Tecnología Digital IV: Redes de Computadoras

Clase 12: Nivel de Transporte - Parte 6

Lucio Santi & Emmanuel Iarussi

Licenciatura en Tecnología Digital
Universidad Torcuato Di Tella

15 de abril de 2025

Agenda

- Servicios del nivel de transporte
- Multiplexación y demultiplexación
- Transporte no orientado a conexión: UDP
- Transferencia de datos confiable
- Transporte orientado a conexión: TCP
- Protocolos modernos: QUIC
- Introducción al control de congestión

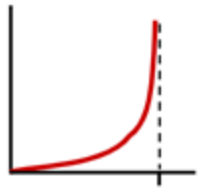
Principios del control de congestión

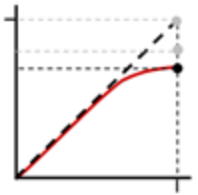
Congestión

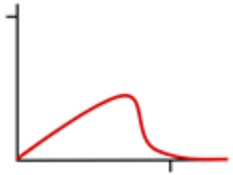
- En términos informales, *múltiples emisores enviando muchos datos, colapsando los recursos de la red*
- Síntomas:
  - Delays prolongados (encolamiento en routers)
  - Pérdida de paquetes (*overflow* de buffers en routers)
- Distinto a control de flujo:
    - Control de flujo: un emisor saturando a un receptor
- Uno de los problemas más **trascendentales** en redes de computadoras

Causas y costos de la congestión

- El *throughput* no puede exceder la capacidad
- El delay aumenta a medida que nos aproximamos a la capacidad
- Las pérdidas/retransmisiones degradan el *throughput*
- Las retransmisiones innecesarias degradan el *throughput* aún más
- Se desperdician los recursos de la red utilizados para transportar paquetes eventualmente descartados





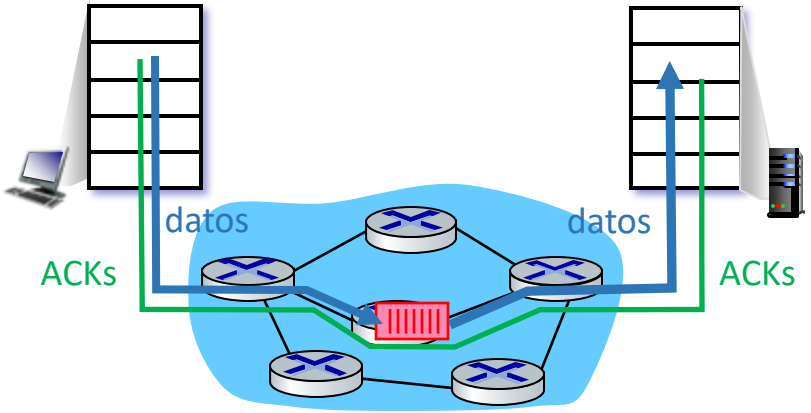


Estrategias para el control de congestión

Control de congestión *end-to-end*

- Sin retroalimentación explícita de la red
- La congestión se “infiere” del delay y/o las pérdidas observadas

