

Capítulo 3: Capa Transporte - II

ELO322: Redes de Computadores Agustín J. González

Este material está basado en:

- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet*. Jim Kurose, Keith Ross.

Capítulo 3: Continuación

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
 - ❑ 3.4.2 Protocolos con pipeline: Go-Back-N y Selective Repeat
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP

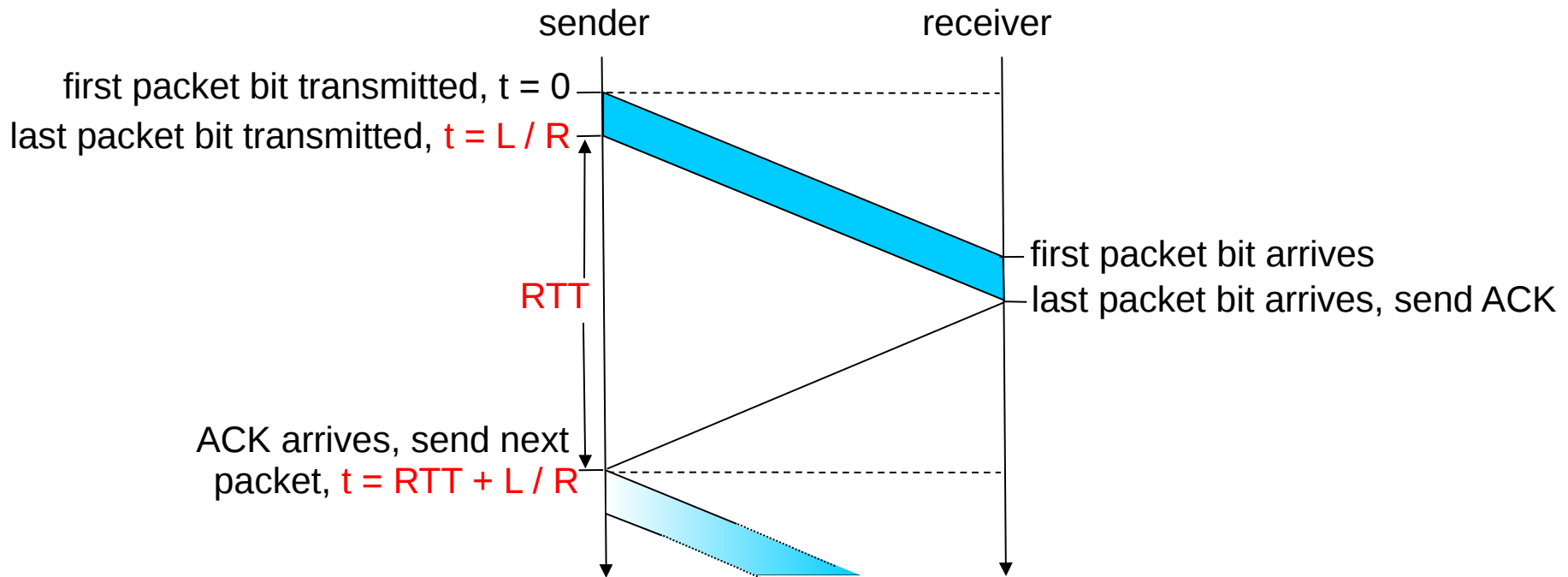
Utilización del Canal en Stop & Wait

- La utilización del canal o transmisión se define como: la fracción del tiempo en que está ocupado.

$$Utilización_{\text{Canal o transmisor}} = \frac{\text{tiempo en que Tx está transmitiendo}}{\text{tiempo total}}$$

- Evaluemos cuál es la utilización del canal de transmisión en protocolo Stop & Wait.

Utilización del canal de transmisión rdt3.0: protocolo stop & wait



$$Utilización_{transmisor} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027 = 0,027$$

L : Tamaño de paquete, RTT : Round-trip time, R : tasa de transmisión

Baja utilización, ¿recuerdan cómo se mejora esto?

Desempeño de rdt3.0

- ❑ rdt3.0 funciona, pero su desempeño es malo
- ❑ Ejemplo: R = enlace de 1 Gbps, 15 ms de retardo extremo a extremo, L = paquetes de 1KB, RTT = 30ms.

$$T_{\text{transmitir}} = \frac{L}{R} = \frac{8 \text{ Kb/paquete}}{10^9 \text{ b/s}} = 8 \mu\text{s}$$

