Control de congestión y buffers

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

Grado en Ingeniería en Tecnologías de Telecomunicación, 4º

Incremento/Decremento en control de congestión

{M|A}I{M|A}D

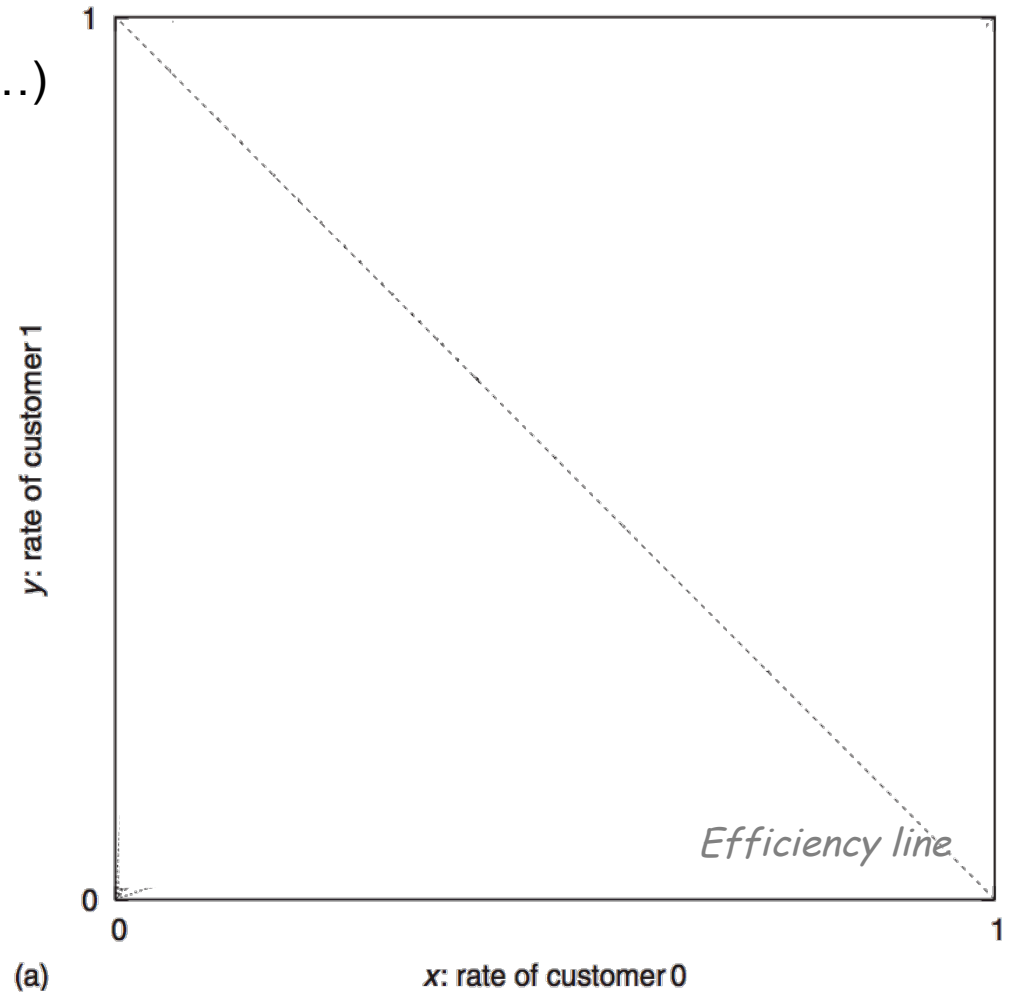
- x(t) : tasa de envío
- Planteamos una familia de controles para esta tasa:

$$x(t+\Delta t) = \begin{cases} a_i + b_i x(t) & \text{si no hay congestión} \\ a_d + b_d x(t) & \text{si hay congestión} \end{cases}$$

- MIMD: *Multiplicative Increase, Multiplicative Decrease*
  $a_i=0$; $a_d=0$; $b_i>1$; $0<b_d<1$
- AIAD: *Additive Increase, Additive Decrease*
  $a_i>0$; $a_d<0$; $b_i=1$; $b_d=1$
- AIMD: *Additive Increase, Multiplicative Decrease*
  $a_i>0$; $a_d=0$; $b_i=1$; $0<b_d<1$
- MIAD: *Multiplicative Increase, Additive Decrease*
  $a_i=0$; $a_d<0$; $b_i>1$; $b_d=1$

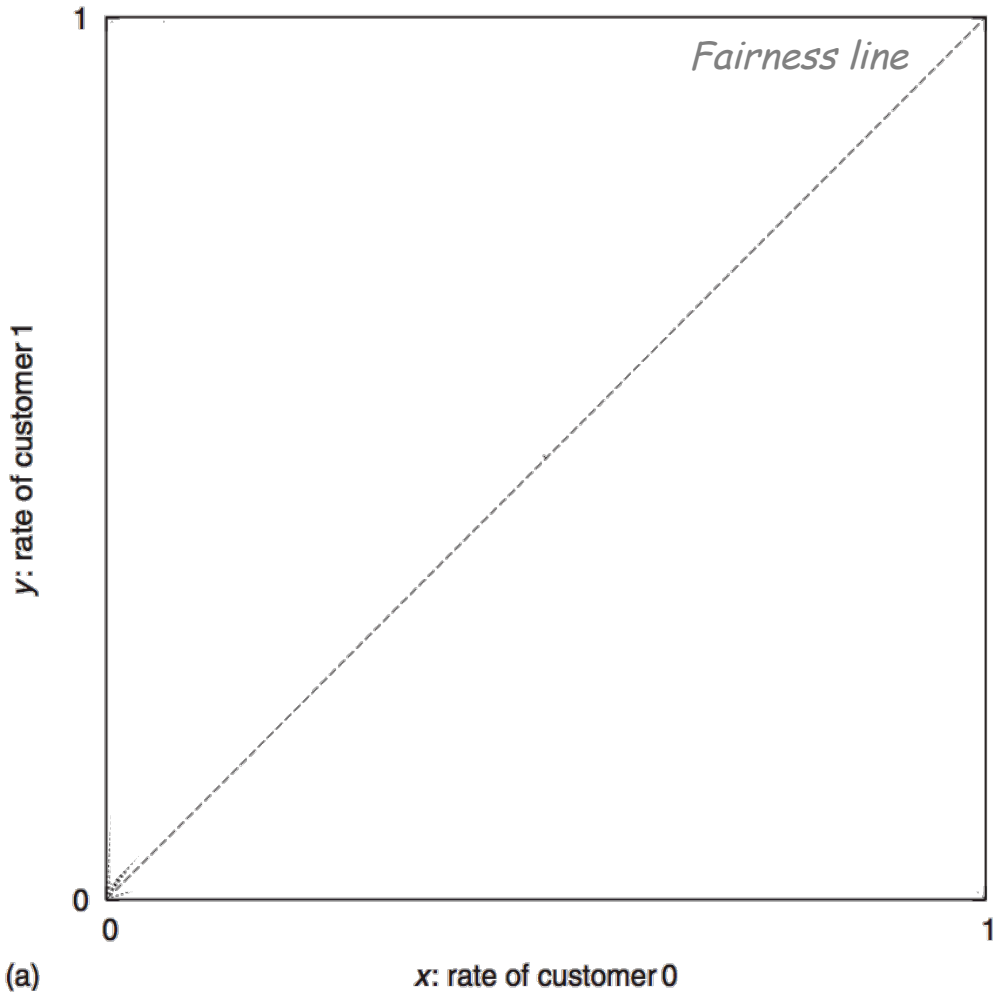
AIAD, MIMD, MIAD

- Supongamos dos fuentes saturando un enlace (capacidad unidad)
- Uso eficiente (*efficiency*)
  - Por debajo capacidad libre
  - Por encima pérdidas
- Reparto justo (*fariness*) (…)



AIAD, MIMD, MIAD

- Supongamos dos fuentes saturando un enlace (capacidad unidad)
- Uso eficiente (*efficiency*)
  - Por debajo capacidad libre
  - Por encima pérdidas
- Reparto justo (*fariness*)
- (...)

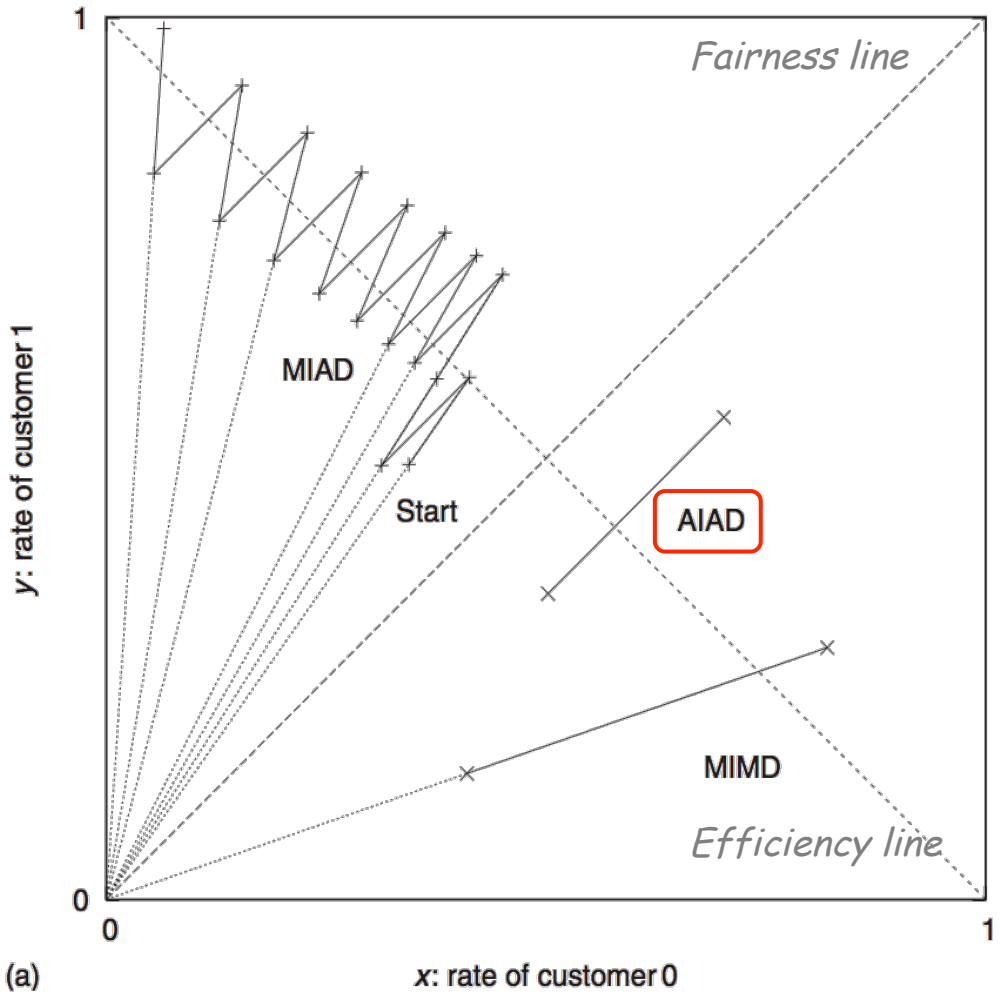


AIAD, MIMD, MIAD

AIAD

- Si empiezan bajo línea de eficiencia …
- Aumentan tasa igual ambos (paralelo a *fairness*)
- Hasta cruzar la línea de eficiencia y tener pérdidas
- Decrementa tasa igual ambos (paralelo a *fairness*)

$$x(t+\Delta t) = \begin{cases} a + x(t) & \text{no congestión} \\ -a + x(t) & \text{congestión} \end{cases}$$

