

# Capítulo 3: Capa Transporte - IV

## ELO322: Redes de Computadores Agustín J. González

Este material está basado en:

- ▣ Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet 3rd* edition. Jim Kurose, Keith Ross Addison-Wesley, 2004.

# Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Administración de conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP

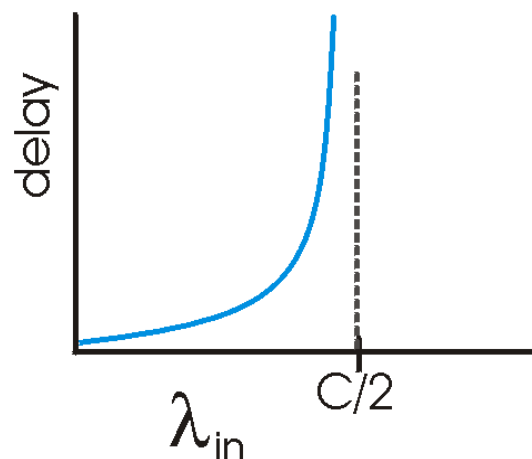
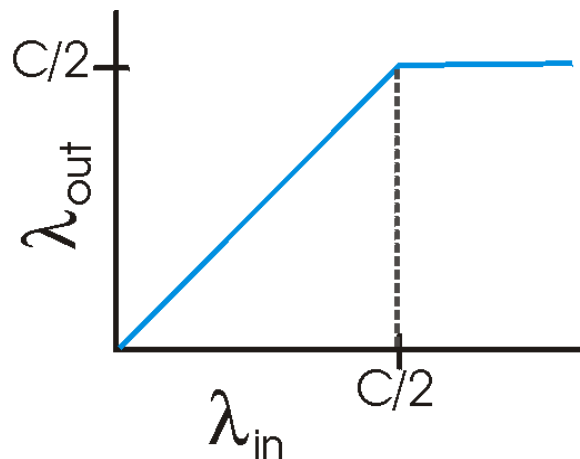
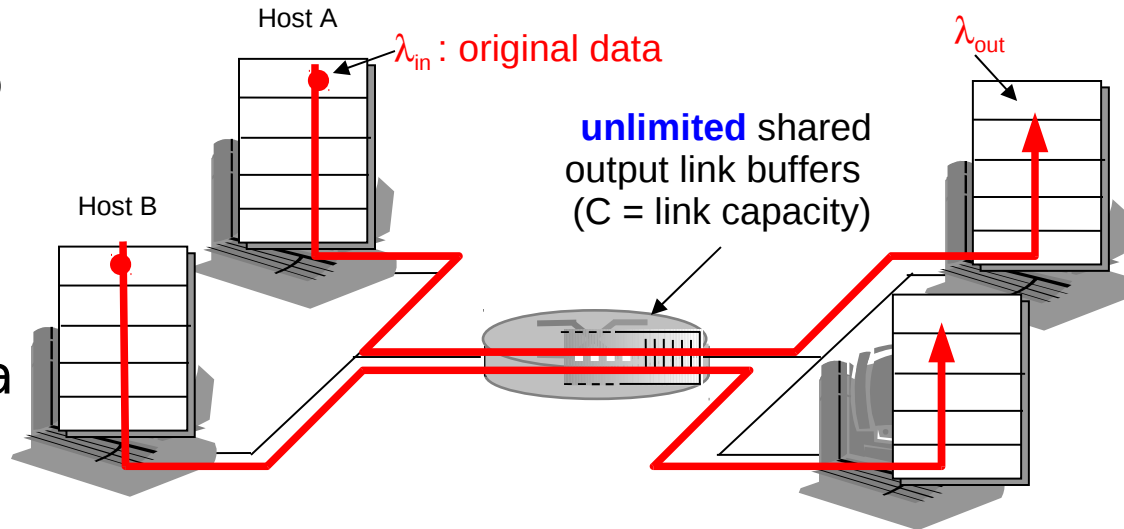
# Principios del control de congestión

## Congestión:

- Informalmente: “demasiadas fuentes enviando muchos datos muy rápido para que la red lo maneje”
- Es distinto a control de flujo, éste ocurre entre dos aplicaciones.
- Manifestaciones:
  - Pérdidas de paquetes (buffer overflow en routers)
  - Grandes retardos (en las colas en los router)
- Uno de los problemas top-10!

