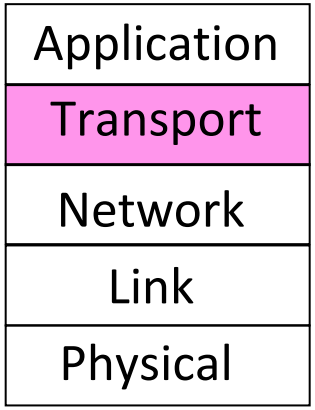
Capítulo 3

Capa de Transporte
Transferencia de datos confiable y control de flujo



Metas

- **Ejercitaremos los siguientes asuntos:**
  1. **Entrega de datos confiable**
  2. Protocolo de parada y espera
  3. Protocolos de tubería
  4. Control de flujo en la capa de transporte
  5. Control de flujo en TCP

Entrega de datos confiable

- La capa de transporte debe soportar al menos un protocolo para **entrega de datos confiable**.
- Estudiaremos distintos protocolos de entrega de datos confiable.
  - Los protocolos irán desde los más simples a los más complejos.
- **Estos protocolos asumen que el canal puede:**
  - Corromper paquetes
  - Perder paquetes
  - La transferencia de datos es en un sentido, o sea hay un emisor y un receptor.
- El protocolo más simple que vemos es el de **parada y espera**. Luego veremos protocolos más complejos llamados de **tubería**.
- Estos protocolos se pueden usar tanto en capa de transporte como en capa de enlace de datos.
  - Pues entrega confiable de datos es un problema de esas capas.

Preliminares

- Dijimos que la CT se ocupa de uso de temporizadores y retransmisiones de paquetes.
  - Paquetes perdidos deben **retransmitirse**.
- **Sabemos que un paquete no se perdió**
  - porque fue confirmado con un **paquete de confirmación de recepción.**
- **¿Cómo sabemos que un paquete se perdió?**
  - Podemos asumir que si pasa un cierto tiempo y no fue confirmado entonces se perdió y hay que retransmitirlo.
- **Para medir el tiempo:**
  - Usar **temporizadores** (timers)

Preliminares

- **Situación:** Se perdió una confirmación de recepción y se envió el paquete de nuevo.
- **Problema:** El mismo paquete llega dos o más veces al receptor y la capa de transporte la pasa a la capa de aplicación más de una vez.
  - **¿Cómo evitar entregar a la capa de aplicación paquetes repetidos?**
- **Solución:** asignar **números de secuencia** a los paquetes que salen.
  - La idea es que dado un número de secuencia de un segmento que acaba de llegar,
  - el receptor puede usar ese número de secuencia para decidir si el segmento es un duplicado y en ese caso descartarlo.

Metas

- **Ejercitaremos los siguientes asuntos:**
  1. Entrega de datos confiable
  2. **Protocolo de parada y espera**
  3. Protocolos de tubería
  4. Control de flujo en la capa de transporte
  5. Control de flujo en TCP

Protocolo de Parada y Espera

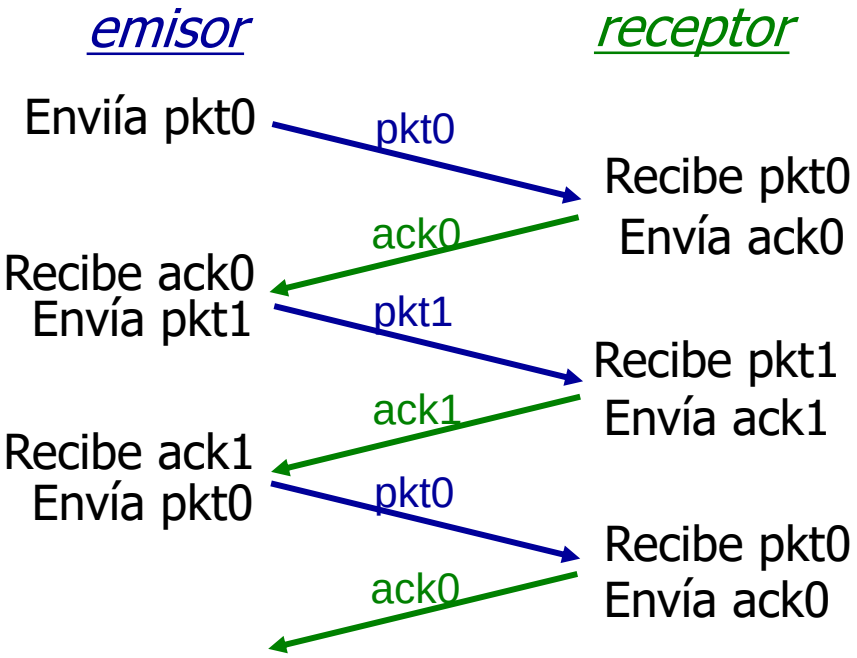
- **Suposición**: el canal de comunicaciones subyacente puede perder paquetes (de datos, de ACKs)
  - Los paquetes tienen N° de secuencias (con 1 bit es suficiente).
  - Se trabaja con Acks
    - El receptor debe especificar N° de secuencia del paquete siendo confirmado.
  - Se usan retransmisiones de paquetes.
    - Para esto se requiere de uso de temporizadores.

Comportamiento del emisor:

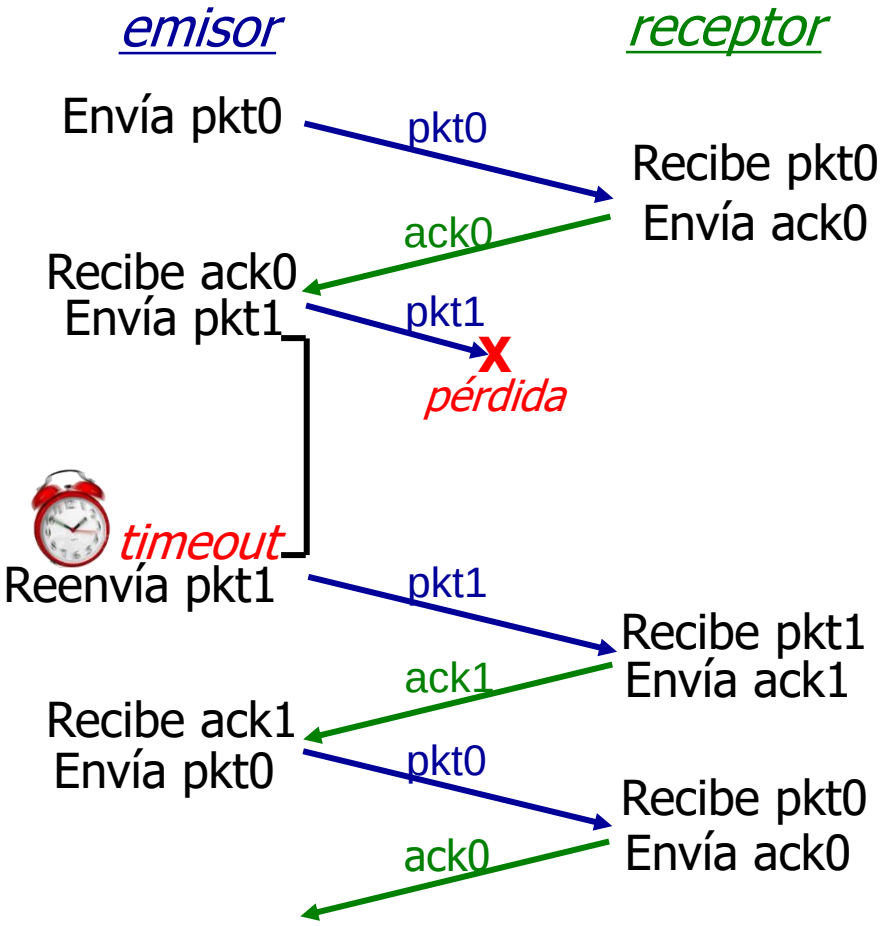
1. El emisor envía paquete P y **para** de enviar.
2. **Espera**: El emisor espera una cantidad "razonable" de tiempo para el ACK
3. Si llega el ACK a tiempo, se envía siguiente paquete. Goto 2.
4. Sino se retransmite paquete P. Goto 2.

- Si hay paquete o ACK demorado pero no perdido:
  - La retransmisión va a ser un duplicado con igual N° de secuencia ; luego se descarta en el receptor.

