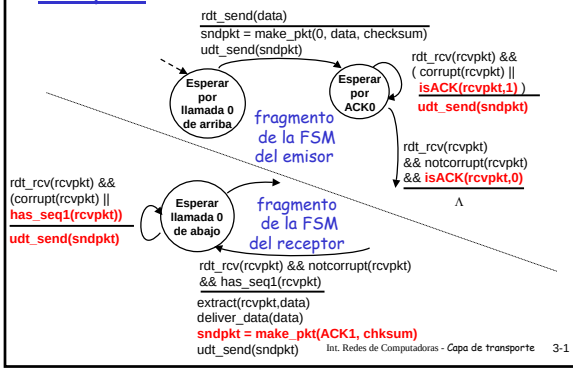
rdt2.2: fragmentos del emisor y receptor



rdt3.0: canales con errores *y pérdidas*

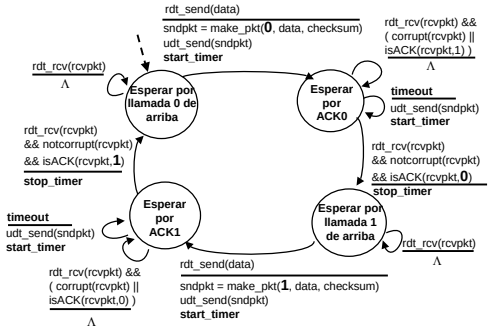
Nuevo supuesto: el canal subyacente también puede perder paquetes (datos o ACKs)

- suma de comprobación, nros. de secuencia, ACKs y retransmisiones nos ayudan, pero no lo suficiente

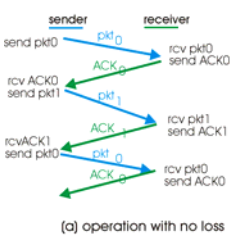
Enfoque: el transmisor espera por el ACK un tiempo "razonable"

- retransmite si no recibe el ACK en dicho tiempo
- si el paquete (o ACK) sólo está retrasado (no perdido):
  - la retransmisión será un duplicado, pero el nro. de secuencia maneja ésto
  - El receptor debe especificar el nro. de secuencia del paquete que está siendo ACKed
- requiere *countdown timer*

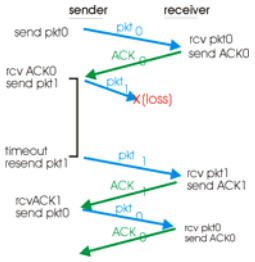
rdt3.0: emisor



rdt3.0 en acción

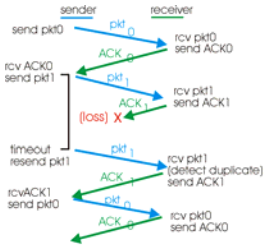


(a) operation with no loss

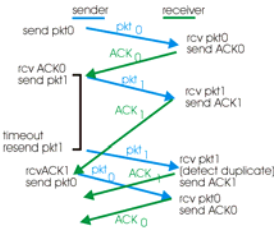


(b) lost packet

rdt3.0 en acción



(c) lost ACK



(d) premature timeout

Performance de rdt3.0

- rdt3.0 funciona, pero su *performance* no es buena
- Ejemplo: enlace de **1 Gbps**, retardo de propagación **15 ms**, paquetes de **1000 bytes** (8000 bits):

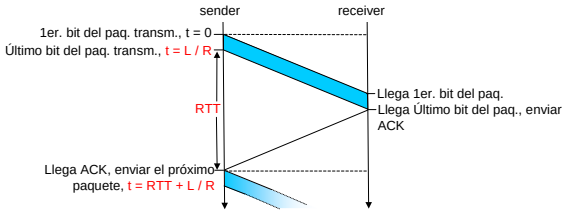
$$t_{\substack{transmitir \\ un\ paquete}} = \frac{L}{R} = \frac{8000\ \text{bits}}{10^9\ \text{bps}} = 8\ \mu\text{s}$$

