# Nivel de transporte: UDP y TCP

## Fundamentos de Redes de Ordenadores

Departamento de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación

Universidad Rey Juan Carlos

Diciembre 2022

# Contenidos

# Contenidos

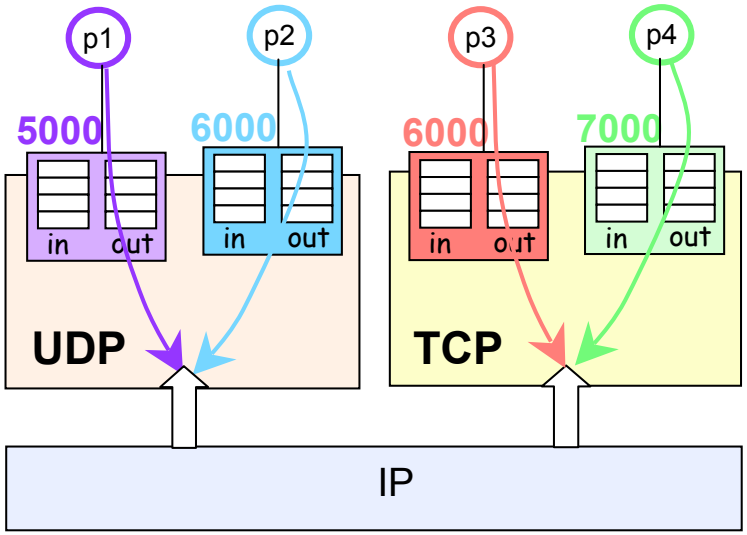
# Introducción

- El **Nivel de Red** ofrece servicio *best-effort*:
  - Se pueden perder mensajes
  - Se pueden desordenar mensajes
  - Se pueden duplicar mensajes
- El **Nivel de Transporte** se encarga de gobernar el acceso múltiple a la red de los diversos procesos de la misma máquina que quieran usarla: En TCP/IP se hace a través de los puertos.
- Hay dos protocolos principales que ofrecen un servicio de nivel de transporte:
  - **UDP**: no orientado a conexión y no fiable.
  - **TCP**: orientado a conexión y fiable.

# Multiplexación, demultiplexación

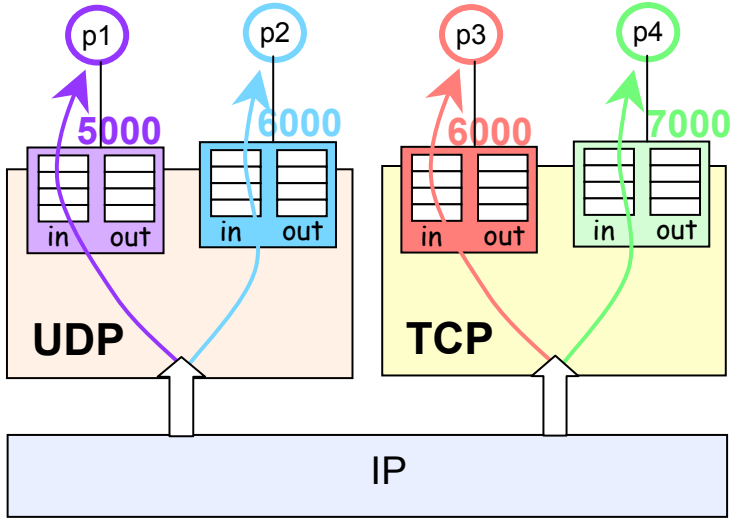
- El Nivel de Transporte TCP/IP:
  - multiplexa las unidades de datos que envían las aplicaciones a través de los puertos, encapsulándolas en unidades de datos de UDP o TCP
  - demultiplexa las unidades de datos de UDP y TCP, pasando los datos a las aplicaciones.

**Multiplexación**




Envío de datos en el nivel de transporte

**Demultiplexación**



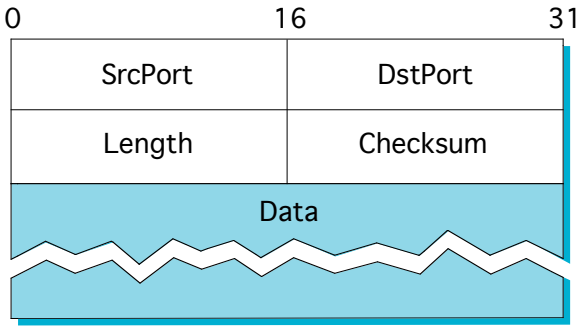

Recepción de datos en el nivel de transporte

# Contenidos

# UDP

- Servicio de entrega de datagramas no fiable y no ordenado
- Proporciona multiplexación
- No proporciona control de flujo
- Los extremos se identifican mediante puertos
- Checksum opcional de todo el datagrama UDP. Si se utiliza y no se pasa la comprobación, se descarta el datagrama.

Formato del datagrama UDP



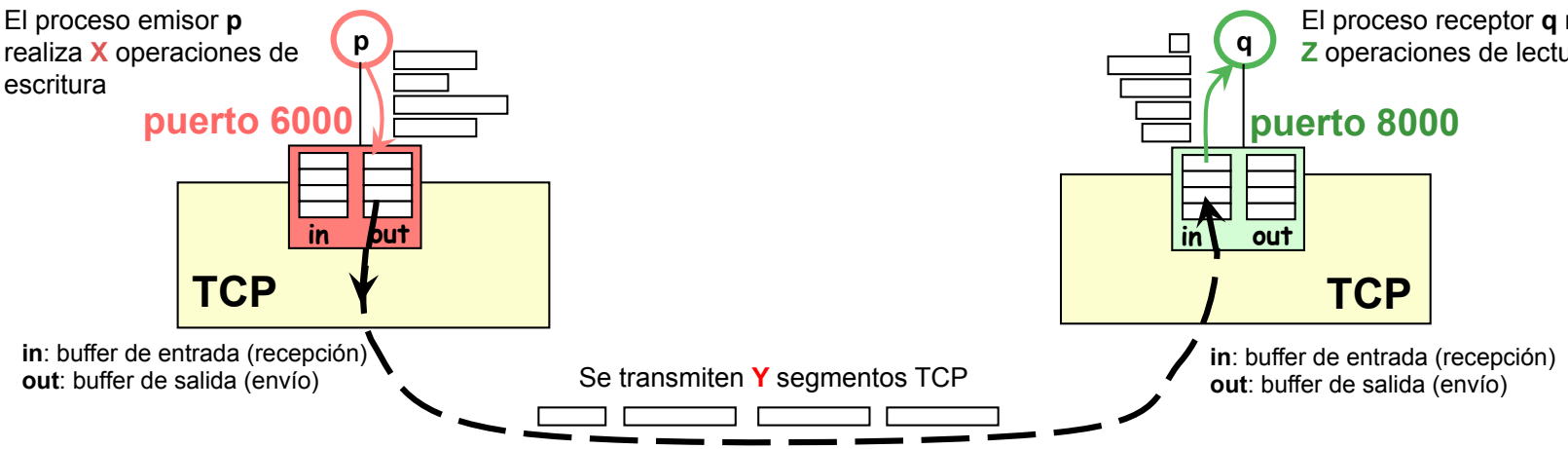
# UDP

- Los datagramas UDP se encapsulan dentro de la parte de datos de un datagrama IP.
- Una aplicación que utilice UDP para transmitir datos, producirá exactamente un datagrama UDP cada vez que la aplicación quiera enviar datos. Dicho datagrama UDP se encapsulará en un datagrama IP. Si ese datagrama IP va a exceder el tamaño máximo de la unidad de datos del nivel de enlace (ej: Trama Ethernet), se fragmentará.
- Es un protocolo mucho más ligero que TCP y para aplicaciones de red que se ejecuten dentro de una subred (no hay encaminadores, luego las pérdidas son poco probables), puede compensar.

# Contenidos

# Características del protocolo TCP

- Orientado a conexión: Fases de establecimiento de conexión, intercambio de datos y cierre de la conexión.
- Envío de datos de forma fiable: el proceso receptor recibe los datos sin pérdidas, duplicados ni desorden
- Envío de datos como flujo de bytes:
  - El proceso emisor escribe un flujo de bytes.
  - TCP envía segmentos del tamaño que considera adecuado.
  - El proceso receptor lee un flujo de bytes.




- Las conexiones son *full duplex*: ambos lados pueden enviar datos simultáneamente.

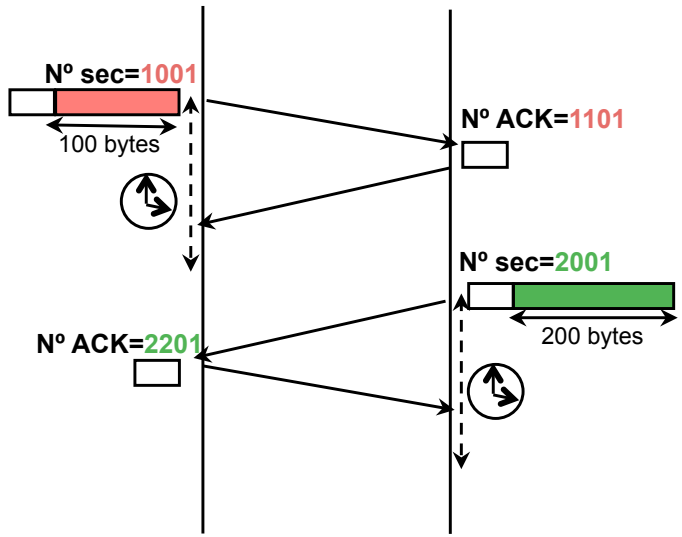
# Servicio Orientado a Conexión

- La transmisión de datos en una conexión TCP presenta las fases:
  - establecimiento de la conexión
  - intercambio de datos
  - finalización de la conexión.
- Una vez establecida la conexión, funciona en ambos sentidos: ambos extremos pueden transmitir y recibir datos simultáneamente.

# Servicio Fiable

- Éste es el primer nivel (de abajo a arriba) en TCP/IP en el que se proporciona fiabilidad.
- Objeto: Recuperarse de las posibles pérdidas y desorden producido por IP.

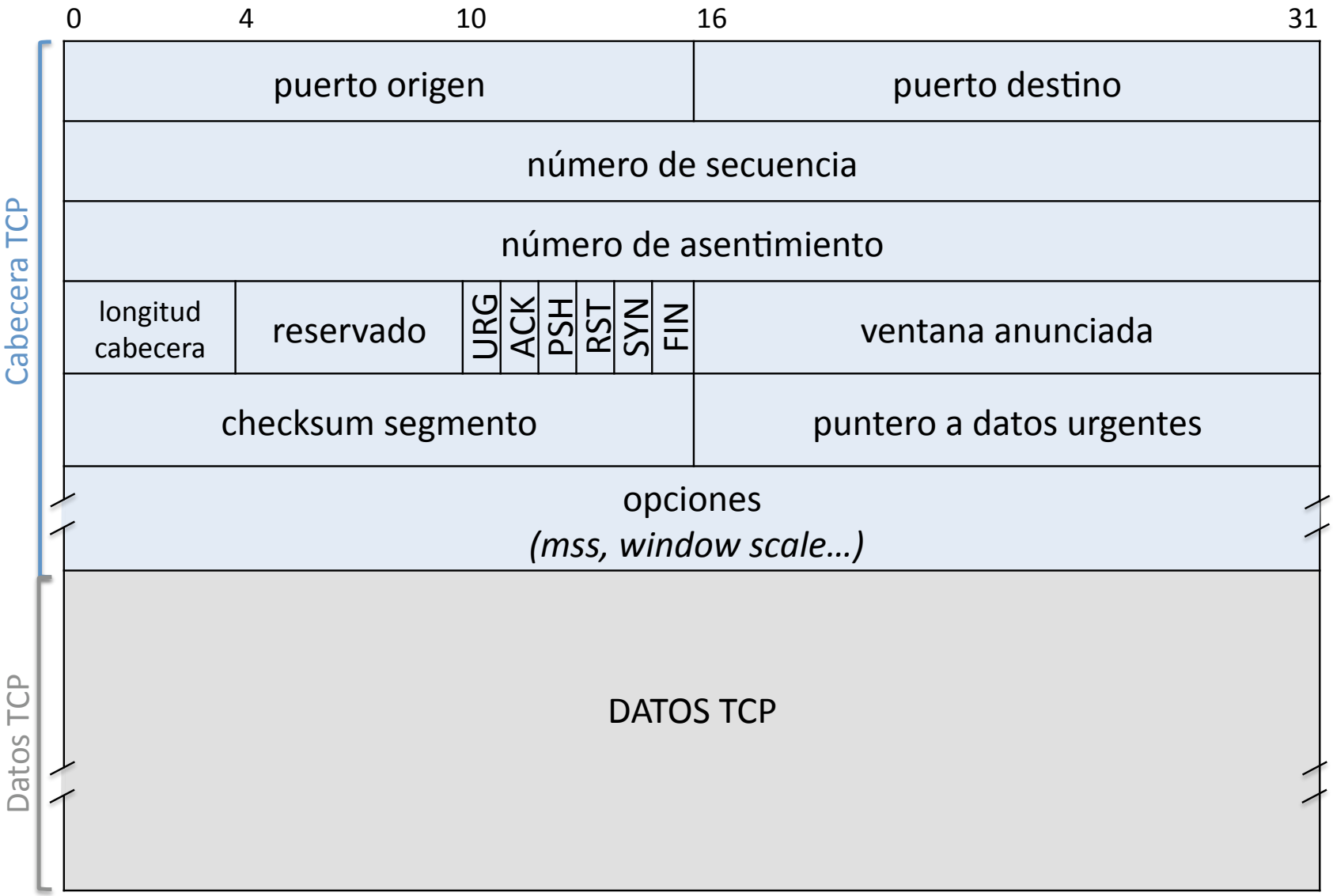
- Idea básica:
  - Los segmentos con datos llevan un número de secuencia.
  - El receptor de los datos debe mandar asentimientos (ACKs).
  - Para cada segmento con datos transmitido se espera un plazo de tiempo a que llegue su ACK. Si vence el plazo y no se ha recibido su ACK se retransmite el segmento.
  - Para asentimientos y retransmisiones se utiliza un protocolo de ventana.
  - El receptor reordena segmentos y descarta duplicados.




- Como ambos extremos pueden transmitir datos, cada lado usa sus propios números de secuencia.

# Contenidos

# TCP: formato de segmento



# TCP: formato de segmento

- **Puertos**: TCP los asocia con la aplicación origen y destino del segmento (como UDP).
  - Cada conexión TCP se identifica con una 4-tupla: `(IP_Origen, Puerto_Origen, IP_Destino, Puerto_Destino)`
- **Longitud cabecera**: Tamaño de la cabecera en palabras de 32 bits. La cabecera sin opciones tiene longitud 5 (20 bytes).
- **Checksum**: Obligatorio, sobre todo el segmento TCP. Si no se pasa la comprobación, se descarta el segmento.

# Flags

- **SYN**: Establecimiento de conexión
- **FIN**: Finalización de conexión
- **ACK**: Hay información en el campo número de ACK
- **RST**: Situación de error.
  - Ejemplo: Una aplicación cliente quiere establecer una conexión con una aplicación que escucha en el puerto 6000 de otra máquina, y en esa máquina no hay ninguna aplicación esperando conexiones en ese puerto 6000.
- **PSH**: El receptor debe "entregar los datos a la aplicación".
  - No sólo se entrega ese segmento, sino también todos los datos anteriores asentidos que el receptor tuviera pendientes de entregar a la aplicación.
- **URG**: Pueden enviarse datos denominados urgentes que el receptor debe pasar inmediatamente a la aplicación, lo antes posible, incluso fuera de orden.
  - Se indican mediante el empleo del flag URG: Cuando está activado, el campo puntero a datos urgentes apunta al último byte de datos urgentes del segmento.