Parada y Espera en Acción



(a) Sin pérdida

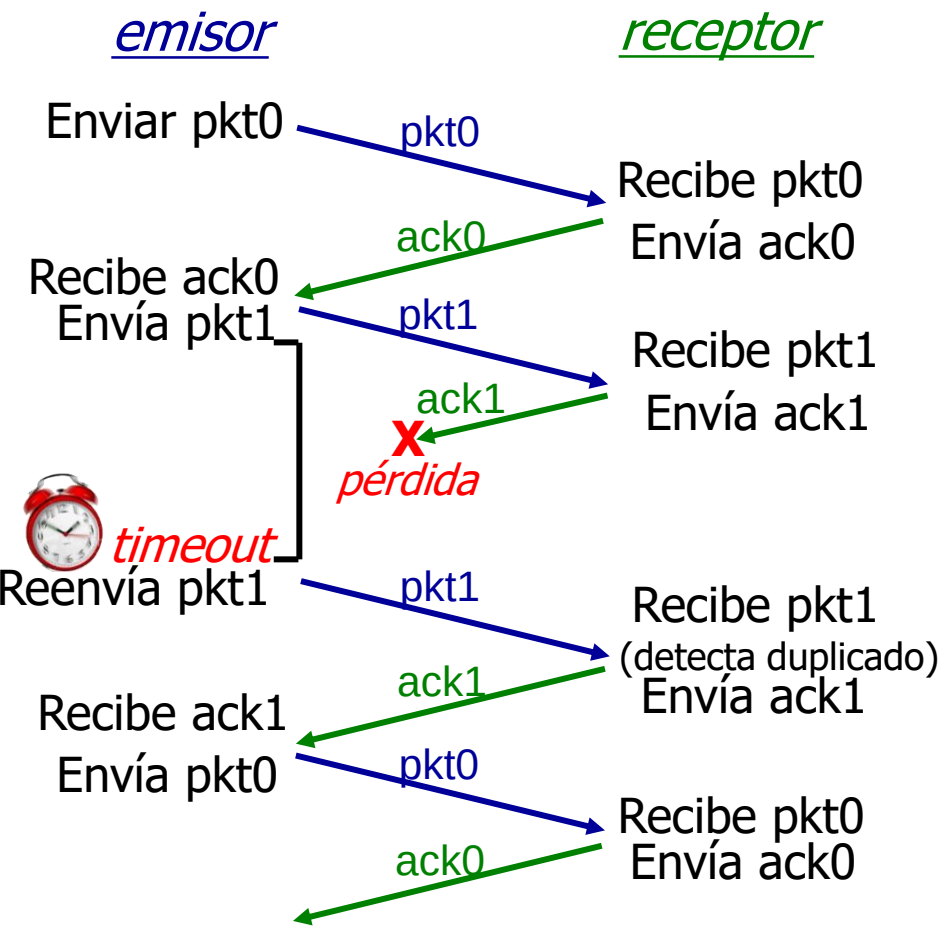


(b) Pérdida de paquete

- **¿Qué pasa si se pierde el pkt1?**

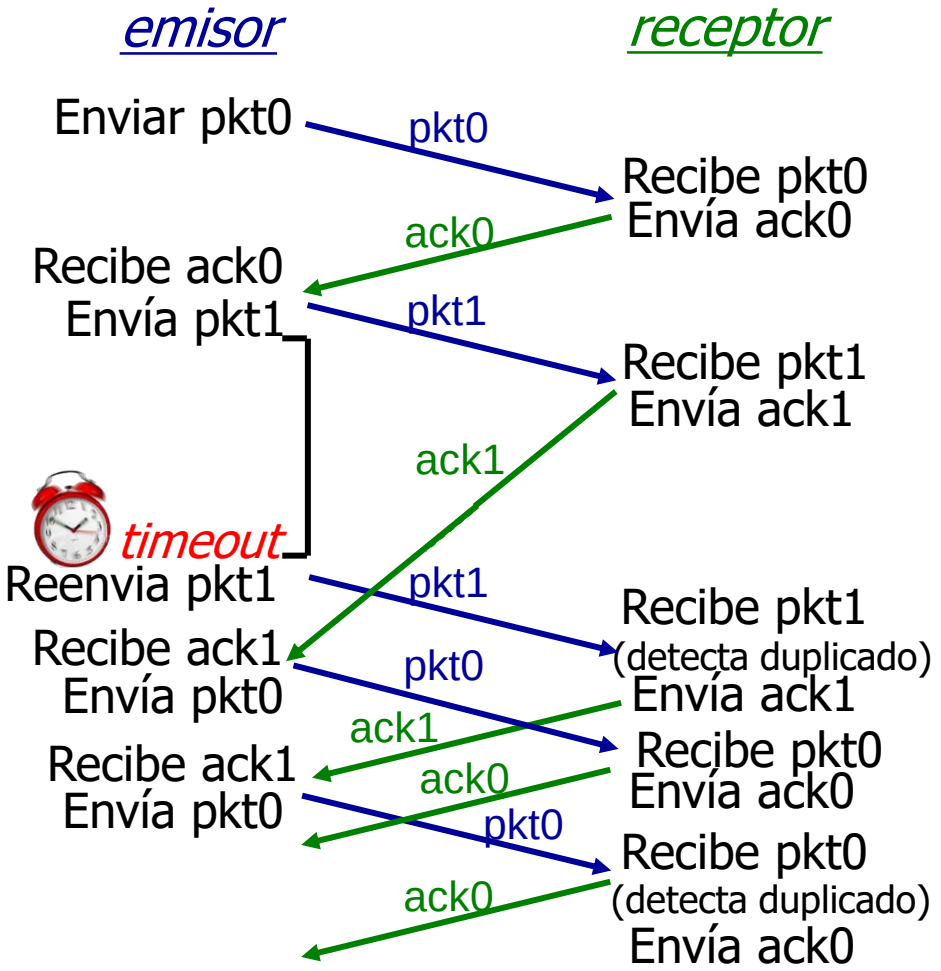
Parada y Espera en Acción

- ¿Qué pasa si se pierde el ack1?



(c) Pérdida de ACK

- ¿Qué pasa si el ack1 se demora?

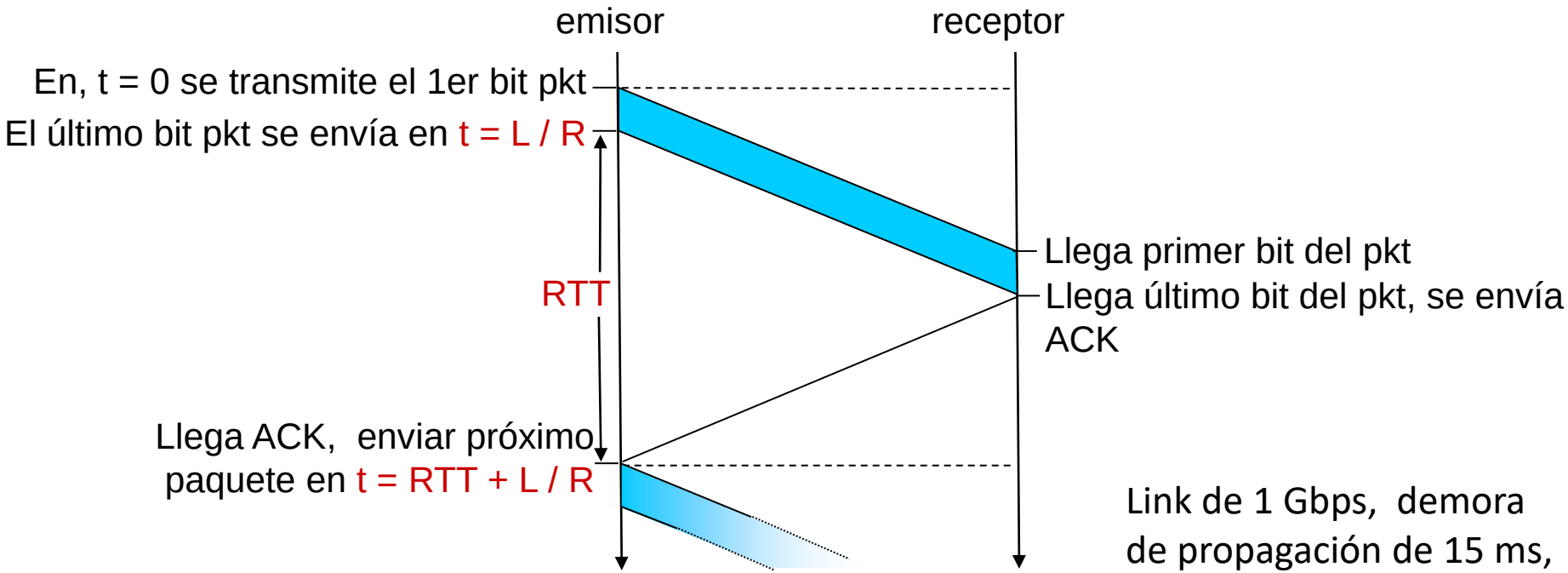


(d) Timeout prematuro/ ACK demorado

Desempeño de Parada y Espera

- Parada y espera tiene un **desempeño pobre.**
- Ejemplo: link de 1 Gbps, demora de propagación de 15 ms, paquetes de 8000 bit:
  - $D_{envío}$ es la demora en enviar un paquete.
  - U_{sender} : ***utilización***– fracción del tiempo en que el emisor está ocupado enviando.
  - RTT es tiempo de ida y vuelta de un bit: RTT = 30 msec.
- El protocolo de red limita el uso de recursos físicos.

Operación de Parada y Espera



Link de 1 Gbps, demora de propagación de 15 ms, paquetes de 8000 bit.

$$D_{envío} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

Suposición: RTT fijo

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

Metas

- **Ejercitaremos los siguientes asuntos:**
  1. Entrega de datos confiable
  2. Protocolo de parada y espera
  3. **Protocolos de tubería**
  4. Control de flujo en la capa de transporte
  5. Control de flujo en TCP

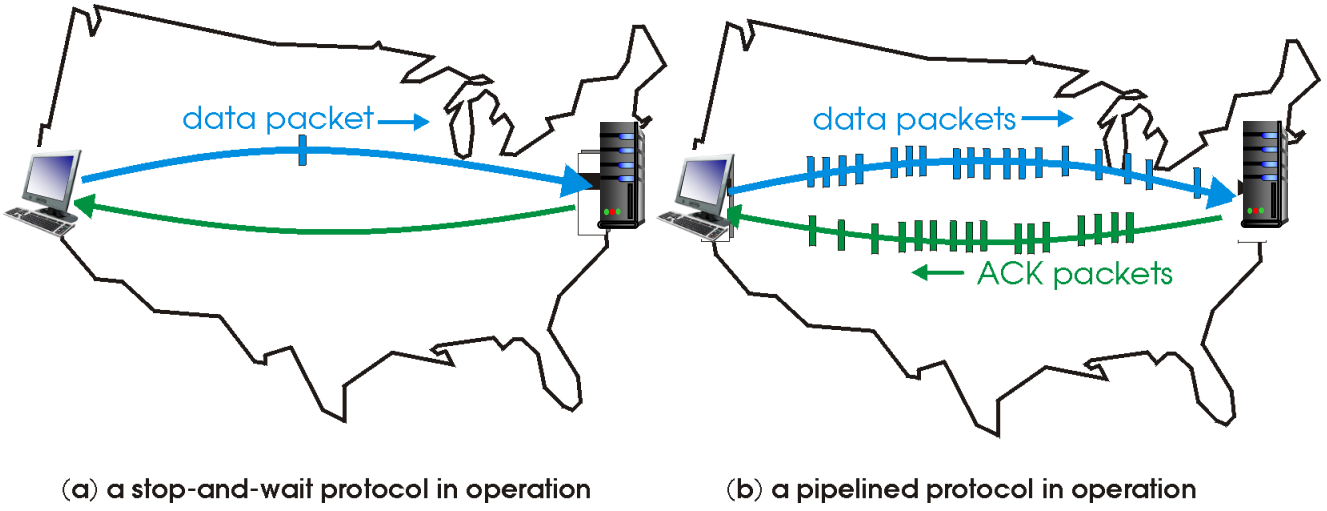
Protocolos de control de flujo

- **Protocolos de tubería**
  - Protocolo Retroceso-N
  - Protocolo de Repetición Selectiva

Protocolos de tubería

Tubería: el emisor puede enviar múltiples paquetes al vuelo a ser confirmados

- El rango de números de sequencia debe ser incrementado usando palabras de más de un bit.
- Hay que usar búferes en el emisor.

