TCP

y control de congestión en Internet

Area de Ingeniería Telemática
http://www.tlm.unavarra.es

Redes
4º Ingeniería Informática

Hoy...

1. Introducción a las redes
2. Tecnologías para redes de área local
3. Conmutación de circuitos
4. Tecnologías para redes de área extensa y última milla
5. Encaminamiento
6. Arquitectura de conmutadores de paquetes
7. Control de acceso al medio
8. **Transporte extremo a extremo**

¿Por qué se pierden los paquetes?

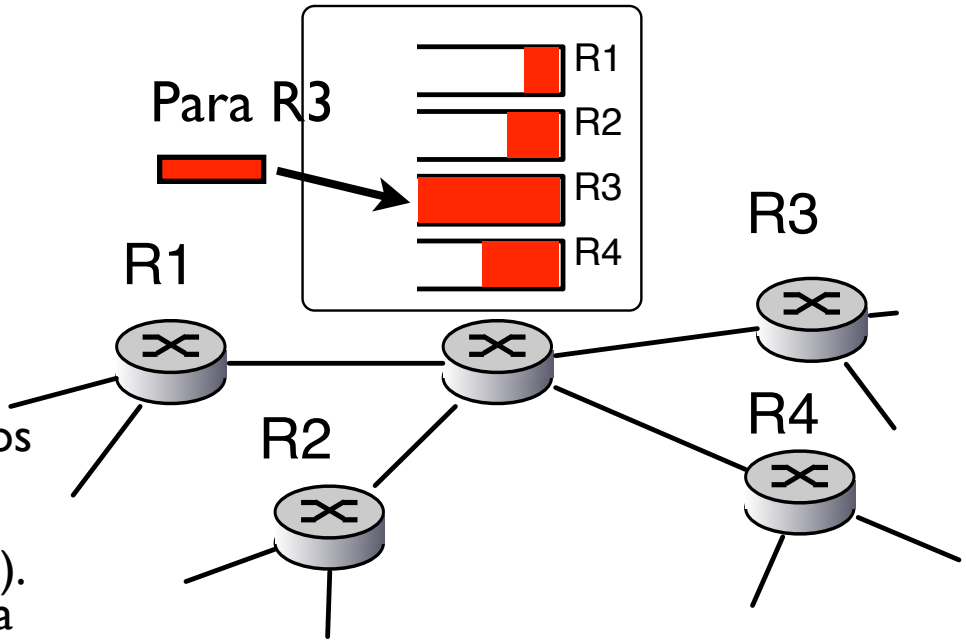
- Varias causas
  - **Errores de transmisión**.

    Modifican bits aleatorios de los paquetes. Los niveles que hacen detección de errores se ven obligados a descartar el paquete al no estar seguros de entregarlo correctamente

    Independiente del tráfico
  - **Desbordamiento de buffer**

    Un router que debe reenviar un paquete se encuentra con que todos sus recursos de memoria (por ejemplo en la cola de paquetes esperando la salida están ocupados). Debe descartar el paquete por falta de recursos

    Dependiente de la carga de la red



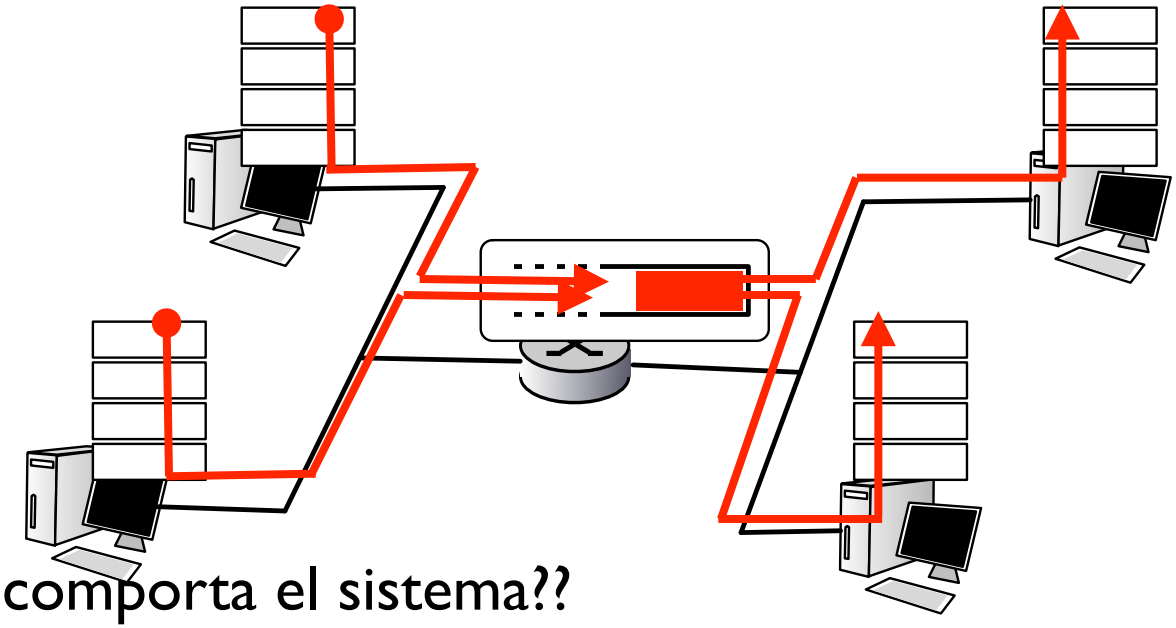
Congestión de red

¿Qué es?

- Demasiadas fuentes enviando demasiado trafico para que la red pueda manejarlo
- No es lo mismo que el control de flujo
  - Relacionado con que el receptor pueda manejar el tráfico
  - La congestión puede darse aunque todos los receptores sean capaces de aceptar el trafico que se está enviando
- Efectos:
  - Pérdidas de paquetes (debido a desbordamiento de colas de routers)
  - Tiempos de espera elevados (debido a tiempos de espera en colas altos)
- Es uno de los problemas más importantes de redes

Escenario 1

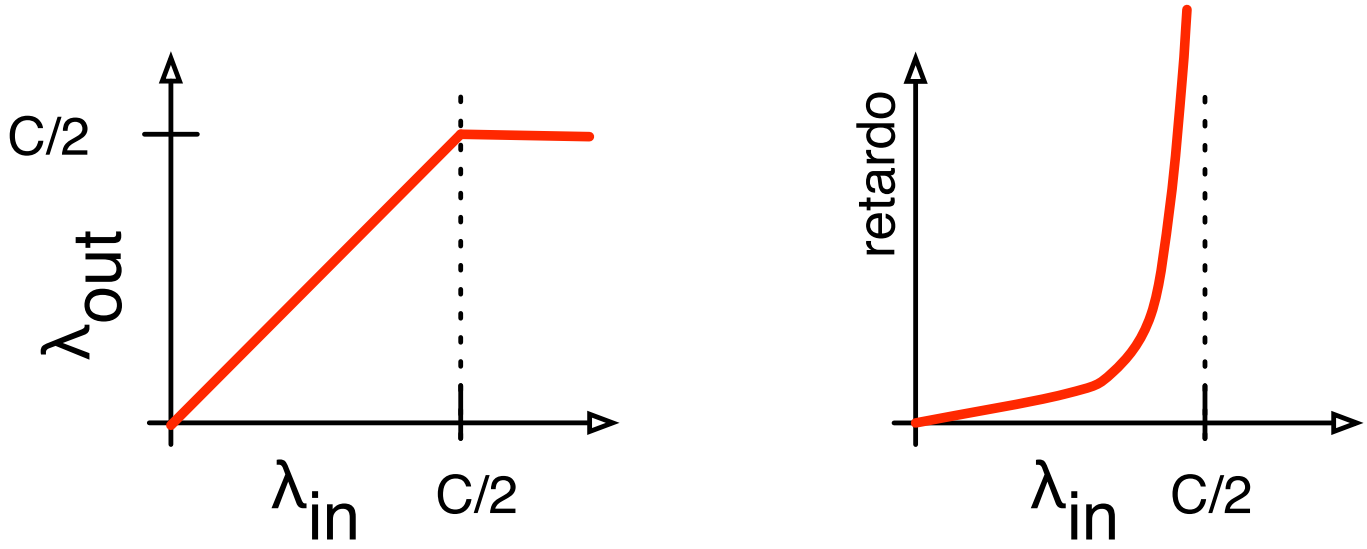
- Dos emisores y dos receptores
- Un router común (supongamos que con buffer infinito)
  - Nunca ocurren retransmisiones
  - Las dos fuentes envian a una media de λ_{in}
  - La capacidad del router es C



- Como se comporta el sistema??