# Número de secuencia

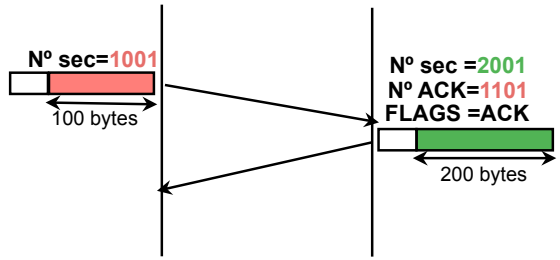
- Cada segmento con datos lleva un número de secuencia, `SequenceNum`, de 32 bits.
- El número de secuencia numera bytes, y NO segmentos: identifica el número de orden del primer byte de datos que lleva el segmento.
- Al establecerse una conexión se elige un número de secuencia inicial (ISN: *Initial Sequence Number*) de forma que:
  - No puedan confundirse segmentos aún en tránsito de una conexión antigua al abrir una nueva
  - No puedan predecirse a priori (para evitar ataques)

# Número de asentimiento (I)

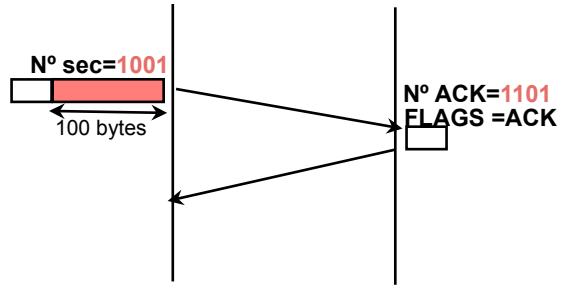
- El receptor de segmentos de datos tiene que asentir los que le llegan correctamente, activando el flag `ACK` y rellenando el campo `AcknowledgmentNum`.
- En general, se envía un asentimiento por cada segmento con datos que se recibe. A veces se espera un poco para asentir varios segmentos a la vez con un único ACK.
- El número de asentimiento indica el número de secuencia del primer byte que NO se tiene, asintiéndose de esta manera hasta el número de byte anterior incluido.
- En el funcionamiento básico de TCP no hay forma en que el receptor asiente los bytes del 300 al 700 excepto el trozo 400-500.
  - Una opción de TCP permite usar acks selectivos (SACKs) en una conexión si emisor y receptor soportan dicha opción. En este curso no estudiaremos esta opción.

# Número de asentimiento (II)

- Siempre que hay datos que enviar, se aprovecha para mandar en ese mismo segmento la información del número de asentimiento. Es decir se envían los datos y el asentimiento de los datos recibidos (*piggybacking*).




- Si el lado que ha recibido datos no tiene nada que enviar, construirá un segmento (sólo con la cabecera de TCP) donde enviará el número de asentimiento que le corresponda.
  - Importante: Cada lado de la conexión utiliza sus números de secuencia (partiendo de su ISN) y asiente los números de secuencia que está usando el otro extremo.



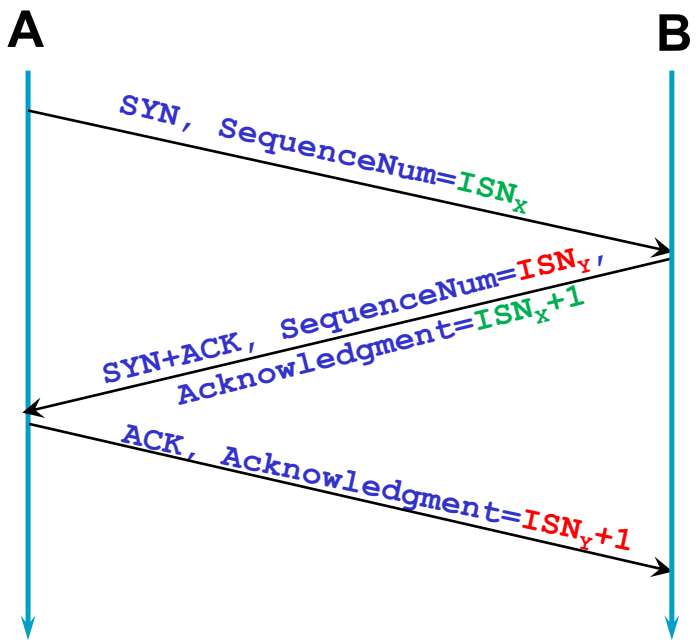
# Ventana anunciada (o ventana de flujo)

- Para coordinar el envío de segmentos de datos y asentimientos entre emisor y receptor se utiliza un protocolo de ventana. Este protocolo proporciona:
  - Fiabilidad
  - Control de flujo: El receptor puede "frenar" a un emisor demasiado rápido para él
- El receptor indica en el campo `AdvertisedWindow` (**ventana anunciada** o de flujo) la cantidad de bytes que puede recibir a partir del número de byte que aparece en el campo de número de asentimiento.
  - La ventana anunciada está directamente relacionada con el tamaño del *buffer* de recepción que tiene el receptor para almacenar los datos que vaya enviando el emisor.
- El emisor puede transmitir esa bytes aunque no reciba más asentimientos, pero una vez transmitidos tendrá que parar hasta recibir nuevos asentimientos del receptor.
  - Cuando el lado receptor envíe nuevos valores para número de ACK y ventana anunciada, el lado emisor podrá continuar con la transmisión de datos nuevos.
- Como ambos extremos pueden enviar datos, hay dos ventanas diferentes, una para cada sentido de la comunicación.

# Contenidos

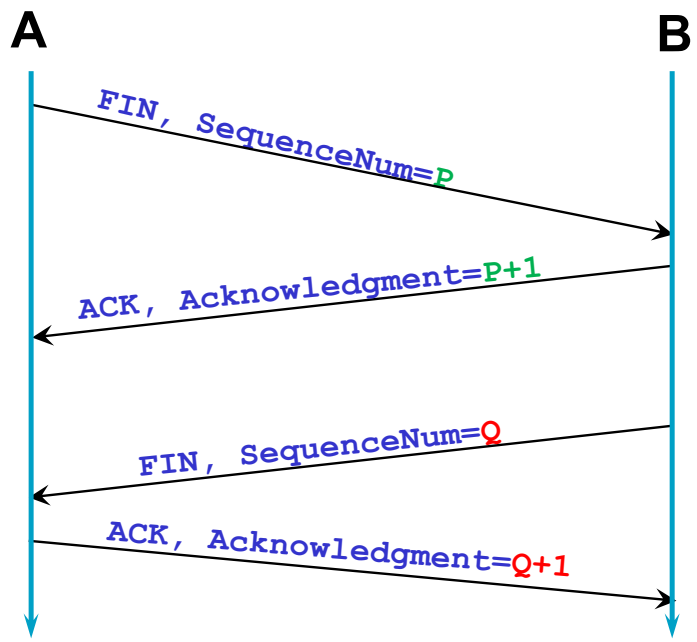
# Establecimiento y cierre de la conexión

## Establecimiento de la conexión




Se llama **cliente** al lado que envía el primer SYN, y **servidor** al otro lado

## Cierre de la conexión



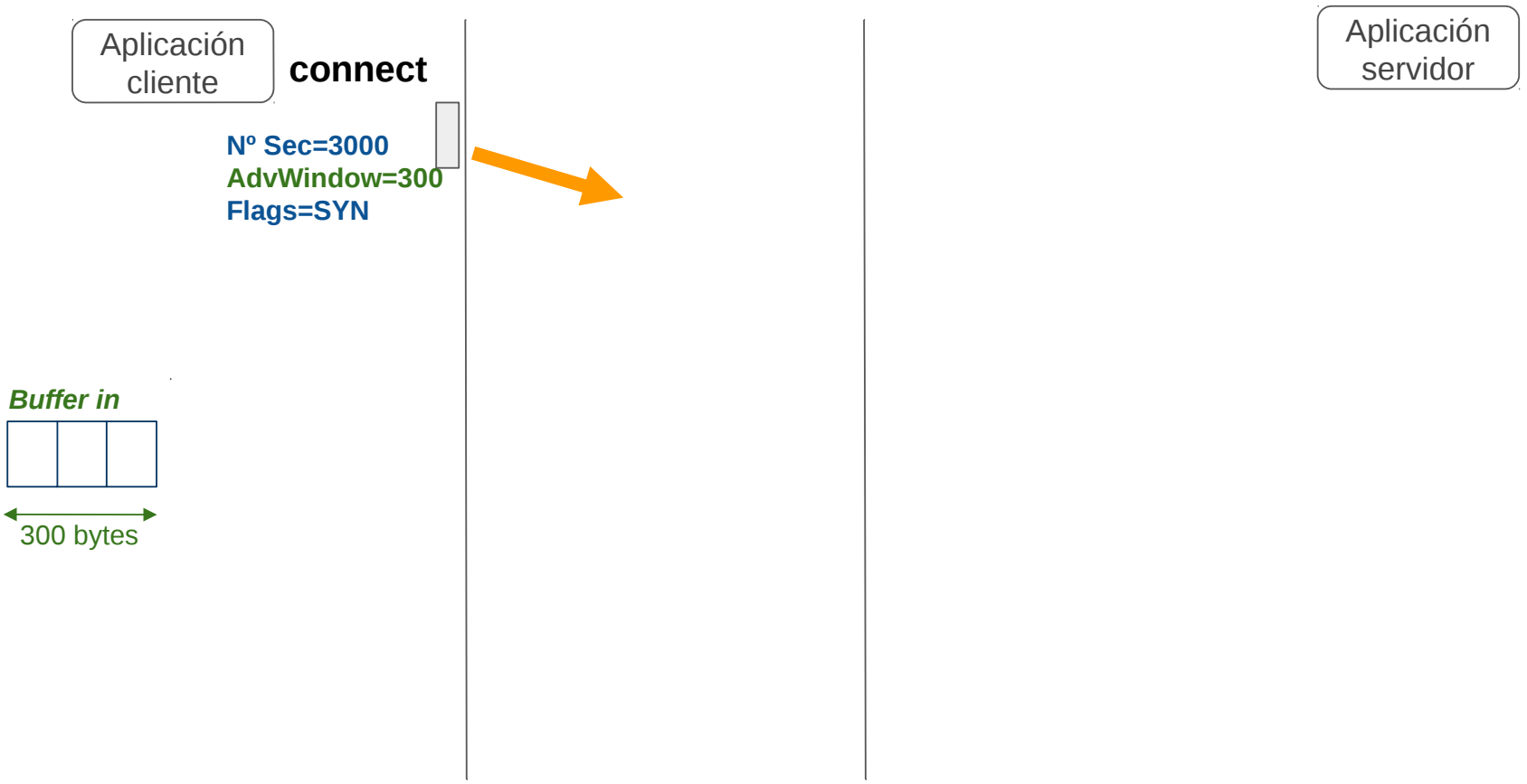

Normalmente el cliente enviará el primer FIN, pero podría ser al revés.

# Contenidos

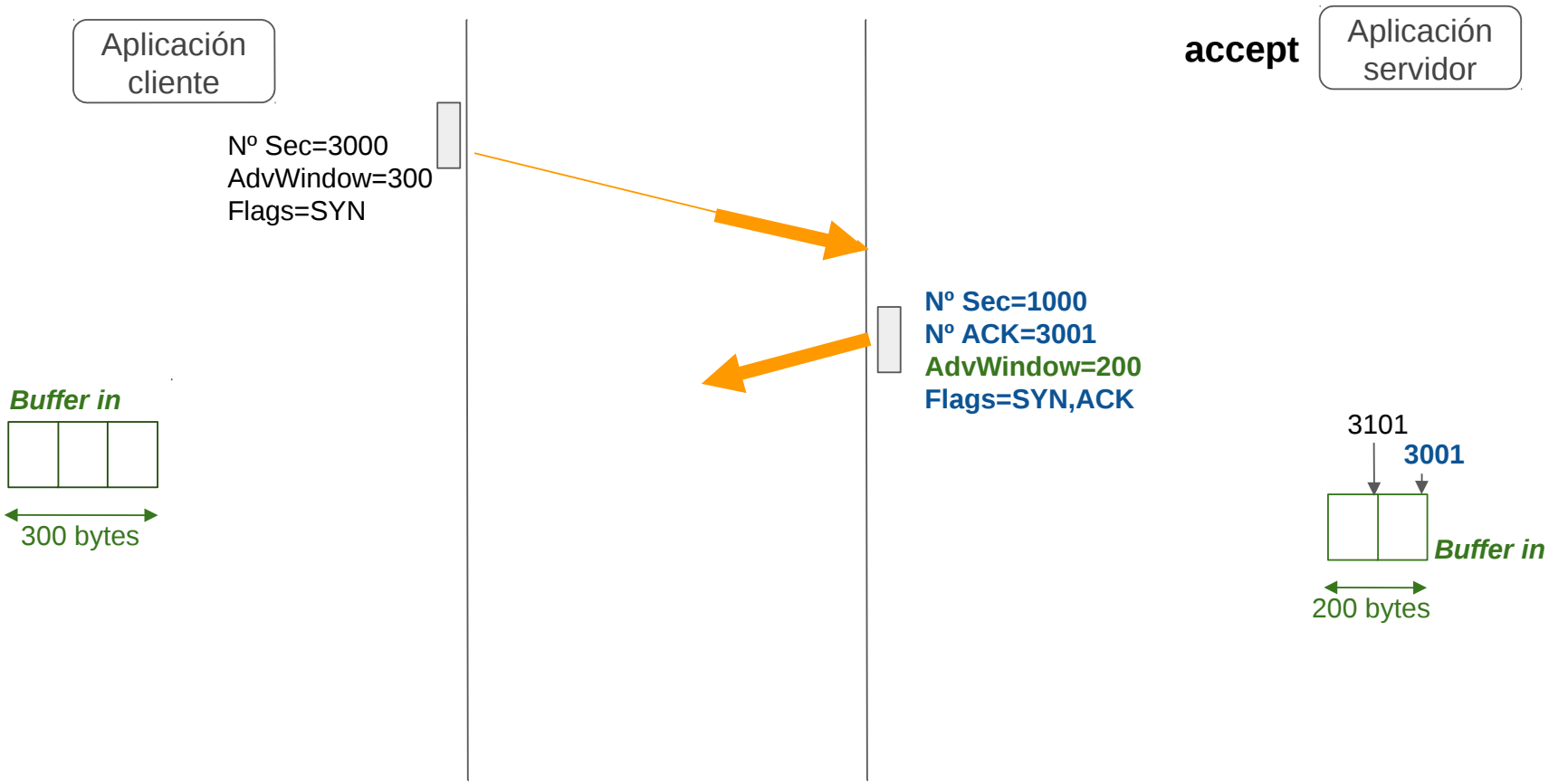
# Ejemplo: establecimiento de conexión (I)

**listen** Aplicación servidor

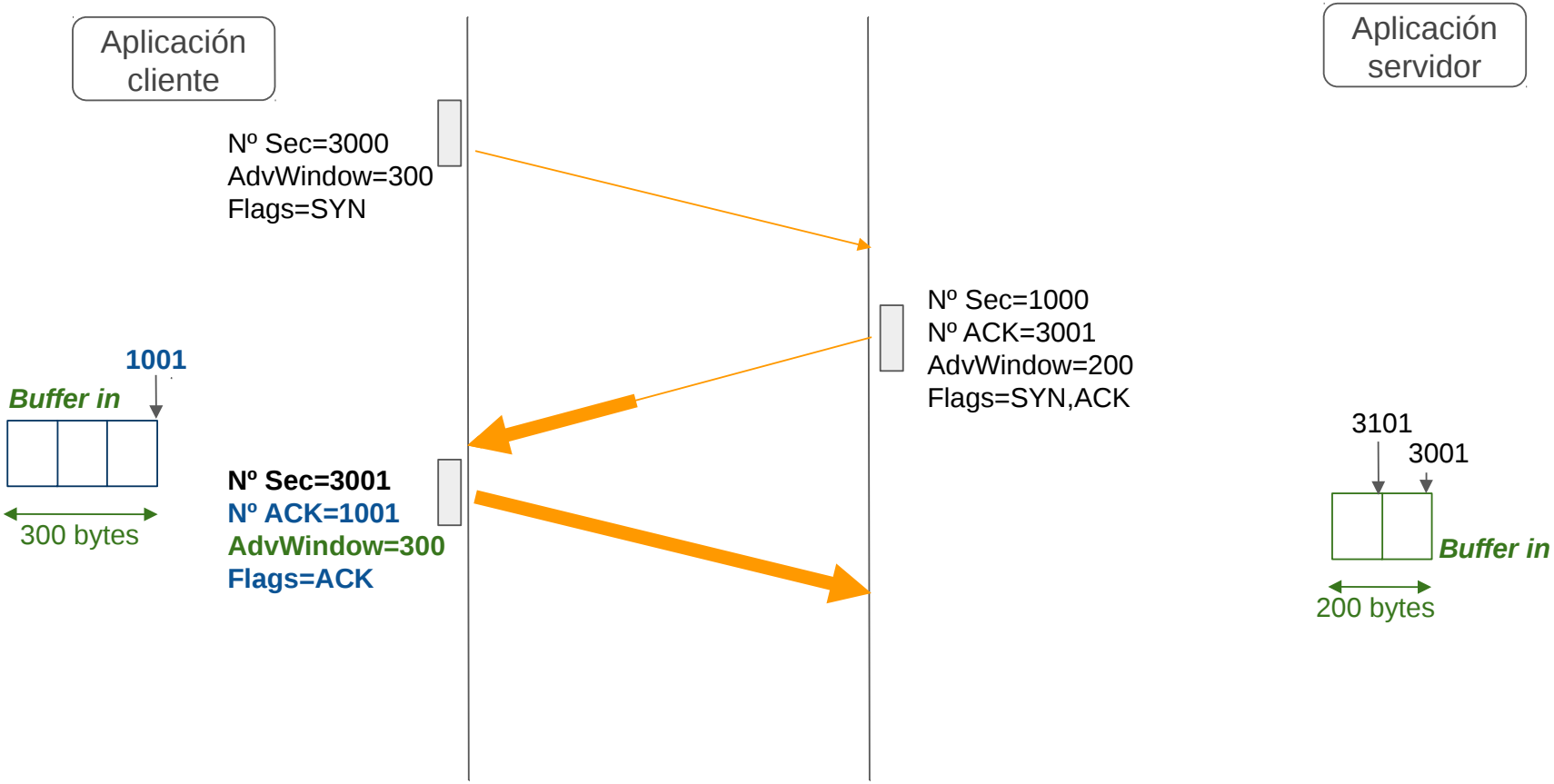
# Ejemplo: establecimiento de conexión (II)