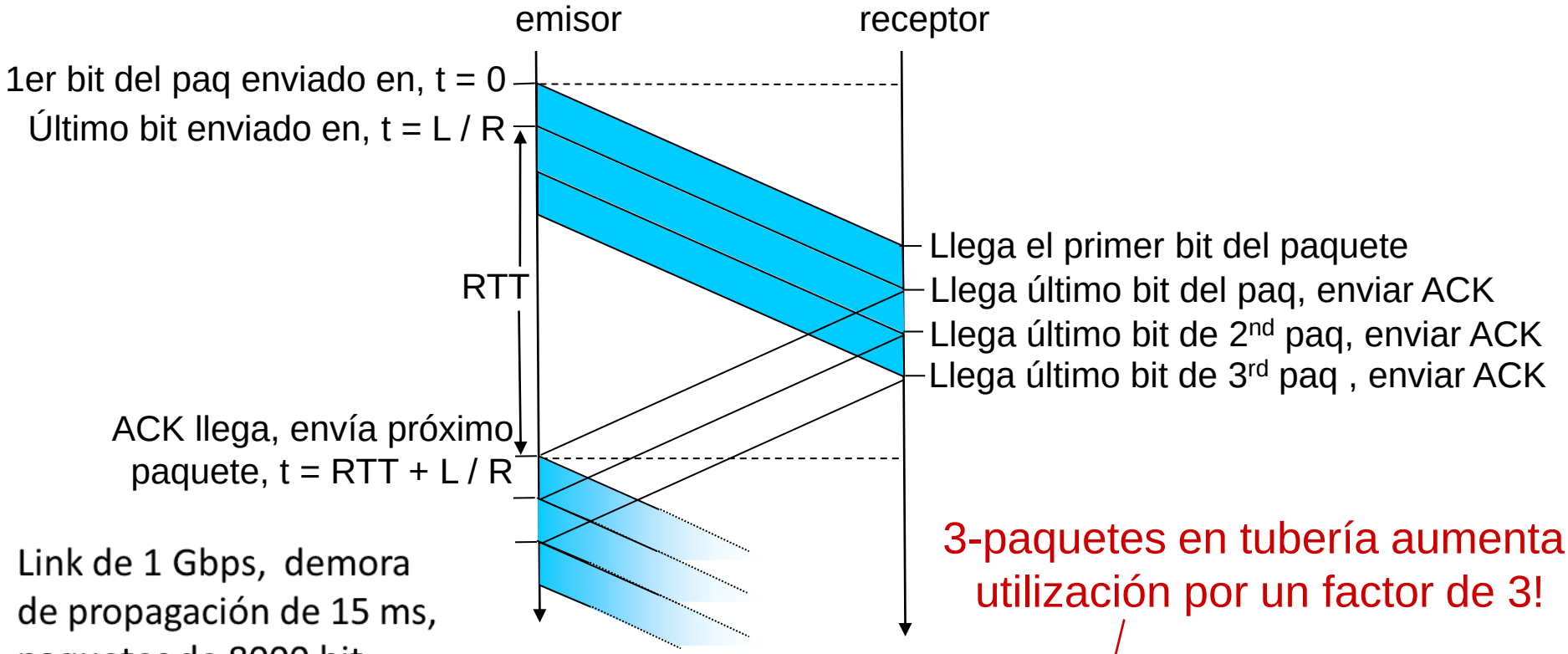


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Hay dos formas genéricas de protocolos de tubería:
  - *retroceso N* y *repetición selectiva*

Tubería: utilización incrementada



Link de 1 Gbps, demora de propagación de 15 ms, paquetes de 8000 bit.

3-paquetes en tubería aumenta utilización por un factor de 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Suposición: el RTT es fijo y no varía (p.ej: dos hosts unidos por cable).

Protocolos de tubería: visión general

Retroceso-N:

- Receptor envía *ack acumulativo*
  - No confirma paquetes si hay un agujero.
- El emisor tiene un timer para el paquete más viejo no confirmado
  - Cuando expira el timer retransmite todos los paquetes no confirmados.

Repetición selectiva:

- El receptor envía *confirmaciones individuales* para cada paquete
- El emisor mantiene un timer para cada paquete no confirmado
  - Cuando el timer expira, retransmite solo ese paquete no confirmado.

Uso de búferes en el emisor

- La **ET emisora *debe*** manejar **búferes para los mensajes de salida**.
- **Esto es necesario porque:**
  - puede hacer falta retransmitirlos
- **¿Cómo se usan búferes en el emisor?**
  - El emisor almacena en búfer todas los segmentos hasta que se confirma su recepción.

Protocolos de control de flujo

- **Protocolos de tubería**
  - **Protocolo Retroceso-N**
  - Protocolo de Repetición Selectiva

Retroceso N

- **Si un paquete *T* a la mitad de una serie larga se daña o pierde:**
  - La CT receptora debe entregar paquetes a la capa de aplicación en secuencia.
  - Por lo que no se pueden entregar a la capa de aplicación los paquetes que llegaron **bien** después de *T*.
- **Problema**: **¿qué debe hacerse con los paquetes correctos que le siguen a un paquete que se perdió?**

Retroceso N

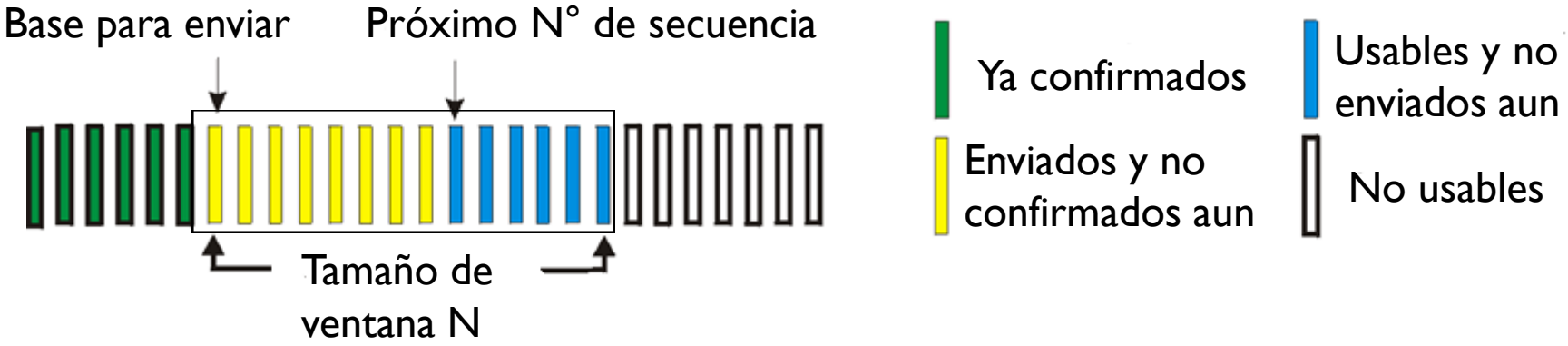
Solución: Con retroceso N el receptor descarta todos los paquetes subsecuentes al paquete perdido, sin enviar ack para los paquetes descartados.

Retroceso N

- **Suposición:**
  - Hay un límite en la cantidad de paquetes enviados y no confirmados + paquetes por enviar que puede almacenar el emisor en búferes.
- **¿Cómo representar ese conjunto de paquetes del emisor eficientemente?**
- Usar ***intervalos de números de secuencia*** dentro del ***espacio de números de secuencia.***
  - Un intervalo de esos recibe el nombre de **ventana corrediza**.

Retroceso-N: en el emisor

- La "ventana" permite hasta *N* paquetes consecutivos sin confirmar
- **ventana emisora** = tramas enviadas sin ack positivo o tramas listas para ser enviadas.