Escenario 1

- **Entrada y salida**. Cuanto throughput obtenemos del sistema?
  λ_{out} para cada λ_{in} aceptable (con un scheduler perfecto)

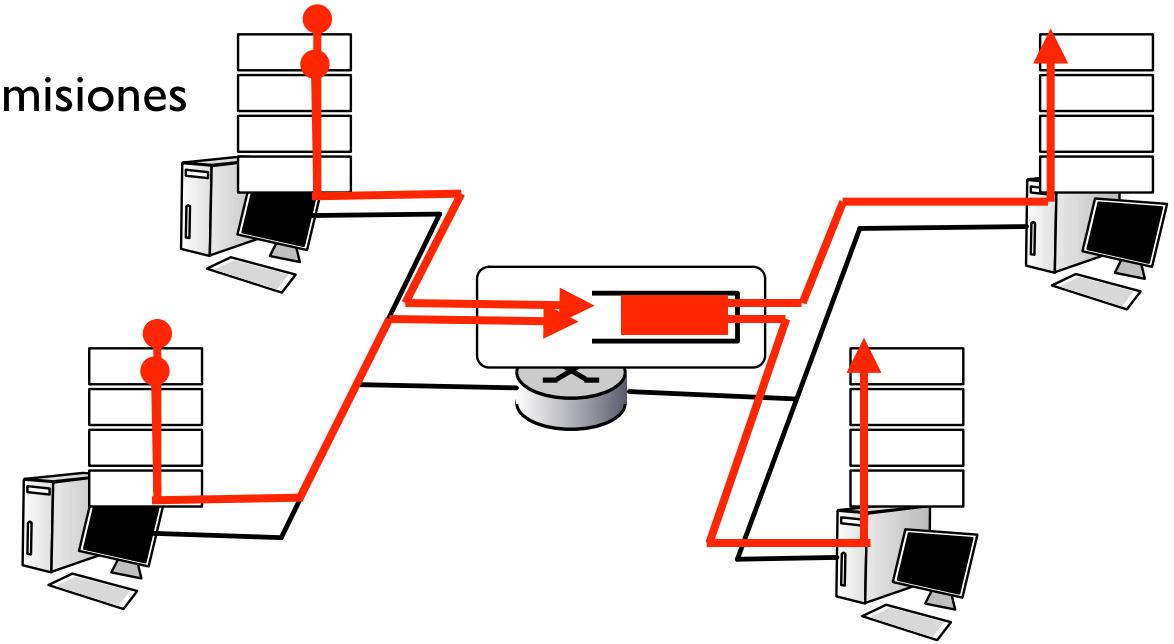


- Pero el retardo aumenta indefinidamente
- Cada vez tiene más paquetes que procesar
- Recursos infinitos no garantizan un buen funcionamiento

Escenario 2

- Igual que el anterior pero ahora el router tiene recursos finitos.
  - Los paquetes no esperan indefinidamente (retardo limitado)
  - Si se transmite a mucha velocidad algunos paquetes se perderan y el nivel de transporte los retransmitirá
  - Tráfico

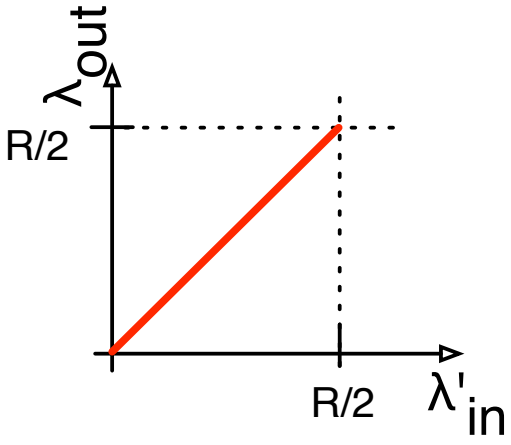
    $\lambda'_{in} = \lambda_{in}$ + retransmisiones



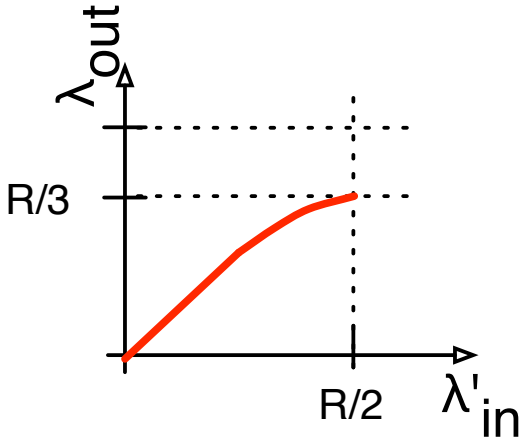
Escenario 2

- $\lambda_{in} = \lambda_{out}$ (goodput)
- Envío perfecto (no hay retransmisiones hasta que alcancemos R/2)

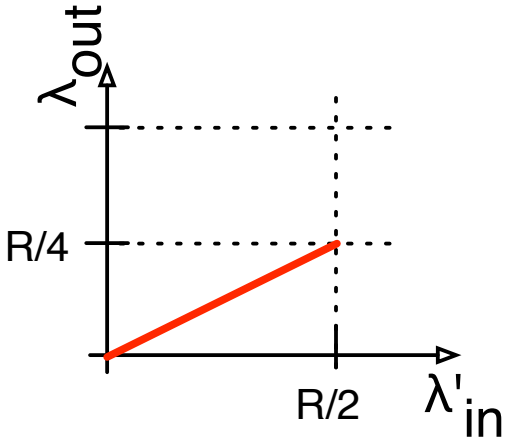
  Retransmisión implica $\lambda'_{in} > \lambda_{out}$



Perfecto
no retransmisiones



Retransmisiones sólo por desbordamiento

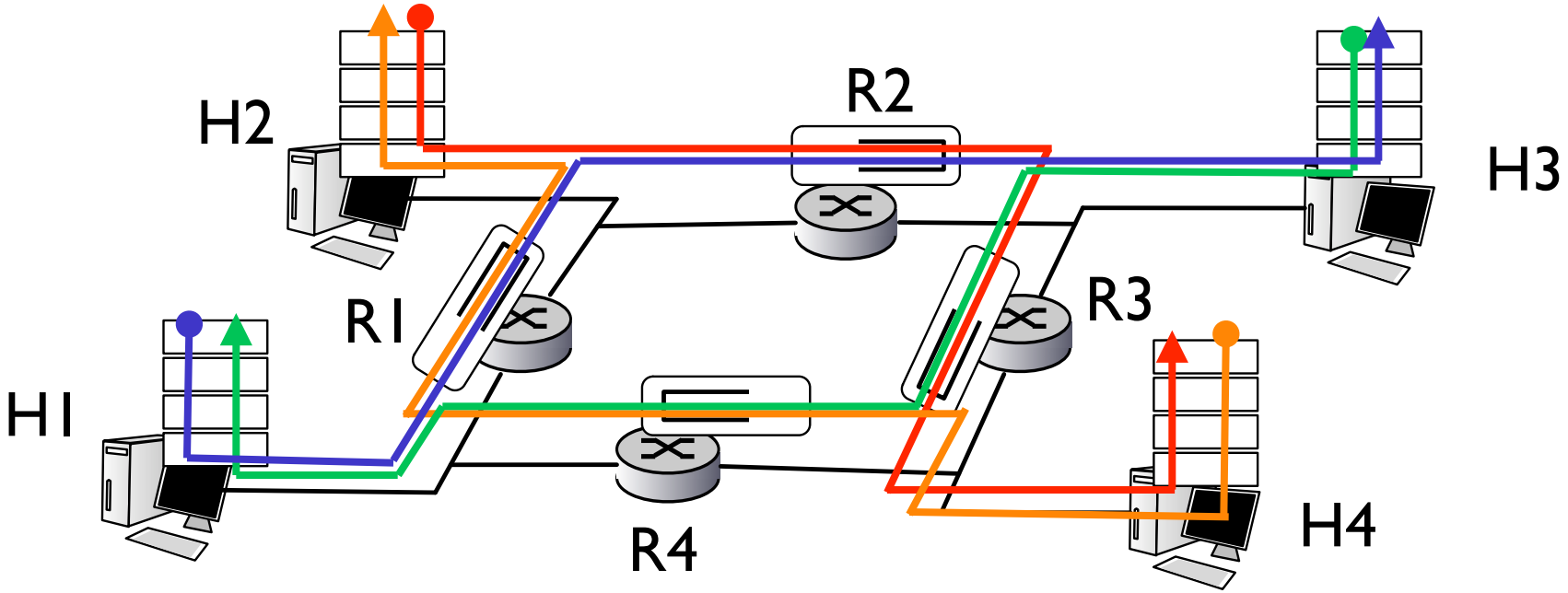


Retransmisiones por timeout prematuro

- Coste de la congestión
  - Más trabajo (retransmisiones) para el mismo resultado (goodput)
  - Retransmisiones innecesarias: capacidad perdida

Escenario 3

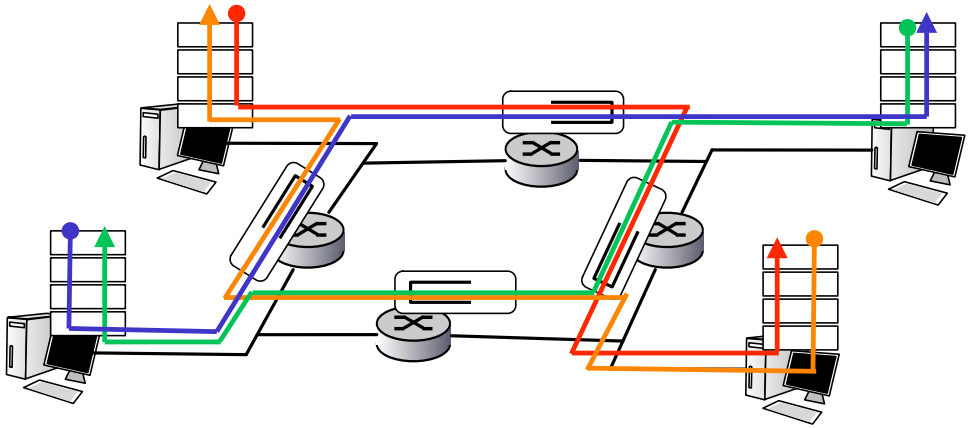
- 4 fuentes, buffers finitos y múltiples caminos
- Retransmisiones por timeout
- ¿Qué pasa al aumentar λ_{in} y por tanto λ'_{in} ?