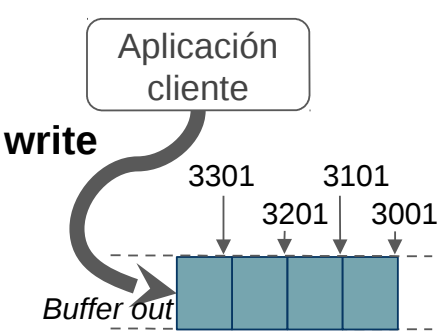
# Ejemplo: establecimiento de conexión (III)

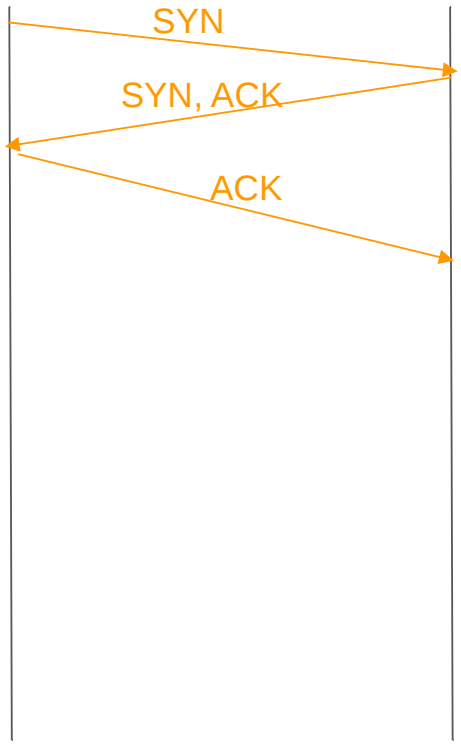


# Ejemplo: establecimiento de conexión (IV)

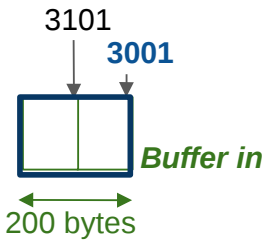


# Ejemplo: envío de datos (I)

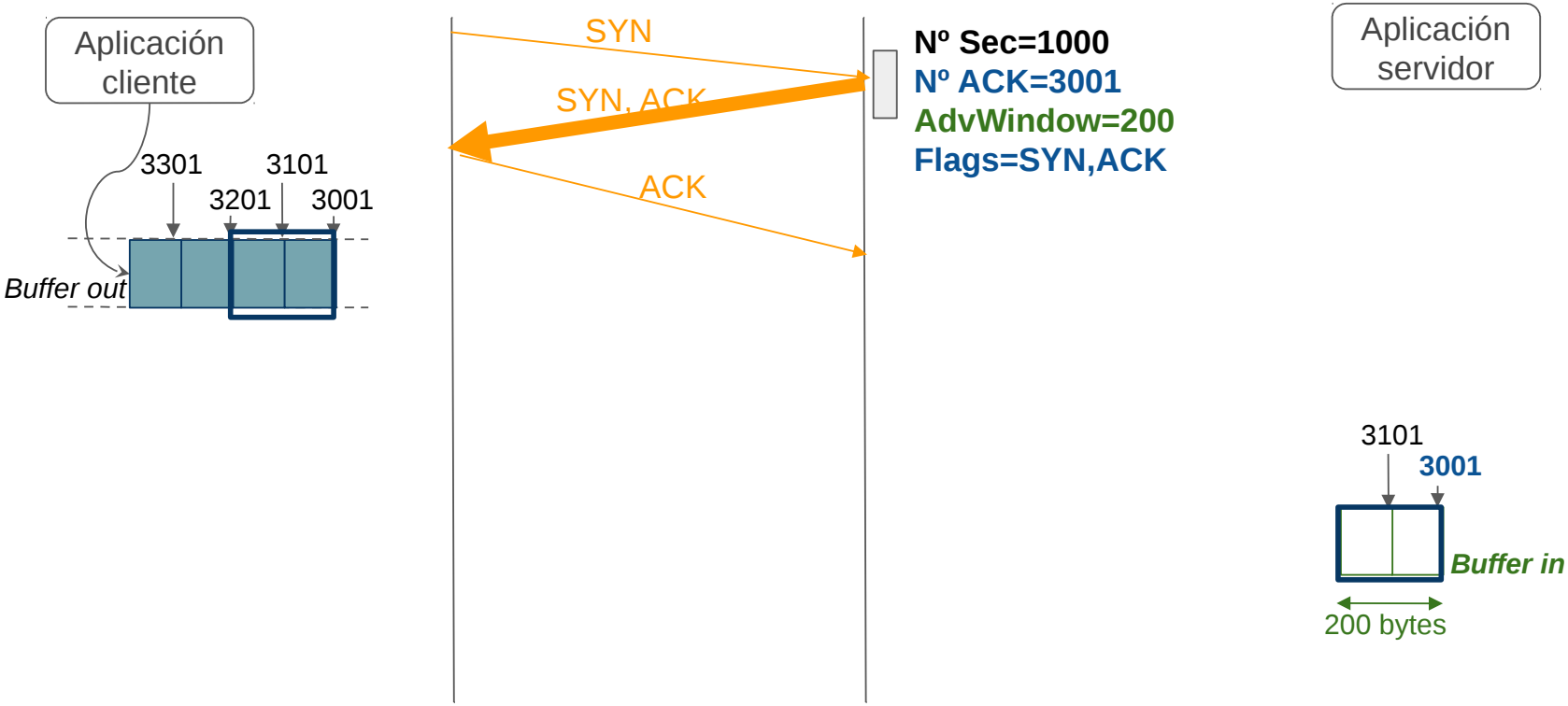


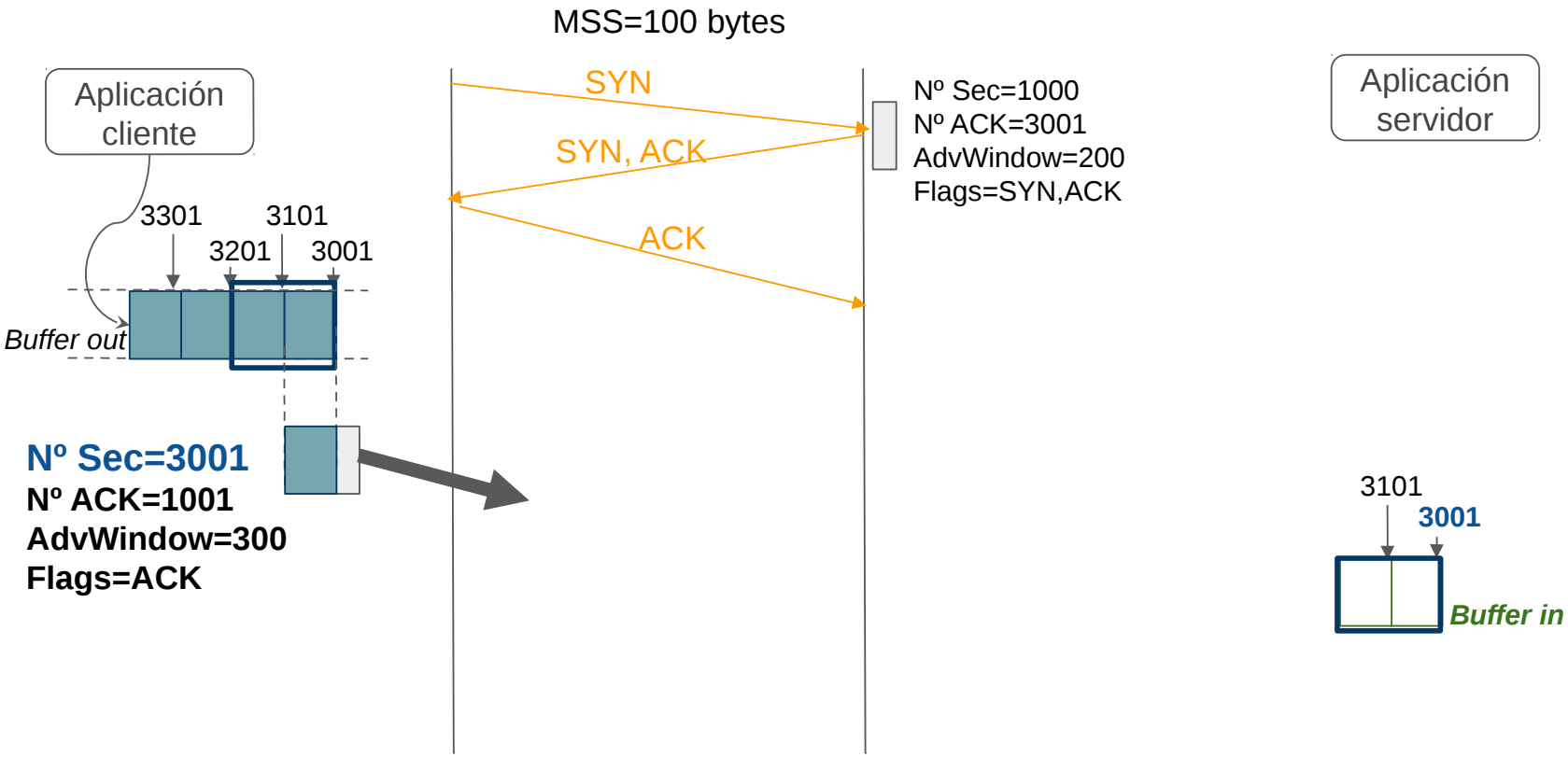

Aplicación servidor



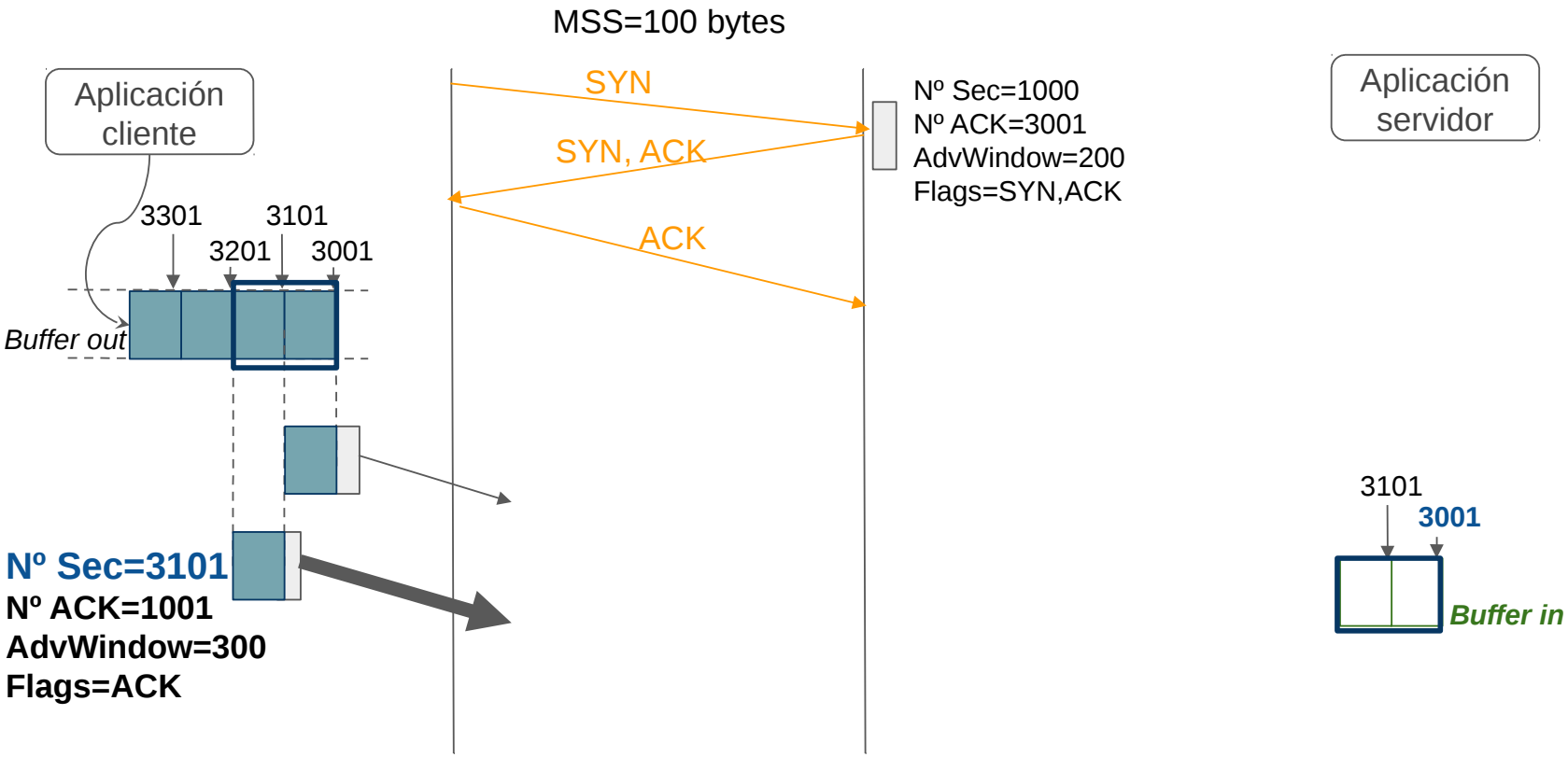
# Ejemplo: envío de datos (II)



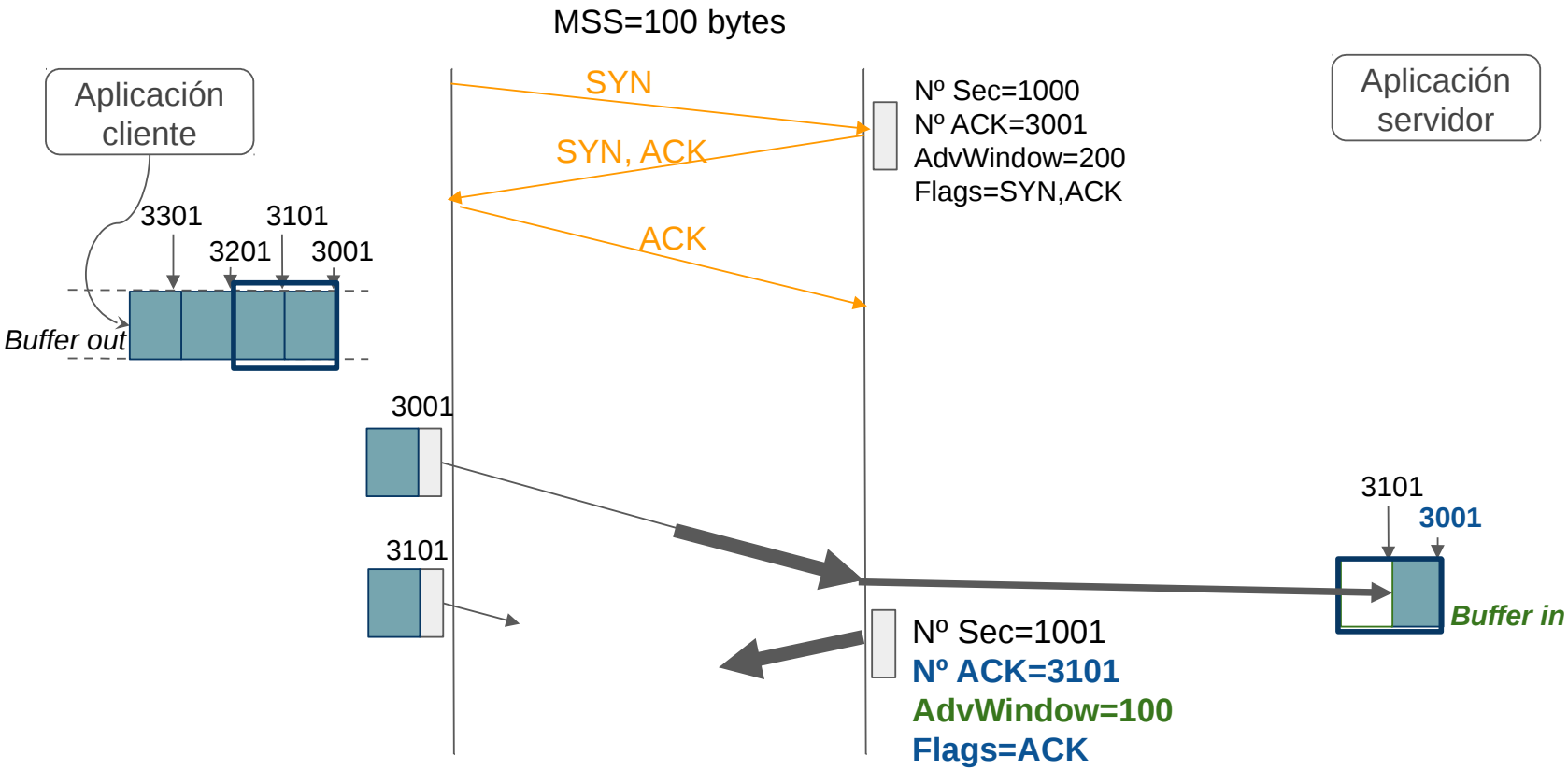
# Ejemplo: envío de datos (III)