# Causas/costos de congestión: escenario 1 (buffer $\infty$ , sin re-envío)

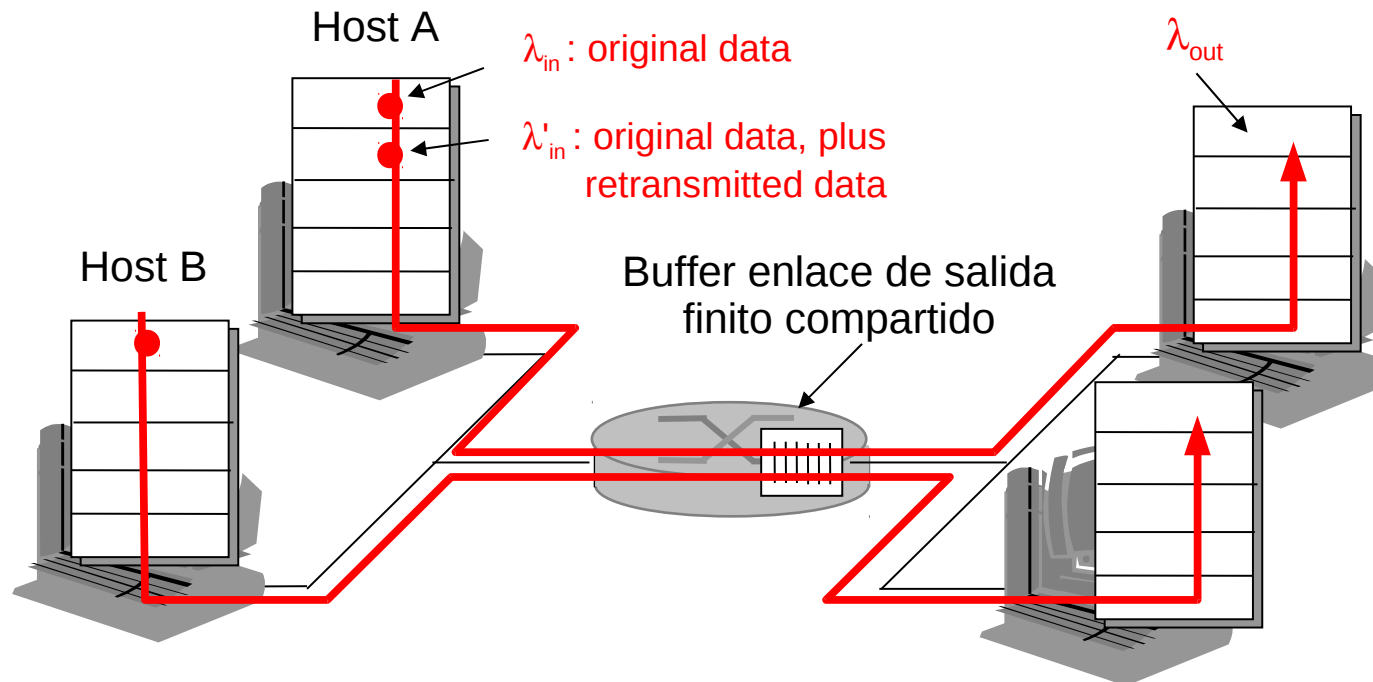
- dos transmisores, dos receptores
- un router, buffer tamaño **infinito**
- **sin retransmisión**
- $\lambda_{in}$  datos enviados por la aplicación (bytes/sec)
- $\lambda_{out}$  datos recibidos por la aplicación (bytes/sec)



- grandes retardos en estado congestionado (muchos paquetes esperando en cola)
- máximo flujo posible (throughput = C) de datos

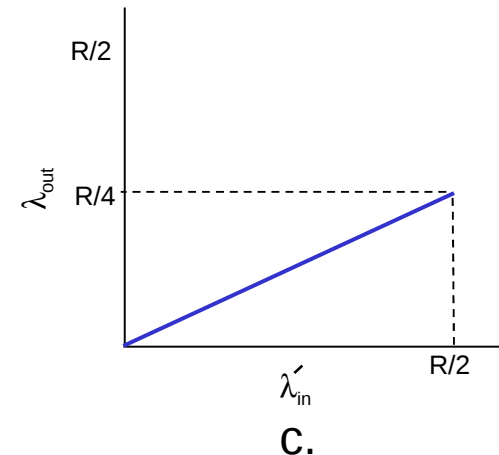
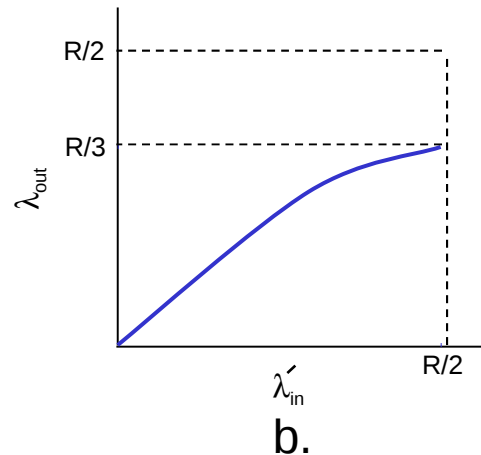
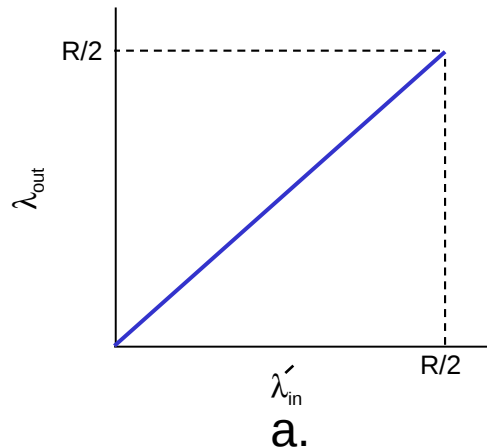
# Causas/costos de congestión: escenario 2 (buffer no $\infty$ , con re-envío )

- un router, buffer *finito*
- Se pierden paquetes que no tienen espacio en buffer.
- transmisor **retransmite** paquetes perdidos



# Causas/costos de congestión: escenario 2

- (a) caso perfecto, sin pérdidas ni retransmisiones:  $\lambda_{in} = \lambda'_{in}$
- (b) retransmisión sólo en caso de pérdida de segmentos (no por timeout prematuro):  $\lambda'_{in} > \lambda_{out}$
- (b) retransmisión de paquetes hace que  $\lambda'_{in}$  sea más grande (que el caso perfecto) para el mismo  $\lambda_{out}$  (No necesariamente  $R/3$  como cota)
- (c) retransmisiones innecesarias (timeout prematuros) : enlaces envían múltiples copias del paquete (e.g. dos re-envíos por paquete)



