

# Transporte fiable

## Ventana deslizante y go-back-N

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

Arquitectura de Redes, Sistemas y Servicios  
Grado en Ingeniería en Tecnologías de  
Telecomunicación, 2º

# Temario

1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. Transporte fiable
7. Encaminamiento

# Temario

1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. **Transporte fiable**
7. Encaminamiento

# Nota sobre las unidades

- 1 byte son 8 bits (1B=8b)
- Aunque midiendo memoria se suelen usar prefijos k,M,G,T en potencias de 2

(por ejemplos k para  $2^{10}=1024$  M para  $2^{20}=1048576$  )

No es correcto. Hay un estandar para esto

KiB = 1024B MiB =1048576

- **En transmisión de datos se usan los prefijos del S.I.**  
 1kB =  $10^3$ B 1MB =  $10^6$ B 1GB =  $10^9$ B ...
- Las velocidades de transmisión se suelen dar en bits por segundo (kbps, Mbps...). Cuidado con la diferencia entre B y b  
 1MBps=1MB/s=8Mbps=8Mb/s
- Ejemplo en Ethernet la velocidad es 10Mbps. Un paquete de 1000B se transmite en  $(1000B \cdot 8b/B) / 10Mbps = 0.0008s = 0.8ms$

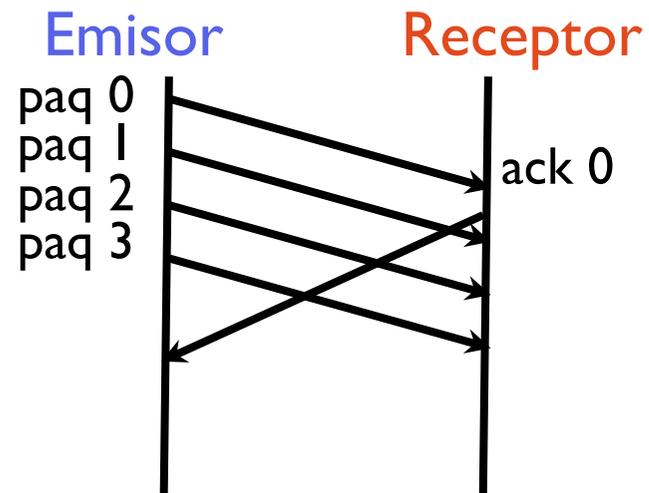
# Stop & wait: prestaciones

- Los protocolos de tipo stop & wait garantizan transporte fiable pero tienen poca eficiencia en enlaces en los que el RTT es grande comparado con el tiempo de transmisión de un paquete
- En el caso mejor transmitimos un paquete cada RTT por lo que el throughput de datos máximo es  $\text{tamaño de paquete} / \text{RTT}$
- Si hay pérdidas aun menos
- Puede ser aceptable en los casos en los que RTT y tiempo de transmisión del ACK es despreciable comparado con el tiempo de transmisión, pero necesitamos protocolos capaces de enviar a más velocidad sobre enlaces con velocidad de transmisión alta



# Protocolos más eficientes

- Para aumentar la eficiencia, se envían varios paquetes (ventana de paquetes) mientras llega el ACK  
 Varios paquetes en la red por confirmar
  - Se usan más números de secuencia que 0 y 1
  - Emisor y receptor necesitarán buffer para varios paquetes
  - Varias políticas para reaccionar a los errores
    - Go-Back N
    - Selective repeat
    - Los dos son conocidos también como Ventana deslizante (sliding window)



# Protocolo Go back-N

- Empezando por lo más simple:
- Número de secuencia en el paquete
- Se permite una ventana de N paquetes sin confirmar
- **Cada ACK confirma todos los paquetes anteriores (cumulative ACK)**
- Timeout al iniciar la ventana
- **Si caduca el timeout se retransmite la ventana**
- **Estado en el emisor**  
 base= número de secuencia mas bajo que aun no ha sido confirmado  
 nextseqnum= siguiente numero de secuencia que voy a usar  
 buffer con los paquetes enviados hasta que se confirmen y los descarte
- **Estado en el receptor**  
 expectedseqnum= siguiente numero de secuencia que espero recibir

