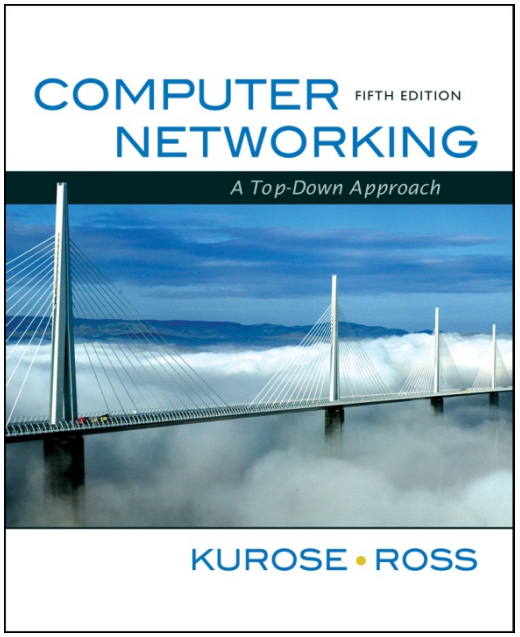
Capa de transporte
Nivel 4



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach 5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Capítulo 3: Capa de transporte (Nivel 4)

Objetivos:

- **Comprender servicios** que suele facilitar la capa de transporte:
  1. Multiplexación/demul-tiplexación (extensión de la entrega host-host, a proceso-proceso)
  2. Transferencia fiable y/o ordenada de datos
  3. 3.1 Control de flujo (saturación conexión)

     3.2 Control de congestión (saturación general)
- Aprender las implementaciones **concretas** de protocolos de la capa de transporte **en Internet**:
  - **UDP**: transporte no orientado a la conexión y no fiable (1)
  - **TCP**: transporte orientado a la conexión, fiable (1,2,3)

Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

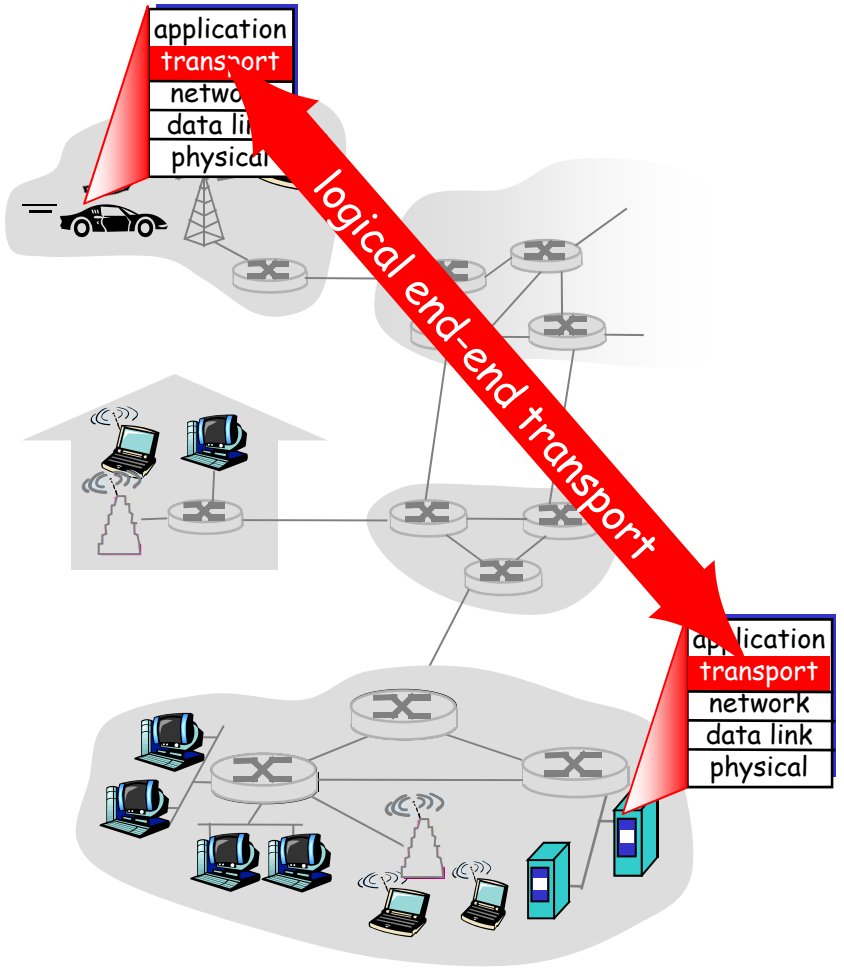
- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

Servicios

- Facilita comunicación lógica entre procesos (**aplicaciones**) corriendo en diferentes hosts
- Los protocolos de transporte se ejecutan en los sistemas finales
  - Lado emisor: Trocea/Agrega* **mensajes** en **segmentos**, que pasa a la capa de red
  - Lado receptor: reensambla* **segmentos** que pasa a la capa de aplicación
- Puede haber más de un protocolo de transporte disponible para las aplics.
  - Internet: TCP y UDP

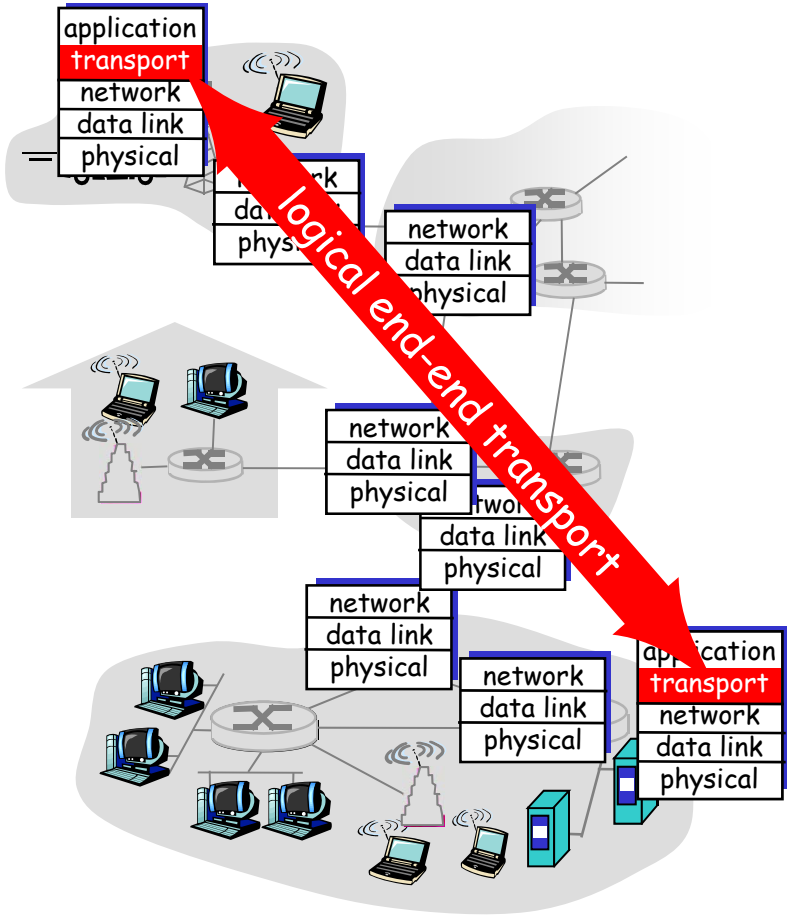


Capas: Transporte vs. Red

- *Capa de red:* Comunicación lógica entre hosts no adyacentes
- *Capa de transporte:* Comunicación lógica entre procesos
  - Basada en servicios que facilita esta capa

Protocolos de nivel de transporte en Internet

- Confiable, recepción en orden (TCP)
  - Control de flujo
  - Control de congestión
  - Requiere setup/iniciali.
- No confiable, recepción desordenada: UDP
  - Extensión del tráfico "best-effort" de IP
- Servicios no implementados:
  - Garantía de retardos
  - Garantía de ancho de banda



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

Multiplexación / demultiplexación

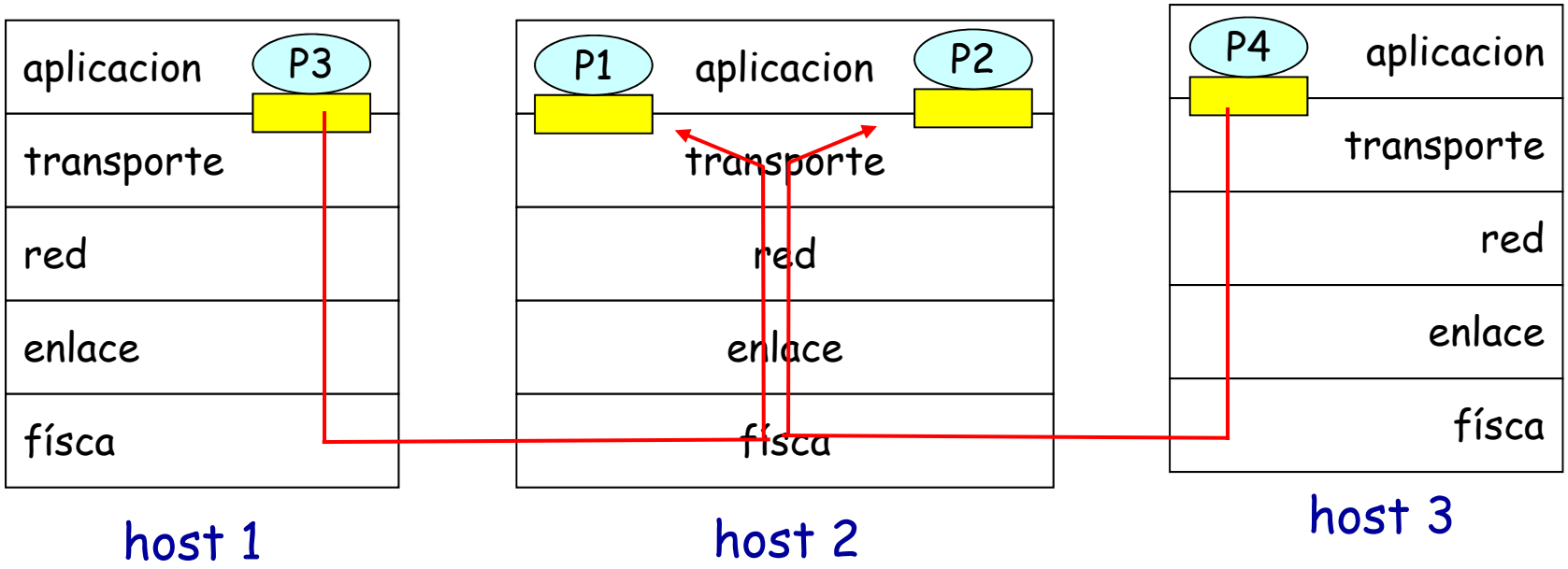
Demultiplexación en host receptor:

Distribuir los segmentos recibidos al socket correcto

Multiplexación en host que envía:

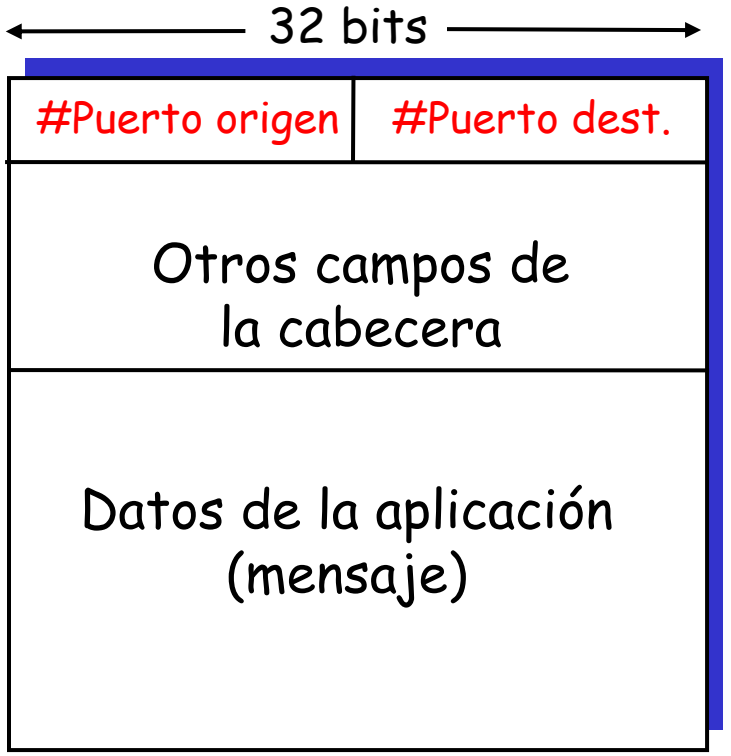
Encapsular el tráfico convenientemente y enviarlo al nivel de red

= socket = proceso



Demultiplexación

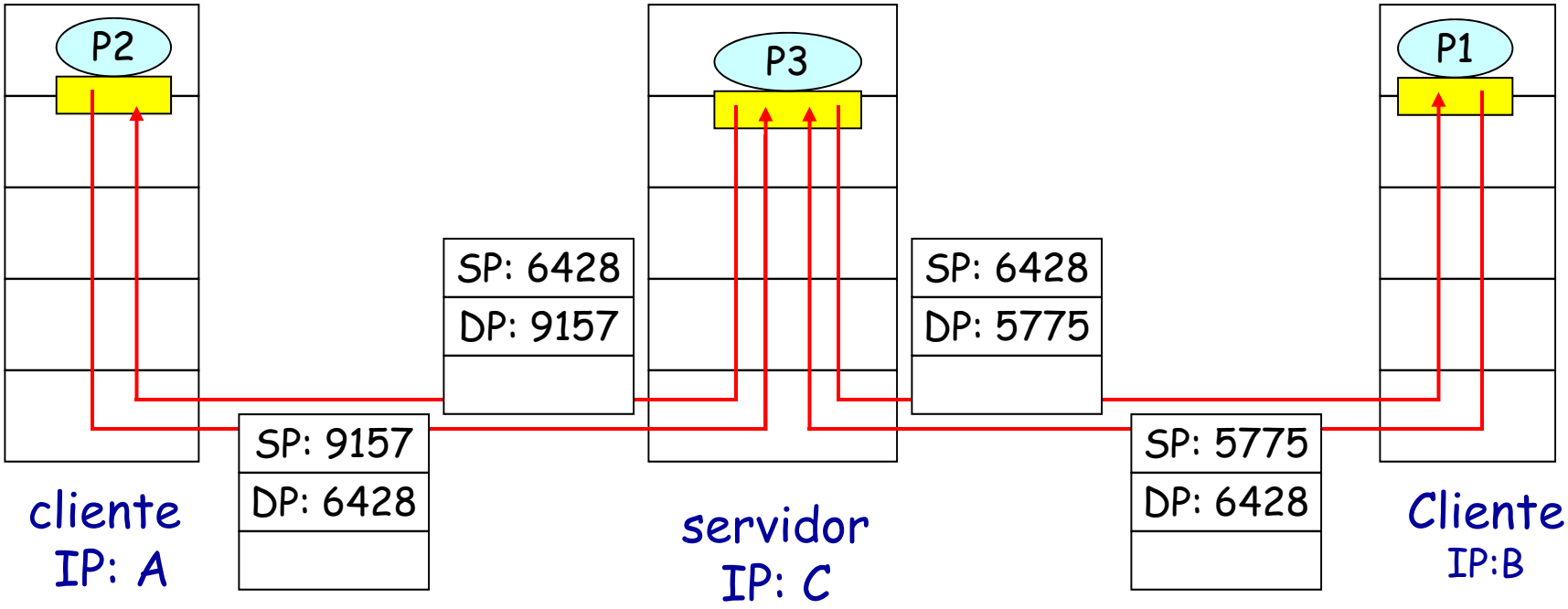
- El host recibe un datagrama IP
  - Recordemos, que cada datagrama tiene direcciones IP (origen y destino)
  - Cada datagrama transporta 1 segmento de nivel 4 (==transporte)
  - Cada segmento tiene un campo denominado puerto origen y destino
- En general es posible usar las direcciones IP y (**números**) de puerto para identificar un socket (aunque UDP modelo más simple)



Formato de segmentos TCP/UDP

Demultiplexación sin conexión

```
DatagramSocket serverSocket = new DatagramSocket(6428);
  //server C
```