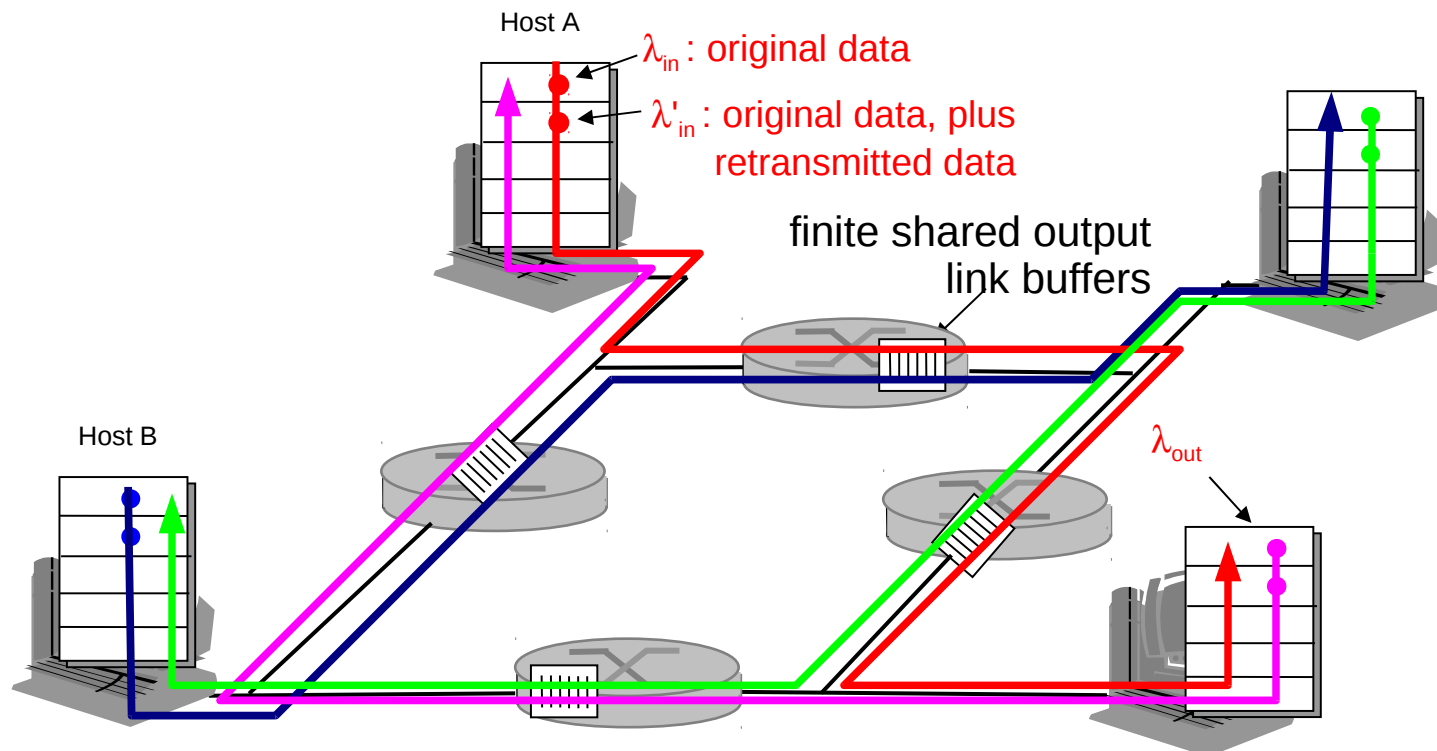
## “costos” de congestión:

- más trabajo (retransmisión) para lograr el transporte de datos  
“goodput”

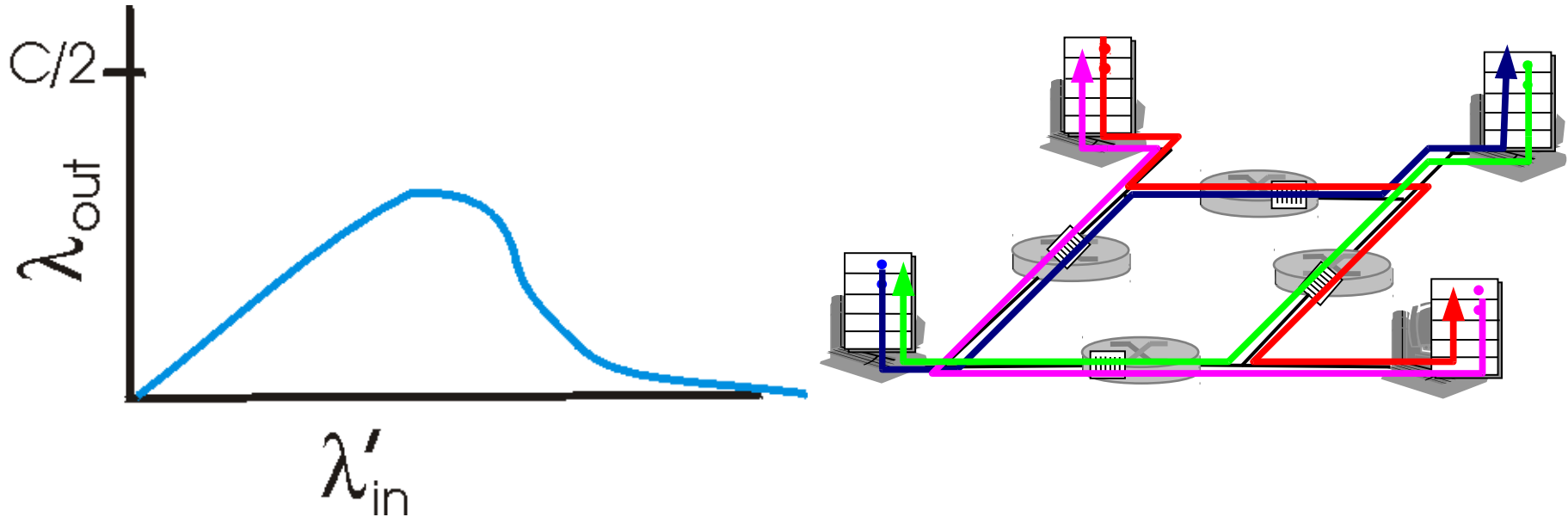
# Causas/costos de congestión: escenario 3