# Eventos en el emisor (Go back-N)

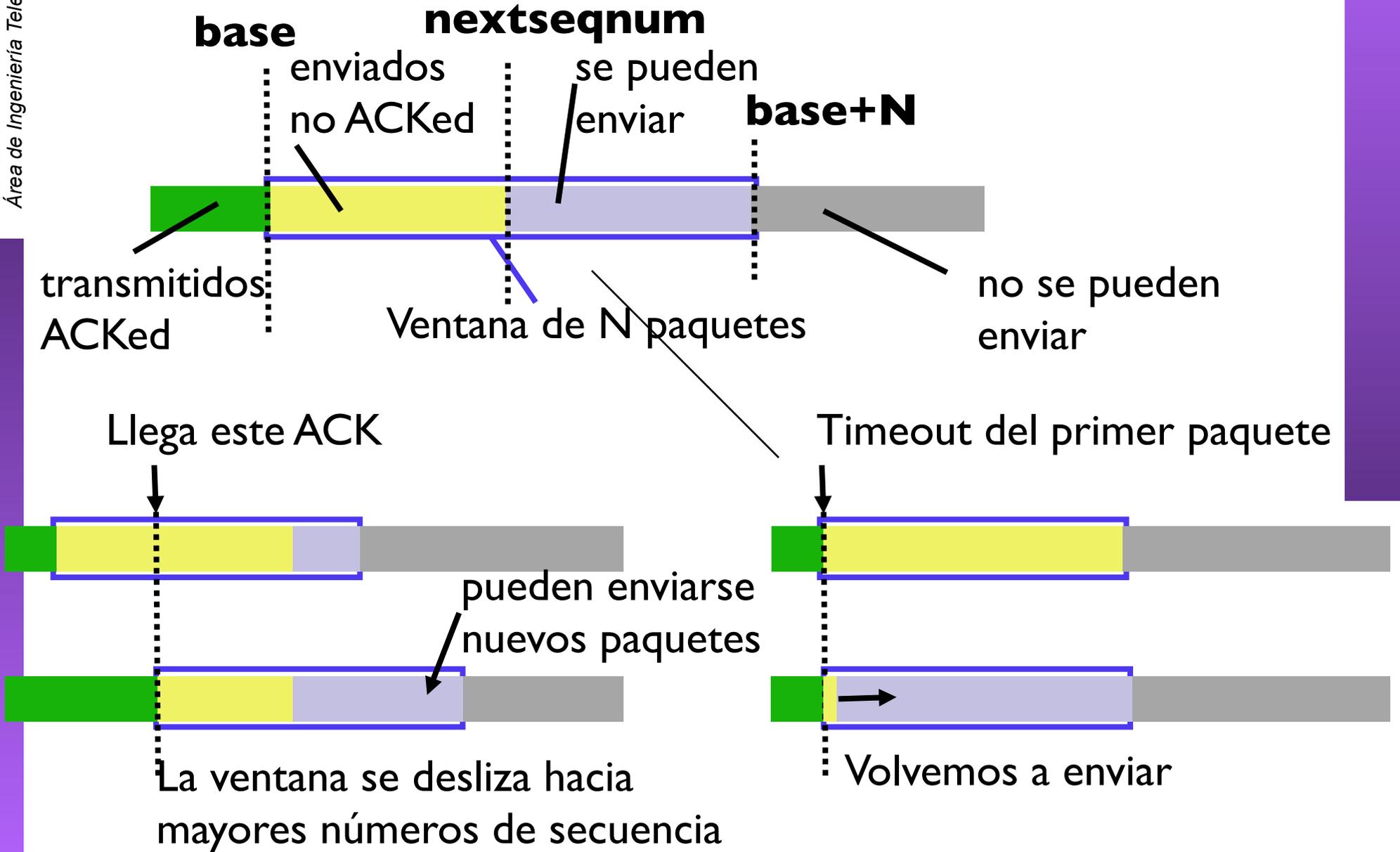
- **Recibo datos de la aplicación**
  - Si puedo enviar (  $\text{nextseqnum} < \text{base} + N$  )
    - Envía el paquete
    - Si es el primero (  $\text{nextseqnum} == \text{base}$  )
      - Inicia temporizador
    - $\text{nextseqnum}++$
  - Si no puedo enviar (  $\text{nextseqnum} == \text{base} + N$  )
    - Rechazar el dato de la aplicación
    - La aplicación tendrá que esperar (normalmente tiene un buffer para que los paquetes esperen a que se puedan enviar)