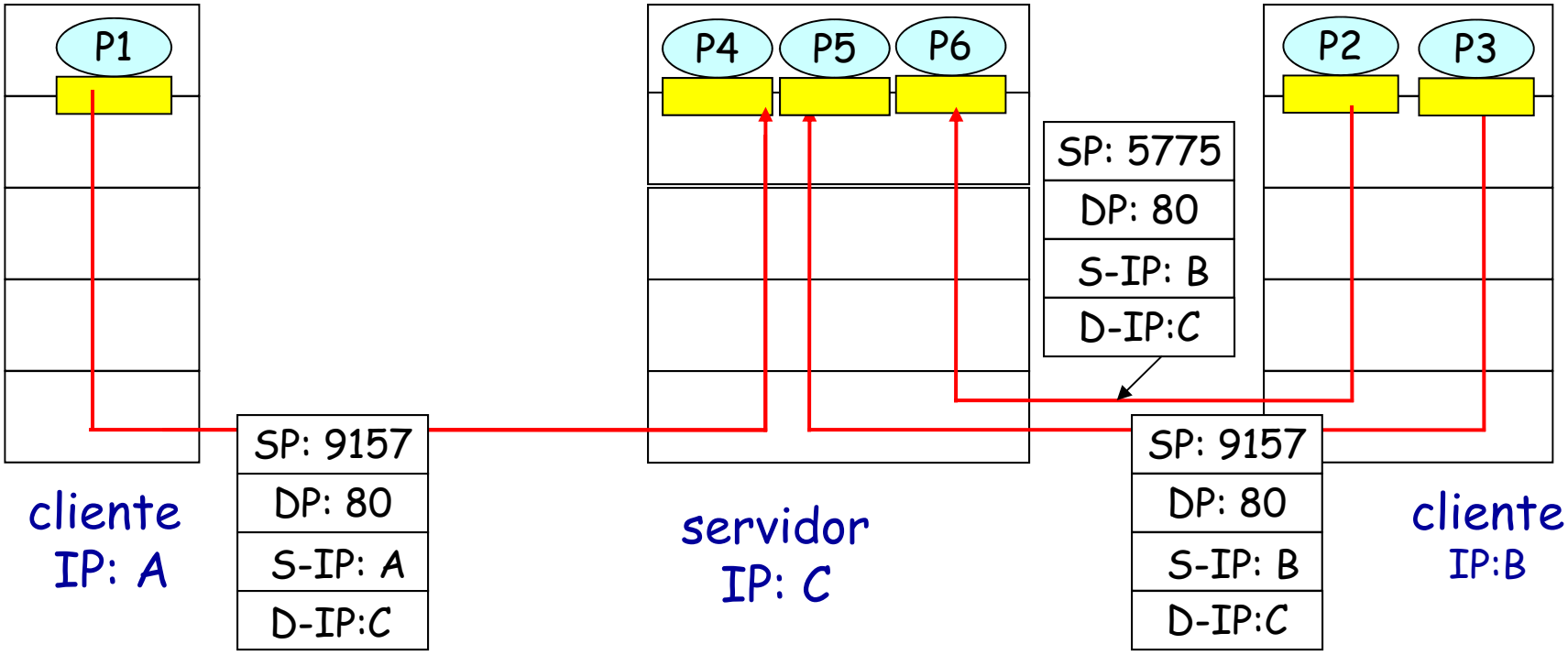


SP en envío solo sirve como identificación del socket de retorno

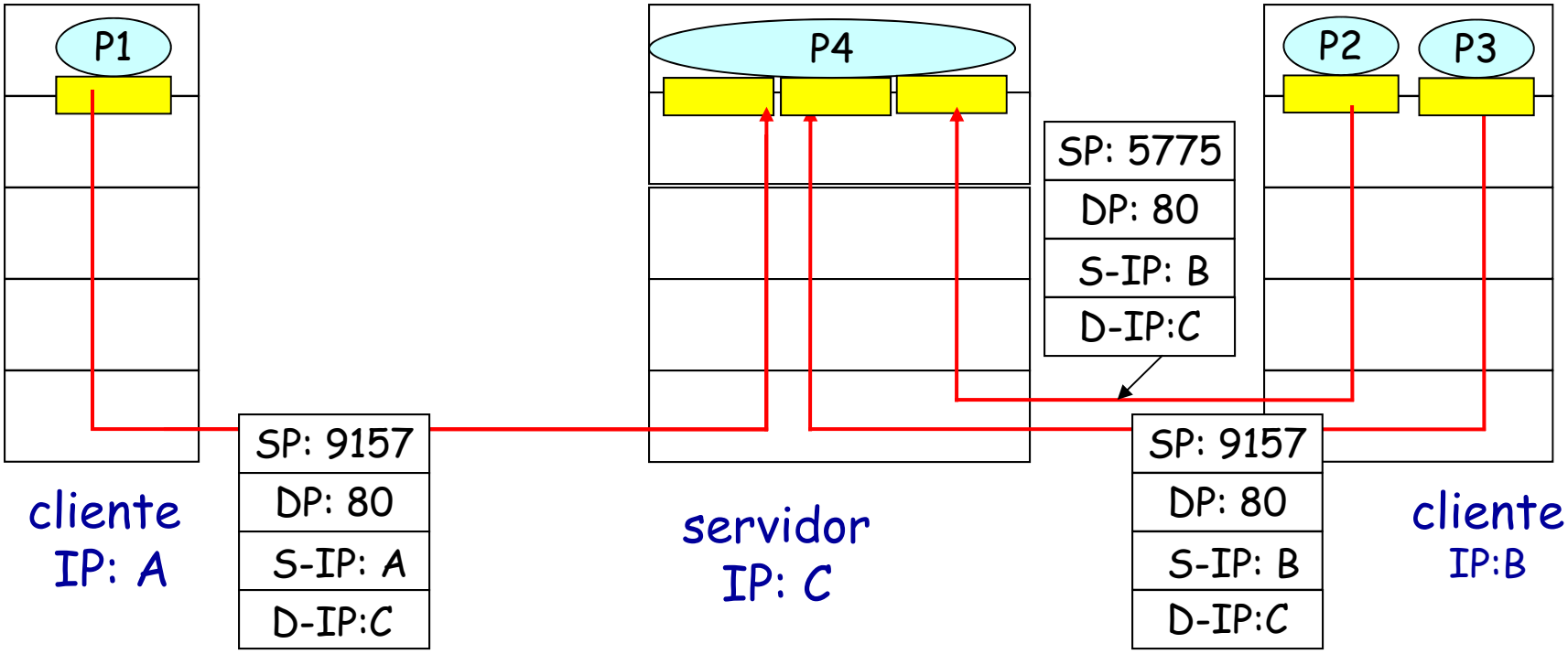
Demux orientado a la conexión

- Un socket TCP queda identificado, en cambio por 4-tupla:
  - Dirección IP origen
  - Dirección IP destino
  - Número de Puerto origen
  - Número de Puerto dest.
- Esto es, el host receptor usa estos 4 valores para direccionar al socket correcto
- Los servidores por tanto pueden soportar muchos sockets TCP concurrentemente:
  - Cada socket identificado por su propia 4-tupla
- Los servidores web, e.g., pueden tener un socket para cada conexión de cada cliente
  - Por ejemplo, HTTP no-persistente creará una socket para cada petición

Demux orientado a la conexión



Demux orientado a la conexión: Web Server usando hilos



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

UDP: User Datagram Protocol [RFC 768]

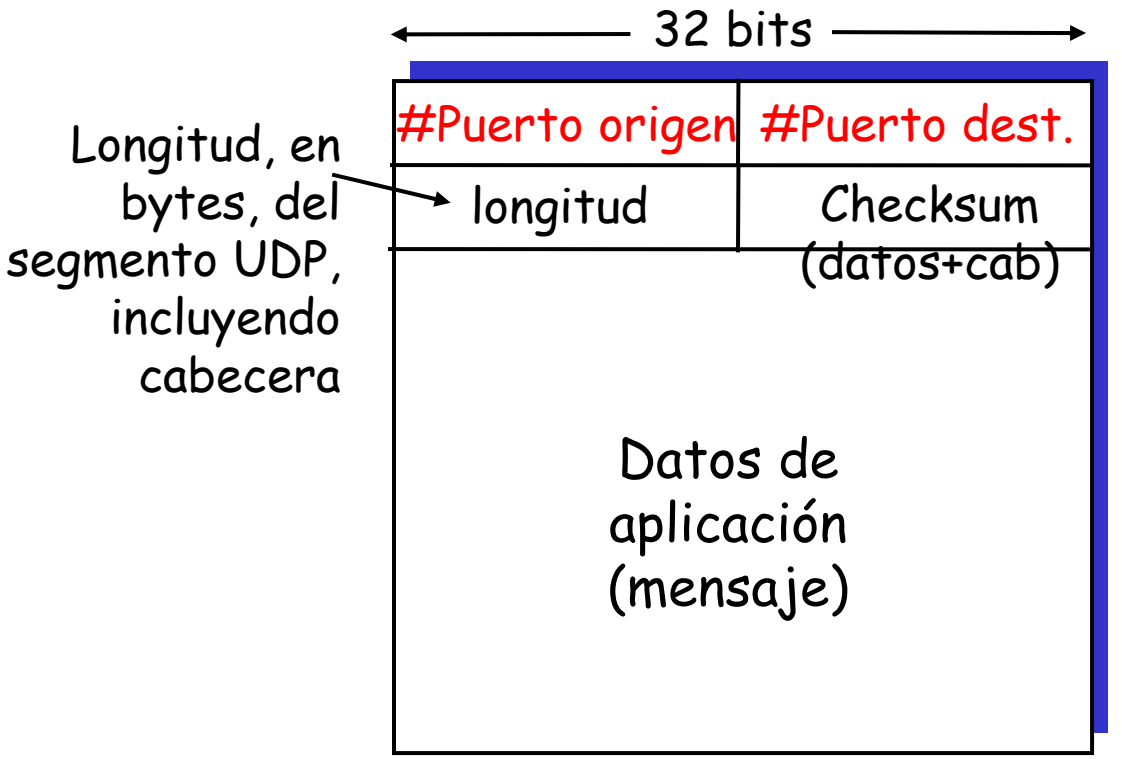
- Servicio "Best effort", los segmentos UDP pueden ser:
  - Perdidos
  - Segmentos/Mensajes entregados fuera de orden a la aplicación
- *Sin conexión:*
  - No establecimiento de la conexión emisor / receptor
  - Cada segmento UDP es manejado de manera **independiente** a los otros (flujo, congestión!) y como una **entidad**

¿Por qué UDP?

- El establecimiento de la conexión añade retardo
- Cabecera más pequeña
- Cuando el retardo supera en importancia a las perdidas
  - Los servicios adicionales de TCP añaden retardo
- Sin control de flujo/congestión: UDP puede transmitir a la tasa que quiera ¿?
  - Posible implementación con-fiabilidad en capa superior

UDP: cabecera

- Usos:
  - Aplicaciones multimedia
    - Tolerantes a las perdidas
    - Sensibles a la caudal/latencia
  - DNS
  - SNMP (monitoriz.)



Formato de un segmento UDP

Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

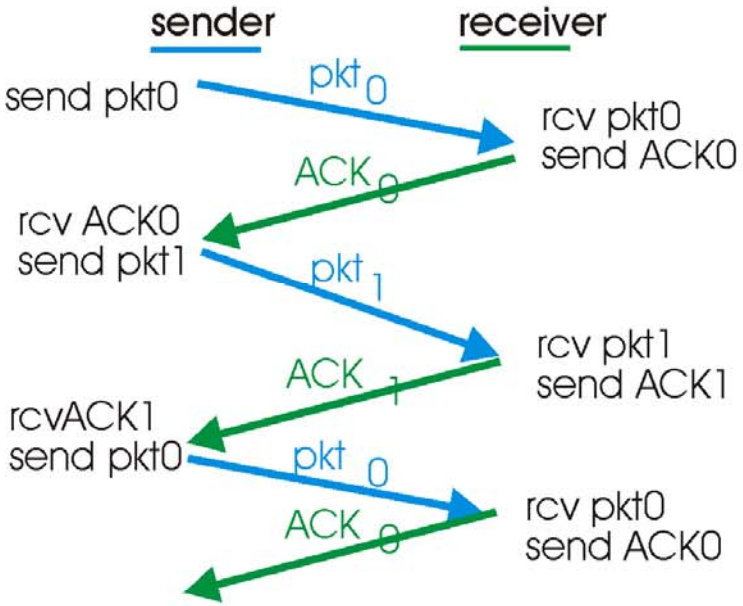
3.6 Principios de control de congestión

3.7 Control de congestión en TCP

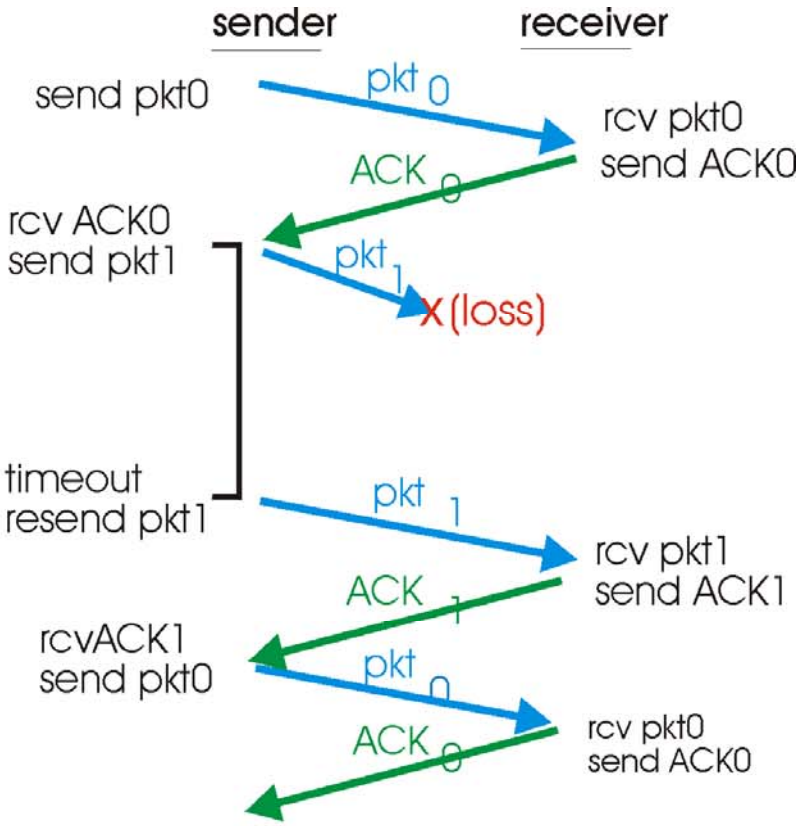
Transferencia datos fiable

- Elementos necesarios:
  - Detectar errores → *Checksum* (sumas de comprobación)
  - Detectar perdidas → Temporizadores
  - Detectar perdidas → Numerar elementos
    - Número de secuencia
    - Reconocimientos
      - » positivos (ACKS) y negativos (NACKS)

Parada y espera

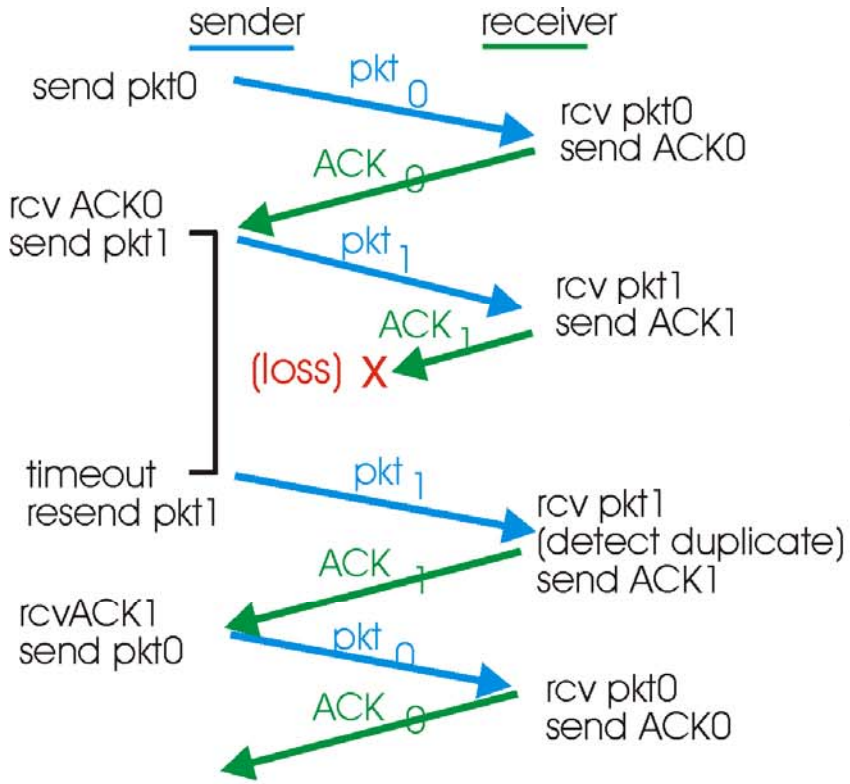


(a) operation with no loss

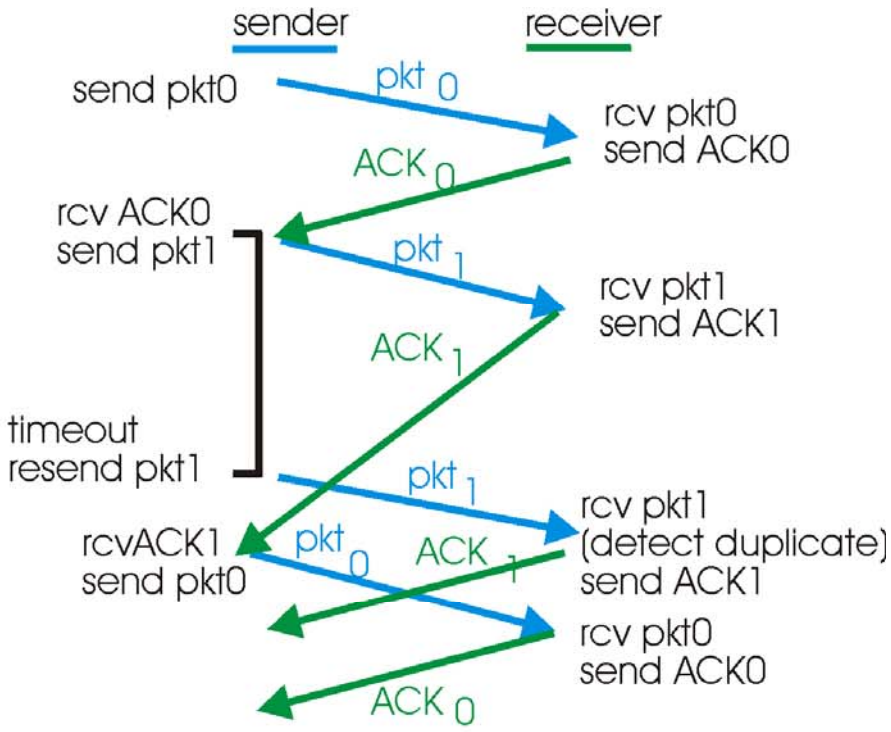


(b) lost packet

Parada y espera



(c) lost ACK



(d) premature timeout

Parada y espera

- Parada y espera funciona, pero su rendimiento es pobre
- E.g.: Enlace de 1 Gb/s, 15 ms tiempo de propagación, paquetes de 1000 Bytes:

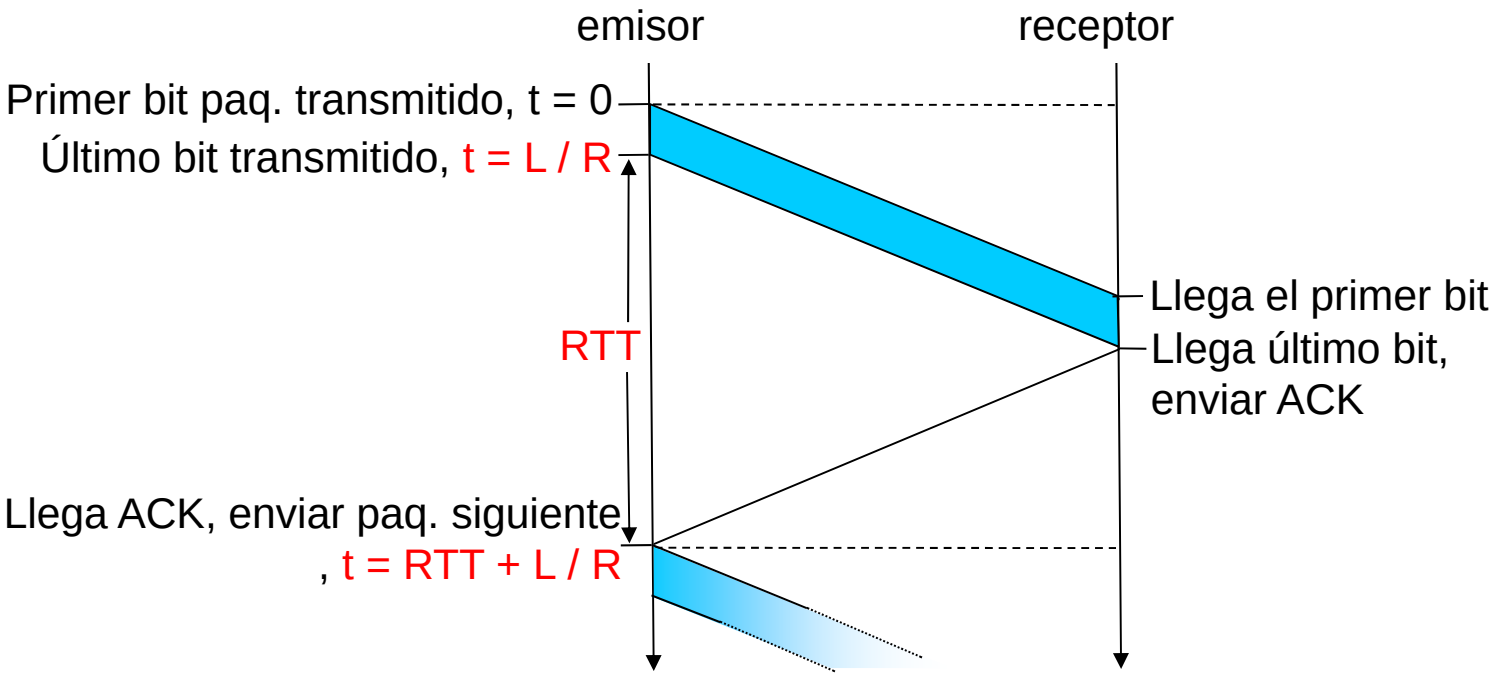
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\,\text{b/s}} = 8\,\text{microsegundos}$$

  - U_{emisor}: utilización$_{emisor}$: fracción de tiempo en el que el emisor emite

$$U_{emisor} = \frac{L / R}{RTT + L / R} = \frac{0.008}{30.008} = 0.00027$$

  - Lo que resulta en un throughput de 267 kb/s en un enlace de 1 Gb/s!
  - El protocolo de transporte limita el uso de los recursos de la red!

Parada y espera

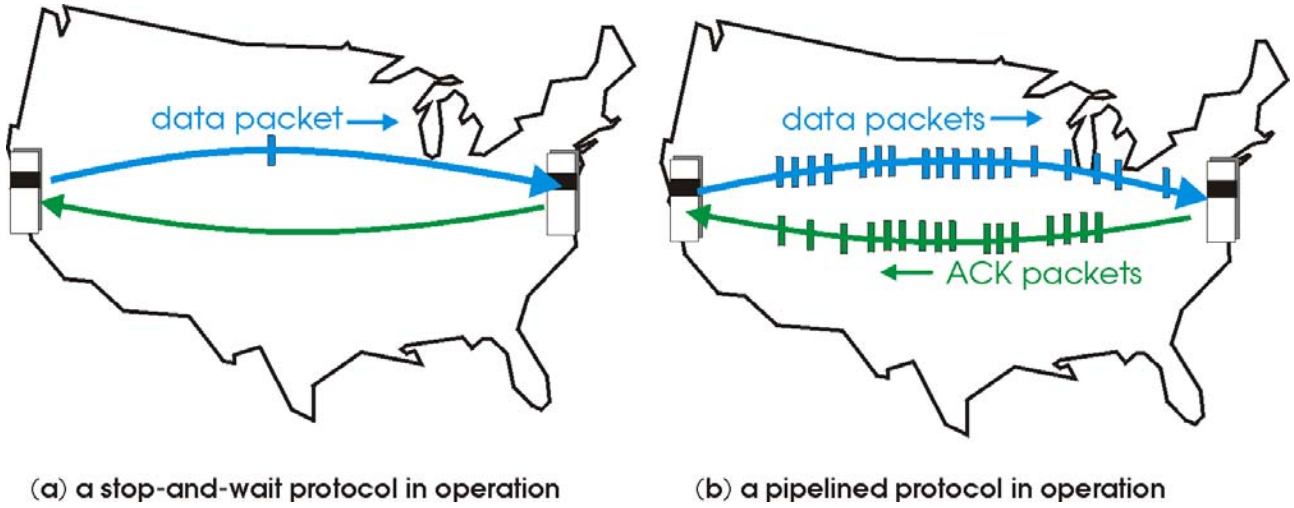


$$U_{emisor} = \frac{L / R}{RTT + L / R} = \frac{0.008}{30.008} = 0.00027$$

Protocolos Pipelined (procesamiento en cadena)

El emisor permite múltiples envíos, "al vuelo", antes de llegar los correspondientes ACK

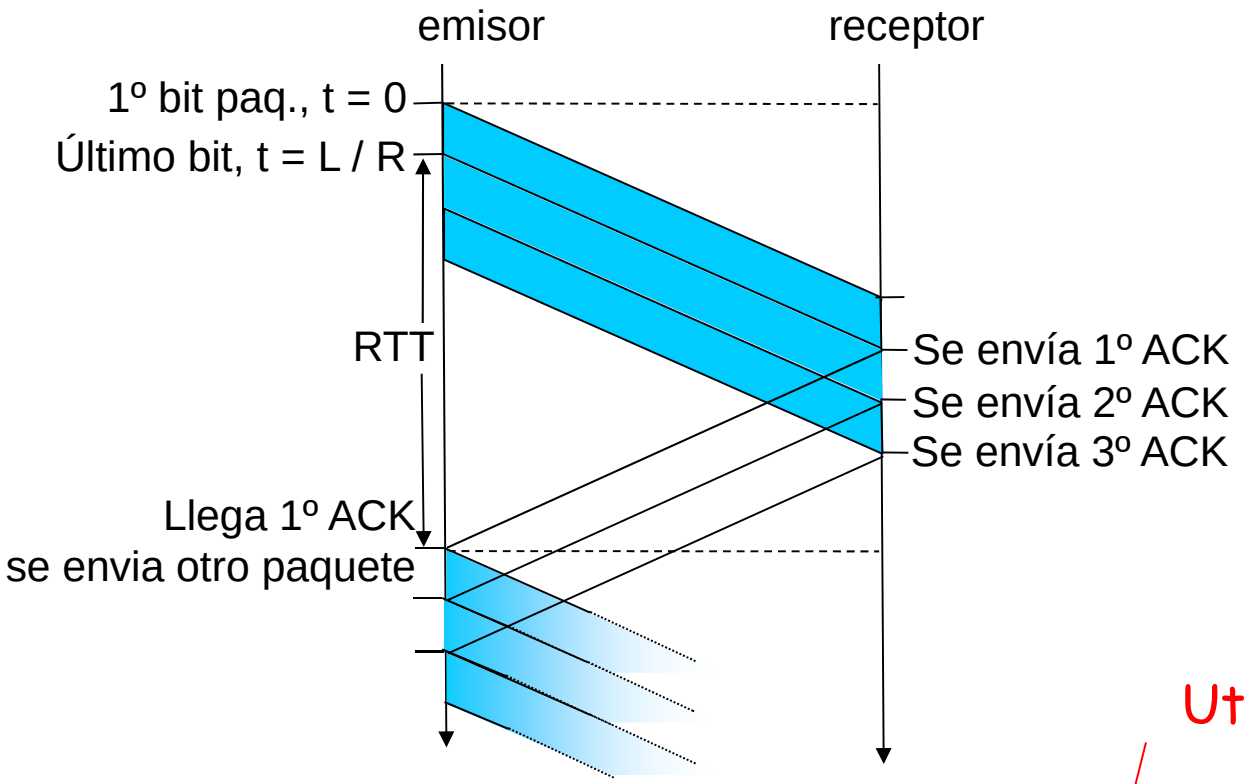
- El número de secuencia debe abarcar más bits incrementarse (== ventana deslizante)
- Se requiere buffer en emisor y/o receptor



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Formas genéricas de implementar estos protocolos : *go-Back-N, repetición selectiva…*

Pipelining: incremento de la utilización



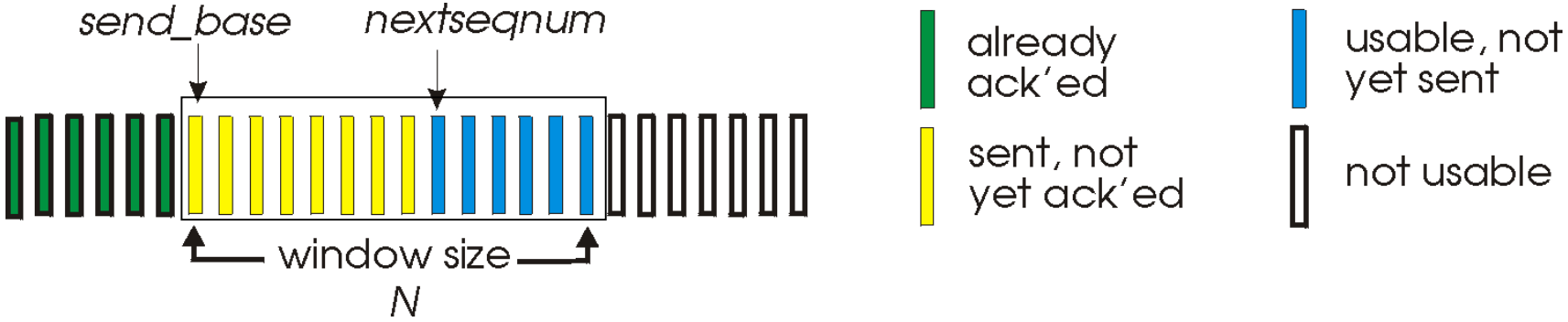
Utilización x 3!

$$U_{emisor} = \frac{3 * L / R}{RTT + L / R} = \frac{0.024}{30.008} = 0.0008$$

Ventana deslizante (GBN*)

Emisor:

- Número de secuencia están limitados
- El tamaño de la "ventana" [deslizante] hasta un máximo de N segmentos* no confirmados

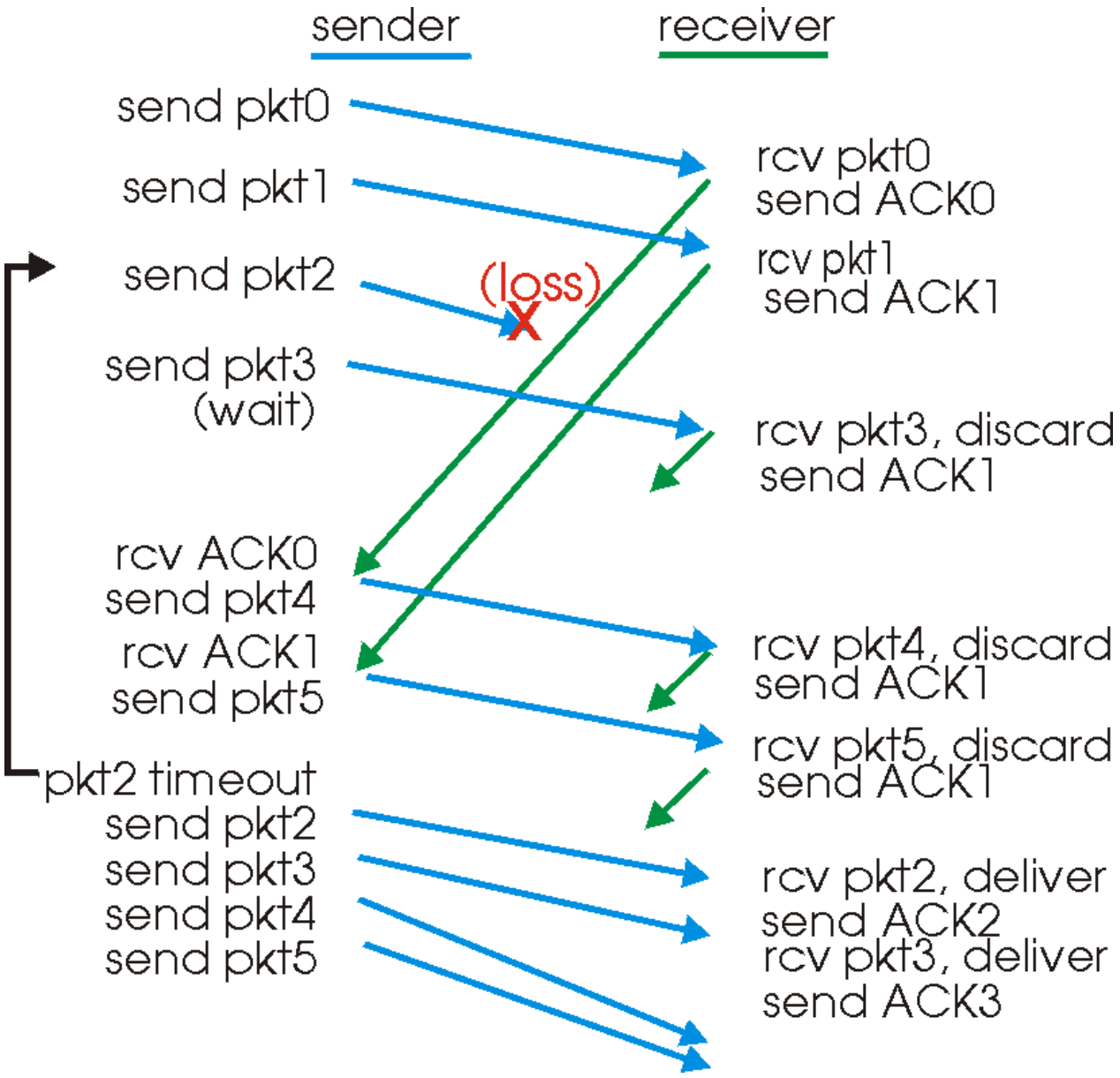


- **ACK(n) es acumulativo**: se confirman todos los paquetes con número de secuencia menor o igual a n (buffer recepción igual a 1 → intuitivo)
- **El extremo receptor en caso de perdidas/desorden devuelve la confirmación del último segmento recibido en secuencia**
- **Cada paquete recibido es confirmado por su respectivo ACK**

GBN en acción

En el ejemplo:
- Tamaño de ventana=?
- |Números de secuencia| -1

Supongamos 2 bits para el número de secuencia (00,01,10,11),
- Cual es el tamaño máximo de la ventana en segmentos?



