Redes de Ordenadores

Control de congestión en TCP

Mikel Izal Azcárate
(mikel.izal@unavarra.es)

En clases anteriores

- TCP y UDP
- TCP
  - Transporte fiable
  - Control de flujo
  - Manejo de conexiones
- El problema de la congestión en redes

Hoy

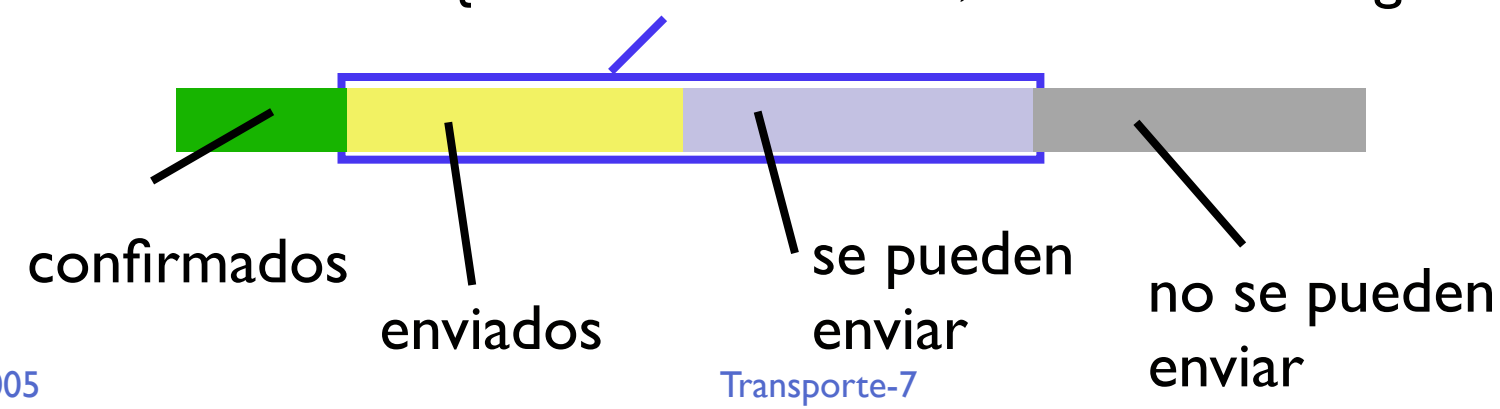
- Control de congestión en TCP

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 }



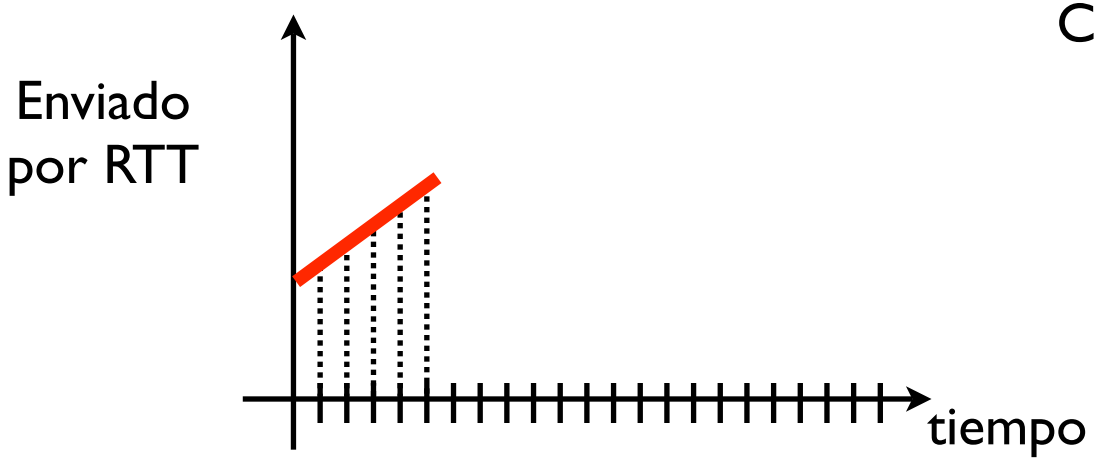
TCP: Control de congestión

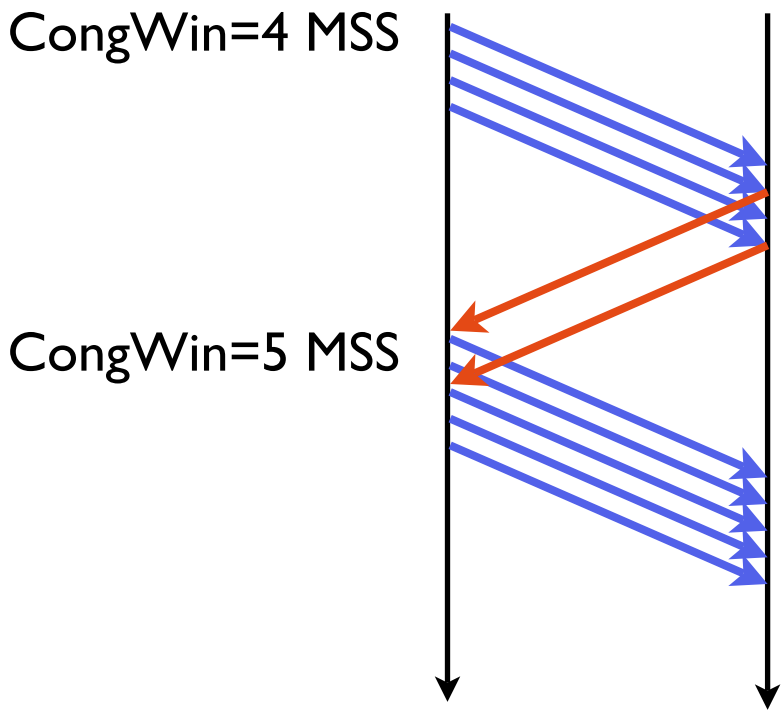
- ¿Cómo percibe TCP la congestión?
  - Perdida de paquetes = congestión
    - Evento de timeout
    - 3 ACKs duplicados (Fast retransmit)
- Ajustando la ventana de congestión
  - Congestion avoidance (evitación de congestion)
  - Slow start
  - Fast recovery
  - Timeout
- MSS: maximun segment size
 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

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
  - 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



