Transporte fiable

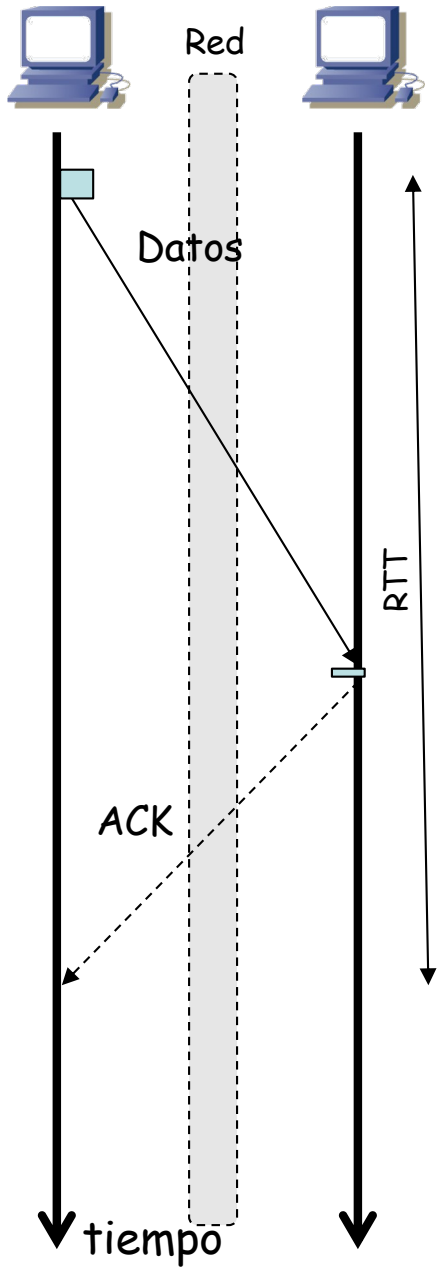
Area de Ingeniería Telemática
http://www.tlm.unavarra.es

Arquitectura de Redes, Sistemas y Servicios

Pipelining

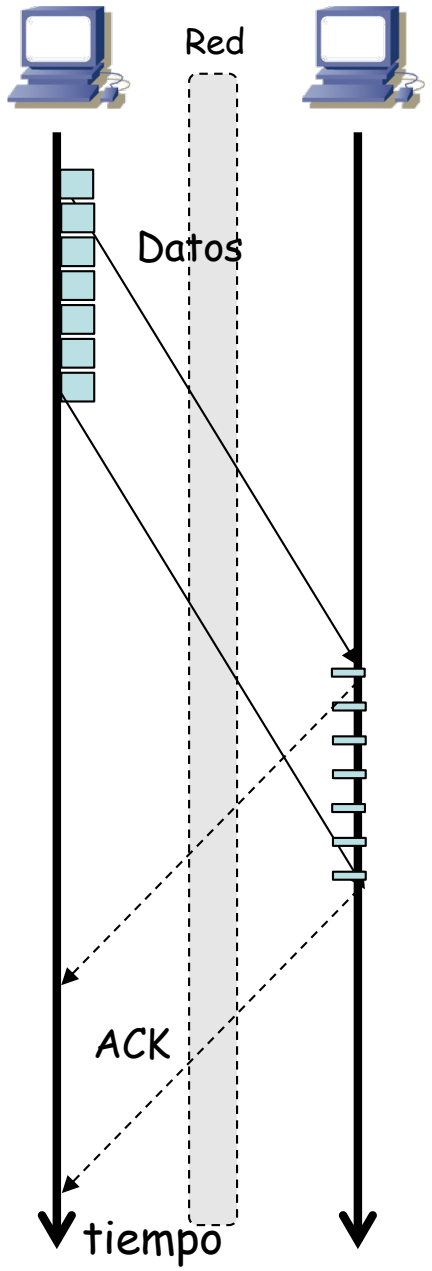
Stop&Wait: Efecto del RTT

- Paquete de 1518 bytes enviado por interfaz a 100Mb/s
- Tiempo de transmisión de 121μs
- Enlace transatlántico: RTT de 60ms
- Camino europeo: RTT de decenas de ms
- 10ms es aprox. 100x ese tiempo de tx
- También el tiempo de transmisión del ACK es despreciable
- La fuente está mucho tiempo esperando a que el paquete atraviese la red
- Su enlace al primer conmutador queda libre rápidamente
- En la figura no está a escala el tiempo de transmisión y el RTT, el de transmisión es mucho más pequeño



Pipelining

- Mandamos varios paquetes sin esperar ACK del anterior
- Ejemplo con datos:
  - RTT 10ms, L=1250bytes @ 100Mb/s, tiempo tx paquete = 0.1ms
  - Stop&wait: L/RTT aprox. 1Mb/s
  - Si enviamos 10 paquetes, tiempo tx total 1ms
  - El RTT hasta el último ACK: 10ms+1ms = 11ms
  - Enviamos 10xL bytes en 11ms: 9Mb/s
- Puede haber múltiples paquetes en vuelo
- Hay que numerarlos para poder confirmarlos
- Podemos numerar los paquetes o los bytes en ellos
- Se ha podido perder uno cualquiera de ellos, o un ACK (o más)
- El emisor debe guardarlos hasta ser confirmados
- Receptor puede tener que reordenarlos



upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

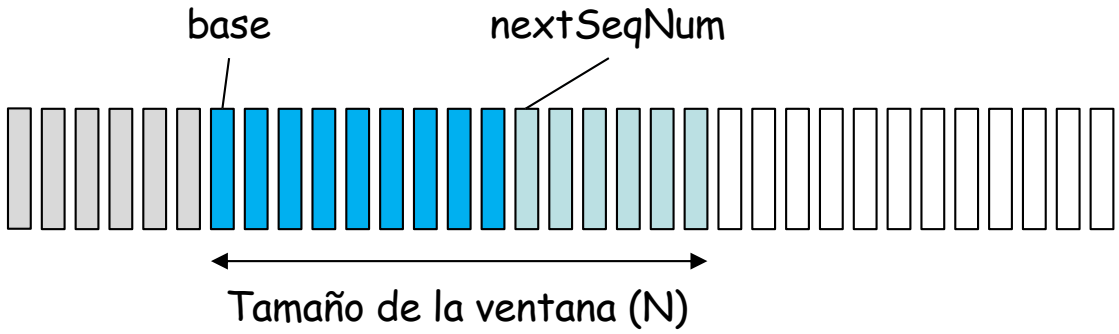
ARQUITECTURA DE REDES, SISTEMAS Y SERVICIOS

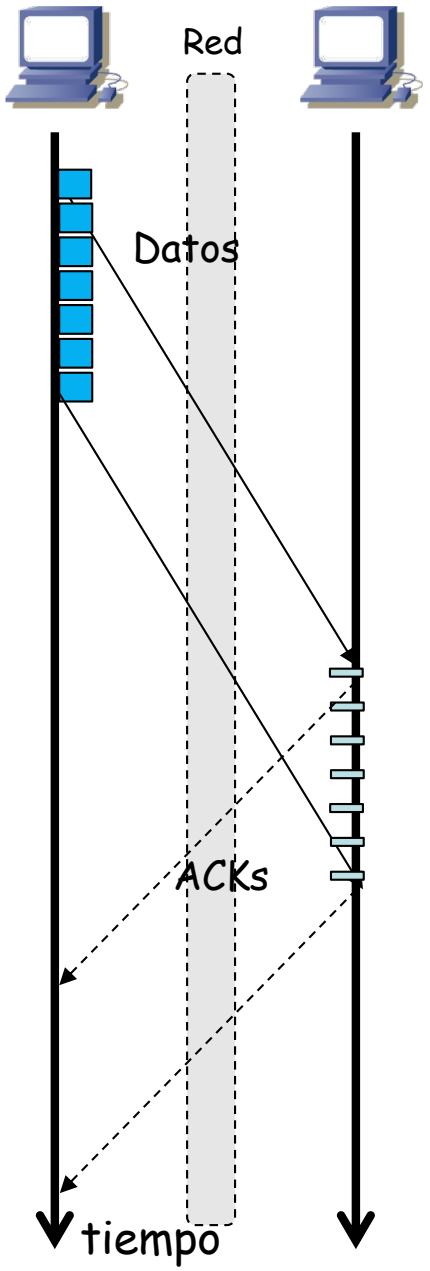
Área de Ingeniería Telemática

Go-Back-N

Go-Back-N

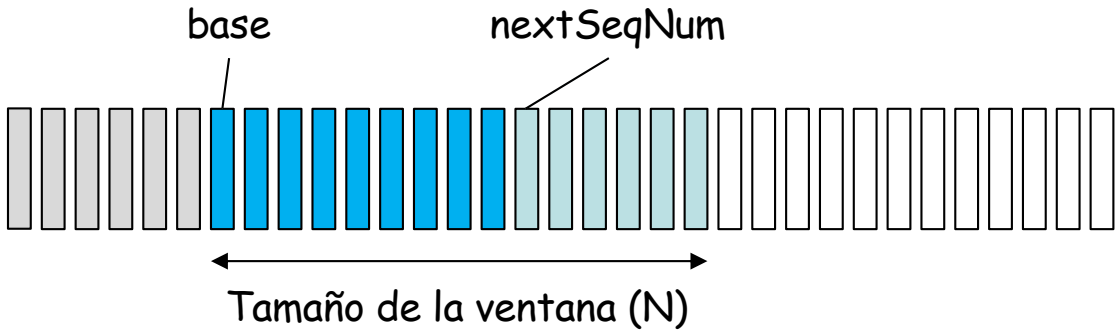
- Numerar los paquetes; confirmaciones acumuladas
- Hasta N paquetes enviados sin confirmar ("ventana" de tamaño N)
- Aritmética módulo para numerar los paquetes
- Timer asociado al envío del primer paquete de la ventana
- Si caduca (timeout) se reenvía todo lo que haya en la ventana enviado sin confirmar
- Y se reinicia el timer
- *base* = nº de secuencia más bajo sin confirmar
- *nextSeqNum* = siguiente nº de secuencia a usar
- En el receptor: *expectedSeqNum* = siguiente número de secuencia esperado