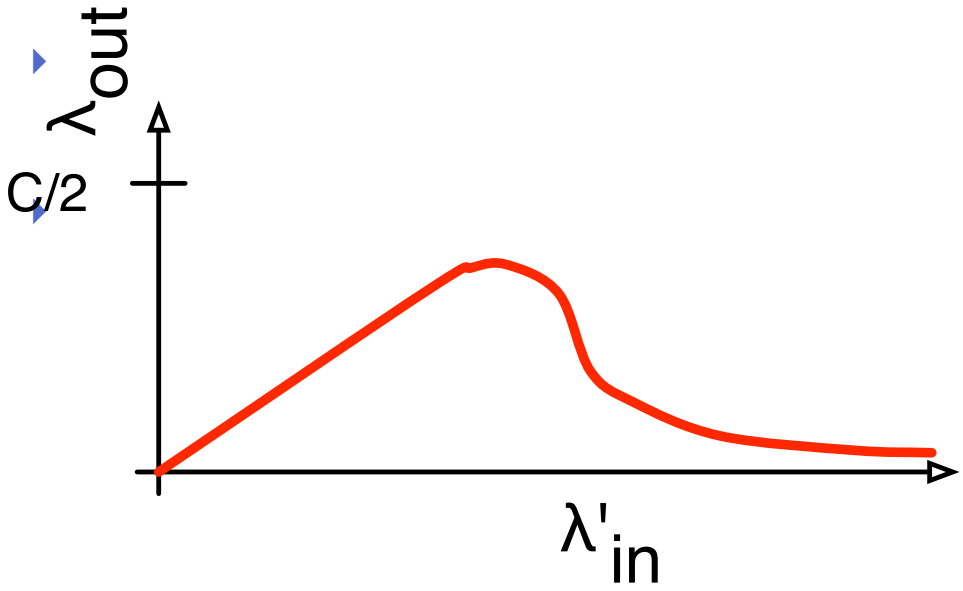


Escenario 3

- El trafico que entra a R2 proveniente de R1 tiene límite R

  Cuando la carga es muy alta R2 esta saturado por el trafico de H2



- 

Con carga alta el trafico util que se cursa tiende a 0

Cuando se descarta un paquete la capacidad usada para llevarlo hasta ese punto se desperdicia

Control de congestión

- **Problema**:

  cuando la red está saturada por alta carga proveniente de muchas fuentes

  se pierden paquetes, se cursa menos tráfico útil y aumenta el retardo de los paquetes que llegan a su destino

  Los protocolos de transporte **reaccionan aumentando las retransmisiones causando más carga y más congestión**

- Si estamos en esta situación es más rentable que los emisores no envíen tan rápido para mantenerse por debajo de esta carga total...

  Como conseguimos esto?

  Qué pasa si la mayoría hacen eso pero unos pocos no?

Control de congestión

Dos enfoques para control de congestión

- Control de congestión extremo a extremo
  - No hay indicación explicita de la congestión
  - Los extremos estiman la congestión basandose en sus propias observaciones de la red
    - Perdidas observadas
    - Retardo observado
  - Los extremos reacciónan a la congestión reduciendo la carga: disminuyendo retransmisiones, tasa de envíos, aumentando timeouts...
  - **Esta es la filosofía que utiliza TCP (clasico)**

Control de congestión

Dos enfoques para control de congestión

- Control de congestión asistido por la red
  - Los routers informan de la congestión a los extremos de la comunicación
    - Modificando un bit para indicar congestión:

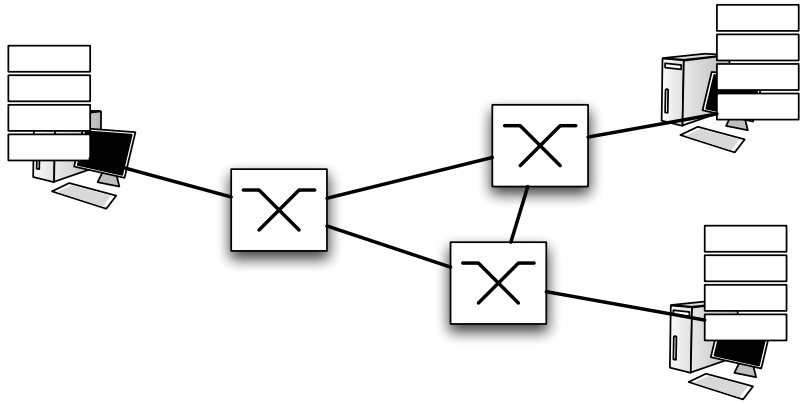
      SNA, DECnet, **TCP/IP ECN (RFC2481)**, ATM
    - Indicando la tasa máxima a la que puede enviar

      ATM ABR

    - Remember **ECN: Explicit Congestion Notification**

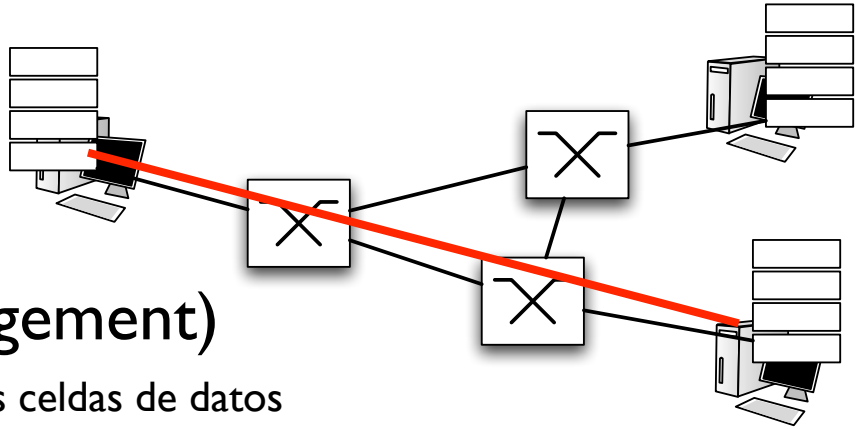
Ejemplo

- Control de congestión asistido por la red en ATM
- ATM red de comunicaciones para transporte de RDSI de banda ancha
  - Filosofía de conmutación de paquetes con paquetes de tamaño pequeño y fijo (celdas) y con circuitos virtuales
  - Orientado a conexión
  - Estado de las conexiones en cada nodo (switch)



Control de congestión en ABR

- ABR (Available Bit Rate)
  - Si el camino está descargado puede usar más capacidad
  - Si el camino está cargado el emisor debe limitarse a una capacidad mínima garantizada



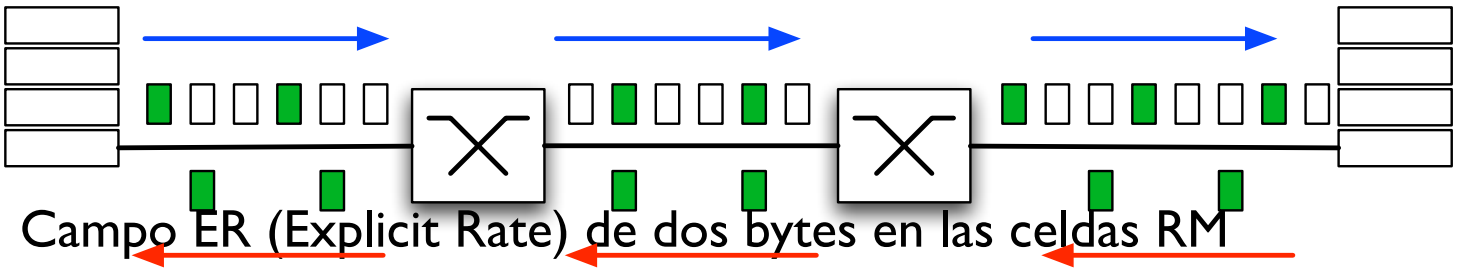
- Celdas RM (resource management)
  - Enviadas por el emisor mezcladas con las celdas de datos
  - Bits en estas celdas permiten indicar la congestión de red
    - NI bit **No Increase rate** Indica congestión moderada
    - CI bit **Congestion Indication**
  - El emisor devuelve estas celdas sin tocar esos bits

Control de congestión en ABR

- Varios mecanismos
  - Bit EFCI en las celdas de datos (Explicit Forward Congestion Indication).