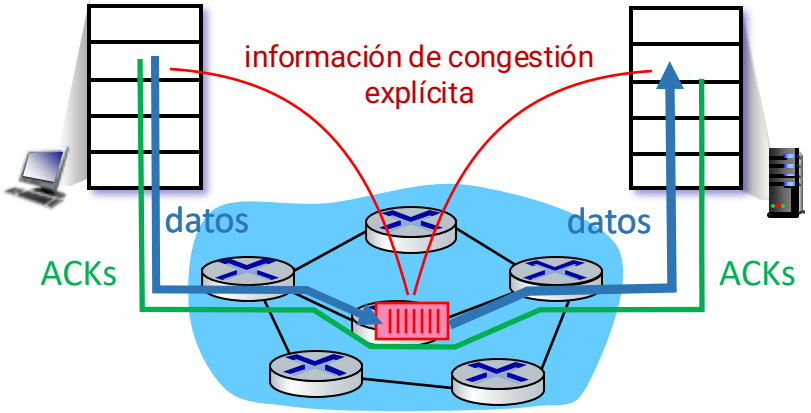
  - Enfoque adoptado por **TCP**



Estrategias para el control de congestión

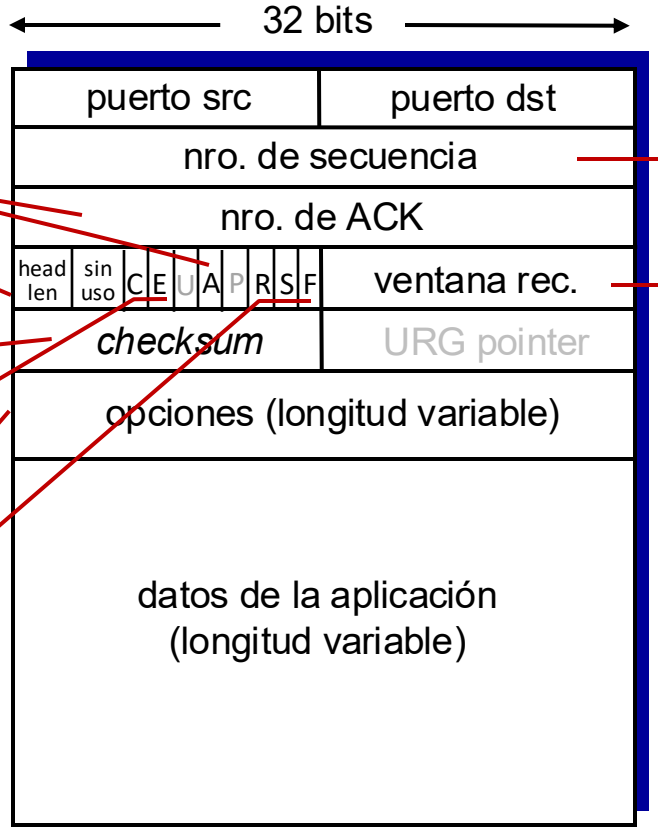
Control de congestión asistido por la red

- Los routers proveen **feedback directo** a los hosts con flujos pasando a través de ellos
- Pueden indicar el nivel de congestión o bien definir explícitamente la tasa de envío



  - Implementando e.g. en la extensión **ECN** (*Explicit Congestion Notification*) de TCP

Estructura del segmento TCP



ACK: nro. de secuencia del próximo byte esperado
flag `A`: indica ACK

longitud del *header*

Mismo que en UDP
(*Internet checksum*)

flags `C`, `E`: notificación de congestión

opciones de TCP

flags `RST`, `SYN`, `FIN`: manejo de conexión

cuenta cantidad de bytes enviados por el *stream* de datos

cantidad de bytes que el receptor está dispuesto a aceptar

00: No ECN
10: ECN-capable
11: Congestion Experienced (CE)

Control de congestión en TCP

Control de congestión en TCP: *AIMD*

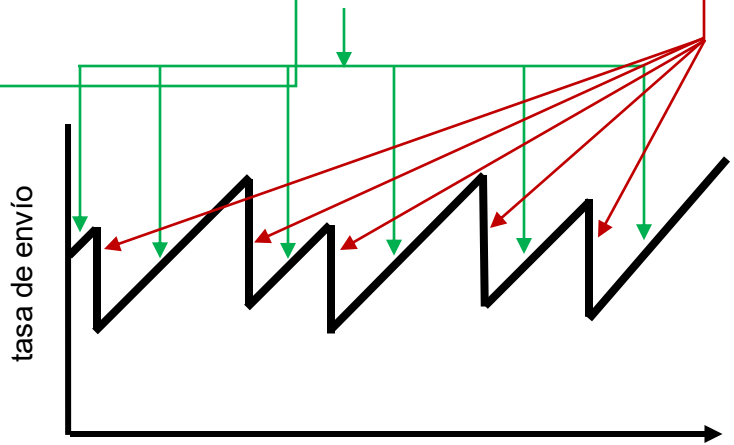
- **Enfoque**: los emisores pueden aumentar la tasa de envío hasta detectar pérdidas (congestión); luego, decrementan la tasa ante dicha pérdida