- cuatro transmisores
- rutas con multihops
- timeout/retransmisiones

**Q:** ¿Qué pasa cuando  $\lambda_{in}$  se incrementa y  $\lambda'_{in}$  crece?



# Causas/costos de congestión: escenario 3



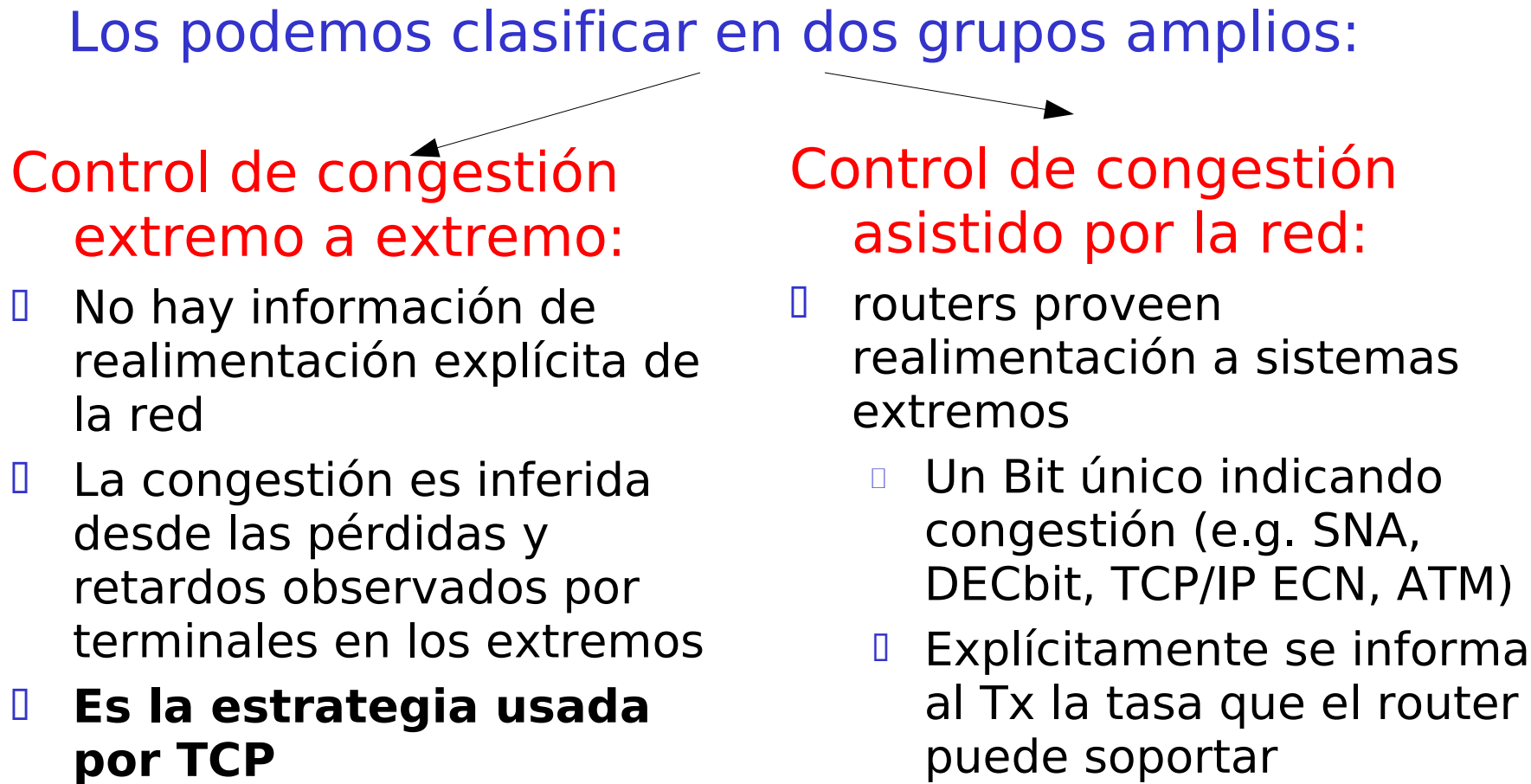
## Otro “costo” de congestión:

- cuando se descartan paquetes, cualquier capacidad (de router) usada anteriormente “upstream” pasa a ser un recurso desperdiciado!



# Estrategias para control de congestión

Los podemos clasificar en dos grupos amplios:



Control de congestión extremo a extremo:

- No hay información de realimentación explícita de la red
- La congestión es inferida desde las pérdidas y retardos observados por terminales en los extremos
- **Es la estrategia usada por TCP**

Control de congestión asistido por la red:

- routers proveen realimentación a sistemas extremos
  - Un Bit único indicando congestión (e.g. SNA, DECbit, TCP/IP ECN, ATM)
  - Explícitamente se informa al Tx la tasa que el router puede soportar

# Caso de estudio: Control de congestión en ATM ABR (tecnología de red capa 3 y menores)

## ABR: Available Bit Rate:

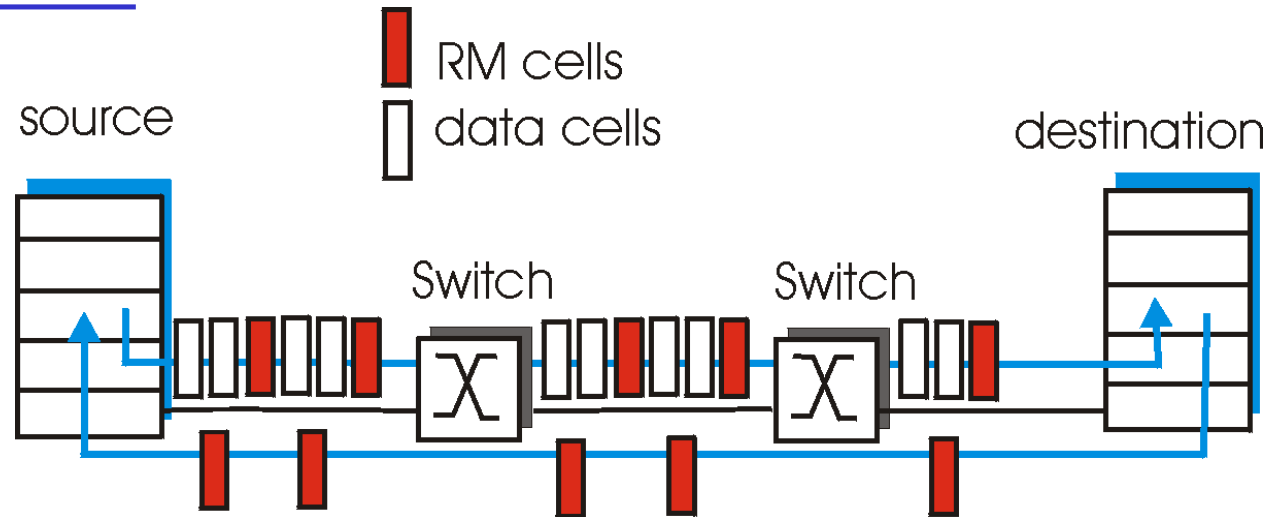
- Es un servicio “elástico” o flexible
- Si camino del Tx tiene poca carga,
  - Tx debería usar ancho de banda disponible
- Si camino de Tx a Rx está congestionado,
  - Tx reduce a un mínimo la tasa garantizada