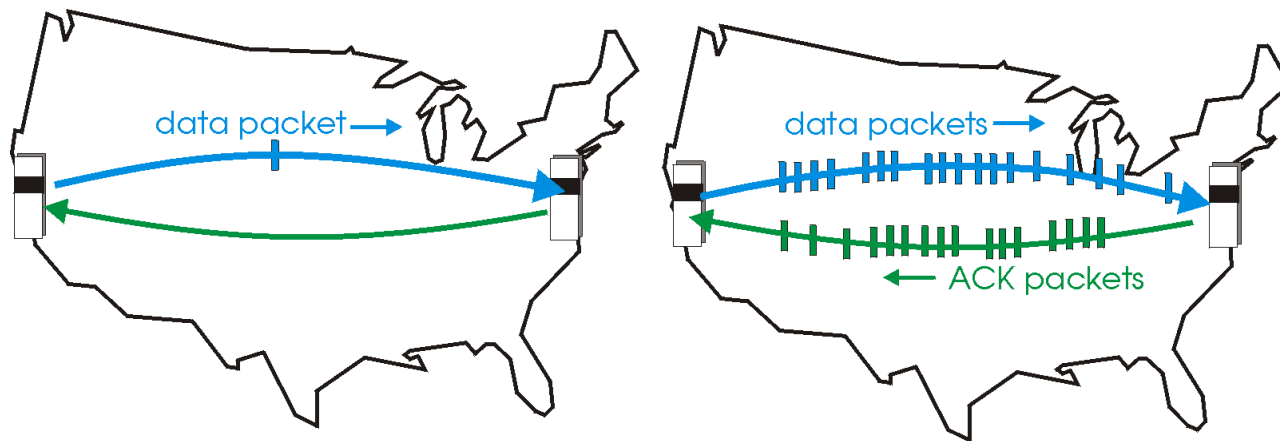
$$U_{\text{transmisor}} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027 = 0.027 \%$$

- $U_{\text{transmisor}}$: **utilización del transmisor o canal** = fracción de tiempo que el transmisor/canal está ocupado transmitiendo
- 1 paquete de 1KB cada ~30 ms -> 33kB/s tasa de transmisión en enlace de 1 Gbps
- Protocolo de red limita el uso de los recursos físicos!

Protocolos con Pipeline

Con Pipeline: Transmisor permite múltiples paquetes en tránsito con acuse de recibo pendiente

- El rango de los números de secuencia debe ser aumentado
- Se requiere buffers en el Tx y/o Rx