Additive Increase

aumentar tasa de envío en **1 MSS** en cada RTT hasta detectar pérdidas

Multiplicative Decrease

reducir la tasa de envío a la **mitad** ante cada pérdida

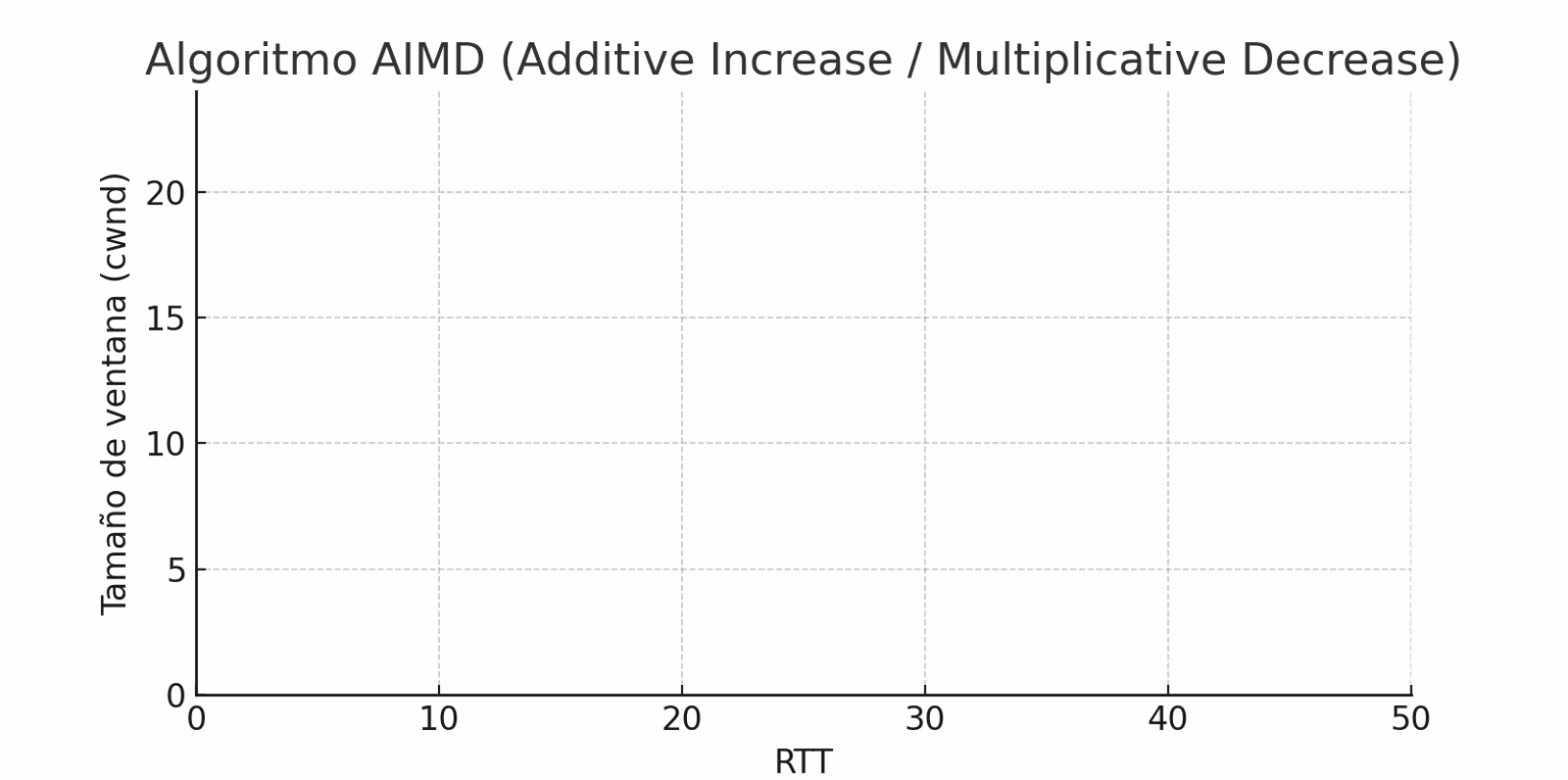


MSS: tamaño máximo del *payload* de un segmento TCP (en bytes)

- Puede negociarse al establecer la conexión a través de una opción TCP

AIMD describe un patrón de *serrucho*

AIMD


Algoritmo AIMD (Additive Increase / Multiplicative Decrease)
Tamaño de ventana (cwnd)
0
5
10
15
20
0
10
20
30
40
50
RTT