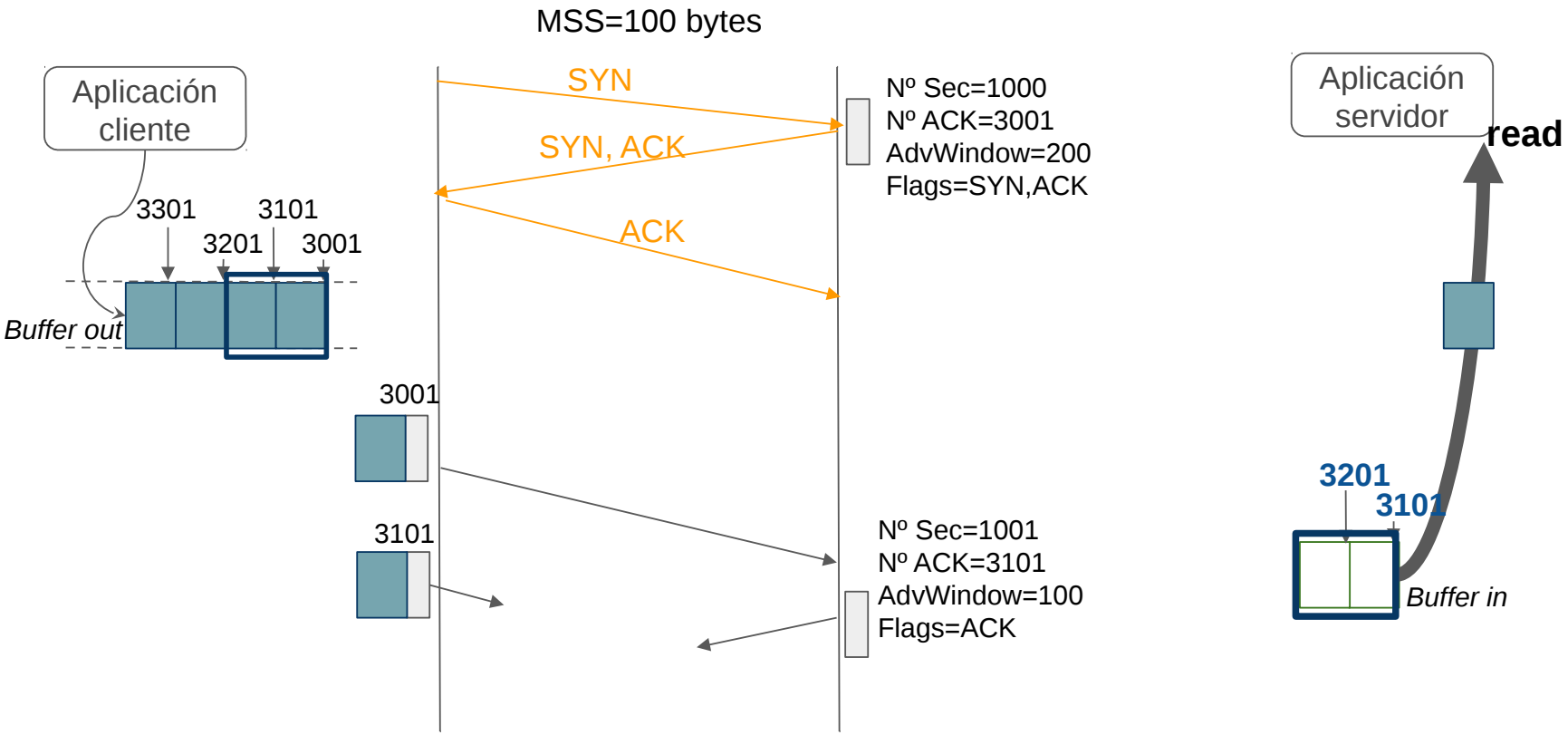
# Ejemplo: envío de datos (IV)



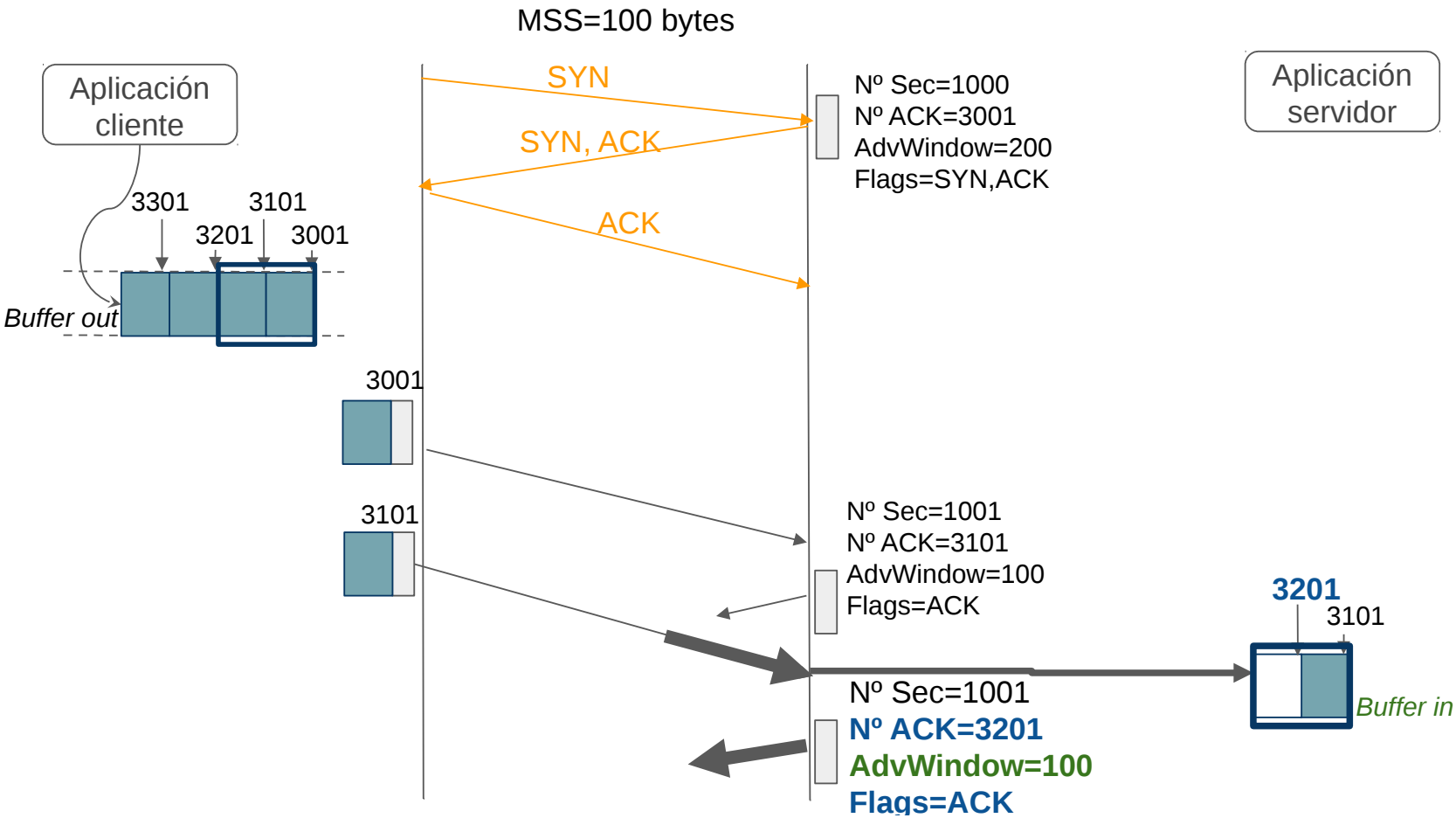
# Ejemplo: envío de datos (V)



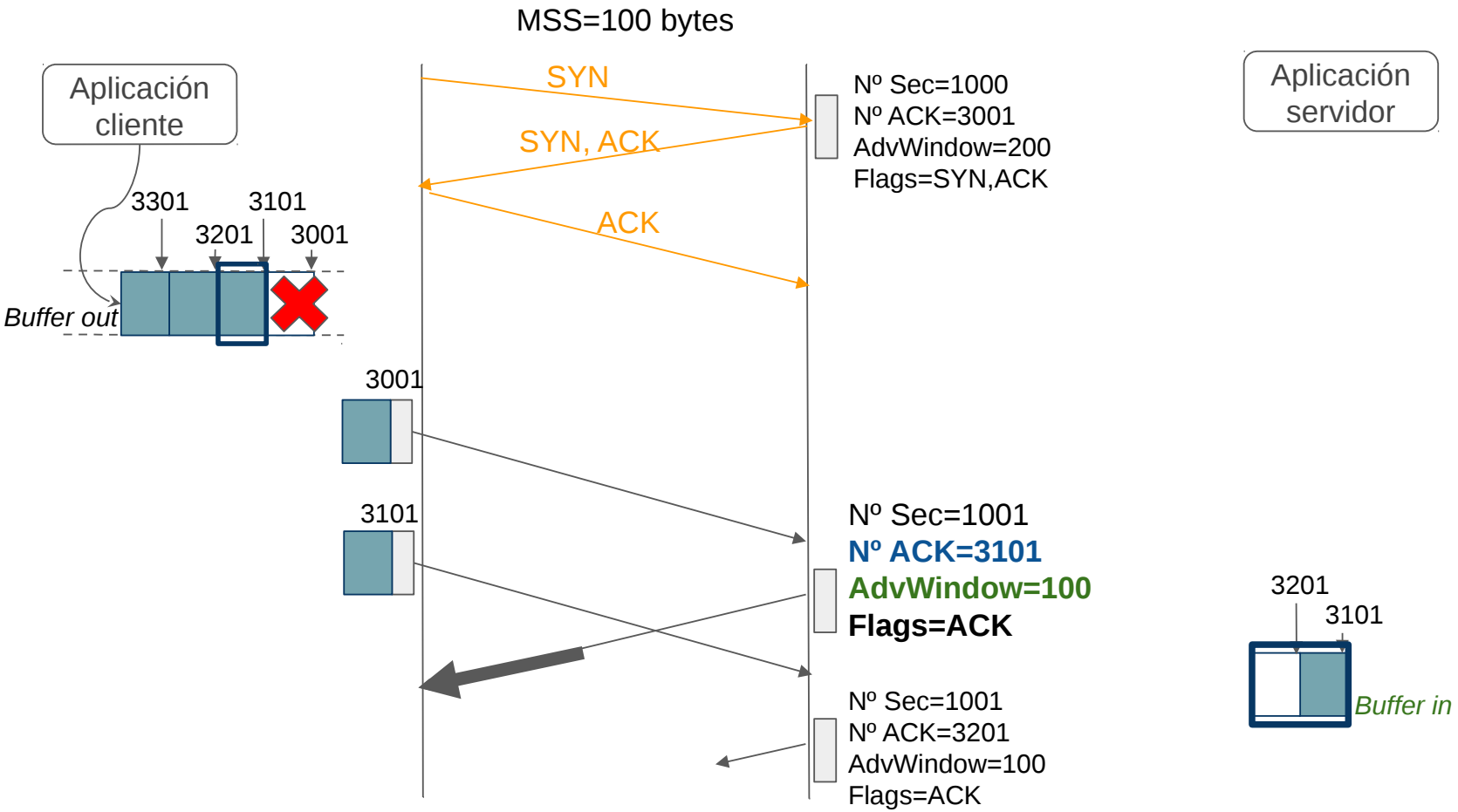
# Ejemplo: envío de datos (VI)



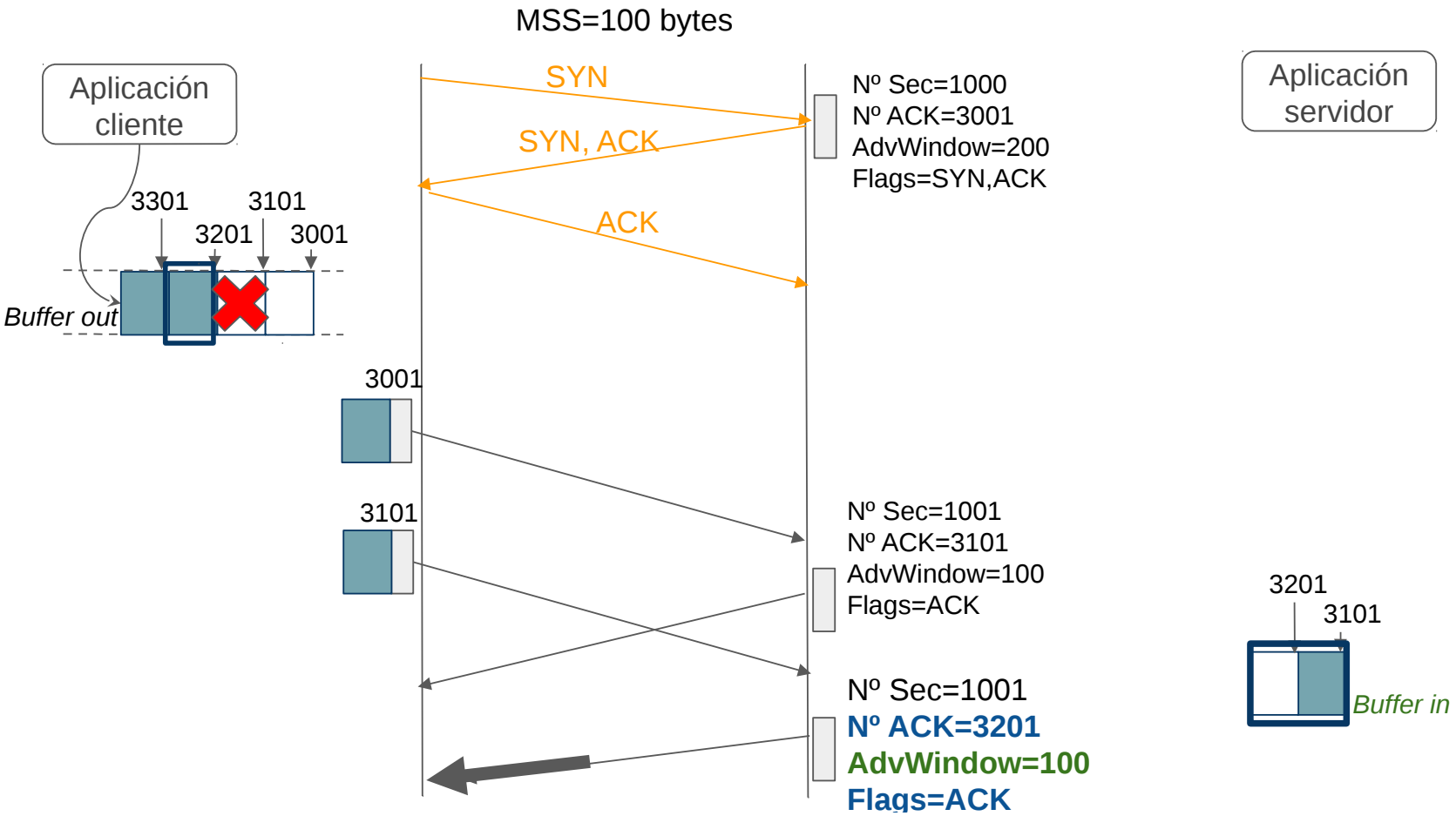
# Ejemplo: envío de datos (VII)



