Nivel de transporte: UDP y TCP

Arquitectura de Redes de Ordenadores y Arquitectura de Internet

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

Abril de 2016

Contenidos

Contenidos

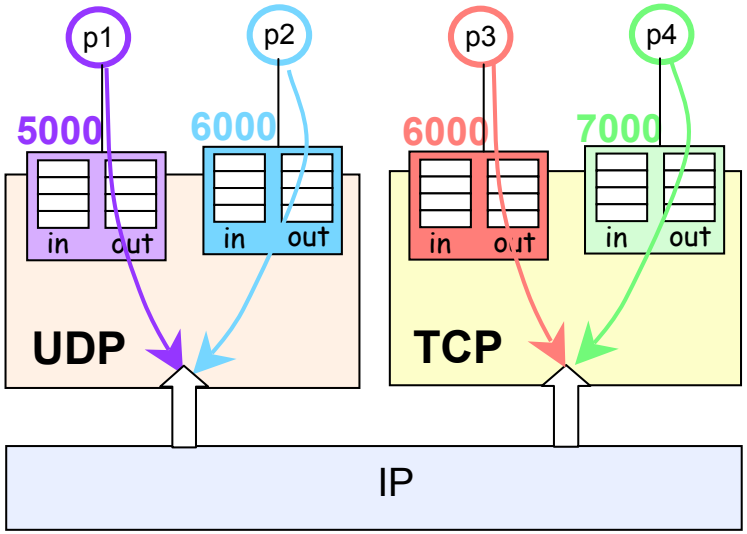
Introducción

- 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

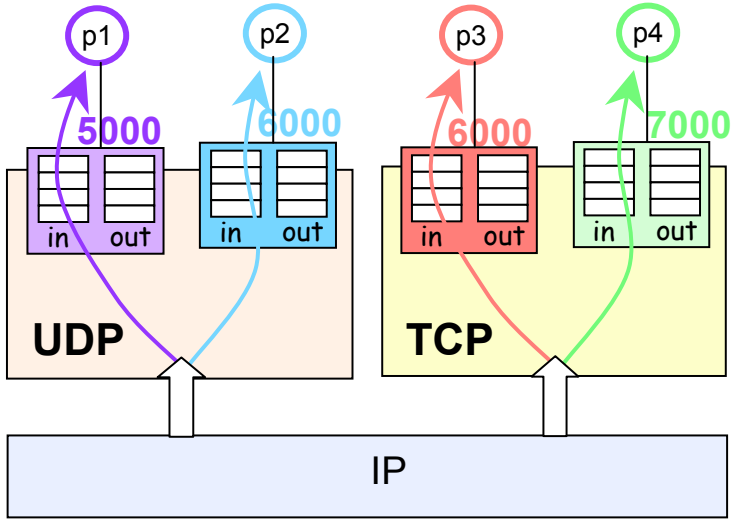
- El Nivel de Transporte TCP/IP:
  - multiplexa las unidades de datos que envían las aplicaciones a través de los puertos, encapsulándolas en unidades de datos de UDP o TCP
  - demultiplexa las unidades de datos de UDP y TCP, pasando los datos a las aplicaciones.

Multiplexación



Envío de datos en el nivel de transporte

Demultiplexación



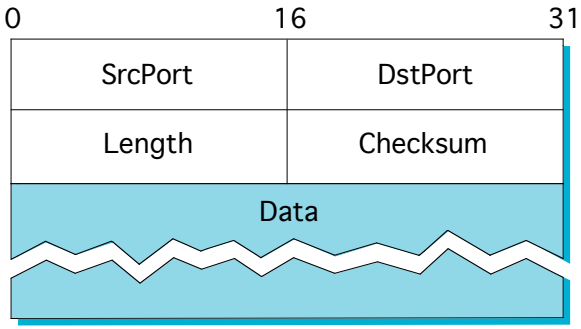
Recepción de datos en el nivel de transporte

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



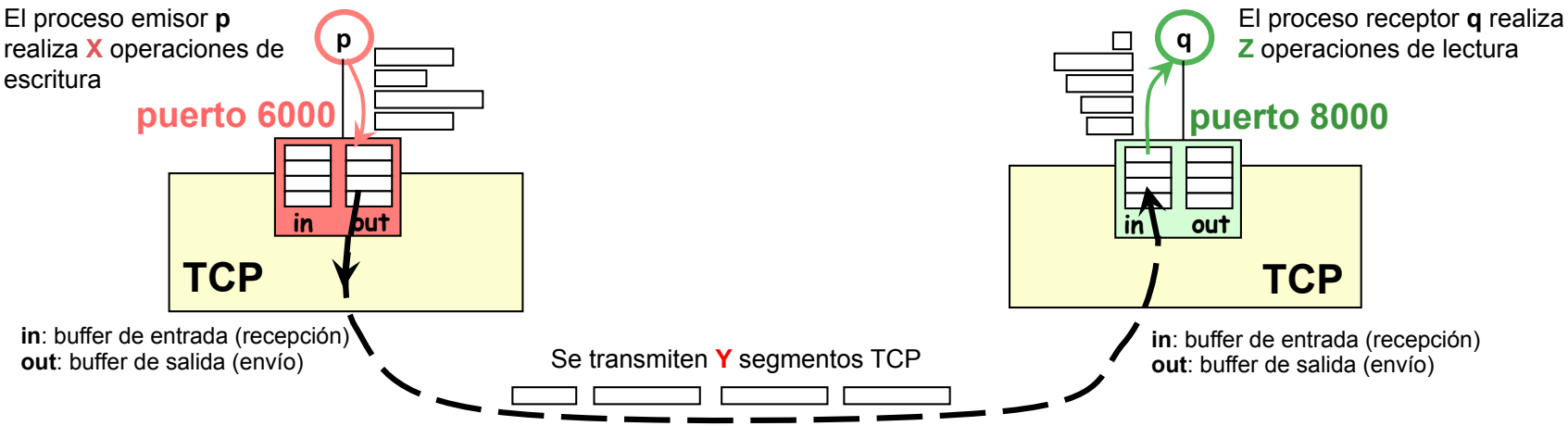
UDP

- 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.

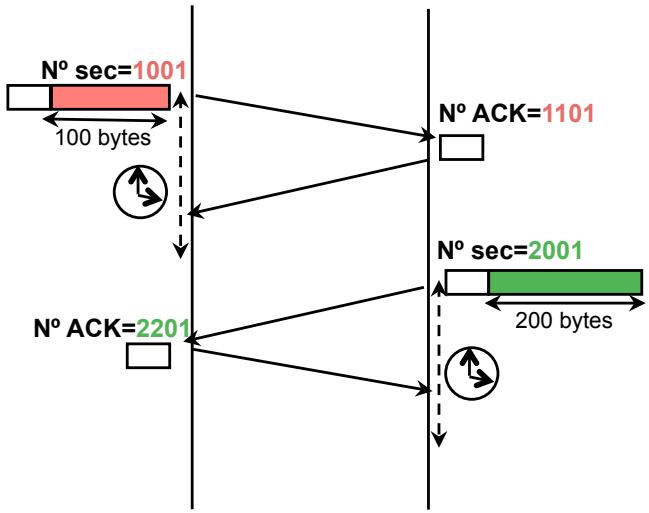
Servicio Orientado a Conexión

- 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.

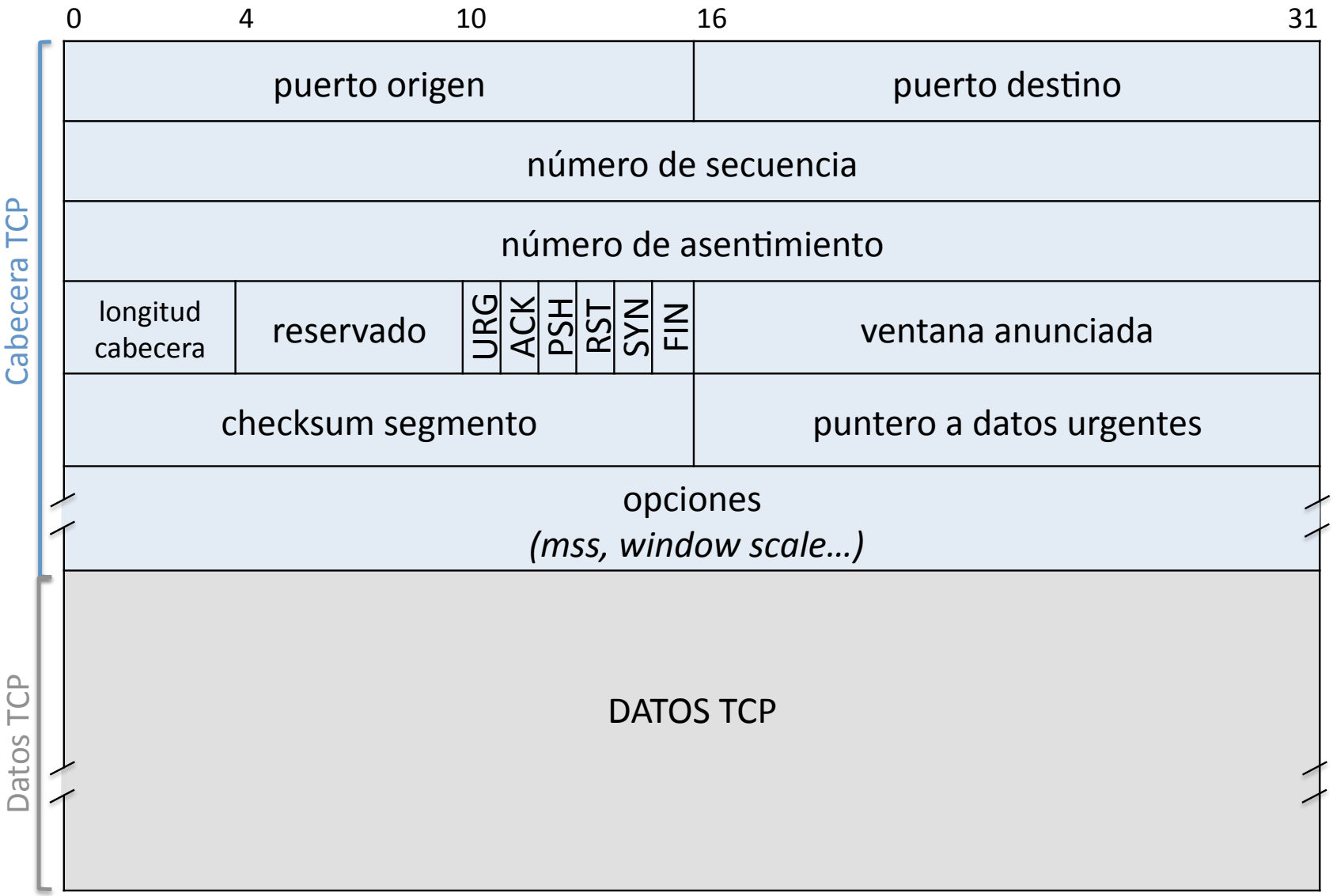
- Idea básica:
  - Los segmentos con datos llevan un número de secuencia.
  - El receptor de los datos debe mandar asentimientos (ACKs).
  - Para cada segmento con datos transmitido se espera un plazo de tiempo a que llegue su ACK. Si vence el plazo y no se ha recibido su ACK se retransmite el segmento.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