  - U_{emisor}: utilización del canal – tiempo del *sender* ocupado enviando datos

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

  - Paquetes de **1KB** cada **30 msec** -> **33kB/sec** en un enlace de **1 Gbps**
  - Los protocolos de red limitan el uso de los recursos físicos

rdt3.0: operación *stop-and-wait*

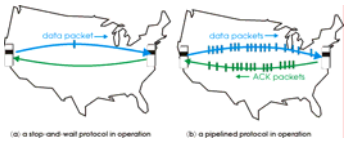


$$U_{emisor} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Pipelined protocols

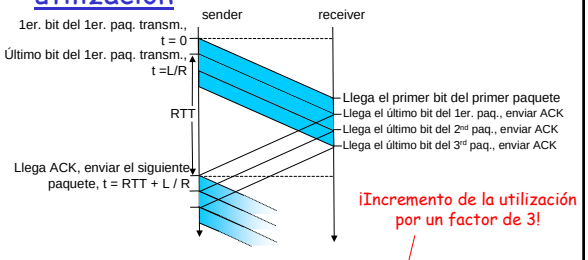
Pipelining: el emisor permite el envío de múltiples paquetes a ser reconocidos

- el rango de los números de secuencia se debe incrementar
- *buffering* en el *sender* y/o en el *receiver*



- Dos formas genéricas de *pipelined protocols*: *Go-Back-N (Retroceder N), Selective Repeat (Repetición Selectiva)*

Pipelining: incremento de la utilización



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

Pipelining Protocols

Go-back-N: titulares

- El transmisor puede tener hasta N paquetes no-ACKed en el *pipeline*
- El receptor sólo envía ACKs acumulativos
  - No envía el ACK de un paquete si hay un hueco
- El transmisor tiene un *timer* para el paquete más viejo no-ACKed
  - Si el *timer* vence, retransmite todos los paquetes no-ACKed

Selective Repeat: titulares

- El transmisor puede tener hasta N paquetes no-ACKed en el *pipeline*
- El receptor envía ACKs para paquetes individuales
- El transmisor tiene un *timer* para cada paquete no no-ACKed
  - Cuando el *timer* vence, retransmite sólo el paquete no-ACKed

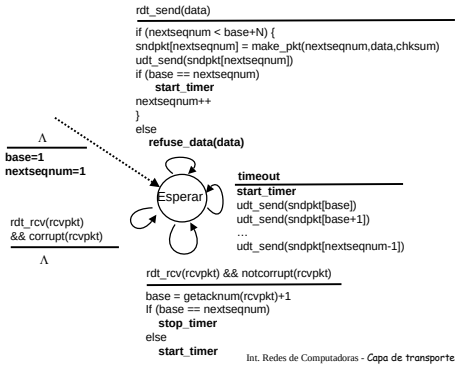
Go-Back-N

Emisor:

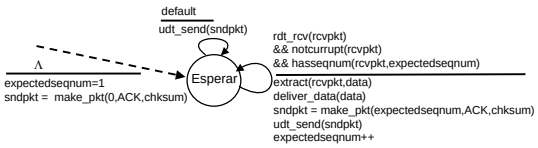
- Número de secuencia de **k bits** en el encabezado del paquete
- "window" permitida de hasta **N paquetes** consecutivos no-ACKed
- Si la ventana no está llena, se puede seguir transmitiendo

- ACK(n): ACKs de todos los paquetes hasta él, incluyendo el nro. de sec. n -- "**ACK acumulativo**"
  - podría recibir ACKs duplicados (ver receptor)
- *timer* único: para el paquete más viejo aún no-ACKed
- *timeout(n)*: retransmite el paquete **n** y todos aquellos con nro. de sec. mayor y dentro de la ventana