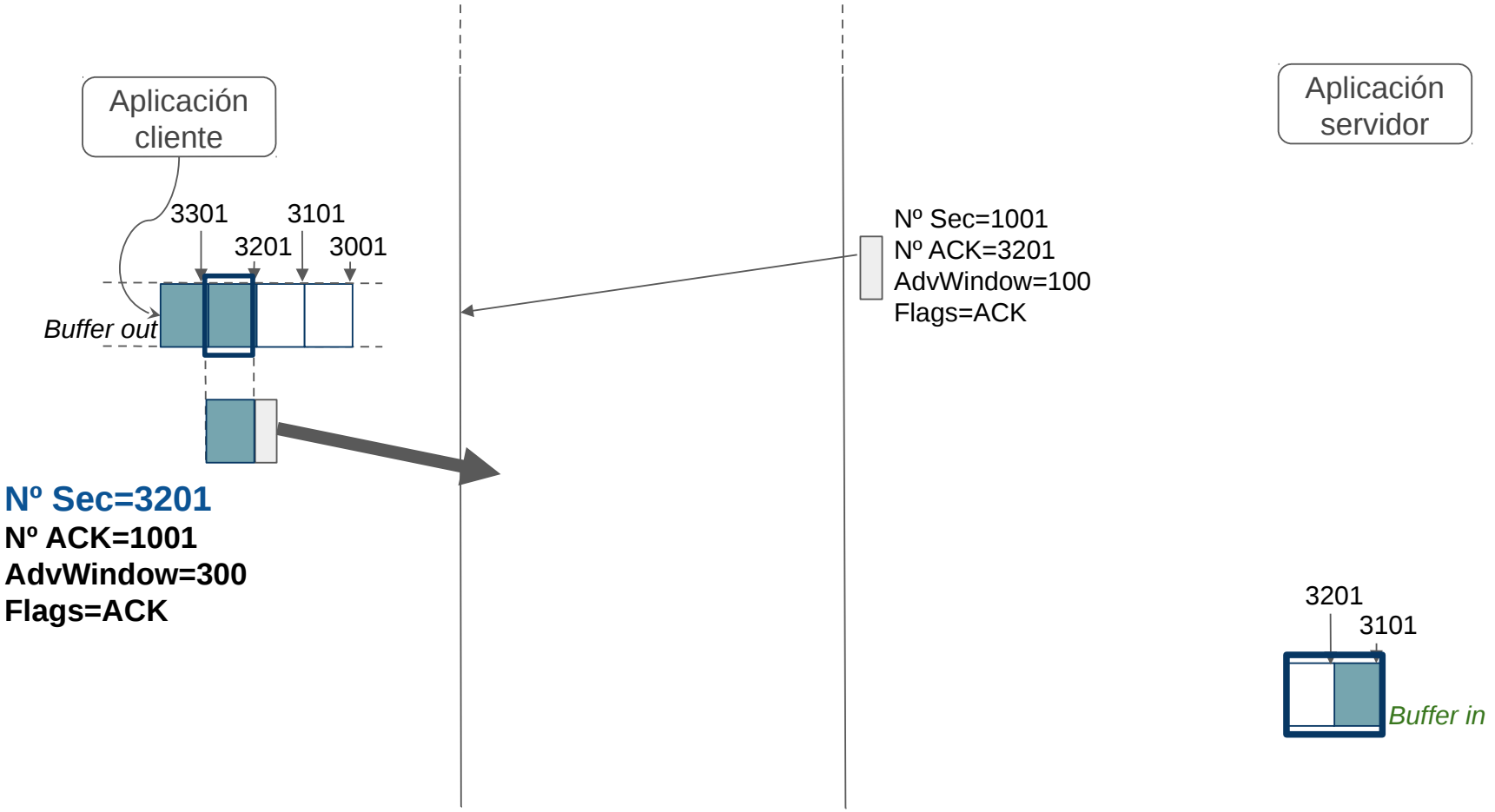
# Ejemplo: envío de datos (VIII)



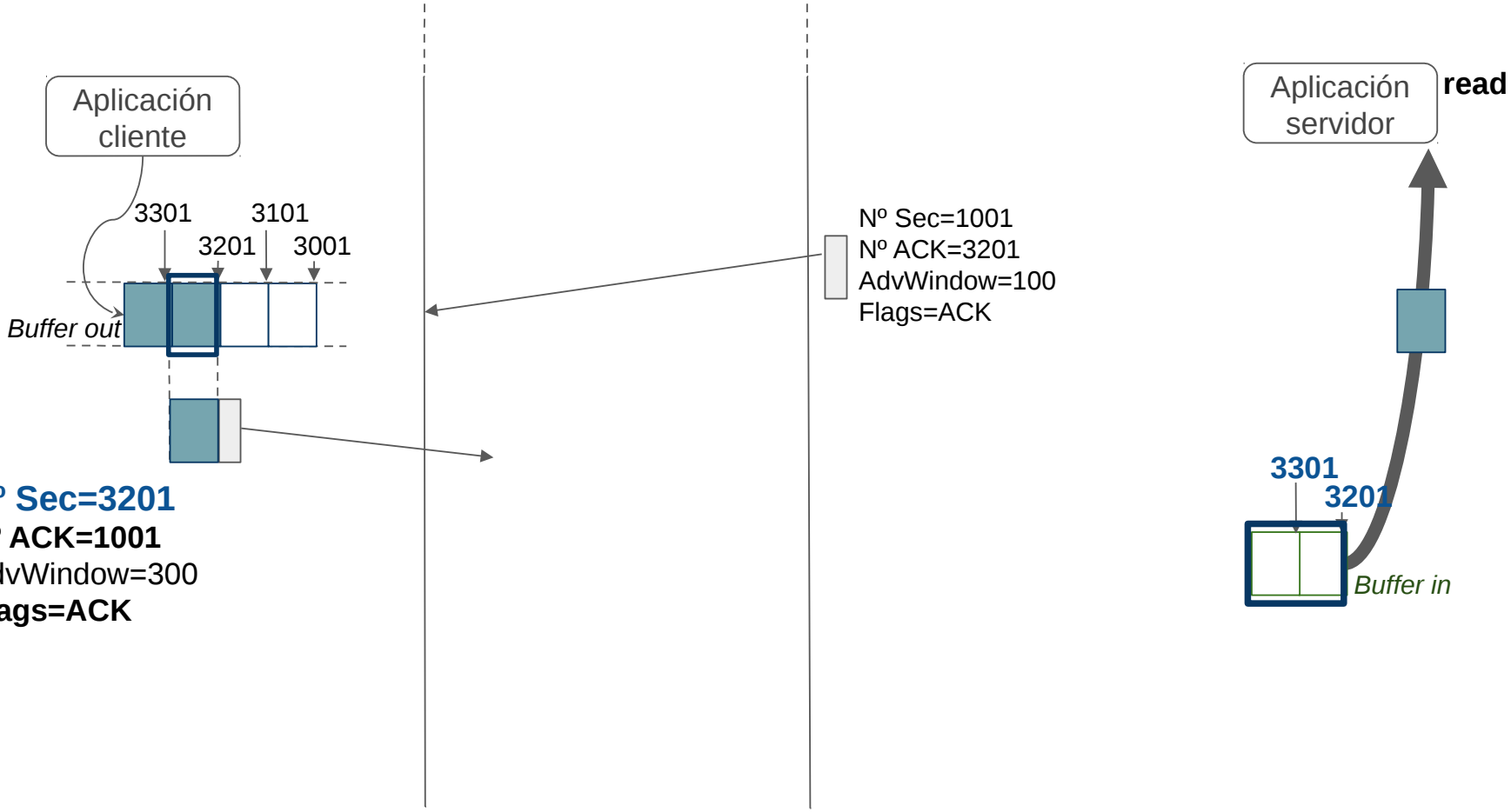
# Ejemplo: envío de datos (IX)



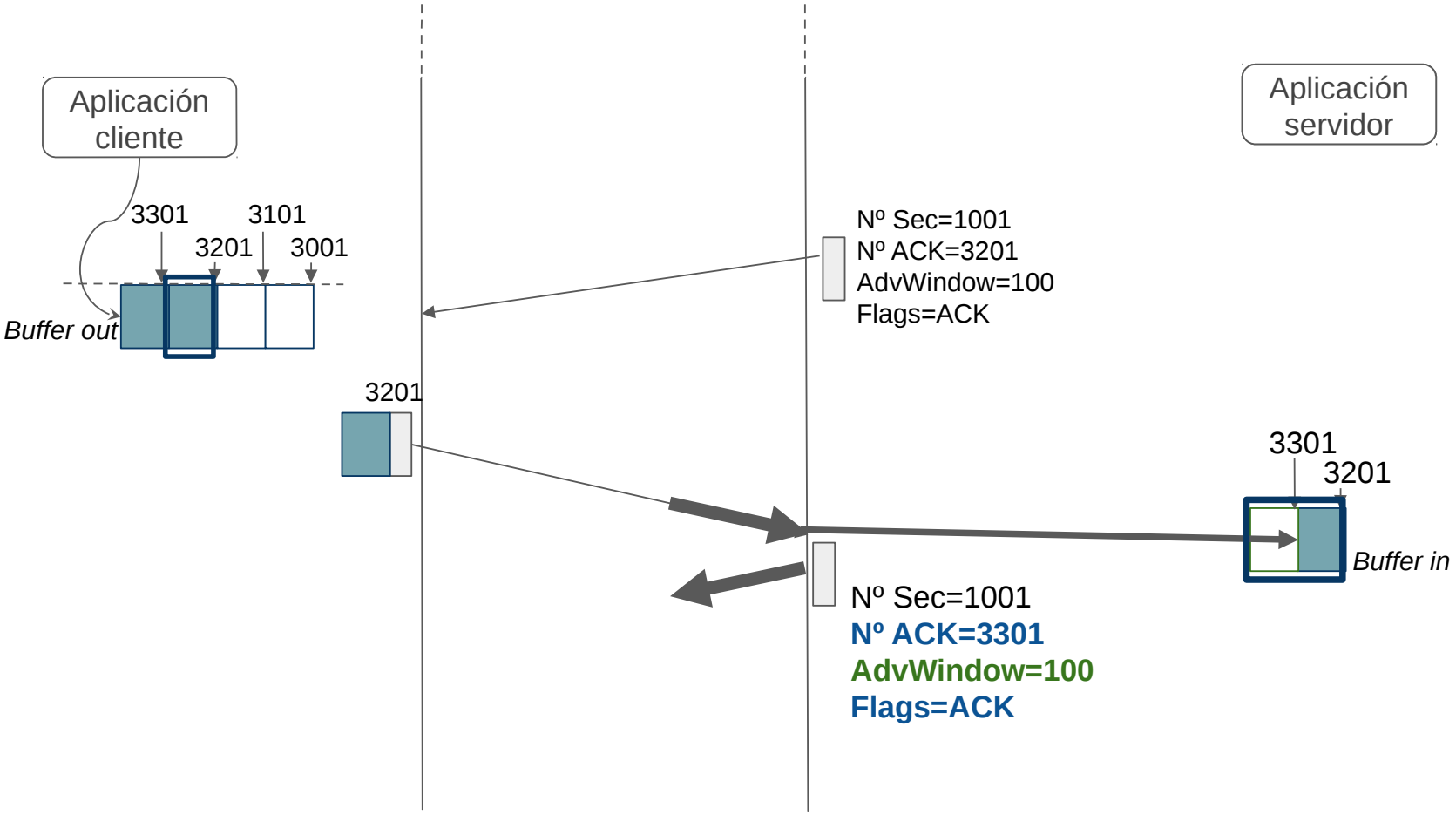
# Ejemplo: envío de datos (X)



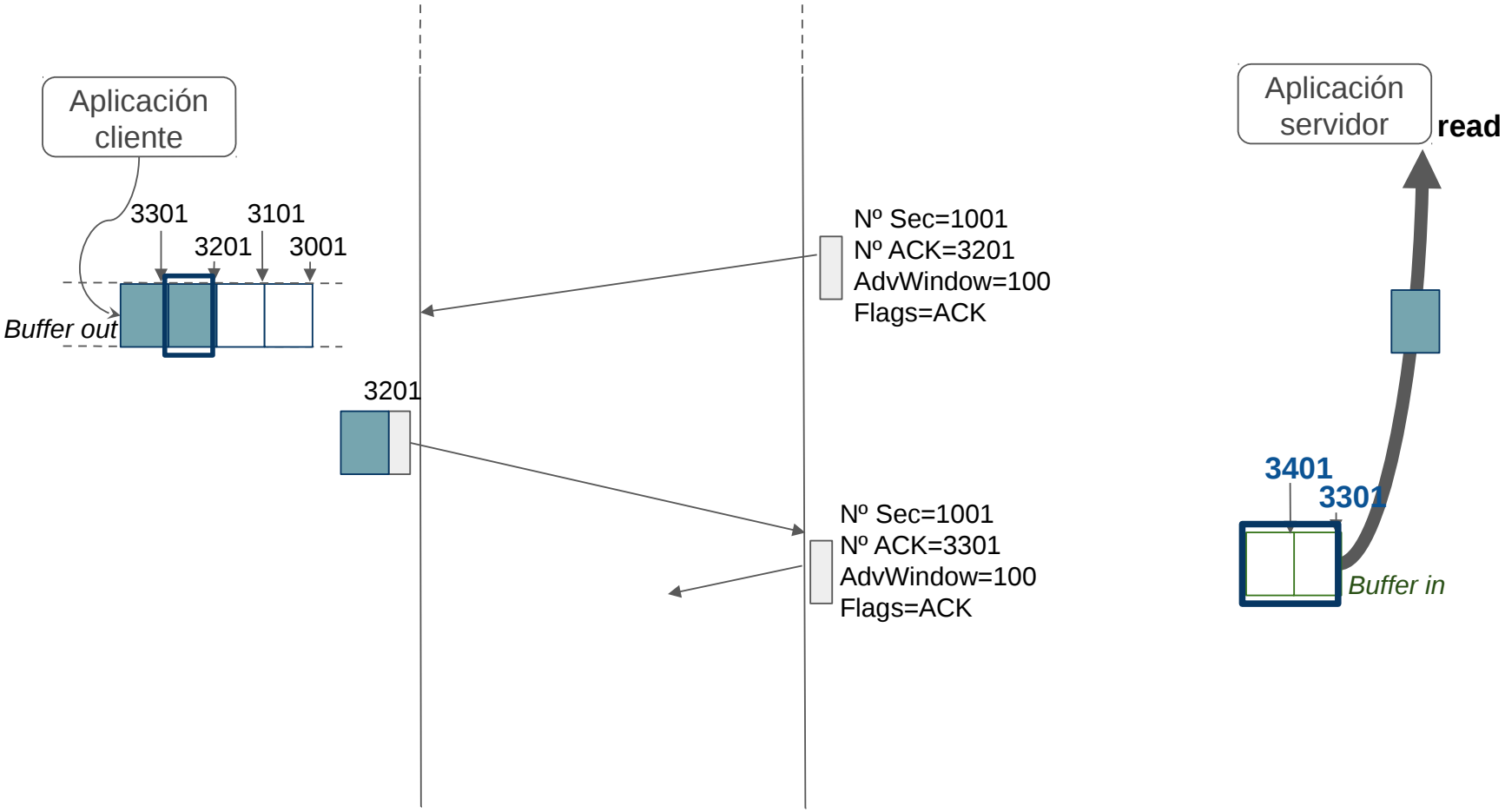
# Ejemplo: envío de datos (XI)



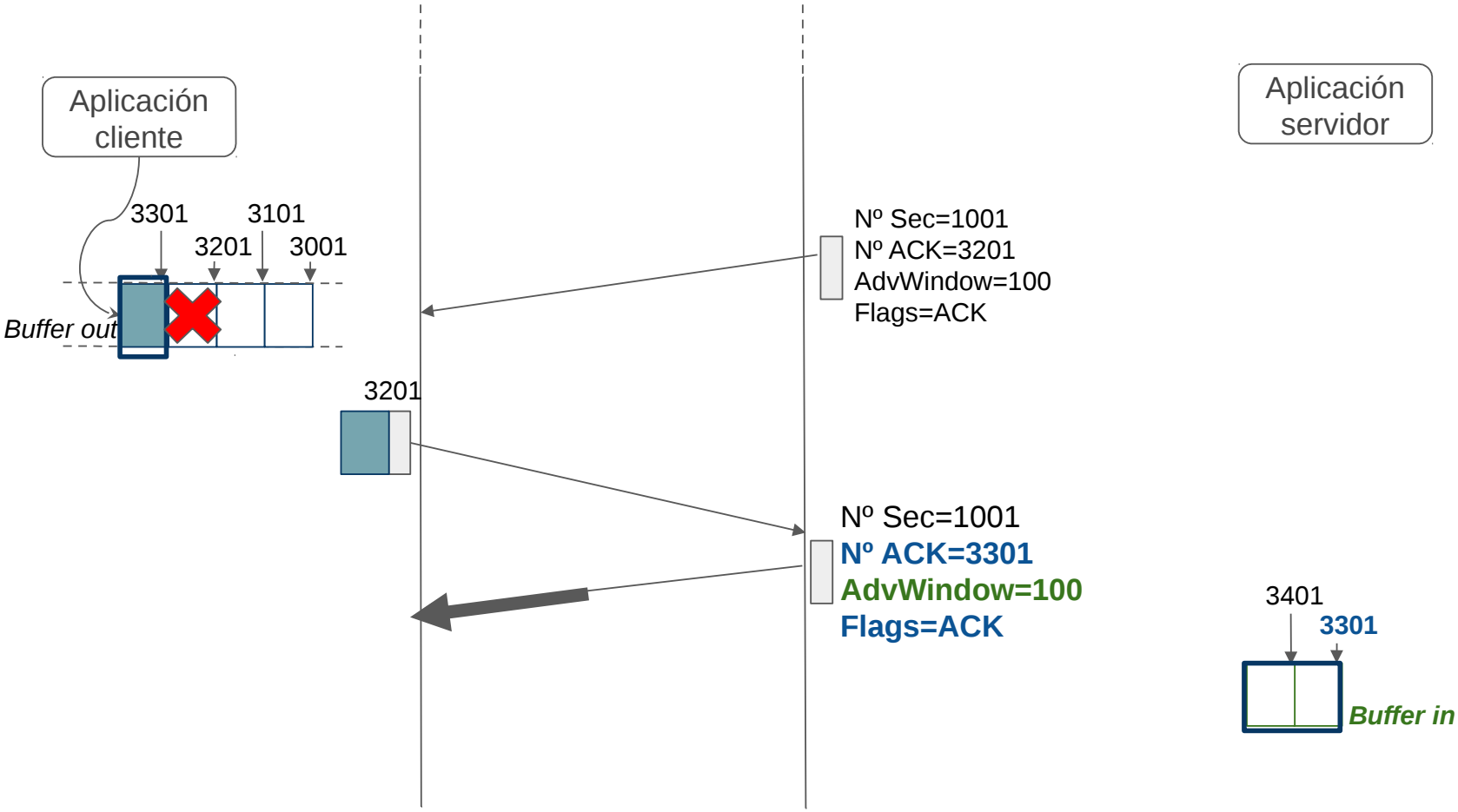
# Ejemplo: envío de datos (XII)



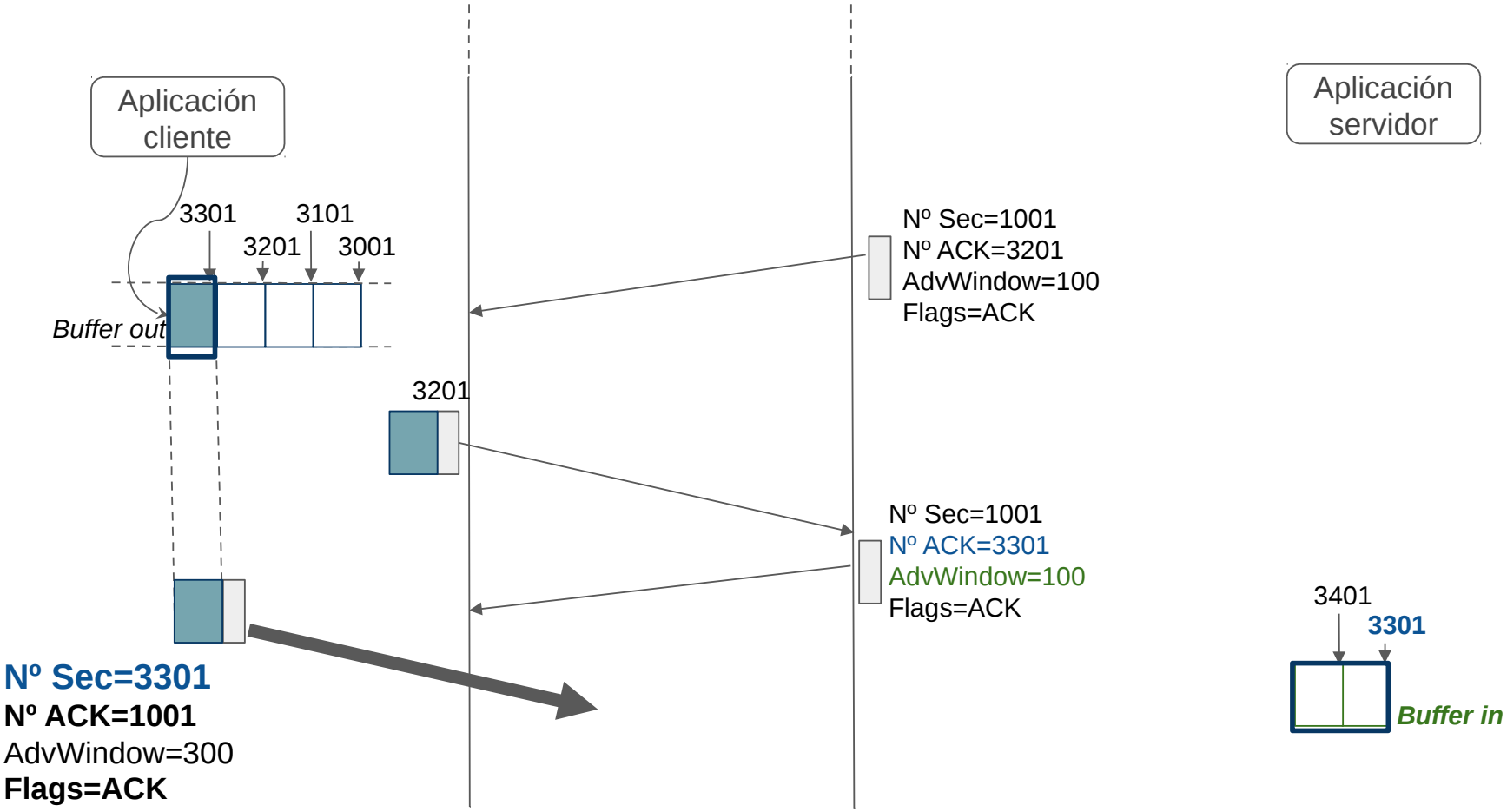
# Ejemplo: envío de datos (XIII)



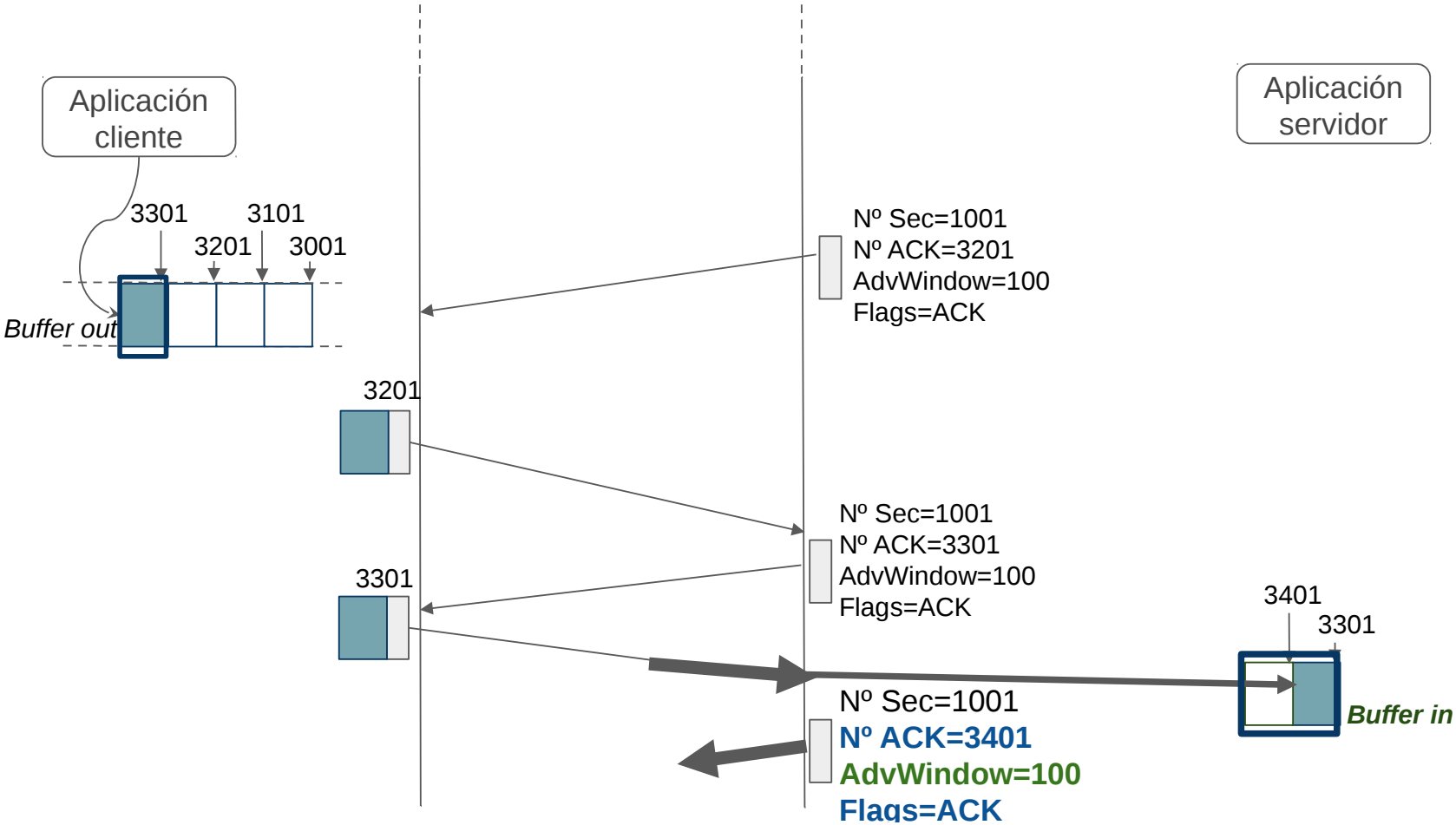
# Ejemplo: envío de datos (IX)



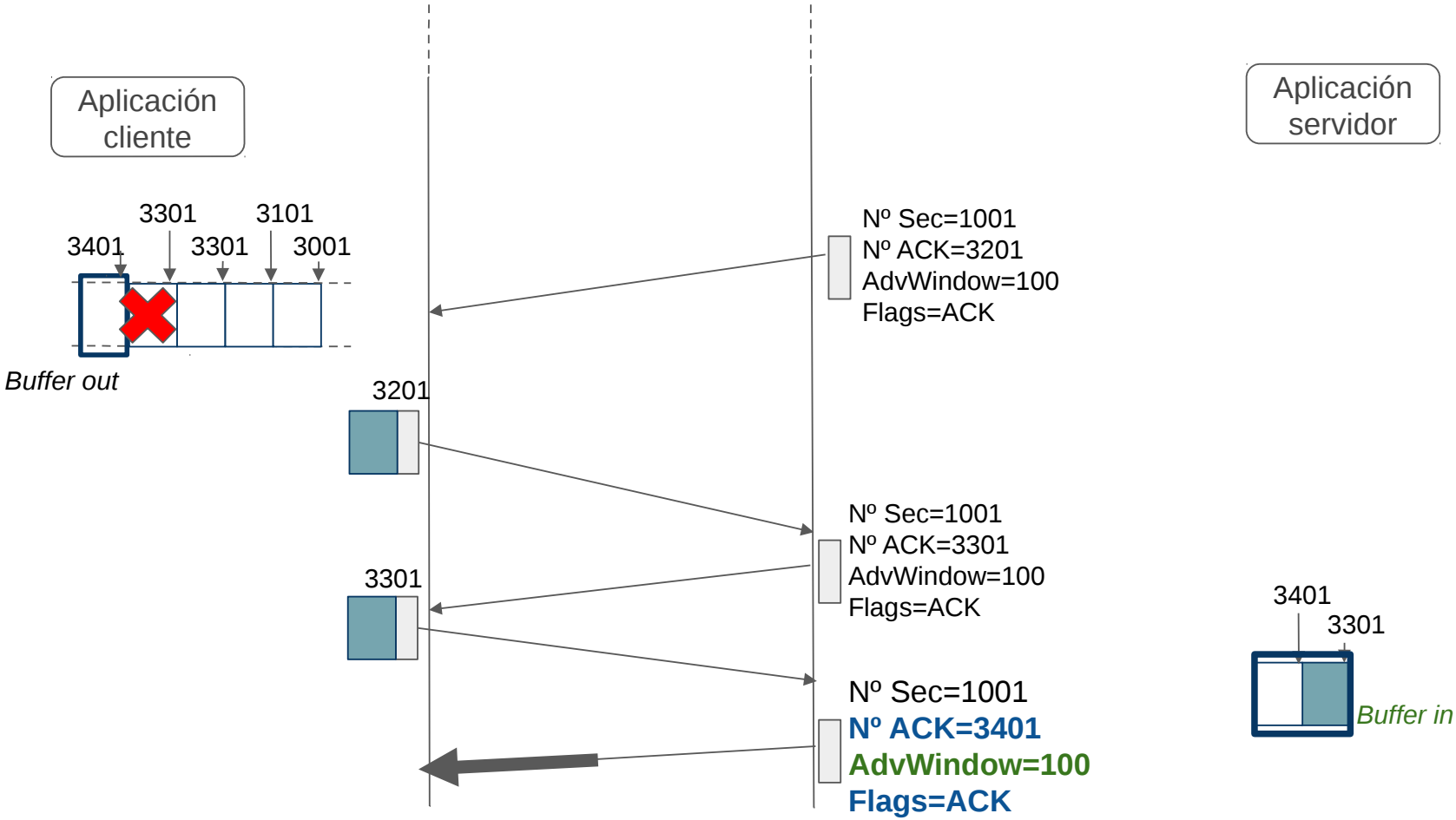
# Ejemplo: envío de datos (XIV)



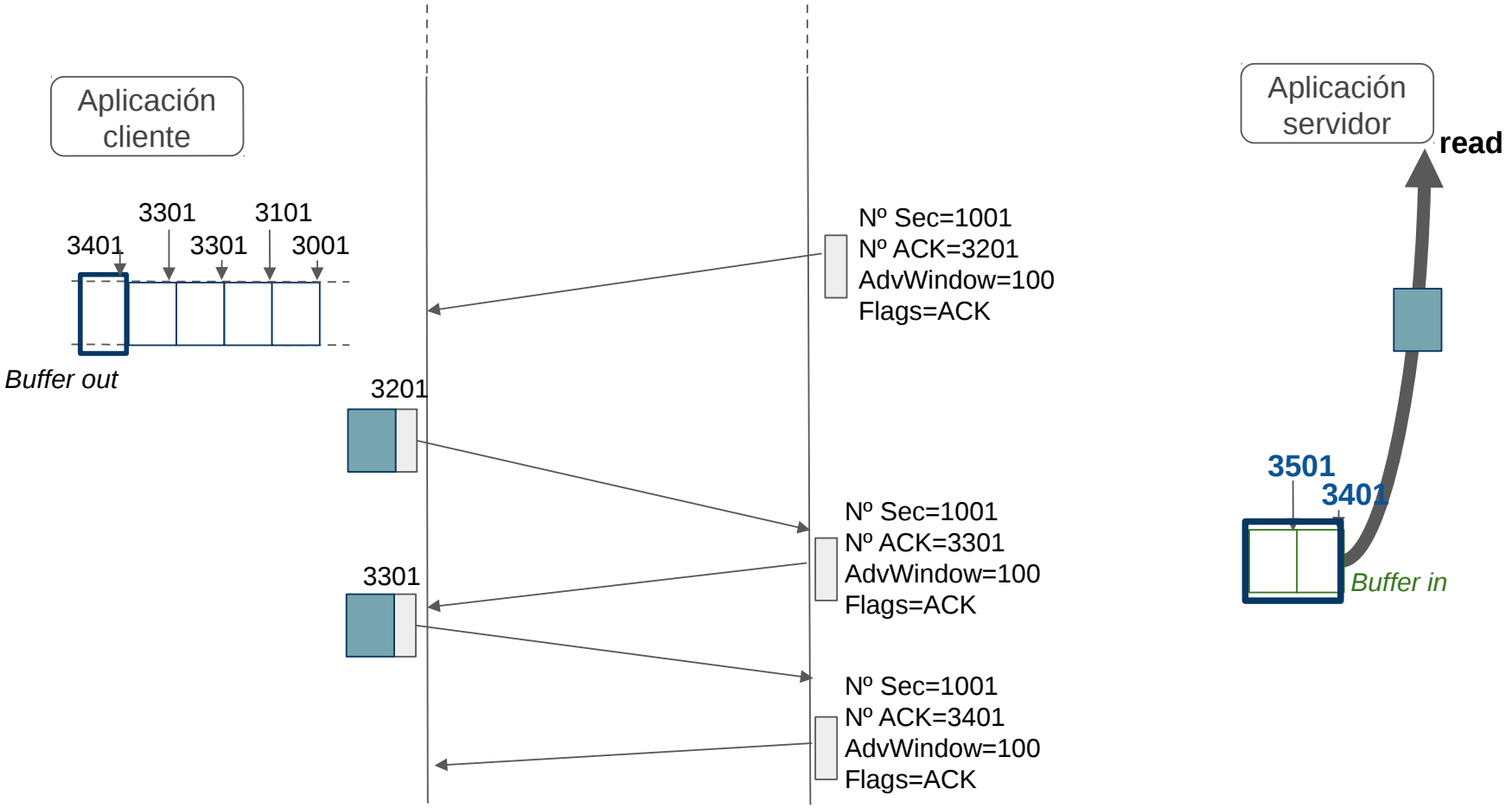
# Ejemplo: envío de datos (XV)



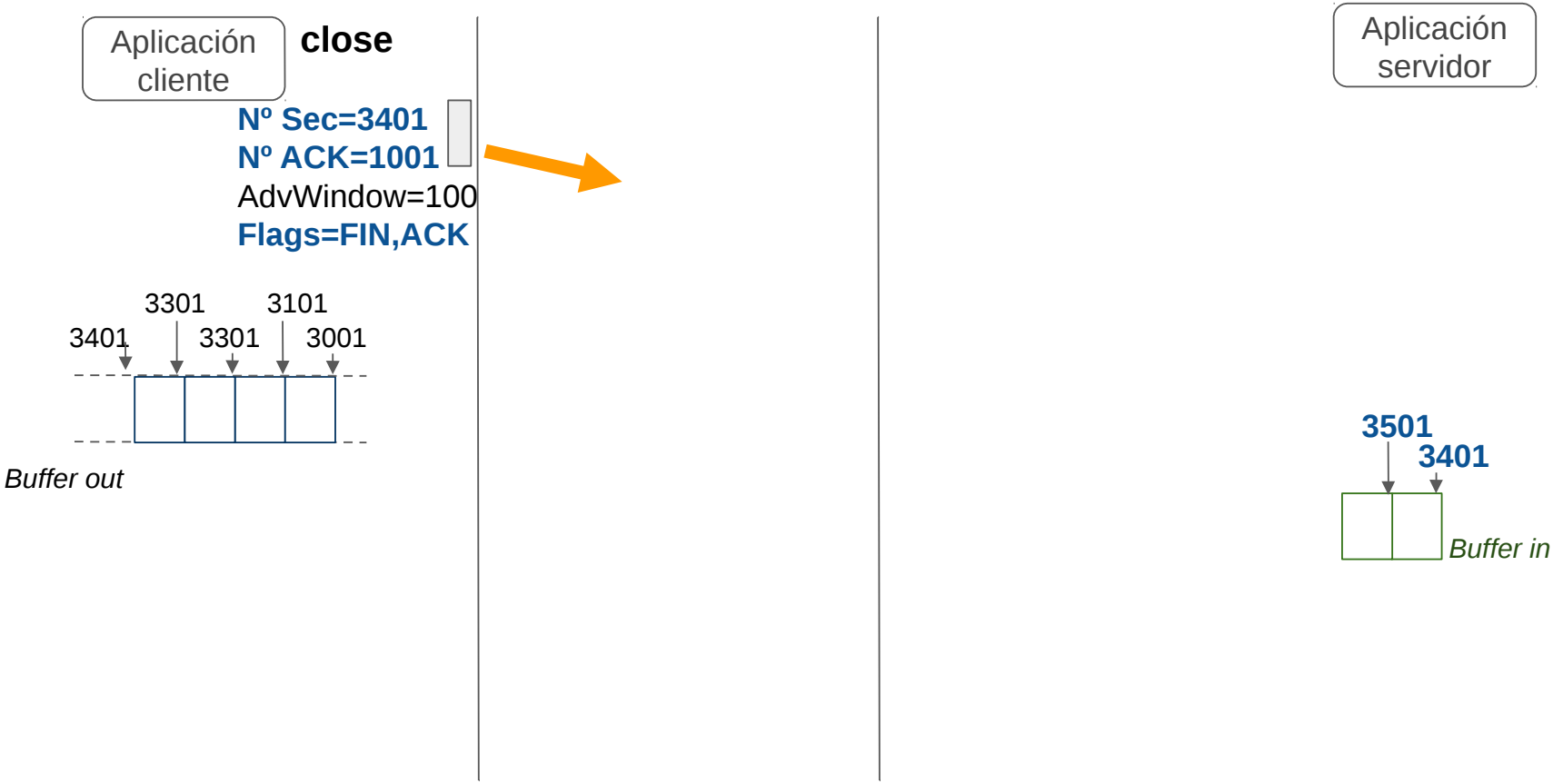
# Ejemplo: envío de datos (XVI)



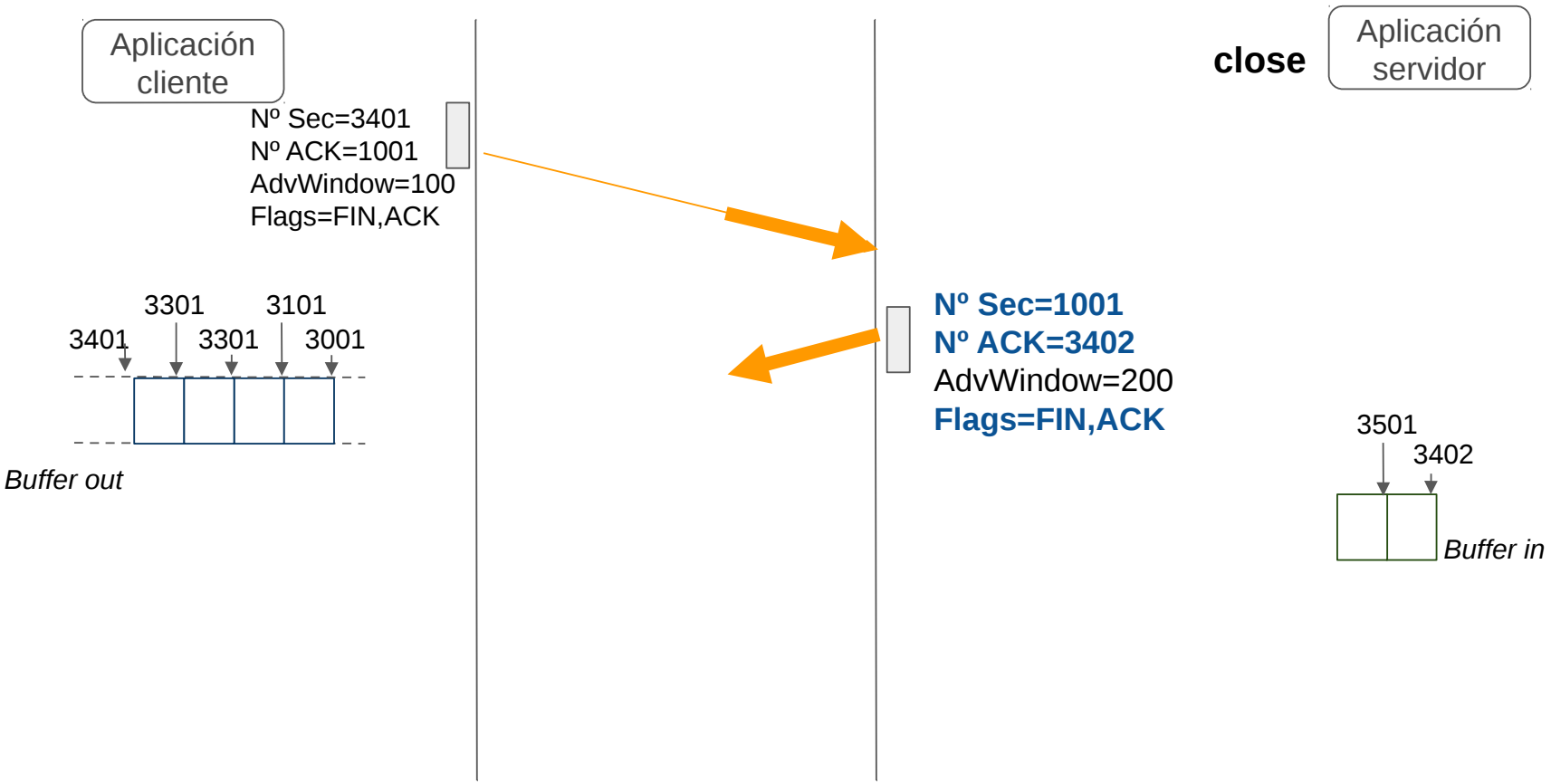
# Ejemplo: envío de datos (XVII)



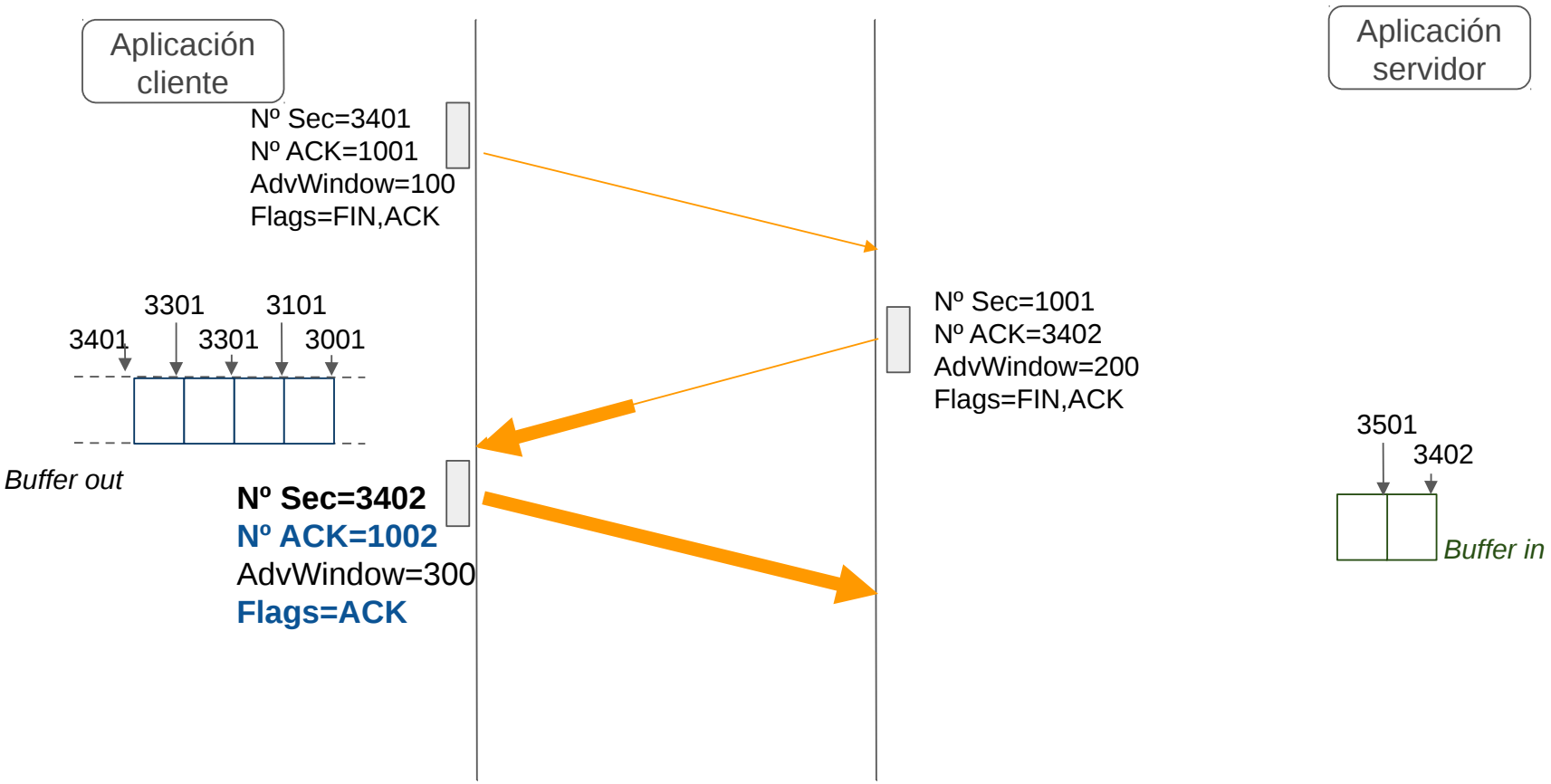
# Ejemplo: cierre de conexión (I)



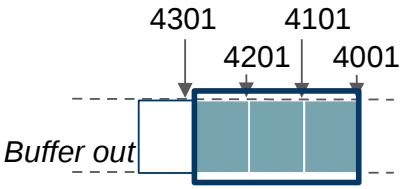
# Ejemplo: cierre de conexión (II)