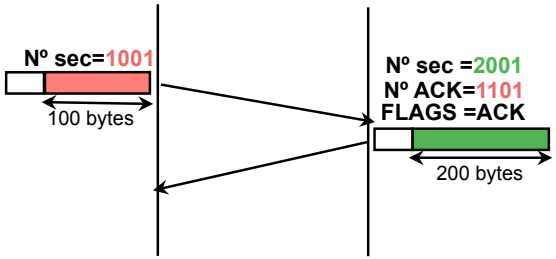
- Cada segmento con datos lleva un número de secuencia, `SequenceNum`, de 32 bits.
- El número de secuencia numera bytes, y NO segmentos: identifica el número de orden del primer byte de datos que lleva el segmento.
- Al establecerse una conexión se elige aleatoriamente un número de secuencia inicial para que no se confundan segmentos aún en tránsito procedentes de conexiones diferentes.
- A 100 Mbps los números de secuencia darán la vuelta en 6 minutos.

Número de asentimiento

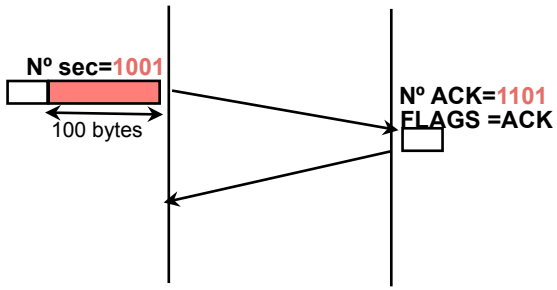
- El receptor de segmentos de datos tiene que asentir los que le llegan correctamente, activando el flag `ACK` y rellenando el campo `AcknowledgmentNum`.
- NO es necesario enviar un asentimiento por cada segmento con datos que se recibe. Se puede esperar a asentir varios segmentos de una sola vez.
- El número de asentimiento indica el número de secuencia del próximo byte que se espera recibir, asintiéndose de esta manera hasta el byte anterior incluido.
- En TCP básico no hay "rechazo selectivo": No hay forma en que el receptor le diga al emisor que tiene 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. Aquí no estudiaremos esta opción.

Número de asentimiento

- 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 número de secuencia inicial) y asiente los números de secuencia que está usando el otro extremo.

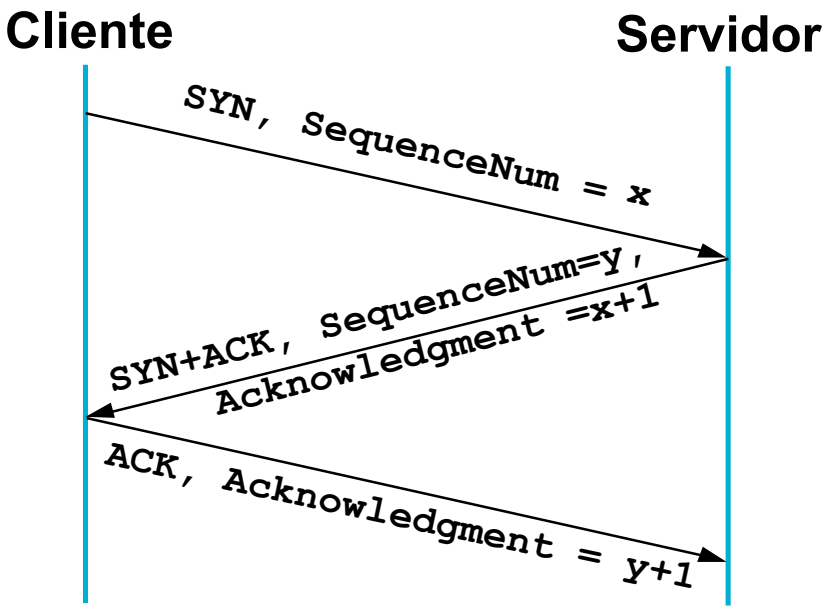
Ventana anunciada (o ventana de flujo)

- Se usa un protocolo de ventana para coordinar el envío de segmentos de datos, este mecanismo se denomina control del flujo.
- El receptor indica en el campo `AdvertisedWindow` (ventana anunciada o ventana de flujo) el número de bytes (a partir del indicado en el número de asentimiento) que está dispuesto a recibir del emisor. Esta cantidad de bytes está directamente relacionada con el tamaño del *buffer* de recepción que tiene reservado la implementación de TCP.
- El emisor puede transmitir esos bytes aunque no reciba 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 de flujo diferentes, una para cada sentido de la comunicación.