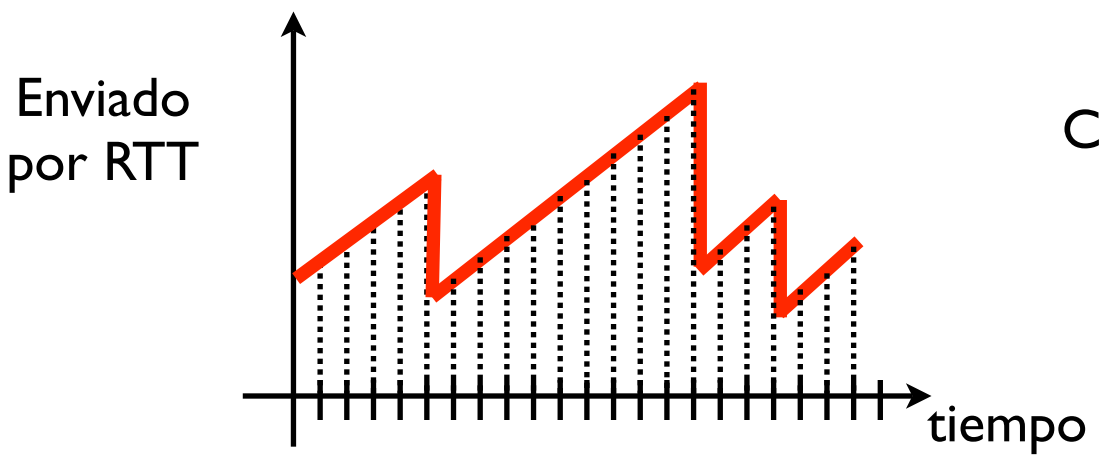


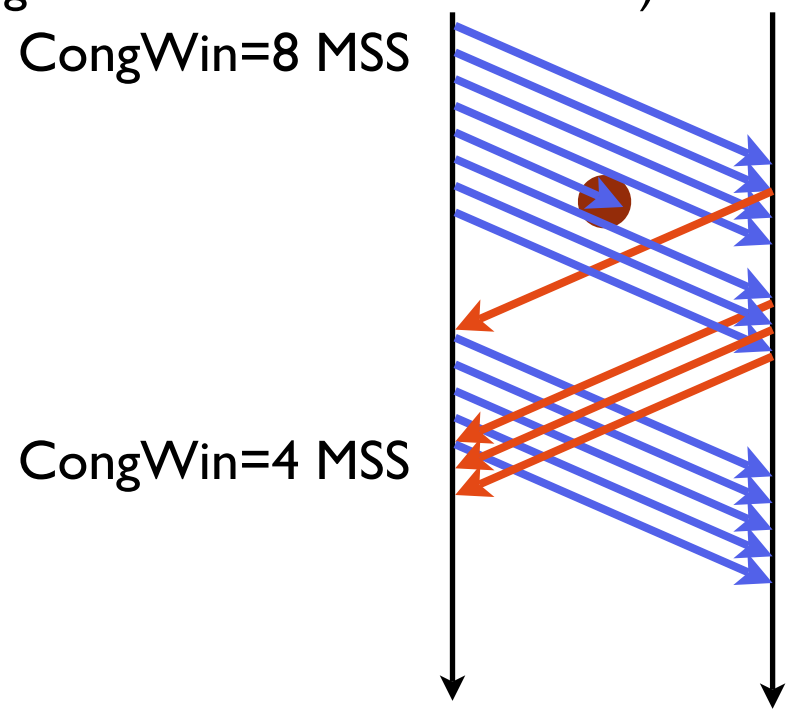
TCP: Congestion avoidance

- 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

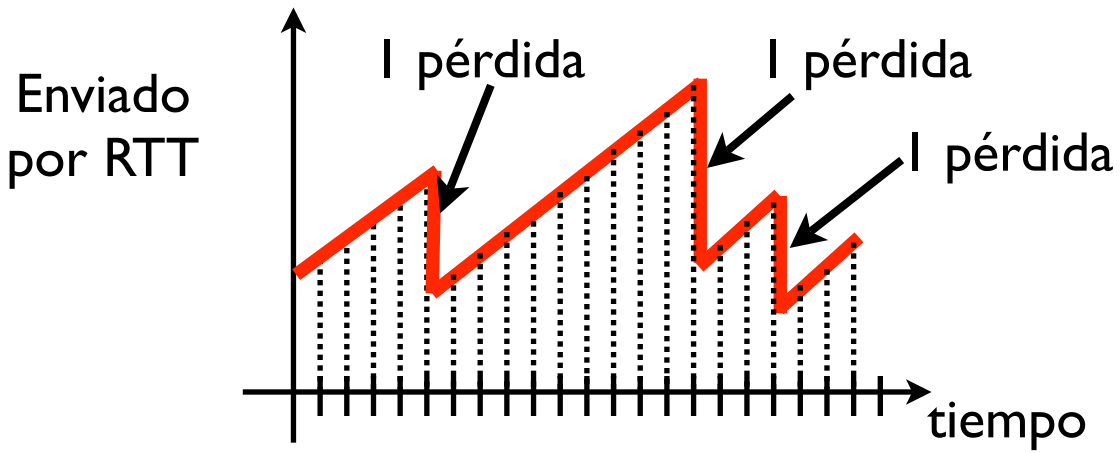




TCP: Congestion avoidance

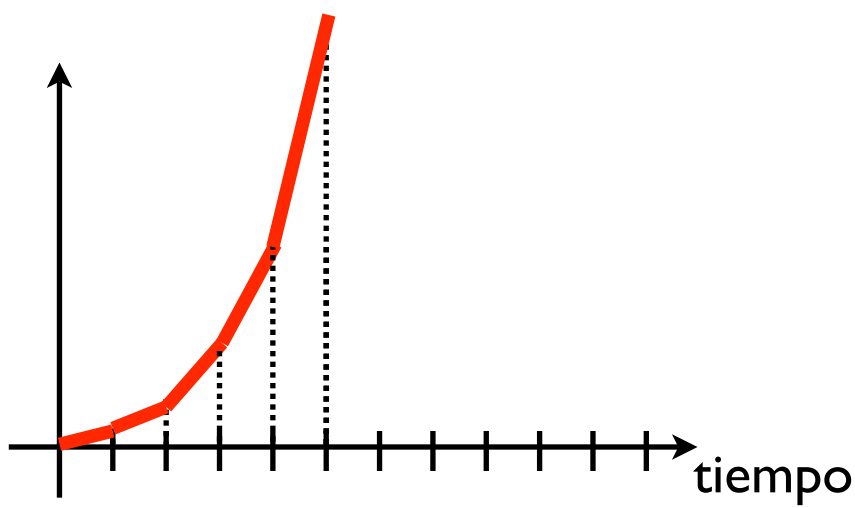
- Se puede ver que la tasa se va ajustando alrededor del punto de congestión
  - si hay muchos errores la tasa baja y se tarda más en recuperarla
  - si hay pocos errores el crecimiento lineal es capaz de llegar mas a mas velocidad
  - Este mecanismo se suele llamar **AIMD**

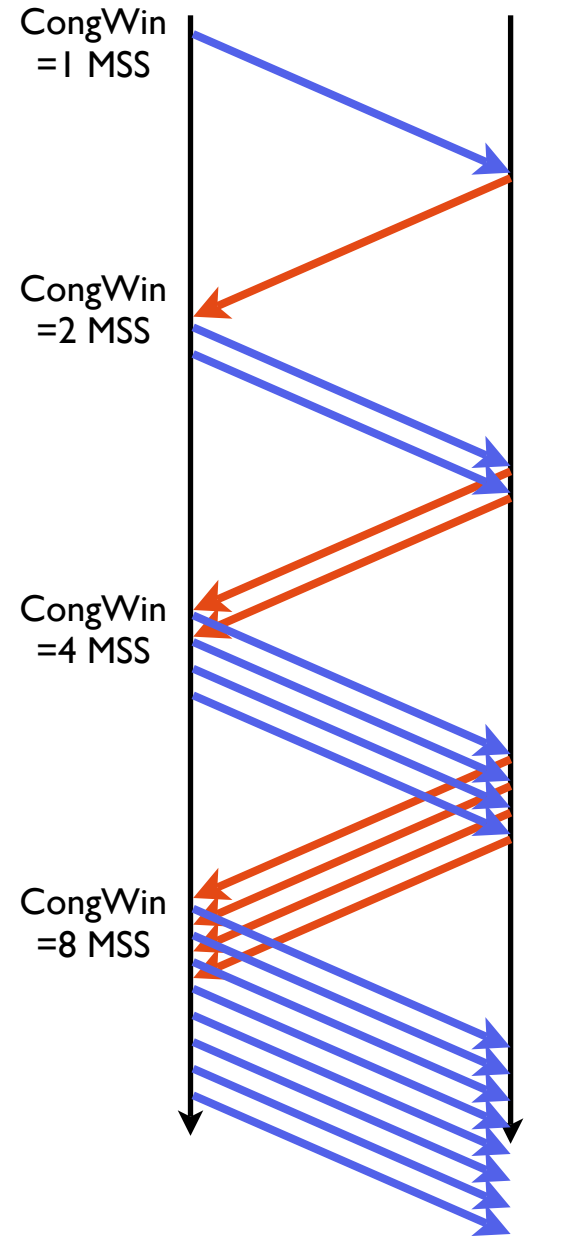
    Additive Increase Multiplicative Decrease
- Y cómo empieza la conexión??
  - Con CongWin = 1 MSS



TCP: Slow Start

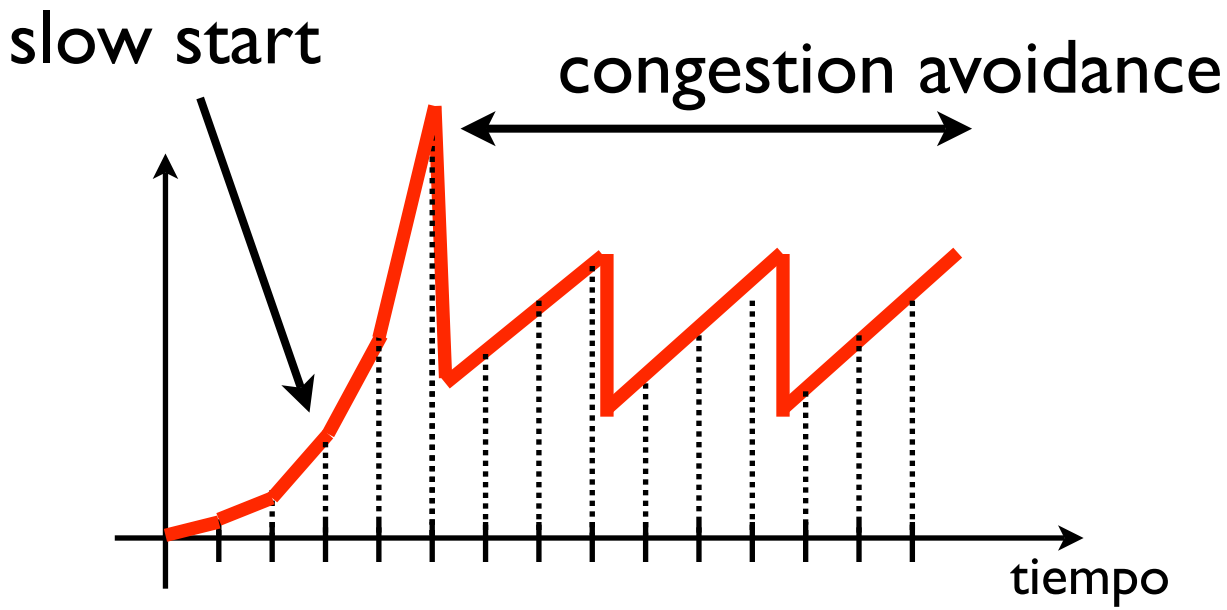
- En el principio CongWin=1 MSS pero...
  - Crecer linealmente es demasiado precavido, no tenemos motivos para pensar que hay congestión
  - Se utiliza un enfoque más agresivo, crecimiento exponencial
  - **Slow Start**: cada vez que transmitimos una ventana con éxito la ventana se dobla





TCP: Slow Start

- El slow start se mantiene hasta que se produce una pérdida (o bien se alcanza la ventan de control de flujo) haciendo entrar en congestion avoidance
  - Si la perdida se detecta por ACKs duplicados...
    - Fast retransmit + fast recovery CongWin = CongWin/2
- Y si se produce un timeout?

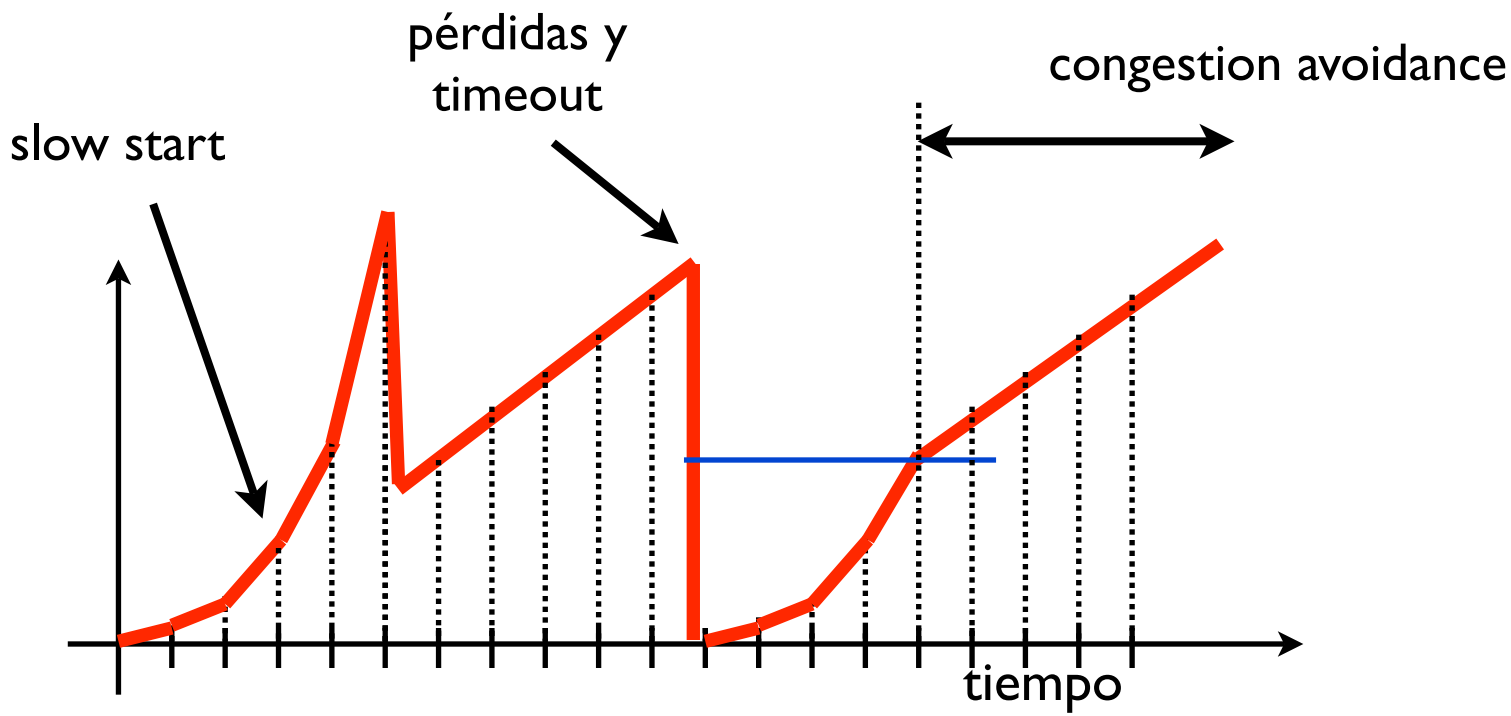


TCP: pérdidas y congestión

- 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

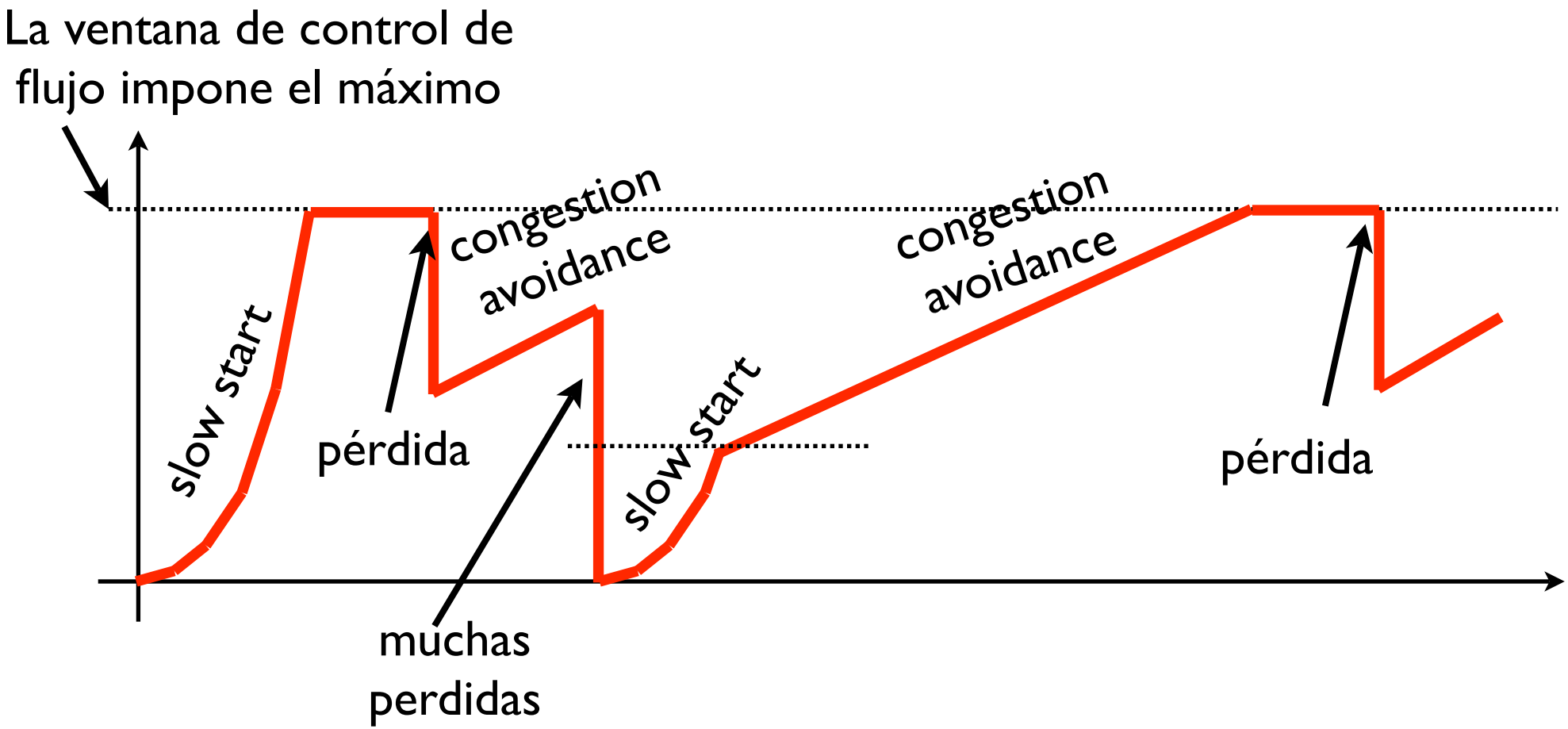
TCP: control de congestión

- 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