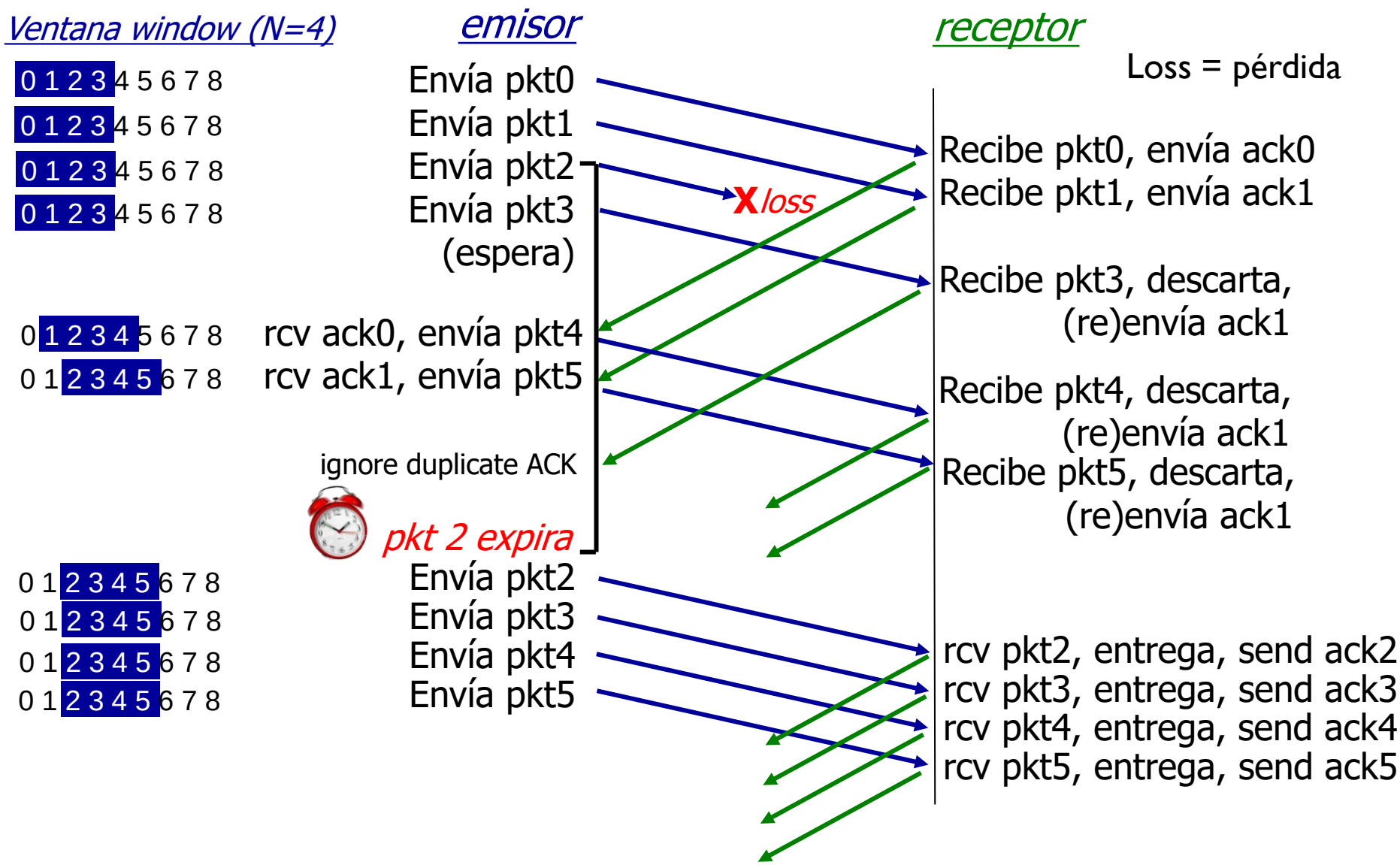


- *timeout(n):* retransmite paquete *n* y todos los paquetes de mayor N° de secuencia en la ventana.

Retroceso N

- **Si mando un paquete de confirmación solamente, ¿qué número de secuencia debe tener?**
- Enviar ACK con N° de secuencia más alto tal que los N° de secuencia anteriores fueron recibidos.
  - A esto se le llama **ACK acumulativo**.
- Si se pierde un segmento llegan bien varios de los siguientes, para estos se generan **ACKs duplicados**.
- Para los números de secuencia, el receptor maneja variable *expectedSeqnum* que es el número de secuencia más chico que no llegó aun.

Retroceso-N en acción



Retroceso N

- **¿Si el espacio de secuencia es de MAX_SEQ + 1 números de secuencia (estos comienzan desde 0), se puede hacer la ventana emisora de tamaño MAX_SEQ + 1?**
- La respuesta es no (Justificación en las 2 filminas siguientes).
- **Conclusión**: El tamaño de la ventana emisora no puede superar MAX_SEQ cuando hay MAX_SEQ + 1 números de secuencia.

Retroceso N

- Considere la siguiente situación con MAX_SEQ = 7.
  1. El emisor envía paquetes 0 a 7.
  2. Llega al emisor una confirmación de recepción, superpuesta para paquete 7.
  3. El emisor envía otros 8 paquetes, con los números de secuencia 0 a 7.
  4. Ahora llega otra confirmación de recepción superpuesta para el paquete 7.
  5. ¡No se sabe si item 4. es un reenvío de ACK o uno nuevo!

Retroceso N

- ¿llegaron con éxito los 8 paquetes que correspondían al segundo bloque o se perdieron (contando como pérdidas los rechazos siguientes a un error)?
  - En ambos casos el receptor podría estar enviando el paquete 7 como confirmación de recepción.
    - El emisor no tiene manera de saberlo.

- **Por lo tanto:** el tamaño de la ventana emisora no puede superar MAX_SEQ cuando hay MAX_SEQ + 1 números de secuencia.

Retroceso N

- **Problema:** **¿Cómo evitar que haya más de *MAX_SEQ* paquetes sin ack pendientes?**
- **Solución**: prohíbir a la CR que moleste con más trabajo.
- **Implementación**: Usar *enable_network_layer* y *disable-network_layer*.

Retroceso N

- **¿Cuál es el problema principal de retroceso N?**
- El uso ineficiente del canal frente a segmentos perdidos o demorados.

Protocolos de control de flujo

- **Protocolos de tubería**
  - Protocolo Retroceso-N
  - **Protocolo de Repetición Selectiva**

Repetición Selectiva

- **¿Qué ocurre si un paquete *T* a la mitad de una serie larga se pierde?**
- La CT receptora debe entregar paquetes a la capa de aplicación en secuencia**.**
  - Por lo que no se pueden entregar a la capa de aplicación los paquetes que llegaron **bien** después de *T*.
- **Problema**: **¿qué debe hacerse con los paquetes correctos que le siguen a un paquete que se perdió?**

Repetición Selectiva

- **Solución (Repetición Selectiva):**
  - Los paquetes en buen estado recibidos después de un **paquete dañado** *E* se **almacenan en búfer**.
  - Cuando el paquete *E* llega correctamente, el receptor entrega a la capa de aplicación, en secuencia, todos los paquetes posibles que ha almacenado en el búfer.

Repetición Selectiva

- **Mecanismo común de retransmisiones:**
  - El temporizador de *E* termina y el emisor lo manda de nuevo.
- **Una solución mejor:**
  - Uso de una ack negativa (NAK) por el receptor.
    - Así se estimula la retransmisión de paquetes antes que los temporizadores terminen y así se mejora el rendimiento.
  - **¿Y si la NAK se pierde?**

Repetición Selectiva

- El receptor confirma individualmente todos los paquetes recibidos correctamente.
  - Hay búferes para paquetes según se necesiten para su entrega eventual en orden a la capa de aplicación.
- El emisor solo reenvía paquetes para los cuales el ACK no fue recibido o se recibió un NAK.
  - Hay un temporizador del emisor para cada paquete no confirmado.

Repetición Selectiva

- **Ventana del emisor**
  - Contiene *N* N°de secuencias consecutivos
  - Limita N°de sequencias a enviar a paquetes no confirmados.
- **¿Qué tipos de paquetes puede haber en la ventana del emisor? (ayuda considerar que estamos en repetición selectiva)**
- Como se confirman todos los paquetes que llegan y puede haber paquetes perdidos:
  - Paquetes enviados y confirmados porque antes hay paquetes no confirmados
  - Paquetes enviados y no confirmados
  - Paquetes listos para enviarse en búfer