    El destino reacciona devolviendo celdas RM con el bit CI a 1
  - Celdas RM con bits NI y CI

    El destino las devueve sin tocar los bits (salvo poner CI)
  - Los switchs pueden incluso generar RMs hacia atras



  - Campo ER (Explicit Rate) de dos bytes en las celdas RM

    Los switchs del camino pueden modificar el valor

Control de congestión en Internet

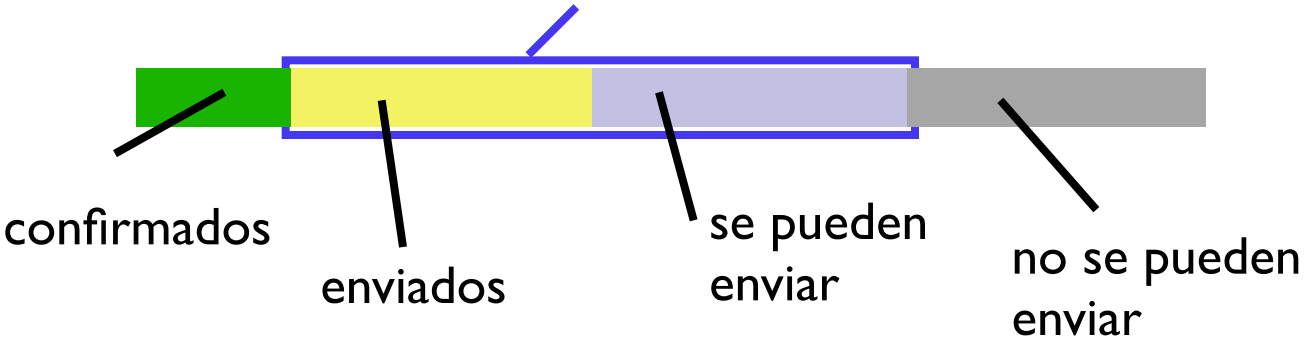
- Pero Internet se basa en un nivel de red simple que no asiste en la detección de la congestión
- TCP (originalmente) utiliza control de congestión extremo a extremo
  - Como inferimos que hay congestión en la red?
    - Perdidas?
    - Retardo?
  - Qué hacemos para evitarlo?
    - Reducir la tasa de envío? Cómo?
    - Aumentar el timeout de retransmisión?
- Si hay errores bajar la tasa de transferencia?
- Algo más?

TCP: Control de congestión

- Usa control de congestión extremo a extremo
  - La ventana deslizante de transporte fiable tenia un limite máximo impuesto por el control de flujo igual a la ventana anunciada por el receptor
  - Se utiliza otra **ventana de congestión** que limita los datos que se pueden enviar a la red dependiendo de la percepción que tiene TCP de la congestión
  - La ventana anunciada depende del receptor
  - La ventana de congestión se reduce ante la congestión limitando la tasa a la que enviamos

$$v = \frac{CongWin}{RTT}$$

Ventana = min { ventana anunciada, ventana de congestión }



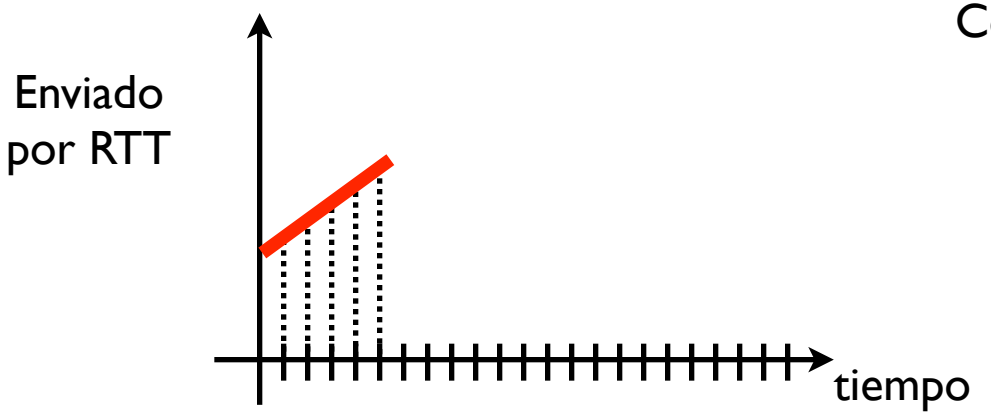
TCP: Control de congestión

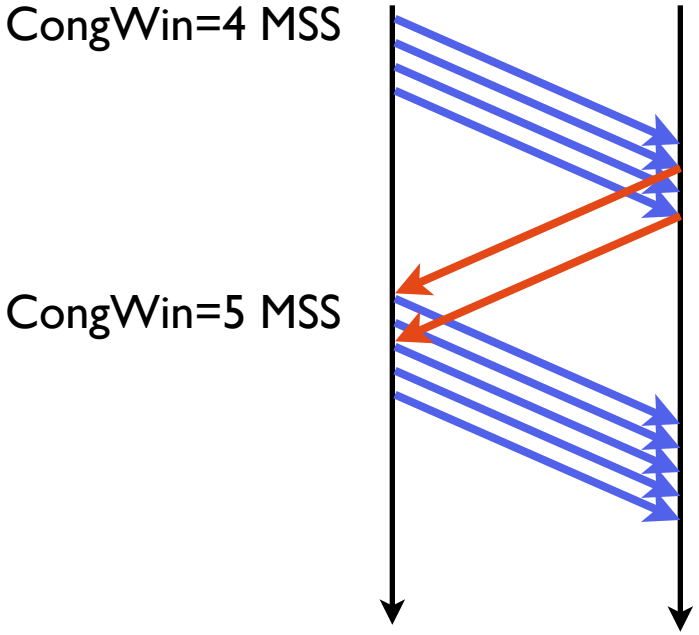
- ¿Cómo percibe TCP la congestión?
  - Perdida de paquetes = congestión
    - Evento de timeout
    - 3 ACKs duplicados (Fast retransmit)
- Ajustando la ventana de congestión
  - Congestion avoidance (evitación de congestion)
  - Slow start
  - Fast recovery
  - Timeout
- MSS: maximun segment size
  - Aun cuando TCP utiliza secuencias y ACKs por bytes individuales cuando tiene mucho que enviar utiliza un máximo tamaño de segmento, que llamaremos MSS
  - Negociado en la apertura de conexión

TCP: Congestion avoidance

- Tras detectar congestión TCP entra en una fase de evitación de congestión (congestion avoidance)
  - Si la ventana de congestión tiene un valor de N MSS y es menor que la ventana anunciada por el receptor
  - En **congestion avoidance** se abre 1 MSS cada vez que enviamos con exito toda la ventana
  - La ventana sube linealmente

    Buscando encontrar el punto de equilibrio sin aumentar demasiado la congestión



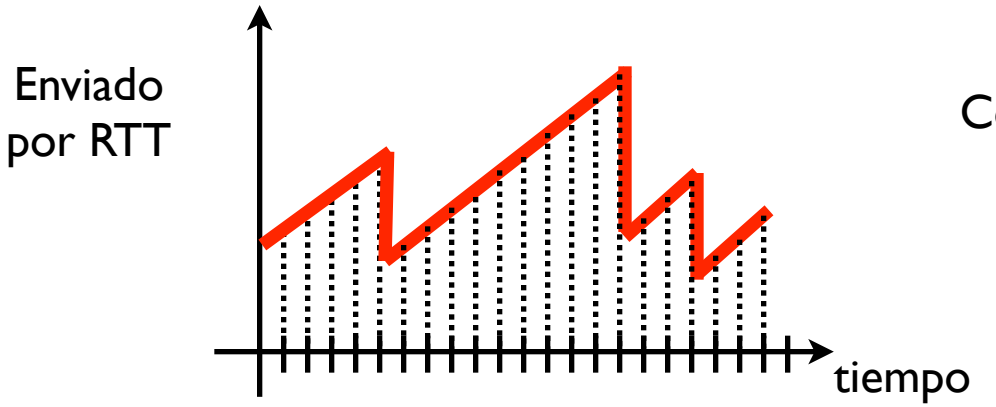


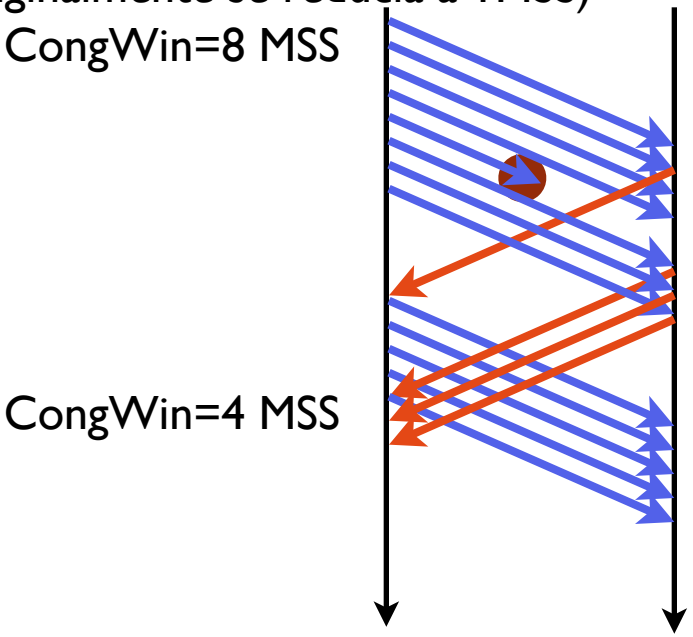
TCP: Congestion avoidance

- Si en esta situación se produce una pérdida
  - De un sólo paquete, lo que provoca 3 ACKs duplicados. Se interpreta como congestión ligera
  - La ventana se reduce inmediatamente a la mitad