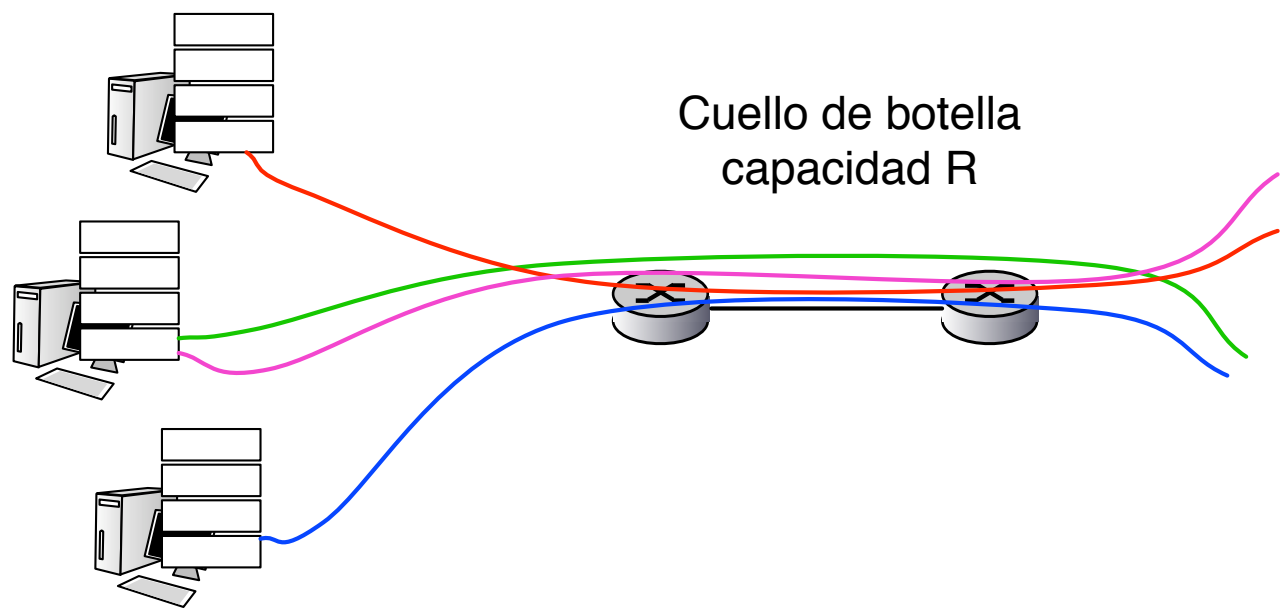


TCP: eficiencia de la conexión

- 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)

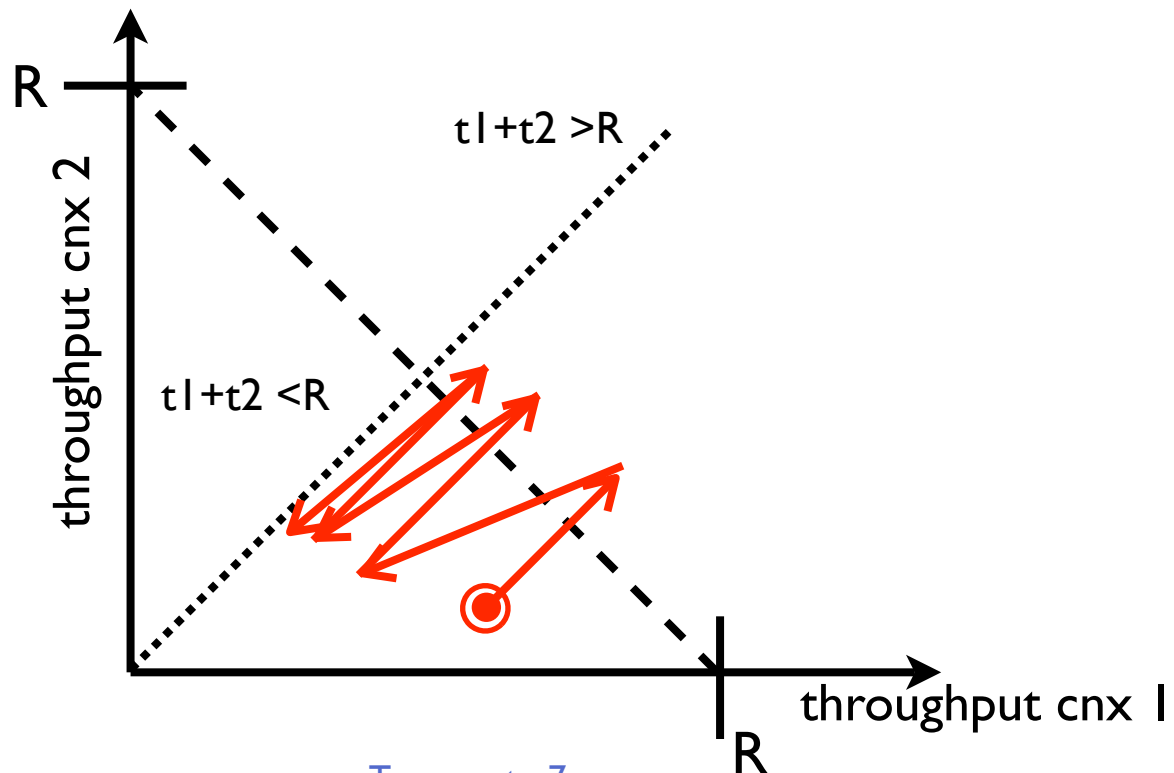
- Equidad para TCP
  - Si N conexiones TCP comparten un enlace la capacidad del enlace R debería repartirse entre todas por igual R/N



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
- Próxima clase:

  Ultimos detalles de TCP

  Mejoras y futuro de TCP