TCP AIMD

En *Multiplicative Decrease*, la tasa de envío:

- Se reduce siempre a 1 MSS ante una pérdida (*TCP Tahoe*, versión “original” del protocolo de control de congestión, 1980s)
- Se reduce a la mitad ante una pérdida indicada por la recepción de **tres ACKs duplicados** (*TCP Reno*, fines de los 90s)

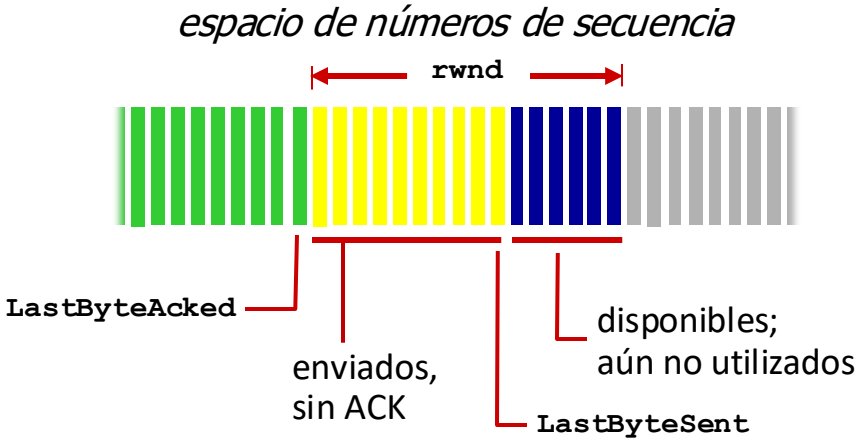
- AIMD es un algoritmo **distribuido** y **asincrónico**
- Tiene buenas propiedades, e.g.
  - Optimiza flujos congestionados a lo largo de toda la red
  - Es *justo* y *estable*

Estabilidad en redes

En redes de computadoras, la **estabilidad** implica que:

1. **La red se utiliza eficientemente**, es decir, la tasa de bits disponible se aprovecha sin desperdicio.
2. **La congestión no crece indefinidamente**, ni causa pérdidas masivas.
3. **Múltiples flujos TCP pueden coexistir** y adaptarse mutuamente, sin que uno domine o sature al resto.
4. El sistema **converge a un equilibrio** donde cada flujo ajusta su tasa en respuesta a las condiciones de la red.

Control de congestión en TCP



Comportamiento del emisor:

- *En esencia*, enviar **`cwnd`** bytes, esperar ACKs al cabo de un RTT y enviar más bytes luego

$$\text{tasa} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/s}$$

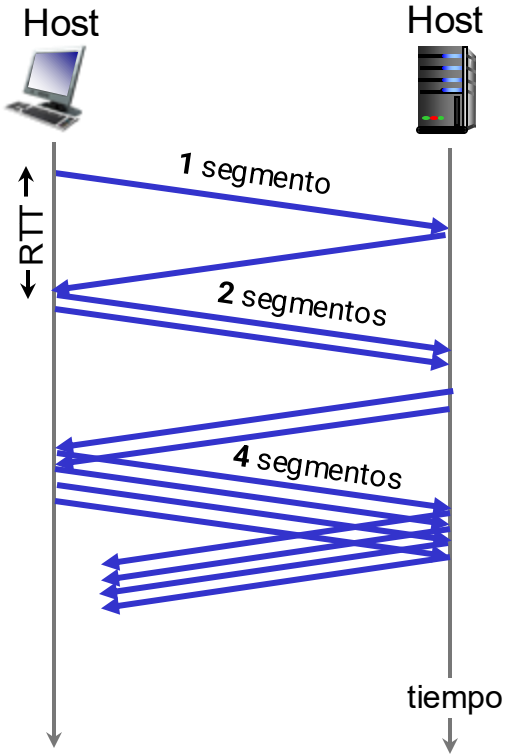
- TCP limita la cantidad de datos *en vuelo*:

```
LastByteSent - LastByteAcked ≤ mín(cwnd, rwnd)
```

- La *ventana de congestión*, **`cwnd`**, se ajusta dinámicamente en respuesta a la congestión observada en la red