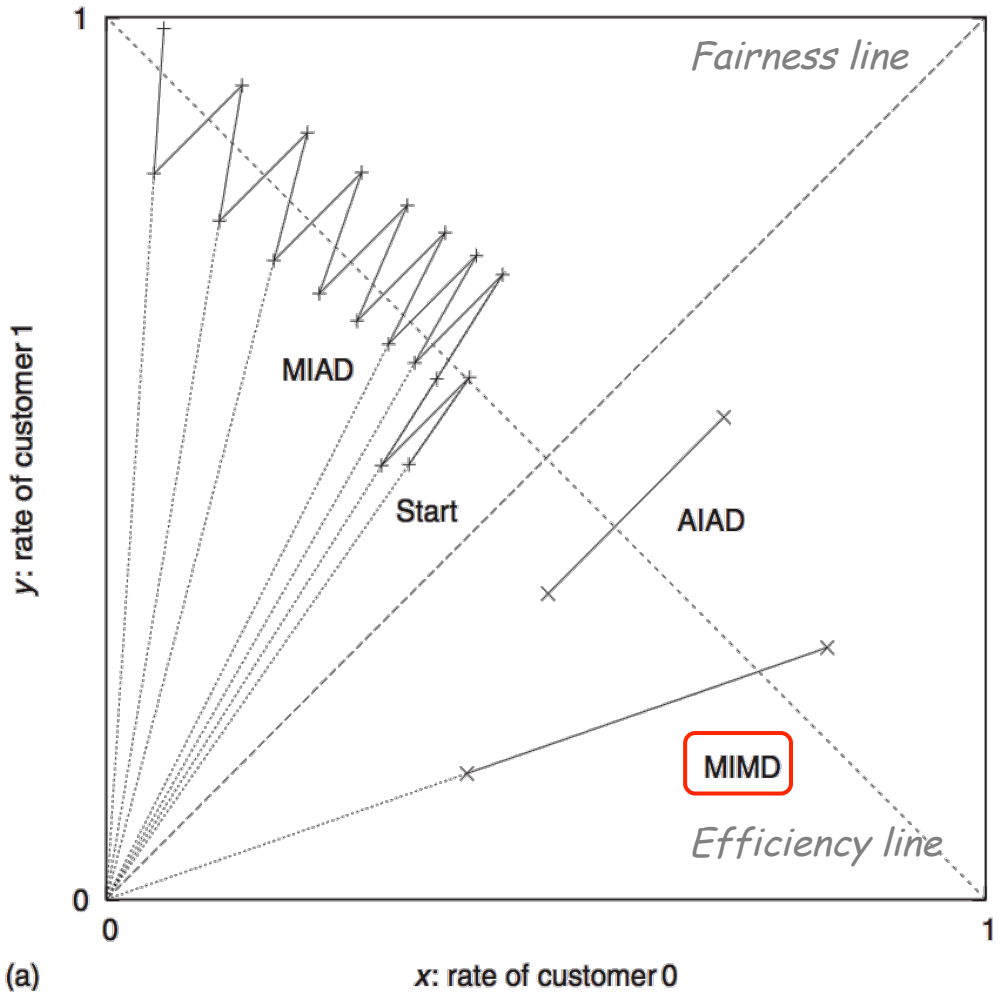


AIAD, MIMD, MIAD

MIMD

- Si empiezan bajo línea de eficiencia …
- Aumentan tasa igual ambos en un factor (recta por origen)
- Hasta cruzar la línea de eficiencia y tener pérdidas
- Decrementan tasa igual ambos en un factor (recta por origen)

$$x(t+\Delta t) = \begin{cases} bx(t) & \text{no congestión} \\ -bx(t) & \text{congestión} \end{cases}$$

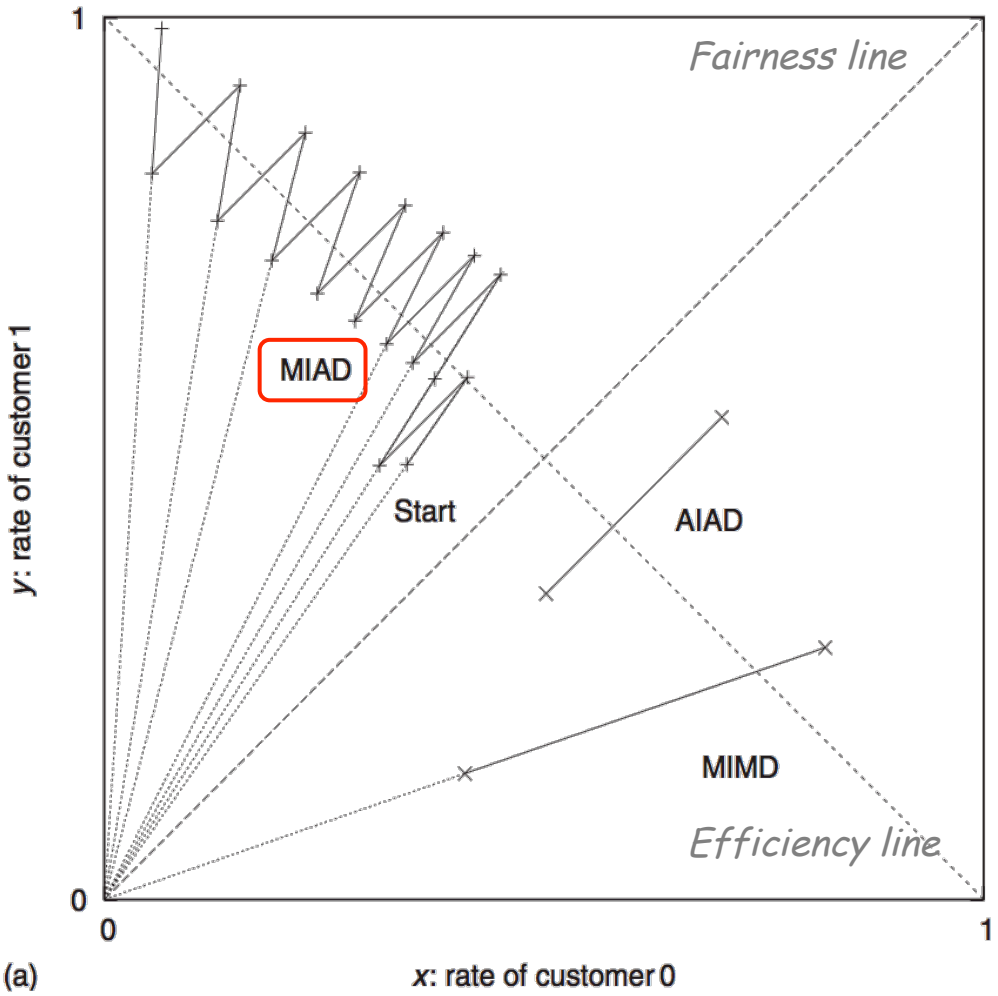


AIAD, MIMD, MIAD

MIAD

- Si empiezan bajo línea de eficiencia …
- Aumentan tasa igual ambos en un factor (recta por origen)
- Hasta cruzar la línea de eficiencia y tener pérdidas
- Decrementan tasa igual ambos (paralelo a *fairness)*
- Converge a injusto
- Todo para el que empezó con más