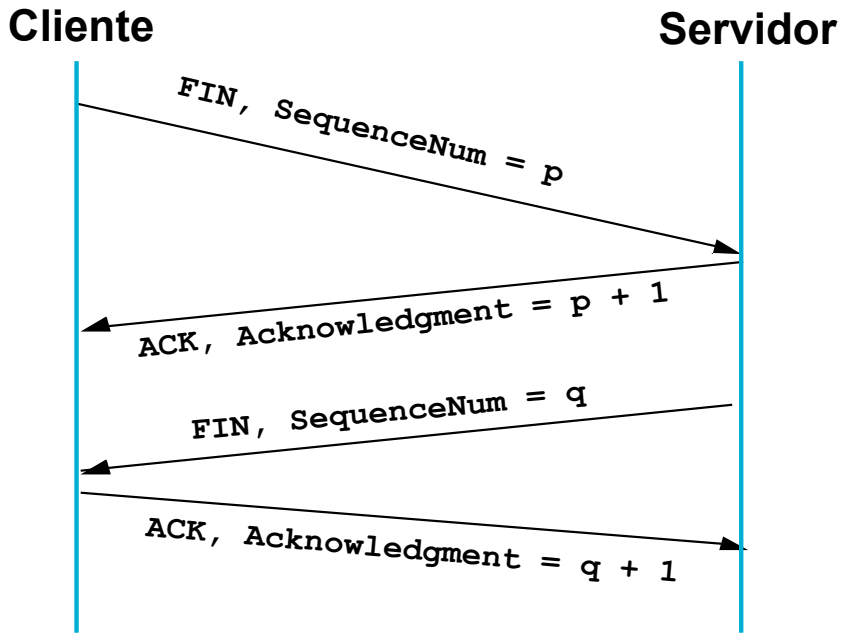
Contenidos

Establecimiento y cierre de la conexión

Establecimiento de la conexión

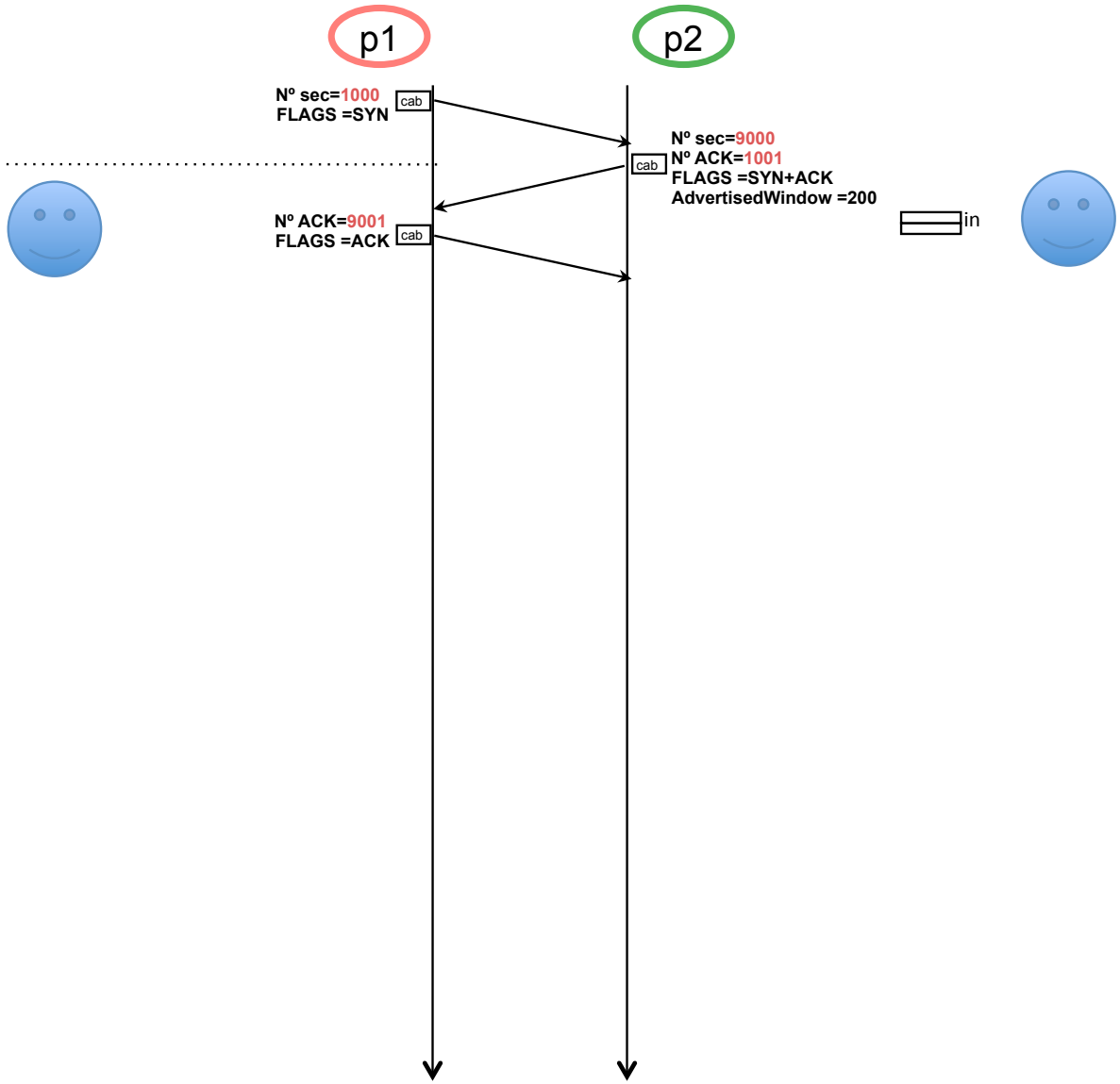


Cierre de la conexión

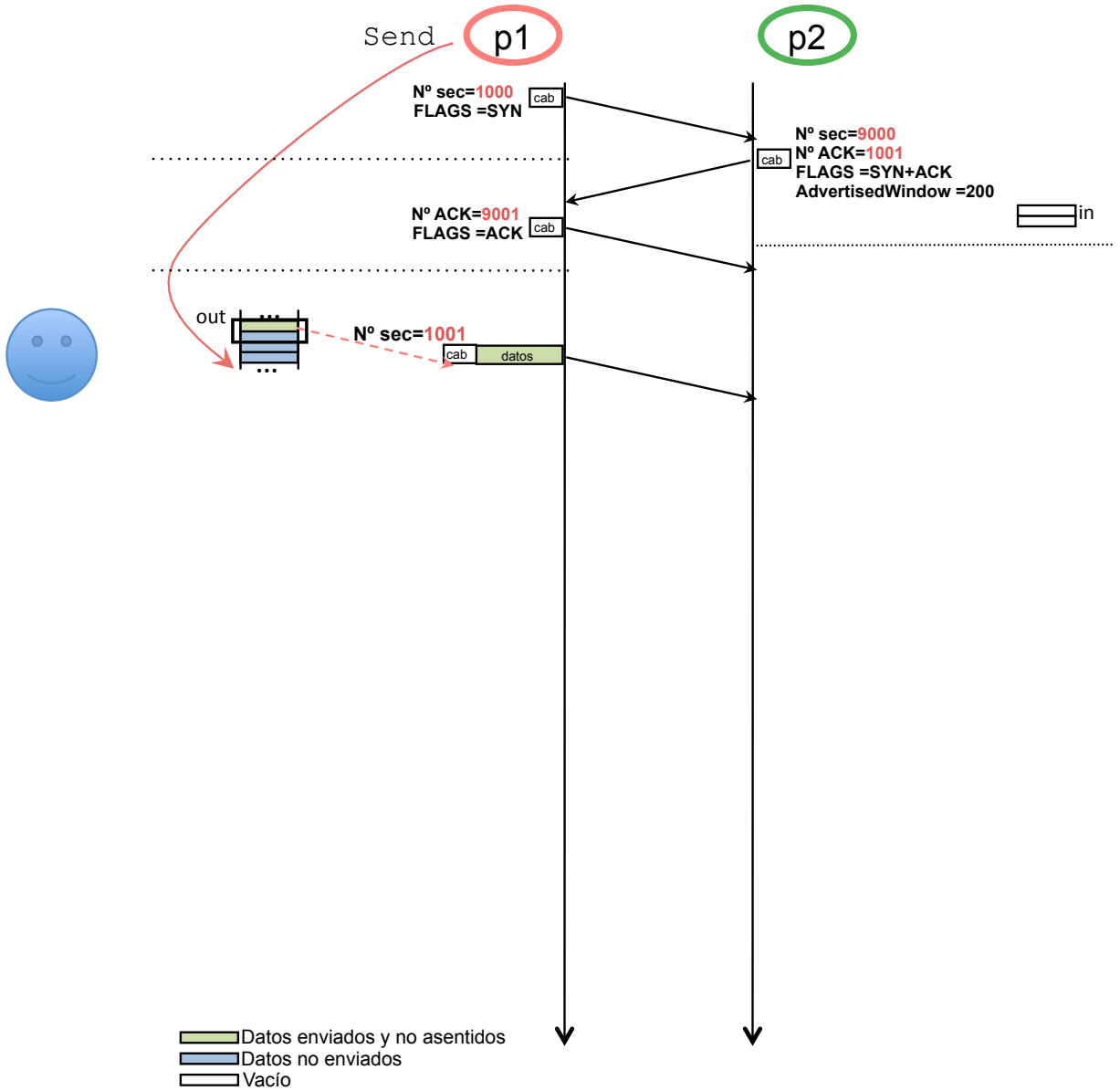


Contenidos

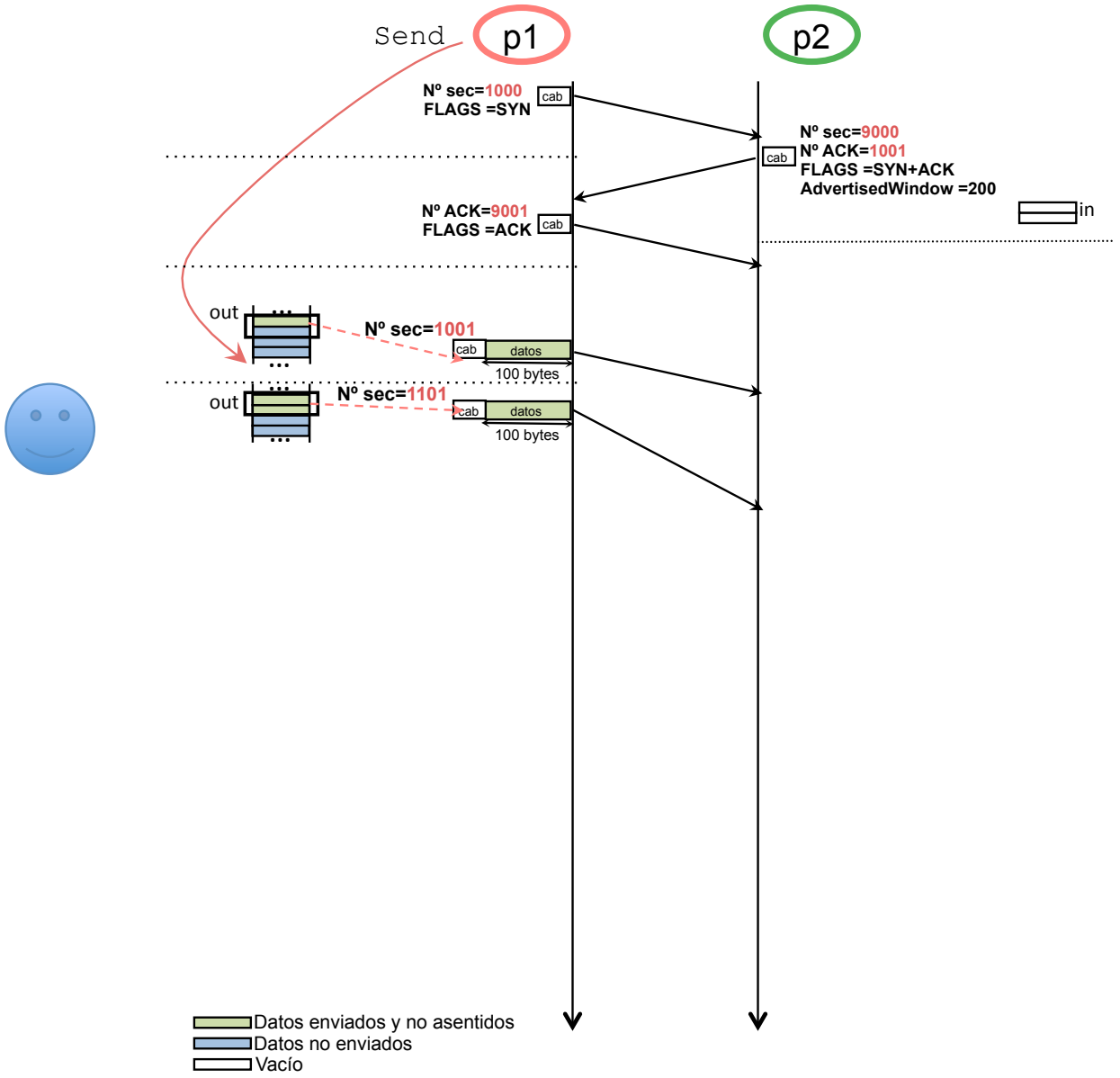
Ejemplo


p1
p2
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
cab
Nº ACK=9001
FLAGS =ACK
cab
in