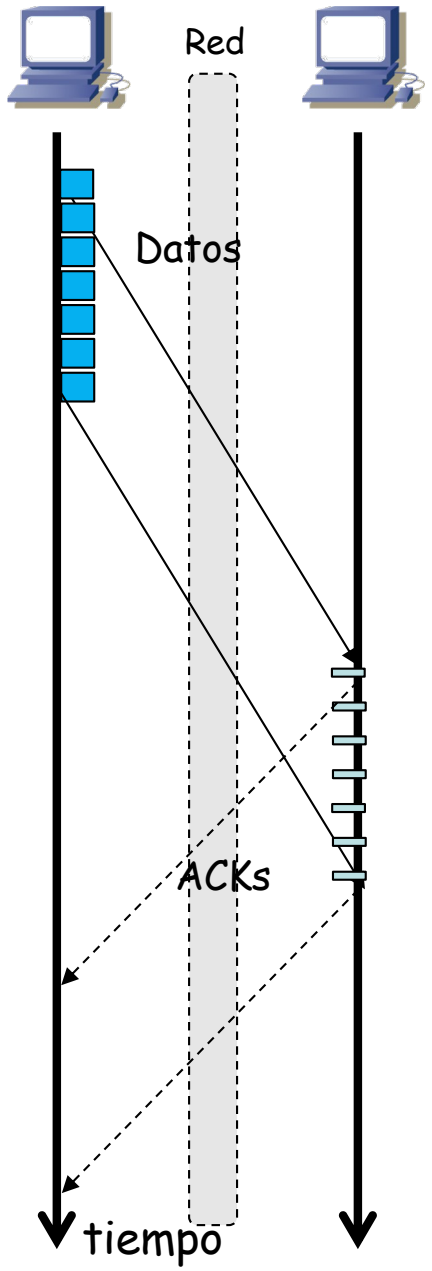




Go-Back-N: ACKs

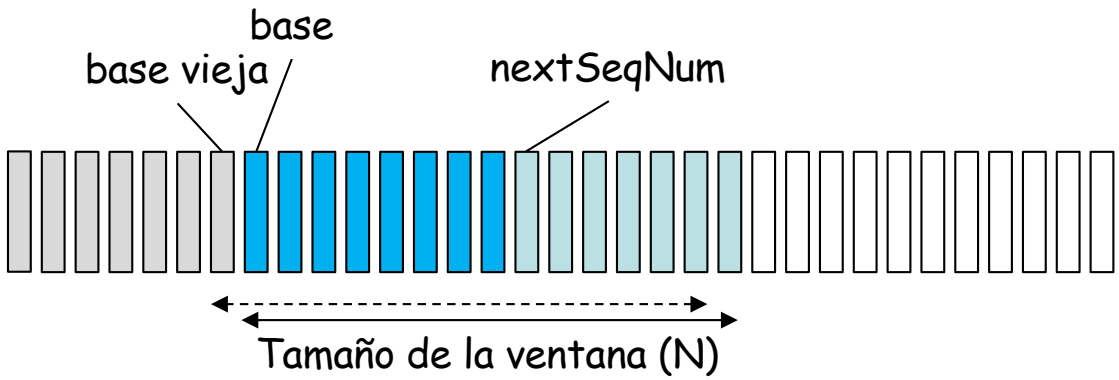
- Receptor espera paquete K, al recibirlo envía ACK diciendo que espera paquete K+1
- Un ACK “desliza” la ventana
- Ejemplo:
  - ACK del primer paquete enviado

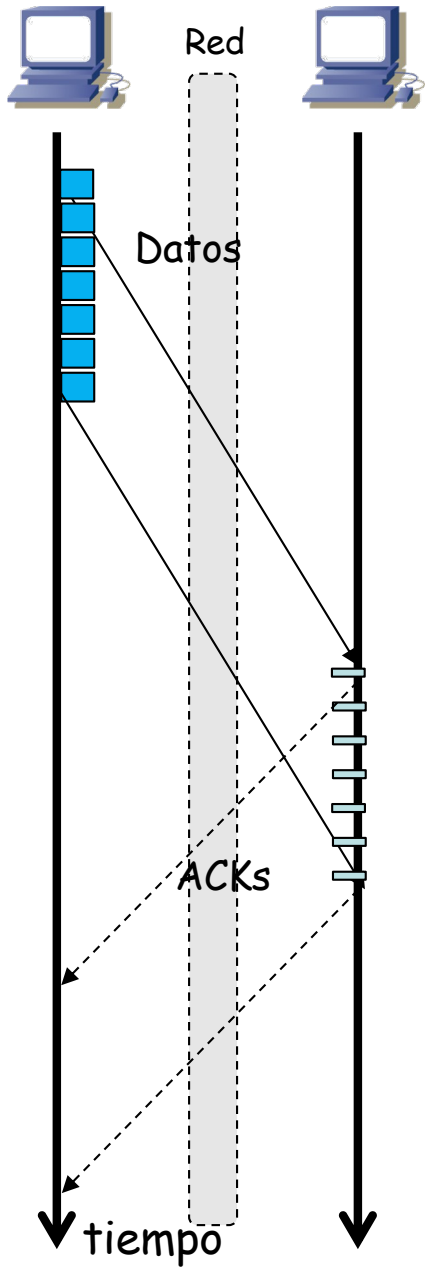




Go-Back-N: ACKs

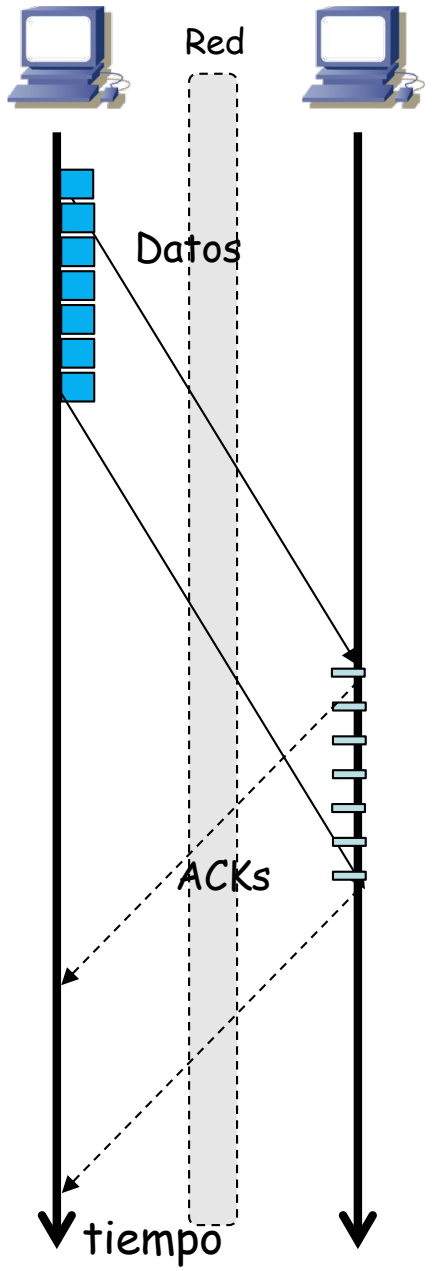
- Receptor espera paquete K, al recibirlo envía ACK diciendo que espera paquete K+1
- Un ACK “desliza” la ventana
- Ejemplo:
  - ACK del primer paquete enviado
  - Sale de la ventana (aumenta *base*)
  - Un nuevo hueco en la ventana por la derecha
  - Aún quedan paquetes en vuelo, se reinicia el timer

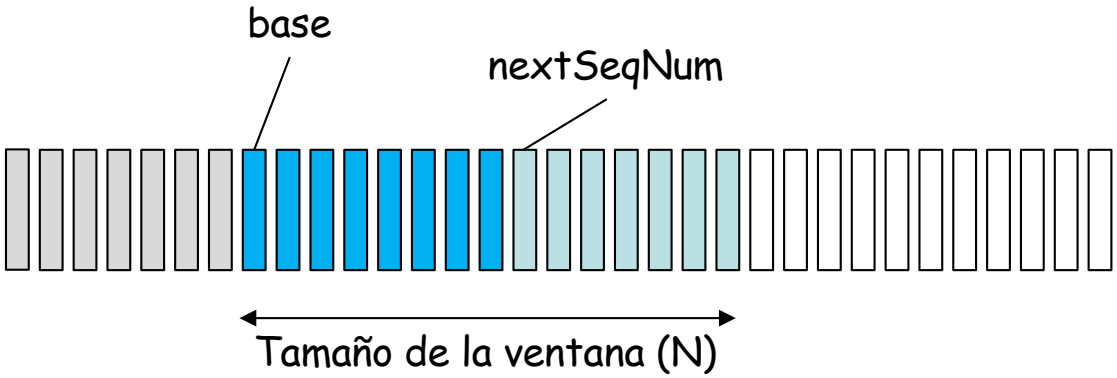




Go-Back-N: ACKs

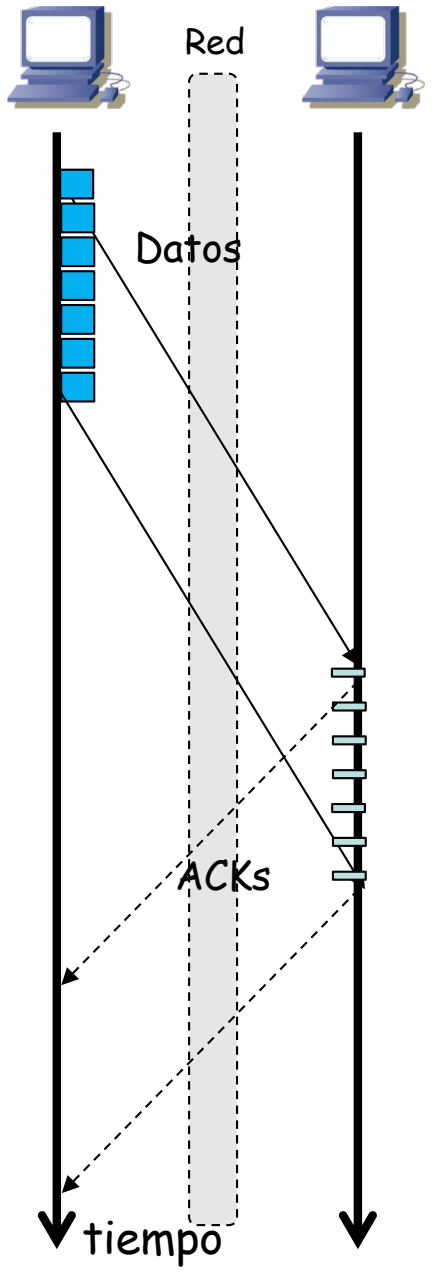
- Ejemplo:
  - Receptor espera paquete K, recibe K y K+1, envía ACK diciendo que espera paquete K+2
  - El ACK “desliza” la ventana

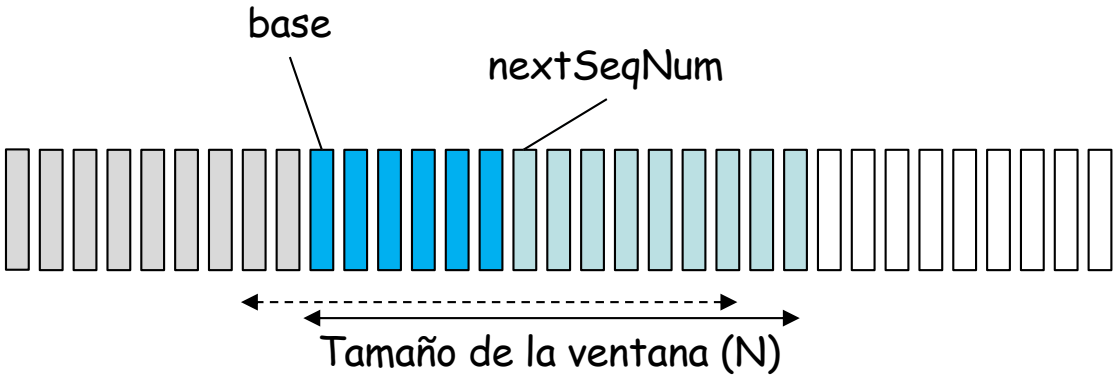




Go-Back-N: ACKs

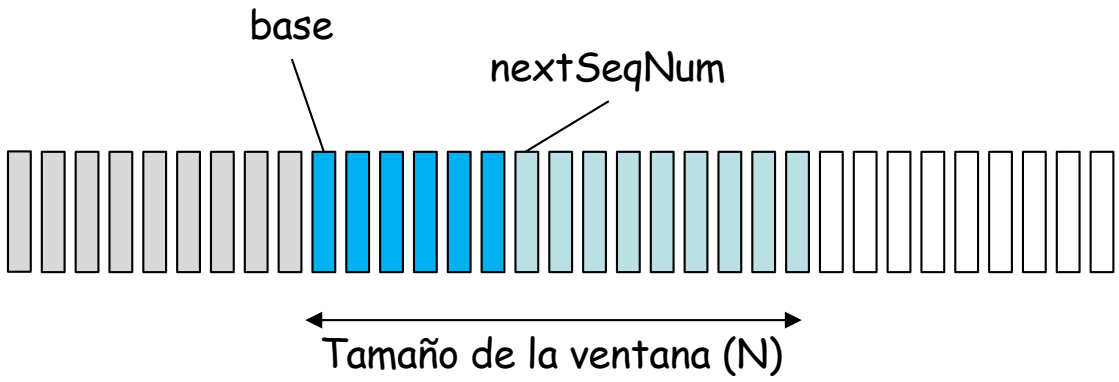
- Ejemplo:
  - Receptor espera paquete K, recibe K y K+1, envía ACK diciendo que espera paquete K+2
  - El ACK “desliza” la ventana
  - Dos paquetes salen de la ventana
  - Dos nuevos huecos (números de secuencia que se pueden emplear)
  - Aún quedan paquetes en vuelo, se reinicia el timer