# Eventos en el emisor (Go back-N)

- **Recibo un ACK (de ack\_seq)**
  - Avanza la ventana hasta la posición confirmada (base=ack\_seq)
  - Si aun quedan paquetes pendientes recalcular el temporizador
  - Al aumentar base puede enviar los siguientes paquetes que de la aplicación hasta llenar la ventana
- **Caduca el timeout del primer paquete de la ventana**
  - Reenvía todos los paquetes enviados en la ventana (desde base hasta nextseqnum-1)
  - Reinicia el temporizador

# Eventos en el emisor (Go back n)

## Ventana deslizante

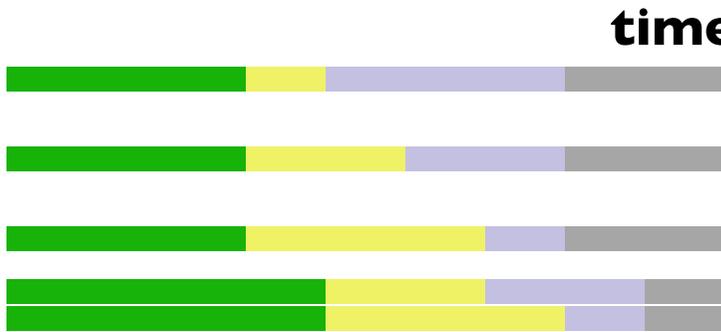
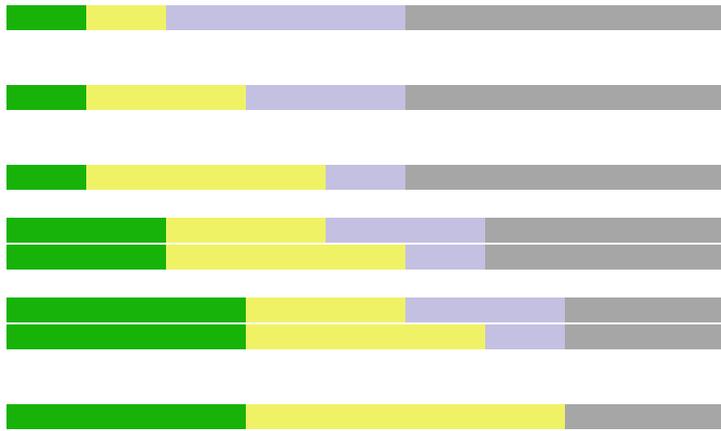


# Eventos en el receptor (Go back-N)

- **Llega el paquete esperado**
  - Envía un ACK indicando el siguiente esperado
  - Entrega datos al nivel superior
- **Llega otra paquete**
  - Envía un ACK indicando el paquete que estoy esperando
  - Descarta los datos

# Go back-N

## Ventana de 4 paquetes



...

Emisor

paq 0

paq 1

paq 2

paq 3

paq 4

paq 5

paq 2

paq 2

paq 3

paq 4

paq 5

Receptor

ack 1 Ok 0

ack 2 Ok 1

ack 2

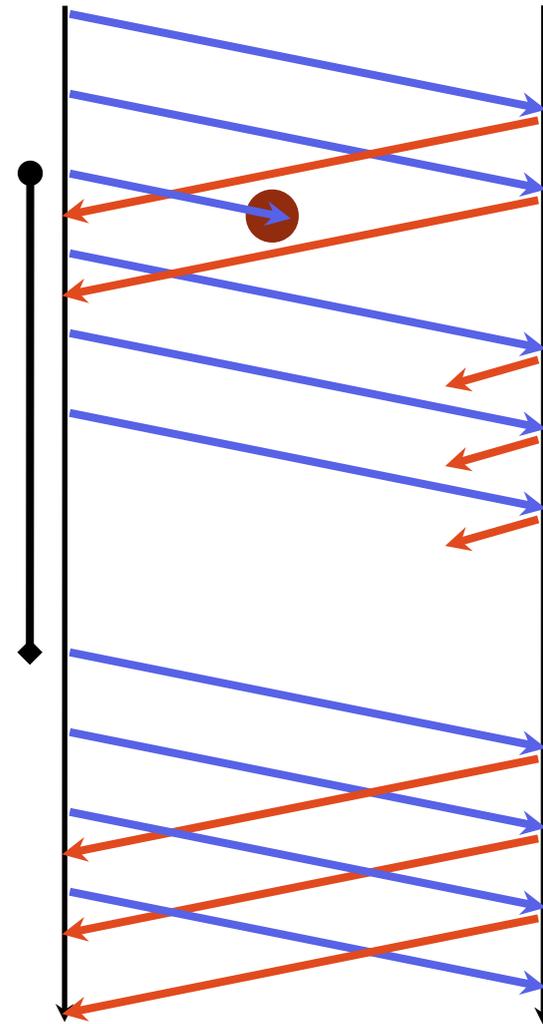
ack 2

ack 2

ack 3 Ok 2

ack 4 Ok 3

ack 5 Ok 4

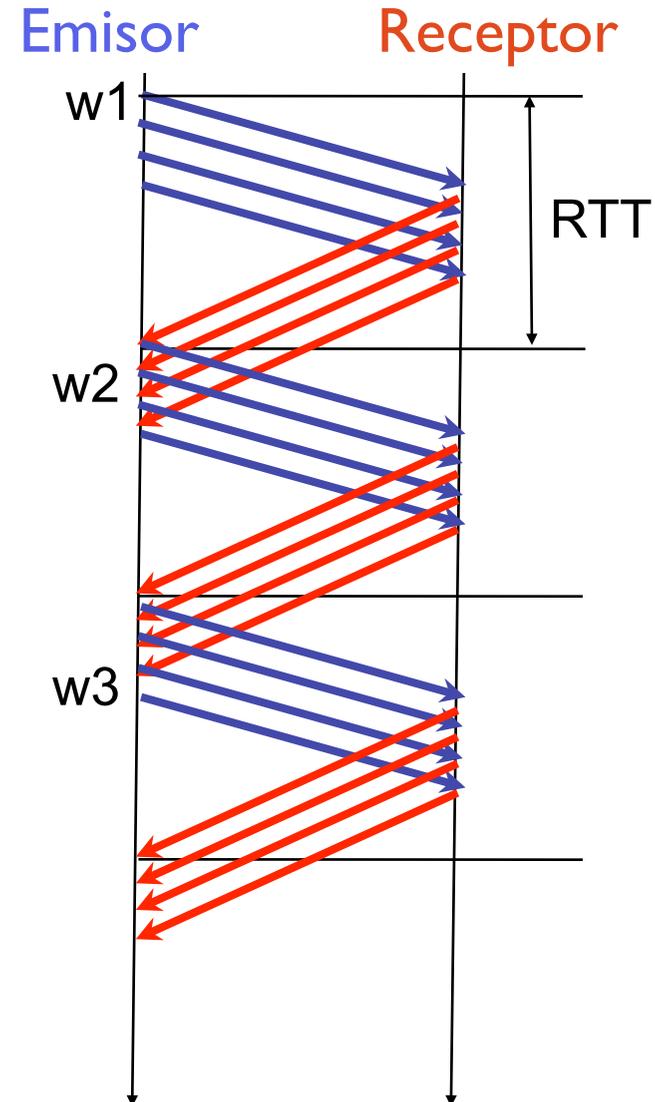


# Resumen Go-back N

- Simple (¿sabríais programarlo?)
- Más eficiente que stop & wait
- ¿Cuál es la velocidad máxima?
- $N s / RTT$  ?
- ¿y si el tiempo de transmisión no es despreciable?
  
- Mejoras fáciles a go-back N?