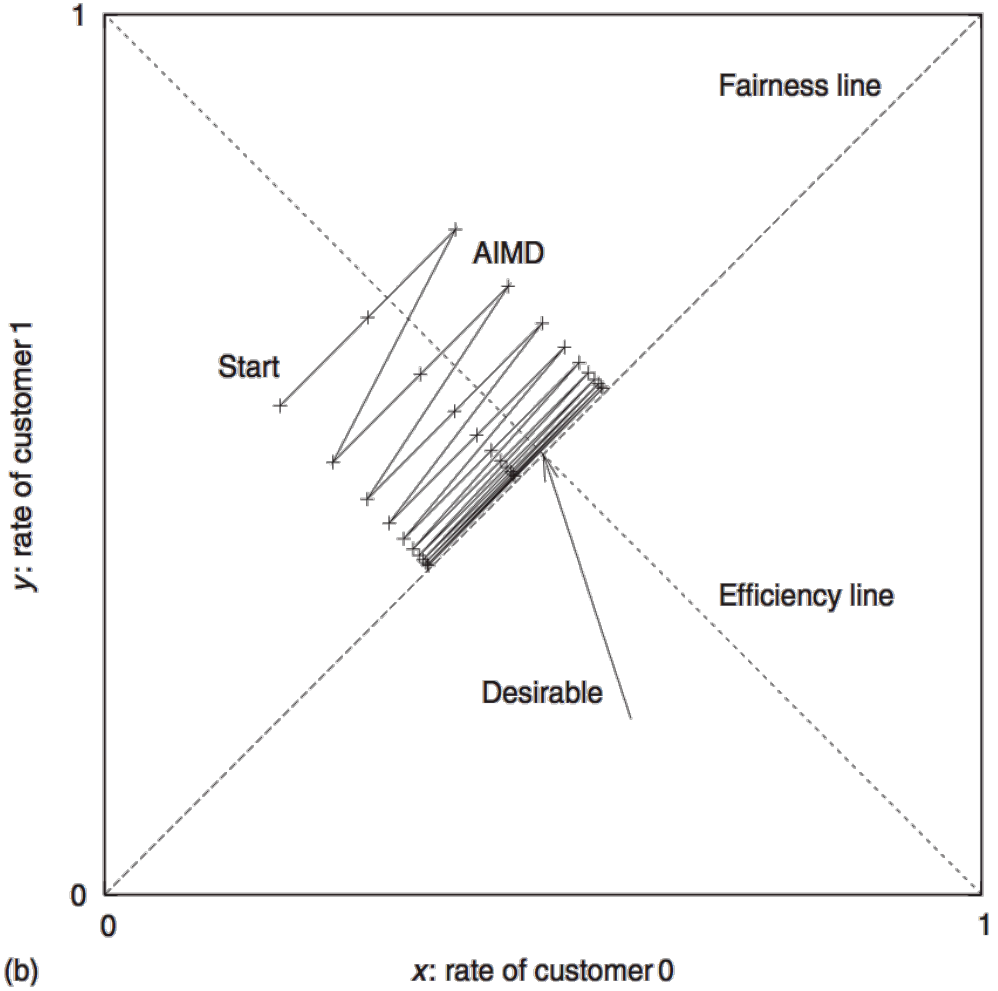
$$x(t+\Delta t) = \begin{cases} bx(t) & \text{no congestión} \\ -a + x(t) & \text{congestión} \end{cases}$$



AIMD

- Si empiezan bajo la línea de eficiencia
- Aumentan tasa igual ambos (paralelo a *fairness*)
- Hasta cruzar la línea de eficiencia y tener pérdidas
- Decrementan tasa igual ambos en un factor (recta por origen)
- Converge a un equilibrio
- Fluctúa en torno al óptimo

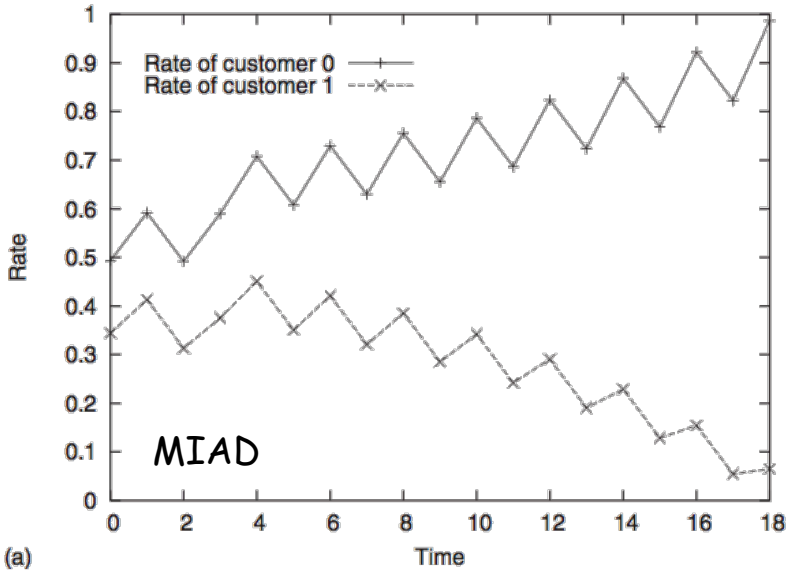
$$x(t+\Delta t) = \begin{cases} a + x(t) & \text{no congestión} \\ -bx(t) & \text{congestión} \end{cases}$$

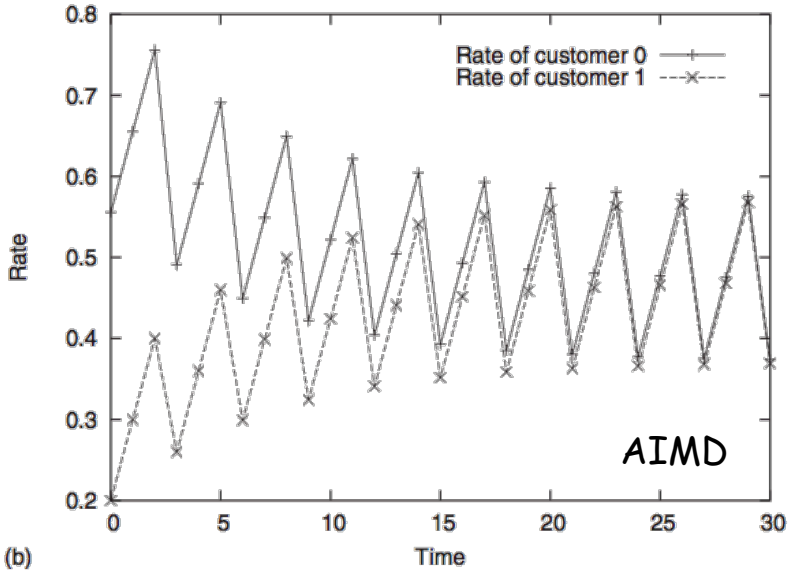


Raj Jain

MIAD y AIMD vs time

- En MIAD se pierde el reparto justo
- En AIMD se logra pero con oscilaciones
- El control de congestión de TCP es básicamente AIMD
- Si el tiempo que tarda la realimentación es diferente para cada fuente puede no dar reparto justo
- El tiempo de la realimentación se basa en timeout que se calcula en base a estimación del RTT
- ¿Cómo hacer ese control de la velocidad?





Rate- vs window- control

Rate- vs Window- control

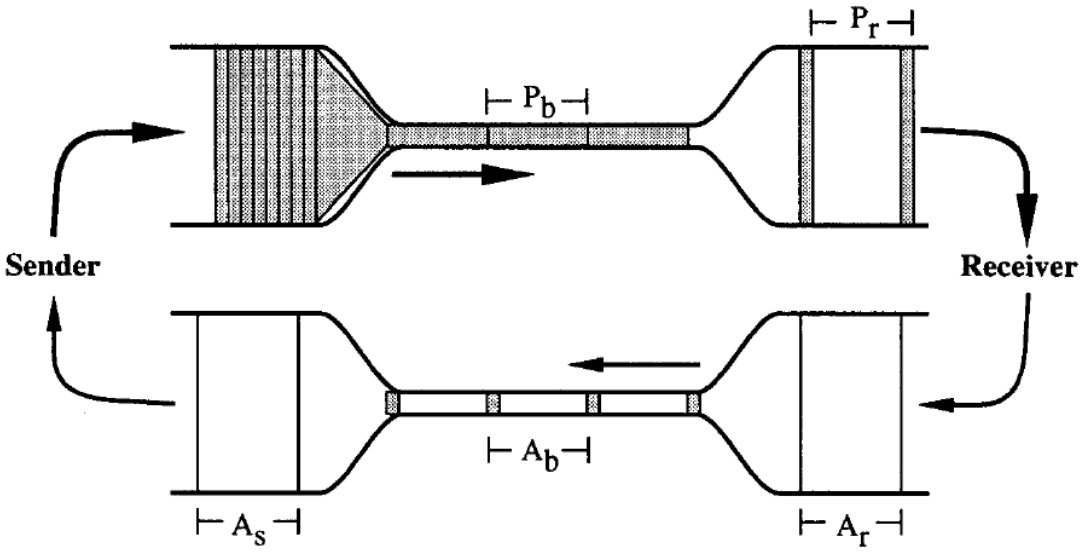
- Dos métodos de control del envío datos:
  - Control basado en la tasa de envío
  - Control basado en ventana

Rate- control

- Control basado en la tasa de envío
- De alguna forma el emisor sabe a qué tasa debe enviar
- Le informa el receptor o un equipo intermedio
- Simple
- Más apropiado para streaming
  - No queremos que se detenga pues la fuente no lo hace
  - Mantendría la tasa ante congestión
  - Si se le notifica de la nueva tasa adecuada podría hacer transcodificación

Window- control

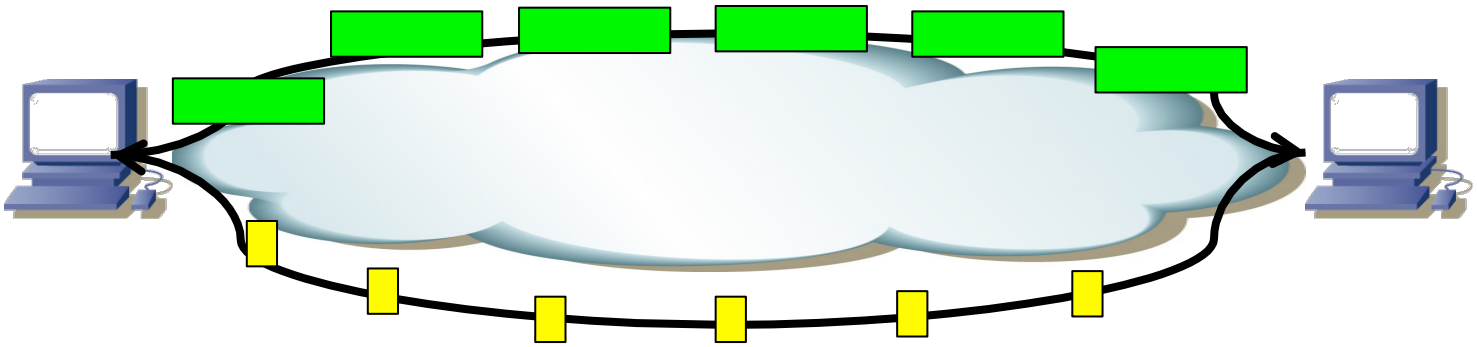
- Control basado en ventana
- Hay un máximo de paquetes que pueden estar “en vuelo”
- Solo se puede introducir uno cuando otro ha salido/llegado
- Receptor notifica a emisor de que un paquete ha llegado
- Más conservador pues deja de enviar si paquetes “no salen”
- Self-Clocking: la tasa de envío está limitada por la de ACKs