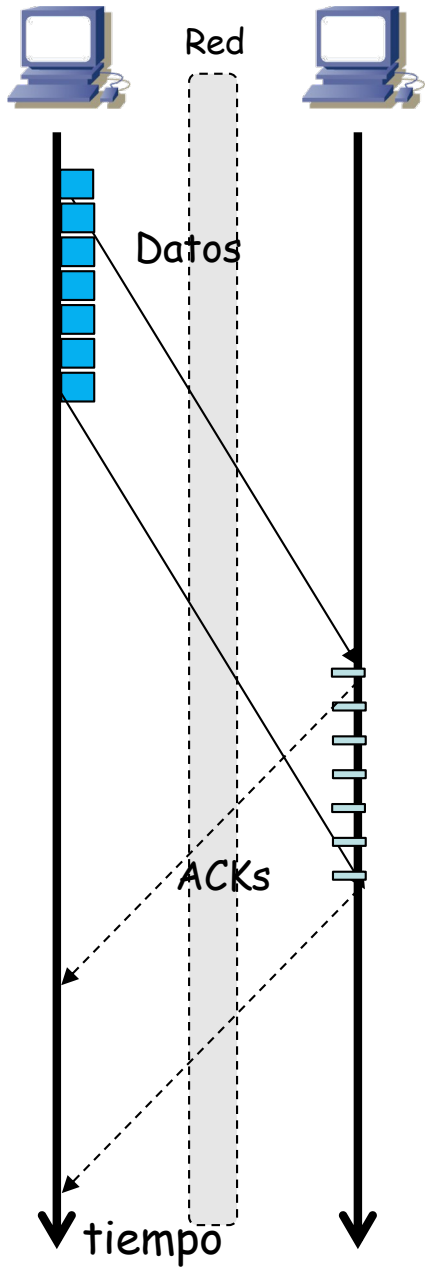




Go-Back-N: ACKs

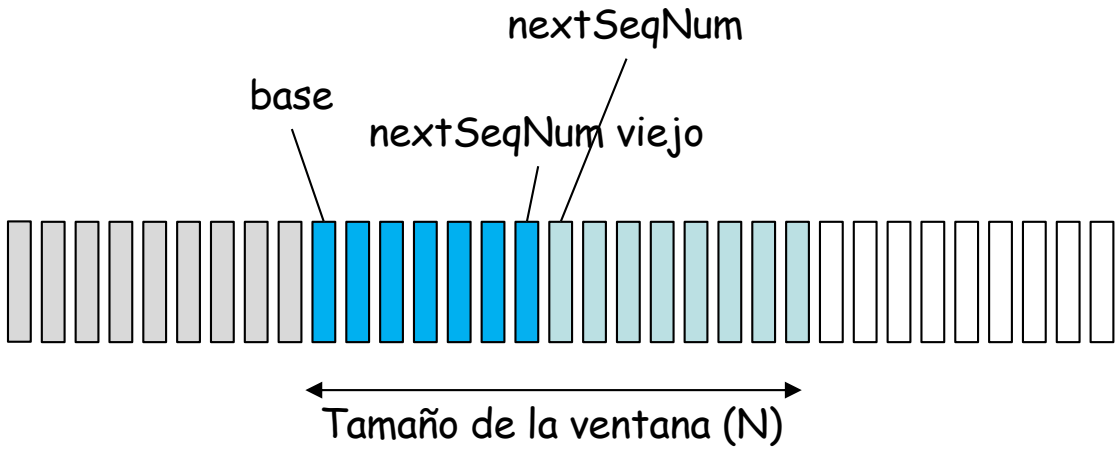
- Nuevos datos a enviar de la aplicación pueden emplear el espacio en la ventana
- Ejemplo:
  - Aplicación envía un nuevo paquete

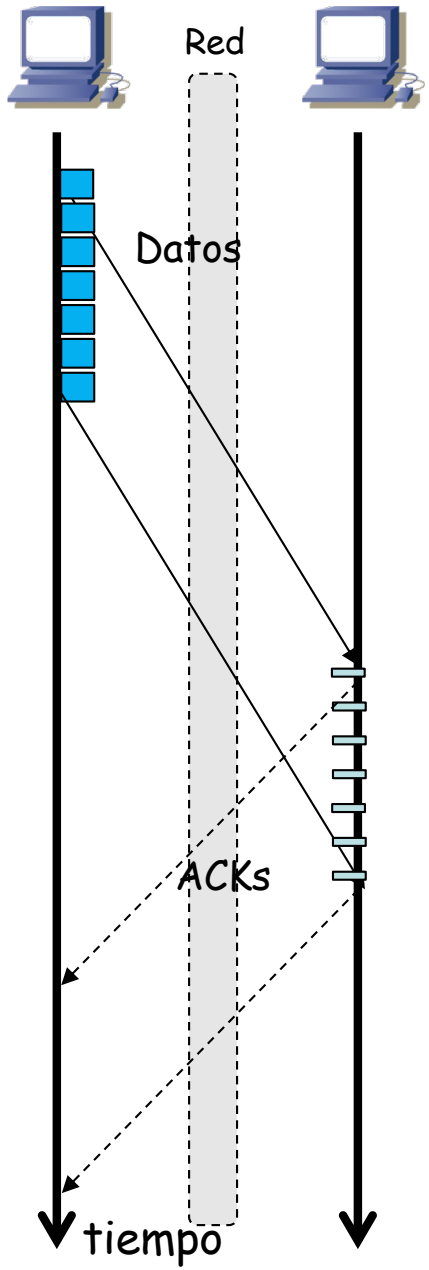




Go-Back-N: ACKs

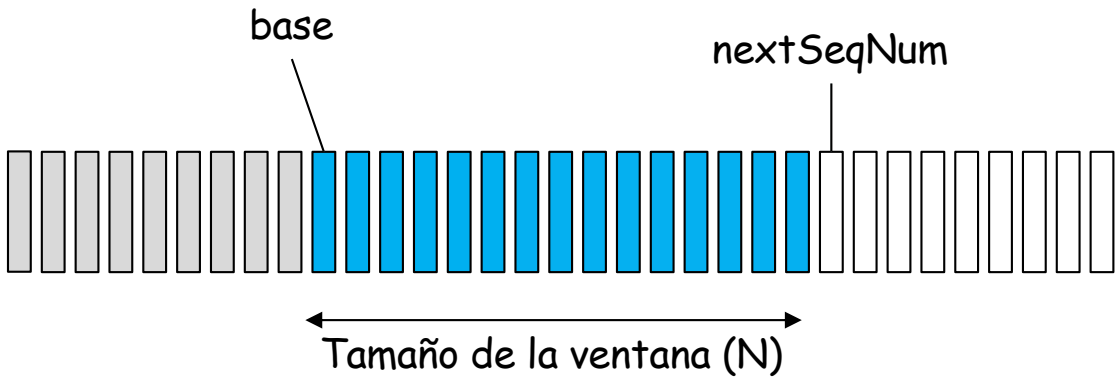
- Nuevos datos a enviar de la aplicación pueden emplear el espacio en la ventana
- Ejemplo:
  - Aplicación envía un nuevo paquete
  - Si hay "hueco" en la ventana se envía con el nº de secuencia correspondiente
  - La ventana no se mueve

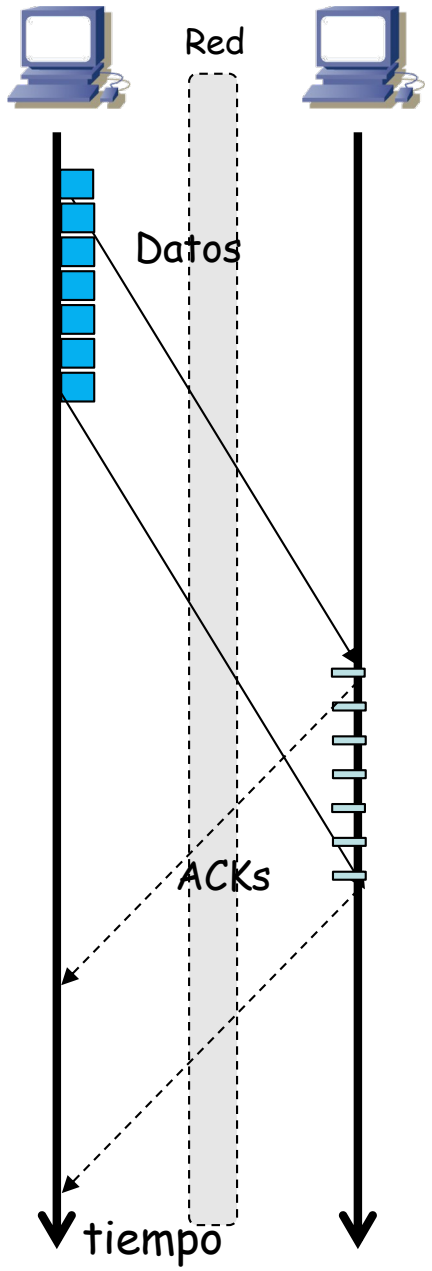




Go-Back-N: ACKs

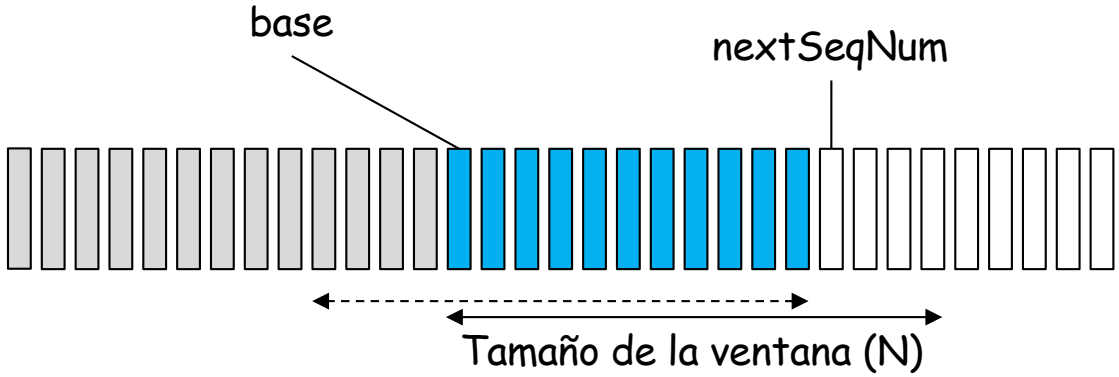
- Si la aplicación entrega más datos sin llegar ACKs se puede “llenar” la ventana
- *nextSeqNum* está ahora fuera de la ventana
- No se pueden enviar más paquetes hasta que se confirmen
- Las confirmaciones son acumuladas, así que confirman todo lo anterior a su número de ACK
- Confirmaciones desplazan la ventana
- Por ejemplo, ACK confirma hasta base+4 paquetes
- Normalmente lo que dice el ACK es el primer paquete que espera (anteriores los confirma)