    A la vez que se hace **fast retrasmit** del paquete perdido

    Esto se conoce como **fast recovery** (originalmente se reducía a 1MSS)
  - Buscando reducir notablemente la tasa de transmisión y colaborar en que la congestión no crezca
  - Si se siguen recibiendo ACKs la ventana sigue creciendo linealmente

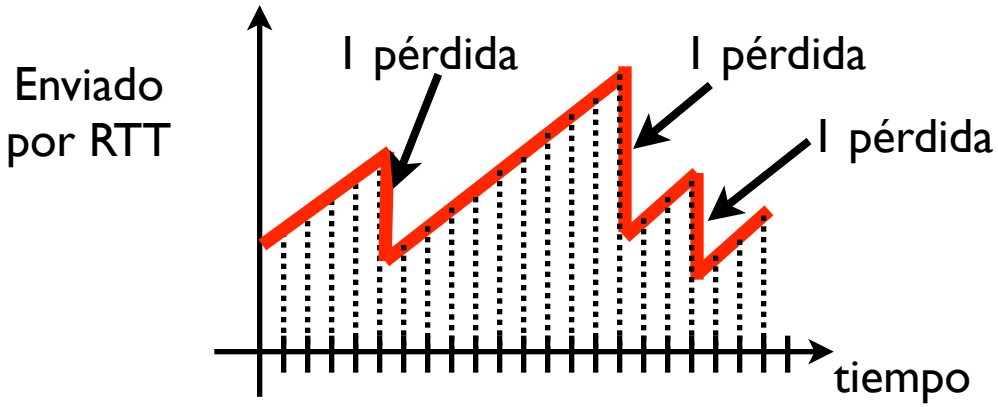




TCP: Congestion avoidance

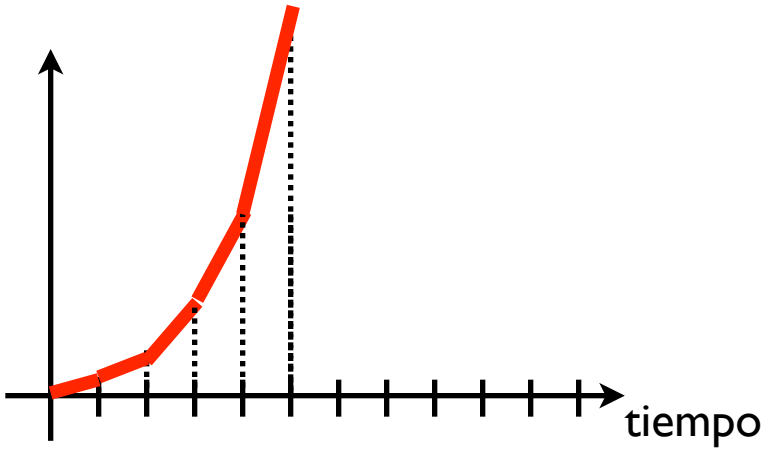
- Se puede ver que la tasa se va ajustando alrededor del punto de congestión
  - si hay muchos errores la tasa baja y se tarda más en recuperarla
  - si hay pocos errores el crecimiento lineal es capaz de llegar mas a mas velocidad
  - Este mecanismo se suele llamar **AIMD**

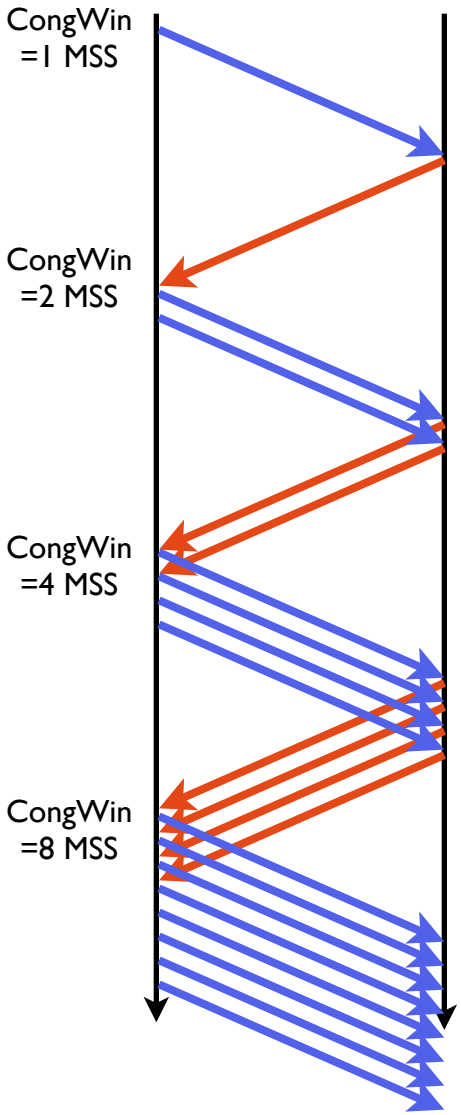
    Additive Increase Multiplicative Decrease
- Y cómo empieza la conexión??
  - Con CongWin = 1 MSS



TCP: Slow Start

- En el principio CongWin=1 MSS pero...
  - Crecer linealmente es demasiado precavido, no tenemos motivos para pensar que hay congestión
  - Se utiliza un enfoque más agresivo, crecimiento exponencial
  - **Slow Start**: cada vez que transmitimos una ventana con éxito la ventana se dobla

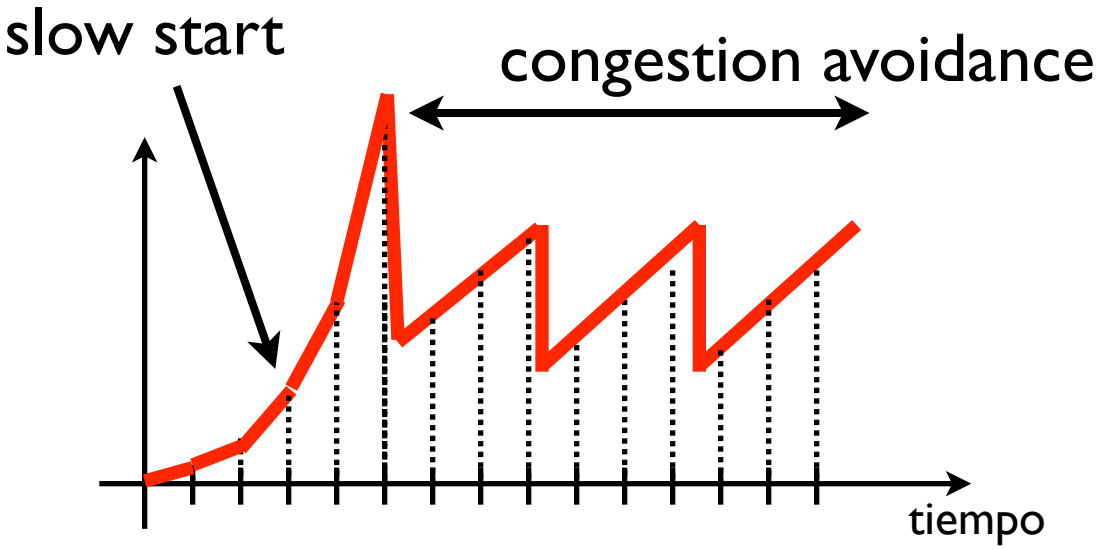




TCP: Slow Start

- El slow start se mantiene hasta que se produce una pérdida haciendo entrar en congestion avoidance

  (o bien se alcanza la ventana de control de flujo)
  - Si la perdida se detecta por ACKs duplicados...
    - Fast retransmit + fast recovery CongWin = CongWin/2
- Y si se produce un timeout?