## Celdas RM (Resource Management):

- Enviadas a intervalos por Tx entre celdas de datos
- bits en celda RM son modificados por switches
  - **Bit NI:** no incrementar tasa (= congestión moderada)
  - **Bit CI:** Congestion Indication
- Celdas RM son retornadas al Tx por el Rx con bits intactos

ATM: Asynchronous Transfer Mode

# Caso de estudio: Control de congestión en ATM ABR



- En celda RM hay campo ER (explicit rate) de dos bytes:
  - Un Switch congestionado puede bajar valor de ER en la celda
  - Tasa de envío del Tx se ajusta a la tasa mínima soportable en el camino entre fuente y destino (la del switch más crítico)
- En celda de datos hay Bit EFCI (explicit forward congestion indicator): éste es fijado en 1 por switch congestionado
  - Si celda de datos precedente a celda RM tiene el EFCI marcado, el destino marca bit CI en celda RM retornada.

# Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Administración de conexión
- 3.6 Principios del control de congestión
- 3.7 **Control de congestión en TCP**

# Control de Congestión en TCP

- Usa control extremo a extremo (**sin asistencia de la red**)

- Tx limita su ventana de transmisión:

$$\begin{aligned} & LastByteSent - LastByteAcked \\ & \leq \min \{ CongWin, RcvWindow \} \end{aligned}$$

- Si Rx tiene espacio, se tiene:

$$tasa_{Aprox} = \frac{CongWin}{RTT} \text{ [Bytes/sec]}$$

- **CongWin** es dinámica y función de la congestión percibida de la red
- **RcvWindow** es el número de bytes que el Rx puede recibir en su buffer, la suponemos grande y no limita la tasa de envío.

¿Cómo el Tx percibe la congestión?

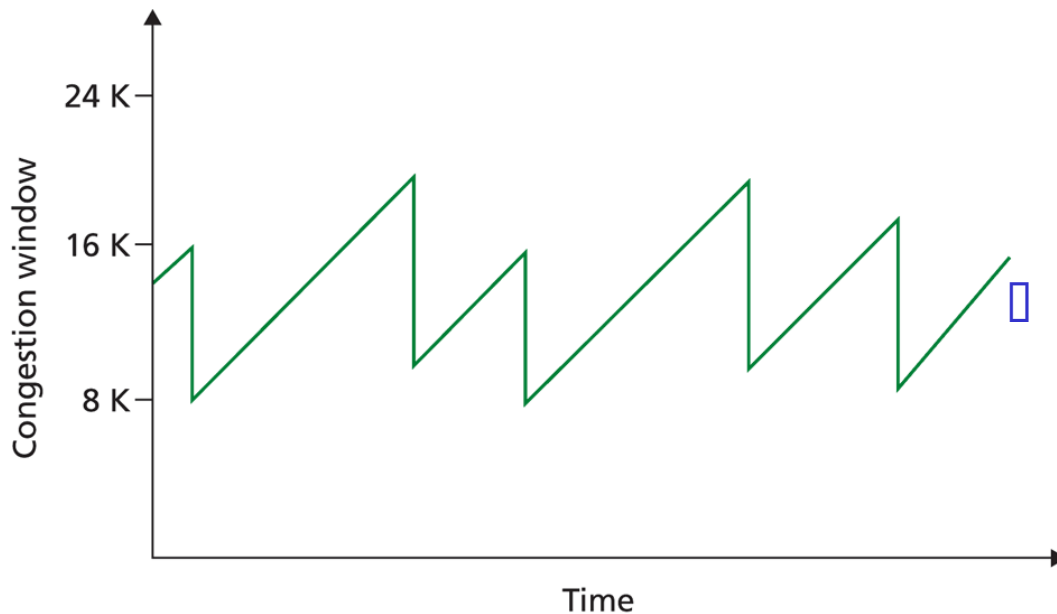
- **Evento de pérdida** = timeout ó 3 acks duplicados
- Tx TCP reduce tasa (**CongWin**) después de un evento de pérdida

Hay tres mecanismos:

- **AIMD** (Additive-Increase, Multiplicative-Decrease)
- **“Partida lenta”**
- **Conservativo** después de evento de **timeout**

# TCP AIMD (Additive-Increase, Multiplicative-Decrease)

- **Decrecimiento multiplicativo:** reducir **CongWin** a la mitad luego de pérdida