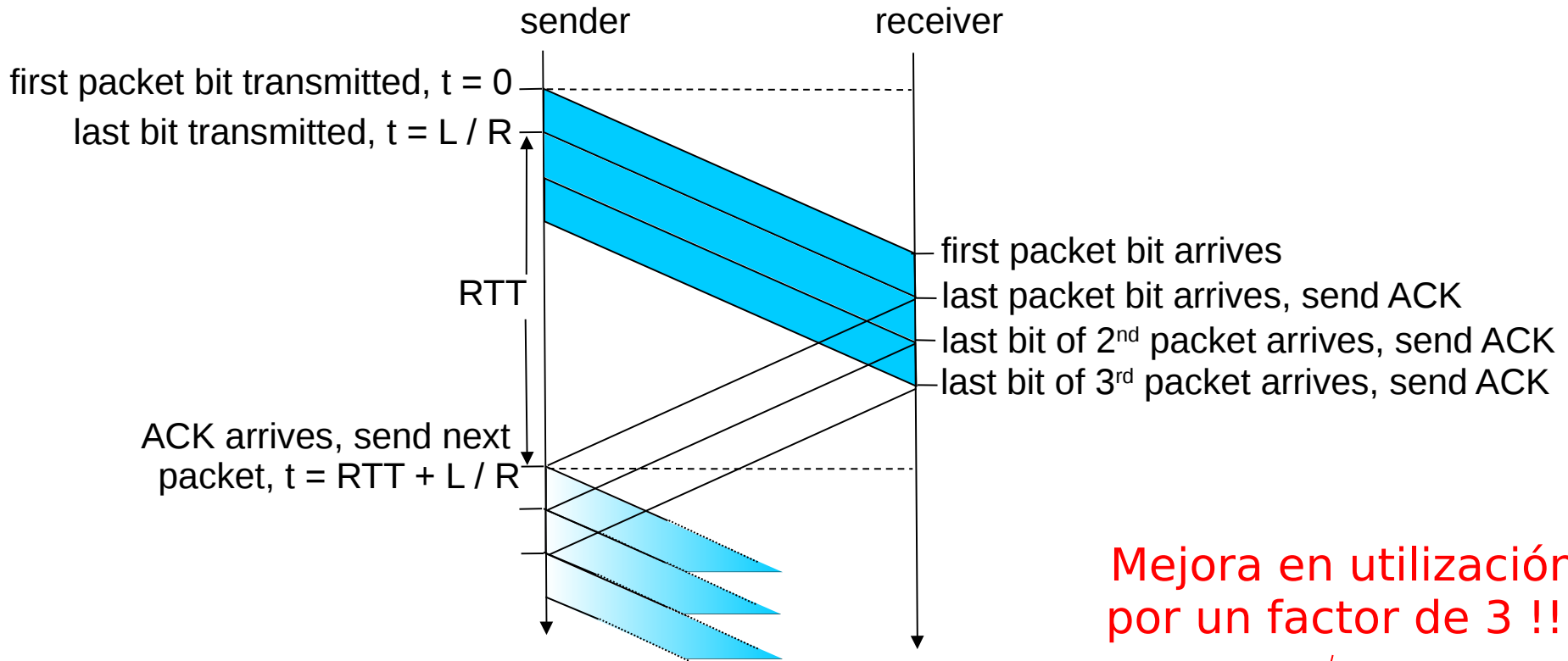


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Hay dos formas genéricas de protocolos con pipeline: *go-Back-N* y *selective repeat* (repetición selectiva)

Pipelining: utilización mejorada



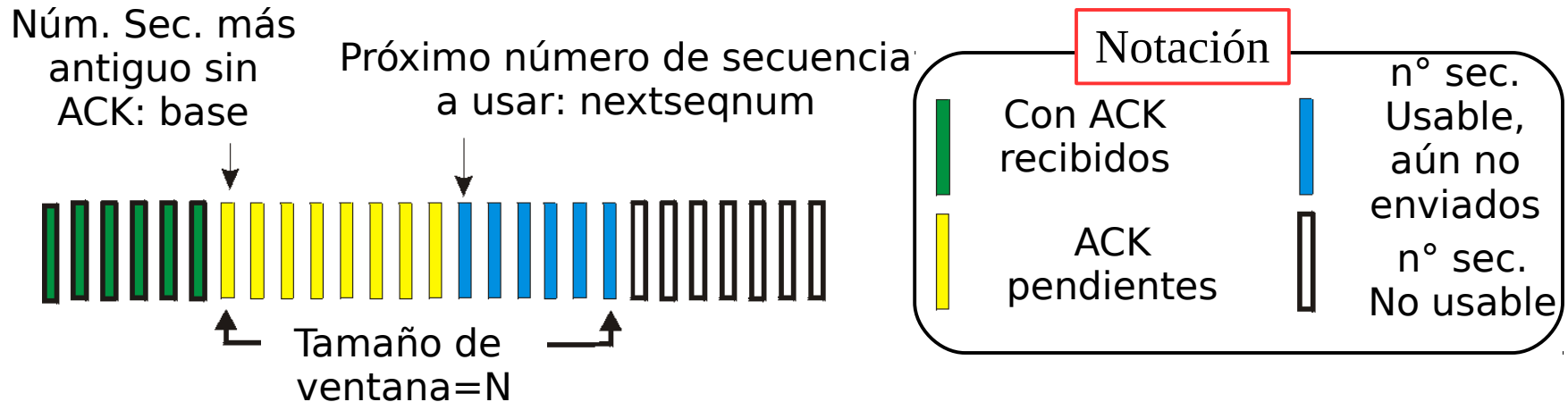
Mejora en utilización por un factor de 3 !!!

$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008 = 0.08\%$$

Go-Back-N: estrategia

Transmisor:

- ❑ # de secuencia de k-bits en el encabezado del paquete
- ❑ “ventana” de hasta N paquetes **consecutivos** con acuse de recibo pendiente



- ❑ Cuando llega un ACK(n): da acuse de recibo a todos los paquetes previos, incluyendo aquel con # de secuencia n; corresponde a un “**acuse de recibo acumulado**”
 - Se podría recibir ACKs duplicados (ver receptor)
- ❑ Usa **un timer** para manejar la espera de ack de paquete en tránsito
- ❑ **timeout(n): retransmitir paquete n y todos los paquetes con números de secuencia siguientes en la ventana**

GBN: MEF extendida del Transmisor

rdt_send(data)

```

if (nextseqnum < base+N) {
    sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum)
        start_timer
    nextseqnum++
}
else
    refuse_data(data)
    
```