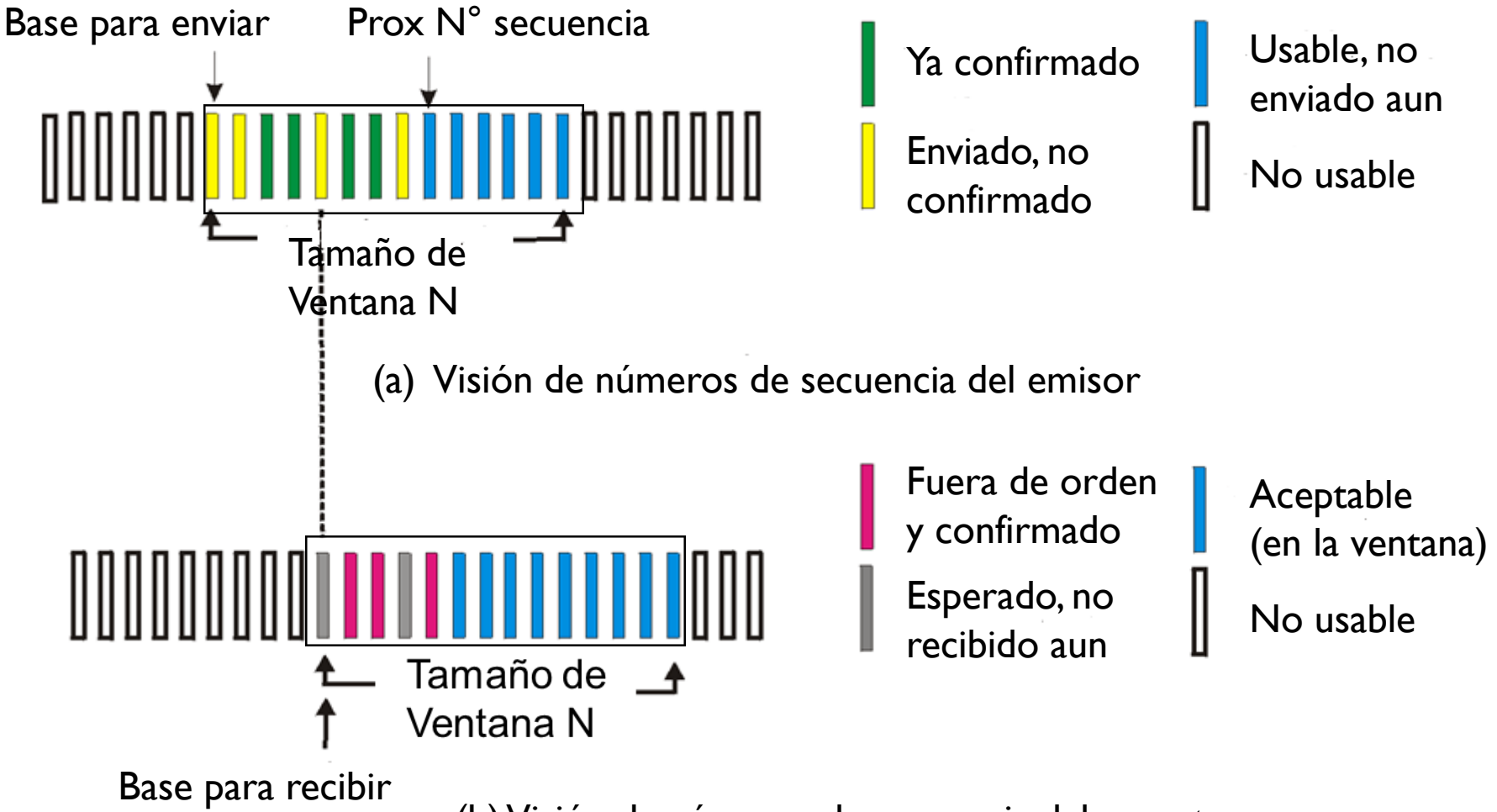
Repetición Selectiva

- **Situación:**
  - Hay un límite para la cantidad de paquetes que puede almacenar en búfer el receptor.
  - **Es necesario almacenar en búfer paquetes**
    - porque puede perderse un paquete y llegar otros a continuación del mismo y en repetición selectiva estos se almacenan.
- **Problema: ¿Cómo representar el conjunto de paquetes que puede almacenar en búfer el receptor?**
- **Solución**: Usar ***intervalos de números de secuencia*** dentro del ***espacio de números de secuencia*.**
  - Un intervalo de esos recibe el nombre de **ventana corrediza**.

Repetición Selectiva

- **Tipos de paquetes que puede haber en la ventana del receptor:**
  - Paquetes esperados y no recibidos
  - Paquetes recibidos fuera de orden
  - Paquetes aceptables en la ventana que no han llegado aun
- **Se mantiene en búfer un paquete aceptado por la ventana receptora**
  - hasta que todos los que le preceden hayan sido pasados a la capa de aplicación.

Repetición Selectiva: ventanas del emisor y del receptor



Repetición Selectiva

- **Algunos detalles de repetición selectiva:**
  - tamaño de ventana emisora comienza en 0 y crece hasta MAX_SEQ.
  - El receptor tiene un búfer para cada N° de secuencia en su ventana.
  - **¿Qué se hace cuando llega un paquete?**
  - Cuando llega un paquete, su número de secuencia es revisado para ver si cae dentro de la ventana.
    - De ser así, y no ha sido recibido aun, se acepta y almacena.

Repetición selectiva

Emisor

Datos vienen de arriba:

- Si el próximo N° secuencia a enviar de la ventana está disponible, almacenar y enviar paquete

timeout(*n*):

- Reenviar paquete *n*, reiniciar timer

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

- marcar paquete *n* como recibido
- Si *n* es paquete más pequeño no confirmado, ***avanzar base de ventana*** al siguiente N° secuencia no confirmado.

Receptor

pkt *n* en [base rcv, base rcv +N-1]

- Enviar ACK(n)
- Fuera de orden: almacenarlo
- En orden: entregar (también entregar paquetes en bufer en orden), avanzar ventana al siguiente paquete que no ha sido recibido aun.

pkt *n* en [base rcv-N, base rcv-1]

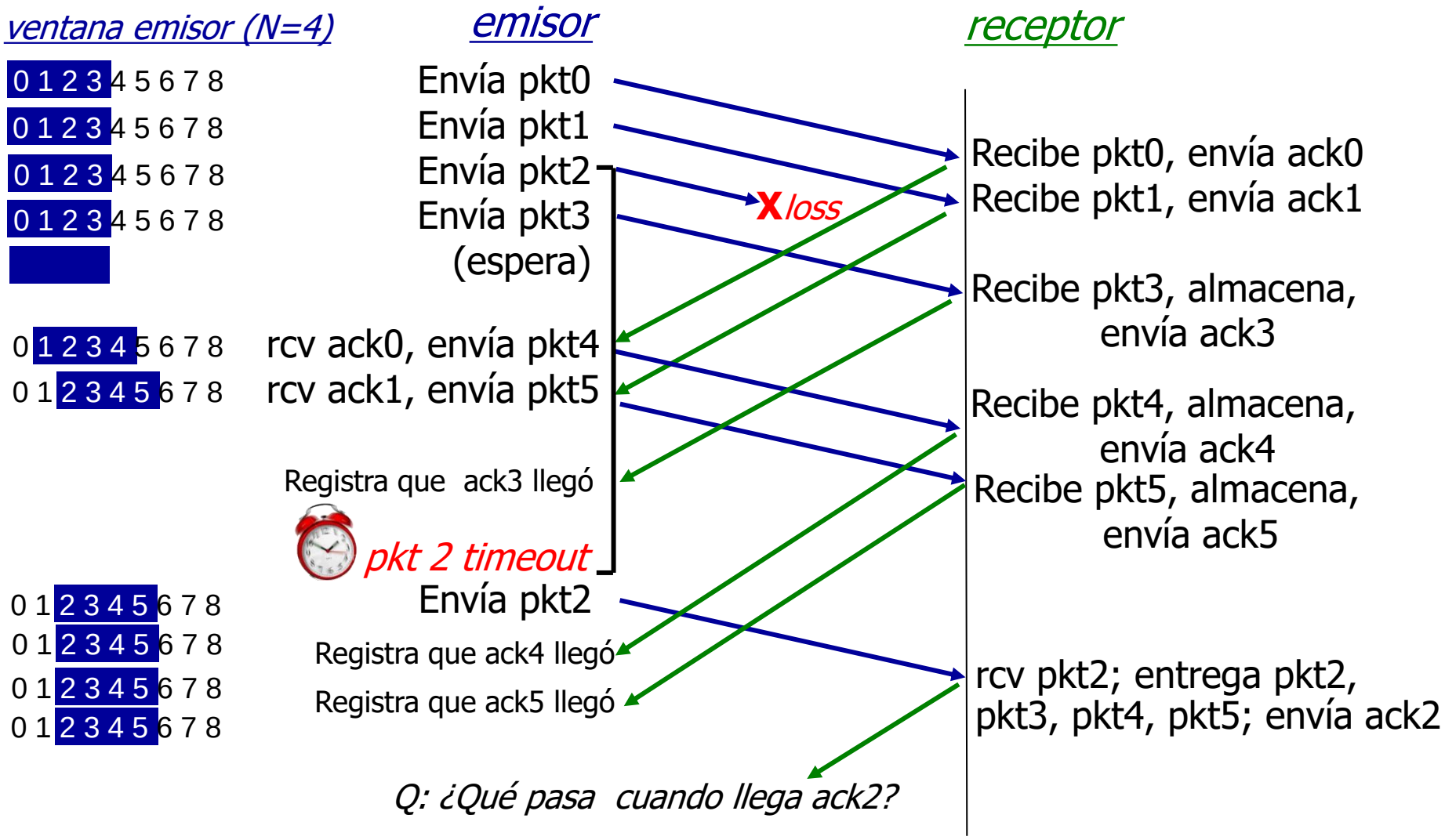
- Enviar ACK(n)

Sino:

- ignorar

Súposiciones: *N* es tamaño de ventana del receptor. Algoritmo sin NAKs

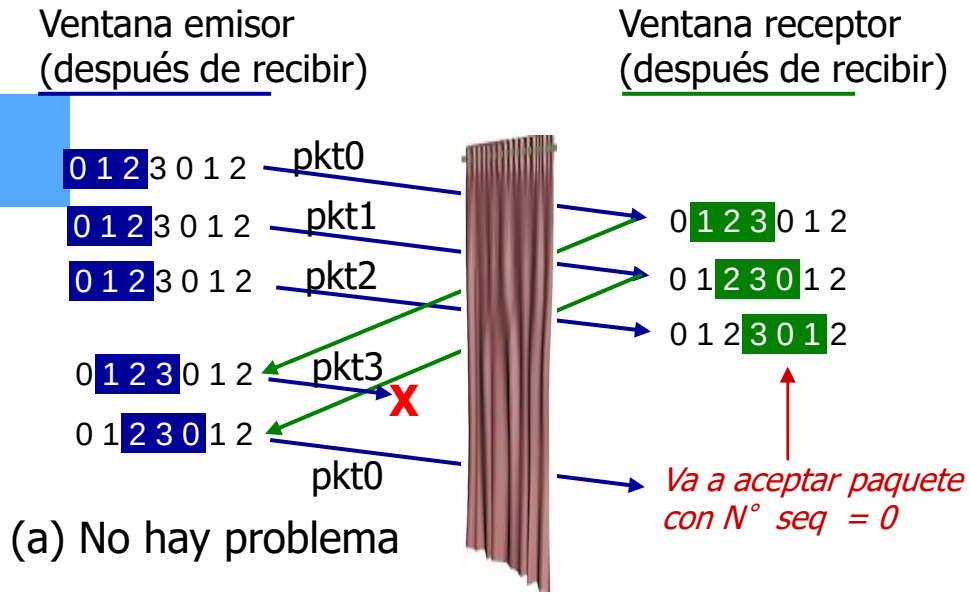
Repetición selectiva en acción



Dilema de repetición selectiva

Ejemplo:

- Espacio de sequencia: 0, 1, 2, 3.
- Tamaño ventana =3 en emisor y receptor

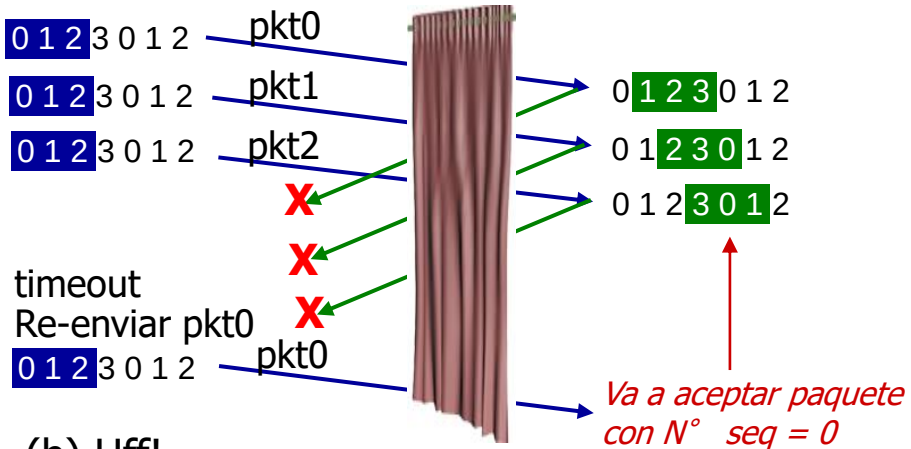


(a) No hay problema

- El receptor no ve la diferencia en 2 escenarios:!
- Datos duplicados aceptados como nuevos en (b)

El receptor no puede ver el lado del emisor.
¡El comportamiento del receptor es idéntico en ambos casos !
¡Algo está muy mal!

Q: **¿Qué relación entre tamaño de N° secs y tamaño de ventana en receptor debe haber para evitar el problema en (b)?**



(b) Uff!

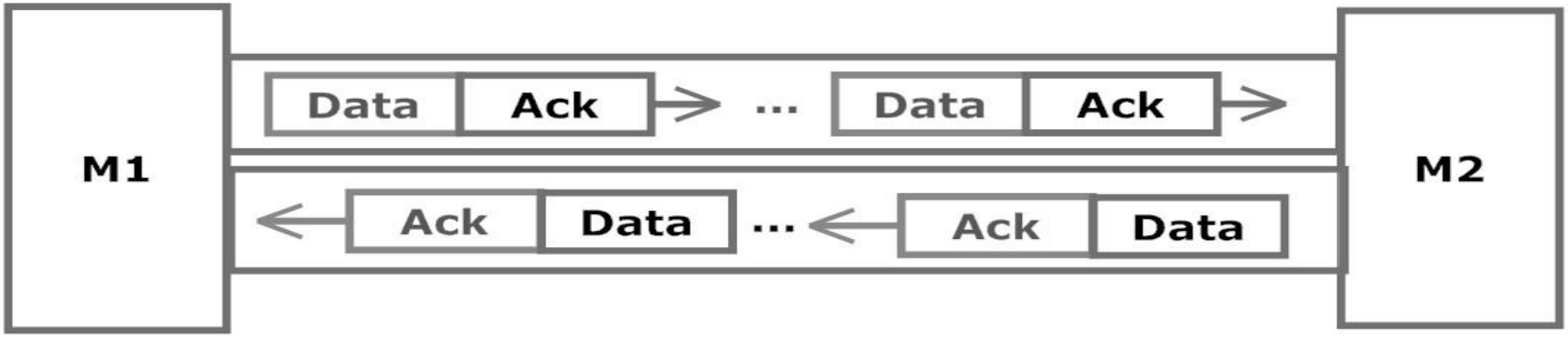
Repetición Selectiva

- **Regla para el tamaño de la ventana receptora:**
  - Tamaño de ventana receptora = (*MAX_SEQ* + 1)/2.
  - Con tamaños mayores de ventana receptora no funciona.

Repetición Selectiva

- En encabezado de paquete hay N° de secuencia de *k* bits.
- **Problema**: **¿Cómo transmitir datos entre dos máquinas y en ambas direcciones eficientemente?**

Repetición Selectiva



- **Solución**: **llevar a caballito (piggybacking)**.
  - cuando llega un segmento *S* con datos, el receptor se aguanta y espera hasta que la capa de aplicación le pasa el siguiente paquete *P*.
  - La confirmación de recepción de *S* se anexa a *P* en un segmento de salida (usando el **campo ack** en el encabezado del segmento de salida).

Repetición Selectiva

- **Problema**: ¿Cómo extender repetición selectiva para tener flujos de datos entre 2 máquinas en las dos direcciones?
- **Solución**:
  - Se usa llevar a caballito.
  - La capa de transporte para mandar un ack, debe esperar por un paquete al cual superponer un ack.

Repetición Selectiva

- **Problema:** **¿Cómo evitar retrasar demasiado envío de confirmaciones de recepción por no tener tráfico de regreso?**
- **Solución**: **método que usa temporizador auxiliar**
  - tras llegar un paquete de datos en secuencia, se arranca un temporizador auxiliar mediante *start_ack_timer*.
  - Si no se ha presentado tráfico de regreso antes de que termine este temporizador, se envía un paquete de ack independiente.

Repetición Selectiva

- tiempo de temporizador auxiliar << tiempo de temporizador de retransmisiones.
  - << significa mucho menor.
- **¿Por qué?**
  - para asegurarse que la ack de un paquete correctamente recibido llegue antes que el emisor termine su temporización y retransmita el paquete.

Metas

- **Ejercitaremos los siguientes asuntos:**
  1. Entrega de datos confiable
  2. Protocolo de parada y espera
  3. Protocolos de tubería
  4. **Control de flujo en la capa de transporte**
  5. Control de flujo en TCP

Control de flujo

- **Control de Flujo**: Hay que evitar que un host emisor rápido desborde a un host receptor lento.
- **Tipo de control de flujo del que se ocupa la capa de enlace de datos:**
  - Control de flujo entre dos máquinas directamente conectadas entre sí (pueden ser enrutador o host).
- **¿Por qué puede necesitarse control de flujo en la capa de transporte si la capa de enlace de datos lo hace?**
- El receptor puede demorarse en procesar mensajes debido a los problemas de la red:
  - pérdida de segmentos,
  - no se pueden procesar segmentos porque faltan anteriores.

Uso de búferes

- Podemos asumir que el receptor maneja búferes para los mensajes que llegan.
- **¿Porqué esto es necesario?**
  - Si la llegada de segmentos del emisor es mucho más rápido que el receptor para procesar los segmentos recibidos,
    - entonces el receptor necesitará poder almacenar segmentos antes de procesarlos.
  - El receptor puede acumular una cantidad de segmentos suficientes antes de pasarlos a la capa de aplicación para que los procese.
  - Los segmentos pueden llegar desordenados;
    - por lo tanto si llegan un grupo de segmentos y faltan segmentos previos a ellos, habrá que almacenarlos segmentos de ese grupo en buffer.

Uso de búferes

- **Problema: ¿Qué hace el receptor con los búferes si tiene varias conexiones?**
- **Solución 1:** se usan los búferes a medida que llegan segmentos.
- **Solución 2:** se dedican conjuntos de búferes específicos a conexiones específicas.
- **Comparar las dos soluciones**

Uso de búferes

- **Problema: ¿Qué hace el receptor cuando entra un segmento? (me refiero al uso de búferes)**
- **Solución**:
  - Cuando entra un segmento el receptor intenta adquirir un búfer nuevo;
  - si hay uno disponible, se acepta el segmento; de otro modo se lo descarta.

Control de flujo

- **Suposición:** cambia el patrón de tráfico de la red; se abren y cierran varias conexiones en el receptor.
- **Consecuencias:**
  - El receptor y el emisor deben ajustar dinámicamente sus alojamientos de búferes.
    - Esto significa ventanas de tamaños variables.
  - Ahora el emisor no sabe cuántos datos puede mandar en un momento dado, pero sí sabe cuántos datos le gustaría mandar.
- **Problema: ¿Qué reglas cumpliría un protocolo entonces?**

Control de flujo

- **Solución**: El host emisor **solicita espacio en búfer en el otro extremo.**
  - Para estar seguro de no enviar de más y sobrecargar al receptor.
  - Porque sabe cuánto necesita.
- **¿Qué pasa con el receptor al recibir ese pedido?**
  - Sabe cuál es su situación y cuánto espacio puede otorgar.
  - Aquí el receptor reserva una cierta cantidad de búferes al emisor.
- Los búferes podrían repartirse por conexión, o no.
- **¿Qué pasa si los búferes se reparten por conexión y aumenta la cantidad de conexiones abiertas?**
  - El receptor necesita ajustar dinámicamente sus reservas de búferes.