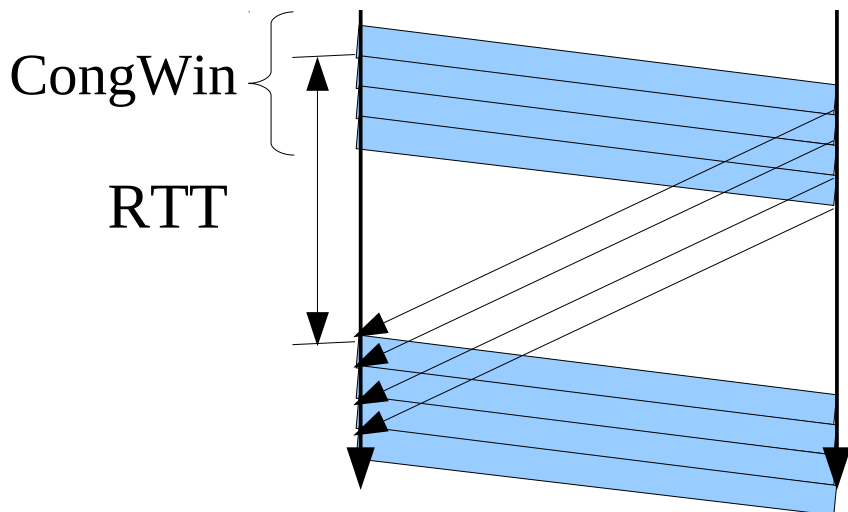
- **Aumento aditivo:** aumenta **CongWin** en 1 MSS cada RTT en ausencia de pérdida. En algunas implementaciones **CongWin** incrementa en  $MSS \times (MSS / CongWin)$  por cada ACK recibido.
- MSS (Maximum Segment Size) es la máxima cantidad de datos que se envía en cada segmento sin fragmentarse.

**Figure 3.50** ♦ Additive-increase, multiplicative-decrease congestion control

# Aumento aditivo

- La idea es aumentar un MSS luego de un RTT.
- Podemos aproximarnos aumentando la CongWin cada vez que se recibe un ACK de manera que al completar 1 RTT hayamos sumado un MSS.

- Se envía como máximo CongWin bytes y esperamos por el acuse de recibo.



$$NumSegmentos = NumAcks = \frac{CongWin}{MSS}$$

$$Incr. = \frac{MSS}{NumAcks} = \frac{MSS}{\frac{CongWin}{MSS}} = \frac{MSS * MSS}{CongWin}$$

- Incr. : Incremento por cada ACK

Si hay ACK retardados, el incremento debe ser mayor

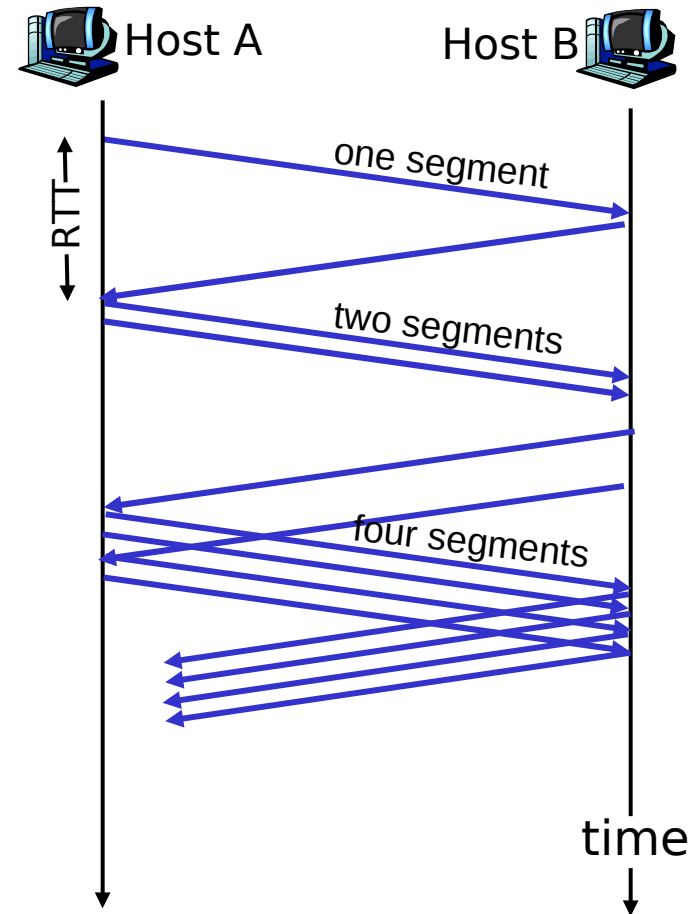
# Partida lenta en TCP (slow start)

- Cuando la conexión comienza, **CongWin** = 1 MSS
  - Ejemplo: MSS = 500 bytes & RTT = 200 msec
  - Tasa inicial = 20 kbps
- Ancho de banda disponible puede ser  $\gg$  MSS/RTT
  - Es deseable aumentar tasa rápidamente hasta una tasa respetable
- Cuando la conexión comienza, aumentar tasa **exponencialmente** rápido hasta tener primer evento de pérdida
- Se le llama Slow Start porque parte desde tasa muy abaja.



# Partida Lenta en TCP (más)

- Cuando la conexión comienza, aumentar tasa exponencialmente hasta primera pérdida:
  - **Duplicar CongWin** cada RTT
  - Es hecho incrementando **CongWin** en 1 MSS por cada ACK recibido
- Resumen: tasa inicial es lenta pero se acelera **exponencialmente** rápido



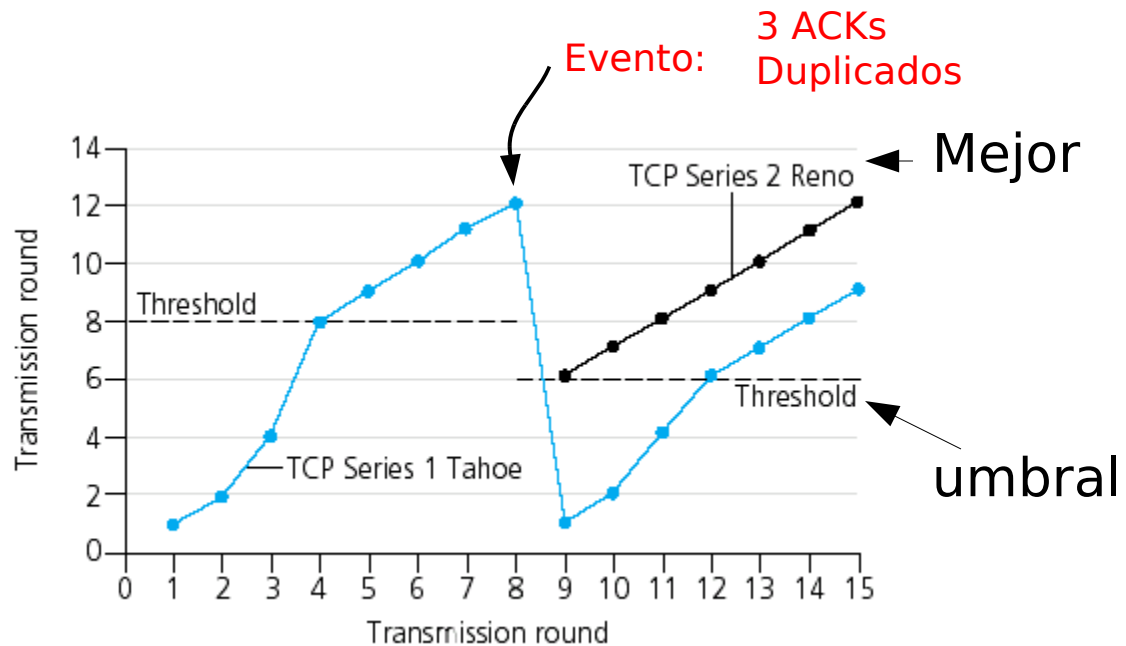
# Reacción ante eventos de timeout

**Q:** ¿Cuándo debería cambiar el aumento de exponencial a lineal?

**A:** Un buen criterio es: Cuando **CongWin** llega a 1/2 de su valor antes del timeout.

## Implementación:

- Umbral variable (variable threshold)
- Ante evento de pérdidas, el umbral es fijado en 1/2 de CongWin justo antes de la pérdida



**Tahoe:** primera versión de control de congestión en TCP. No distinguía entre timeout o ACK duplicados.

**Reno:** versión siguiente en TCP. Sí distingue timeout de ACK duplicados. Es como TCP opera hoy.

# Reacción ante eventos de timeout (cont)

- Después de **3 ACKs duplicados**:
  - **CongWin** baja a la mitad
  - Luego la ventana crece **linealmente**
- Después de un **timeout**:
  - **CongWin** es fijada en 1 MSS;
  - Luego la ventana crece **exponencialmente** hasta un umbral, luego crece linealmente

## Filosofía:

- 3 ACKs duplicados indican que la red es capaz de transportar algunos segmentos (sólo llegan fuera de orden en el Rx). Se perdió uno pero llegaron los otros y por eso tenemos ACKs duplicados
- timeout antes de 3 duplicados es “más alarmante” (no llegaron!)

# Resumen: Control de Congestión en TCP

- Cuando **CongWin** está bajo el **Threshold** (**umbral**), Tx está en fase **slow-start**, la ventana de transmisión crece exponencialmente (un MSS por cada ACK).
- Cuando **CongWin** está sobre **Threshold**, Tx está en fase **abolición de congestión**, la ventana crece linealmente (aprox. un MSS por cada RTT).
- Al **tercer ACK duplicados**, **Threshold** pasa a **CongWin/2** y **CongWin** pasa a **Threshold**.
- Cuando ocurre un **timeout**, **Threshold** pasa a **CongWin/2** y **CongWin** se lleva a 1 MSS.

# Control de congestión del Tx TCP

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulta en una duplicación de CongWin cada RTT.
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin + MSS * (MSS/CongWin)	Aumento aditivo, resulta en aumento de CongWin en aprox. 1 MSS cada RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Recuperación rápida, implementando reducción multiplicativa. CongWin no caerá a 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Ingresa a Partida Lenta (slow start)
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin y Threshold no cambian

# Throughput Simplificado de TCP (tasa de transferencia de datos lograda)

- ¿Cuál es el throughput promedio de TCP como una función del tamaño de ventana **CongWin** y RTT?
  - Ignoremos **slow start** ya que al ser exponencial es una fase muy corta
- TCP pide ancho de banda adicional al incrementar  $W$  en 1 MSS por cada RTT hasta una pérdida
- Sea  $W$  el tamaño de la ventana (en bytes) cuando ocurre una pérdida.
- Cuando la ventana es  $W$ , el throughput es  $W/RTT$
- Justo después de la pérdida, la ventana cae a  $W/2$ , y el throughput a  $W/2RTT$ .
- Throughput promedio entre  $W/2RTT$  y  $W/RTT$  es  $0.75 W/RTT$
- Esto debido a que el throughput crece linealmente entre ambos valores.

# Futuro de TCP

- Ejemplo: segmentos de 1500 bytes, RTT de 100ms, queremos throughput de 10 Gbps
- Requiere tamaño de ventana **CongWin**  $W = 83.333$  (segmentos en tránsito)

- Throughput en términos de tasa de pérdida (L) es:

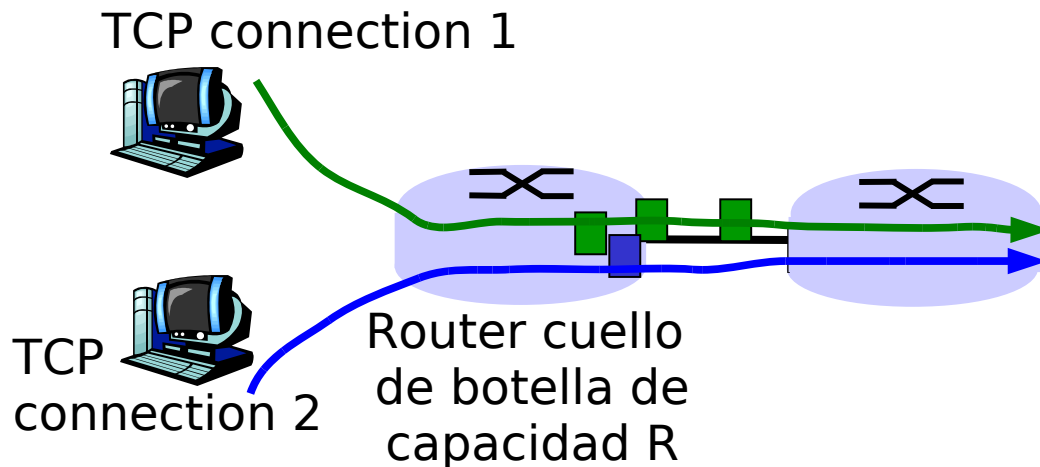
$$Avg. Throughput = \frac{1,22 * MSS}{RTT \sqrt{L}}$$