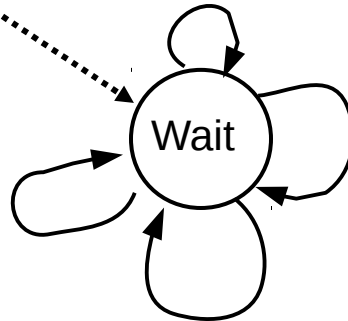
Condición inicial

Λ
base=1
nextseqnum=1

Es una MEF, con otra notación

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

Λ

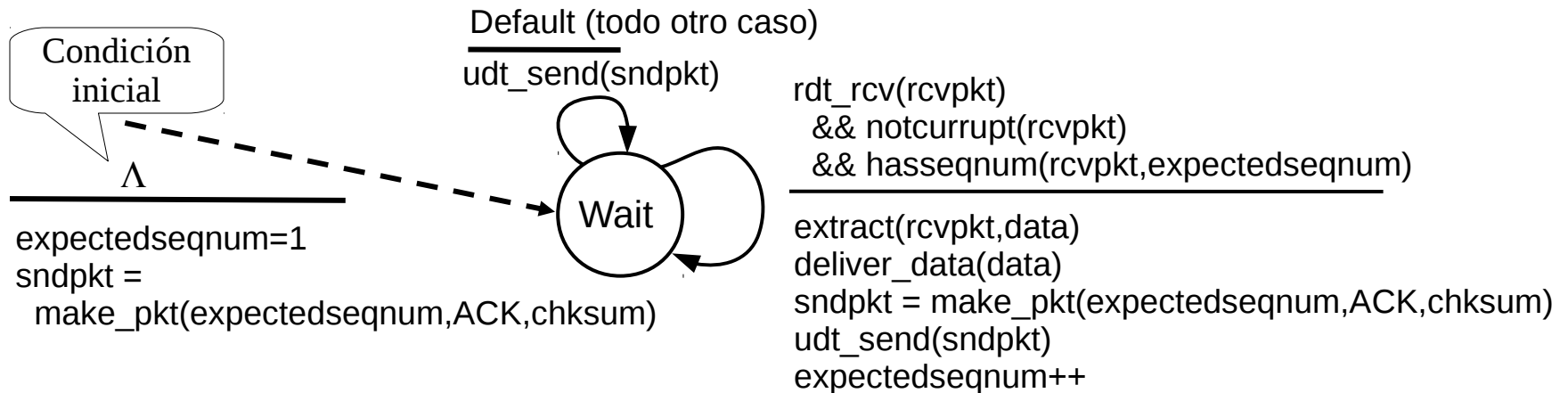


timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
...
udt_send(sndpkt[nextseqnum-1])

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
stop_timer
else
start_timer

Descripción del texto

GBN: MEF extendida del Receptor



- Usa sólo ACK: siempre envía ACK de paquete correctamente recibido con el # de secuencia mayor **en orden**
 - Puede generar ACKs duplicados. ¿Cuándo?
 - Sólo necesita recordar **expectedseqnum**
- Paquetes fuera de orden:
 - Descartarlos (no almacenar en buffer) => **no requiere buffer en receptor! Sólo para almacenar el paquete recibido.**
 - Re-envía ACK del paquete de mayor número de secuencia en orden

GBN en acción

sender window (N=4)

sender

receiver

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

send pkt0
 send pkt1
 send pkt2
 send pkt3
 (wait)