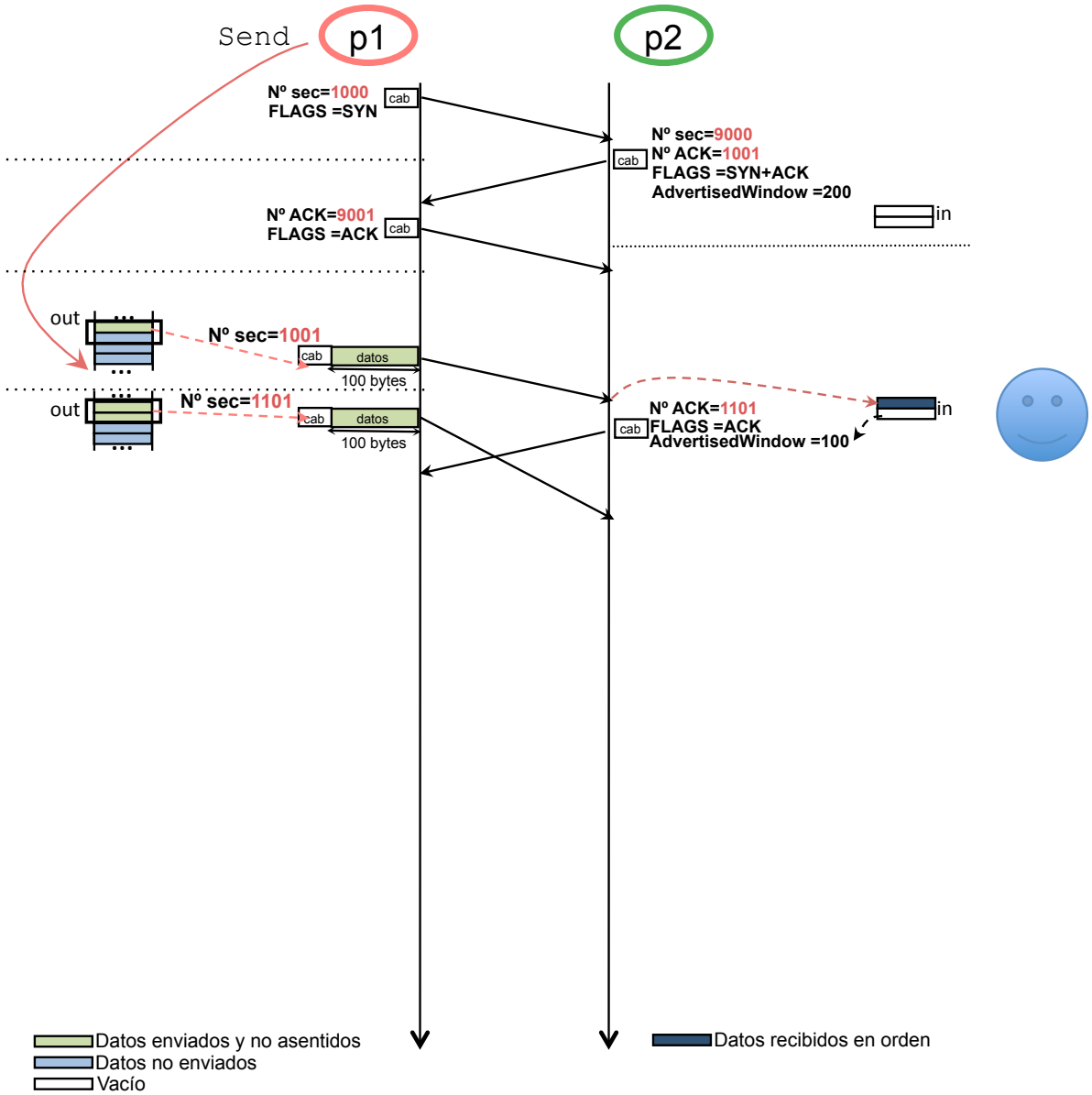
Ejemplo


Send
p1
p2
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
cab
Nº ACK=9001
FLAGS =ACK
cab
in
out
Nº sec=1001
cab
datos
Datos enviados y no asentidos
Datos no enviados
Vacío

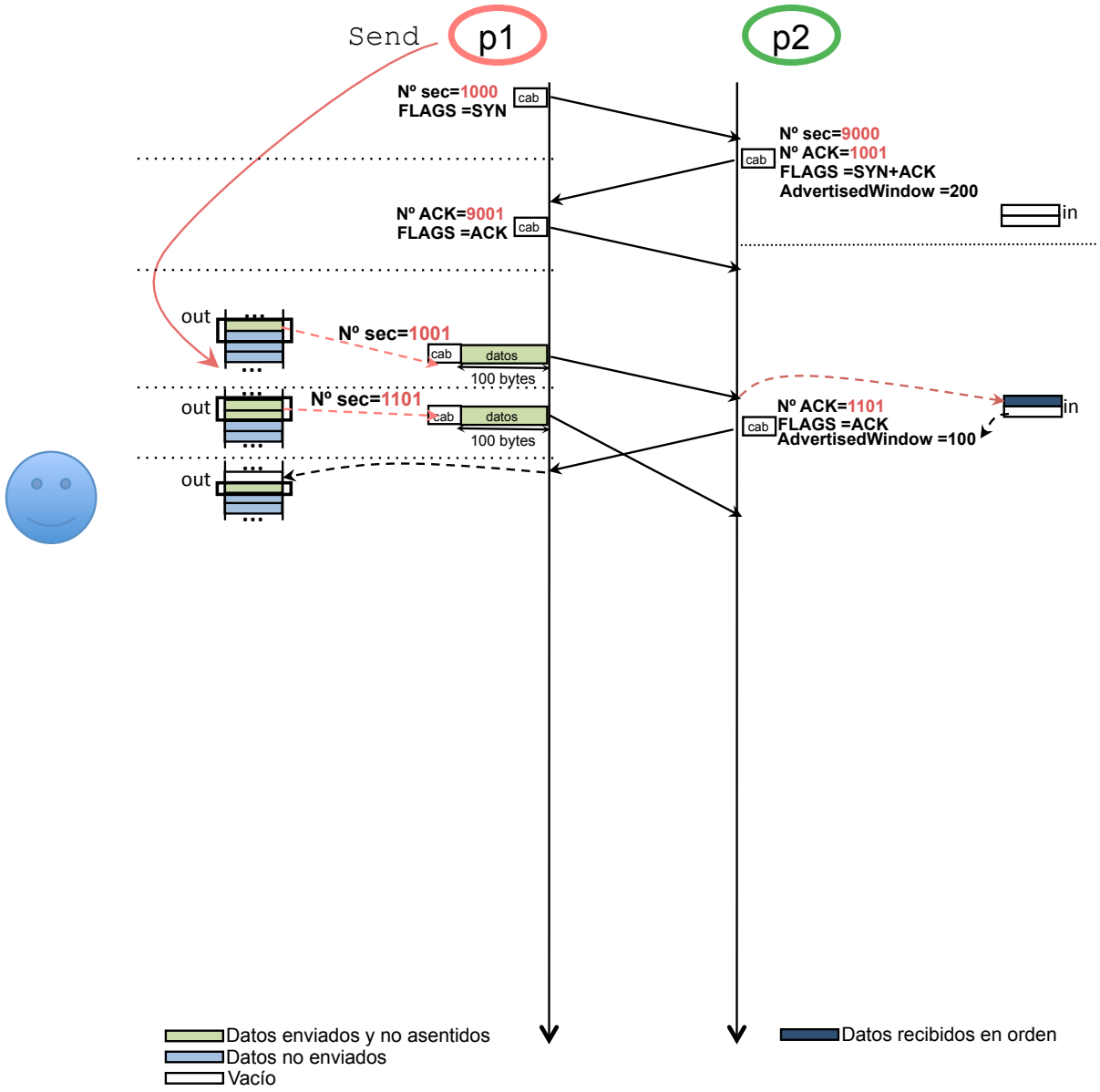
Ejemplo


Send
p1
p2
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
out
Nº sec=1101
datos
100 bytes
Datos enviados y no asentidos
Datos no enviados
Vacío

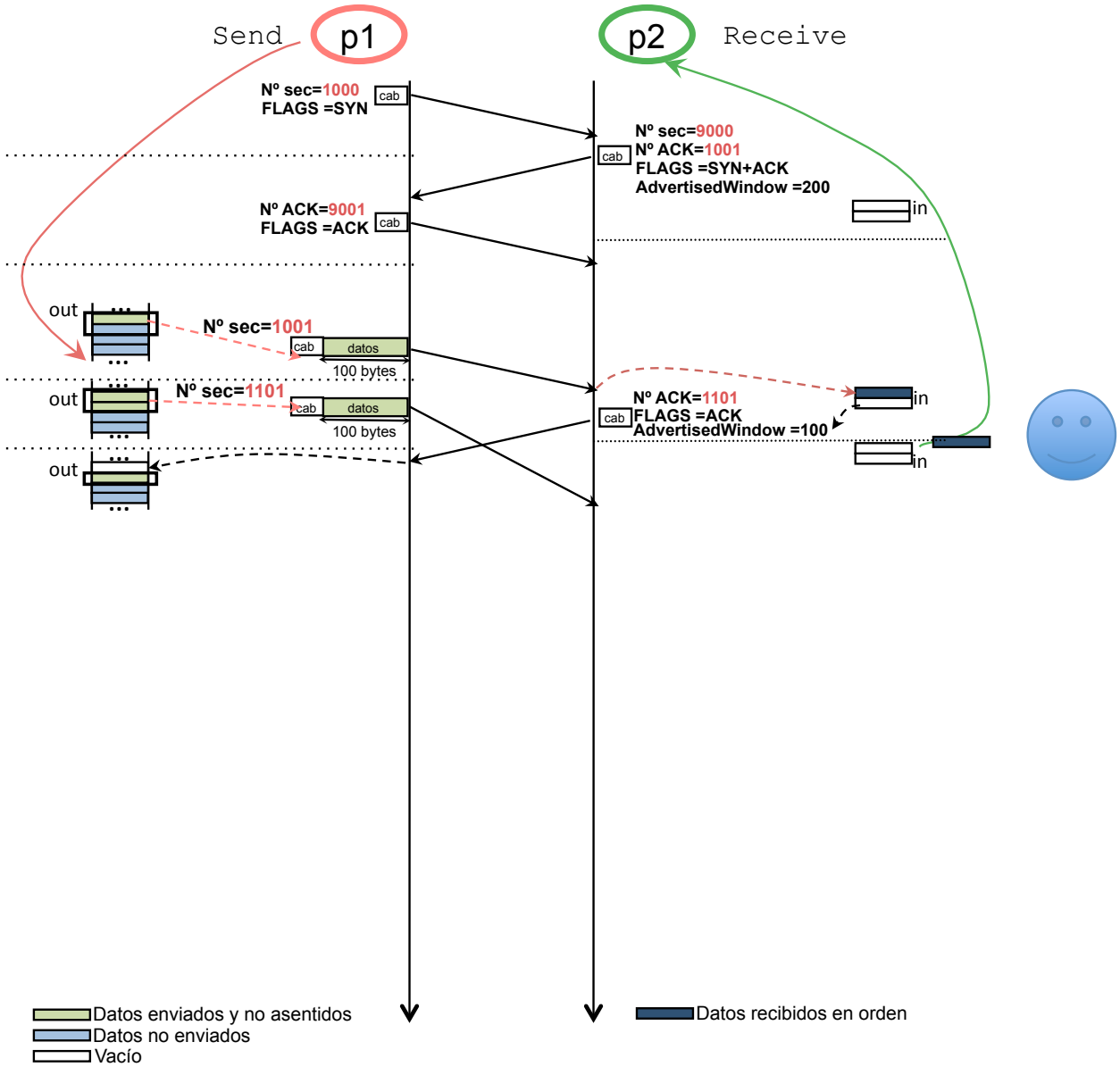
Ejemplo


Send
p1
p2
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
in
out
Nº sec=1001
datos
100 bytes
out
Nº sec=1101
datos
100 bytes
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

