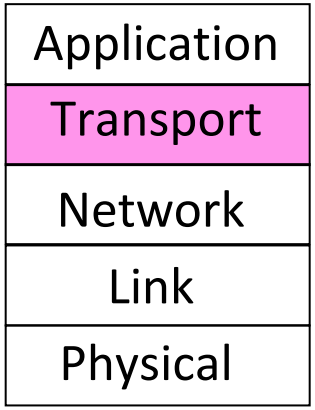
Capítulo 3

Capa de Transporte
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

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).

Uso de búferes en el receptor

- Podemos asumir que el receptor maneja búferes para los mensajes que llegan.
- **Esto es necesario porque:**
  - Si la llegada de segmentos del emisor es mucho más rápida 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.

Control de flujo

- **Situación**: La **capa de aplicación** lee los mensajes que llegan, pero no necesariamente al instante en que los datos llegan.
  - En lugar de hacer eso, la aplicación receptora puede **estar ocupada con otra tarea** y puede no intentar leer los datos hasta bastante después que estos llegaron.
  - Si la **aplicación es demasiado lenta** en leer los datos, el emisor puede saturar los búferes del receptor.

Control de flujo

- La **capa de red** puede tornar al receptor más lento y con menos capacidad de almacenamiento.
  - **Situación 1**: Un enrutador en la ruta entre emisor y receptor daña un paquete;
    - este error se va a detectar por la capa de transporte cuando el paquete dañado llegue al receptor.
    - Si luego de ese paquete dañado llegan varios buenos, la capa de transporte tendrá que almacenarlos y el receptor va a ponerse más lento y con menos capacidad de búfer.
  - **Situación 2**: El algoritmo de enrutamiento hace que cambien las rutas y rutas más lentas son reemplazadas por rutas más rápidas,
    - esto puede hacer que paquetes lleguen al receptor fuera de orden.
    - Si esto sucede, entonces va a obligar a la capa de transporte a almacenar paquetes fuera de orden en búfer y el receptor va a ponerse más lento y con menos capacidad de búfer.

Control de flujo

- La **capa de transporte** puede tornar al receptor más lento y con menos capacidad de almacenamiento.
  - **Situación**: la cantidad de conexiones abiertas aumenta drásticamente.
    - La cantidad de búfer para cada conexión disminuye y el receptor se pone más lento por la cantidad de aplicaciones aumentada.
    - Esta situación sumada a las anteriores puede producir desbordamiento de búferes.

Control de flujo

- **¿Por qué puede necesitarse control de flujo en la capa de transporte si la capa de enlace de datos lo hace?**
  - La capa de enlace de datos no maneja ninguna de las situaciones anteriores.
  - Por lo tanto, necesitan ser tenidas en cuenta por la capa de transporte.
- Si nos quedamos solo con los protocolos de comunicación confiable anteriores, estos no son suficientes para evitar desbordamiento de búferes en el receptor.
  - Hace falta definir un protocolo especial para el control de flujo.

Uso de búferes

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

Uso de búferes

- **Problema: ¿Qué hace el receptor con respecto al uso de búferes cuando entra un segmento?**
- **Solución (que no resuelve el problema de control de flujo)**:
  - 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

- **Cosas que sabemos:**
  - Se pueden dar las situaciones anteriores
  - 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 de control de flujo 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.
- **Cuando el receptor recibe este 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.
- **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

- **Funcionamiento de la comunicación entre host emisor y host receptor usando la solución anterior.**
  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

- **Funcionamiento de la comunicación entre host emisor y host receptor usando la solución anterior – Cont.**
  - El emisor lleva la cuenta de su **asignación de búferes** con el receptor.
  - **Cada vez que el emisor envía un segmento:**
    - Debe disminuir su asignación de búferes disponibles.
  - **Cuando la asignación de búferes (disponibles) llega a 0:**
    - El emisor debe detenerse por completo

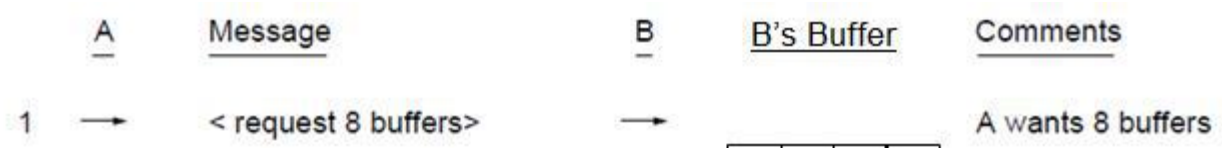
- **Ejercicio**: Suponer que se tiene una conexión entre un emisor y un receptor, que los números de secuencia son de 4 bits (o sea van de 0 a 15). Asumir que el receptor tiene 4 búferes en total, todos de igual tamaño. Suponer que se usa la **solución 2**. Mostrar la comunicación entre emisor y receptor 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

- **Generalización de la situación de la línea 16 de la figura anterior:**
  - La 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

- **No se puede usar el protocolo de control de flujo anterior para TCP.**
  - Porque en TCP los números de secuencia no significan número de paquete.
  - Antes cada búfer ocupado tenía un número de paquete.
  - Ahora los números de secuencia son posiciones en el flujo de datos a enviar.
  - El receptor a lo más puede saber qué rangos de números de secuencia de bytes recibidos tiene en búfer.

Control de flujo en TCP

- **Algunas mejoras que se pueden hacer en relación al protocolo anterior:**
  - Los encabezados de los segmentos recibidos ocupan espacio y no hace falta almacenarlos en búfer.
    - En su lugar se pueden almacenar datos recibidos del flujo de datos.
  - No es necesario que el emisor solicite espacio de búfer al receptor.
    - El receptor sabe de cuanto espacio dispone y cuanto espacio puede otorgar.

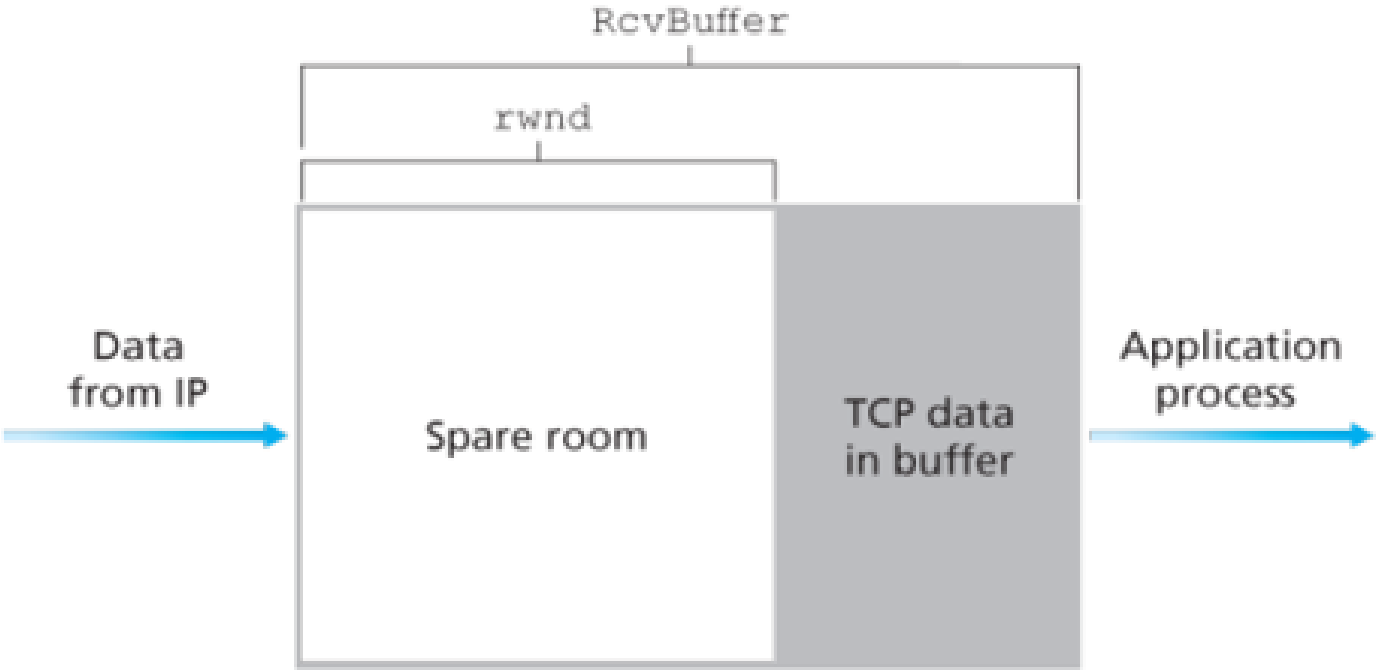
Control de flujo en TCP

- Como no se almacenan encabezados de segmentos, no hace falta guardar segmentos en búferes.
- En su lugar se necesita guardar datos y no hace falta usar varios búferes para esto.
- TCP maneja un **búfer de recepción circular** en el receptor para la conexión (para guardar datos de segmentos que llegan).

Control de flujo en TCP

- Como TCP usa un búfer circular único, el receptor no le puede decir al emisor: ‘te he reservado x búferes’.
- **Para anunciar al emisor la reserva de espacio en búfer:**
  - El receptor puede indicar al emisor la cantidad de bytes consecutivos que se pueden enviar; comenzando por el byte cuya recepción se ha confirmado.
  - A esto se le llama en TCP **tamaño de ventana**.
  - En el encabezado TCP un **campo de tamaño de ventana** (de 16 bits) se usa para indicar esta información.

Control de flujo en TCP



Descripción del estado del búfer del receptor

Control de flujo en TCP

- En TCP el emisor también usa un búfer circular.
- **La cantidad de bytes que el emisor puede enviar al receptor depende**
  - del tamaño del búfer del emisor y del tamaño de ventana.
  - La cantidad de bytes a enviar no debe superar el mínimo de ambos valores.

Control de flujo en TCP

- **La fórmula para calcular el tamaño de ventana el receptor es:**

Tamaño de ventana = RcvBuffer−[LastByteRcvd−LastByteRead]

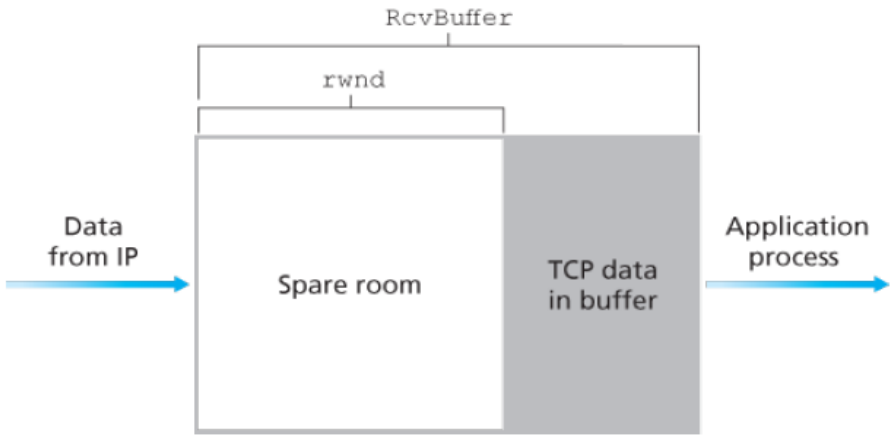


Figure 3.38 The receive window *(rwnd)* and the receive buffer *(RcvBuffer)*

lastByteRcvd es ultimo byte recibido por la capa de aplicación

Control de flujo en TCP

El receptor:

- Cuando la conexión TCP recibe bytes en el orden correcto y en secuencia, coloca los datos en el buffer de recepción.
- El receptor puede confirmar llegada de datos nuevos y anunciar nuevo tamaño de ventana al emisor.
- Si búfer de recepción está lleno, avisar tamaño de ventana de cero.
- Una vez que el receptor entrega a la capa de aplicación X datos de búfer de recepción lleno, puede avisar al emisor de un tamaño de ventana de X.

Control de flujo en TCP

El emisor:

- Si el tamaño de ventana anunciado es cero el emisor no podrá enviar datos.
- El emisor envía segmentos cumpliendo la siguiente propiedad:

$$LastByteSent - LastByteAcked \leq tamaño\ de\ ventana.$$

Control de flujo en TCP

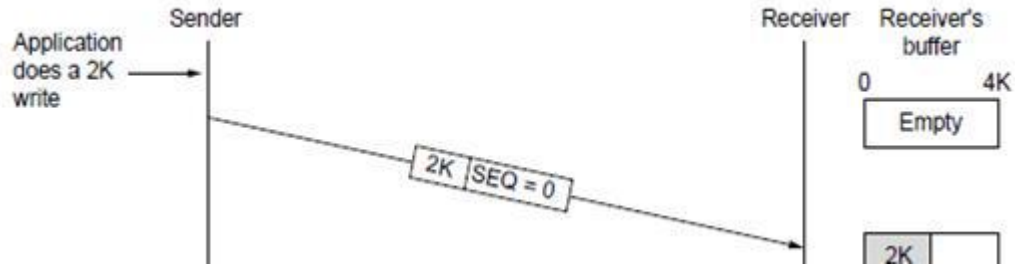
- **Ejercicio**: suponer que hay una conexión entre un emisor y un receptor. El receptor tiene un buffer circular de 4 KB. Mostrar los segmentos enviados en ambas direcciones suponiendo los siguientes cambios de estado en el búfer del receptor:
  1. El búfer del receptor está vacío.
  2. El búfer del 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

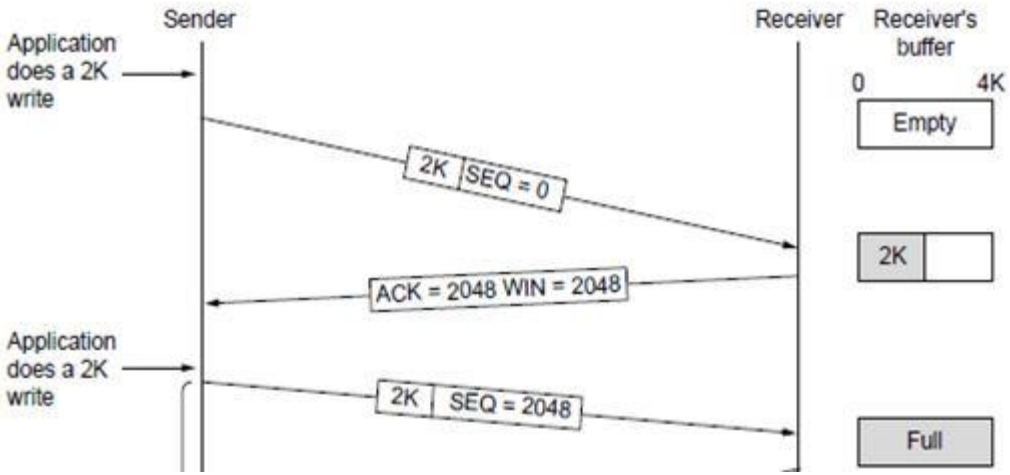
Control de flujo en TCP

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



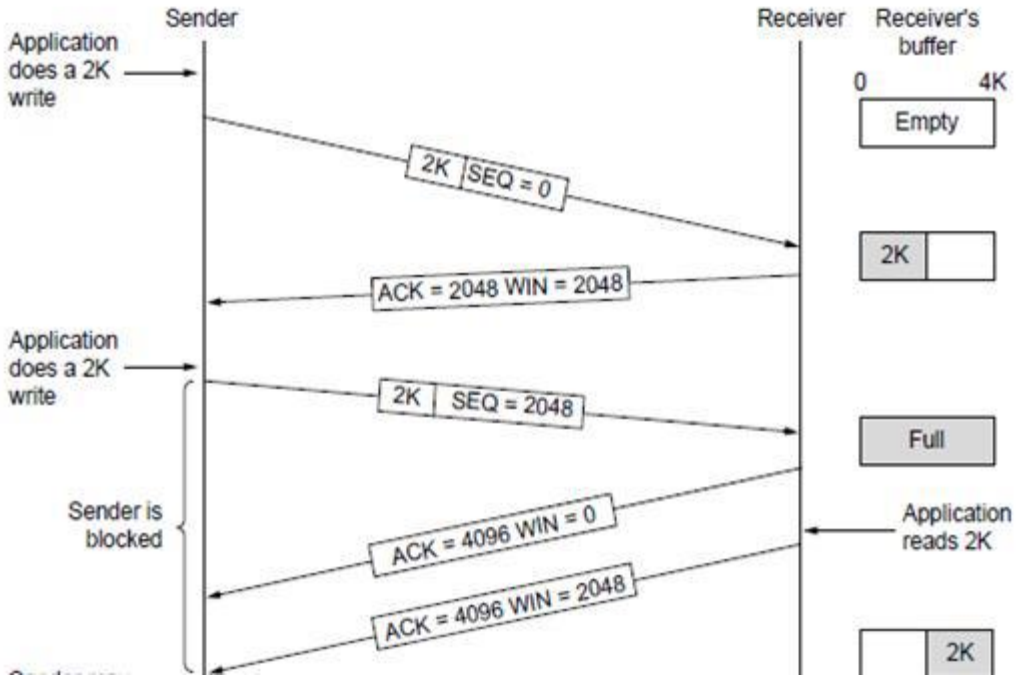
Control de flujo en TCP

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

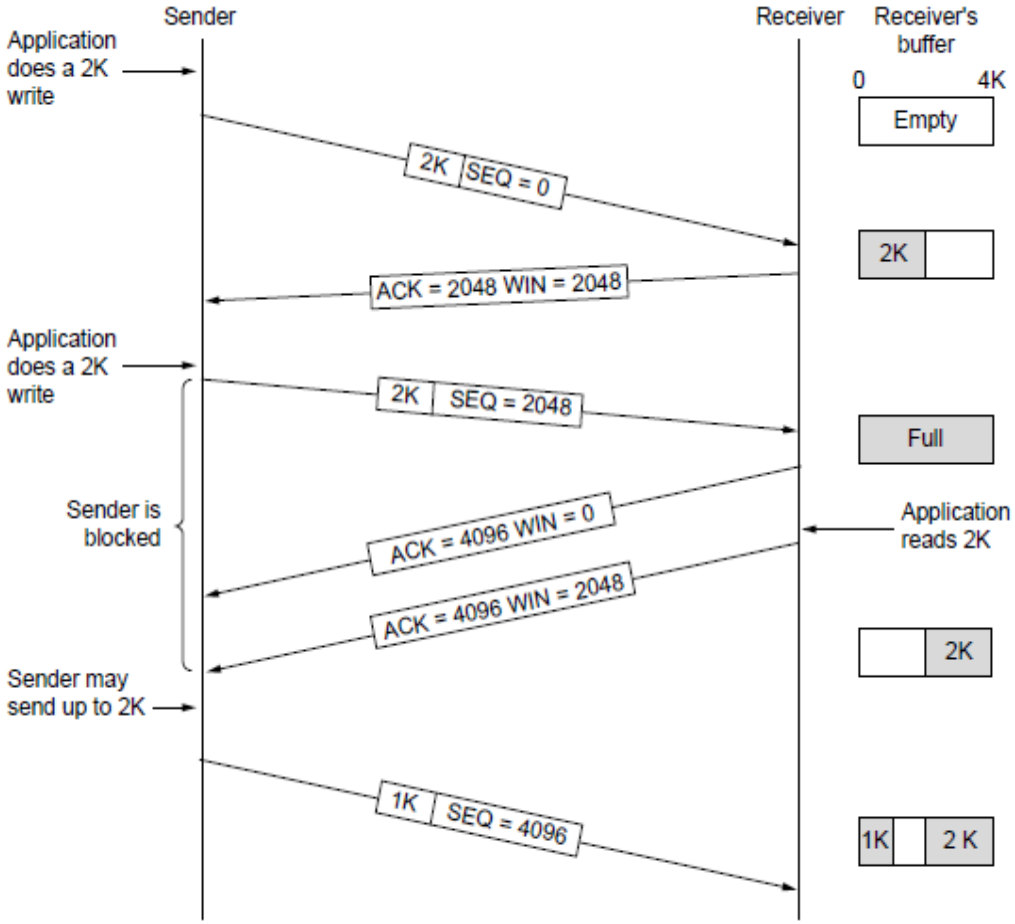
1. El búfer del receptor está vacío.
2. El búfer del receptor tiene 2KB
3. El búfer del receptor tiene 4KB (lleno)
4. La aplicación del receptor lee 2KB
5. El búfer del receptor tiene 3KB



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.

Control de flujo en TCP

- Cuando la ventana es de 0, el emisor no puede enviar segmentos, salvo en dos situaciones:
  1. pueden enviarse **datos urgentes** (p.ej. Para que el usuario elimine el proceso en ejecución en la máquina remota),
  2. el emisor puede enviar un segmento de 1 B para hacer que el receptor ***re-anuncie*** el siguiente byte esperado y el tamaño de la ventana.
     - TCP proporciona esta opción para evitar un bloqueo irreversible si llega a perderse un anuncio de ventana.

Control de flujo en TCP

- **Situación**: En las líneas con alto ancho de banda, alto retardo o ambas cosas, la ventana de 64 KB con frecuencia es un problema.
  - **Ejemplo**: En una línea T3 (44.736 Mbps) se requieren solo 12 mseg para enviar una ventana completa de 64 KB.
    - Si el retardo de propagación de ida y vuelta es de 50 mseg (típico de una fibra transcontinental), el emisor estará inactivo ¾ del tiempo en espera de confirmaciones de recepción.
  - **Ejemplo**: En una conexión satelital la situación es peor aun.
- **Problema**: Un tamaño de ventana más grande permitirá al emisor continuar enviando datos, pero como el campo de tamaño de ventana es de 16 bits, es imposible expresar tal tamaño.

Control de flujo en TCP

- **Solución (opción de escala de ventana)**: permitir al emisor y al receptor negociar un factor de escala de ventana.
  - Ambos lados pueden desplazar el tamaño del campo de ventana hasta 14 bits a la izquierda,
  - permitiendo por lo tanto ventanas de hasta 2^30 bytes.
  - La mayoría de las implementaciones actuales de TCP manejan esta opción.