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

rcv ack0, send pkt4
 rcv ack1, send pkt5

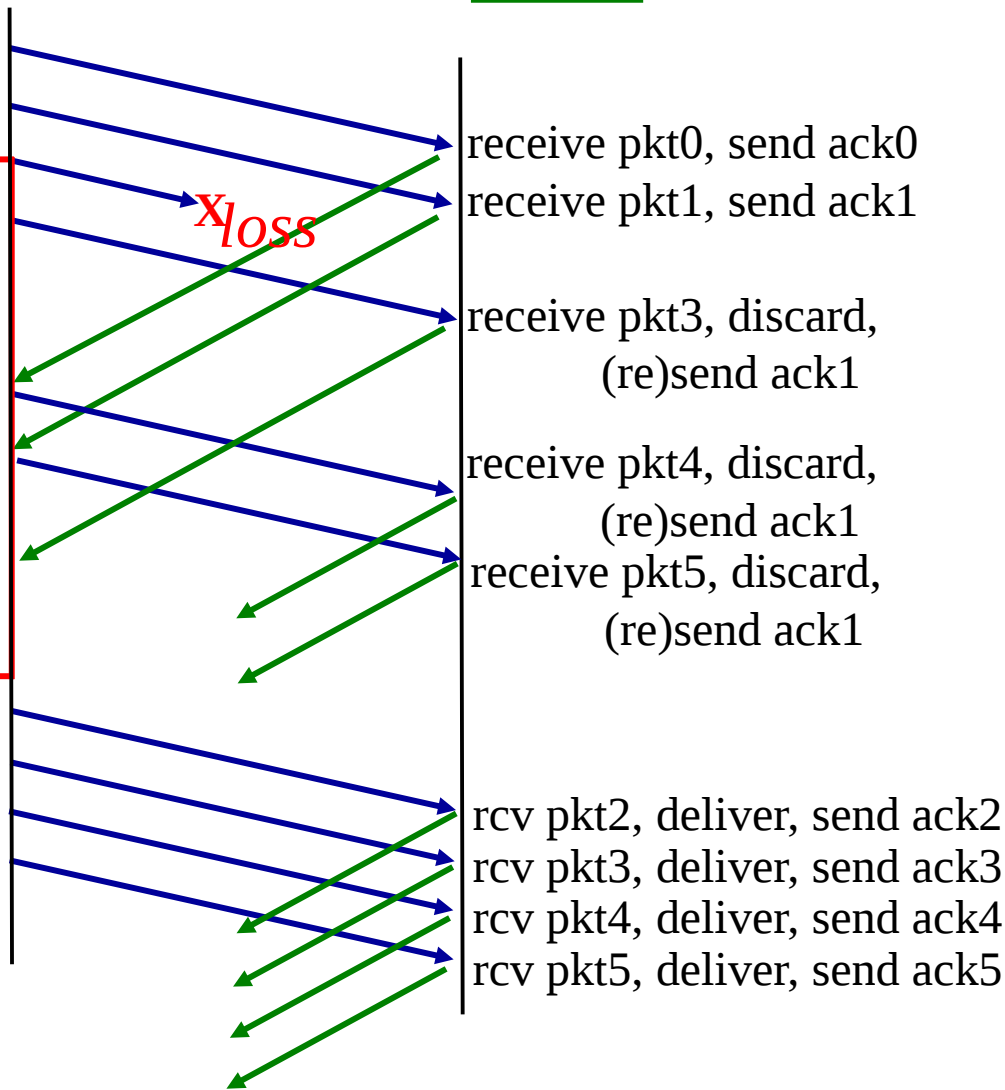
Ignore duplicate ACK



pkt 2 timeout

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

send pkt2
 send pkt3
 send pkt4
 send pkt5



¿Para qué re-enviar paquetes correctamente recibidos?

Go-Back-N: Análisis versión texto guía

❑ Idea Básica:

- Tx: Envía hasta completar ventana.
- Rx: Sólo acepta paquete correcto y en orden

❑ En caso de error o pérdida:

- Tx: Lo detecta por timeout y retransmite todo desde el perdido o dañado en adelante.

❑ Reflexionar:

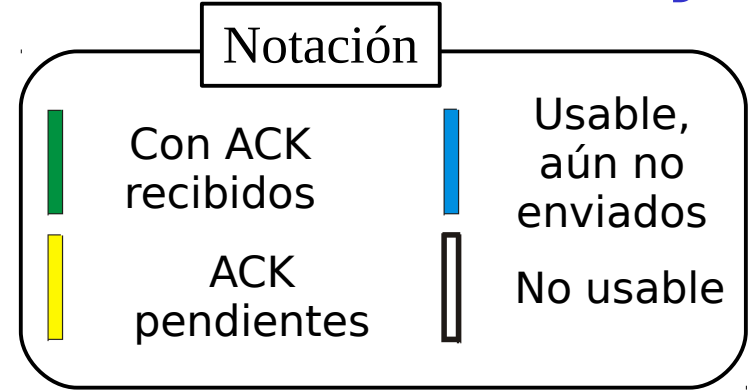
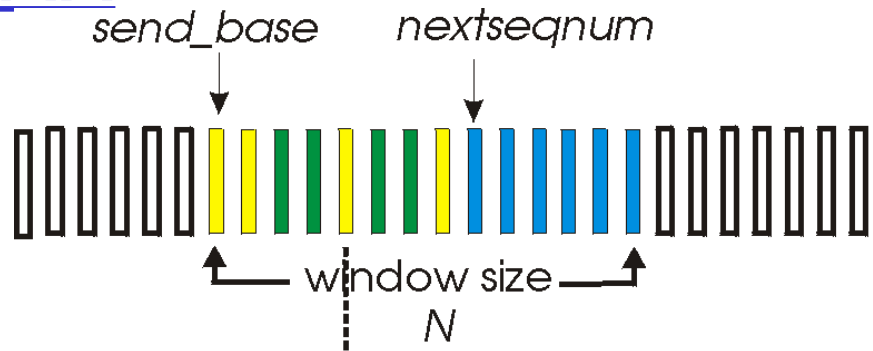
- Una pérdida sólo es detectada por el Tx luego de un timeout. Pero éste se reinicia con cada ACK que no sea el último. Convendría tener un timer por paquete enviado? Pero esto ocuparía más timers.
- Por qué reiniciar timer ante ACK distinto del último?
- ¿Por qué acá un ACK duplicado no es considerado como paquete perdido?