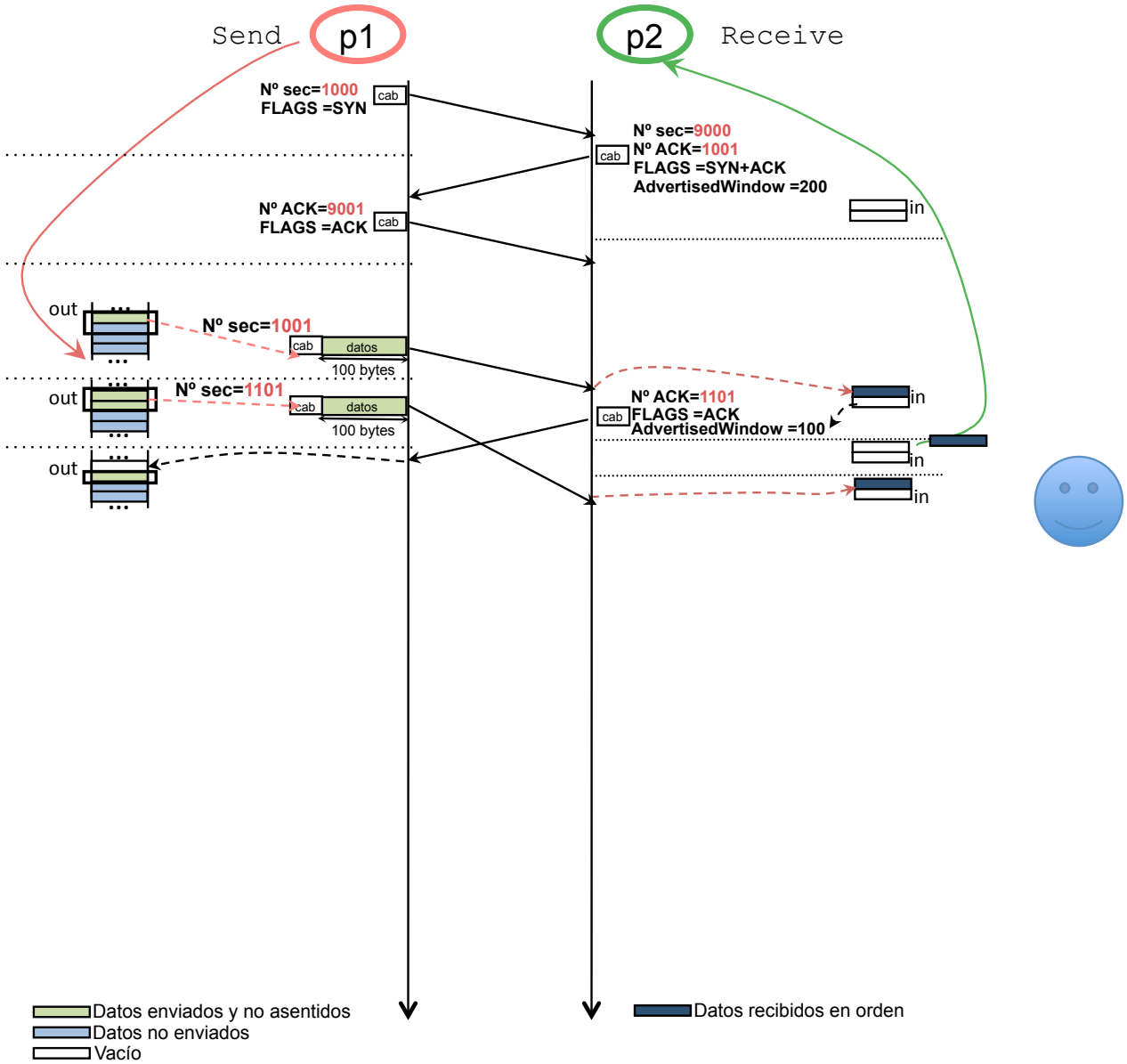
Ejemplo


Send
p1
p2
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
in
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

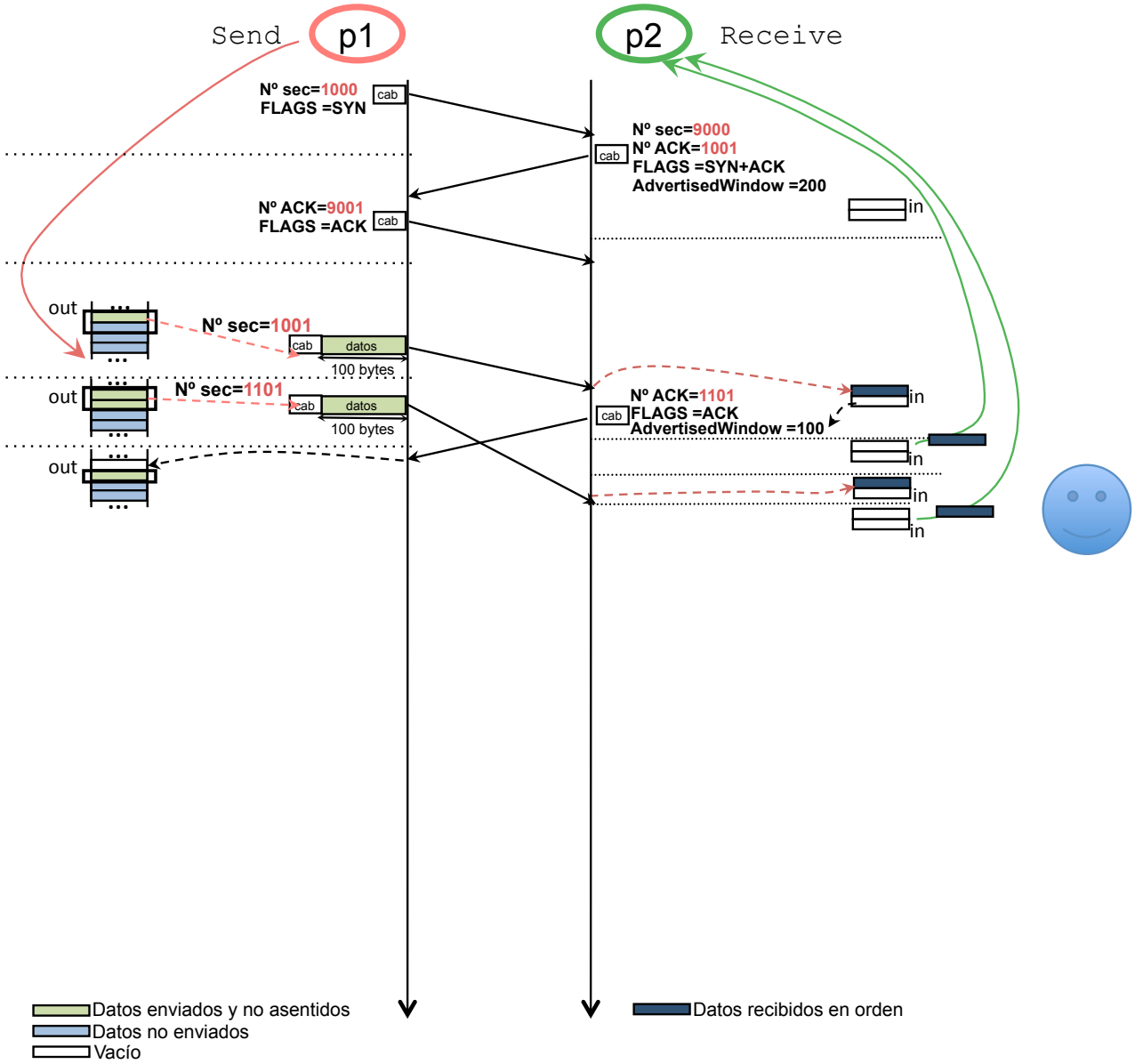
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

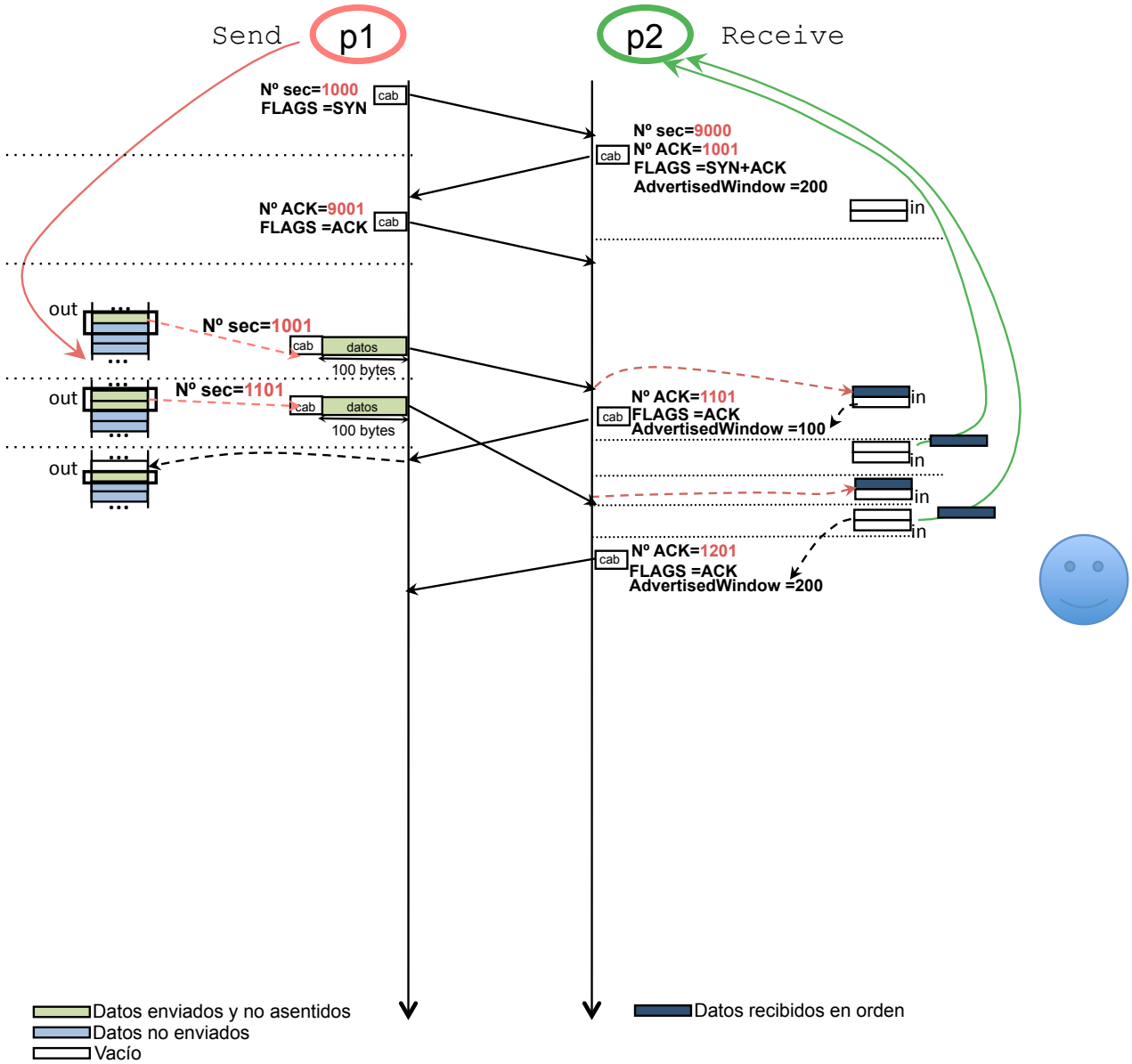
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