Control de flujo

- **¿Cómo funciona la comunicación entre host emisor y host receptor usando la solución?**
  1. **Inicialmente el emisor solicita una cierta cantidad de búferes, con base en sus necesidades percibidas.**
  2. El receptor otorga entonces tantos búferes como puede.
  3. El receptor, sabiendo su capacidad de manejo de búferes podría indicar al emisor ***"te he reservado X búferes"*.**
     - **¿Cómo hace el receptor con las confirmaciones de recepción?**
     - ***El receptor puede incorporar tanto las ack como las reservas de búfer al en el mismo segmento.***

Control de flujo

- **¿Cómo funciona la comunicación entre host emisor y host receptor usando la solución? – Cont.**
  - El emisor lleva la cuenta de su **asignación de búferes** con el receptor.
  - **¿qué pasa con la asignación de búferes (disponibles) en el emisor cada vez que el emisor envía un segmento?**
    - Debe disminuir su asignación
  - **¿qué pasa si la asignación de búferes (disponibles) en el emisor llega a 0?**
    - El emisor debe detenerse por completo

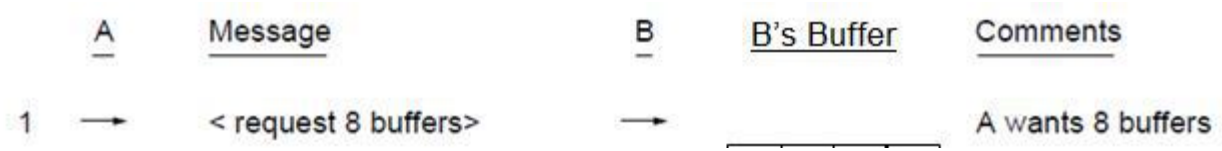
- **Ejercicio**: Suponer que se tiene una conexión entre un emisor y un receptor, que los números de secuencia son de 4 bits (o sea van de 0 a 15). Asumir que el receptor tiene 4 búferes en total, todos de igual tamaño. Suponer que se usa la **solución 2**. Mostrar la comunicación entre emisor y receptor de acuerdo a los siguientes eventos:
  1. El emisor pide 8 búferes.
  2. El receptor otorga 4 búferes y espera el segmento de N° de secuencia 0.
  3. El Emisor envía 3 segmentos de datos , los dos primeros llegan y el tercero se pierde.
  4. El receptor confirma los 2 primeros segmentos de datos y otorga 3 búferes.
  5. El emisor envía dos segmentos de datos nuevos que llegan y luego reenvía el segmento de datos que se perdió.
  6. El receptor confirma todos los segmentos de datos y otorga 0 búferes.
  7. El receptor otorga un búfer
  8. El receptor otorga 2 búferes
  9. El emisor manda 2 segmentos de datos
  10. El receptor otorga 0 búferes
  11. El receptor otorga 4 búferes pero este mensaje se pierde
  - Para segmentos de datos enviados indicar número de secuencia
  - Para segmentos de respuesta indicar cantidad de búferes otorgados y segmentos confirmados, asumir que no se envían datos en estos segmentos.
  - Mostrar asignación de números de secuencia de segmentos recibidos a búferes del receptor

Control de flujo y uso de búferes

1. El emisor pide 8 búferes.

Control de flujo y uso de búferes

1. El emisor pide 8 búferes.



2. El receptor otorga 4 búferes y espera el segmento de N° de secuencia 0.

Control de flujo y uso de búferes

1. El emisor pide 8 búferes.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers

2. El receptor otorga 4 búferes y espera el segmento de N° de secuencia 0.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only

3. El Emisor envía 3 segmentos de datos , los dos primeros llegan y el tercero se pierde.

Control de flujo y uso de búferes

1. El emisor pide 8 búferes.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers

2. El receptor otorga 4 búferes y espera el segmento de N° de secuencia 0.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only

3. El Emisor envía 3 segmentos de datos , los dos primeros llegan y el tercero se pierde.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	•••	0 1 2 3	Message lost but A thinks it has 1 left

4. El receptor confirma los 2 primeros segmentos de datos y otorga 3 búferes.

Control de flujo y uso de búferes

4. El receptor confirma los 2 primeros segmentos de datos y otorga 3 búferes.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4

5. El emisor envía dos segmentos de datos nuevos que llegan y luego reenvía el segmento de datos que se perdió.

Control de flujo y uso de búferes

4. El receptor confirma los 2 primeros segmentos de datos y otorga 3 búferes.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4

5. El emisor envía dos segmentos de datos nuevos que llegan y luego reenvía el segmento de datos que se perdió.

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	1 2 3 4	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	1 2 3 4	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	1 2 3 4	A times out and retransmits

6. El receptor confirma todos los segmentos de datos y otorga 0 búferes.
7. El receptor otorga un búfer

Control de flujo y uso de búferes

6. El receptor confirma todos los segmentos de datos y otorga 0 búferes.
7. El receptor otorga un búfer

Control de flujo y uso de búferes

6. El receptor confirma todos los segmentos de datos y otorga 0 búferes.
7. El receptor otorga un búfer

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	1 2 3 4	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	1 2 3 4	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	1 2 3 4	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	1 2 3 4	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	2 3 4 5	A may now send 5

8. El receptor otorga 2 búferes
9. El emisor manda 2 segmentos de datos

Control de flujo y uso de búferes

8. El receptor otorga 2 búferes
9. El emisor manda 2 segmentos de datos

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	1 2 3 4	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	1 2 3 4	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	1 2 3 4	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	1 2 3 4	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	2 3 4 5	A may now send 5
12	←	<ack = 4, buf = 2>	←	3 4 5 6	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	3 4 5 6	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	3 4 5 6	A is now blocked again

10. El receptor otorga 0 búferes
11. El receptor otorga 4 búferes pero este mensaje se pierde

Control de flujo y uso de búferes

10. El receptor otorga 0 búferes
11. El receptor otorga 4 búferes pero este mensaje se pierde

	A	Message	B	B's Buffer	Comments
1	→	< request 8 buffers>	→		A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	0 1 2 3	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	0 1 2 3	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	0 1 2 3	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	0 1 2 3	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	1 2 3 4	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	1 2 3 4	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	1 2 3 4	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	1 2 3 4	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	1 2 3 4	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	2 3 4 5	A may now send 5
12	←	<ack = 4, buf = 2>	←	3 4 5 6	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	3 4 5 6	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	3 4 5 6	A is now blocked again
15	←	<ack = 6, buf = 0>	←	3 4 5 6	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	7 8 9 10	Potential deadlock

Alojamiento de búferes dinámico. Las flechas muestran la dirección de la transmisión.
(…) indica segmento perdido.

Control de flujo

- **Generalizar la situación de la línea 16 de la figura anterior**
- **Situación**: Información de reserva de búferes viaja en segmento que no contiene datos y ese segmento se pierde.
  - Esto termina ocasionando **deadlock**.
- **Problema**: ¿Cómo evitar esta situación de deadlock?
- **Solución**: Cada host puede enviar periódicamente un **segmento de control** con el ack y estado de búferes de cada conexión.
  - Así el estancamiento ***se romperá*** tarde o temprano.

Metas

- **Ejercitaremos los siguientes asuntos:**
  1. Entrega de datos confiable
  2. Protocolo de parada y espera
  3. Protocolos de tubería
  4. Control de flujo en la capa de transporte
  5. **Control de flujo en TCP**

Control de flujo en TCP

- **No se requiere:**
  - que los emisores envíen datos tan pronto como llegan de la aplicación.
  - que los receptores envíen confirmaciones de recepción tan pronto como sea posible.
  - que los receptores entreguen datos a la aplicación apenas los reciben.
    - Esta libertad puede explotarse para mejorar el desempeño.

Control de flujo en TCP

- **Campo Tamaño de ventana** en el encabezado TCP:
  - **N° de bytes que pueden enviarse** comenzando por el byte cuya recepción se ha confirmado.
  - 0: indica que se han recibido los bytes hasta *n° de confirmación de recepción – 1*, inclusive, pero el receptor quisiera **no recibir más datos por el momento.**
  - El **permiso para enviar** puede otorgarse enviando un segmento con el mismo *n° de confirmación de recepción* y un campo tamaño de ventana **distinto de 0**.

Control de flujo en TCP

- **¿Qué se hace si la ventana anunciada por el receptor es de 0?**
  - El emisor debe detenerse hasta que el **proceso de aplicación del host receptor** retire algunos datos del búfer
  - en cuyo momento el TCP puede anunciar una ventana más grande.

Control de flujo en TCP

- En TCP los hosts en cada lado de una conexión tienen **un buffer de recepción circular** para la conexión.
- Cuando la conexión TCP recibe bytes en el orden correcto y en secuencia, coloca los datos en el buffer de recepción.

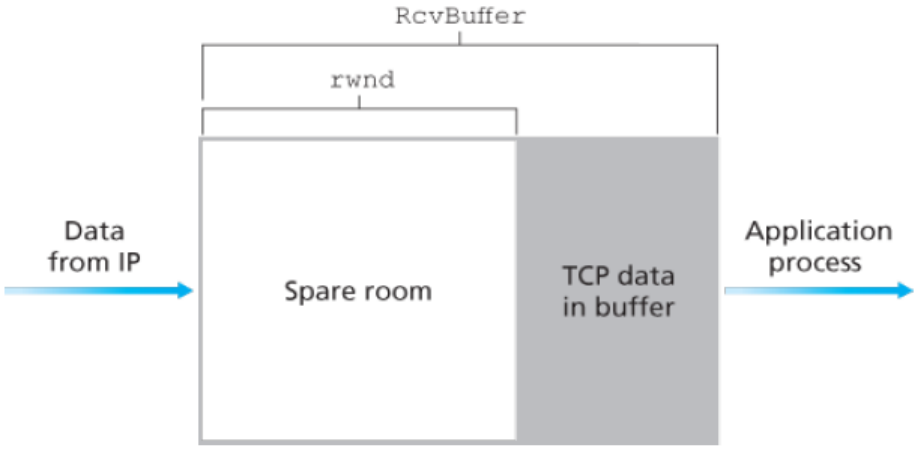


Figure 3.38 The receive window *(rwnd)* and the receive buffer *(RcvBuffer)*

El valor rwnd va en el campo tamaño de ventana

Control de flujo en TCP

- Cálculo de tamaño de ventana:

  Tamaño de ventana =RcvBuffer−[LastByteRcvd−LastByteRead]

- Inicialmente se puede mandar:

  Tamaño de ventana = tamaño RcvBuffer

- Propiedad a respetar en el emisor:

  LastByteSent−LastByteAcked ≤ tamaño de ventana

- **Actividad**: ¿Cómo modificar solución de la filmina 56 para el caso de TCP?

Control de flujo en TCP

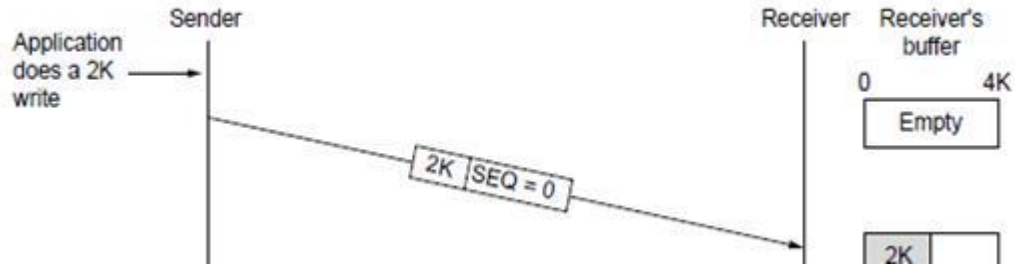
- **Ejercicio**: suponer que hay una conexión entre un emisor y un receptor. El receptor tiene un buffer circular de 4 KB. Mostrar los segmentos enviados en ambas direcciones suponiendo los siguientes cambios de estado en el búfer del receptor:
  1. El búfer del receptor está vacío.
  2. El búfer del receptor tiene 2KB
  3. El búfer del receptor tiene 4KB (lleno)
  4. La aplicación del receptor lee 2KB
  5. El búfer del receptor tiene 3KB
  - Mostrar tamaños y números de secuencia para segmentos enviados.
  - Mostrar tamaño de ventana y número de confirmación de recepción para segmentos recibidos.
  - Mostrar cómo varía el uso del búfer circular.

Control de flujo en TCP

1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB

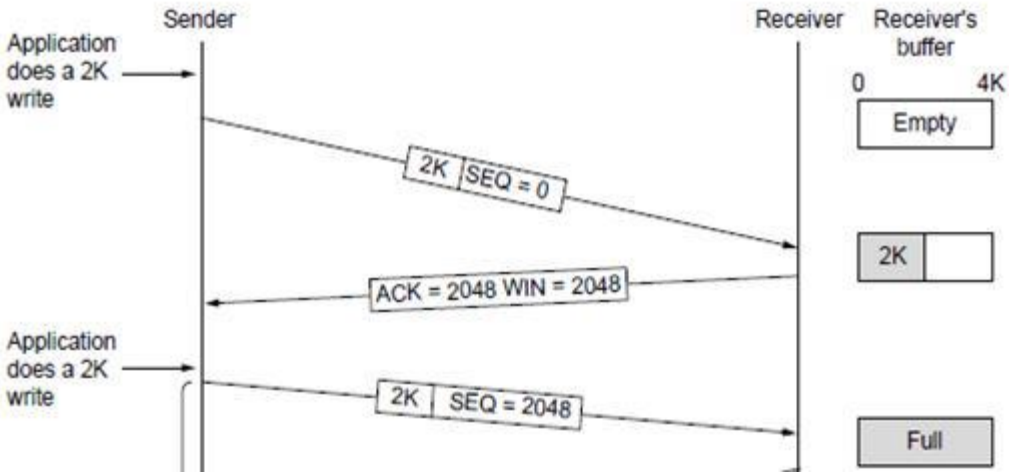
Control de flujo en TCP

1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB
3. El búfer del receptor tiene 4KB (lleno)



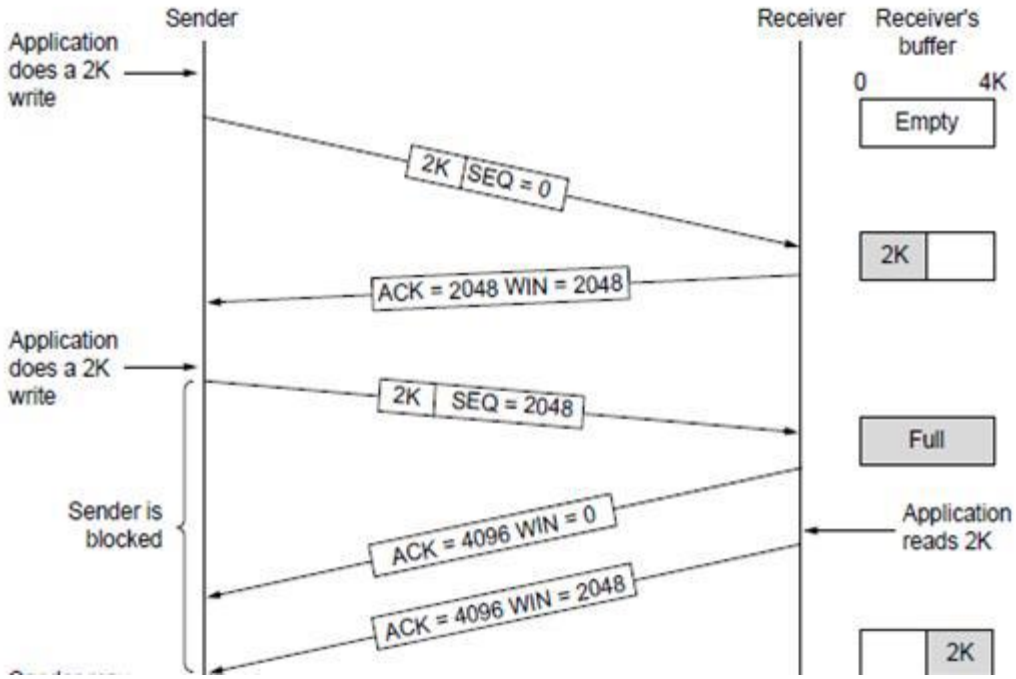
Control de flujo en TCP

1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB
3. El búfer del receptor tiene 4KB (lleno)
4. La aplicación del receptor lee 2KB



Control de flujo en TCP

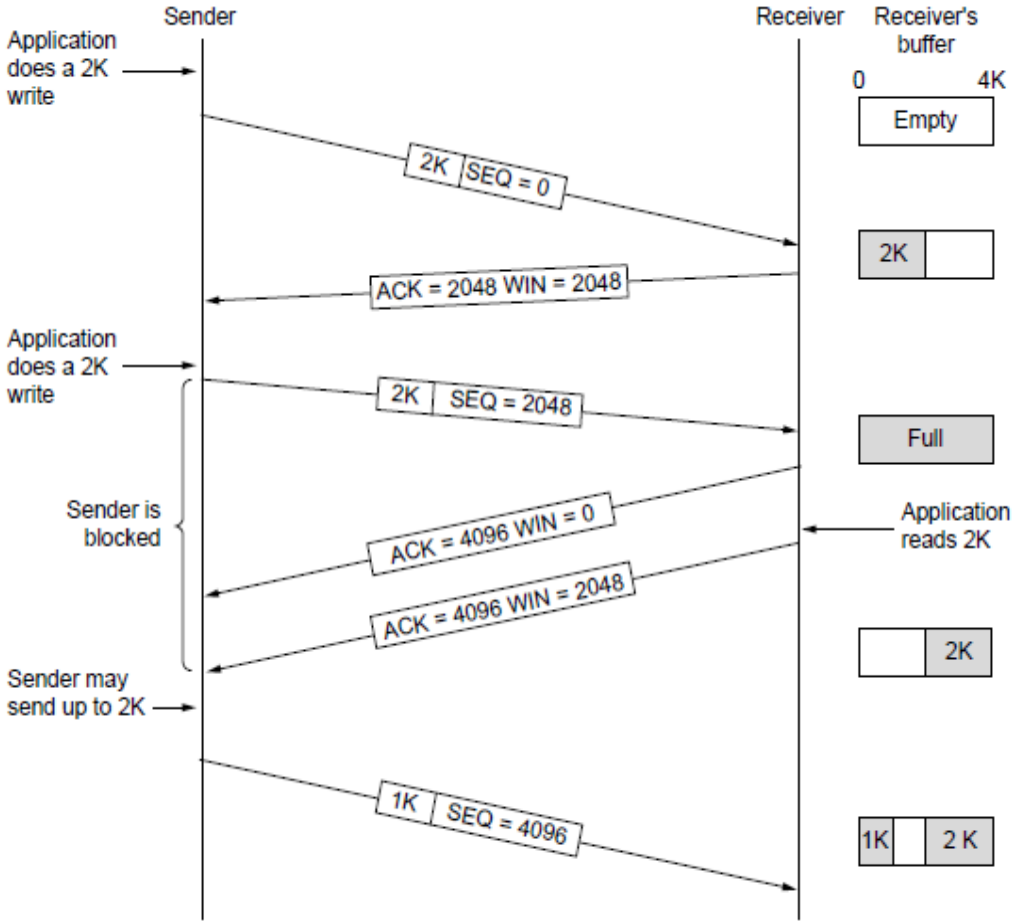
1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB
3. El búfer del receptor tiene 4KB (lleno)
4. La aplicación del receptor lee 2KB
5. El búfer del receptor tiene 3KB



Control de flujo en TCP

1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB
3. El búfer del receptor tiene 4KB (lleno)
4. La aplicación del receptor lee 2KB
5. El búfer del receptor tiene 3KB

- Receptor con búfer circular de 4096 B
- ACK + WIN es el límite del emisor



Control de flujo en TCP

- **Problema**: ¿Cómo manejar pérdidas de segmentos en TCP?
- **Solución 1**: el receptor solicita segmento/s específico/s mediante segmento especial llamado NAK.
  - Tras recibir segmento/s faltante/s, el receptor puede enviar una confirmación de recepción de todos los datos que tiene en búfer.
  - Cuando el receptor nota una brecha entre el número de secuencia esperado y el número de secuencia del paquete recibido, el receptor envía un NAK en un campo de opciones.

Control de flujo en TCP

- **Solución 2**: (**acks selectivos**) el receptor le dice al emisor que piezas recibió.
  - El emisor puede así reenviar los datos no confirmados que ya envió.
  - Se usan dos campos de opciones:
    - **Sack permited option**: se envía en segmento SYN para indicar que se usarán acks selectivos.
    - **Sack option**: Con lista de rangos de números de secuencia recibidos.