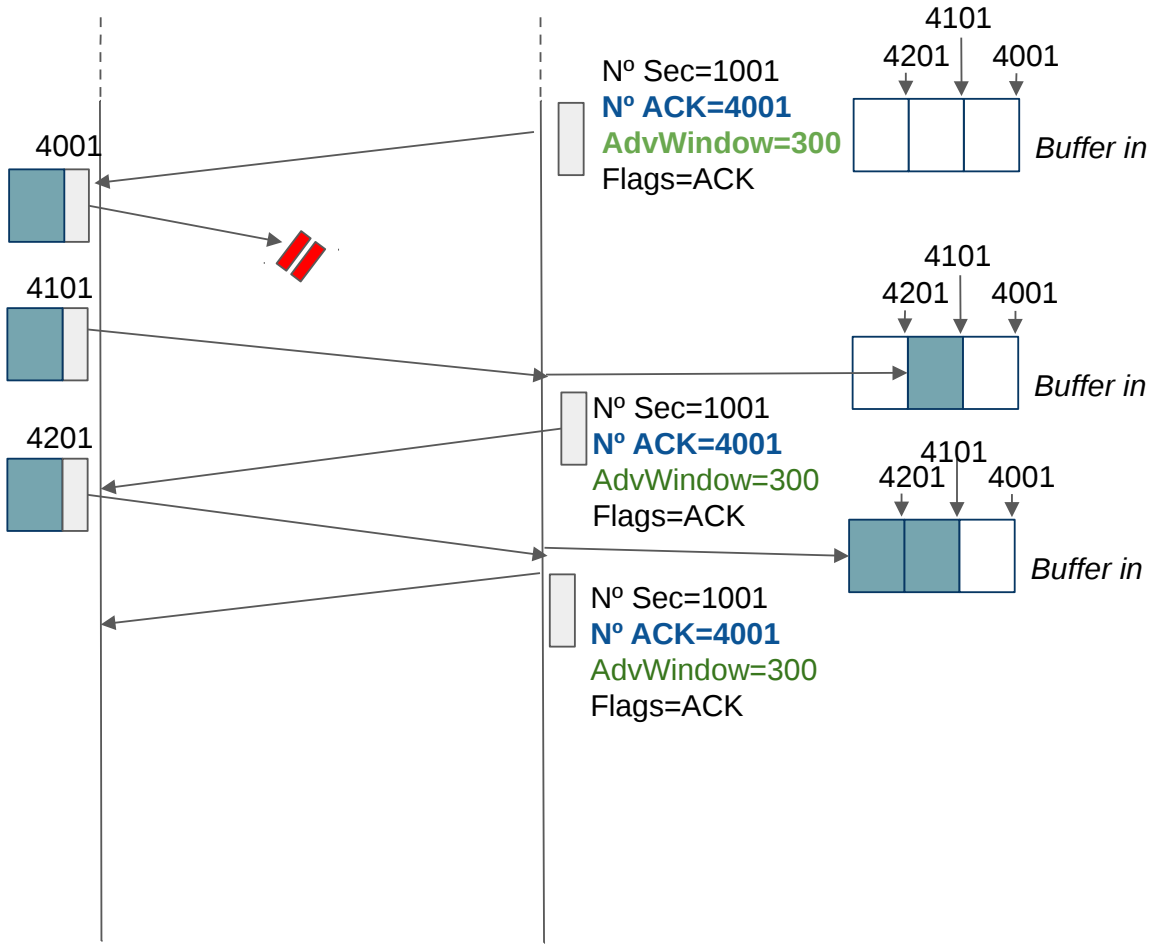


# Ejemplo: cierre de conexión (III)

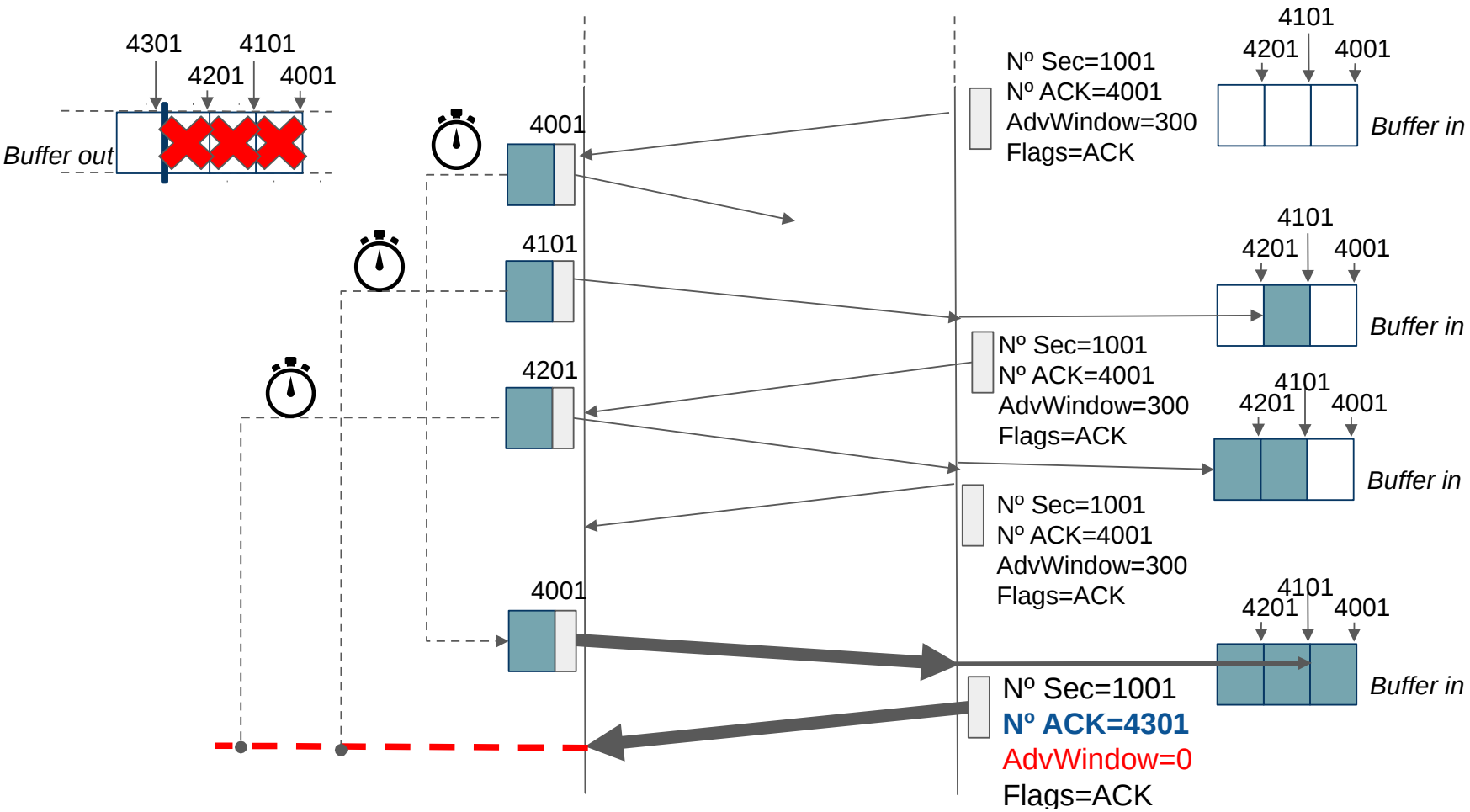


# Ejemplo: pérdida de un segmento



# Ejemplo: retransmisión



# Contenidos

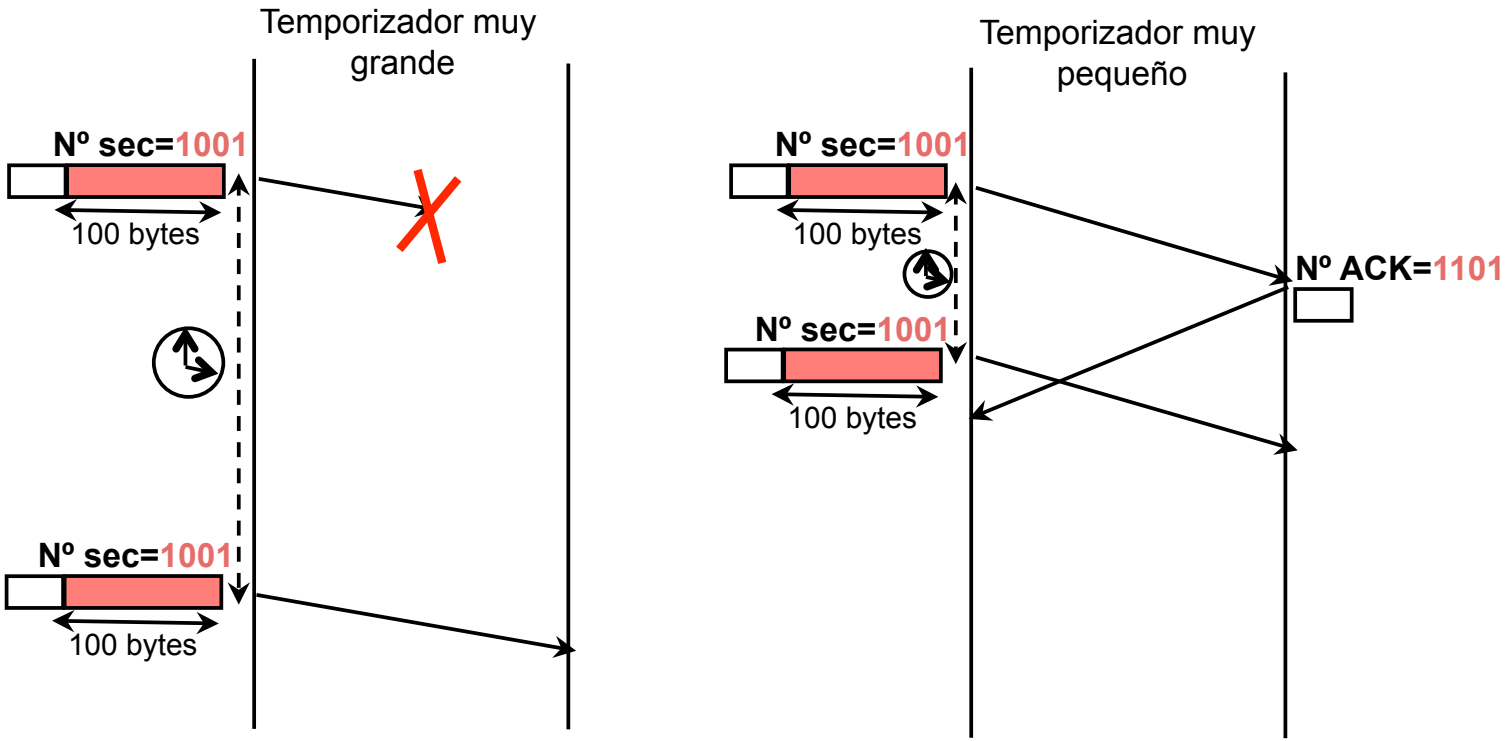
# Sondas de ventana

- Si la aplicación en el lado receptor no lee (`read`) los datos que van llegando, se irá llenando el *buffer* de recepción y se irá anunciando un valor de ventana cada vez menor, que terminará llegando a 0.
- Si el emisor recibe del receptor `AdvertisedWindow=0`, el emisor no puede seguir enviando datos nuevos. El receptor "ha cerrado la ventana".
- En esta situación, el emisor enviará cada cierto tiempo un segmento con un número secuencia igual al último que tiene asentido y **0 bytes de datos**, para provocar que el receptor envíe un asentimientos en el que se pueda comprobar si la ventana se ha vuelto a reabrir. Estos segmentos se denominan sondas de ventana.
- Cuando la aplicación en el lado receptor lea los datos están el *buffer*, éste irá vaciándose y esto se reflejará en nuevos valores en el campo `AdvertisedWindow` de los asentimientos.

# Contenidos

# Plazos de retransmisión

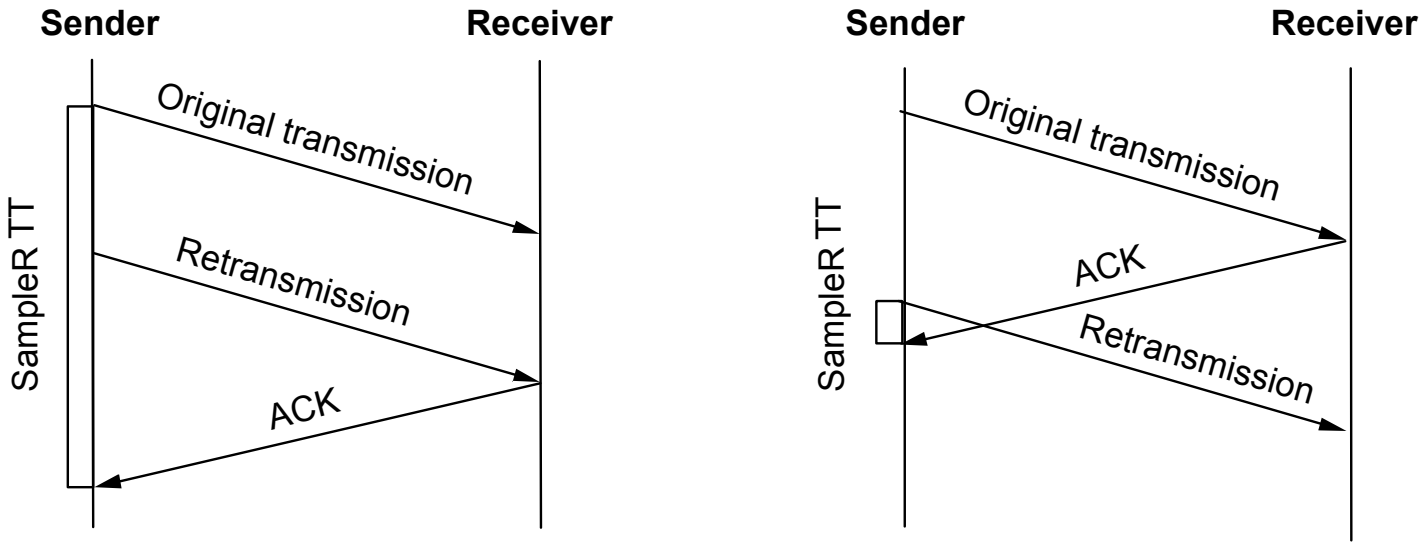
- Cuando se envía un segmento se arranca un temporizador para esperar su asentimiento. Transcurrido el plazo marcado en el temporizador (*timeout*), si no se ha recibido el ACK de ese segmento, se retransmite. Problema: ¿qué plazo es adecuado?
  - Si el plazo es demasiado grande, se tarda mucho en recuperarse de la pérdida.
  - Si el plazo es demasiado pequeño, la retransmisión puede ser innecesaria, lo que genera tráfico superfluo y debería evitarse.



# Retransmisión adaptativa: algoritmo de Karn/Partridge

- Se llama RTT, *round-trip time* (tiempo de ronda), al tiempo que pasa desde que se envía un segmento hasta que llega su ACK.
- El plazo "ideal" debería ser igual al RTT.
- ¿Cómo se sabe cuál es el RTT?
  - Para cada segmento se calcula su RTT.
  - Se define un estimador para el RTT en función de su evolución en el tiempo para los distintos paquetes.
- Como valor de plazo se usa aproximadamente el doble del RTT estimado.
  - Es tan malo generar tráfico innecesario que se usa un valor de plazo muy conservador para evitar quedarse corto.

# Retransmisión adaptativa: cuándo tomar muestras




- No se toman muestras del RTT cuando se retransmite, pues no se puede saber si el ACK recibido corresponde a transmisión original (izda) o a la retransmisión (dcha).

# Retransmisión adaptativa: *Exponential Backoff*

- Cada vez que se retransmite el mismo paquete se dobla el plazo para ese paquete.
  - Se supone que sólo se pierden paquetes por congestión → doblando timeout se contribuye a que la congestión decrezca.
- Este algoritmo de duplicar el plazo en cada retramisión consecutiva recibe el nombre de Exponential Backoff ("retroceso exponencial").

# Contenidos

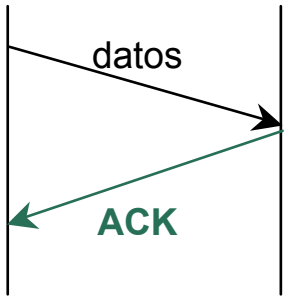
# Extensiones de TCP

- Se implementan como campos opcionales de la cabecera, situados a continuación de los campos fijos de la cabecera.
- No todas las implementaciones de TCP entienden todas las posibles opciones.
- Opciones más habituales:
  - Marcas de tiempo *(timestamps)*:
    - Almacena la hora de envío (timestamp) en los segmentos enviados: el receptor lo copia al enviar un ACK. Permite medir con más precisión el RTT (incluyendo retransmisiones)
  - Extensión del espacio de números de secuencia
    - Para que tarde más en dar la vuelta se utiliza el timestamp + número de secuencia como identificador de segmento
  - Escalado de la ventana anunciada *(Window Scale)*.
  - Tamaño máximo de segmento (MSS: *Maximum Segment Size*).

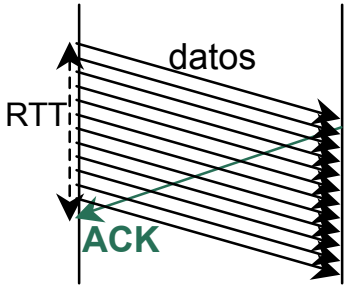
# Contenidos

# Extensiones de TCP
## *Window Scale* (I)

- ¿Cuál debería ser el tamaño de ventana óptimo? El que permita aprovechar toda la capacidad de la red.
  - La idea es no parar de transmitir mientras llega el ACK del primer paquete transmitido.
  - A partir de que llega ese ACK, se genera un hueco en la ventana, por lo que una ventana más grande nunca se aprovechará.
- El número de bytes que se puede transmitir en un RTT es el producto del RTT por el ancho de banda (BW).
- Luego la ventana óptima debería tener un tamaño de RTT x BW.
  - si es menor, el emisor tendrá que parar sin nada que hacer
  - si es mayor, nunca se aprovechará todo su tamaño




Infrautilización de la red




Máximo aprovechamiento de la capacidad de la red

# Extensiones de TCP
## *Window Scale* (II)

- La ventana anunciada, para aprovechar al máximo la capacidad del canal, debería ser = `RTT x Bandwidth`.
  - A 100 Mbps con 100ms de RTT la ventana debería ser = 1.25 MBytes
- Con 16 bits en el campo `AdvertisedWindow` la ventana máxima es de 64 KBytes:
  - en medios rápidos, el emisor no podría transmitir todo lo rápido que le permitiría el medio por culpa de la ventana.
- La opción *Window Scale* aumenta el tamaño de la ventana:
  - El cliente incluye en su segmento `SYN` esta opción, junto con un *factor*.
  - Si el servidor está dispuesto a usar esta opción, en su segmento `SYN+ACK` incluirá también esta opción junto con un *factor* que será el que usará él (y que puede ser diferente al factor que utilizará el cliente)
  - A partir de entonces:
    Ventana Anunciada Real $= 2^{factor} \times AdvertisedWindow$
    - Ej.: Si *factor* es 1, la ventana anunciada real es el doble del valor que viaja en el campo AdvertisedWindow de la cabecera TCP

# Contenidos

# Extensiones de TCP
## MSS

- Se intenta encontrar para la conexión el mayor tamaño de segmento posible que no requiera fragmentación.
- MSS *(Maximum Segment Size)*: en una máquina, máximo tamaño de la parte de datos del segmento TCP que no causa fragmentación, suponiendo que la cabecera IP y la cabecera TCP no tengan opciones.
  - Ej: en Ethernet, el máximo tamaño de un datagrama IP es 1500 bytes, lo que da un MSS=1460 bytes

    **MSS = 1500 - 20 (cabecera IP sin opciones) - 20 (cabecera TCP sin opciones) = 1460 bytes**
- Cuando NO se usa la opción MSS de TCP, cada lado de una conexión hará sus segmentos de tamaño menor o igual a su MSS.
- Cuando SÍ se usa la opción MSS de TCP, en el segmento SYN cada lado incluye el valor de su MSS.
  - Si ambos lados usan esta opción, ambos lados harán segmentos con un tamaño igual al menor de los dos valores de MSS que figuran en los segmentos SYN.

# Path MTU Discovery

- Usar la opción de MSS no garantiza que no vaya a haber fragmentación: en algún salto intermedio puede existir un nivel de enlace con un tamaño de datos menor que los que hay en los extremos.
- Para evitar la fragmentación también en esos casos, se usa *Path MTU Discovery*:
  - Al principio de una conexión, cada lado elige como tamaño el menor entre su MSS y el MSS que le anuncia el otro extremo de la conexión.
  - A los datagramas IP que contienen los segmentos se les activa el flag DF *(Don't Fragment)*.
  - Si un *router* del camino no puede reenviar un datagrama sin fragmentarlo, y ese datagrama tiene el flag DF, descarta el datagrama y envía a su origen un ICMP.
    - Es un ICMP de "destino inalcanzable por necesitarse fragmentación y estar activado el flag DF" (ICMP de tipo=3, código=4)
    - El ICMP incluye el máximo tamaño de datagrama que le permitiría al *router* no fragmentar.
  - El origen, al recibir un ICMP de este tipo, disminuye el tamaño de segmento.

# Contenidos

# Referencias

- L. Peterson, **Computer Networks: A Systems Approach (3rd ed)**: apartado 5.2
- A. Tanembaum, **Computer Networks (4th ed)**: apartado 6.5