V. Jacobson, Proc. ACM SIGCOMM’88

Window- control

- La ventana debe ser lo suficientemente grande para sacar provecho a todo el canal
- Eso es un tamaño al menos BW x RTT
- Eso requiere una buena estimación del RTT

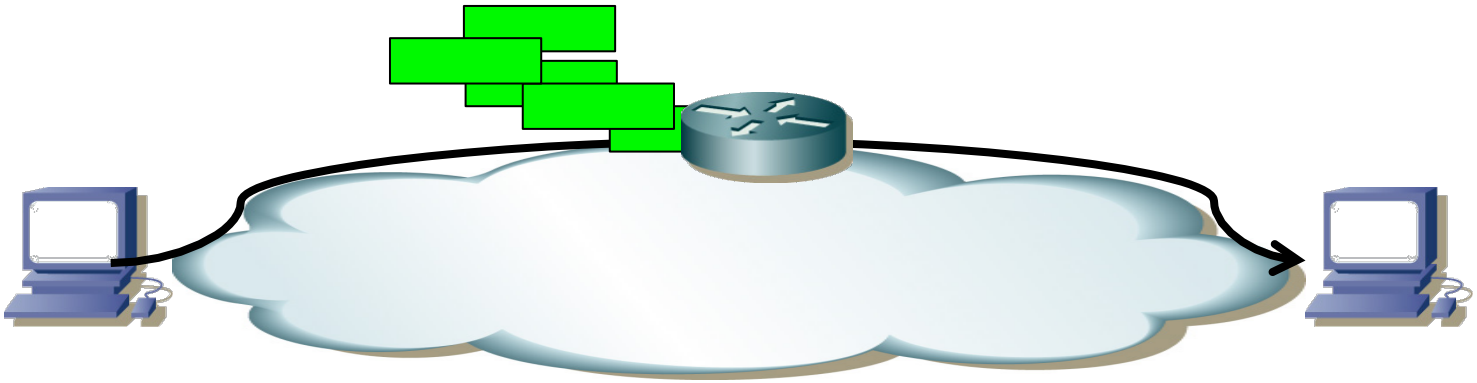


Window- control

- La transferencia se autorregula
- ¿Entonces no puede haber ráfagas?
- (...)

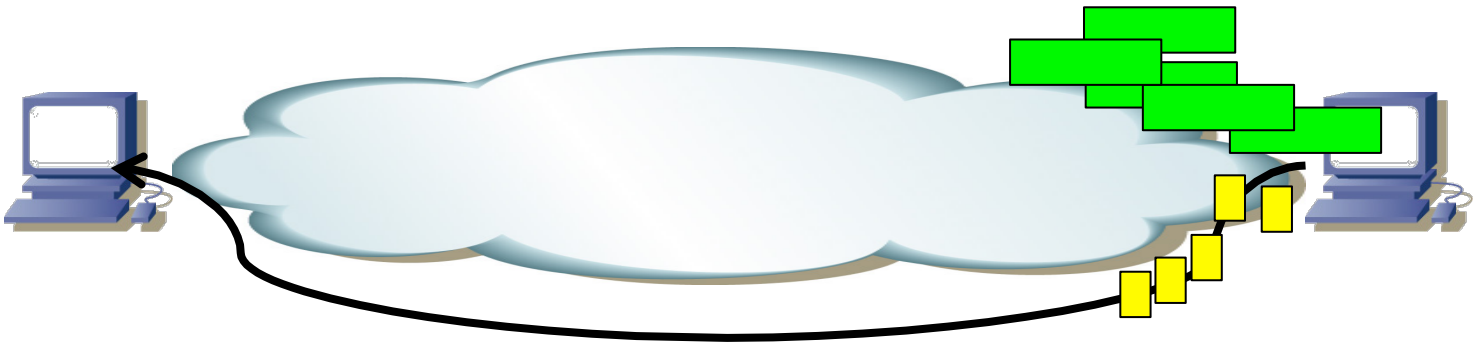
Ráfagas

- Ejemplo:
  - Emisor envía y los paquetes se encolan en un puerto de salida
  - Cuando se transmiten salen juntos a la tasa de ese enlace
  - Puede ser de alta capacidad y solo haber sufrido congestión temporal
  - (…)



Ráfagas

- Ejemplo:
  - Emisor envía y los paquetes se encolan en un puerto de salida
  - Cuando se transmiten salen juntos a la tasa de ese enlace
  - Puede ser de alta capacidad y solo haber sufrido congestión temporal
  - Los ACKs de paquetes que llegan próximos saldrán próximos
  - (...)



Ráfagas

- Ejemplo:
  - Emisor envía y los paquetes se encolan en un puerto de salida
  - Cuando se transmiten salen juntos a la tasa de ese enlace
  - Puede ser de alta capacidad y solo haber sufrido congestión temporal
  - Los ACKs de paquetes que llegan próximos saldrán próximos
  - Y provocarán que se “abra” la ventana de golpe y el emisor pueda enviar varios paquetes a la tasa de su interfaz

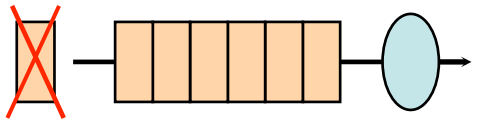
Ráfagas

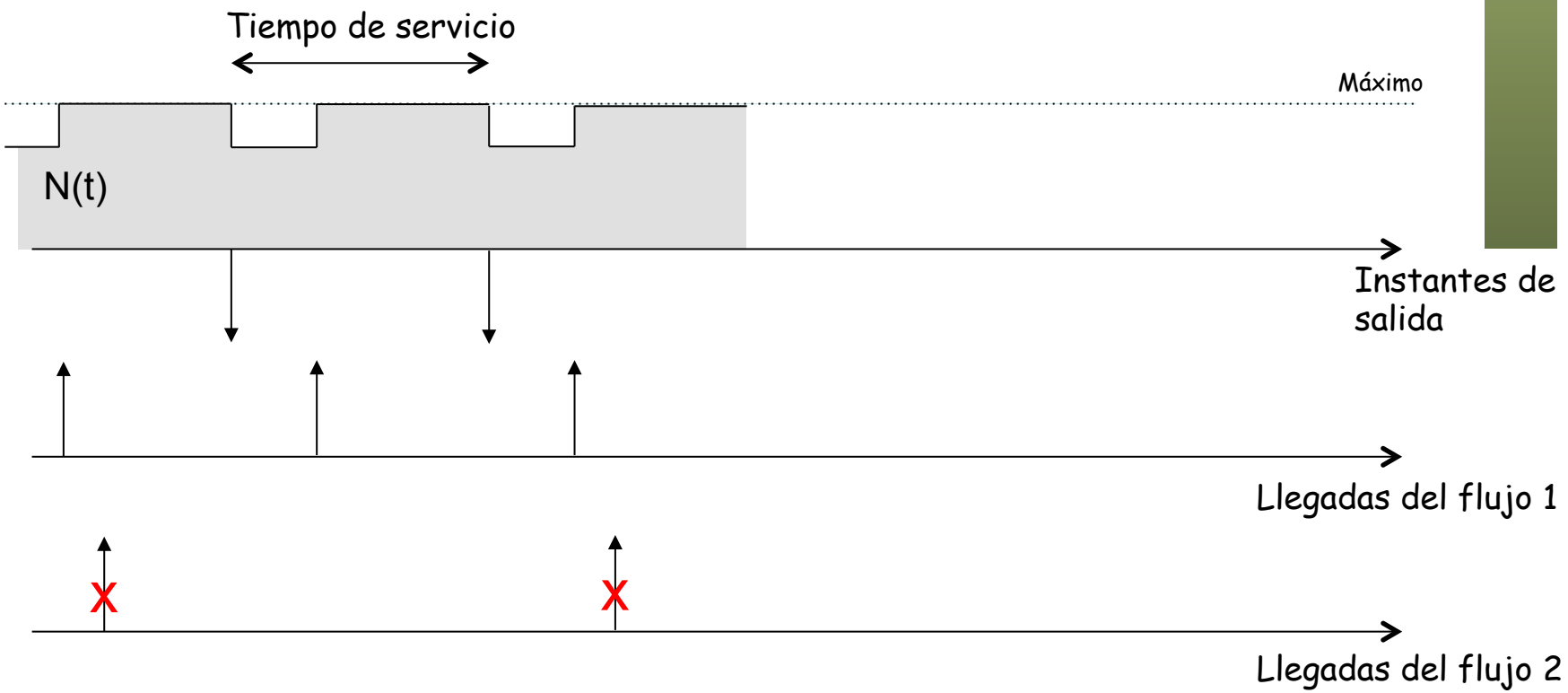
- **Otro ejemplo:**
  - Ha habido una transferencia, que ha podido estar bien auto-regulada
  - Pero ya no hay más datos a transmitir y se han confirmado todos
  - La ventana queda completamente “abierta”
  - Ahora si la aplicación genera una gran cantidad de datos se podrán enviar tantos como la ventana en una ráfaga

Buffers y sincronización

Traffic phase effects

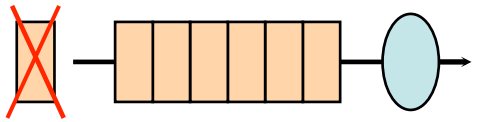
- Un ejemplo
- Todos los paquetes de igual tamaño
- Enlace congestionado, cola llena
- Flujo 1 ocupa el hueco que queda al terminar de transmitir
- Otra vez (llegadas periódicas)
- Llegada del flujo 2 se pierde (...)