GBN: FSM extendida del ***sender***

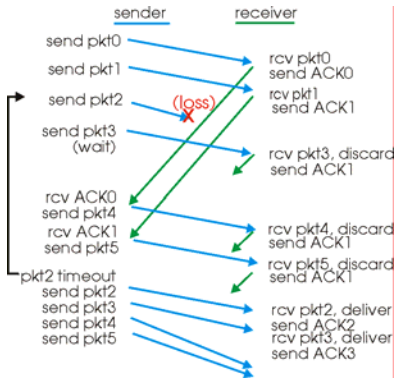


GBN: FSM extendida del *receiver*



- ACK-solamente: siempre enviar ACK para el paquete correctamente recibido y en orden con el mayor nro. de sec.
  - Se podrían generar ACKs duplicados
  - Solamente necesitamos recordar **expectedseqnum**
- Paquetes fuera de orden:
  - Descartar (no almacenar en *buffer*) -> sin buffer en el receptor
  - Re-ACK paquetes con el mayor nro. de sec. en orden

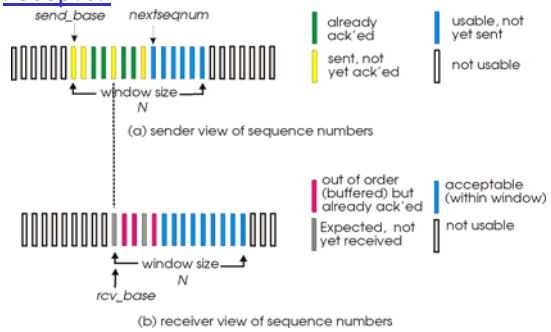
GBN en acción



Selective Repeat

- el receptor envía ACKs *individualmente* para todos los paquetes correctamente recibidos
  - *buffers* para los paquetes, para una eventual entrega en orden a la capa superior
- el emisor re-envía solamente los paquetes para los que no ha recibido su ACK
  - *timer* en el emisor para cada paquete no-ACKed
- ventana del emisor
  - N números de secuencia consecutivos
  - Nuevamente, los límites de los números de secuencia son los paquetes enviados no-ACKed

Selective repeat: ventanas en emisor y receptor



Selective repeat

emisor

datos desde arriba:

- si el próx. nro. de seq. está dentro de la ventana, enviar paquete

timeout(n):

- re-enviar el paquete **n**, re-inicio del *timer*

ACK(n) en [sendbase,sendbase+N]:

- marcar al paquete **n** como recibido
- si **n** es el paquete más pequeño no-ACKed, avanza la base de la ventana al siguiente número de secuencia no-ACKed

receptor

paq. n en [rcvbase, rcvbase+N-1]

- enviar ACK(n)
- fuera de orden: buffer
- en orden: entregar (también entregar los paquetes en orden en *buffer*), avanzar ventana al siguiente paquete no recibido aún

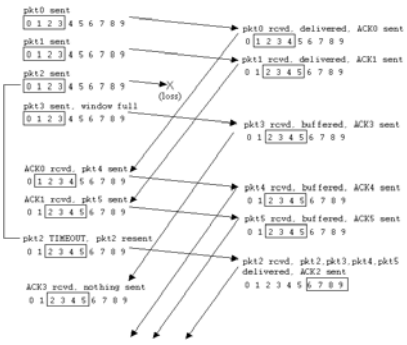
pkt n en [rcvbase-N,rcvbase-1]

- ACK(n)

en otro caso:

- ignorar

La **repetición selectiva** en acción



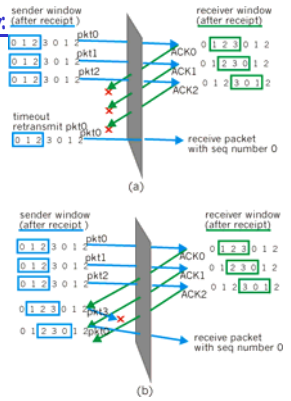
Repetición selectiva: el dilema

Ejemplo:

- seq #'s: 0, 1, 2, 3
- Tamaño de ventana = 3

- para el *receiver* no hay diferencias en los dos escenarios!
- en (a) incorrectamente se pasan datos duplicados como nuevos

P: ¿Qué relación debe haber entre seq # size y el tamaño de ventana?



Mecanismos de transferencia confiable: resumen

- Suma de comprobación
- Temporizador
- Número de secuencia
- Reconocimiento
- Reconocimiento negativo
- Ventana deslizante