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

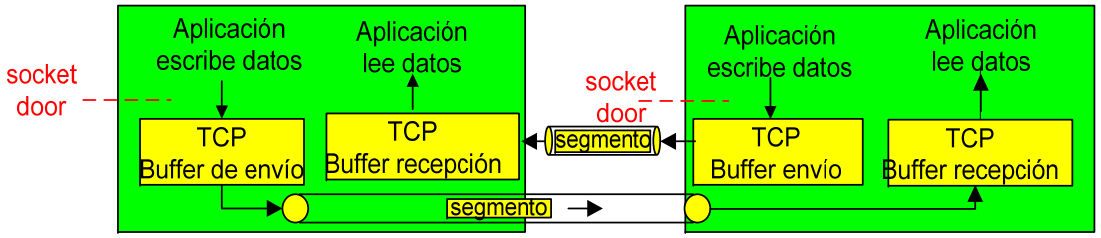
- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

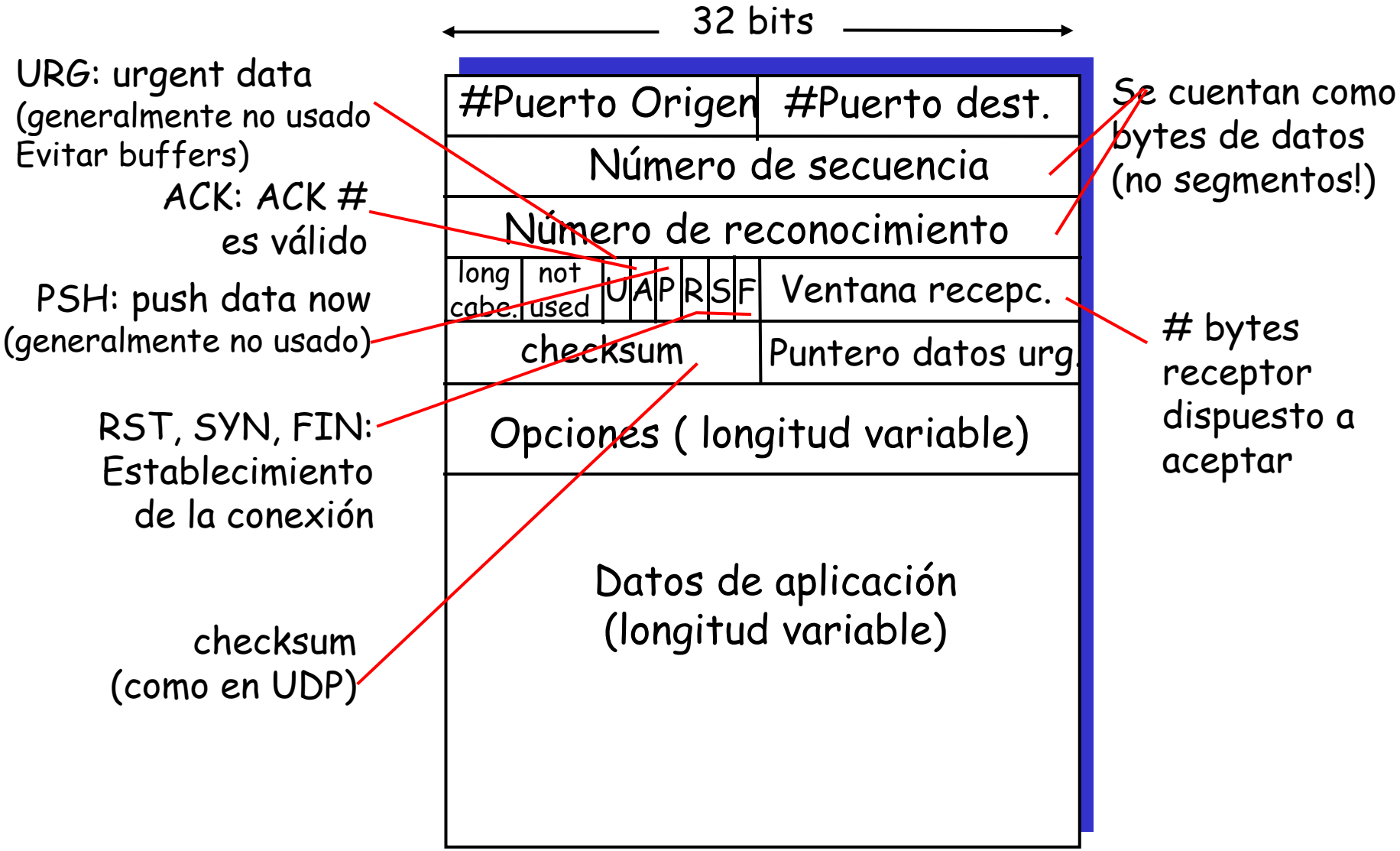
3.7 Control de congestión en TCP

TCP RFCs: 793, 1122, 1323, 2018, 2581

- Punto-a-Punto:
  - 1 emisor, 1 receptor
- Confiable, flujo de bytes ordenados:
  - Agregación/Fragmentación de los mensajes
- Pipelined:
  - El tamaño de la ventana es fijada para control de flujo y congestión*
- *Buffers en ambos extremos*
- full duplex:
  - Flujos bidireccionales en la misma conexión
  - MSS: maximum segment size [cantidad máxima de datos de la capa de aplicación en el segmento, ¿lógico?¿coherente?]
- Orientado-conexión:
  - Handshaking (intercambio de mensajes de control, acuerdo en tres fases) para inicialización



Estructura del segmento TCP



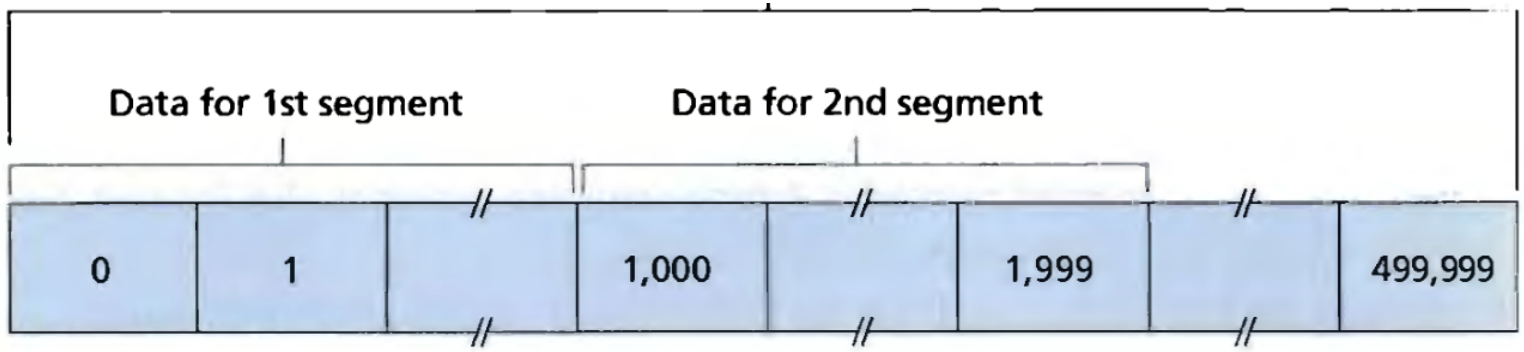
Banderas TCP

- Flags:
  - URG: El puntero urgente es válido (adelantar buffer)
  - ACK: El número de asentimiento es válido
  - PSH: El receptor/emisor debe pasar los datos tan rápido como pueda a la aplicación/nivel inf.
  - RST: Reset de la conexión
  - SYN: Sincronizar los números de secuencia
  - FIN: El emisor no tiene más datos que mandar

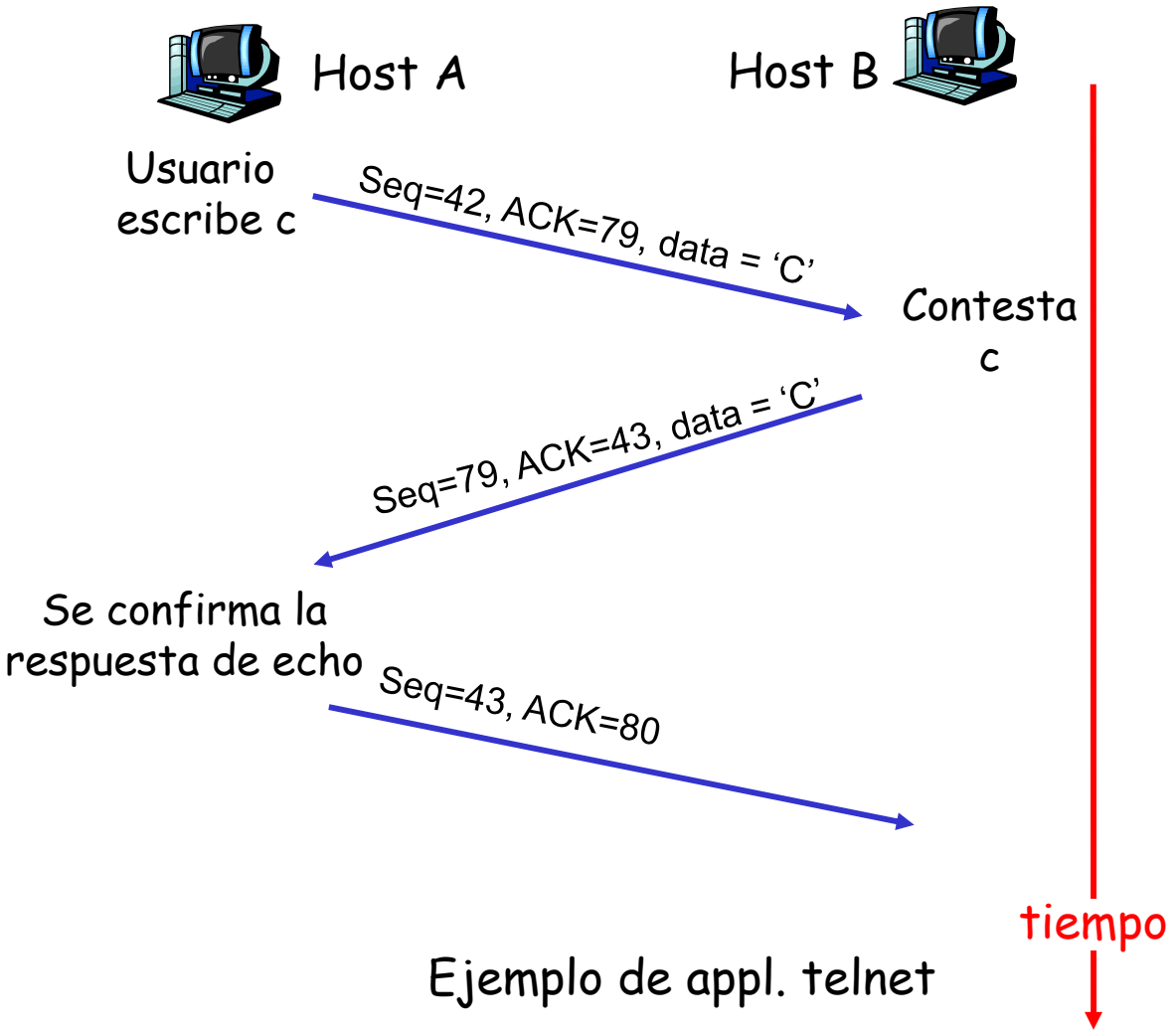
Números de secuencia y de reconocimiento TCP

- Los número de secuencia/reconocimiento hacen referencia al flujo de bytes transmitido y no a la serie de segmentos transmitidos
- El número de secuencia de un segmento es el número del primer bytes del segmento dentro del flujo de bytes
- Se inicializa al azar

- E.g., fichero de 500.000 bytes, MSS 1000 bytes, se considera que los número de secuencia empiezan de forma aleatoria por el 0
- 1º segmento → 0, 2º segmento → el 1000, el 3º → 2000, …



- Número de secuencia:
  - Bytes ya enviados anteriormente* en ese extremo
- Número de reconocimiento (ACK)
  - Bytes recibidos hasta ese momento en ese extremo
- Implementa reconocimiento acumulativos



Ejemplo de appl. telnet

TCP Round Trip Time y Timeout

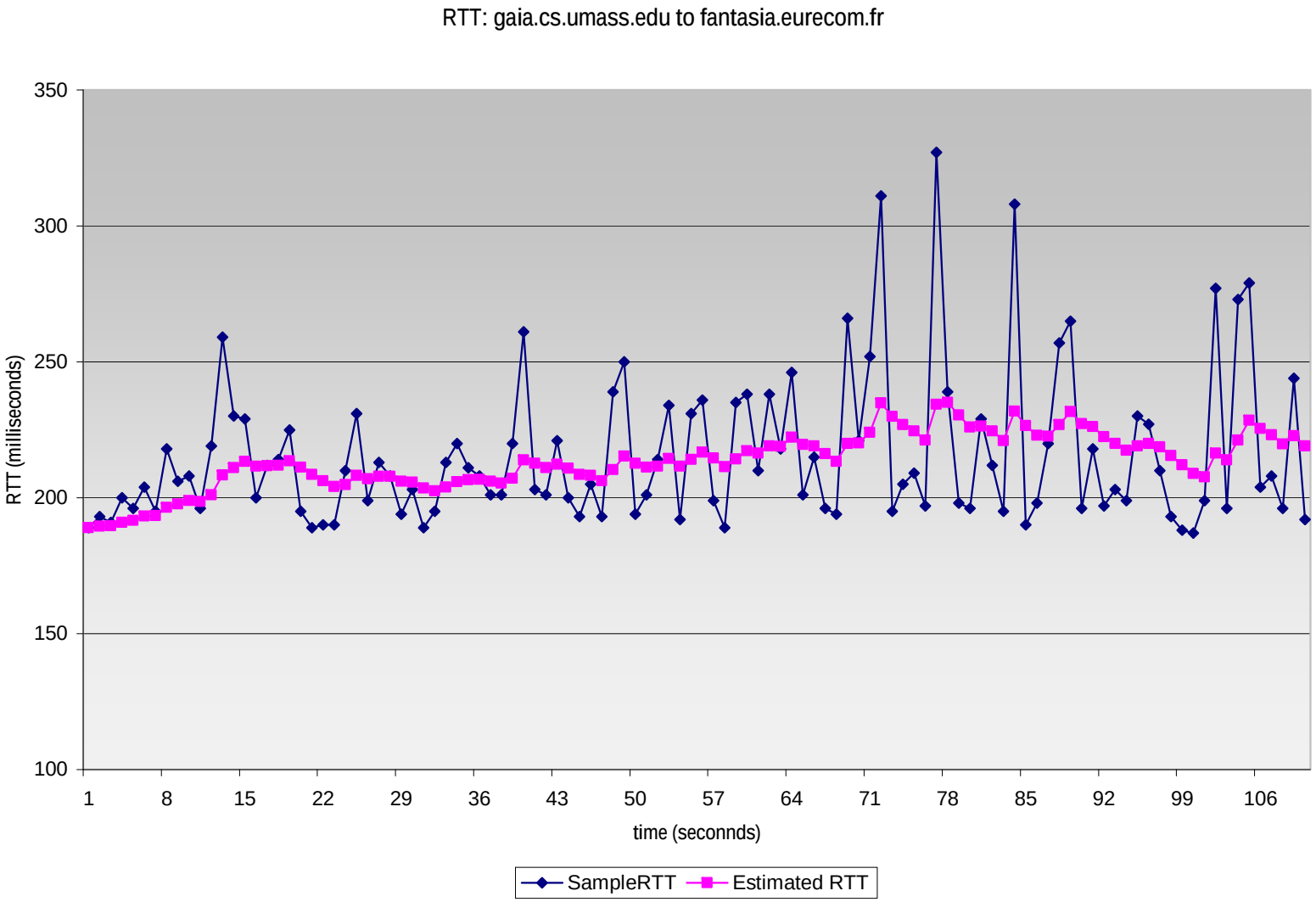
Q: ¿Como fijar el valor para el valor de timeout TCP?

- Mayor que el RTT
  - Pero el RTT varía
- Demasiado pequeño: timeout prematuro
  - Retransmisiones innecesarias
- Demasiado largo:
  - Lenta reacción a perdida de segmentos

Q: Como estimar el RTT?

- **RTTMuestra**: medir el tiempo desde la transmisión de un segmento hasta que se recibe el ACK (solo una medida a la vez)
  - Se ignoran retransmisiones
- **RTTMuestra** variará, de modo que es más adecuado usar una medida más suave
  - Se aplica un modelo pro-mediador

Ejemplo RTT estimación:



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

TCP transferencia de datos fiable

- TCP da una servicio de transferencia de datos **fiable** sobre un canal que no lo es
- TCP usa un **único temporizador** de transmisión (salvo que se especifique lo contrario)
- Las **retransmisiones** suceden cuando:
  - Se cumple el temporizador
  - Reconocimientos duplicados

Eventos TCP :

Aplicación envía datos:

- TCP crea un segmento y transmite el segmento
- Arranca un temporizador si es que no estuviera ya en marcha (esto es, se cuenta con respecto al segmento más viejo no confirmado)

Se recibe segmento fuera de secuencia:

- El extremo receptor devuelve la confirmación del último segmento confirmado
- Si el número de secuencia es mayor al esperado se guarda el segmento

Se cumple el temporizador:

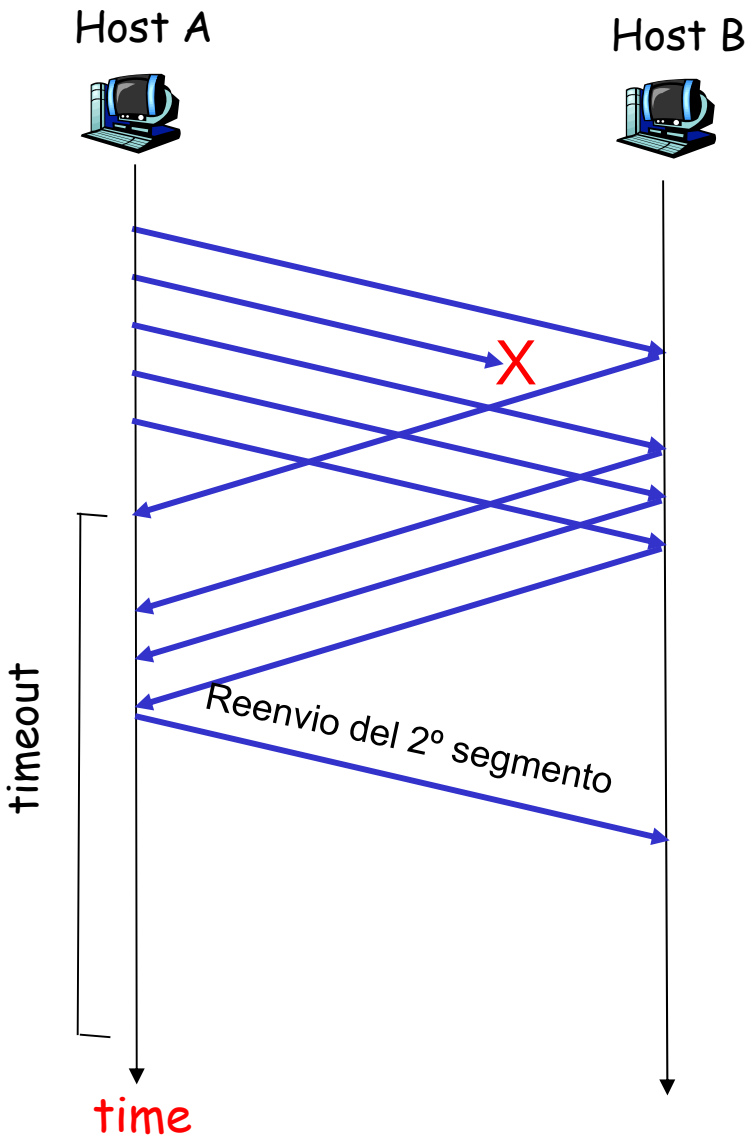
- Se retransmite el segmento que ha ocasionado el fin del temporizador
- Se reinicializa el temporizador

Se recibe ACK:

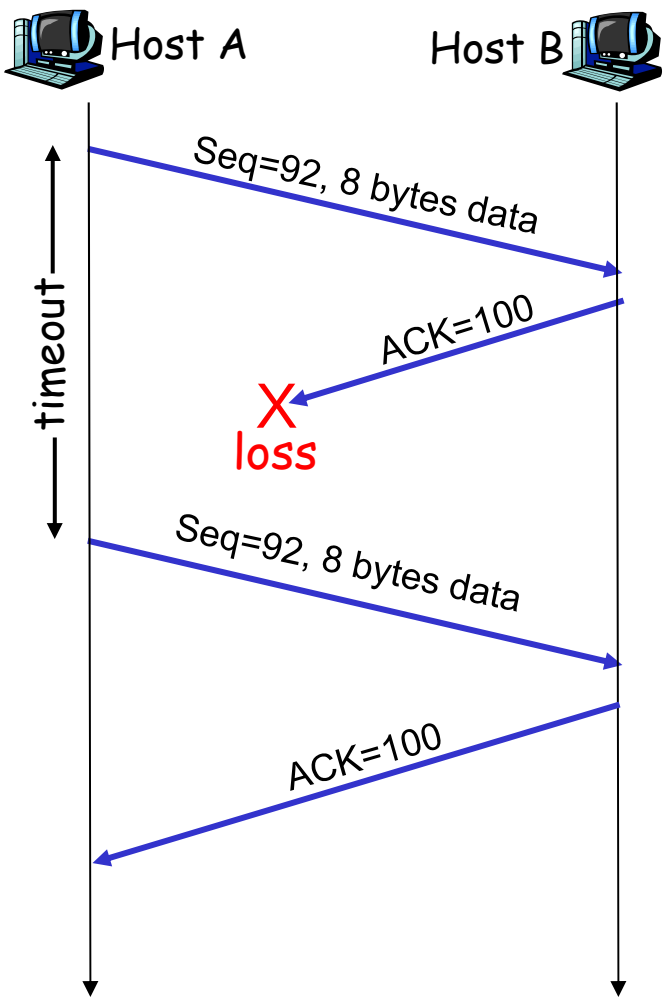
- Si confirma segmentos previos
  - Se actualiza los segmentos confirmados
  - Si quedan segmentos por confirmar, se arranca el temporizador

Retransmisión rápida

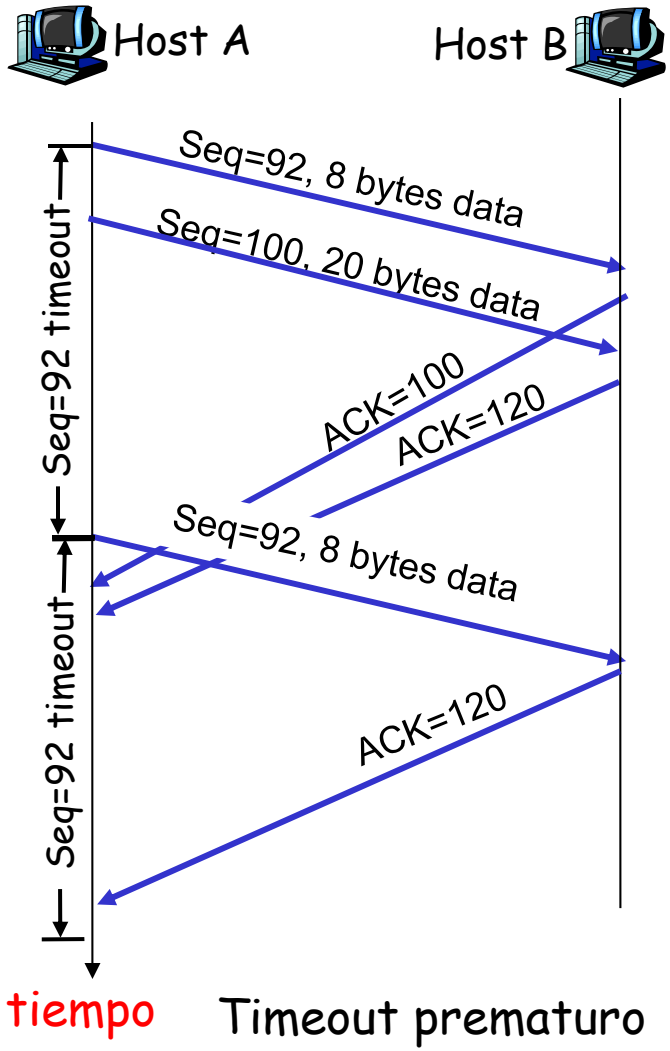
- Los temporizadores pueden resultar largos:
  - Hay que esperar al temporizador y, entonces, reenviar… pero, a veces, es fácil detectar que un segmento no llegó (sin NACKs)
- Por ejemplo, detectando ACKs duplicados:
  - El emisor envía una serie de segmentos
  - Si se pierde un segmento, el receptor le "recuerda" el último ACK correcto
- Si el receptor recibe 3 ACKs duplicados para los mismos datos, asume que el siguiente se ha perdido:
  - Retransmisión rápida: envía el segmento ignorando el temporizador


Host A
Host B
X
timeout
Reenvio del 2º segmento
time

TCP: escenarios de retransmisión

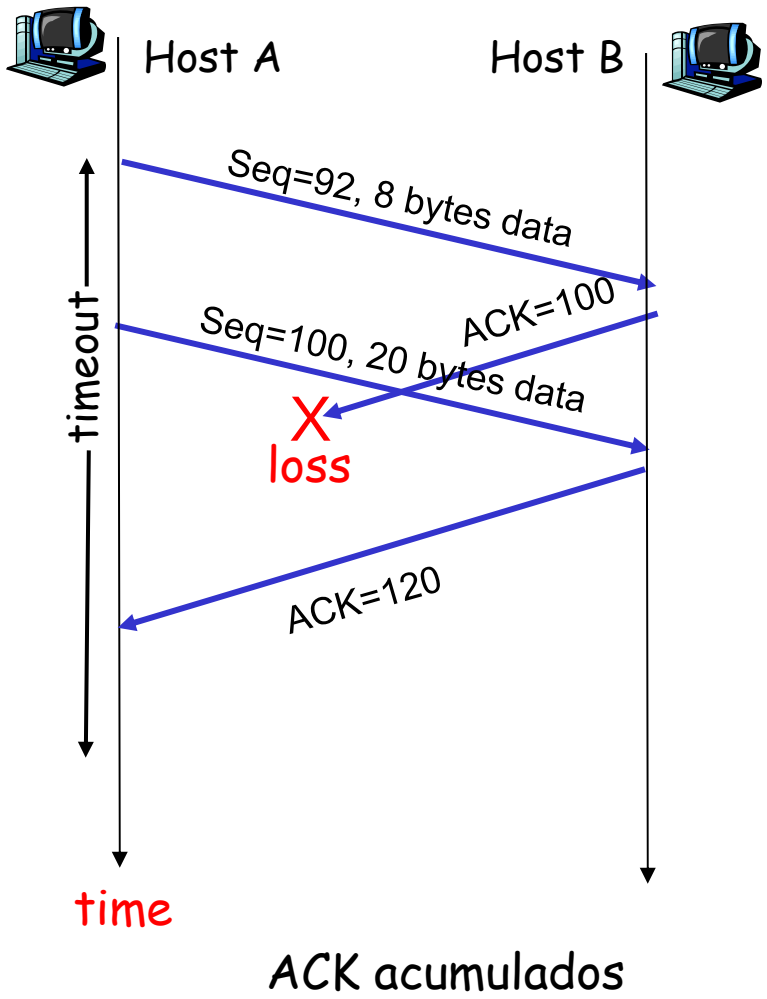


Perdida de ACK



Timeout prematuro

TCP: escenarios de retransmisión



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

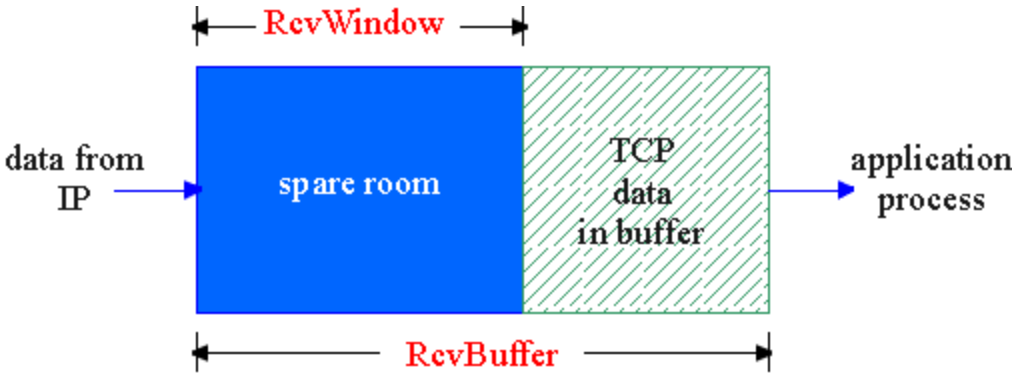
- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

TCP Control de flujo

- Cada lado receptor de TCP tiene un buffer de recepción:



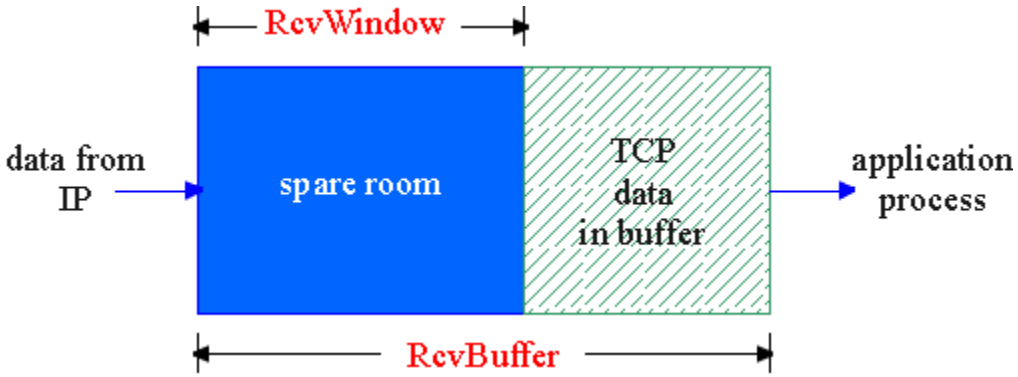
- El proceso de aplicación puede ser lento en leer del buffer (segmentos ya confirmados)

Control flujo

El emisor no debe sobrepasar el buffer del receptor al transmitir demasiado rápido

- Servicio de adaptación de la velocidad: adaptar la tasa de envío a la tasa de lectura de la aplicación

TCP control de flujo: Como funciona



(Asumimos en el dibujo que no se pierden/ordenados paquetes)

- Espacio libre en el buffer

= ventana de recepción [RcvWindow]

= Buffer de recepción - [último byte recibido - último bytes leído]

- El receptor advierte del espacio libre incluyendo el valor de la ventana de recepción en el segmento
- El emisor limita el volumen de datos enviados pero no confirmados a este tamaño de ventana
  - Se garantiza que el buffer no se desborda
  - Si la ventana es 0 puede ser necesario enviar paquetes con datos igualmente
    - Ventana tonta

Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

TCP Gestión de la conexión

Recordatorio: El emisor debe establecer la conexión antes de intercambiar segmentos con datos

- Inicializar variables TCP:
  - Número de seqs.
  - Buffers, variables de control de flujos (e.g. **`RcvWindow`**)

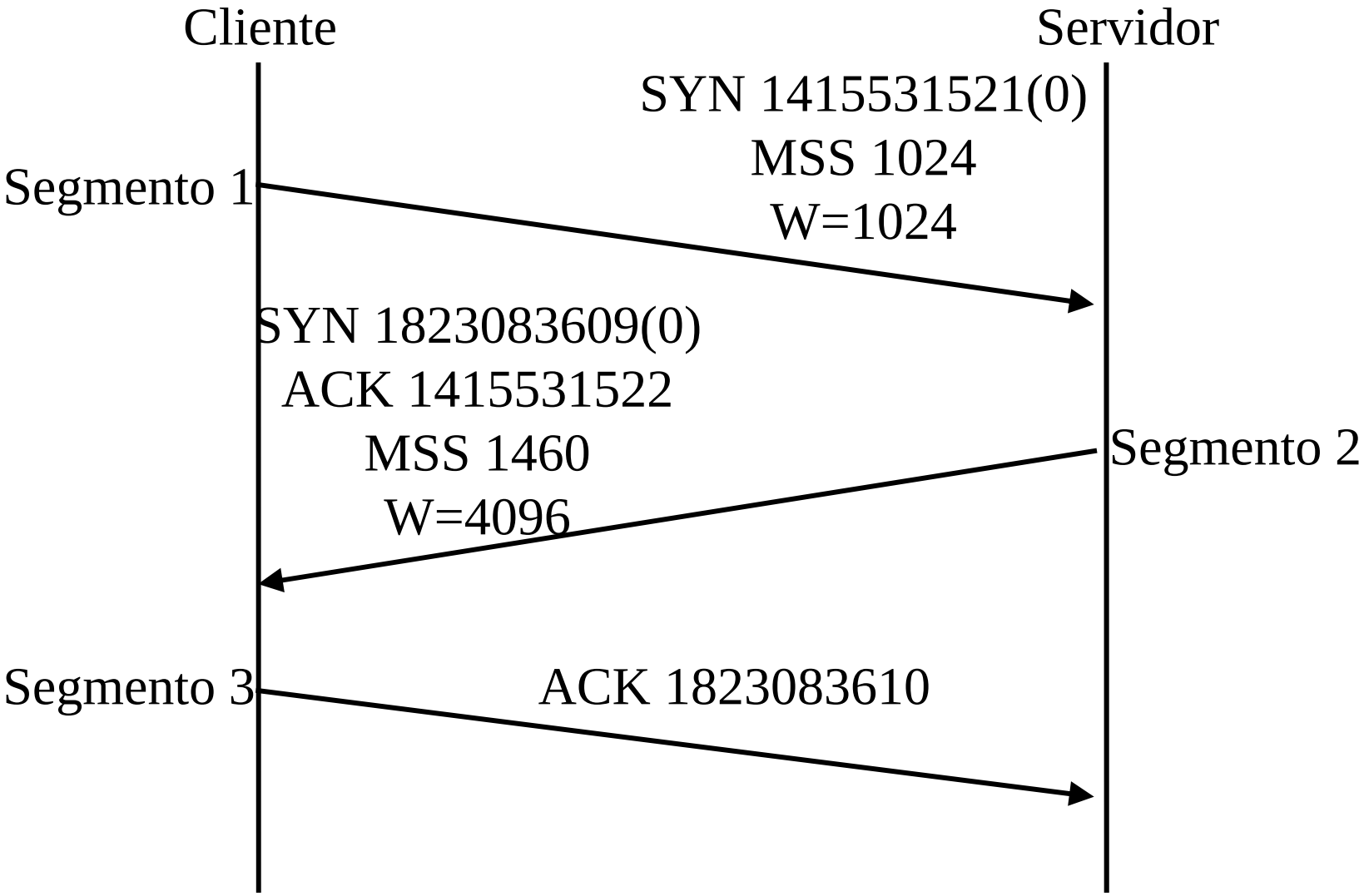
Acuerdo en 3 fases:

Paso 1: El cliente envía un segmento TCP SYN al receptor (otro extremo)
- Especifica núm. de seq. inicial
- Sin datos

Paso 2: El servidor recibe el SYN, responde con un segmento SYNACK
- Servidor reserva los buffers*
- Especifica el número de secuencia inicial de este extremo

Paso 3: El cliente recibe SYNACK, y responde con un segmento ACK, que puede ya contener datos

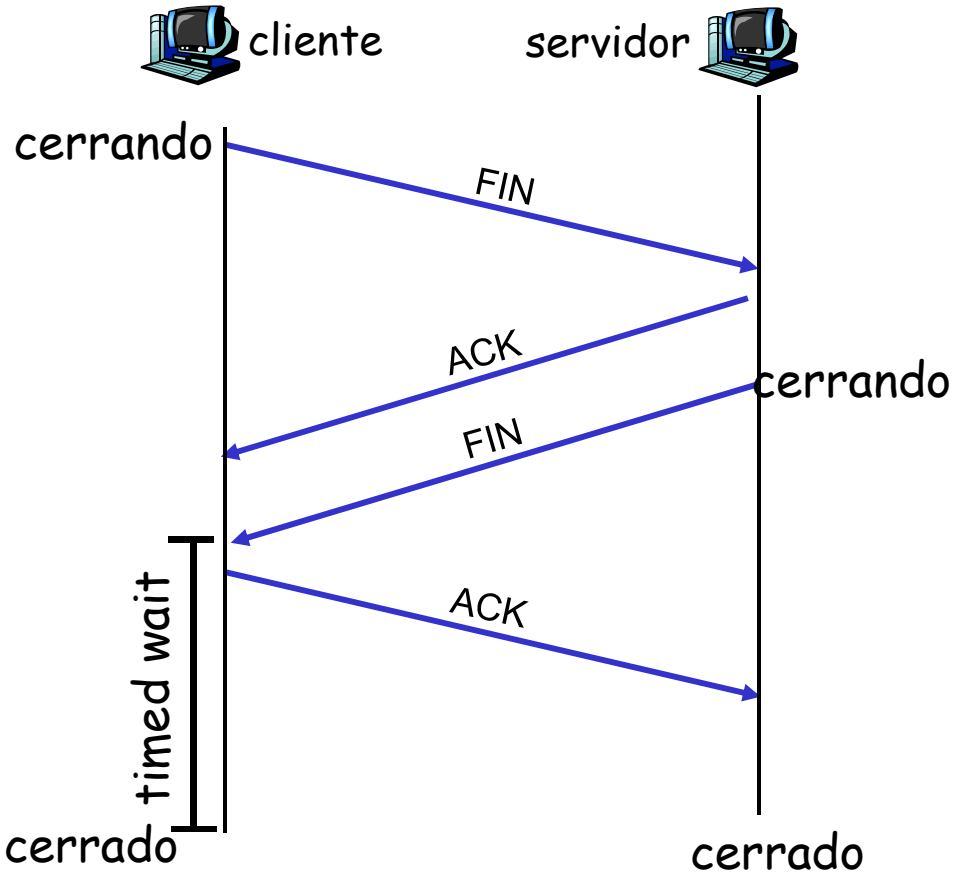
Protocolo de apertura de conexión



Protocolo de cierre de conexión

Paso 1: El cliente envía un segmento TCP con la bandera FIN a el servidor

Paso 2: El servidor recibe FIN, responde con un flag ACK. Cierra la conexión, mandando antes un FIN



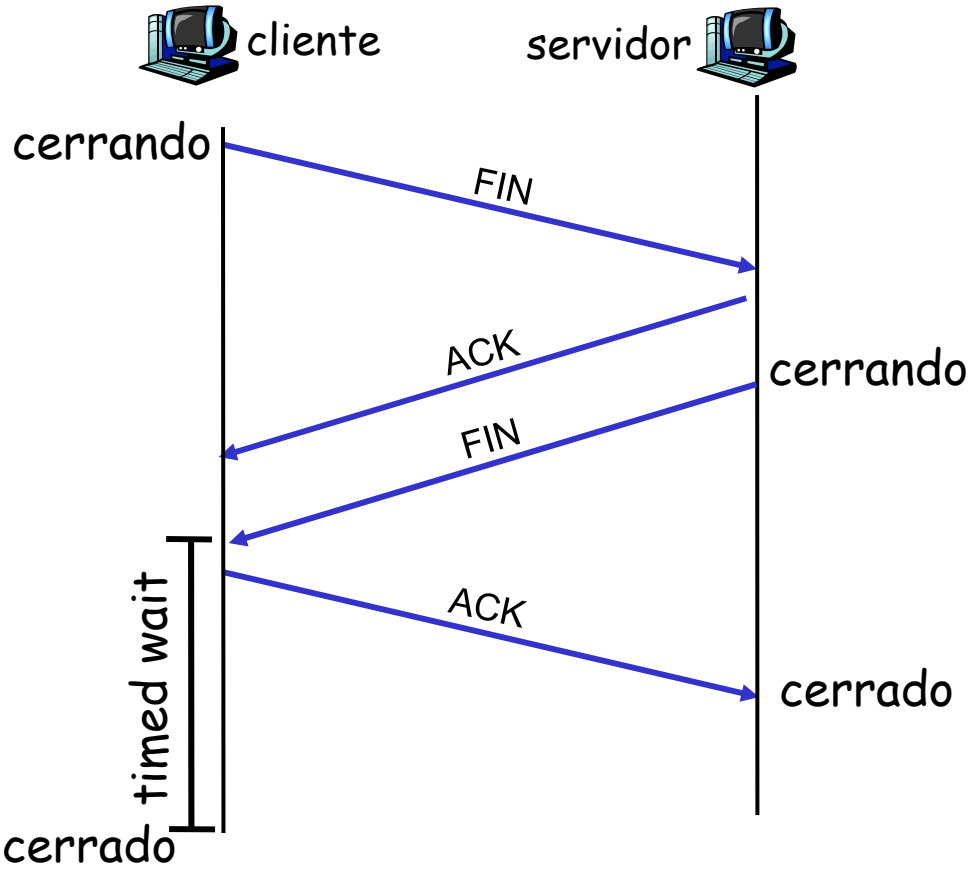
Protocolo de cierre de conexión 2

Paso 3: El cliente recibe el FIN, responde con ACK.

- Se entra en un temporizador donde se responderá con ACK a los (posibles) FINs recibidos

Paso 4: El servidor, recibe el ACK. Conexión cerrada.

Nota: pequeñas modificaciones para FINs simultáneos y casos de error.



Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

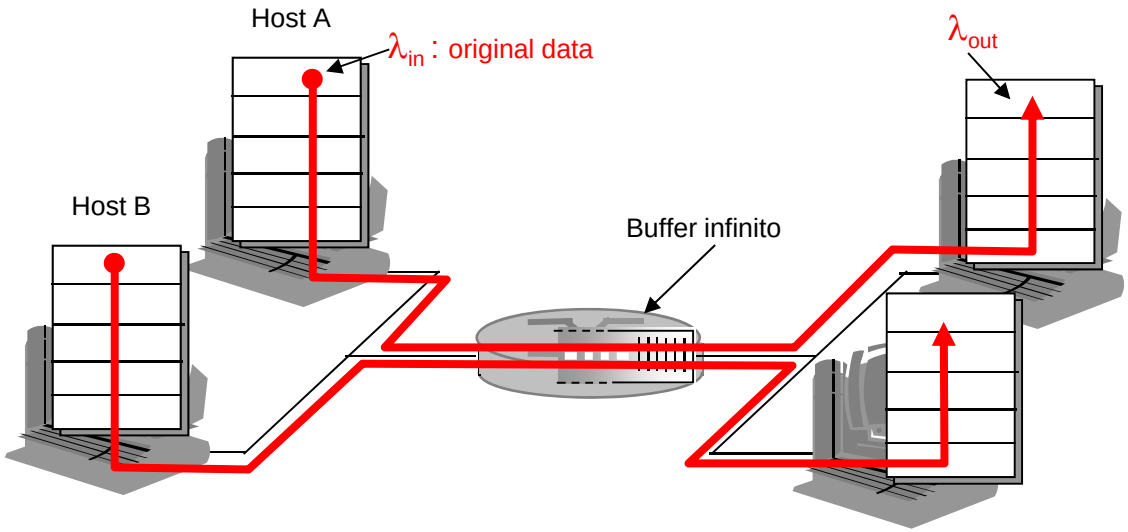
Principios del control de congestión

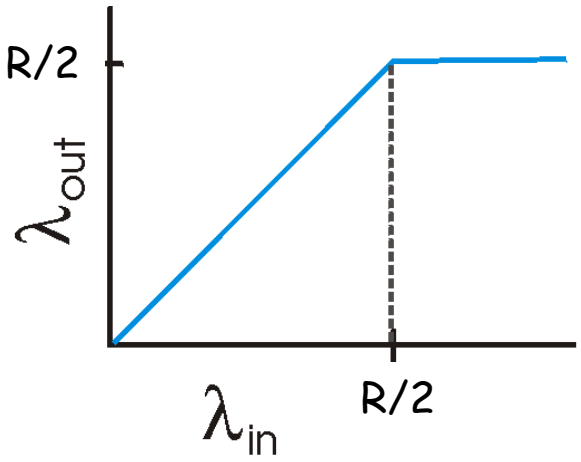
Congestión:

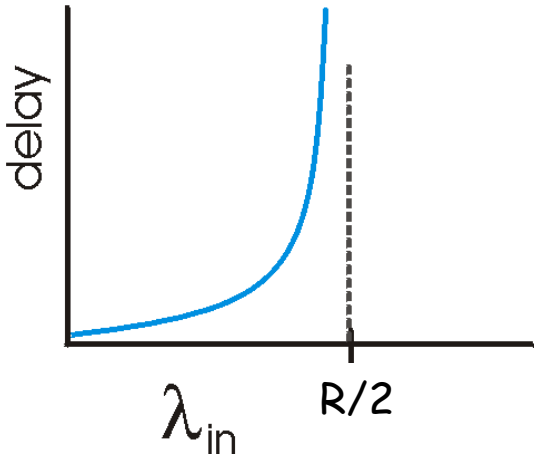
- Informalmente: "Demasiadas fuentes enviando demasiados datos/demasiado rápido para que la red pueda manejarlos"
- Es diferente al control de flujo!
- ¿Cómo nos damos cuenta?:
  - Paquetes perdidos (los buffers de los routers se desbordan)
  - Largos retardos (colas en los buffers de los routers)

Causas/costo de la congestión: escenario 1

- 2 emisores, 2 receptores
- 1 router, buffers infinitos
- Sin retransmisiones



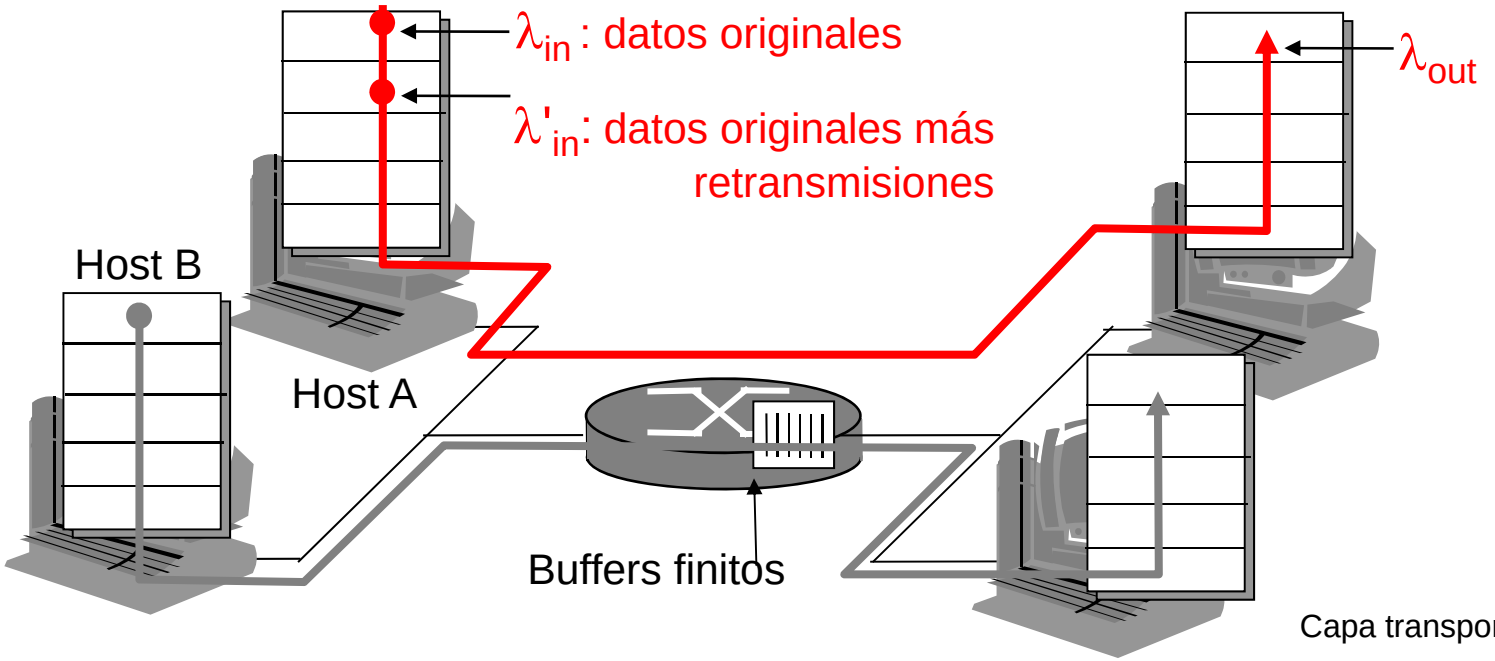




- Grandes retardos en congestión
- **Velocidad de transmisión** (λ)
  - Velocidad a la que la aplicación envía datos (**goodput= velocidad a nivel aplicación**) λ_{in}

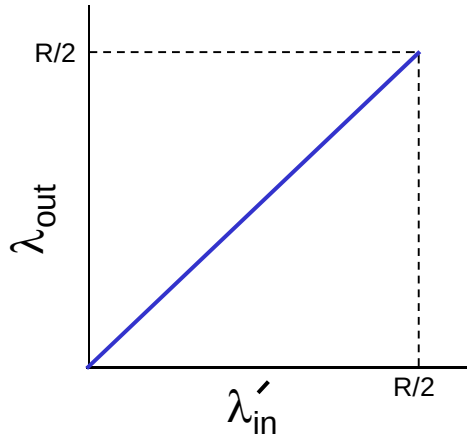
Causas/costo de la congestión: escenario 2

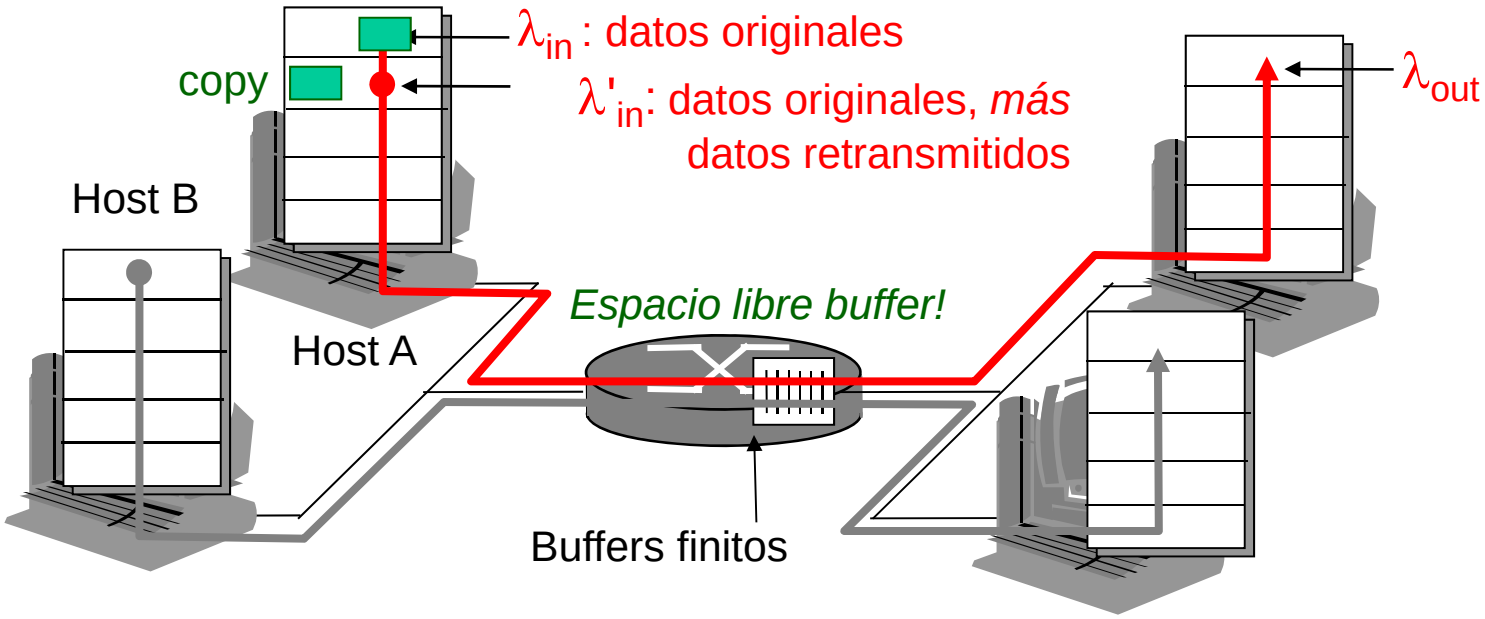
- 2 emisores, 2 receptores, 1 router con buffers *finitos*
- Emisor retransmite paquetes descartados
  - **Carga ofrecida**, goodput+tasa bytes retransmitidos $\lambda_{in}' \geq \lambda_{in}$



Escenario de congestión 2a: caso ideal

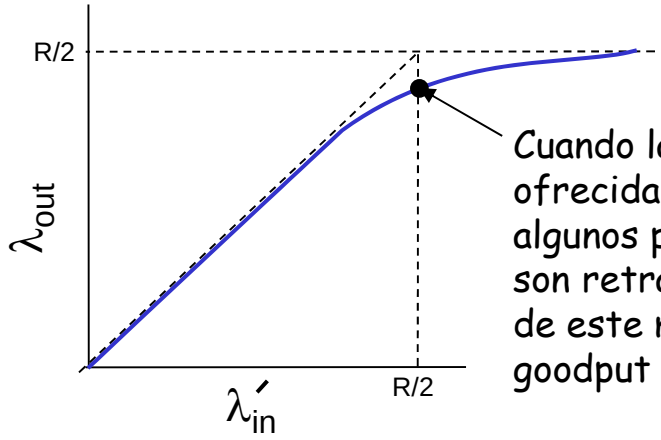
- El emisor solo envía cuando el buffer del router tiene disponibilidad



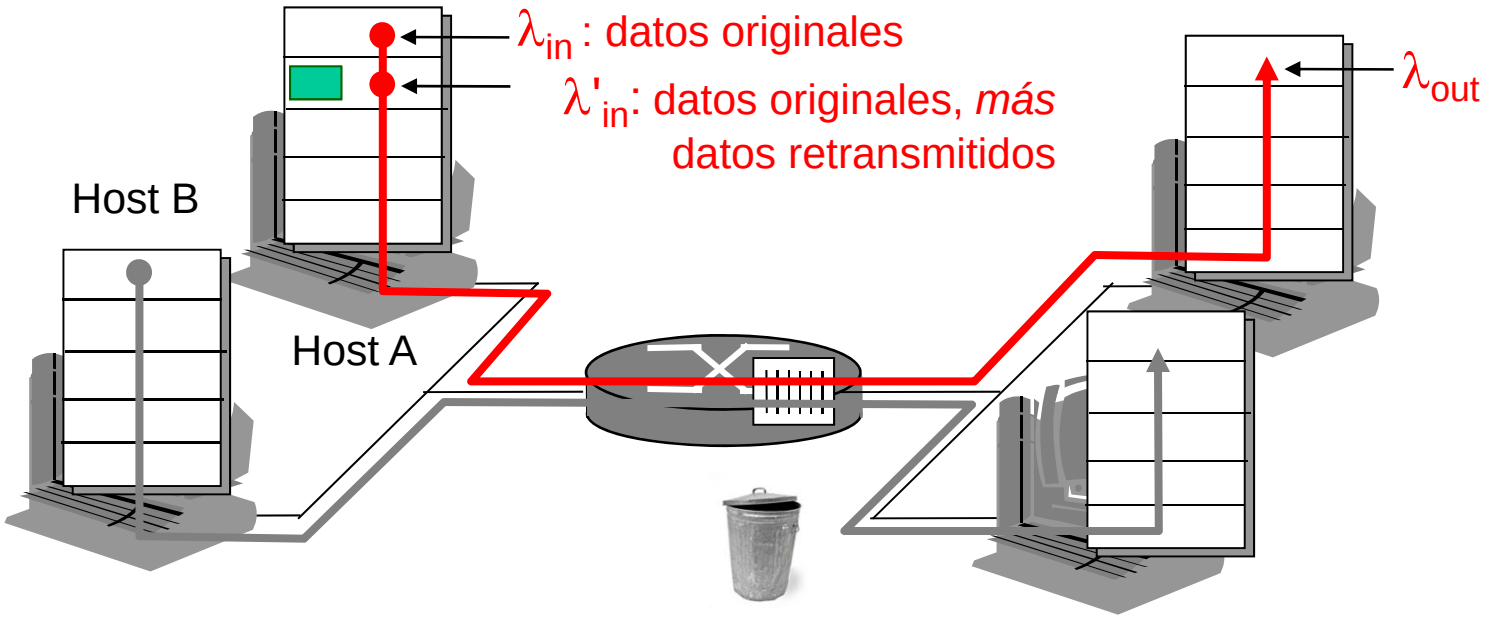


Escenario de congestión 2b: *reenvios*

- Los paquetes pueden ser descartados en el router por tener los buffers llenos

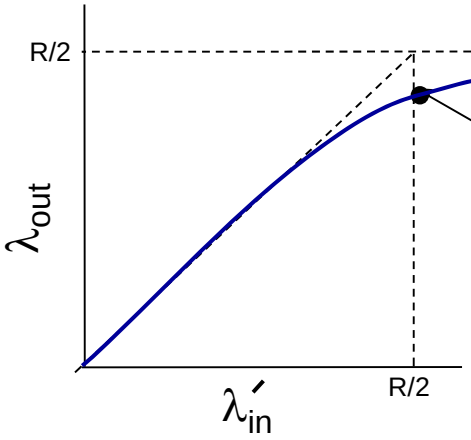


Cuando la carga ofrecida es R/2, algunos paquetes son retransmisiones de este modo el goodput es inferior

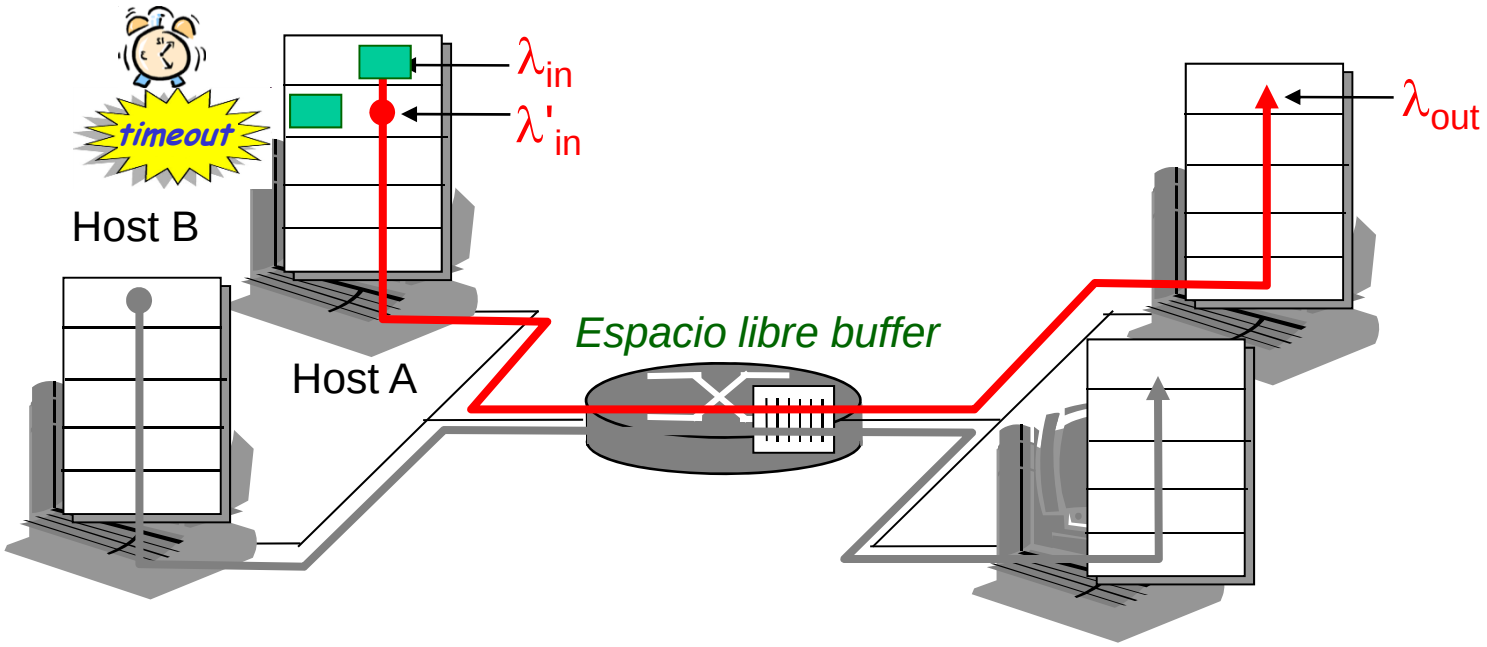


Escenario de congestión 2c: *duplicados*

- Además el emisor puede reenviar dos o más copias del mismo paquete inecesariamente



Cuando se transmite a tasa R/2, algunos paquetes son duplicados disminuyendo la llegada de datos de aplicación a destino!



Escenario de congestión 2c: *costes*

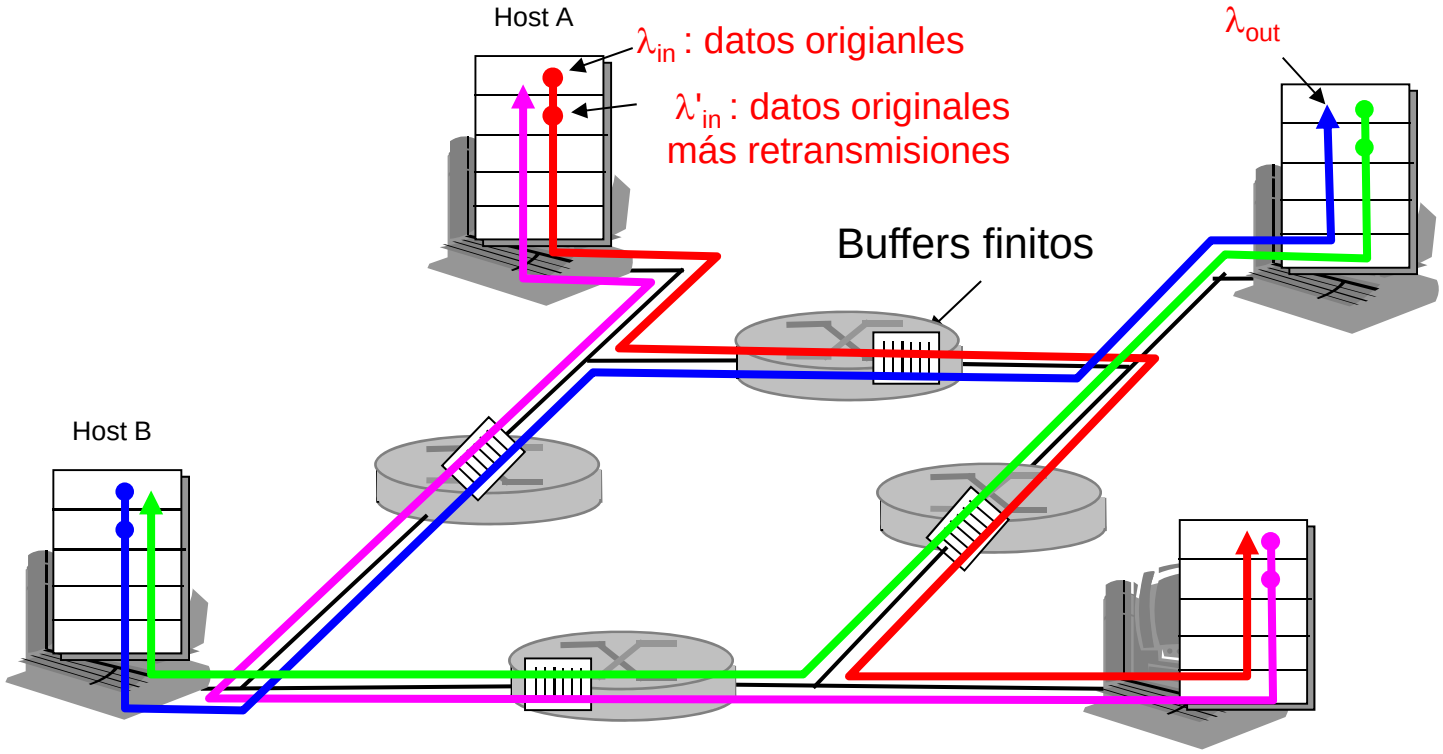
Los "costes" de la congestión:

- Más trabajo para un "goodput" dado, retransmisiones.
- Retransmisiones innecesarias (duplicados): los enlaces llevan tráfico que se descartará en destino
  - Disminuye el goodput

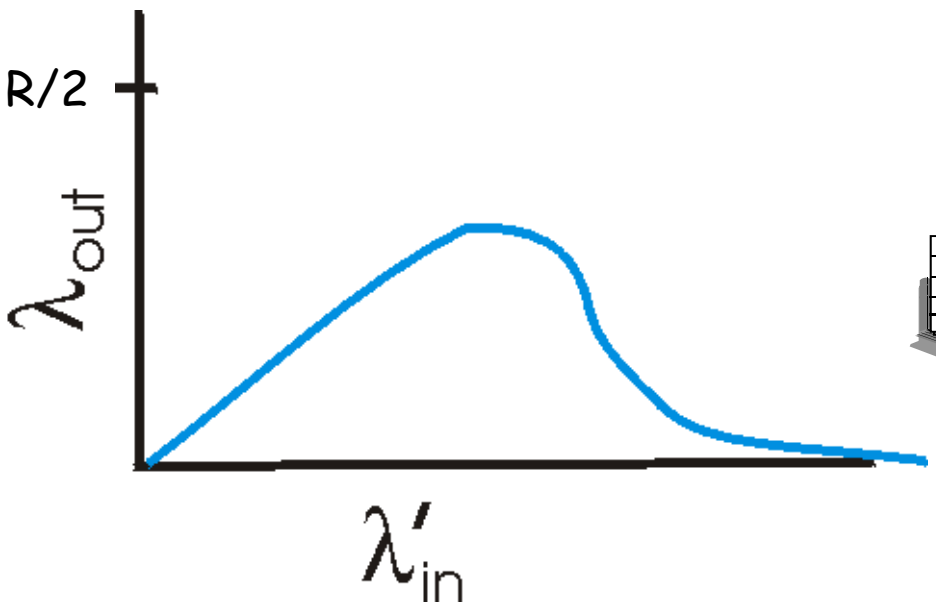
Escenario 3

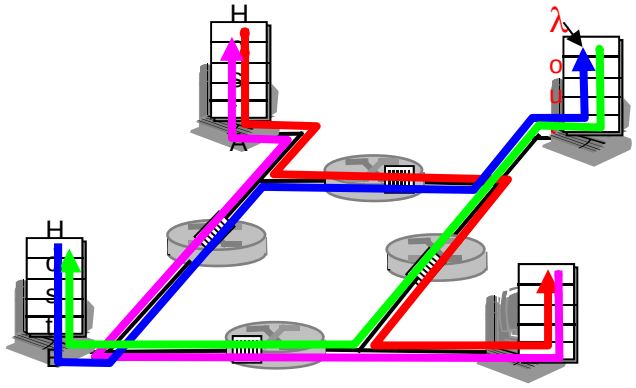
- 4 emisores
- Caminos multisalto
- timeout/retransmisiones

Q: Que sucede cuando aumentamos λ_{in} ?



Coste de la congestión: Escenario 3





Otro coste de la congestión cuando se descarta un paquete, todo la capacidad usada por ese paquete con anterioridad fue inútil.

Métodos para controlar la congestión

2 métodos más comunes de control de congestión:

Control de congestión terminal-a-terminal:

- La capa de red no proporciona soporte explicito a la capa de transporte
- La congestión es inferida analizando el comportamiento de la red en los sistemas terminales (perdida de paquetes, retardos, …)
- Método usado en la práctica por TCP

Control de congestión asistido por la red:

- Los routers proporcionan realimentación explicita a los sistemas finales
  - Por ejemplo, un bit en el paquete que indique congestión
  - Incluso, puede llegar a dar información de cuanto está un router capacitado para ofrecer

Capítulo 3: capa de transporte

3.1 Servicios de la capa de transporte

3.2 Multiplexación y demultiplexación

3.3 UDP

3.4 Transferencia fiable

3.5 TCP

- Estructura de los segmentos
- Fiabilidad en la transmisión de datos
- Control de flujo
- Gestión de la conexión

3.6 Principios de control de congestión

3.7 Control de congestión en TCP

Control de congestión en TCP: detalles

- Se usa una variable adicional: ventana de congestión (cwnd)
- Cantidad de datos no reconocidos en un emisor (==tamaño ventana) no puede exceder el mínimo entre la ventanas de recepción (del otro extremo) y congestión (local)
- **cwnd** es modificado dinámicamente según congestión

¿Cómo el emisor percibe congestión?

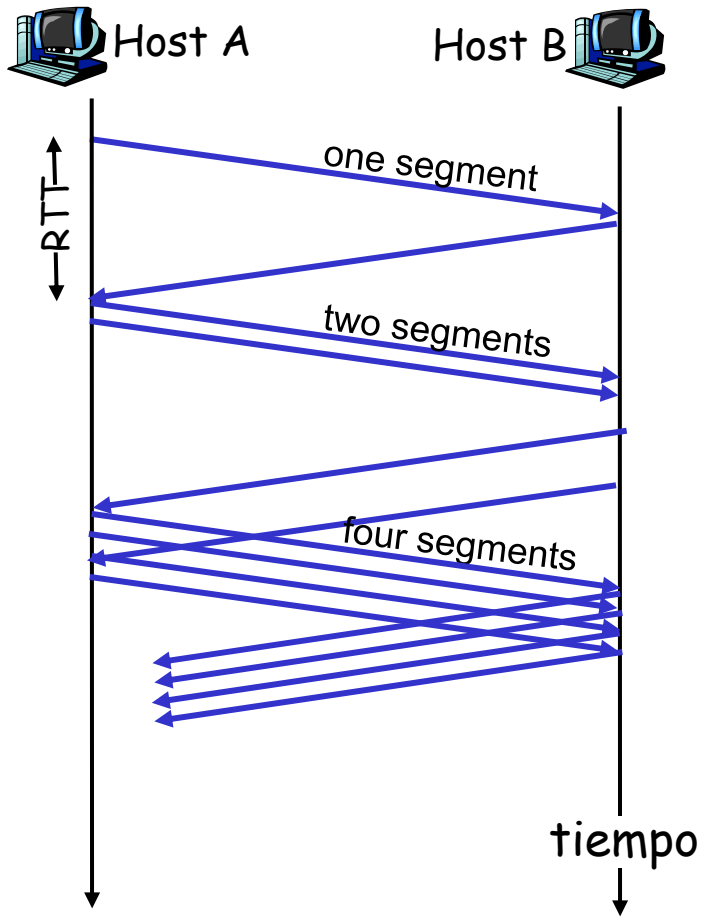
- Pérdidas = timeout o 3 ACKs duplicados
  - El emisor TCP reduce la tasa (**cwnd**) después de estos eventos

¿Cómo? 3 mecanismos:

- Arranque lento
- Evitación de la congestión
- Recuperación rápida

TCP Arranque lento

- Cuando la conexión empieza, se incrementa la tasa exponencialmente hasta el primer evento de perdida:
  - Inicialmente **cwnd** = 1 MSS
  - Cada ACK recibido +1 MSS ≈ "Se dobla **cwnd** cada RTT" (modelo simplificado)
- Lento… La tasa inicial es baja pero crece rápidamente



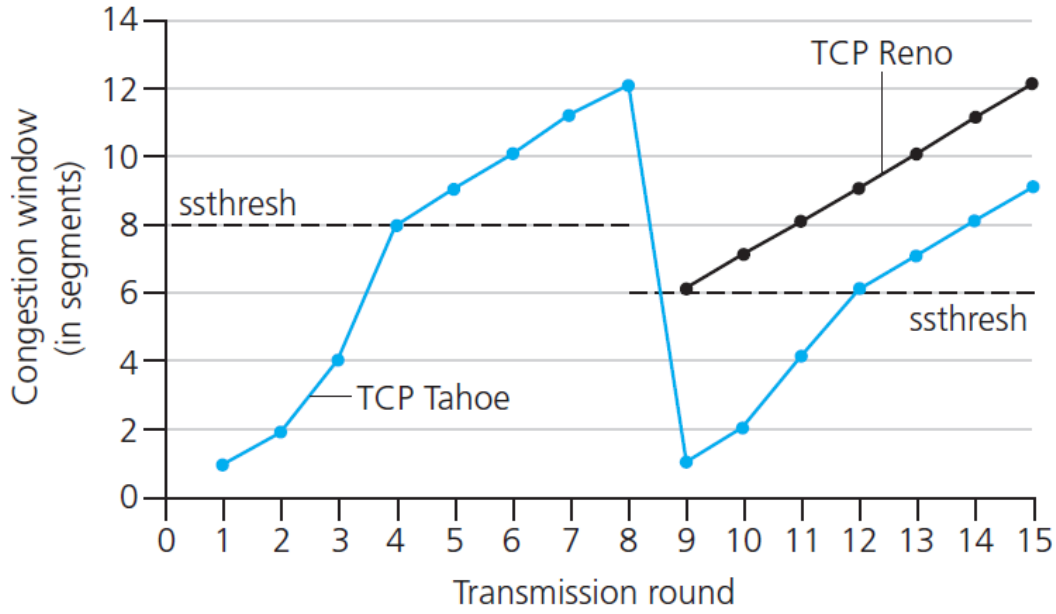
Recuperación rápida (Reno) y Evitación de la congestión

- Después de timeout:
  - **Arranque lento** (cwnd=1)
  - …Hasta un **umbral** que es la mitad de cwnd (anterior) cuando se pasaría a **Evitación de la congestión**
    - MSS/cwnd cada ACK ≈ Ventana crece linealmente 1 MSS por RTT (simplif.)
- Si 3 ACKs duplicados:
  - **TCP Tahoe, igual que timeout**
  - En caso de **TCP Reno (recuperación rápida)**:
    - **cwnd** pasa a ser la mitad (**umbral**)
    - **y Evitación de la congestión sobre umbral**

Filosofía:

- 3 ACKs dups. indican la capacidad de la red de transportar varios segmentos
- timeout indican una situación de congestión más severa

Recuperación rápida (Tahoe y Reno) - Modelo simple

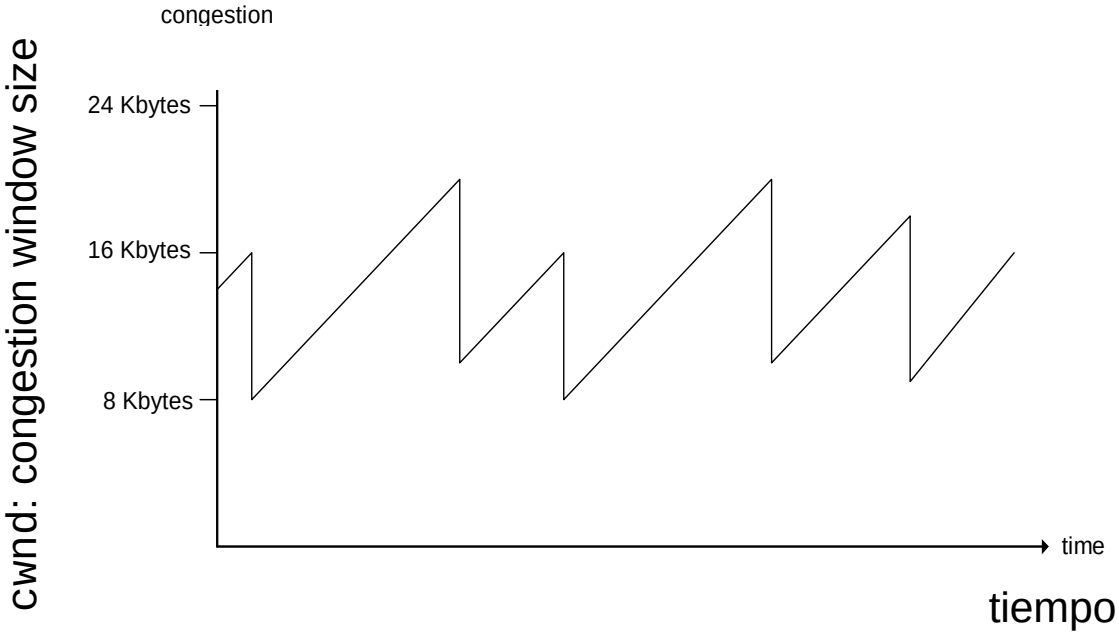


TCP control de la congestión (estado estacionario y simplificado):

crecimiento aditivo y decrecimiento multiplicativo

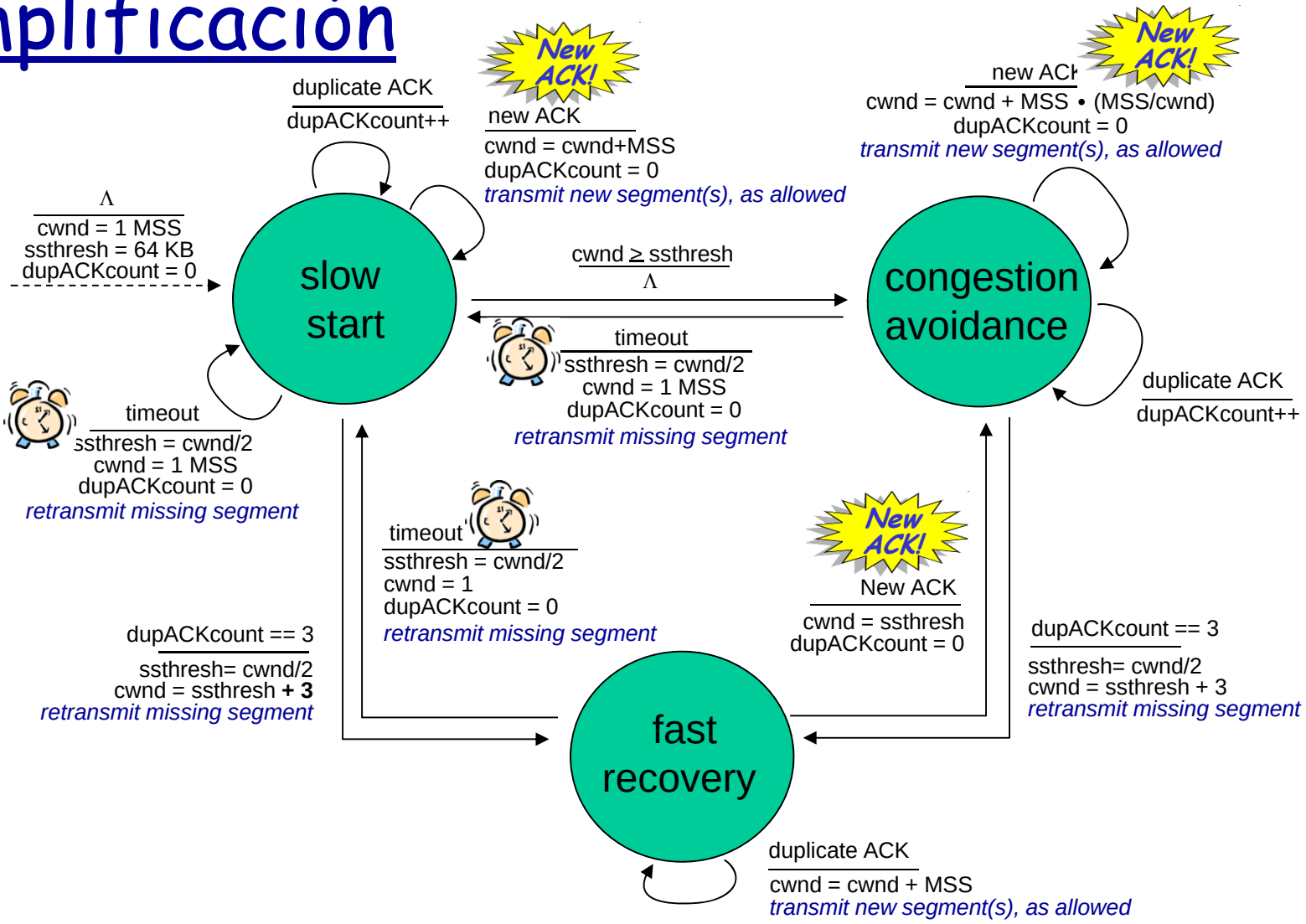
- *Método:* incrementar tasa de transmisión (tamaño de la ventana), hasta que se determinen perdidas
  - *Aditivo:* incrementar **cwnd** (tamaño ventana congestión) "por 1 MSS cada RTT" hasta que se detecten perdidas
  - *Decremento multiplicativo*: fijar **cwnd** a la mitad después de perdida

Comportamiento diente de sierra



tiempo

TCP Control Congestion sin simplificación

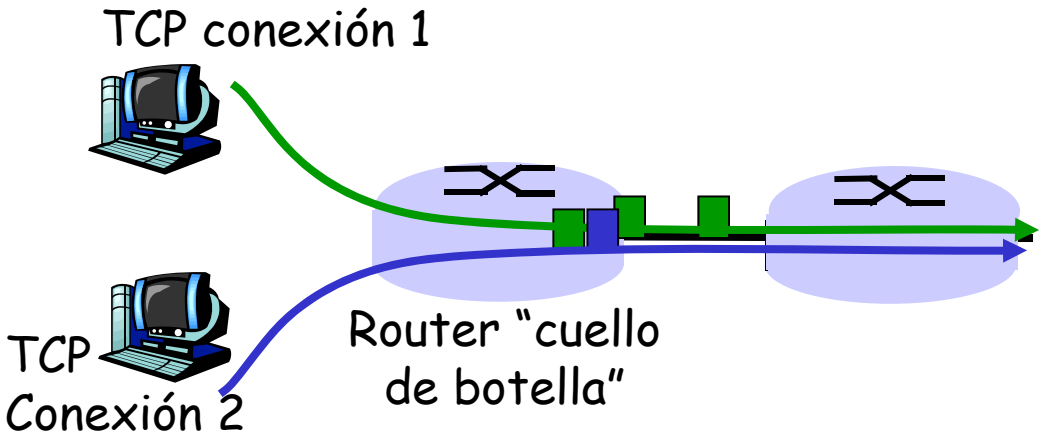


TCP throughput (estacionariedad)

- Cual es el *throughout* medio de TCP en función del tamaño de ventana y RTT?
  - Supongamos, que se puede obviar el arranque lento
- Sea W el tamaño de la ventana cuando hubo perdidas
  - Cuando la ventana es W, *throughput* es aproximadamente W/RTT
  - Justo después del evento, la ventana es fijada a W/2, luego el *throughput* es (W/2)/RTT.
  - Luego el *throughout* medio entre perdidas de forma ideal sería: (0.75 W)/RTT (segmentos/s)

TCP Equidad

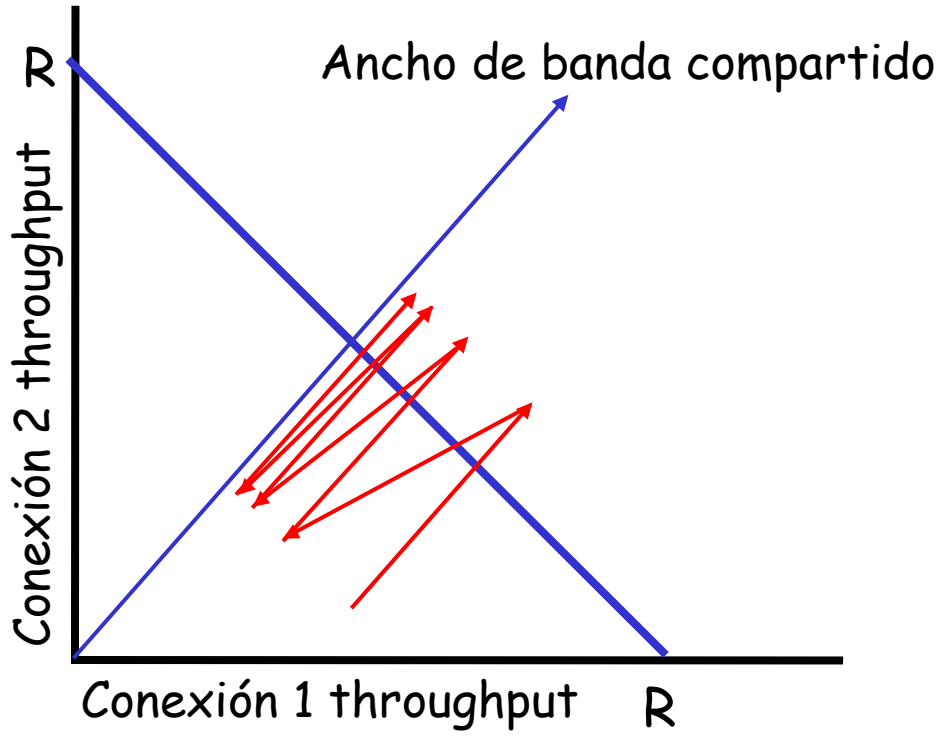
Objetivo de equidad: si K conexiones TCP comparten un enlace cuya capacidad es R, cada una debería tener una tasa media de R/K



¿Por qué es TCP justo?

Ejemplo simple, de dos sesiones compitiendo:

- Los incrementos tienen pendiente 1 cuando el *throughout* aumenta
- Los decrementos son proporcionales



Equidad (más)

Equidad y UDP

- Apl. multimedia muchas veces no usan TCP
  - Prefieren perdidas a reducir la tasa
  - Esto puede perjudicar al resto de usuarios

Equidad y conexiones TCP paralelas

- No hay límite al número de conexiones a abrir entre dos equipos
- Los navegadores, o aplicaciones de tipo FH
- Ejemplo: enlace de tasa R con 10 conexiones:
  - 1 apl. 1 TCP conn., consigue R/10
  - Una apl. abre 11 TCPs, consigue R/2 !