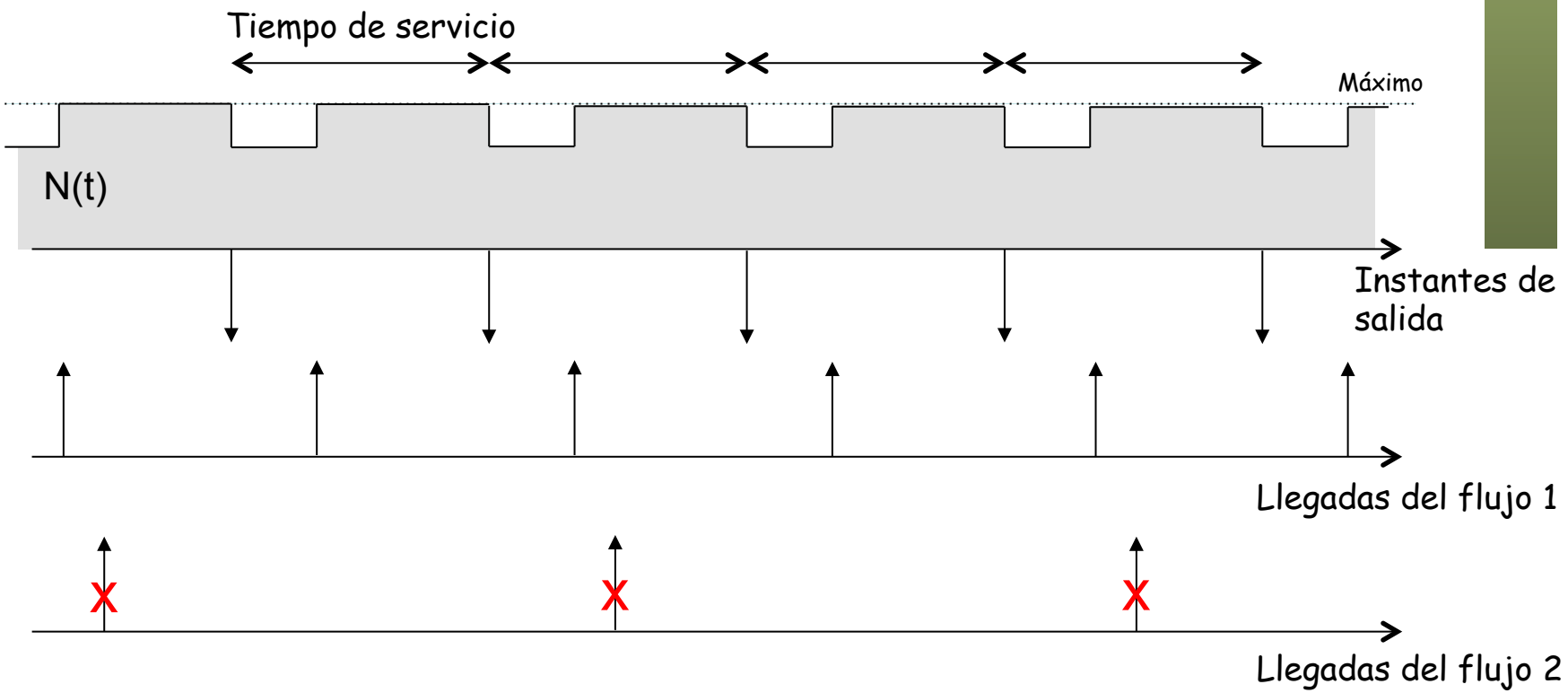




Traffic phase effects

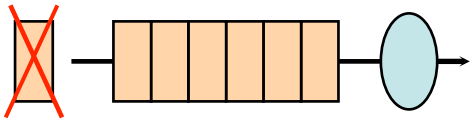
- Flujo 1 ocupa el hueco que queda al terminar de transmitir (otra vez)
- Otra vez
- Otra vez llegada del flujo 2 se pierde (periódicas)
- ¡ Se van a perder siempre !
- Salidas periódicas (siempre hay *backlog* y con paquetes de tamaño constante)



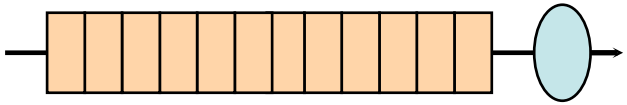


Romper la sincronización

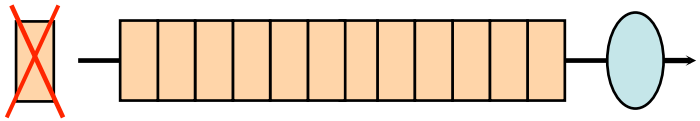
- Una de las causas son las colas drop-tail
- Y que detectamos congestión con pérdidas



- Evitaríamos pérdidas con cola más grande

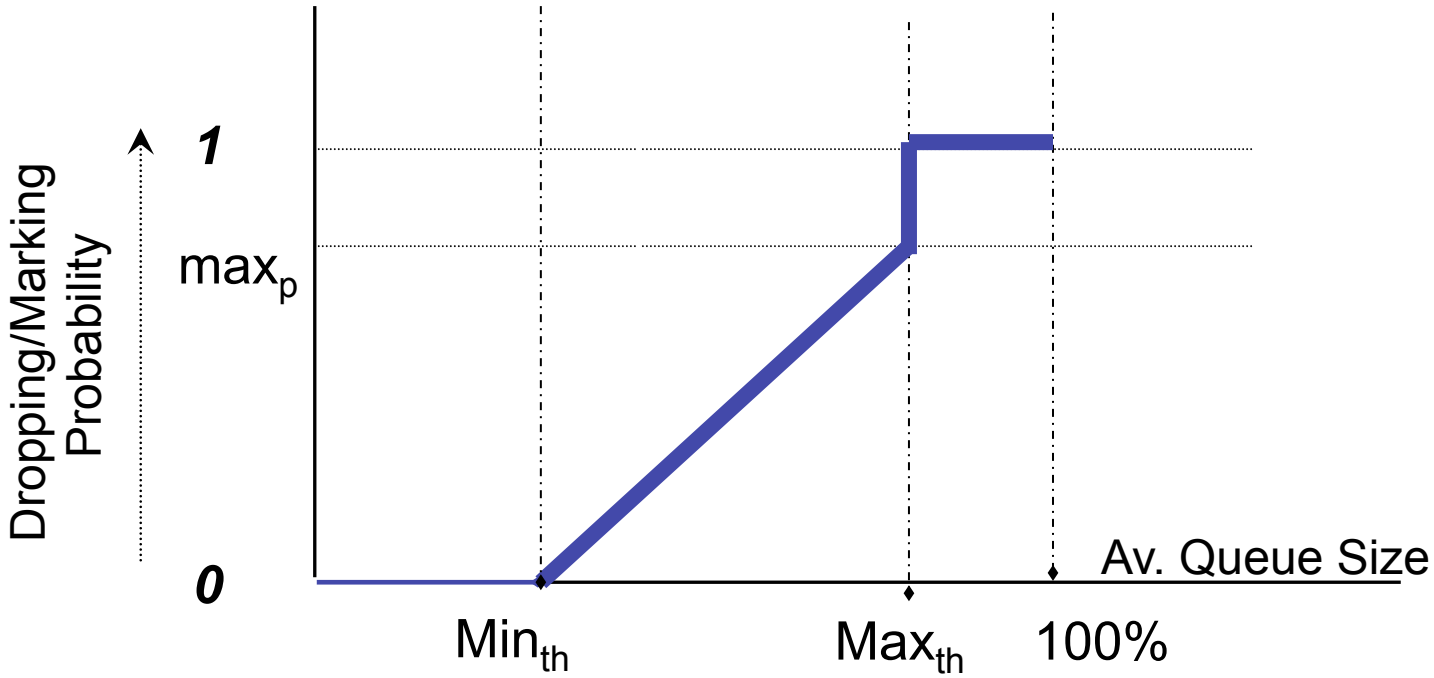


- ¿Resuelve el problema?
- No, no evita la congestión
- Sin pérdidas no hay *feedback* de que hay congestión
- Luego las fuentes siguen aumentando la tasa de envío
- Y acaba habiendo pérdidas



Romper la sincronización

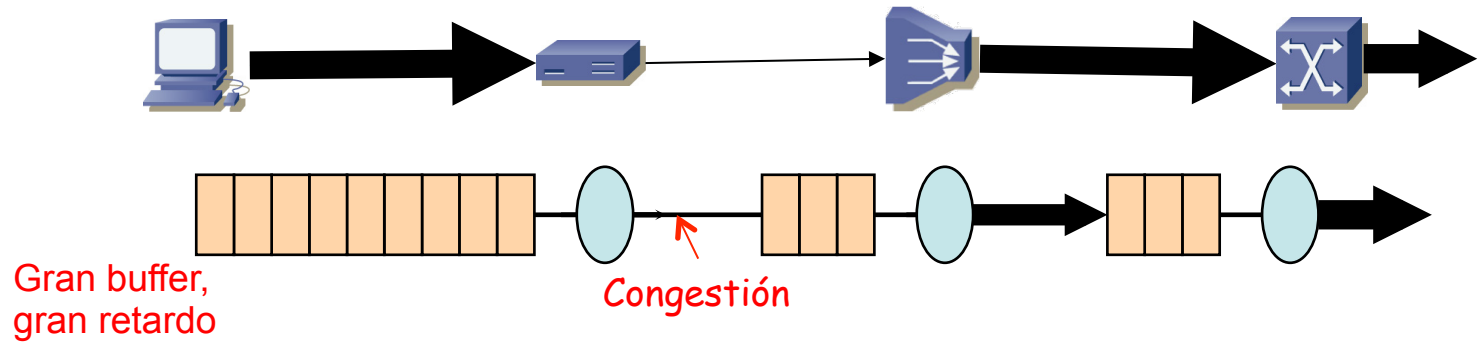
- Introducir aleatoriedad en las pérdidas
- AQM, típicamente RED (Random Early Detection)
- Permite romper la sincronización y mantener la cola poco ocupada
- Complejo de configurar



Tamaño de los buffers

Tamaño de buffers

- ¿Bueno, pero en realidad queremos el buffer ocupado para conseguir mayor throughput, no?
- Aún con un buffer grande habrá pérdidas
- Peor aún: además hay mayor retardo
- De hecho hoy en día se están poniendo buffers muy grandes
- Eso en enlaces congestionados (por ejemplo upstream ADSL) lleva a grandes retardos (centenares de ms o incluso segundos)