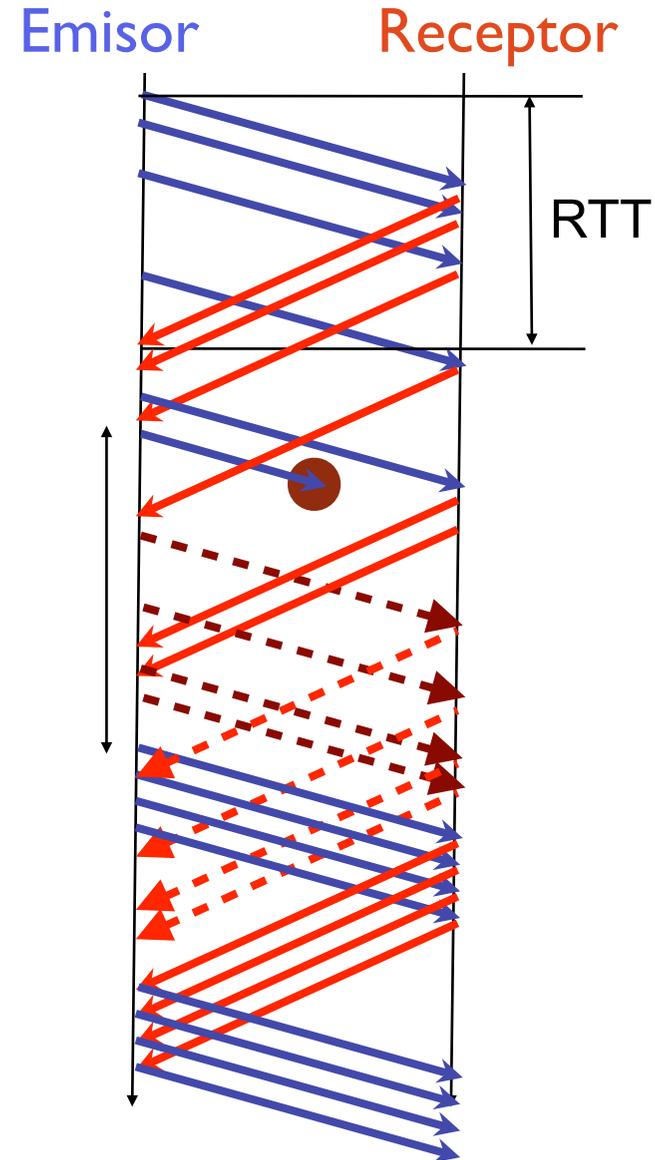
# Velocidad de go-back N

- Supongamos que la aplicación envía constantemente, siempre hay datos pendientes
- Enviamos  $w=N$  s bytes cada RTT
- La ventana empieza a repetirse en cuanto recibimos el primer ACK, no el último
- La velocidad máxima es  $w/RTT = N$  s/RTT
- Pero solo si la ventana se envía entera antes de que vuelva un ACK



# Velocidad de go-back N

- Si hay errores habrá timeouts y retransmisiones
- Si la aplicación no envía datos irá mas despacio
- $w/RTT$  es un limite del protocolo.  
 Depende del valor que elijamos para N y s
- Analizar las prestaciones de estos casos es más difícil



# Velocidad de go-back N

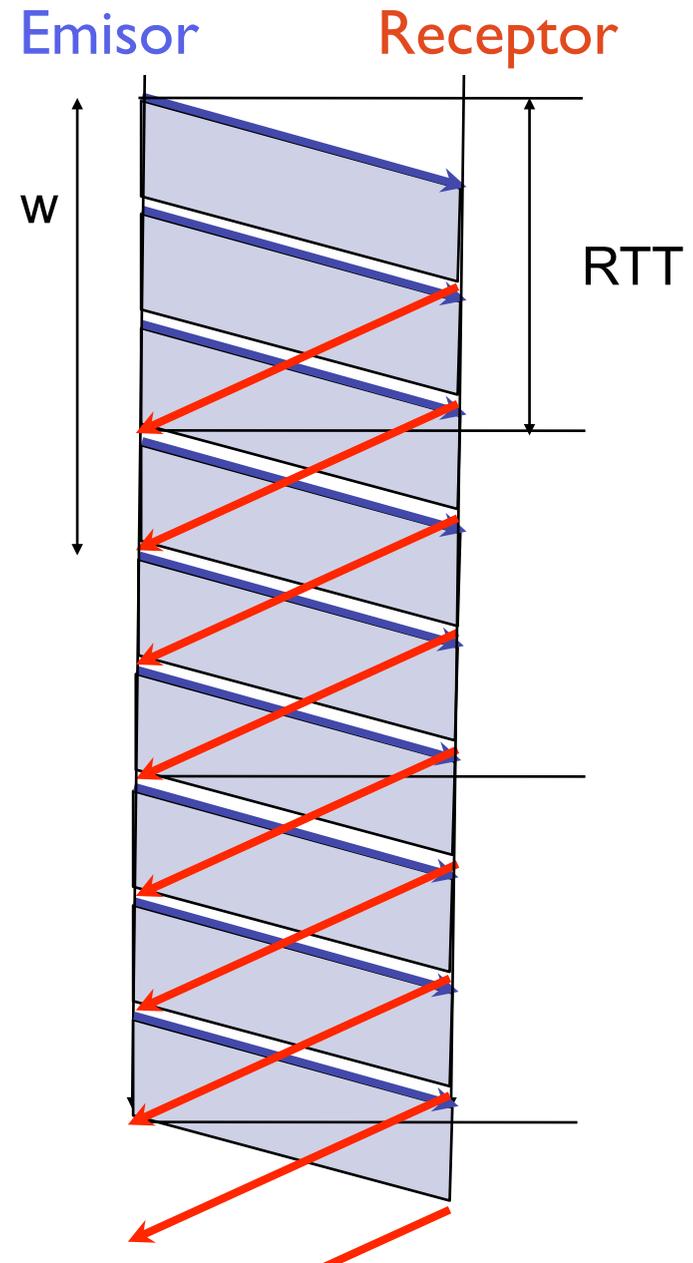
- Puede que no de tiempo a enviar la ventana antes de que vuelva el ACK