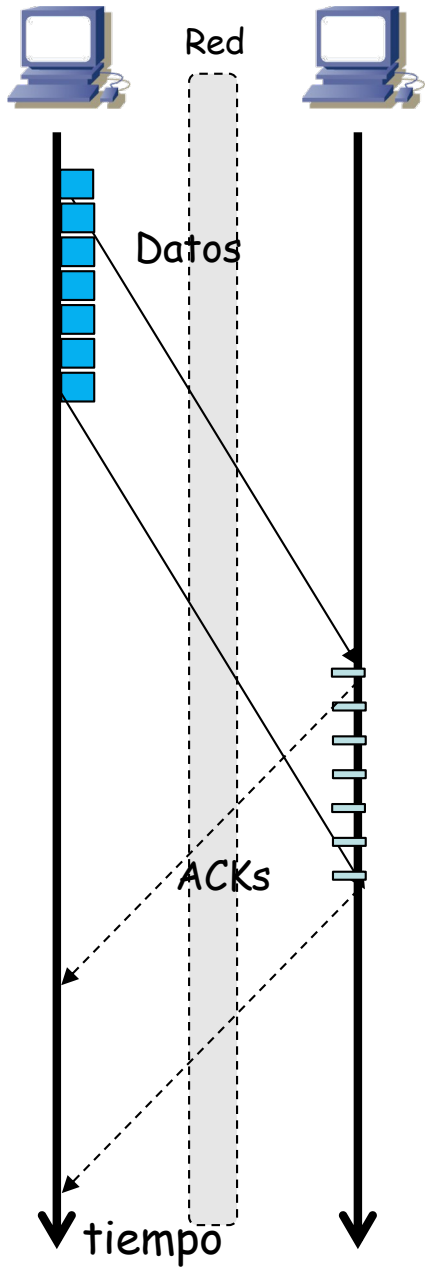




Go-Back-N: ACKs

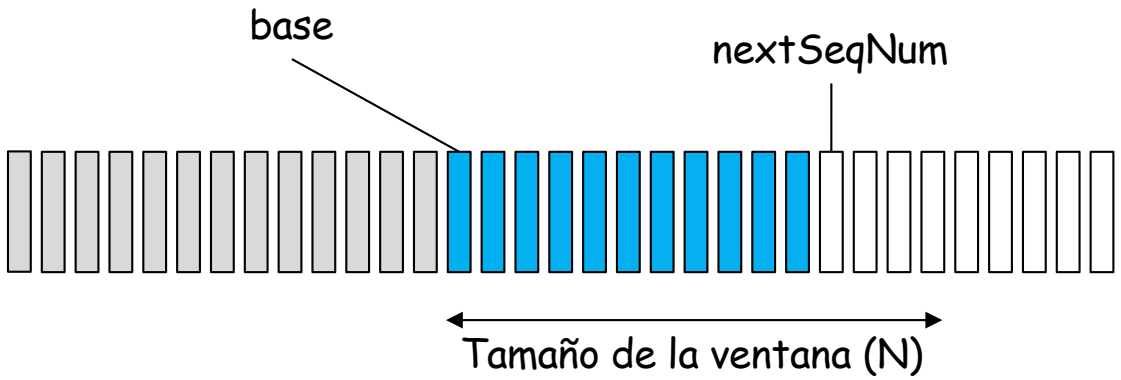
- Si la aplicación entrega más datos sin llegar ACKs se puede “llenar” la ventana
- *nextSeqNum* está ahora fuera de la ventana
- No se pueden enviar más paquetes hasta que se confirmen
- Las confirmaciones son acumuladas, así que confirman todo lo anterior a su número de ACK
- Confirmaciones desplazan la ventana
- Por ejemplo, ACK confirma hasta base+4 paquetes
- Normalmente lo que dice el ACK es el primer paquete que espera (anteriores los confirma)
- Nuevo hueco en la ventana para enviar 4 paquetes
- Aún quedan paquetes en vuelo, se reinicia el timer





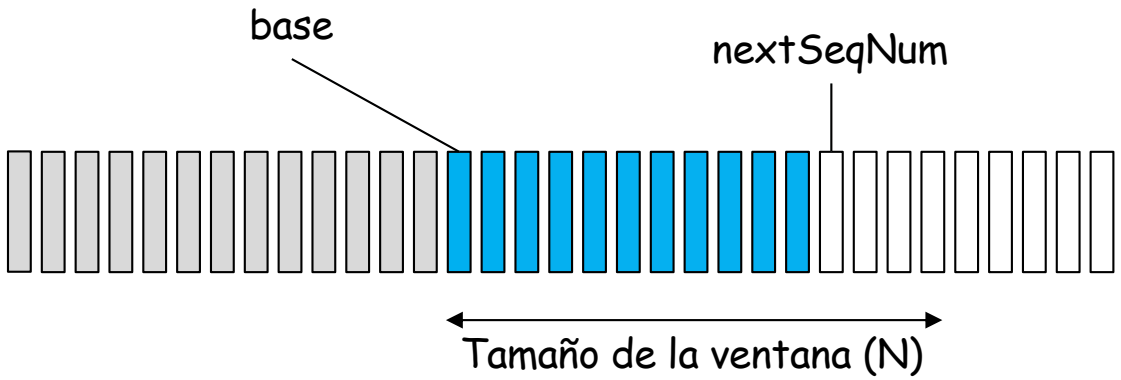
Timeout

- Se reenvían todos los paquetes en vuelo (en la ventana)
- Se reinicia el timer
- Receptor no necesita guardar paquetes desordenados
- Ejemplo:
  - Receptor espera paquete K
  - Recibe paquete K+1
  - Puede descartarlo ya que cuando caduque el timer le reenviarán el K y el K+1



La ventana se “desliza”

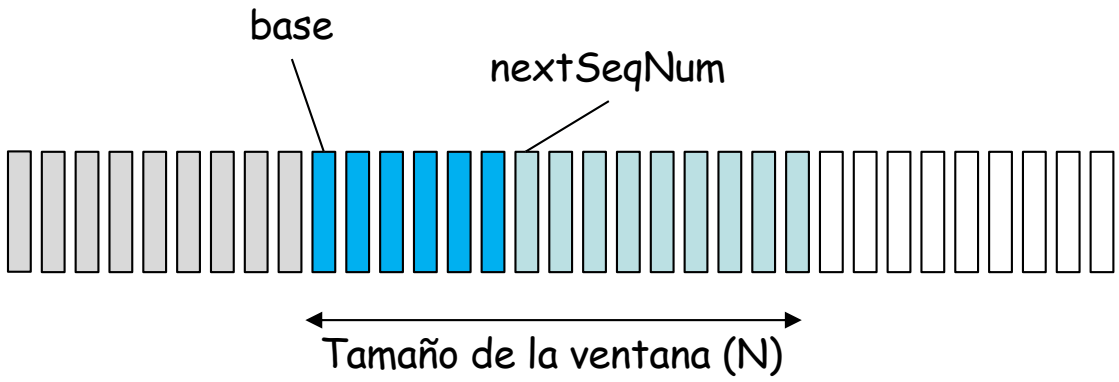
- Se llama a estos protocolos, de “ventana deslizante”
- O “*sliding window*”
- La ventana tiene un tamaño, lo que implica un número máximo de paquete en vuelo sin confirmar
- El receptor puede controlar el ritmo al que envía la fuente controlando el desplazamiento de la ventana (control de flujo)

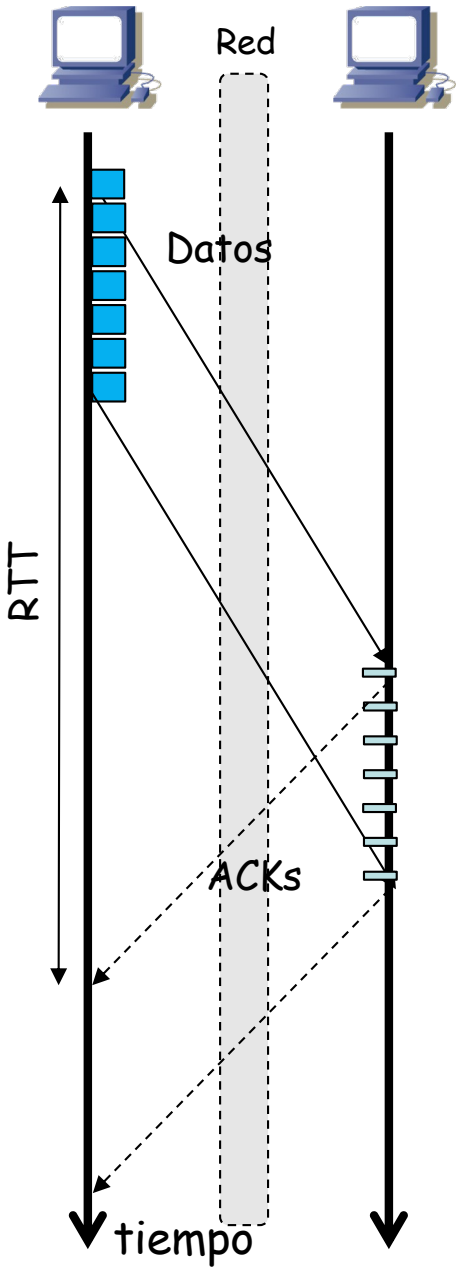


Go-Back-N: rendimiento

Throughput

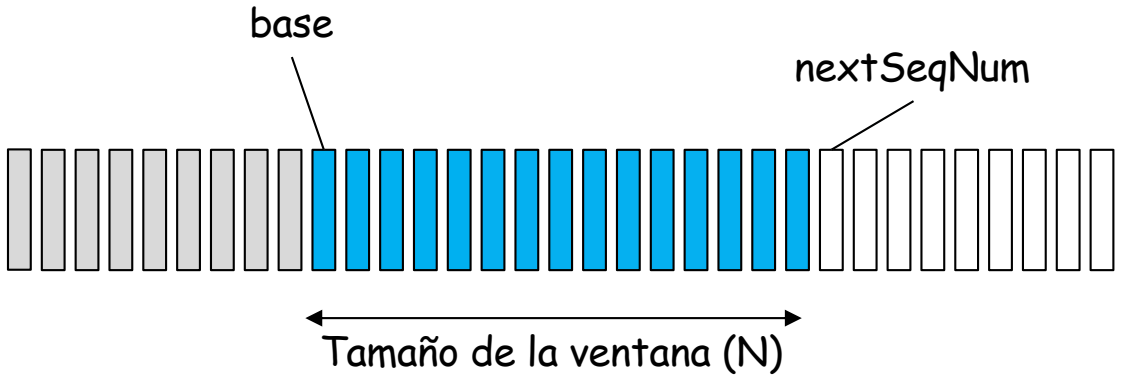
- Lo máximo que podemos enviar es el tamaño de la ventana, hasta que recibamos un ACK
- Al menos vamos a tardar un RTT en recibirlo
- Lo ideal sería estar enviando durante todo ese tiempo