Selective Repeat (repetición selectiva)

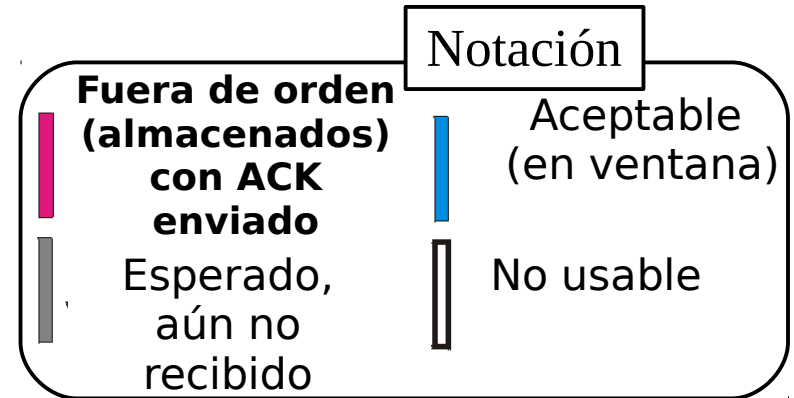
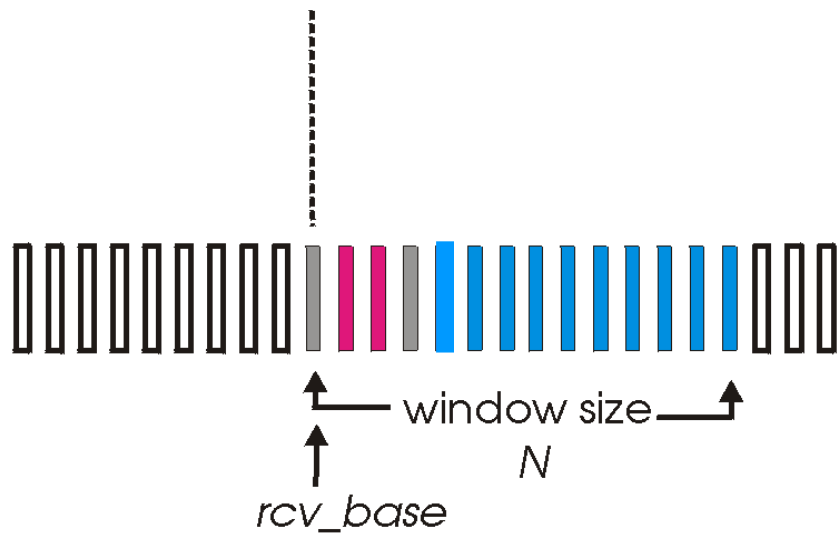
- ❑ Receptor envía acuse de recibo *individuales* de todos los paquetes recibidos
 - Almacena paquetes en buffer, según necesidad para su entrega en orden a la capa superior
- ❑ Transmisor sólo re-envía los paquetes sin ACK recibidos
 - Transmisor usa **un timer por cada paquete** sin ACK
- ❑ Ventana del Transmisor
 - Es la cantidad de números de secuencia consecutivos que puede enviar.
 - Nuevamente limita los #s de secuencia de paquetes enviados sin ACK
- ❑ Existe ventana en Receptor

Selective repeat: Ventanas de Tx y Rx

Rx



a) Vista de los número de secuencia del transmisor



b) Vista de los número de secuencia del receptor

Selective repeat (repetición selectiva)

Transmisor

Ante llegada datos desde arriba:

- Si el próximo # de sec. está en ventana, enviar paquete

Ante timeout(n):

- Re-enviar sólo paquete n, re-iniciar timer

Ante ACK(n) en

[sendbase, sendbase+N]:

- Marcar paquete n como recibido, parar su timer
- Si n es el paquete más antiguo sin ACK, avanzar la base de la ventana al próximo # de sec. sin ACK.

Receptor

Ante llegada paquete n en [rcvbase, rcvbase+N-1]

- Enviar ACK(n)
- Si está fuera de orden: almacenar en buffer
- En-orden: entregar a capa superior (también entregar paquetes en orden del buffer), avanzar ventana al paquete próximo aún no recibido

Ante paquete n en [rcvbase-N, rcvbase-1]

- Enviar ACK(n)

Otro caso:

- ignorarlo

Repetición Selectiva en Acción

sender window (N=4)

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

sender

send pkt0
 send pkt1
 send pkt2
 send pkt3
 (wait)

rcv ack0, send pkt4
 rcv ack1, send pkt5

record ack3 arrived



pkt 2 timeout
 send pkt2

record ack4 arrived
 record ack4 arrived

receiver

receive pkt0, send ack0
 receive pkt1, send ack1

receive pkt3, **buffer,**
send ack3

receive pkt4, **buffer,**
send ack4

receive pkt5, **buffer,**
send ack5

rcv pkt2; **deliver pkt2,**
pkt3, pkt4, pkt5; send ack2

6 7 8 9 10 11

0 1 2 3 4 5
 1 2 3 4 5
 2 3 4 5

2 3 4 5

2 3 4 5

2 3 4 5

Q: Qué pasa cuando llega ack2?