Ejemplo con  $N=4$

El throughput aquí es

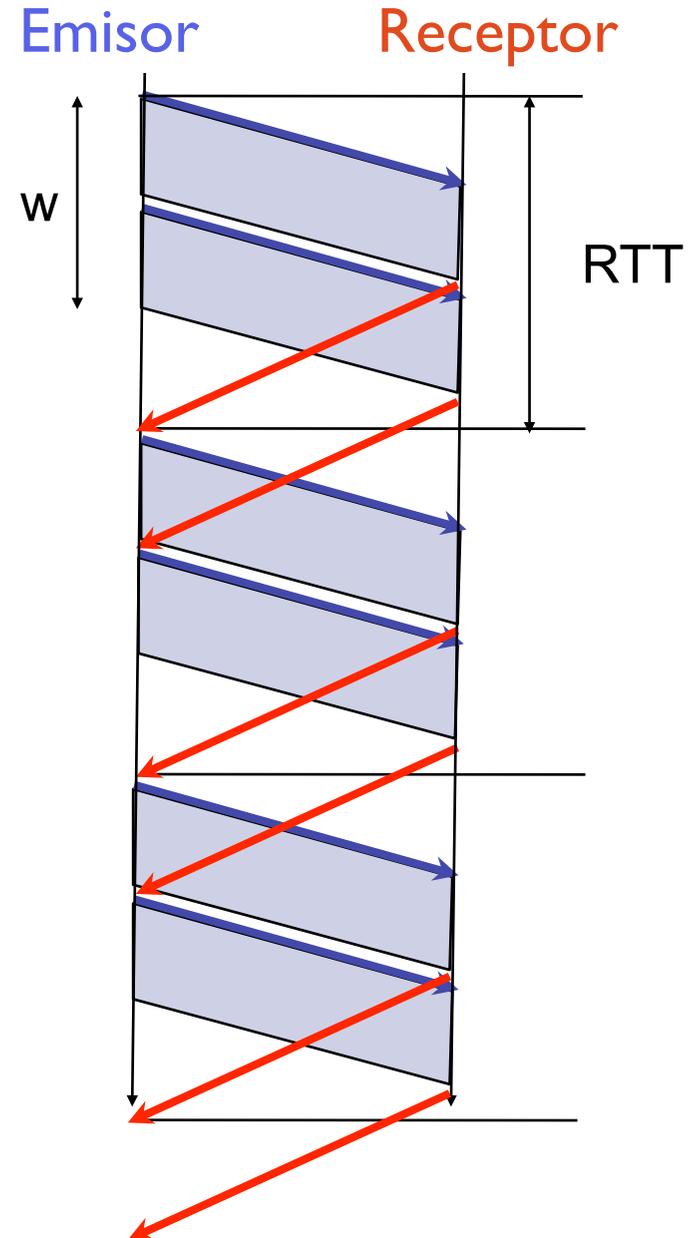
$$3s/RTT < Ns/RTT$$

- El problema es que no nos da tiempo a enviar  $w$  bytes en el tiempo  $RTT$
- Eso ocurre si  $w/C > RTT$  o sea si  $C < w/RTT$
- El límite es  $w/RTT$  salvo que el canal tenga menor  $C$  ( $V_{tx}$ )



# Velocidad de go-back N

- En resumen
- Si  $w < C * RTT$   
 El límite de velocidad lo pone el protocolo  $w/RTT$   
 La ventana cabe en el canal
- Si  $w > C * RTT$   
 El límite lo pone el canal  
 La ventana no cabe en el canal  
 Estamos aprovechando al máximo el canal



# Conclusiones

- Se pueden construir protocolos que garantizan transporte fiable y son más eficientes que stop & wait
- Ventana deslizante
  - Go back N
- Como mejorar go-back-N
  - Aprovechar los paquetes que llegan aunque se hayan perdido los anteriores?