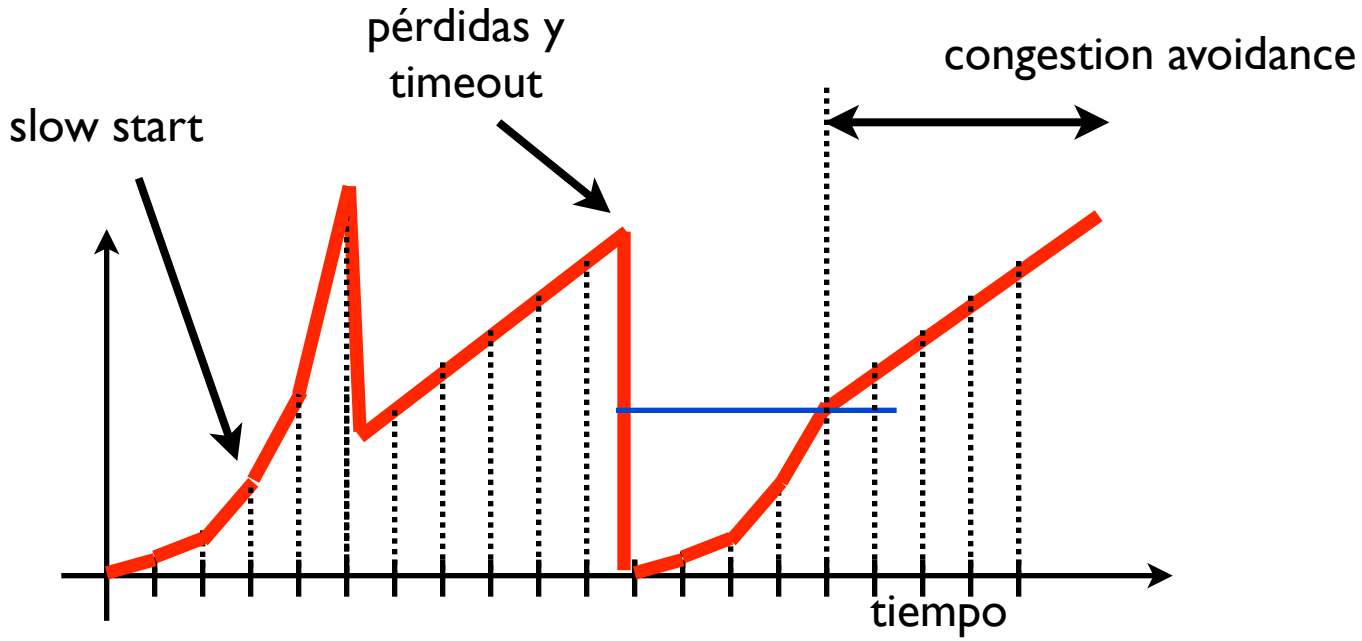


TCP: pérdidas y congestión

- Pérdida detectada por ACK duplicados
  - Congestión "ligera": hay perdidas pero siguen llegando paquetes
  - Acciones:
    - Retransmitir el paquete perdido (Fast retransmit)
    - Bajar la ventana de congestion para reducir la tasa
    - Pasar a congestion avoidance (si no estabas)
- Pérdida detectada por timeout
  - Congestión "grave". Probablemente hemos perdido toda la ventana. Si el timeout ha caducado llevamos un rato parados sin transmitir. Tasa =0
  - Acciones:
    - Poner la ventana de congestión a 1MSS
    - Pasar a slow start
    - Recordar a cuanto estaba la ventana de congestión al producirse el error (poner la variable threshold=CongWin/2)

TCP: slow start

- Despues de una perdida por timeout
  - el slow start no espera a otra pérdida para entrar en congestion avoidance.
  - Pasa a congestion avoidance en cuanto llega al humbral de la mitad de la ventana de congestion al producirse el timeout



TCP: control de congestión

En resumen

- Cuando CongWin está por debajo de Threshold. **Slow start**. La ventana crece exponencialmente
- Cuando CongWin está por encima de Threshold. **Congestion avoidance**. La ventana crece linealmente
- Cuando ocurre un **triple ACK duplicado**. Threshold se pone a CongWin/2 y CongWin a Threshold
- Cuando ocurre un **timeout**. Threshold se pone a CongWin/2 y CongWin se pone a 1MSS

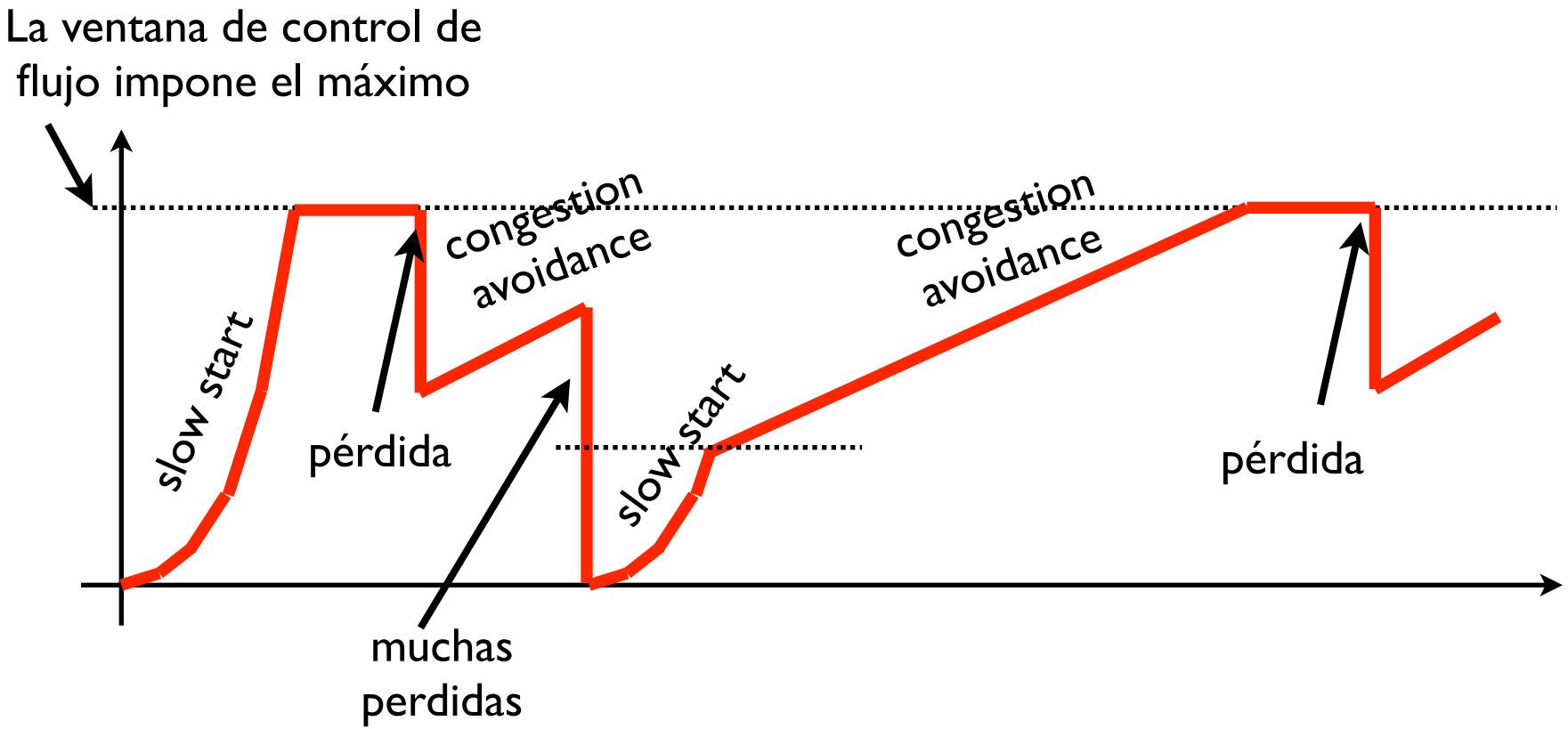
TCP: control de congestión

- Eventos en el emisor

Evento	Estado	Acción	Notas
Recibe ACK para datos no confirmados	Slow Start (SS)	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	CongWin se dobla por cada RTT
Recibe ACK para datos no confirmados	Congestion Avoidance (CA)	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, CongWin aumenta 1 MSS por cada RTT
Perdida detectada por triple ACK duplicado	SS or CA	Threshold = CongWin/2, CongWin = Threshold, Set state to “Congestion Avoidance”	Fast recovery+multiplicative decrease. CongWin nunca baja a menos de 1MSS
Timeout	SS or CA	Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”	vuelve a slow start
ACK duplicado	SS or CA	Incrementar contador de ACKs duplicados para ese segmento	CongWin y Threshold no cambian

TCP: evolución de la conexión

- Ejemplo, una conexión TCP

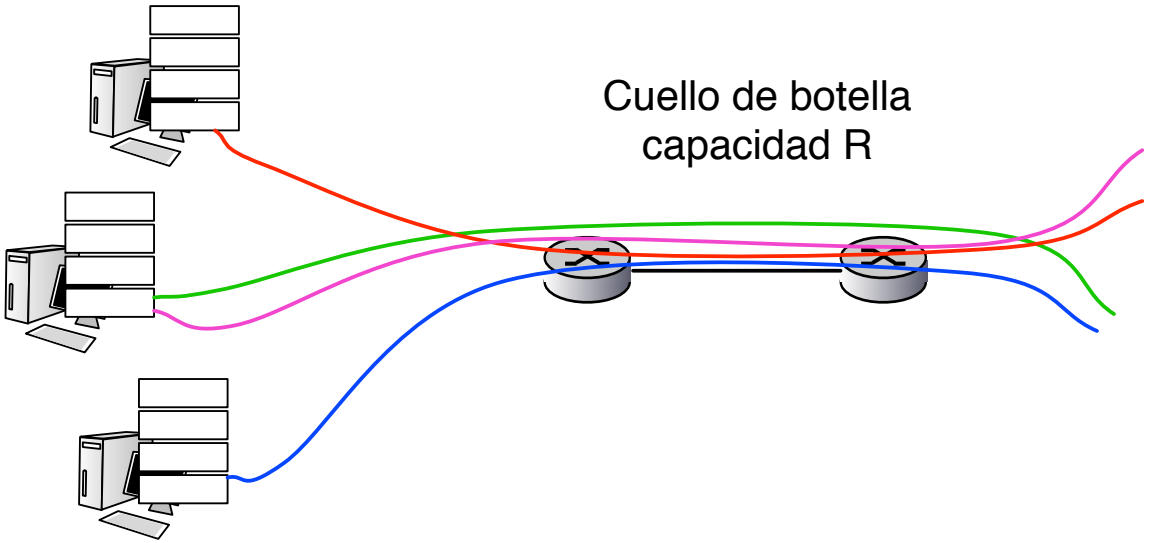


TCP: eficiencia de la conexión

- Sigue el límite de AdvertisedWindow/RTT
- Esto es para una conexión que este siempre enviando
  - Tenga siempre datos en el buffer de emisión
  - La aplicación del receptor lea los datos suficientemente rápido como para que el emisor siempre anuncie la ventana máxima
- Cómo podemos transmitir a mayor velocidad?
  - Mejorando TCP para que permita ventanas mayores (próxima clase)
  - Usando más de una conexión TCP??
  - UDP tiene control de flujo??