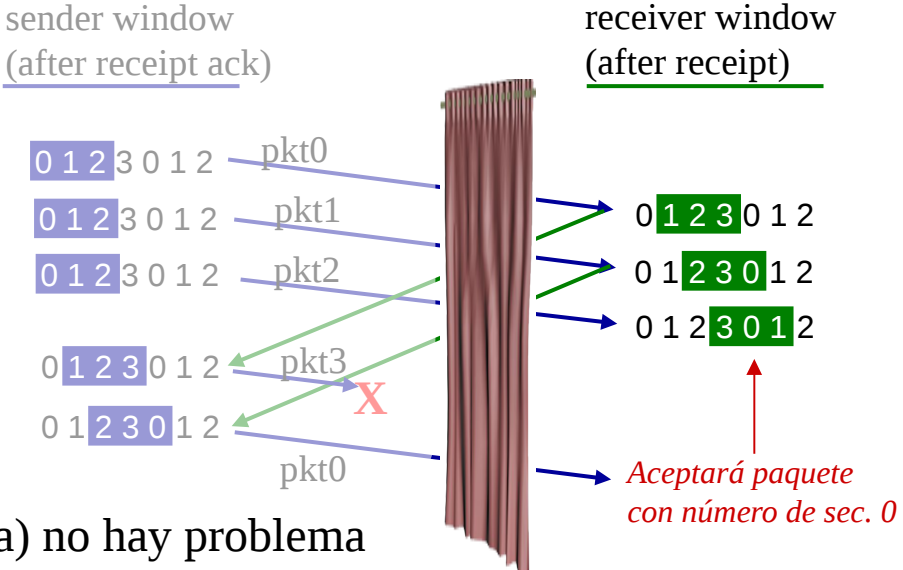
Dilema de la repetición Selectiva

Ejemplo:

- ❑ #s de sec.: 0, 1, 2, 3
- ❑ Tamaño de ventana=3

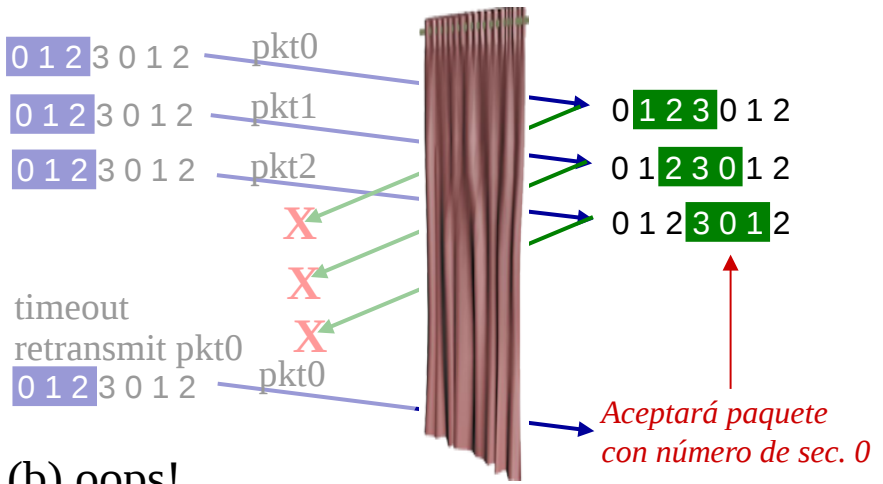
- ❑ Rx no ve diferencia en los dos escenarios!
- ❑ Pasa incorrectamente datos como nuevos en (b)

Q: ¿Qué relación debe existir entre el # de sec. y el tamaño de ventana?



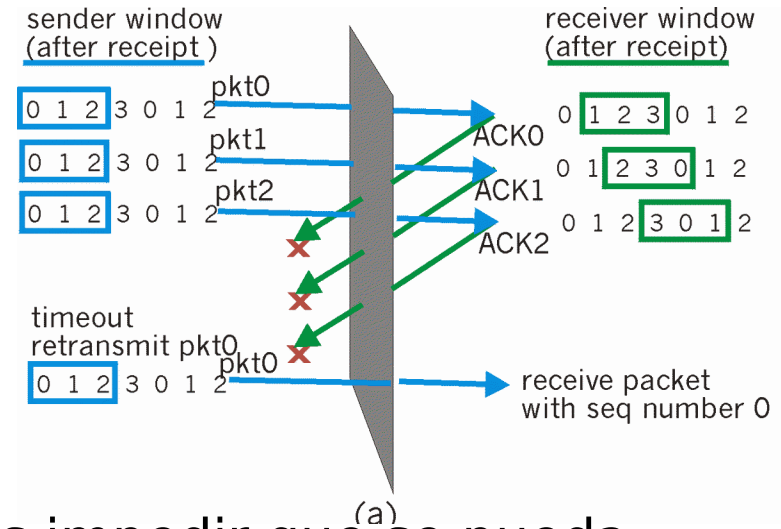
(a) no hay problema

El receptor no puede ver el lado Tx. Igual acción de Rx en ambos casos!
Algo está (muy) mal!