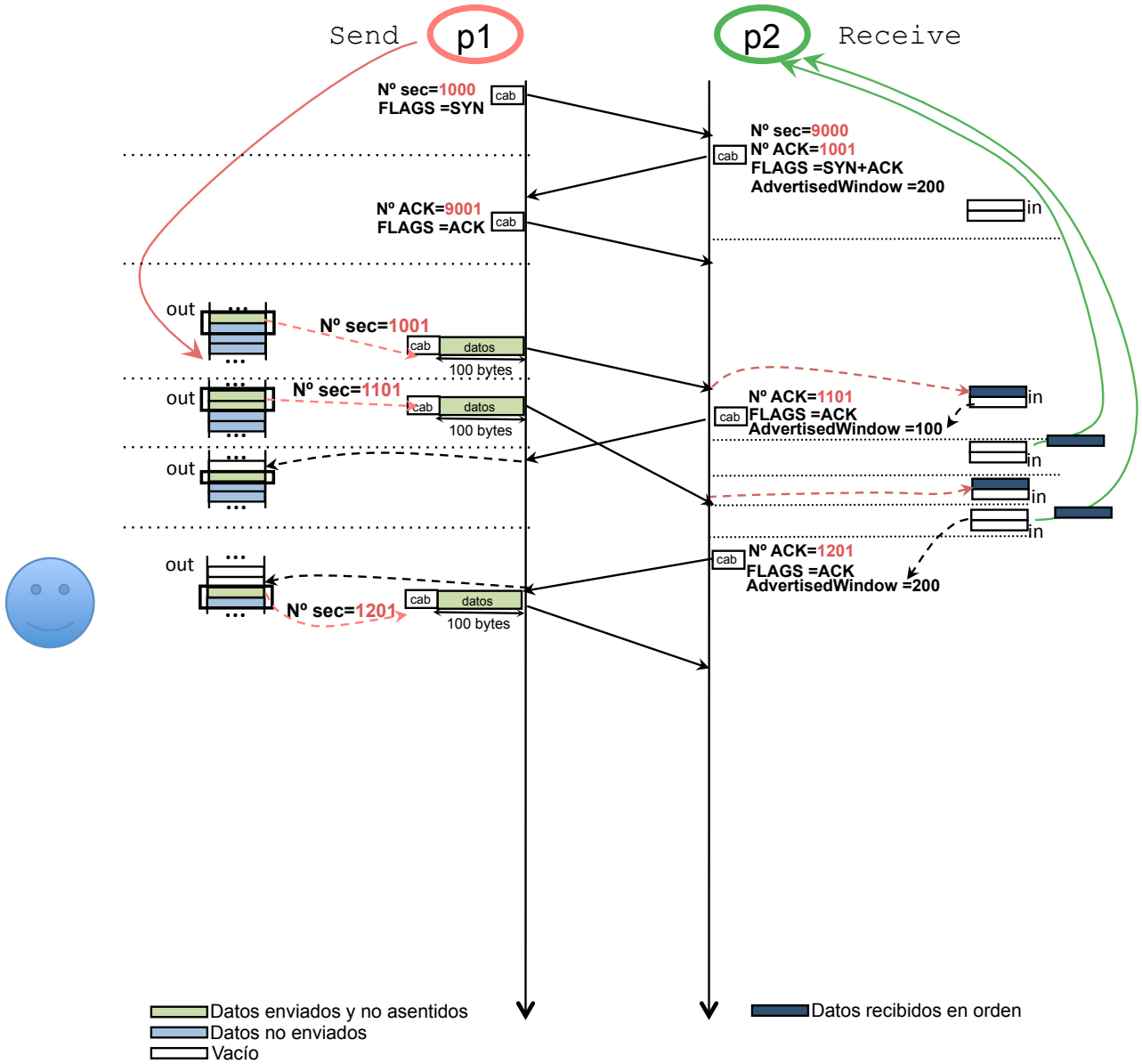
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

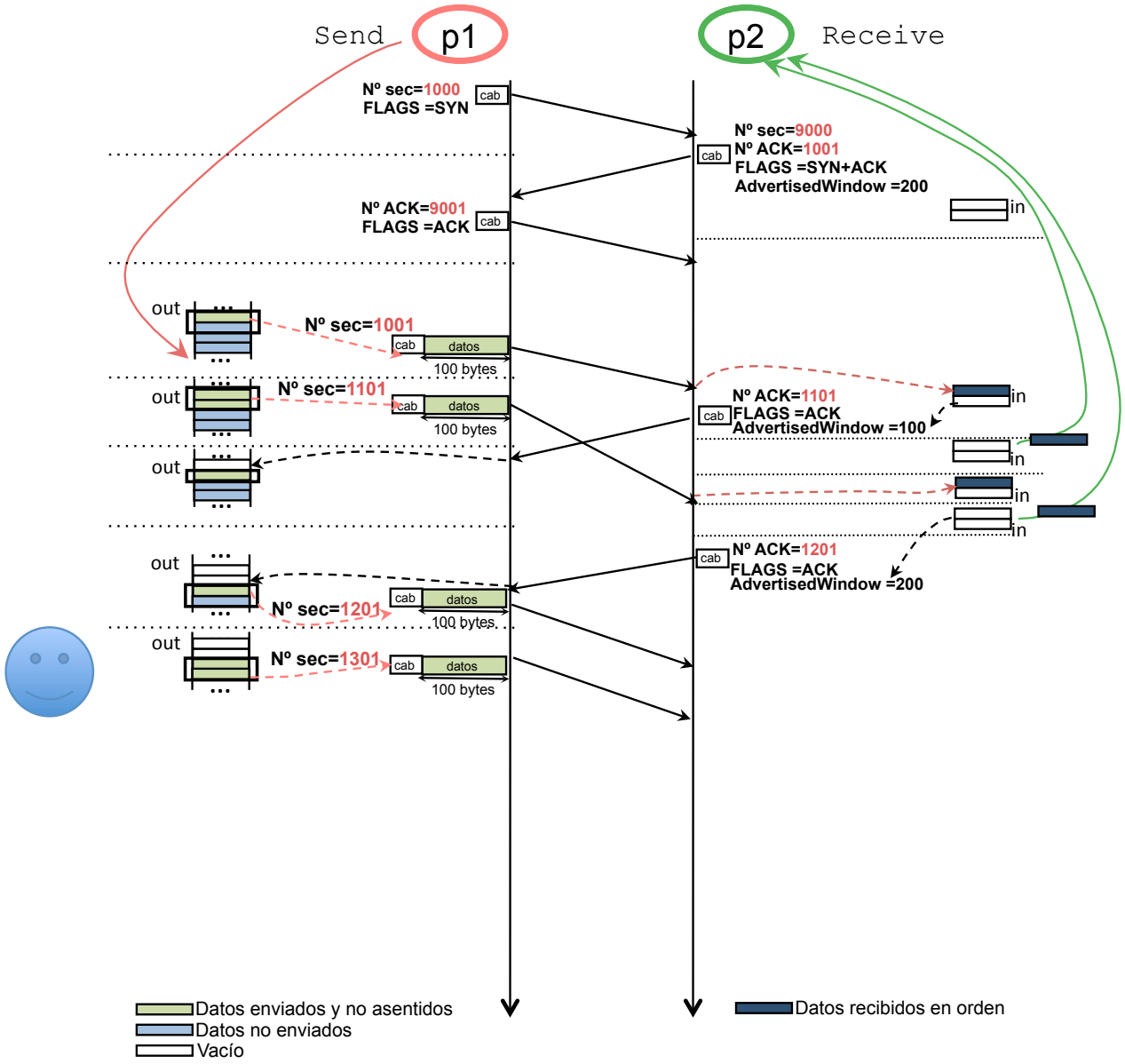
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

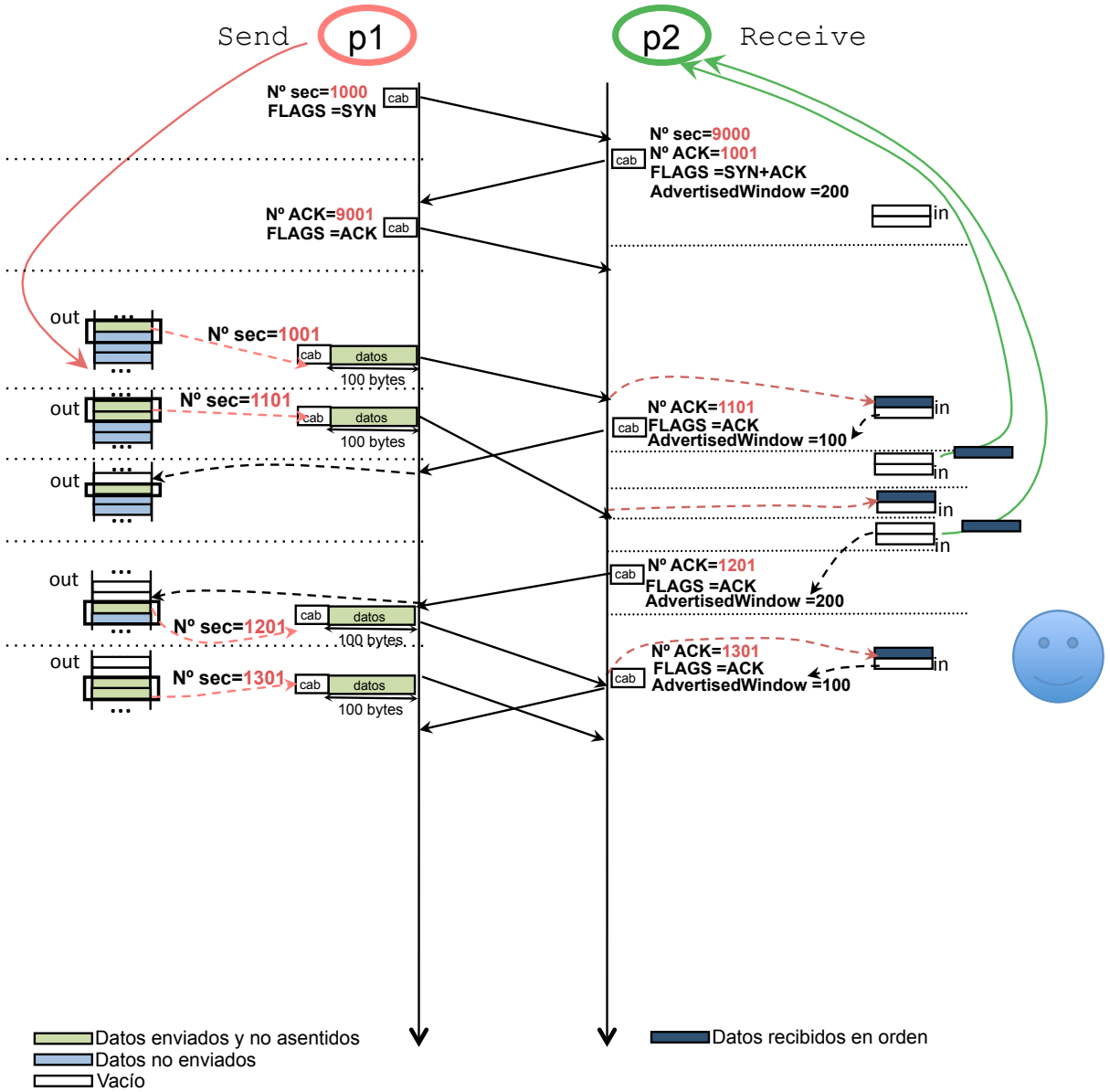
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