Fase de *Slow Start*

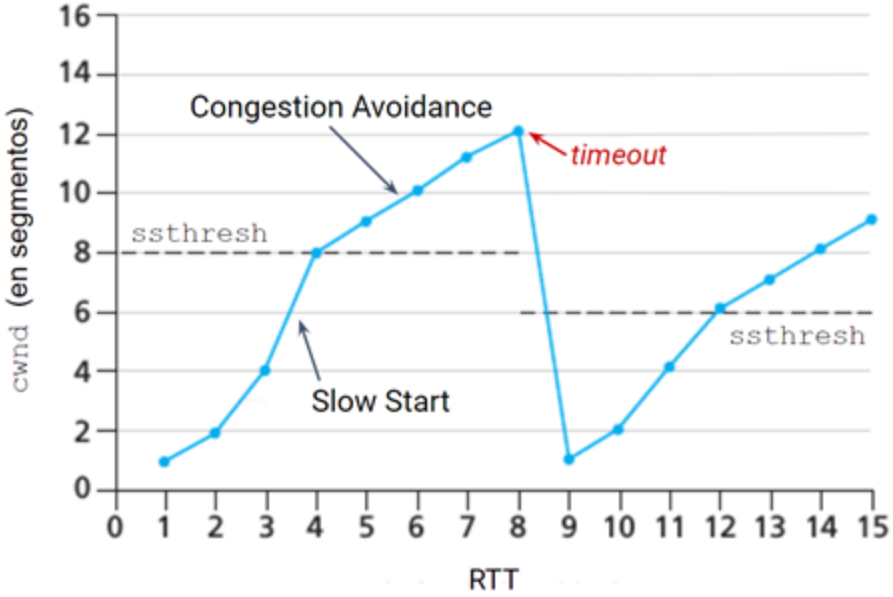
- Al iniciar la conexión, se incrementa la tasa de envío exponencialmente hasta detectar la primera pérdida:
  - Al comienzo, **`cwnd`** = ***k*** **. MSS** (ej. 1 MSS, como en la figura)
  - Se duplica la **`cwnd`** en cada RTT
  - Esto se logra incrementando en una unidad la **`cwnd`** por cada ACK recibido
- La tasa inicial es *lenta*, pero crece **exponencialmente rápido**



De *Slow Start* a *Congestion Avoidance*

- Cuando **`cwnd`** alcanza cierto umbral, el incremento de la tasa de envío deja de ser exponencial y pasa a ser **lineal**

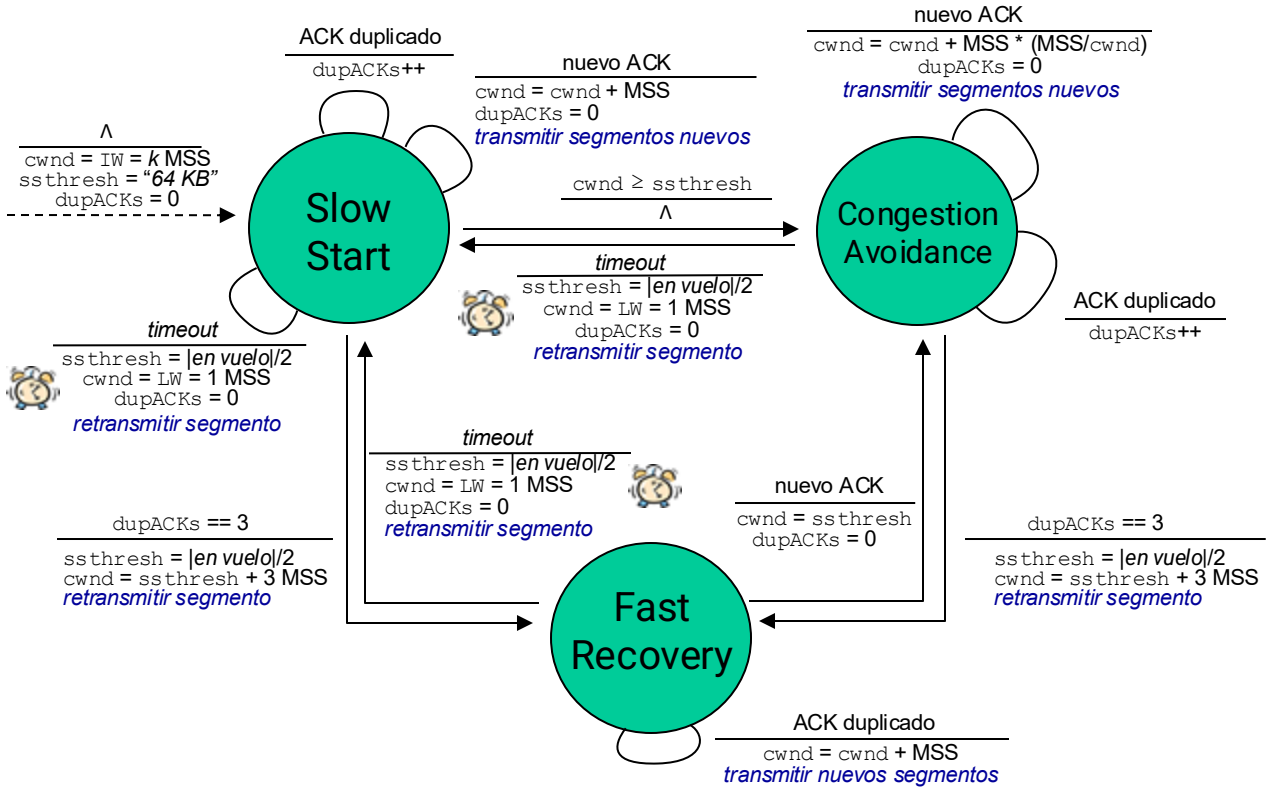
- Este umbral es **`ssthresh`**: se define como la mitad de la cantidad de datos **en vuelo** antes de detectar una pérdida