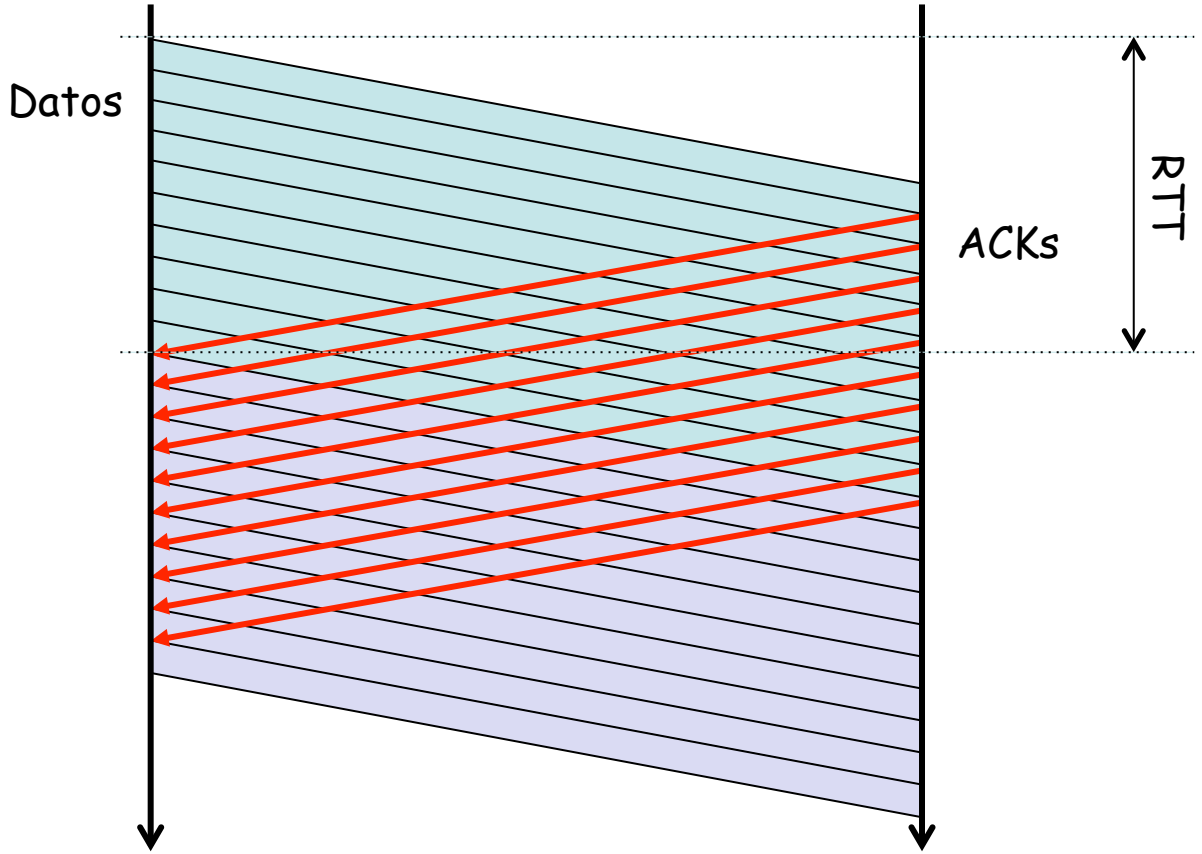
- Aumenta el RTT y con él la pérdida de interactividad (resolución de DNS, comienzo de conexiones cortas, etc)

¿Tamaño del buffer?

- Queremos buffer para absorber la intermitencia del tráfico
- No lo queremos grande para que no incremente el retardo
- Retardo grande aumenta el tiempo que tarda el cliente en recibir notificaciones de congestión
  - Por ejemplo el RTO calculado será mayor
  - Entonces tarda más tiempo en reconocer una pérdida y reaccionar
- ¿Cuál es el tamaño ideal de buffer?

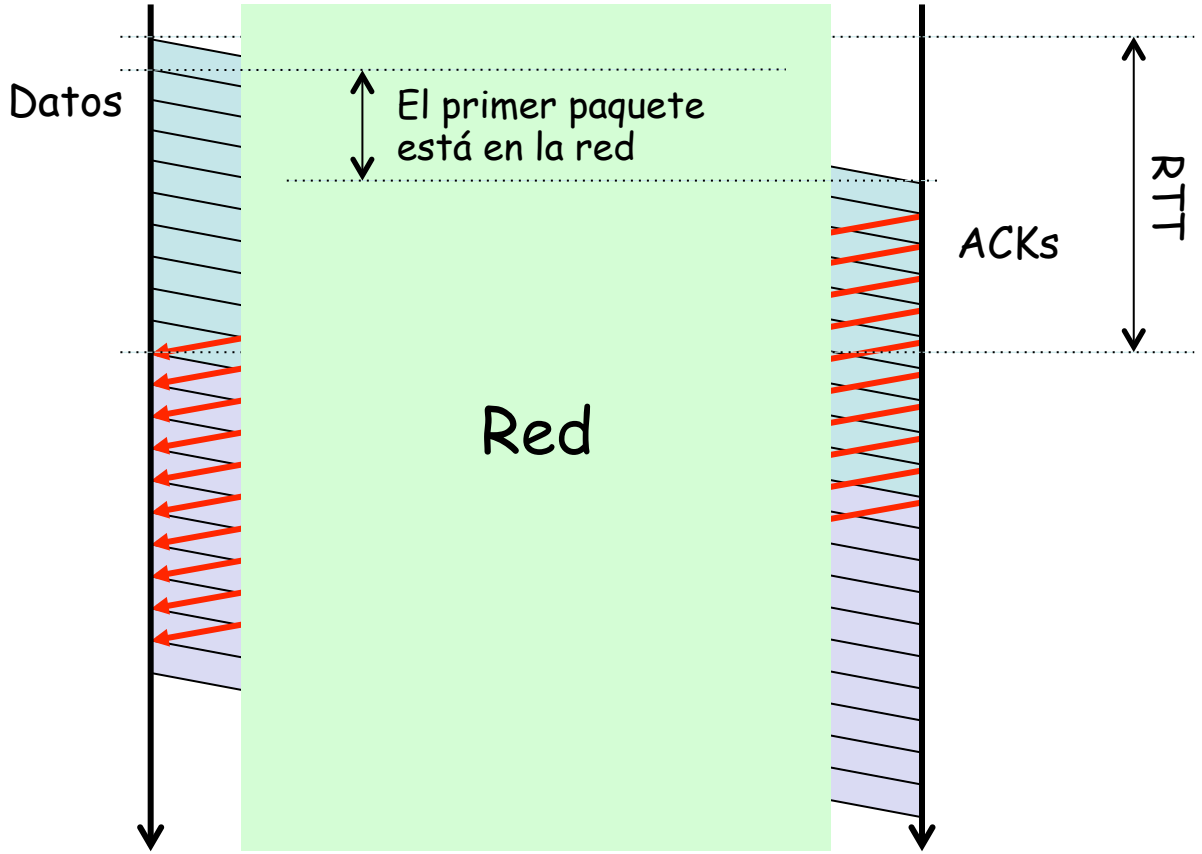
Buffer

- Transporte window-based
- Ventana para saturar = RTT x Capacidad
- (…)



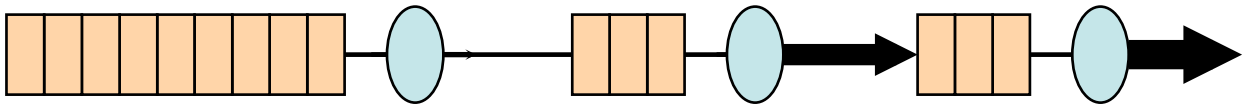
Buffer

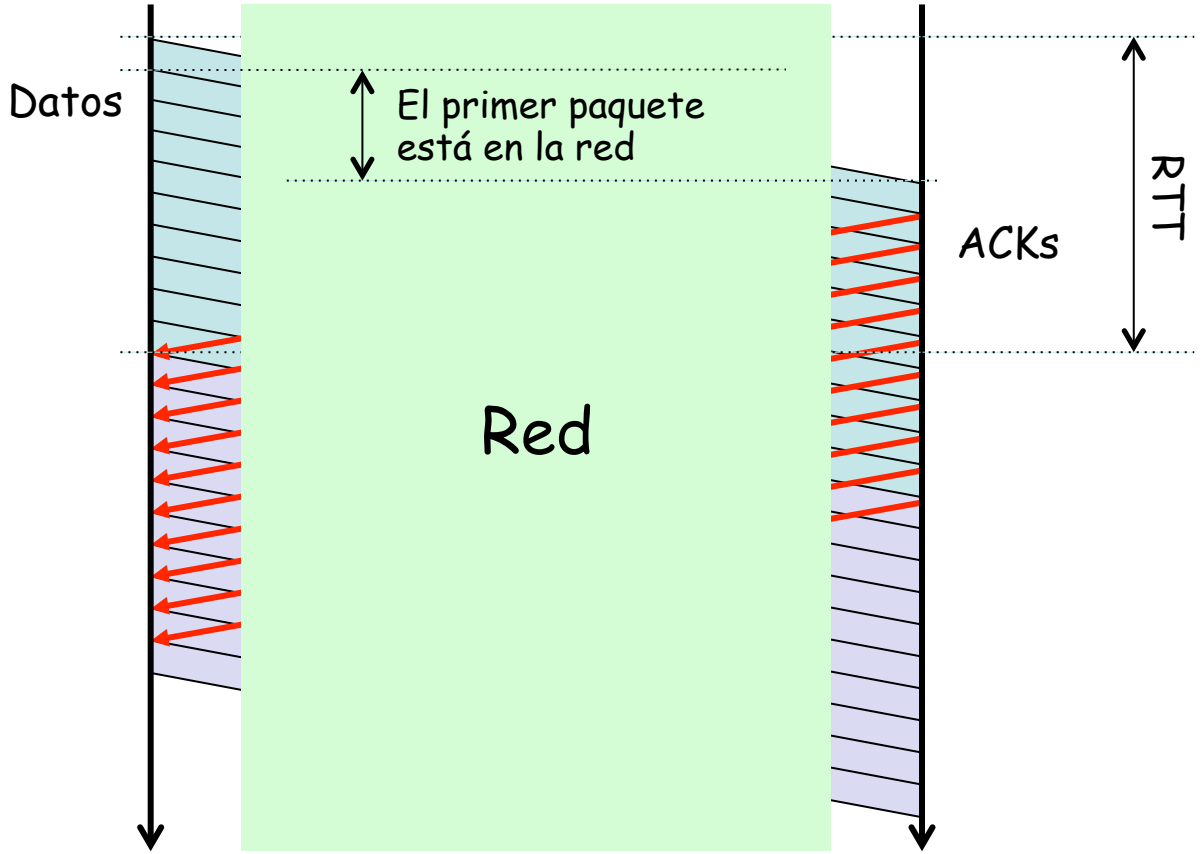
- Transporte window-based
- Ventana para saturar = RTT x Capacidad
- ¿Dónde están los paquetes?
- En la red (…)