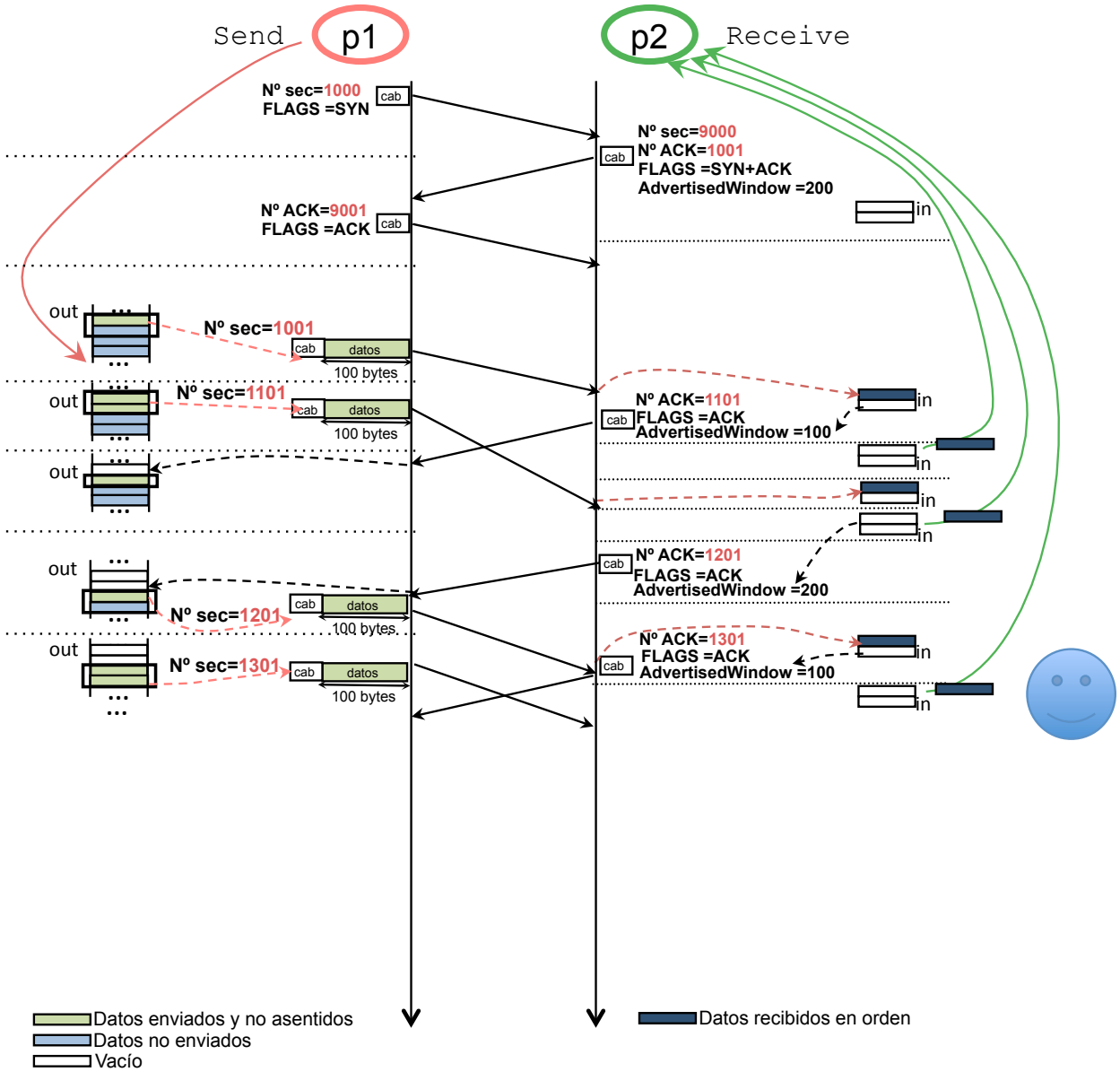
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº sec=1301
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

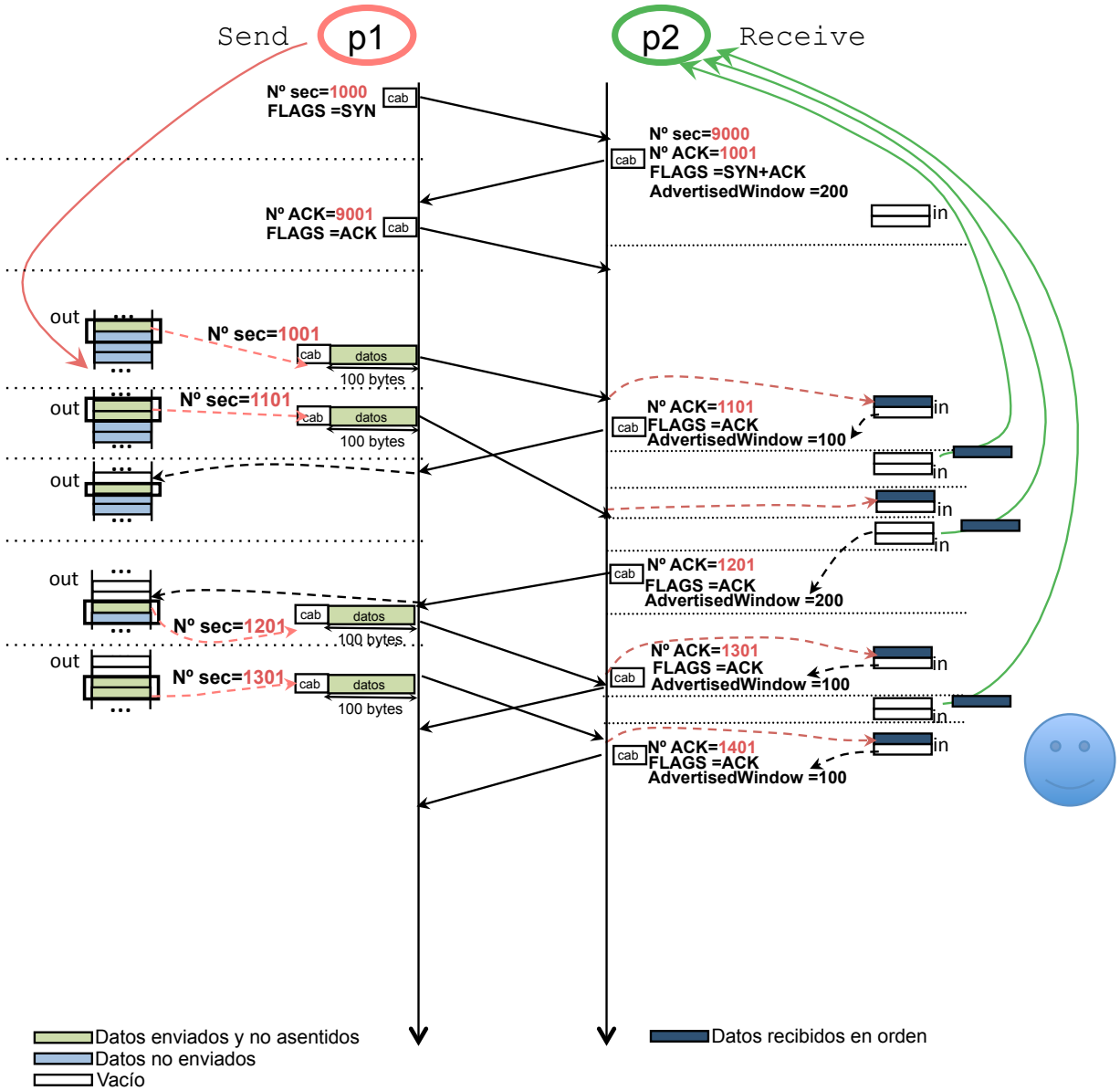
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº sec=1301
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