TCP: equidad (fairness)

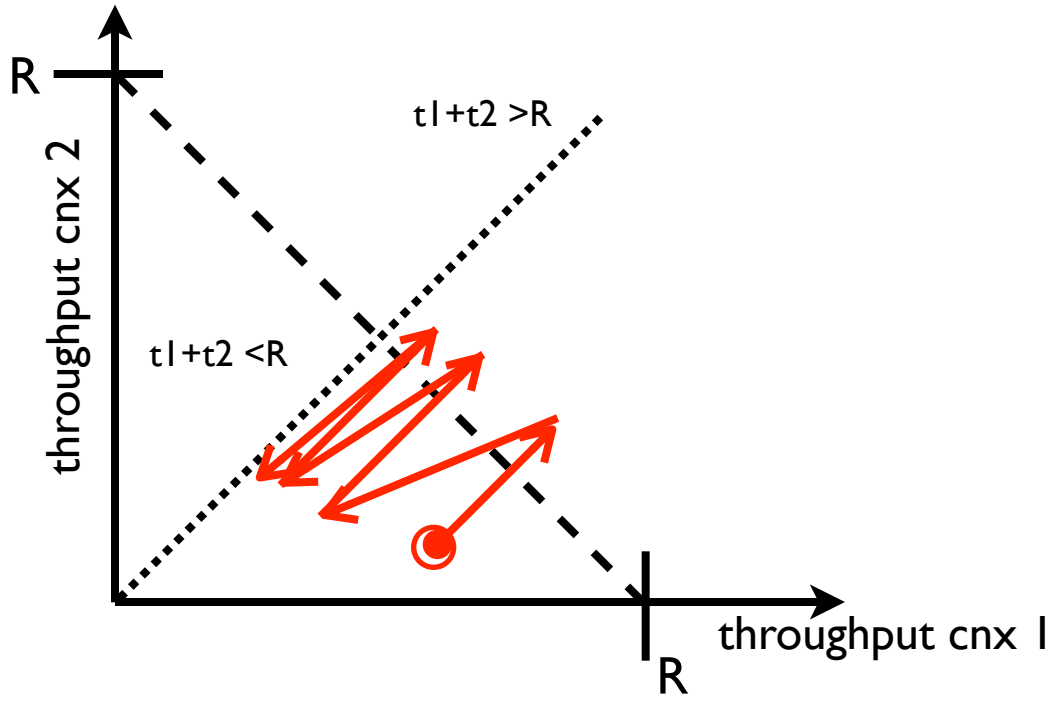
- Equidad para TCP
  - Si N conexiones TCP comparten un enlace la capacidad del enlace R debería repartirse entre todas por igual R/N



TCP: equidad (fairness)

- Por qué es TCP equitativo?
  - 2 sesiones compitiendo por el ancho de banda
  - Modelo simple. Si la capacidad de las dos suma mas que R habra perdidas
  - Additive increase: sube linealmente el throughput,

    Multiplicative decrease: si hay perdidas baja rapidamente



Más sobre equidad

- ¿Qué pasa con UDP?
  - Las aplicaciones multimedia suelen usar UDP para evitar el control de congestión.
  - Envían a tasa constante y no quieren que esta se vea reducida
  - Son capaces de tolerar pérdidas
- Sin embargo
  - Es dificil garantizar la equidad en ese ambiente TCP+UDP
  - El control de congestión de TCP es justo si compite con otros TCPs
  - Esto es un área de investigación activa
    - UDP que sea TCP friendly?
    - Envio de multimedia sobre TCP?

Más sobre equidad

- Las aplicaciones pueden abrir más de una conexión TCP entre dos hosts
  - Así podrían superar la limitación de la ventana de control de flujo de 65KB
  - Los navegadores web hacen esto
  - Parte de la mejora de transferencia de los sistemas P2P viene de este efecto !!
- Sin embargo
  - Esto también dificulta el problema de la equidad en TCP
  - Si en un enlace de capacidad R hay 9 conexiónes TCP
    - Puedo abrir una conexión y conseguir un reparto de R/10
    - O puedo abrir 11 y conseguir R/2 !!!

Conclusiones

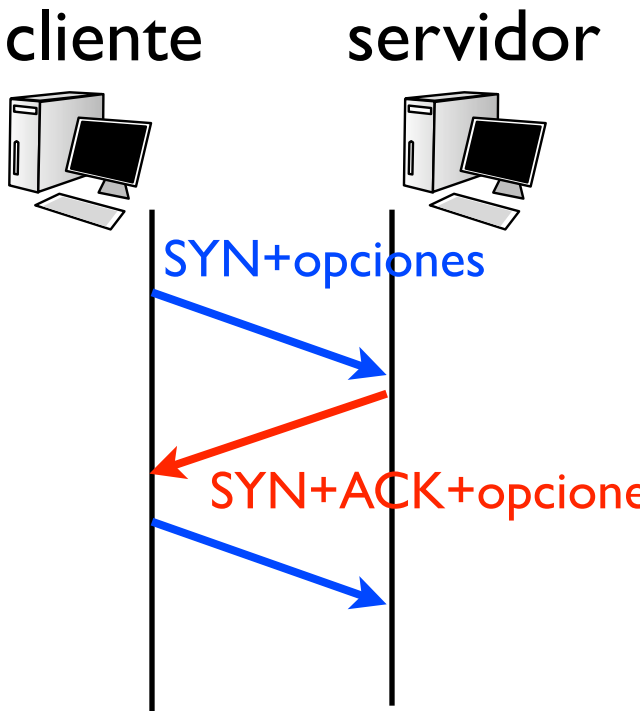
- El control de congestión TCP se basa en una serie de principios simples
  - ventana de congestion
  - AIMD + slow start para el principio
  - reacción a los ACKs duplicados (congestion ligera) y timeouts (congestion severa)
- TCP reparte la capacidad de forma equitativa
  - Pero sigue habiendo problemas en una Internet en la que no todo es TCP
- Las prestaciones de Internet en gran medida son las prestaciones de TCP

Extras

TCP: opciones

- La cabecera TCP reserva espacio para indicar opciones
  - Estas opciones permiten especificar comportamientos opcionales que pueden aparecer en diferentes versiones del protocolo
  - Unas cuantas de estas opciones se utilizan para negociar si la conexión utilizara determinados metodos

    Para ello el cliente en el SYN indica que opciones soporta y el servidor contesta en el SYN cuales acepta
  - Ejemplo: peticion de www.google.com:




```
IP 130.206.169.177.50641 > 66.249.87.104.80: S 0:0(0) win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 465116705 0>
IP 66.249.87.104.80 > 130.206.169.177.50641: S 0:0(0) ack 1 win 8190 <mss 1460>
IP 130.206.169.177.50641 > 66.249.87.104.80: . ack 1 win 65535
IP 130.206.169.177.50641 > 66.249.87.104.80: P 1:335(334) ack 1 win 65535
```

S **<mss 1460,nop,wscale 0,nop,nop,timestamp 465116705 0>**

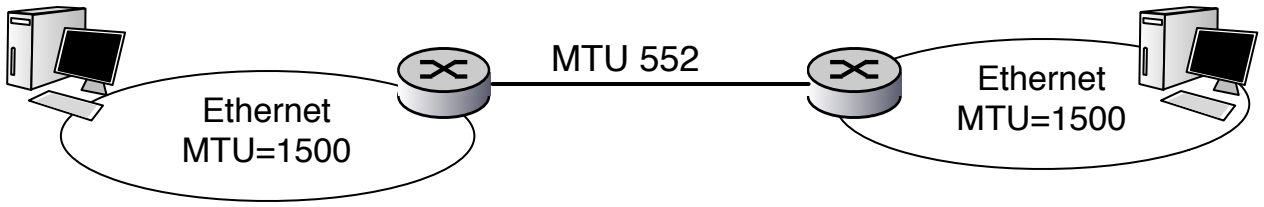
S+ACK **<mss 1460>**

TCP: path MTU discovery

- Ya habiamos visto la opción MSS

  S `<mss 1460,nop,wscale 0,nop,nop,timestamp 465116705 0>`
  S+ACK `<mss 1460>`

- Cada extremo anuncia el MSS que espera recibir
  - Para ello utiliza el valor máximo de paquete que permite su red de area local o interfaz de red. MTU, maximum transfer unit
  - Si, un extremo no anuncia un MSS se toma por defecto 536
- Pero no siempre se puede deducir el MTU de un camino a partir de los MTUs que ven los extremos