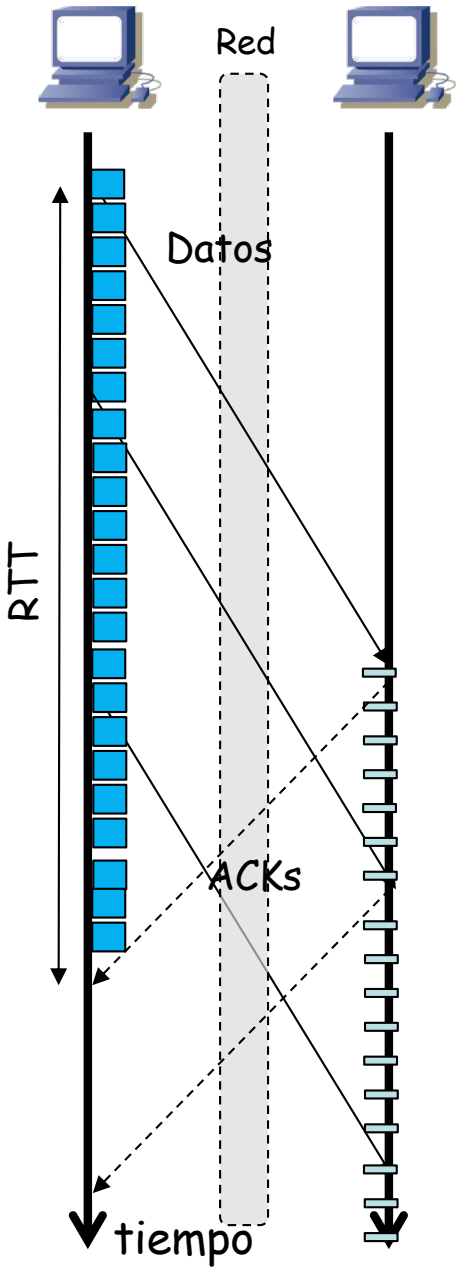




Throughput

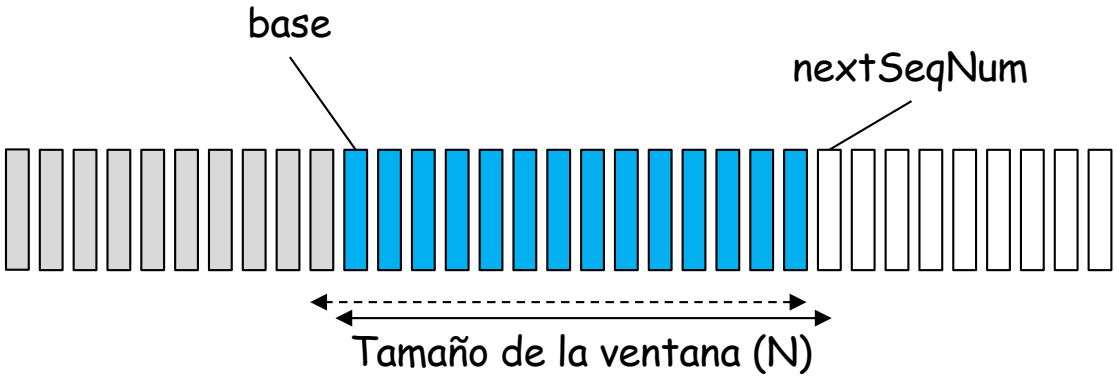
- Lo máximo que podemos enviar es el tamaño de la ventana, hasta que recibamos un ACK
- Al menos vamos a tardar un RTT en recibirlo
- Lo ideal sería estar enviando durante todo ese tiempo
- Para estar enviando todo el tiempo necesitamos una ventana tan grande que tardemos en enviarla todo el RTT
- En el momento en que llegue un ACK se desplazará la ventana y podemos seguir enviando

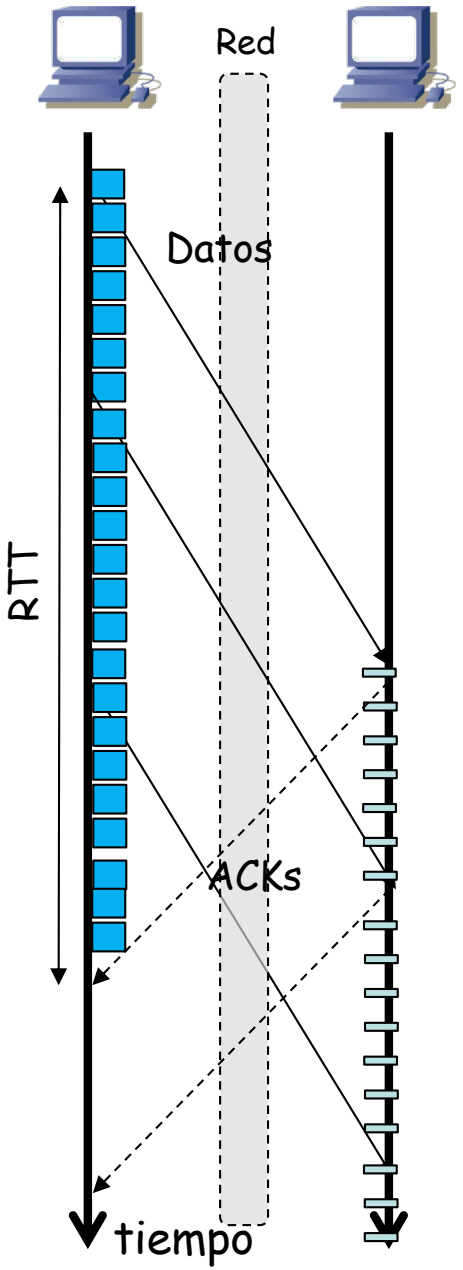




Throughput

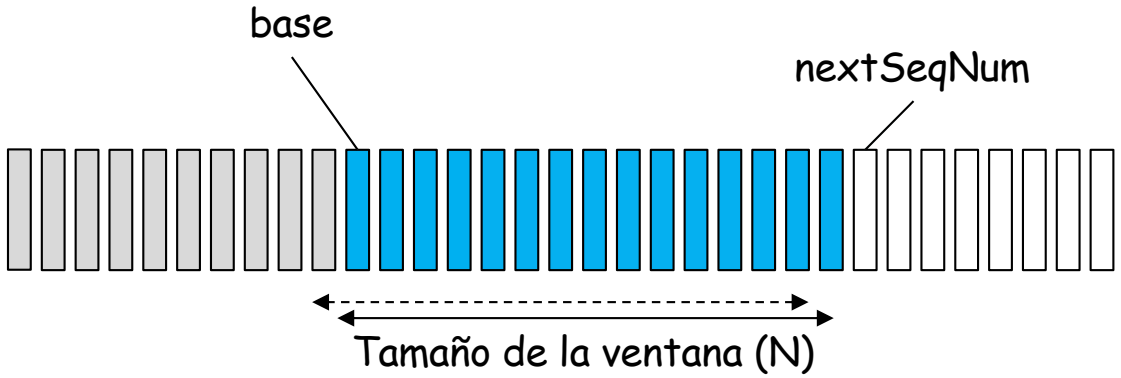
- Lo máximo que podemos enviar es el tamaño de la ventana, hasta que recibamos un ACK
- Al menos vamos a tardar un RTT en recibirlo
- Lo ideal sería estar enviando durante todo ese tiempo
- Para estar enviando todo el tiempo necesitamos una ventana tan grande que tardemos en enviarla todo el RTT
- En el momento en que llegue un ACK se desplazará la ventana y podemos seguir enviando
- Por ejemplo, ACK del primer paquete
- *nextSeqNum* pasa a estar dentro de la ventana y podemos enviar

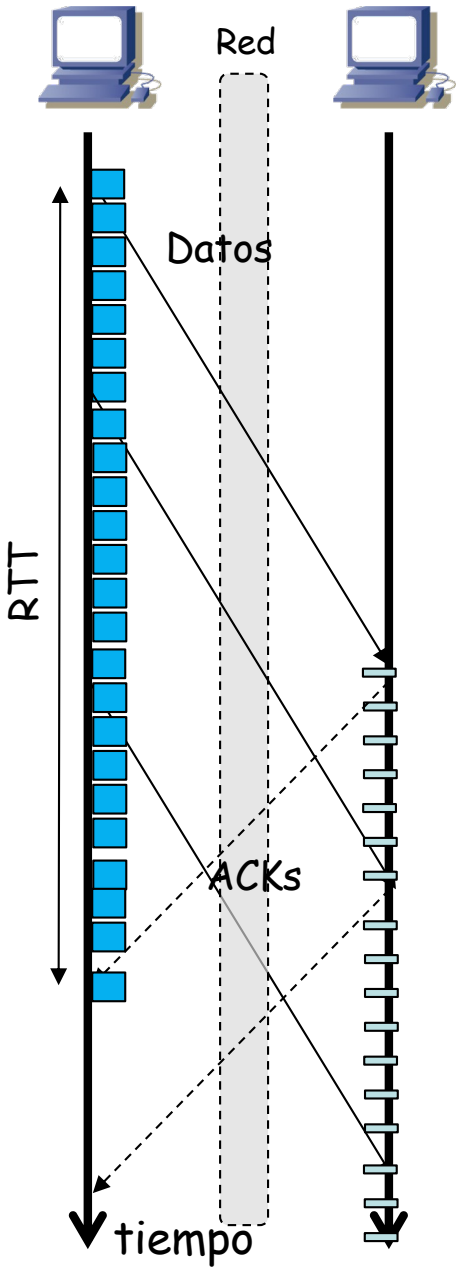




Throughput

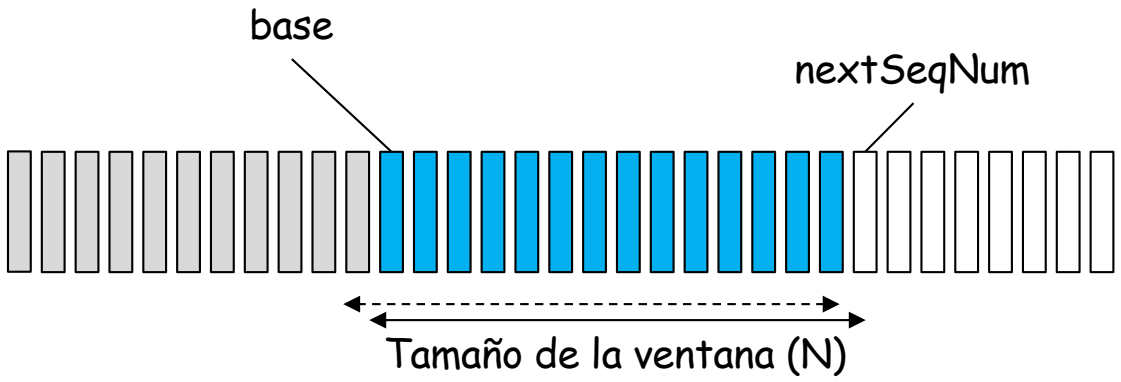
- Lo máximo que podemos enviar es el tamaño de la ventana, hasta que recibamos un ACK
- Al menos vamos a tardar un RTT en recibirlo
- Lo ideal sería estar enviando durante todo ese tiempo
- Para estar enviando todo el tiempo necesitamos una ventana tan grande que tardemos en enviarla todo el RTT
- En el momento en que llegue un ACK se desplazará la ventana y podemos seguir enviando
- Por ejemplo, ACK del primer paquete
- *nextSeqNum* pasa a estar dentro de la ventana y podemos enviar