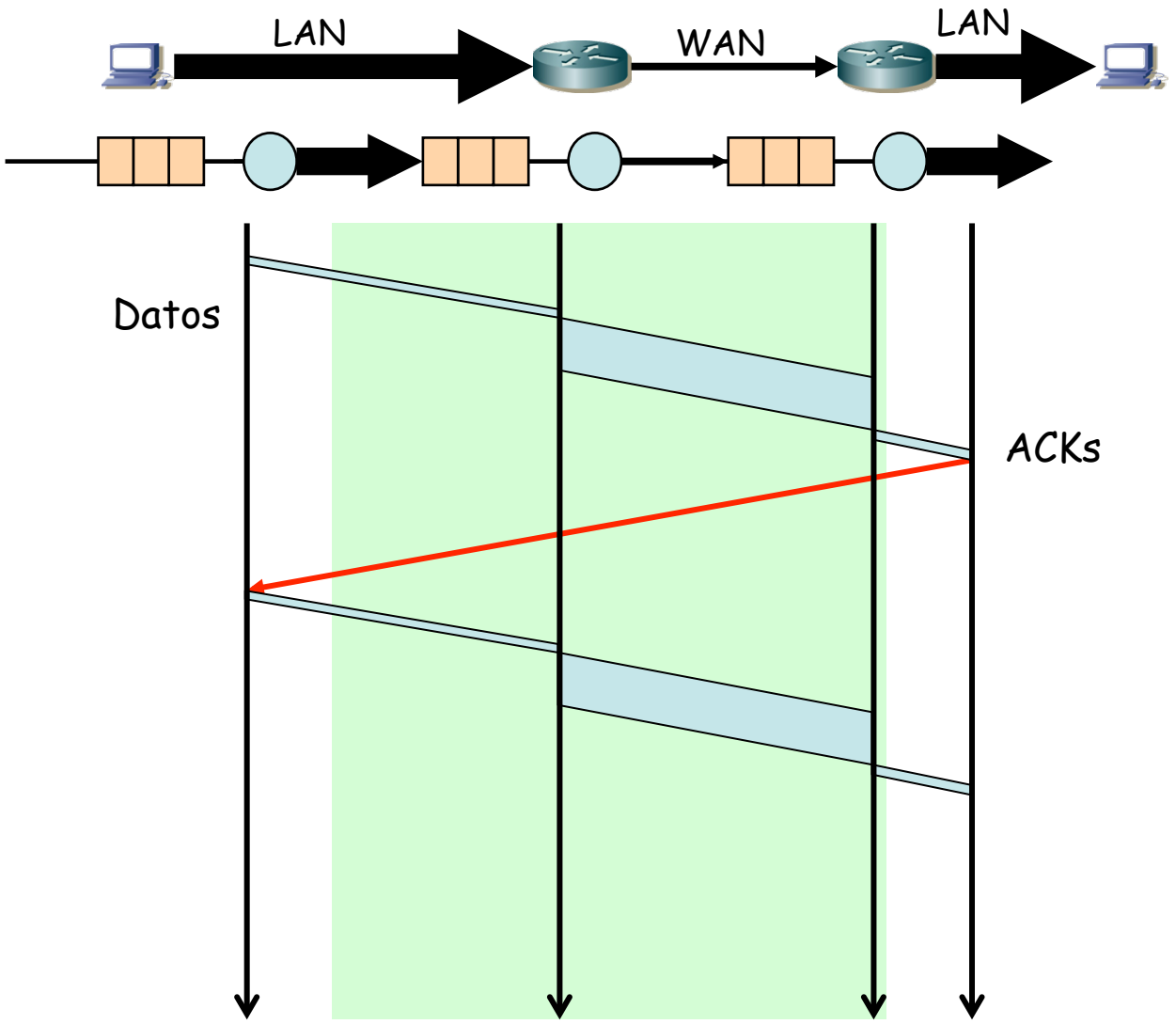
Buffer

- ¿Dónde están los paquetes?
- En buffers o siendo transmitidos



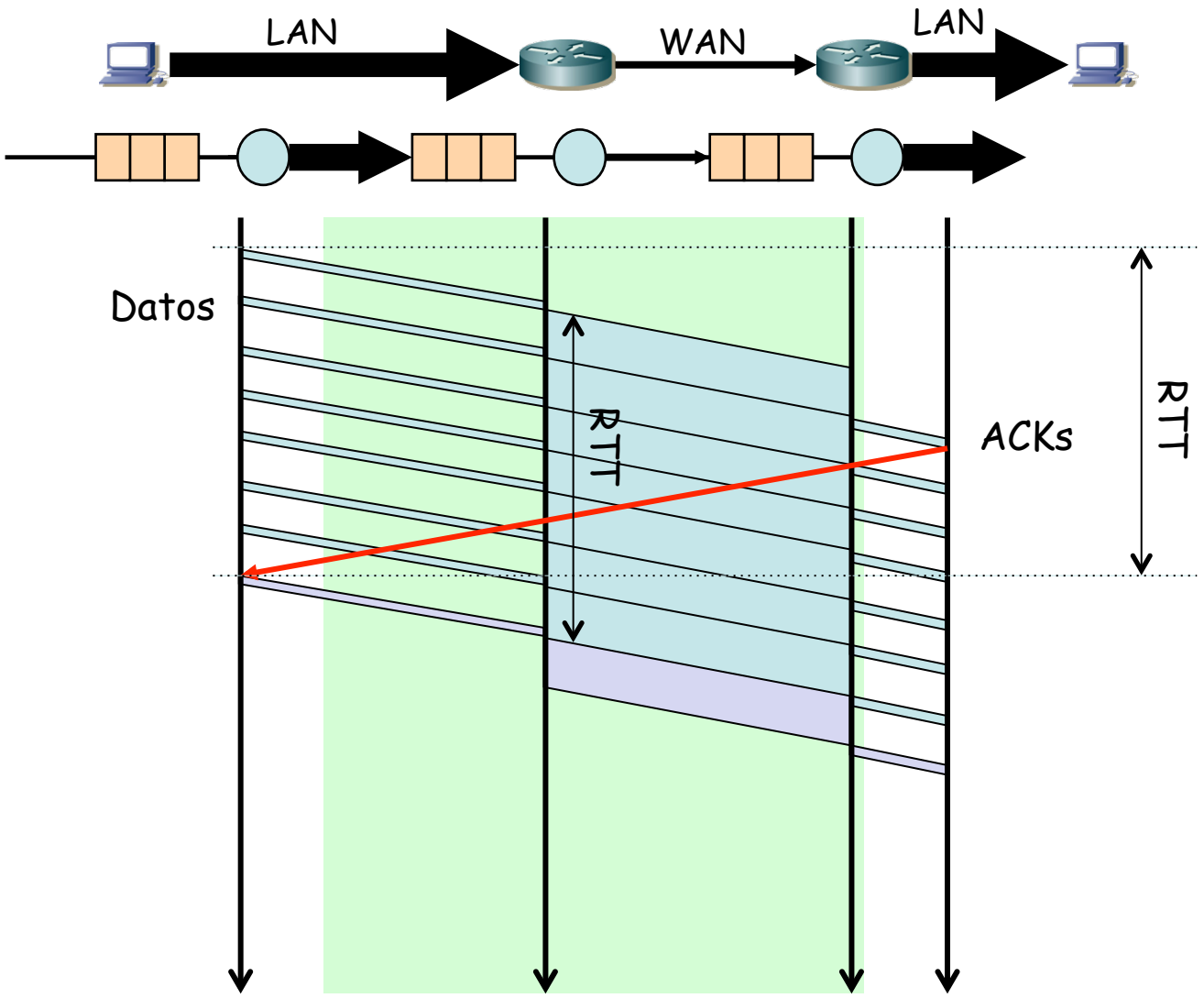


Ejemplo: LAN-WAN-LAN



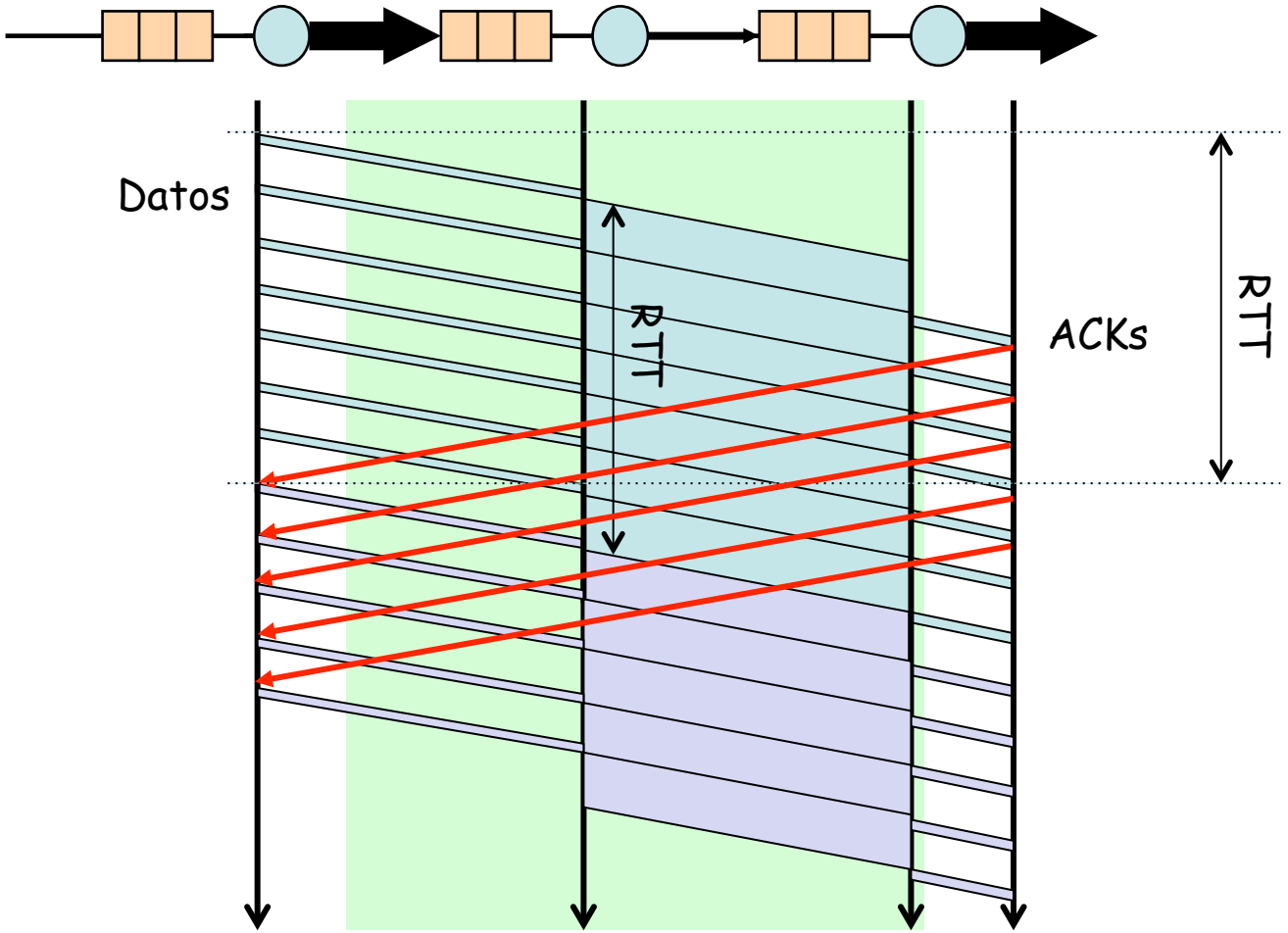
Ejemplo: LAN-WAN-LAN

- Ventana para saturar = RTT x Capacidad_bottleneck



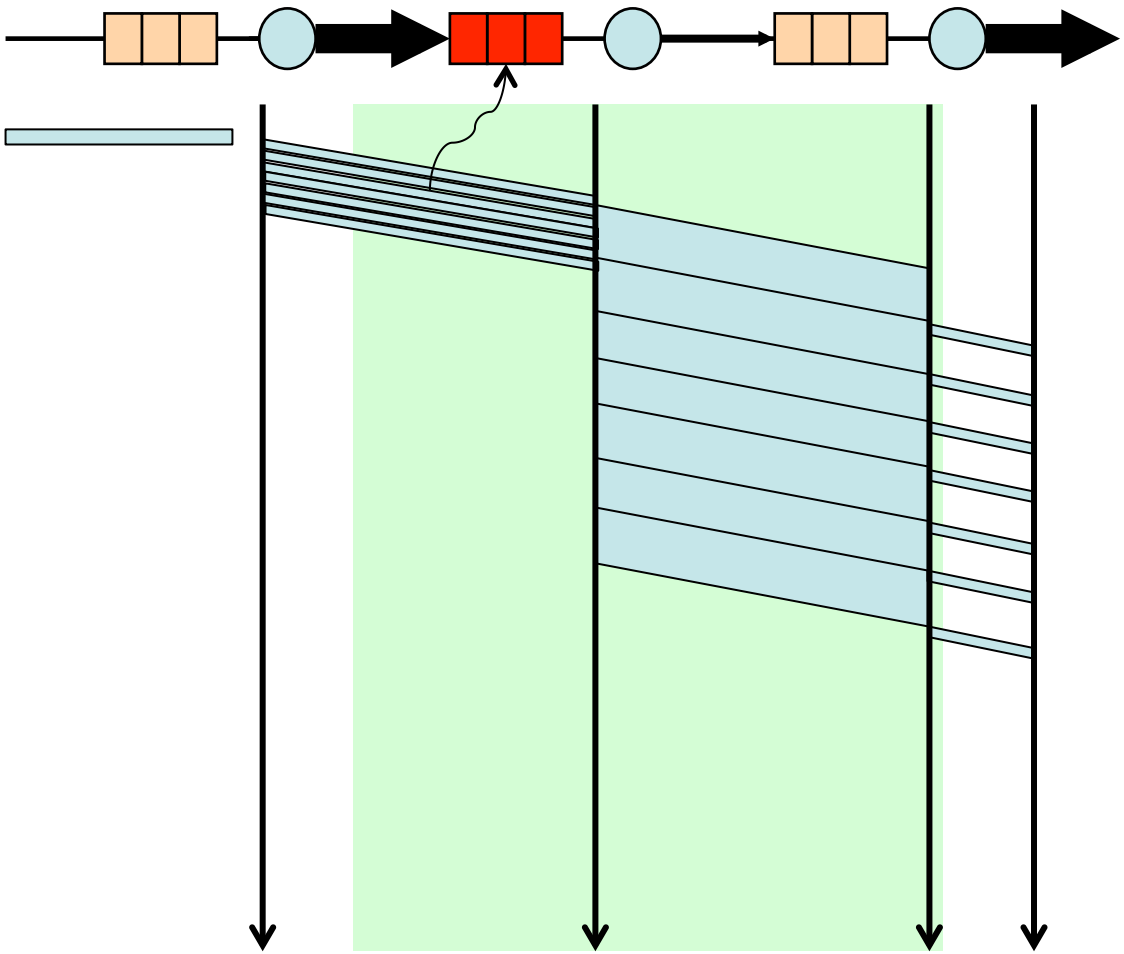
Ejemplo: LAN-WAN-LAN

- Ventana para saturar = RTT x Capacidad_bottleneck
- Y es el *clocking* aportado por los ACKs quien da la regulación



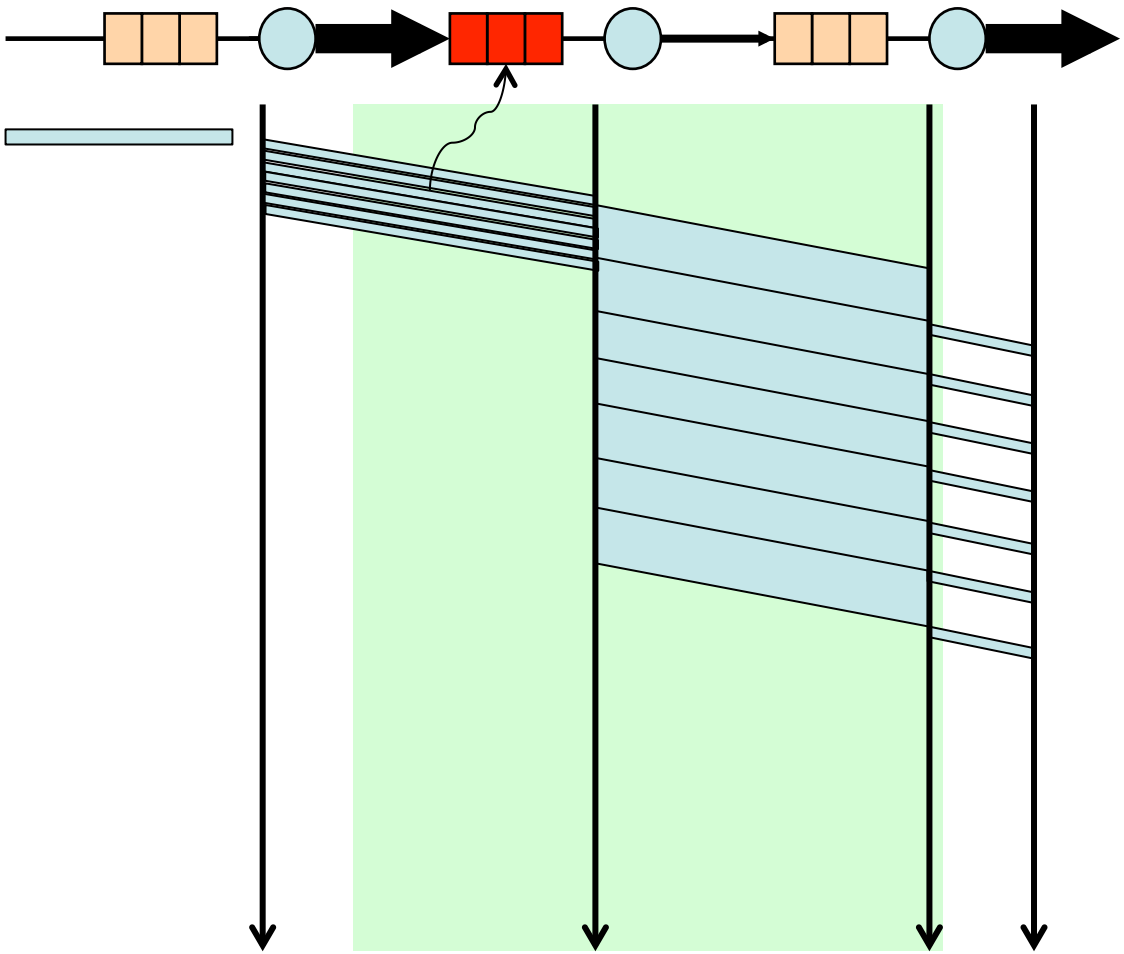
Ejemplo: LAN-WAN-LAN

- Pero si está todo confirmado, y llega un bloque de datos de aplicación
- Se puede enviar toda la ventana como una ráfaga
- Los paquetes quedarán encolados ante el cuello de botella (...)



Ejemplo: LAN-WAN-LAN

- De aquí que tradicionalmente se tienda a dimensionar los buffers para que puedan almacenar ese RTTxCapacidad
- Trayecto a 100 Mbps, RTT = 250 ms ➔ más de 3 MBytes (...)



Ejemplo: LAN-WAN-LAN

- De aquí que tradicionalmente se tienda a dimensionar los buffers para que puedan almacenar ese RTTxCapacidad
- Trayecto a 100 Mbps, RTT = 250 ms ➔ más de 3 MBytes
- Para un gran número n de flujos TCP (no sincronizados) en realidad los requisitos son más del orden $BWxRTT / \sqrt{n}$
- Hay que tener en cuenta que generalmente el tráfico no viene de un solo interfaz ni es una sola conexión