$L = (\text{bytes perdidos}) / (\text{Número total enviados})$

- Para el throughput deseado con el algoritmo de control de congestión actual se toleran probabilidades de pérdida de sólo  $L = 2 \cdot 10^{-10}$  **Wow** (*1 cada 5 mil millones de segmentos*)
- Se requieren nuevas versiones de TCP para enlaces de alta velocidad (interesados ver RFC 3649)

# Equidad en TCP

- **Objetivo de la Equidad (fairness):** Si  $K$  sesiones TCP comparten un mismo enlace de ancho de banda  $R$ , cada una debería tener una tasa promedio de  $R/K$

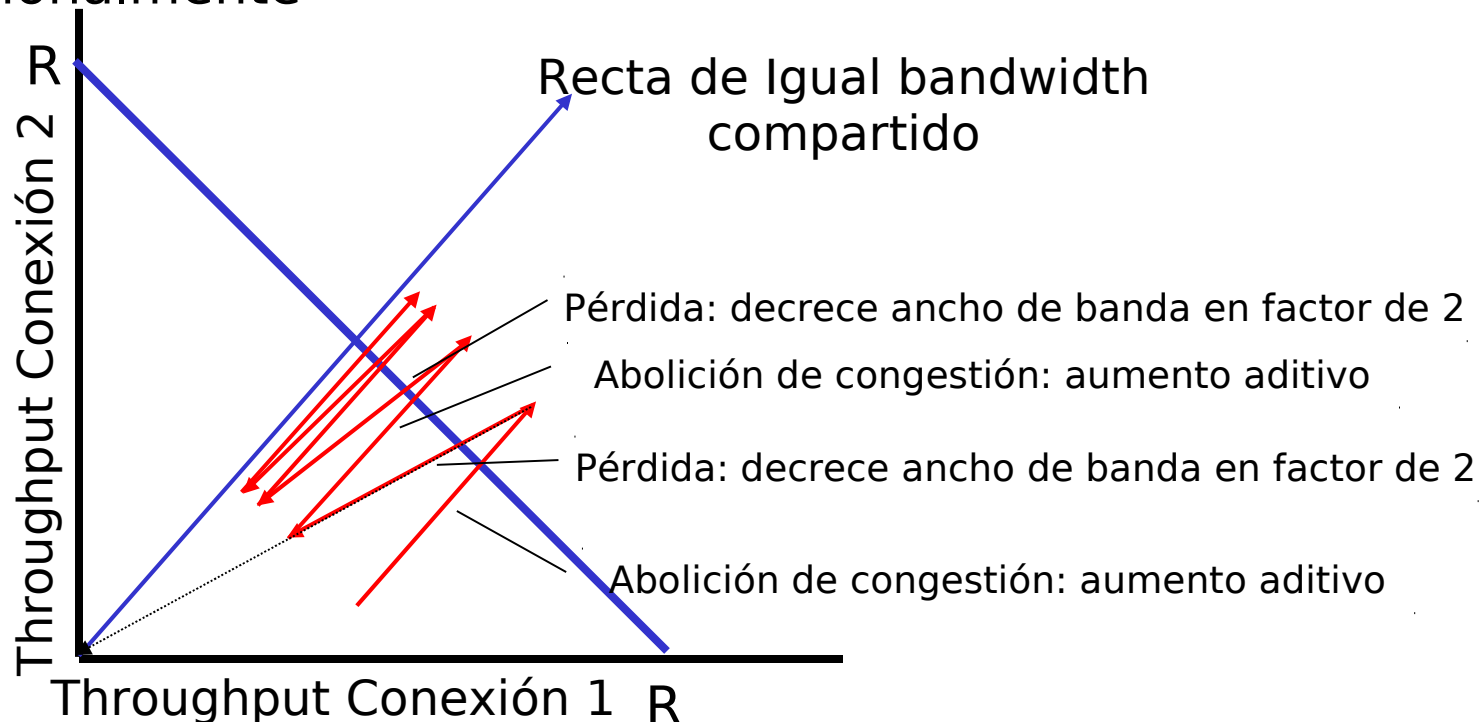




# ¿Por qué TCP es justa?

Supongamos dos sesiones compitiendo:

- Aumento aditivo da pendiente de 1, como aumento de throughput
- Reducción multiplicativa reduce throughput proporcionalmente



# Equidad (más)

## Equidad y UDP

- Aplicaciones Multimedia no usan TCP
  - No quieren tasa ahogada por control de congestión
- En su lugar usan UDP:
  - Envían audio/vídeo a tasa constante y toleran pérdidas de paquetes
- Área de investigación: Hacerlas amistosas con TCP (TCP friendly)

## Equidad y conexiones TCP paralelas

- Nada previene a las aplicaciones de abrir conexiones paralelas entre dos hosts.
- Navegadores WEB hacen esto
- Ejemplo: Sea un enlace de tasa  $R$  soportando 9 conexiones;
  - Una aplicación nueva pide 1 conexión TCP, obtendrá  $R/10$
  - Si la aplicación nueva pide 11 conexiones TCP, ésta obtendrá  $11R/20$ , más de  $R/2$ !

# Capítulo 3: Resumen

- Principios detrás de los servicios de capa transporte:
  - multiplexing, demultiplexing
  - Transferencia confiable de datos
  - Control de flujo
  - Control de congestión
- Uso e implementación en Internet
  - UDP
  - TCP

## A continuación

- Dejaremos la “periferia” o “edge” de la red (capas aplicación y transporte)
- Nos internaremos en el centro de la red “network core”