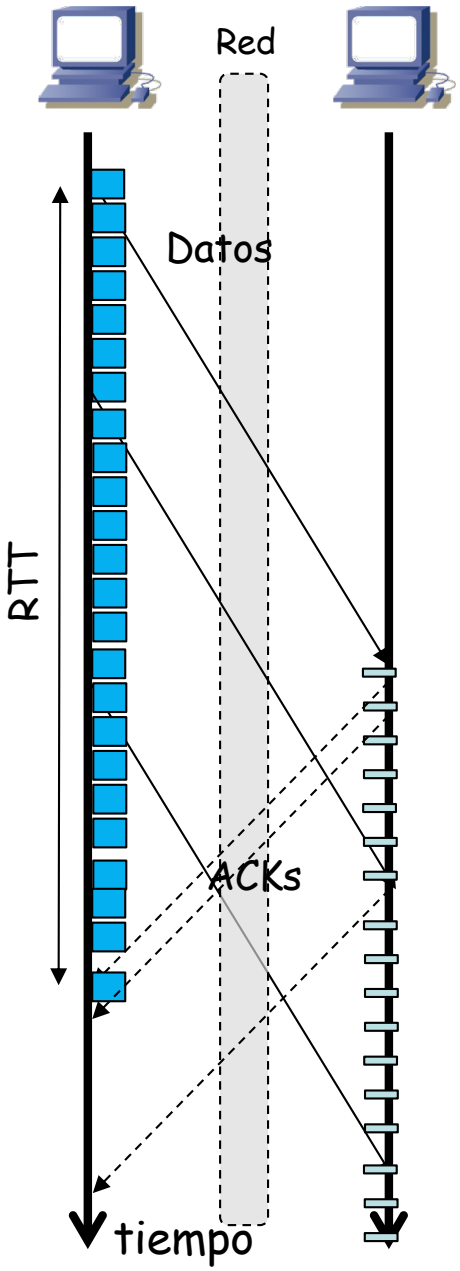




Throughput

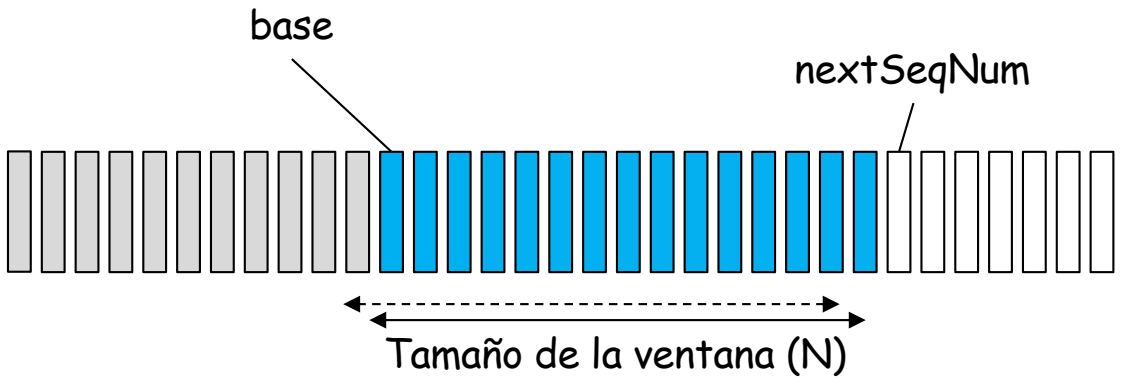
- Si se envía un ACK por cada paquete que llega nos llegarán aproximadamente al ritmo que enviamos los paquetes
- Y por cada uno se desplazará la ventana
- Y enviaremos un nuevo paquete
- Los ACK pasan a controlar el envío de paquetes
- Por cada uno que llega enviamos un paquete de datos nuevo

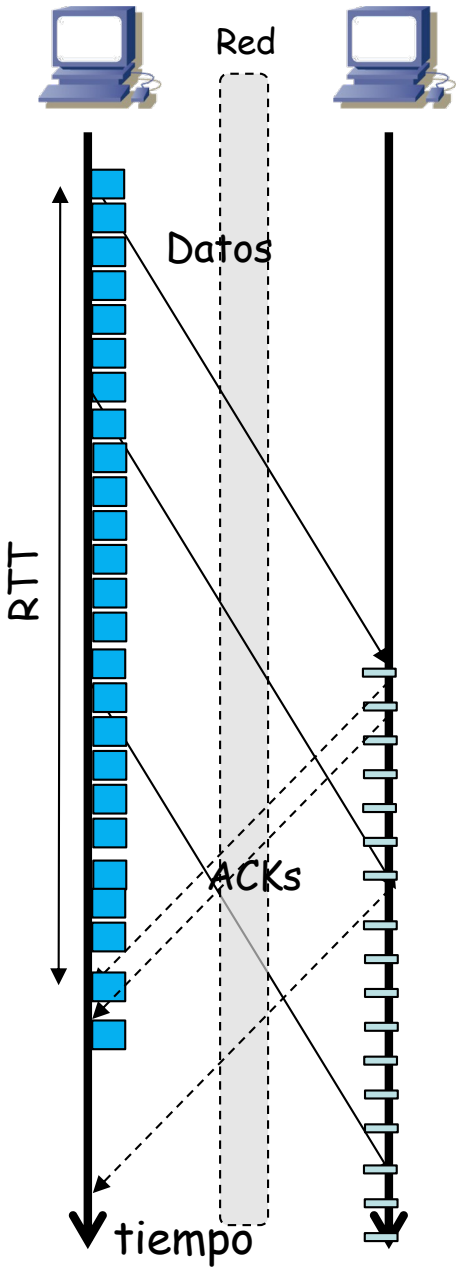




Throughput

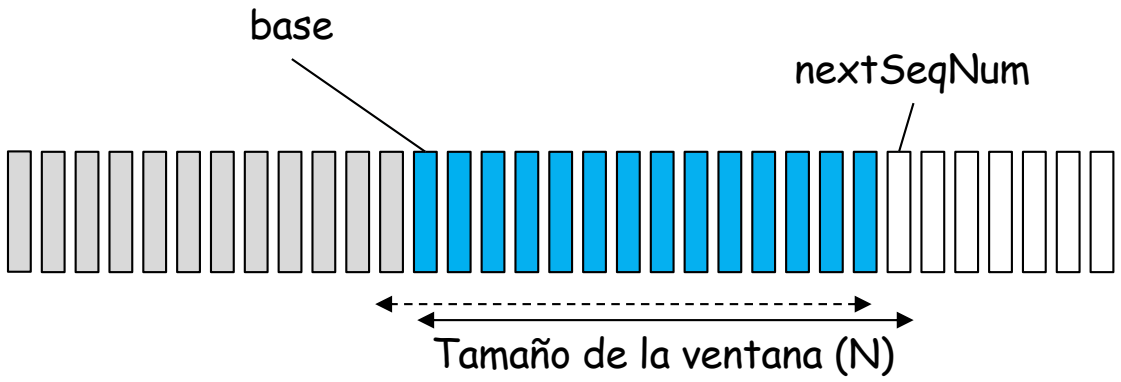
- Si se envía un ACK por cada paquete que llega nos llegarán aproximadamente al ritmo que enviamos los paquetes
- Y por cada uno se desplazará la ventana
- Y enviaremos un nuevo paquete
- Los ACK pasan a controlar el envío de paquetes
- Por cada uno que llega enviamos un paquete de datos nuevo

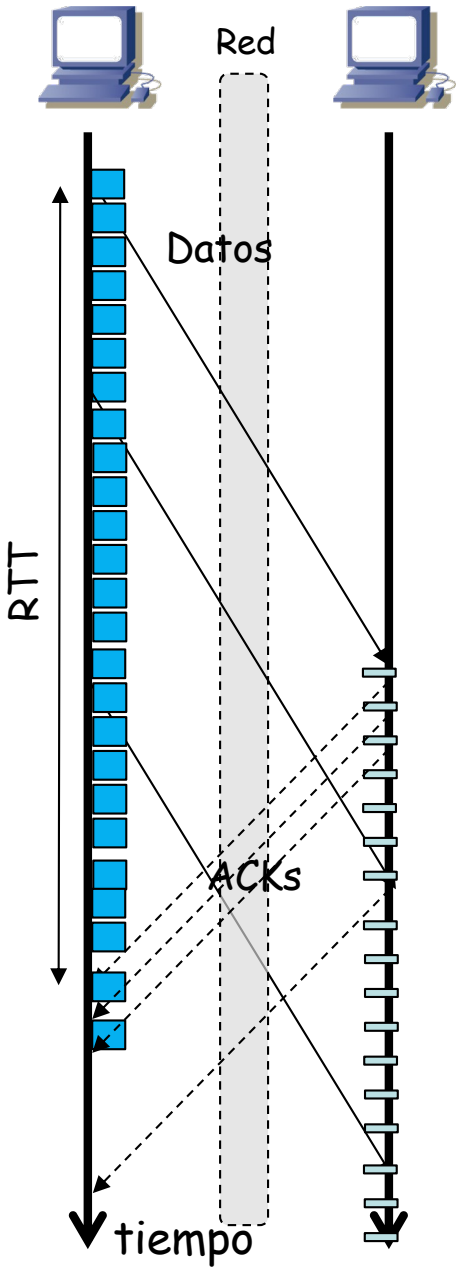




Throughput

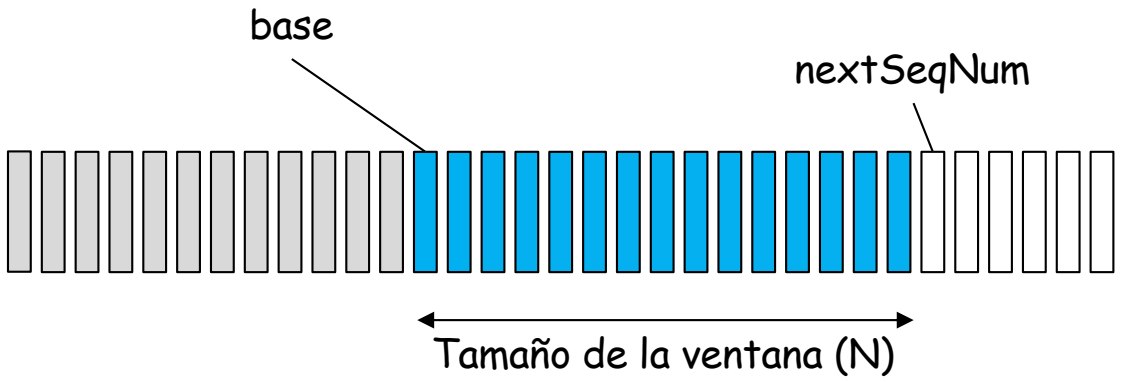
- Si se envía un ACK por cada paquete que llega nos llegarán aproximadamente al ritmo que enviamos los paquetes
- Y por cada uno se desplazará la ventana
- Y enviaremos un nuevo paquete
- Los ACK pasan a controlar el envío de paquetes
- Por cada uno que llega enviamos un paquete de datos nuevo