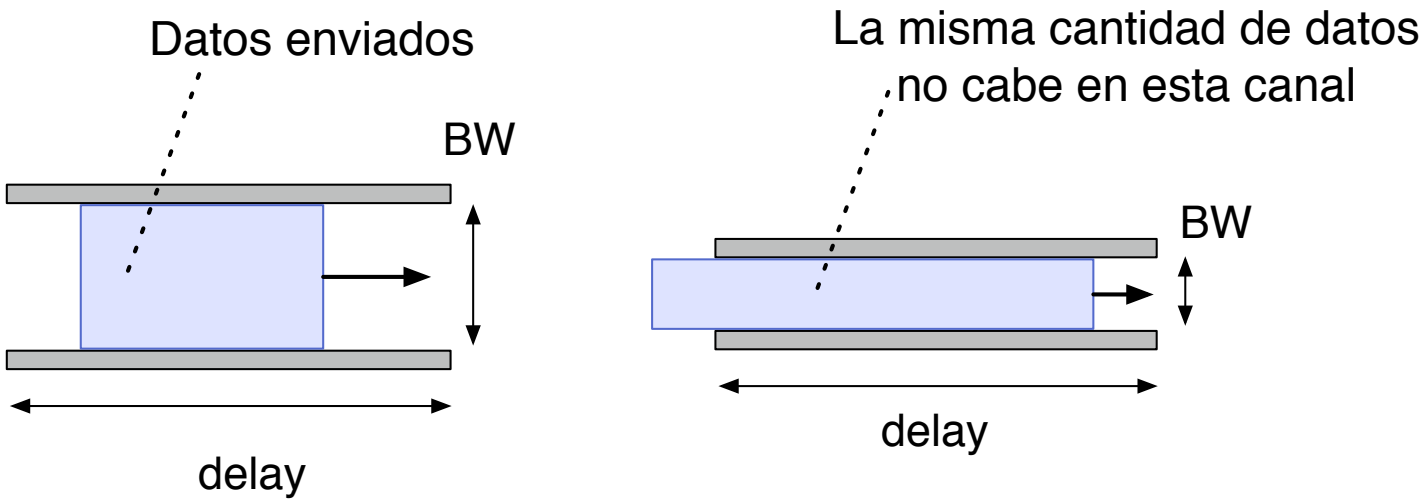


TCP: path MTU discovery

- Después de la negociación TCP utiliza el mínimo MSS
- Envía los paquetes de datos indicando al nivel de red que no acepte fragmentación: Bit Don'tFragment si tiene que fragmentar descarta
- Si los routers del camino deben fragmentar envían un paquete informando del error y TCP baja el MSS y reintenta
- Estos errores no disminuyen la ventana de congestión, aunque si reinician el slow start
- El proceso se repite periódicamente por si los cambios de ruta

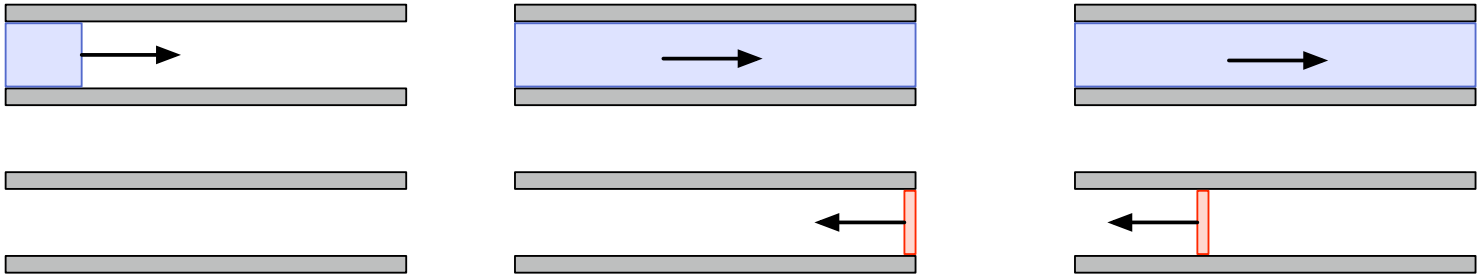
TCP: alta velocidad

- TCP es capaz de transmitir en condiciones muy desfavorables
- Sin embargo muestra algunos problemas para aprovechar las condiciones demasiado buenas
- Producto ancho de banda por retardo de un canal
  - bytes que es capaz de almacenar el canal
  - para aprovechar el canal la ventana de TCP debe de ser mayor que el producto BWxRTT

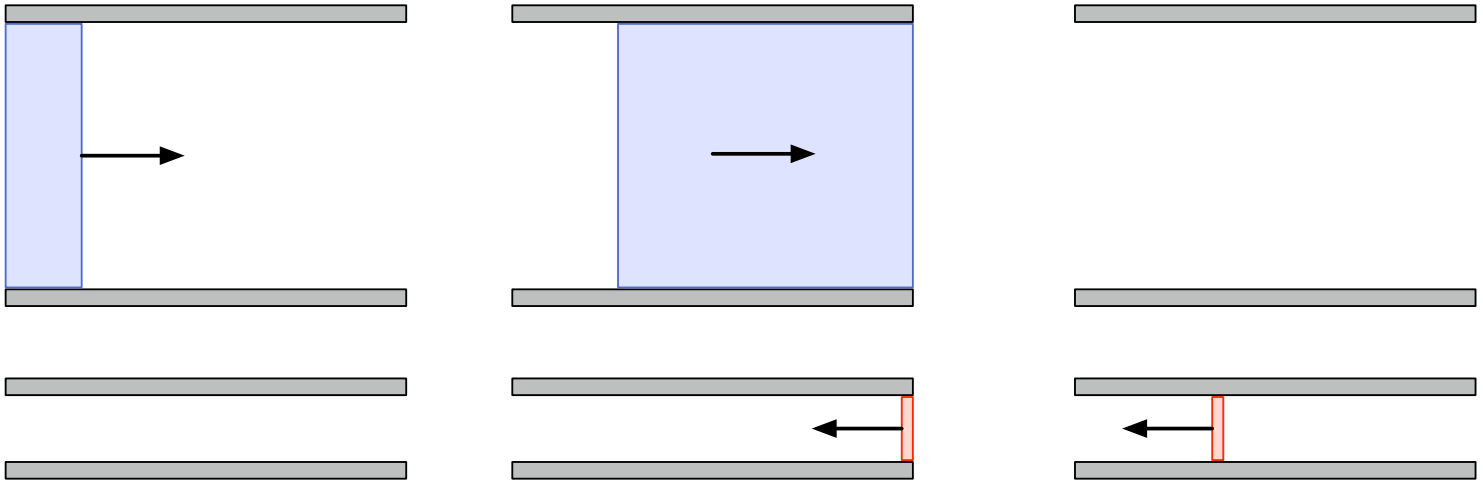


TCP: alta velocidad

- Canal con BWxRTT < window



- Canal con BWxRTT > window



TCP: alta velocidad

- Solo podemos aprovechar un canal con BWxRTT menores que la ventana de TCP
- La ventana anunciada máxima de TCP es de 65KB

  (y todo por poner un campo de solo 16bits)
- Los canales de elevado BWxRTT se les suele llamar LFN (long fat networks)

Red	BW(bps)	RTT(ms)	BWdelay(bytes)
Ethernet 10Mb	10M	3	3750
T1 transcontinental	1544K	60	11580
T1 satélite	1544K	500	96500
T3 transcontinental	45M	60	337500
Gigabit, transcontinental	1000M	60	7500000

TCP: window scale

- Opción window scale permite utilizar tamaños de ventana mayores
  - Si los dos extremos soportan windowscale negocian una escala en el handshake
  - para esa conexión el campo ventana anunciada indica

$$AdvertisedWindow * 2^{WindowScale}$$

- Pero las LFN siguen siendo problemáticas
  - Una pérdida puede detener la transmisión y vaciar el canal
  - Al utilizar TCP con window scale pueden surgir problemas cuando el número de secuencia se da la vuelta
  - El hacer sólo una estimación de RTT por ventana es un porblema
  - Es el momento de abandonar el Go-back-N por el selective reject??

TCP: RFC 1323

- RFC-1323 propone extensiones a TCP para LFNs
  - Básicamente propone usar WindowScale
  - Y una opcion TIMESTAMP que permite medir RTTs por cada paquete
  - SACK selective ACK TCP habia sido propuesto pero no se incluyo en el RFC1323

- SACK sigue siendo experimental
  Aunque hay implementaciones comerciales

TCP: evolución

- Las versiones de TCP han ido evolucionando manteniendo compatibilidad con las anteriores
- TCP Tahoe
  - AIMD. Sin fast recovery. 3dupACKs ponen la ventana a 1
- TCP Reno
  - AIMD. Fast recovery... El más estandar, es el que hemos explicado
- TCP Vegas
  - Intenta predecir la congestión observando el retardo
- TCP Hybla, TCP Westwood, TCP NewReno, TCP Fast...