- Esta nueva fase de incremento lineal se llama *Congestion Avoidance*

TCP Reno: máquina de estados

`IW`: *initial window*
`LW`: *loss window*

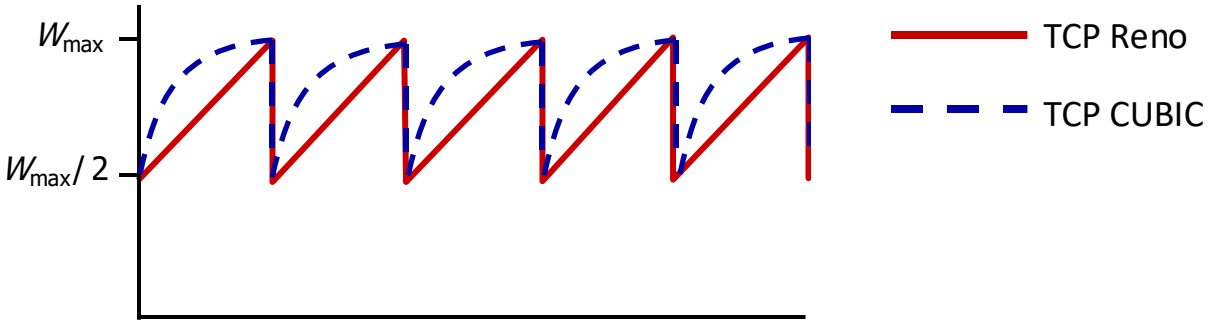


Ejercicio!

- Se tiene una conexión TCP recién establecida en la que las ventanas de recepción de los interlocutores son siempre de **20 KB**
- Uno de ellos decide enviar una cantidad arbitraria de datos al otro
- Suponiendo que la red no está congestionada, calcular cuántos bytes habrá enviado el TCP de dicho emisor al momento de entrar en Congestion Avoidance
  - Asumir una `cwnd` inicial de **2 MSS** = **4 KB** y `ssthresh` inicial de **64 KB**

TCP CUBIC

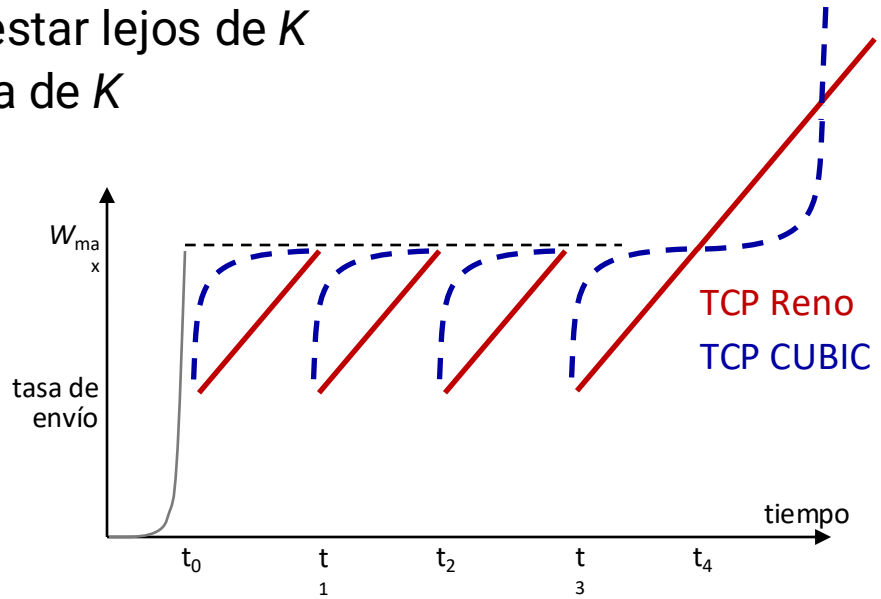
- **TCP CUBIC** (2008; RFC 8312) reformula la estrategia de *sondeo* de capacidad de transmisión disponible en la red
- Intuición:
  - $\boldsymbol{W_{max}}$: tasa de envío con la cual se detectó congestión
  - Luego de reducir la ventana ante una pérdida, incrementarla rápido hacia W_{max} al principio, pero luego acercarse despacio



TCP CUBIC

- ***K***: instante de tiempo en el que la ventana de congestión alcanza W_{max}
- La ventana se incrementa de acuerdo al cubo de la distancia entre el tiempo actual y *K*
  - Incrementos grandes al estar lejos de *K*
  - Incrementos chicos cerca de *K*

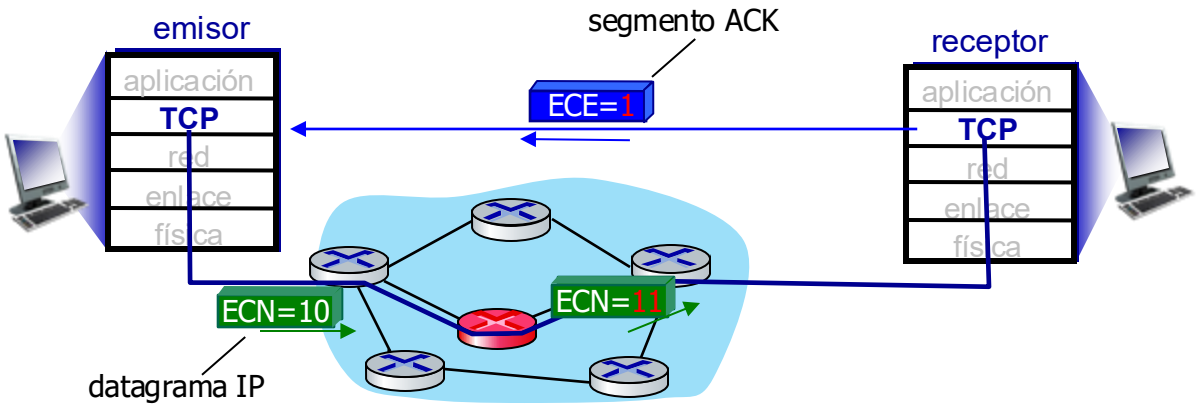
- **TCP CUBIC** es el algoritmo de CC predeterminado en Linux y el más utilizado en los servidores web más comunes



Explicit Congestion Notification (ECN)

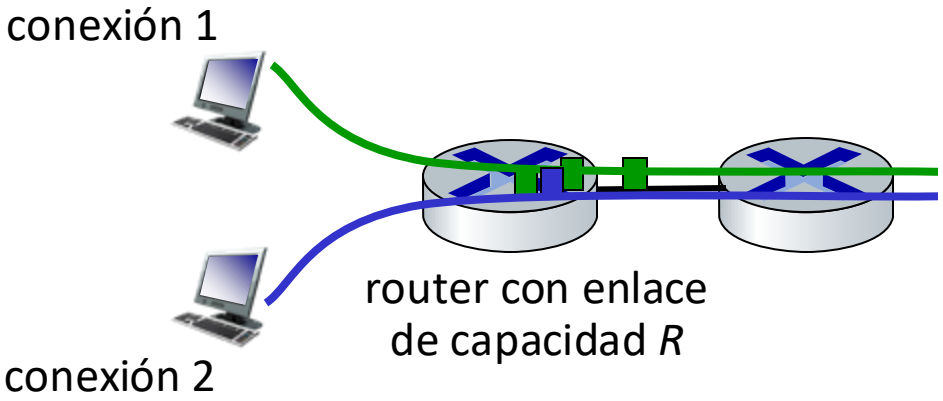
Control de congestión asistido por la red en TCP/IP:

- Utiliza dos bits en el *header* IP (campo Type of Service) marcados por los routers para indicar congestión (decisión a cargo de los operadores de la red)
- El paquete marcado eventualmente llega al receptor TCP
- En el siguiente ACK, se enciende el bit ECE (*ECN Echo*) en el *header* TCP
- El emisor, luego, reduce a la mitad la ventana de congestión



TCP *Fairness*

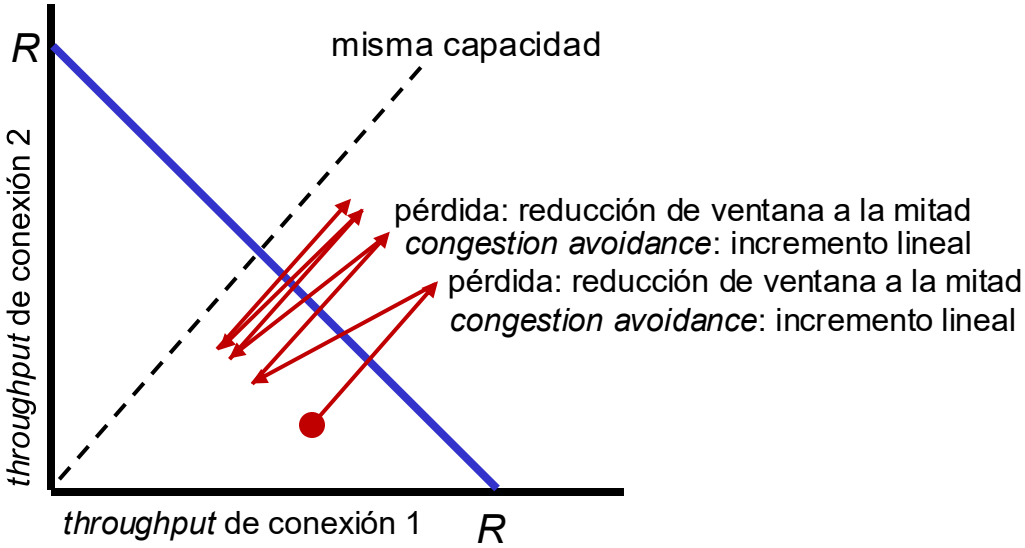
Objetivo: si *K* sesiones TCP comparten el mismo enlace con capacidad *R*, cada una debería tener una tasa promedio de *R*/*K*



TCP *Fairness*

Se demostró que el algoritmo AIMD es *fair* bajo ciertas suposiciones

- Intuición: dos conexiones con igual RTT que comparten un router con capacidad *R* (sin otras conexiones o tráfico UDP)



Fairness en Internet

Tráfico UDP

- Las apps multimedia no suelen utilizar TCP
  - No quieren que la tasa de transmisión se regule vía control de congestión
- Con UDP pueden enviar a tasa constante, tolerando pérdidas
- UDP potencialmente puede “desplazar” el tráfico TCP en los routers

Conexiones TCP en paralelo

- Las apps pueden abrir múltiples conexiones en paralelo entre dos hosts (los navegadores suelen hacerlo)
- Si por ejemplo tenemos un enlace con capacidad *R* y 9 conexiones,
  - Si una app abre una nueva conexión, obtiene una tasa de *R*/10
  - Si una nueva app abre 11 conexiones, obtiene una tasa de *R*/2