(b) oops!

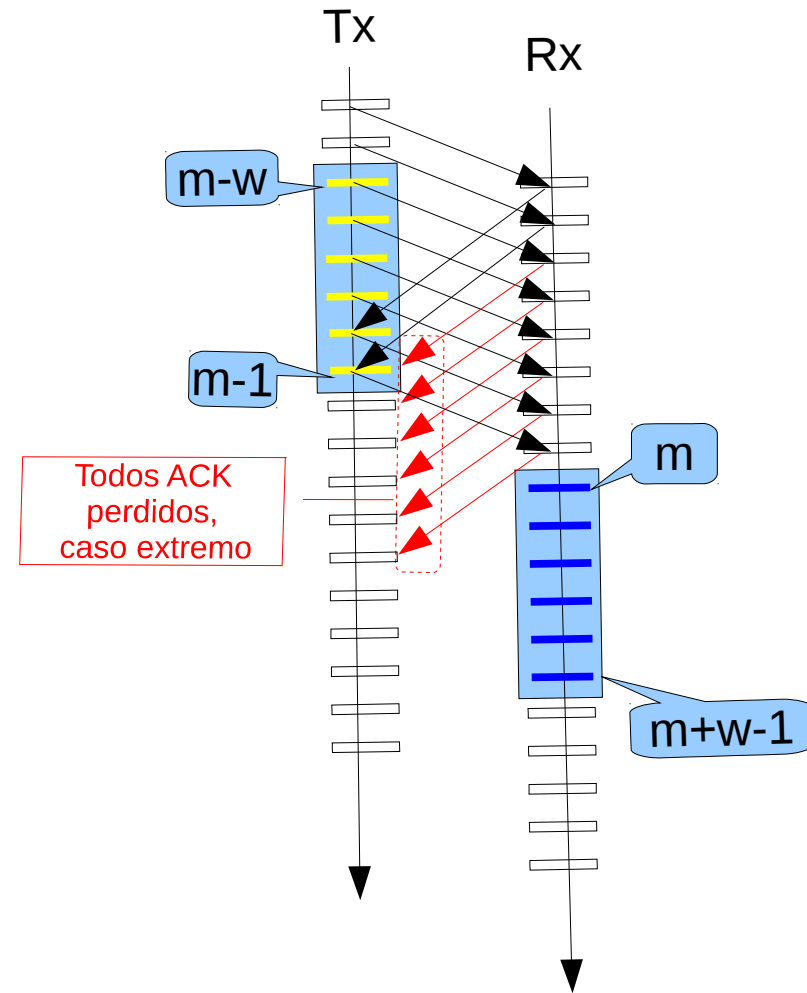
Q: ¿Qué relación debe existir entre el # de sec. y el tamaño de ventana?



- ❑ La clave para evitar este problema es impedir que se pueda producir el escenario de la figura adjunta.
- ❑ Supongamos que la ventana de recepción es $[m, m+w-1]$, por lo tanto Rx ha recibido y enviado ACK del paquete $m-1$ y los $w-1$ paquetes previos a éste.
- ❑ Si ninguno de estos ACK han sido recibidos por el Tx la ventana del transmisor será $[m-w, m-1]$.
- ❑ Así, el mayor número de secuencia de la ventana del Rx será $m+w-1$ y el límite inferior de la ventana del Tx será $m-w$.
- ❑ Para que Rx tome el paquete $m-w$ como duplicado, su número de secuencia debe caer fuera de su ventana.
- ❑ Luego debemos tener un **rango de números de secuencia k** tan grande como para acomodar $(m+w-1)-(m-w)+1=2w$ números de secuencia, luego $k \geq 2w$.
- ❑ **Q: ¿Qué relación debe existir en el caso Go-Back-N?**

Tamaño máximo de ventana en Selective Repeat en más detalle

- Rx espera paquetes en $[m, m+w-1]$
- Tx habiendo enviado toda su ventana, hace timeout al no recibir los acuses de recibos y re-envía paquete con secuencia $m-w$.
- Para que todo re-envío de ventana de Tx sea interpretado como duplicado debo tener números de secuencia distintos para ambas ventanas; luego, # de secuencia debes ser al menos $m+w-1-(m-w)+1 = 2w$.



$$\# \text{ Sec} \geq 2w$$

Capítulo 3: Continuación

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP