basados en técnicas de ventana deslizante
  - *Buffers*, ACKs acumulativos, retransmisión de segmentos, números de secuencia y temporizador (*timeout*)
- Sólo se implementa en los sistemas finales

Campos del segmento TCP

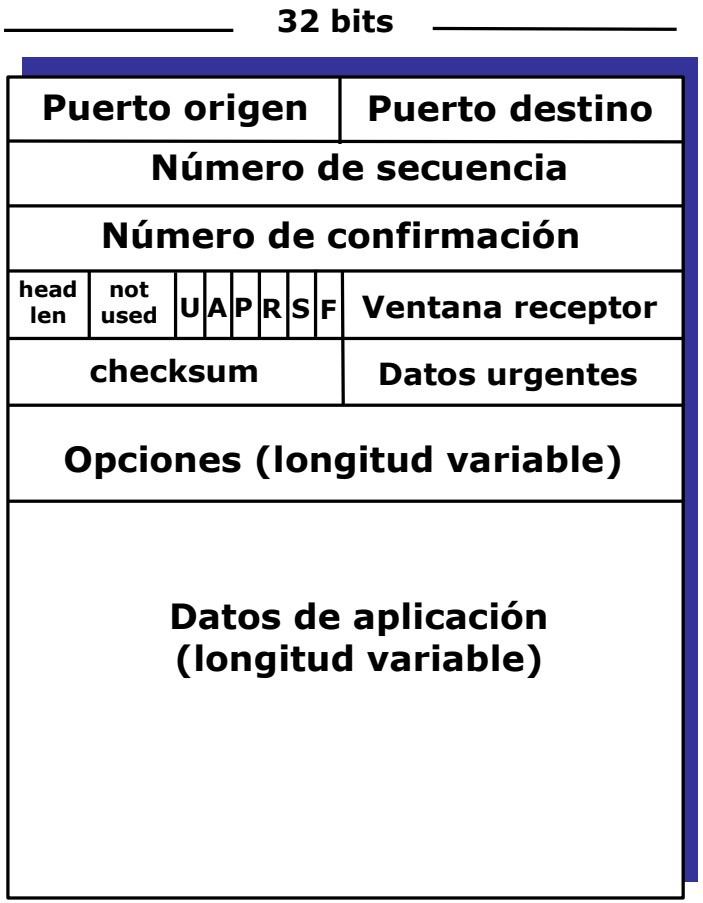
- Formado por dos partes:
  - Cabecera de control
  - Datos del nivel de aplicación (longitud variable)
- Hay una longitud máxima del segmento (*MSS, Maximum Segment Size*):
  - Se define como el tamaño máximo del campo de datos (sin incluir la cabecera)
  - Está condicionado por las capas inferiores (tamaño de la trama)
  - Si los datos de aplicación no caben en un segmento se generan varios



Formato del segmento TDP

Campos del segmento TCP

- Campos principales de la cabecera:
  - Puertos del proceso emisor y del proceso receptor del segmento
  - Número de secuencia y número de confirmación son usados para recuperación de errores
  - Head Len indica la longitud de la cabecera en palabras de 32 bits
  - Flags:
    - ACK (A): el segmento incluye confirmación
    - SYN (S), RST (R), FIN (F): utilizados para establecer, restaurar y cerrar conexiones
  - Ventana receptor se utiliza para control de flujo (número de bytes que se pueden aceptar)
  - Checksum, similar al utilizado por UDP, para controlar si el segmento cambió durante su transmisión (errores)



Formato del segmento TCP

Conexiones TCP

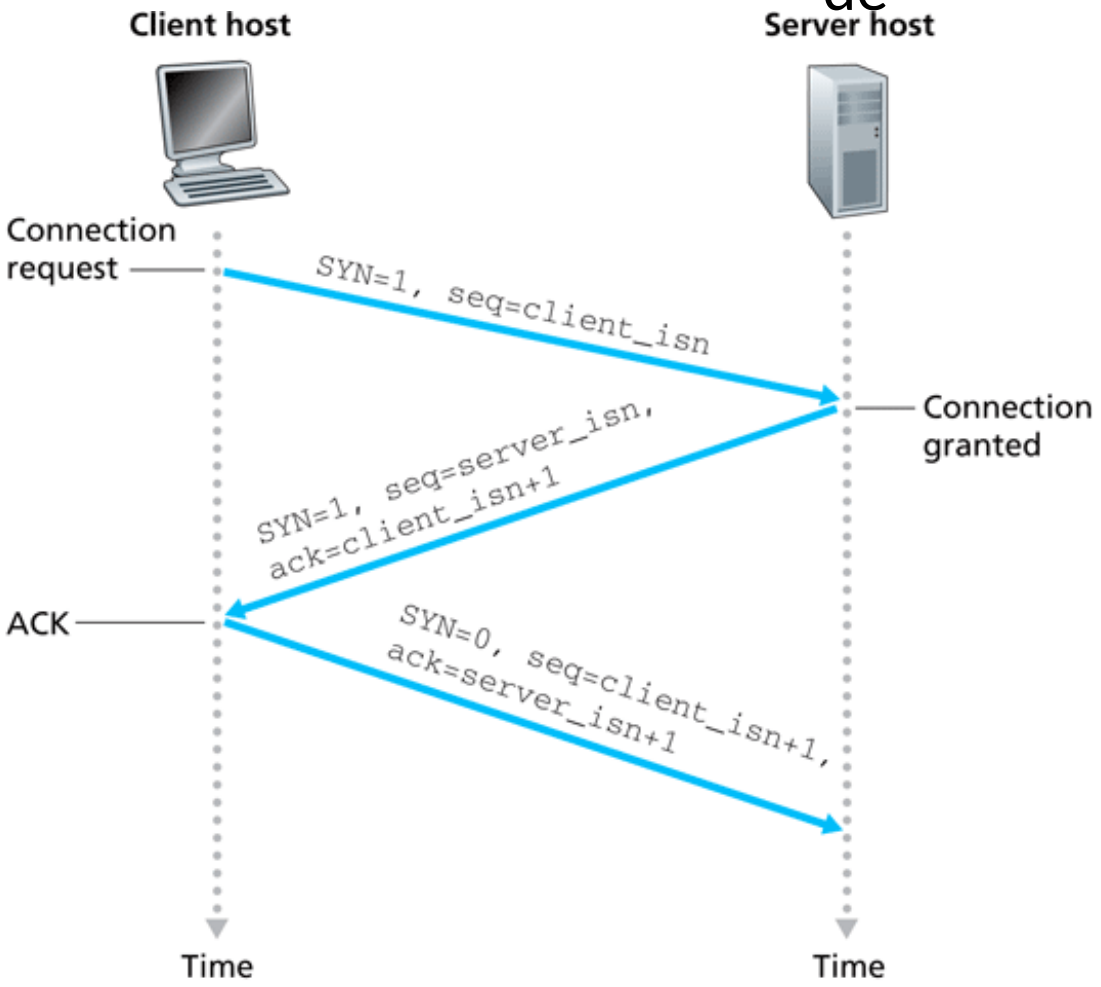
- Tanto el emisor como el receptor deben establecer una conexión antes de comenzar el intercambio de datos procedentes del nivel de aplicación
- El propósito de ese establecimiento es:
  - Inicializar los parámetros relativos a la recuperación de errores
    - Valores de los campos **Número de secuencia** y **Número de confirmación** en ambos sentidos
  - Inicializar los parámetros relativos al control de flujo
    - Valor del campo **Ventana del receptor** en ambos sentidos
  - Reservar *buffers* para la ventana deslizante en ambos sentidos

Conexiones TCP

- El establecimiento consiste en tres fases (*three-way handshake*):
  - Paso 1: el cliente envía un segmento SYN al servidor
    - Especifica el número de secuencia inicial del cliente
    - No incluye datos
  - Paso 2: el servidor recibe el SYN y responde con SYN ACK
    - El servidor reserva los *buffers* necesarios
    - Confirma la recepción del SYN de cliente
    - Especifica el número de secuencia inicial del servidor
  - Paso 3: el cliente recibe el SYN ACK y responde con ACK
    - El cliente reserva los *buffers* necesarios
    - Confirma la recepción del SYN ACK del servidor
    - Puede incluir ya datos para el servidor

Conexiones TCP

- Establecimiento de conexión

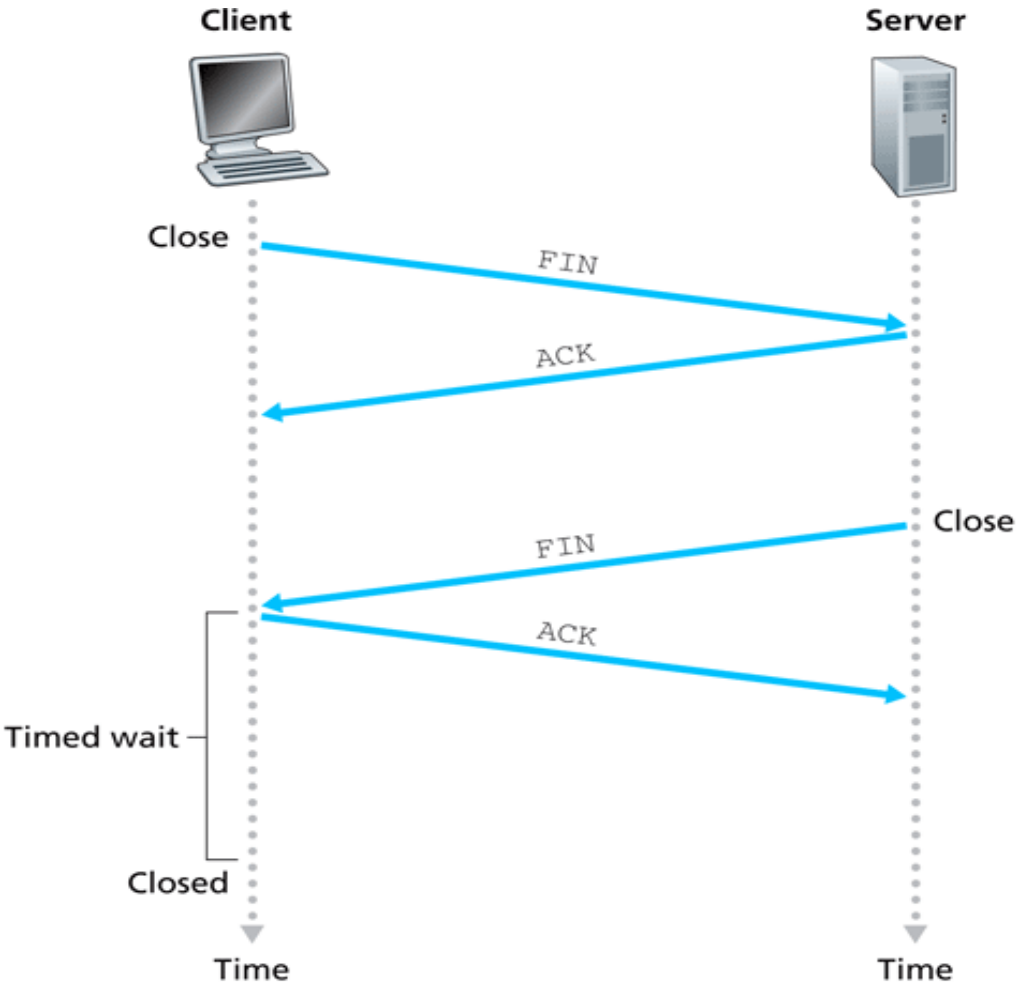


Conexiones TCP

- El cierre consiste en cuatro pasos:
  - Paso 1: el cliente envía un segmento FIN al servidor
  - Paso 2: el servidor recibe el FIN, cierra la conexión, responde con ACK, y envía un FIN
  - Paso 3: el cliente recibe el FIN y responde con un ACK
    - El cliente espera algún tiempo antes de cerrar la conexión
    - Si el ACK se pierde, el servidor enviaría de nuevo un FIN
  - Paso 4: el servidor recibe el ACK y cierra la conexión
- Tanto el emisor como el receptor liberan todos los recursos asociados a la conexión cuando la cierran
  - *Buffers* y puertos

Conexiones TCP

- Cierre de conexión

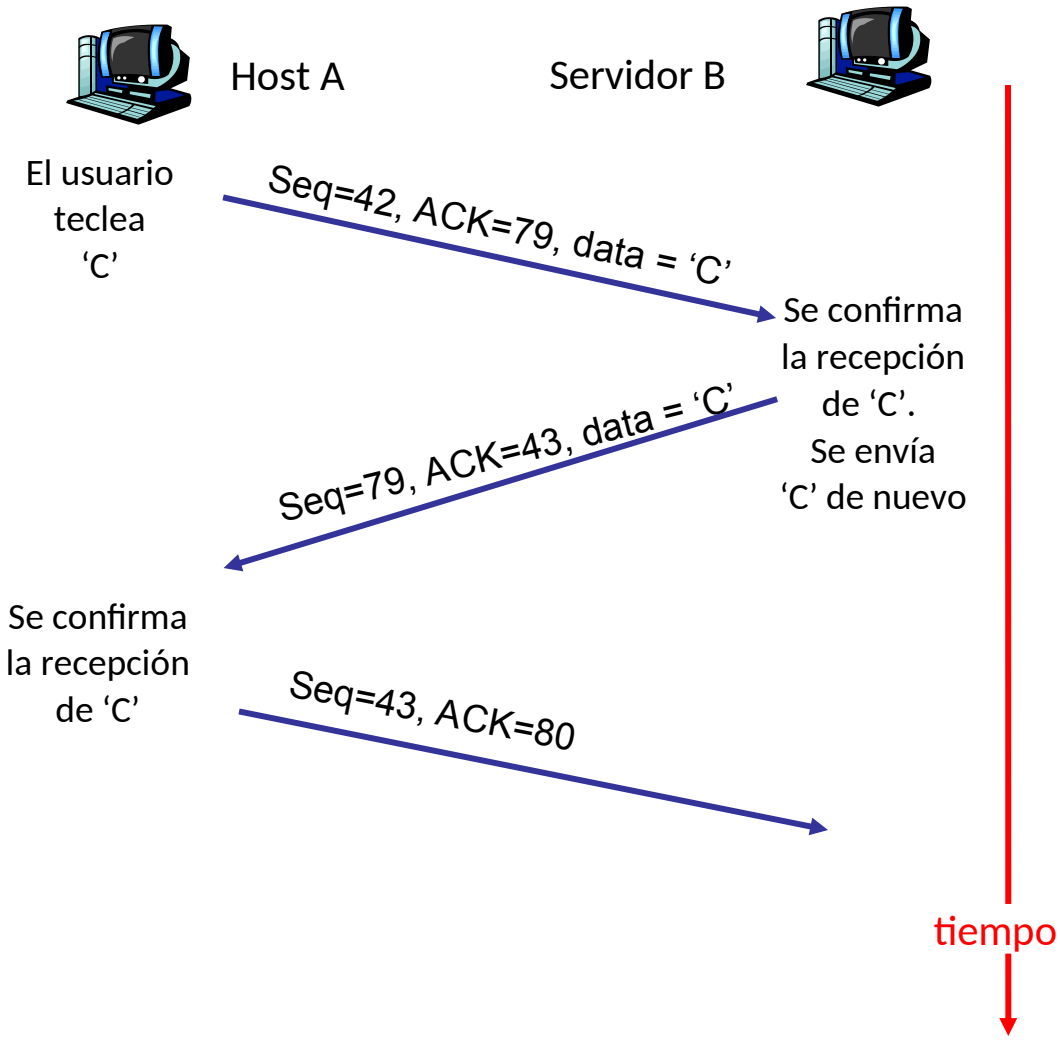


Números de secuencia y confirmaciones

- TCP considera los datos como un **flujo de bytes ordenados**
- Los números de secuencia están relacionados con los bytes transmitidos (no con el número de segmentos)
  - El número de secuencia de un segmento corresponde al primer byte de dicho segmento
  - El número de secuencia del siguiente segmento depende del número de bytes transmitidos anteriormente
- Los números de confirmación también están relacionados con los bytes recibidos (no con el número de segmentos)
  - El número de confirmación de un segmento indica el número de secuencia del byte que se está esperando
  - Las confirmaciones pueden ser acumulativas, es decir, se pueden confirmar varios segmentos simultáneamente
- Si se recibe un segmento fuera de orden, el receptor puede optar por descartarlo o almacenarlo en espera de los anteriores (las implementaciones de TCP habitualmente lo almacenan)

Escenario de ejemplo: Telnet

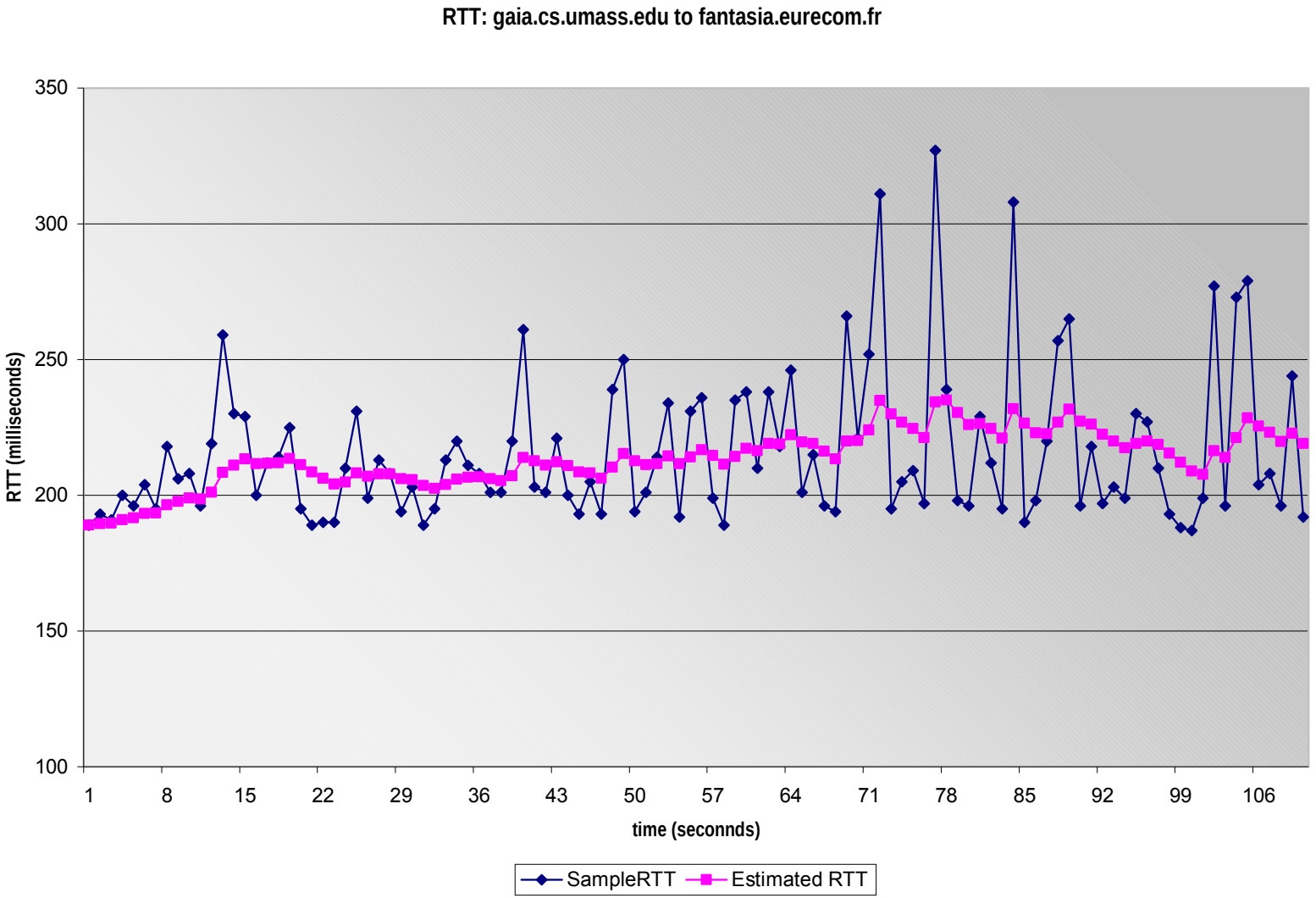
- Uso de los números de secuencia y confirmaciones



Estimación del temporizador (*timeout*)

- ¿Cómo se establece el valor del temporizador?
  - Debe ser mayor que el *Round-Trip Time* (tiempo de ida y vuelta) o RTT necesario para que un segmento pueda llegar a su destino y se reciba su confirmación
  - RTT varía en función del camino seguido por paquetes
- Si fuera demasiado bajo, se podrían producir retransmisiones innecesarias
- Si fuera demasiado alto, se reaccionaría lentamente ante las pérdidas de segmentos
- TCP lleva a cabo una estimación constante del RTT
  - Se mide el tiempo transcurrido desde que se envía un segmento hasta que se recibe su ACK
  - Como los valores obtenidos fluctúan, se calcula la media teniendo en cuenta los valores recientes

Ejemplo de estimación del temporizador

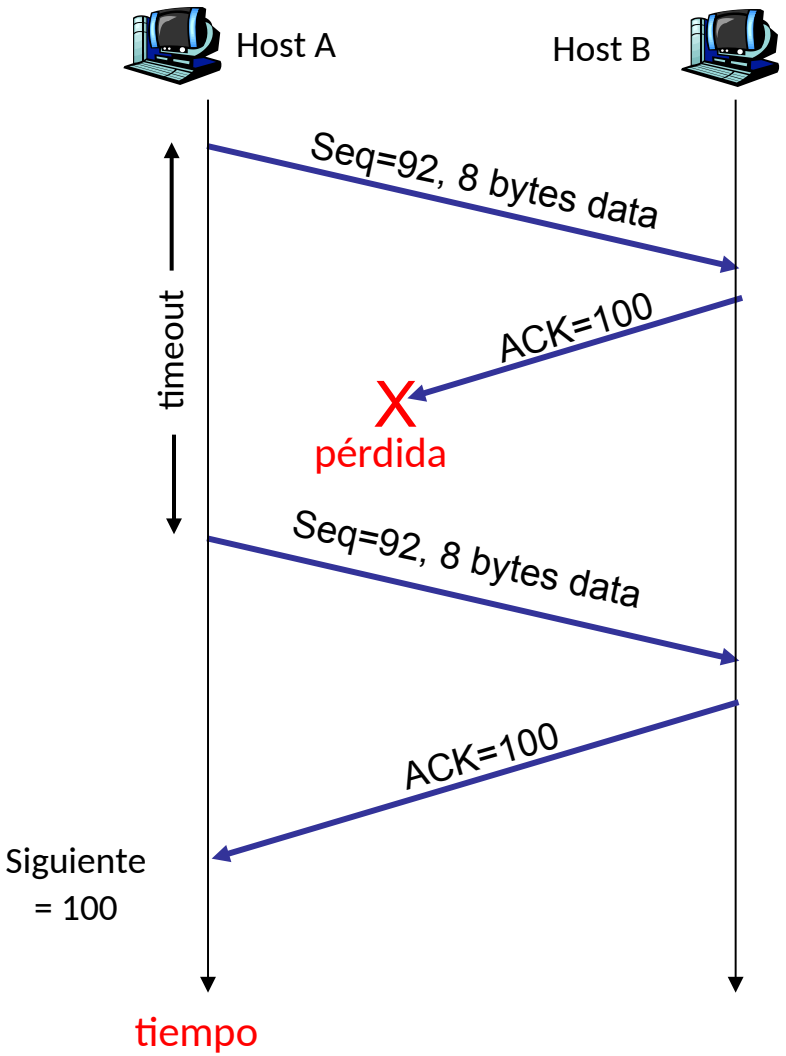


Funcionamiento del emisor

- Al recibir datos desde el nivel de aplicación:
  - Se crea un segmento con el número de secuencia apropiado y se envía
  - Se inicia el temporizador si está detenido (TCP sólo usa un temporizador para controlar el segmento menos reciente)
- Si expira el temporizador (*timeout*):
  - Se retransmite sólo el segmento asociado al temporizador
  - Se reinicia el temporizador
- Si se recibe un ACK:
  - Si está confirmando segmentos aún no confirmados
    - Se actualiza el *buffer* de emisión de acuerdo con la confirmación
    - Se reinicia el temporizador si aún quedan segmentos pendientes de confirmación
  - Si se trata de un ACK repetido (y recibido por cuarta vez)
    - Se asume que el segmento se ha perdido y se reenvía sin esperar a que venza el temporizador (*Fast Retransmit*)

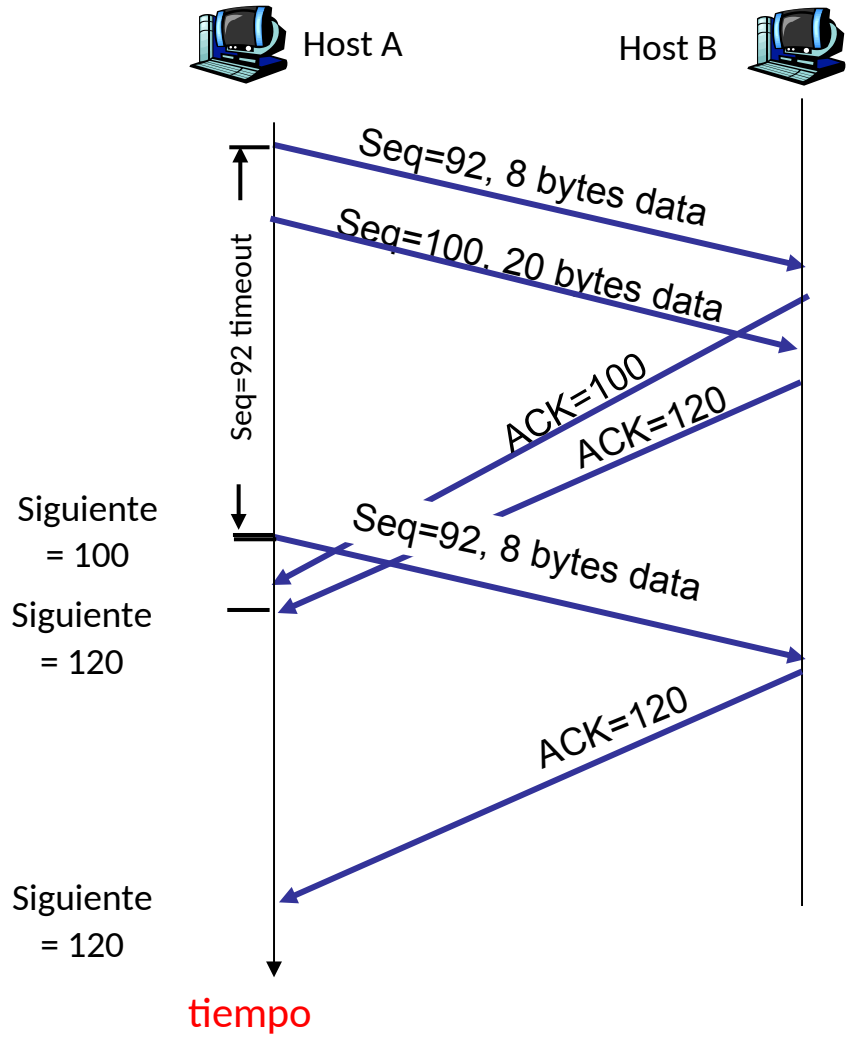
      (La frecuencia de generación de ACKs por parte del receptor es dependiente de la implementación, pero habitualmente genera un ACK duplicado cuando recibe un segmento fuera de orden)
- También existen ACKs selectivos, aunque su uso es opcional

Escenarios de ejemplo (I)



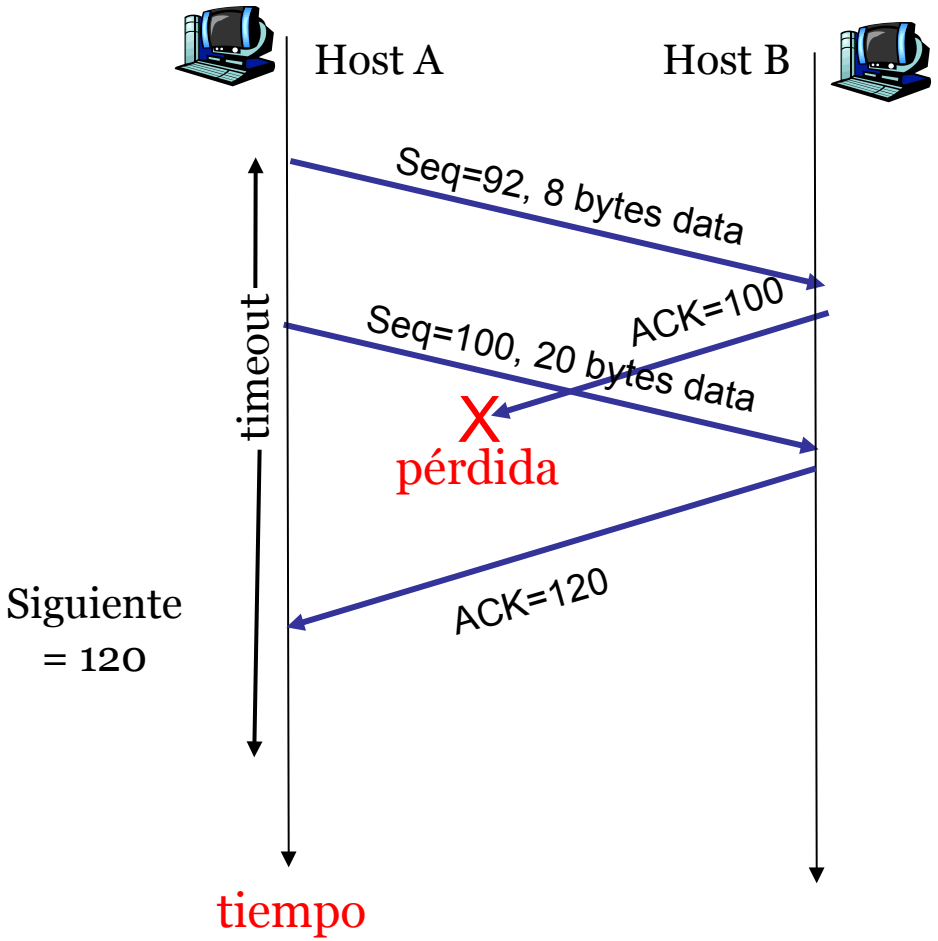
Pérdida de un ACK

Escenarios de ejemplo (II)



Timeout prematuro

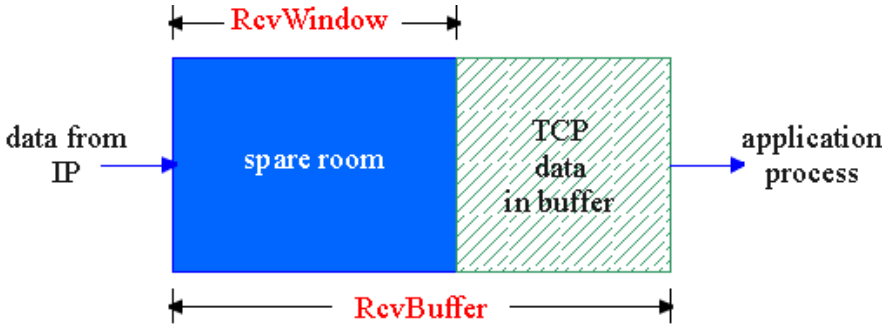
Escenarios de ejemplo (III)



ACK acumulativo

Control de flujo

- El *buffer* de recepción de una conexión TCP es finito
- La aplicación que lee el *buffer* puede ser lenta
- El control de flujo sincroniza la velocidad de emisión con la frecuencia de lectura del *buffer* de recepción
- El receptor informa del espacio disponible en el *buffer* mediante el campo **Ventana del receptor** (cantidad de bytes no usados del *buffer*)
- El emisor limita la cantidad de bytes sin confirmar evitando así el desbordamiento de dicho *buffer*



Tema 2
El nivel de transporte

(1904) Redes de Comunicaciones