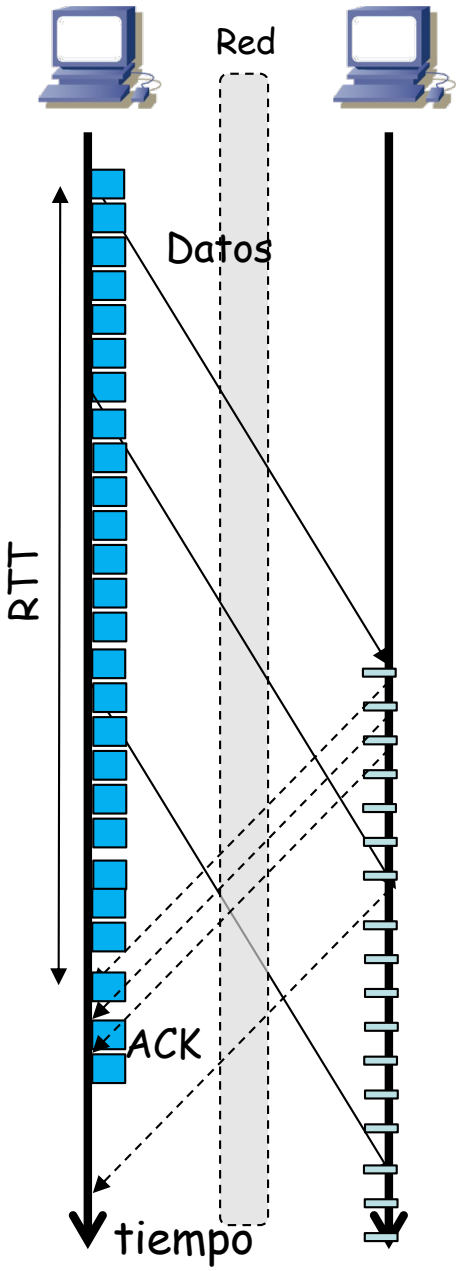




Throughput

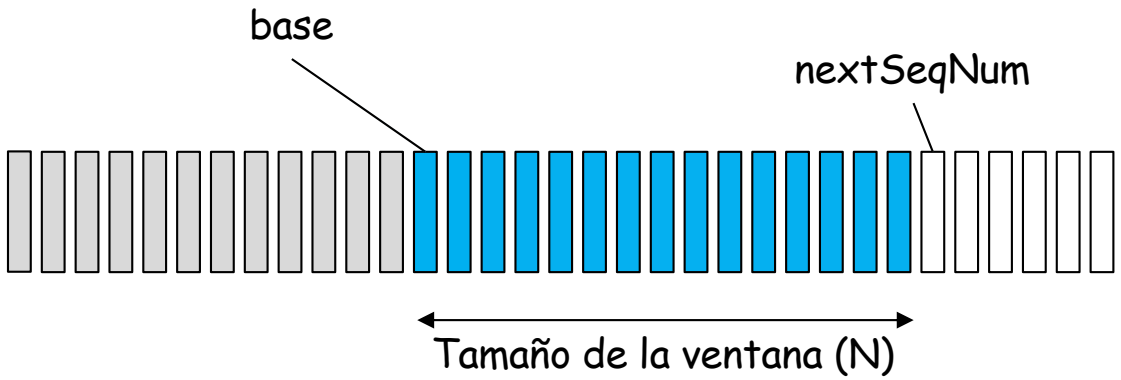
- Si se envía un ACK por cada paquete que llega nos llegarán aproximadamente al ritmo que enviamos los paquetes
- Y por cada uno se desplazará la ventana
- Y enviaremos un nuevo paquete
- Los ACK pasan a controlar el envío de paquetes
- Por cada uno que llega enviamos un paquete de datos nuevo

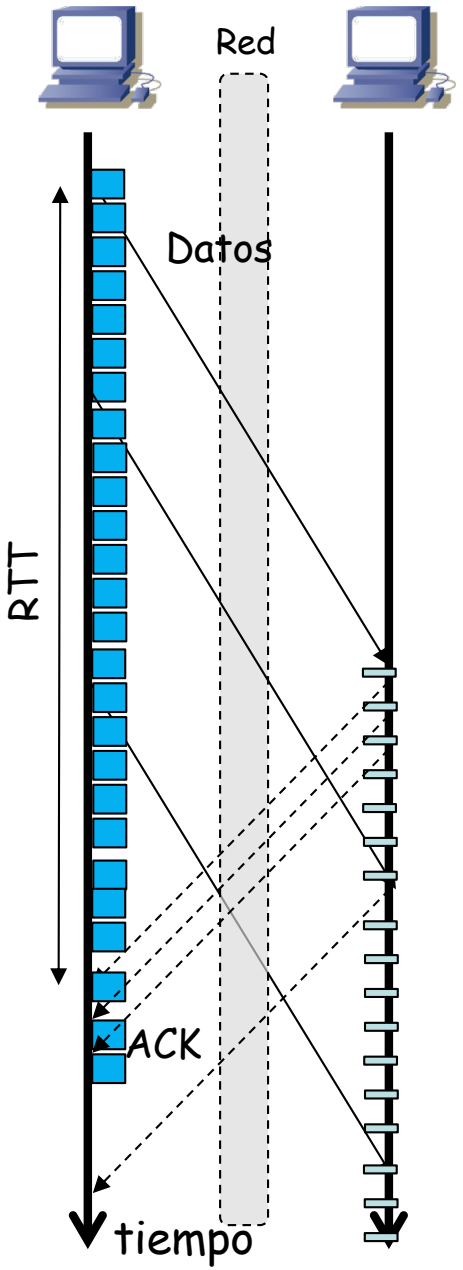




Producto RTT BW

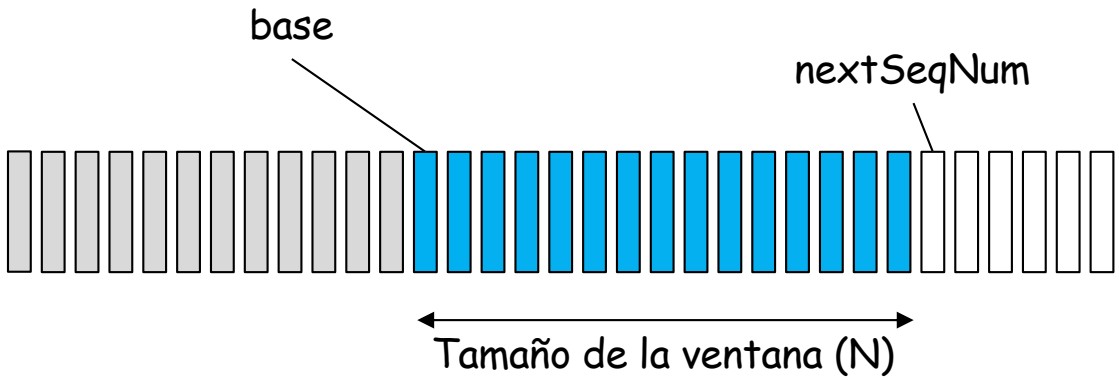
- Enviamos toda la ventana cada RTT
- Si la tasa de transmisión es C pkt/s necesitamos una ventana de al menos CxRTT pkts
- Es lo que llamamos el “producto retardo-ancho de banda”
- Podríamos tener una ventana más grande pero no mejoraría el resultado
- Inversamente, si tenemos una ventana de tamaño W y un RTT tenemos un throughput máximo de W/RTT
- Podemos tener más velocidad en el enlace, pero nos limita el protocolo de ventana deslizante
- Similar al fenómeno con stop&wait

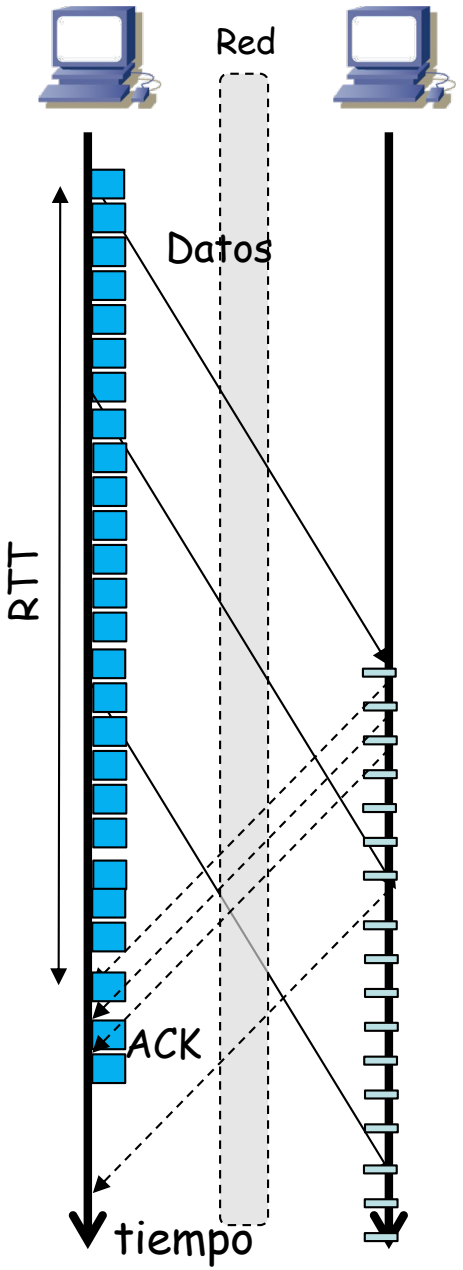




Producto RTT BW

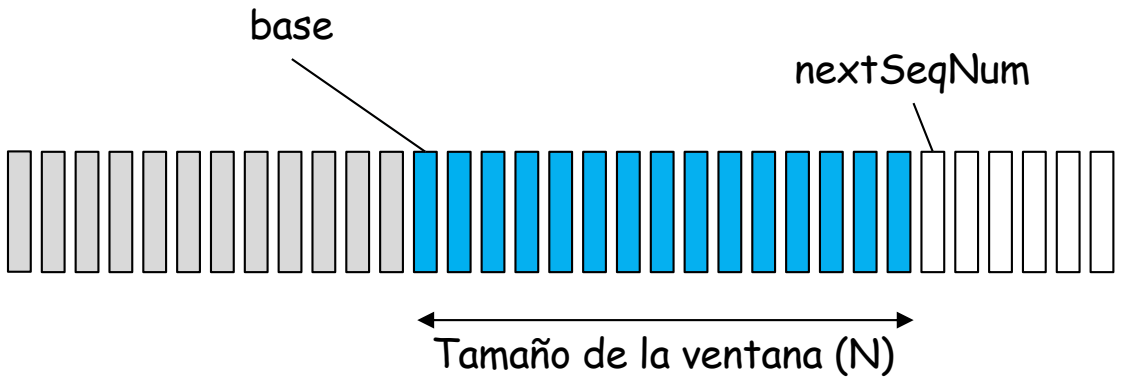
- Ejemplo:
  - Tasa de transmisión de 100Mb/s, RTT de 10ms
  - 100Mb/s x 10ms = 125 Kbytes
  - Una ventana de 125Kbytes tardamos 10ms en enviarla a 100Mb/s
  - Justo cuando estemos terminando de enviar el último paquete de esos 125Kbytes llegará el primer ACK
  - Nos desplaza la ventana y podemos enviar uno nuevo
  - Y los siguientes ACK mantienen el ritmo

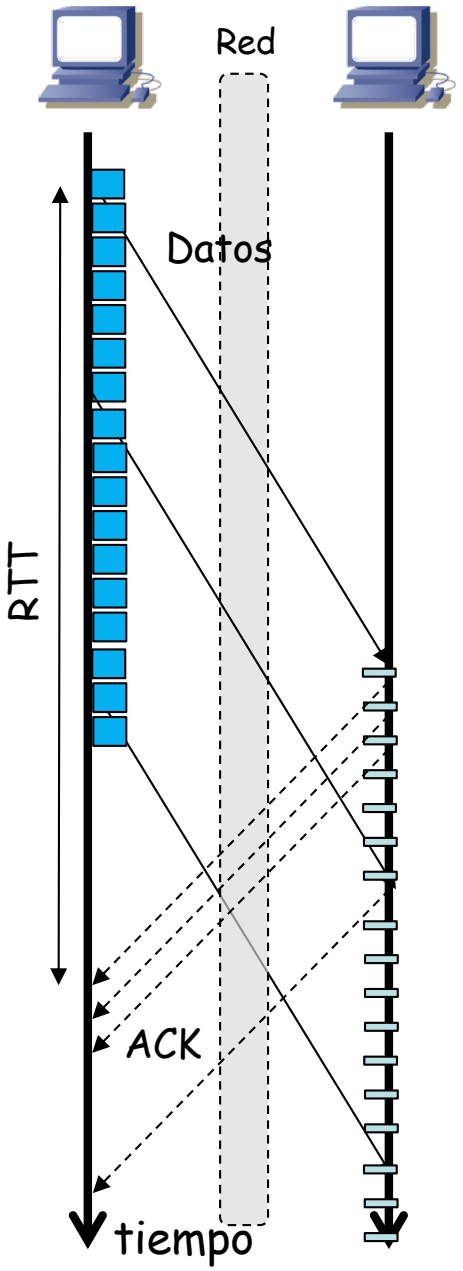




Producto RTT BW

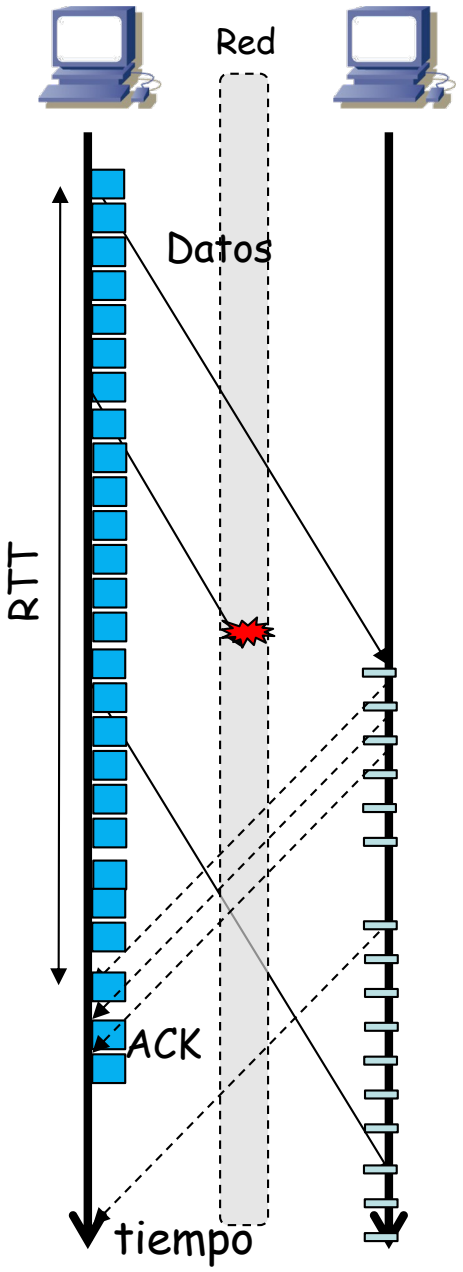
- Si la ventana es más pequeña que este producto no podremos usar el medio todo el tiempo
- El medio queda inactivo porque ya tenemos toda la ventana en vuelo
- Cada RTT podemos enviar toda la ventana, pero no estamos usando el medio todo el tiempo





Ventana y pérdidas

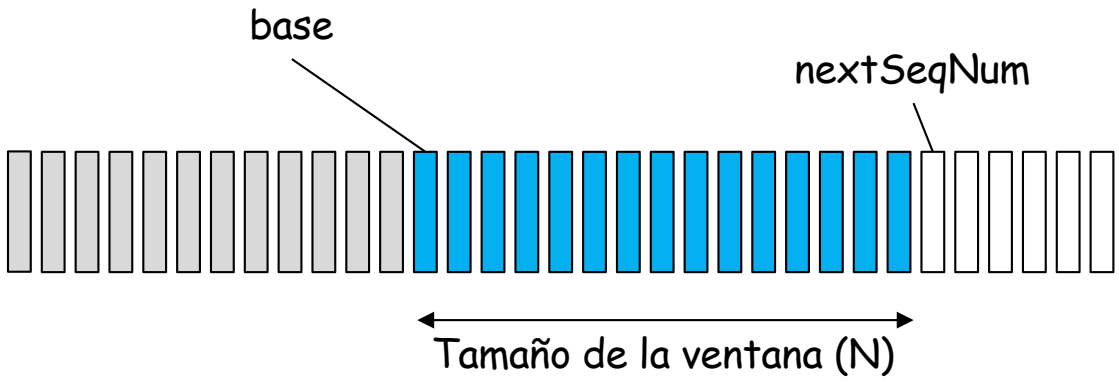
- Podrá haber pérdidas de paquetes
- El emisor no lo va a saber hasta que empiecen a llegar ACKs
- O a no llegar y que caduque el timer
- El emisor sigue enviando mientras le permita la ventana
- Con ACKs acumulados los nuevos ACKs siguen repitiendo el último número de secuencia consecutivo
- En Go-Back-N reenviamos toda la ventana desde el último confirmado
- Muchos los hemos enviado (ocupan tiempo y buffers en la red) para terminar reenviándolos

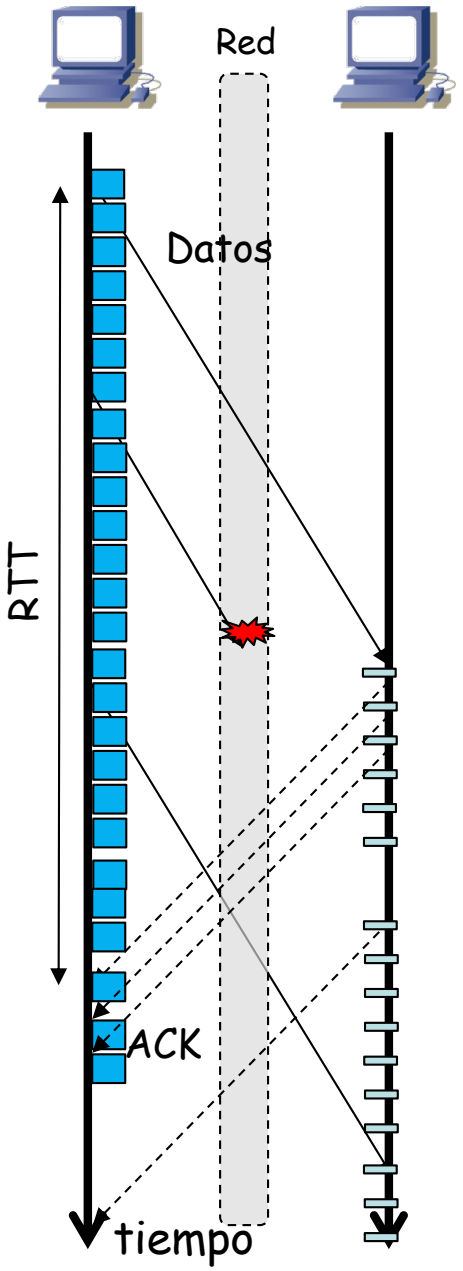


Selective Repeat

Selective Repeat

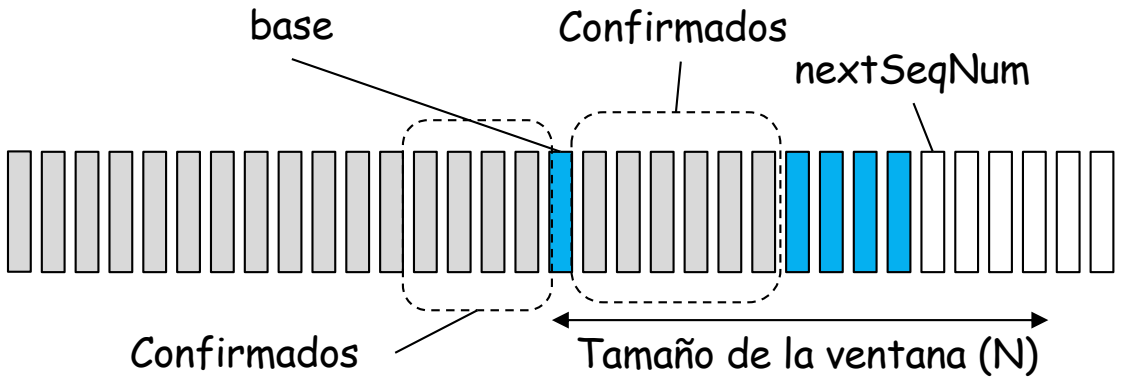
- En vez de confirmación acumulada, cada ACK confirma un conjunto concreto de datos
- Eso permite que los ACKs tras la pérdida confirmen datos, aunque haya un “hueco” en la secuencia
- Emisor necesita retransmitir solo los no confirmados
- Receptor guarda los paquetes desordenados hasta llenar el hueco (entrega en orden)

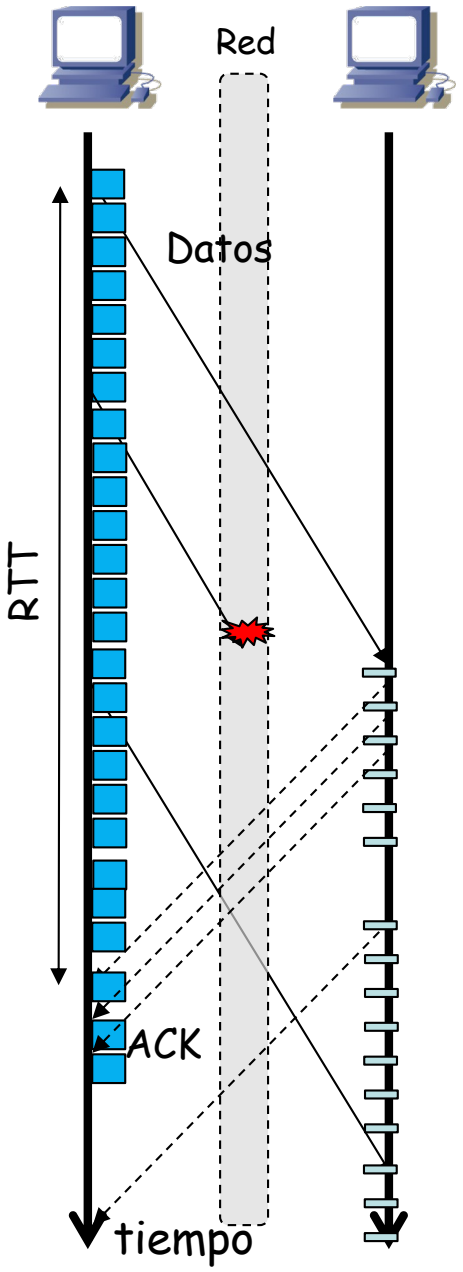




Selective Repeat

- En vez de confirmación acumulada, cada ACK confirma un conjunto concreto de datos
- Eso permite que los ACKs tras la pérdida confirmen datos, aunque haya un “hueco” en la secuencia
- Emisor necesita retransmitir solo los no confirmados
- Receptor guarda los paquetes desordenados hasta llenar el hueco (entrega en orden)
- Timer independiente para cada paquete enviado
- Más complejo para emisor y receptor





upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ARQUITECTURA DE REDES, SISTEMAS Y SERVICIOS
Área de Ingeniería Telemática

Transporte fiable y TCP/IP

TCP

- Ofrece entrega fiable y ordenada
- Numera los bytes: en cada segmento indica el número de secuencia del primer byte de datos
- Confirmación acumulada
- Confirmación selectiva posible pero opcional (SACK = Selective Acknowledgement)
- Receptor indica el tamaño máximo de la ventana
- Controla con ella cuánto puede enviar el emisor (control de flujo)
- Le añadiremos más mecanismos para ofrecer también control de congestión

UDP

- Nada