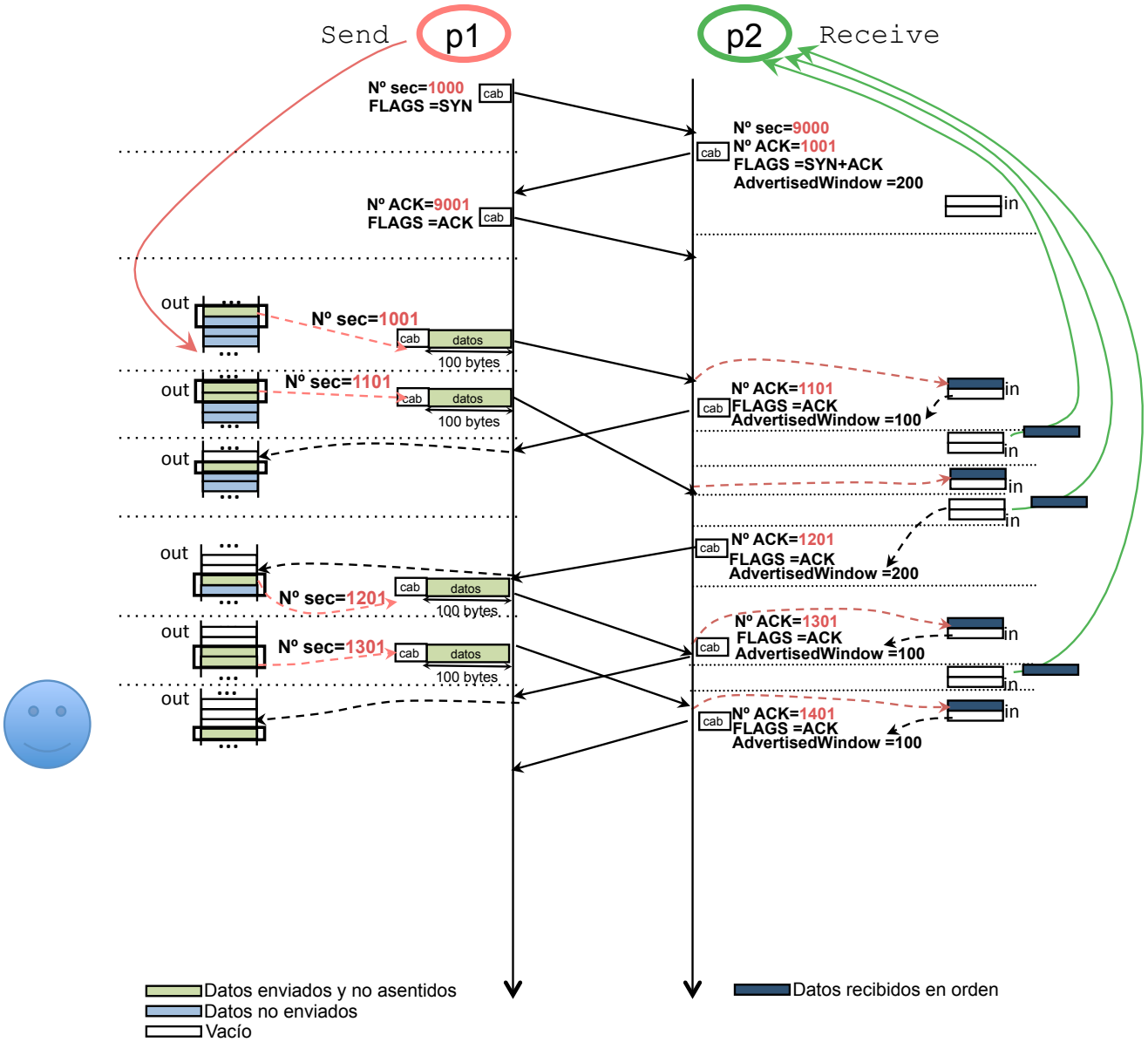
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
in
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº sec=1301
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

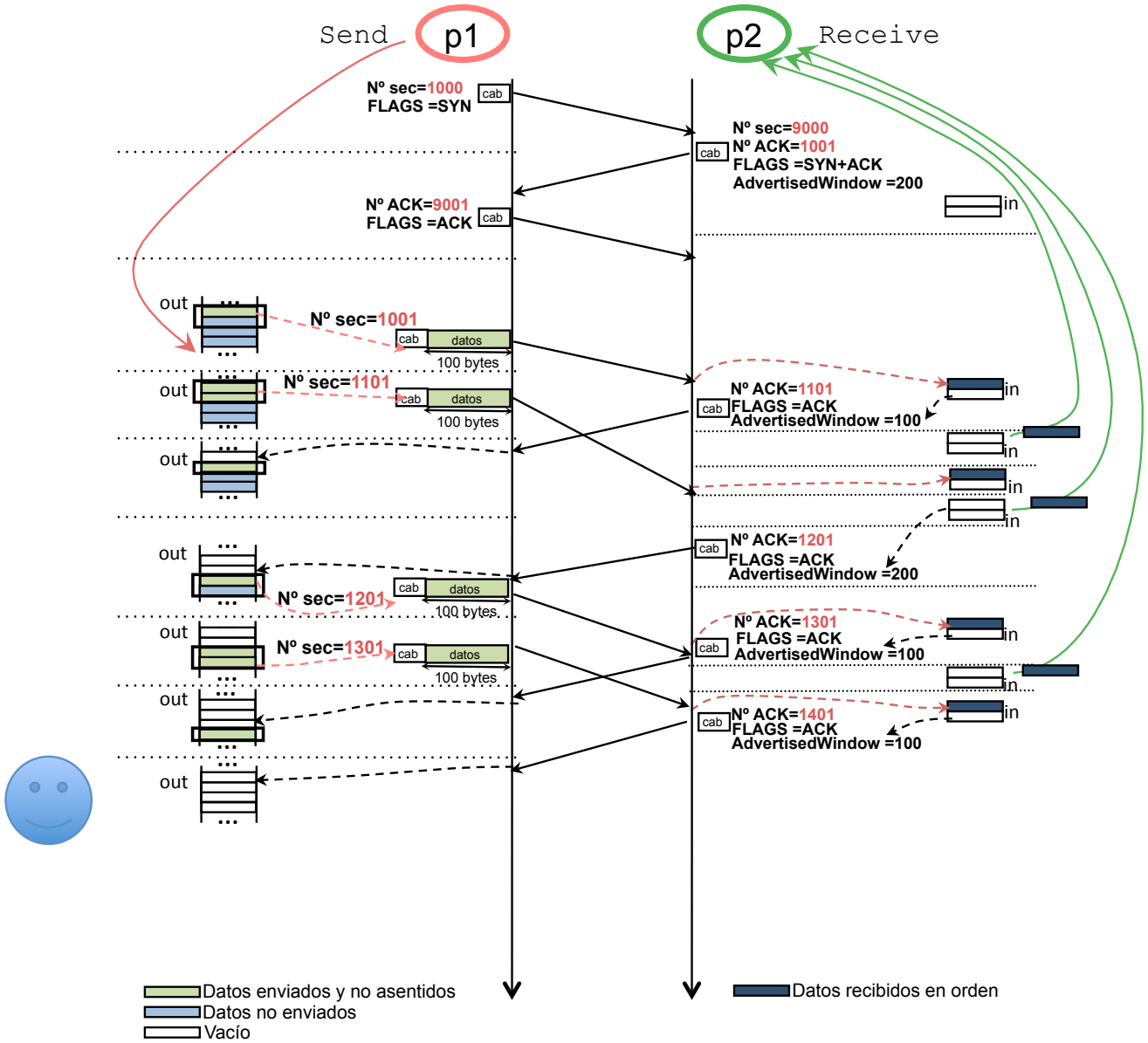
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
in
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº sec=1301
Nº ACK=1401
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

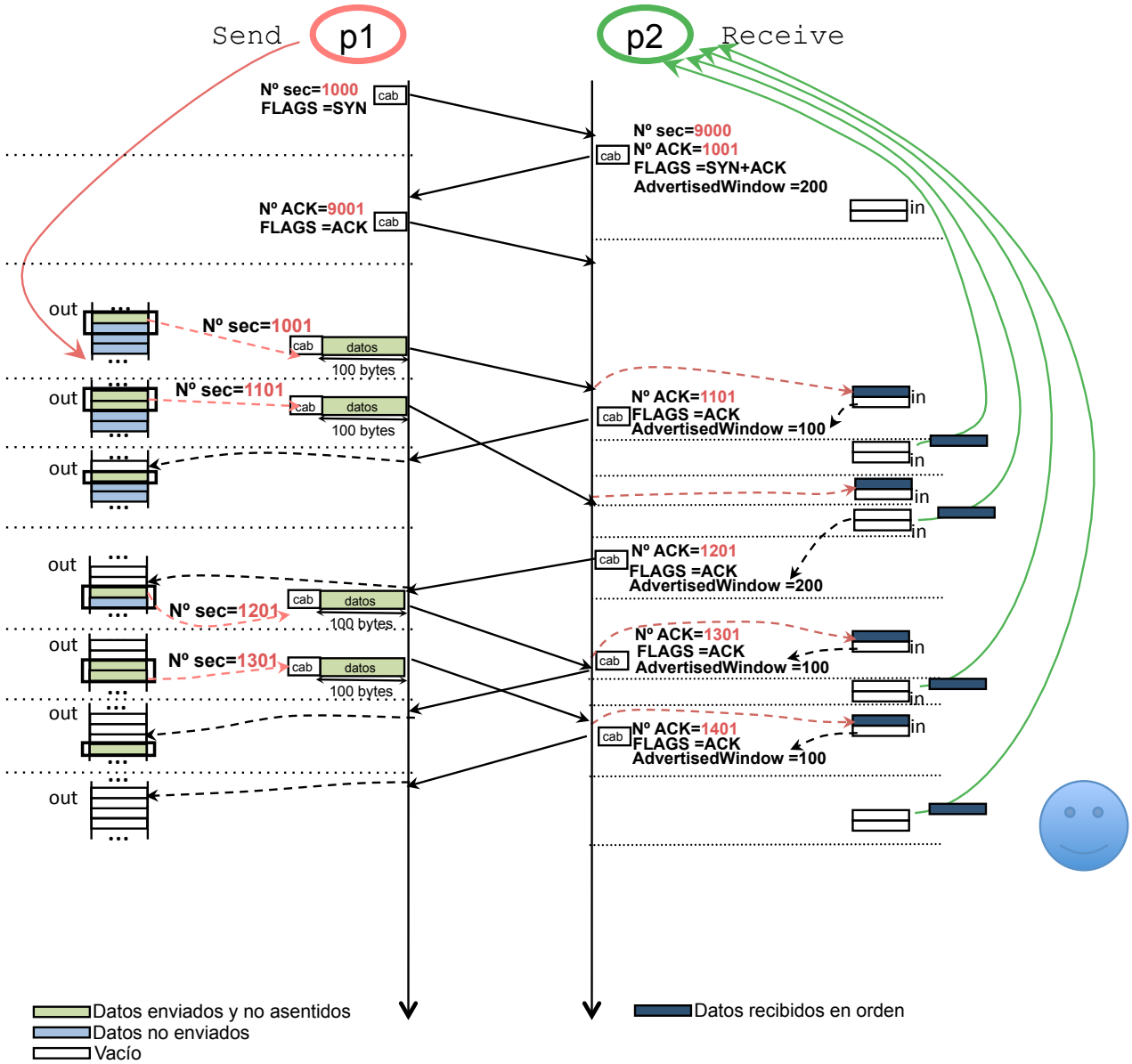
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
in
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº sec=1301
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1401
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

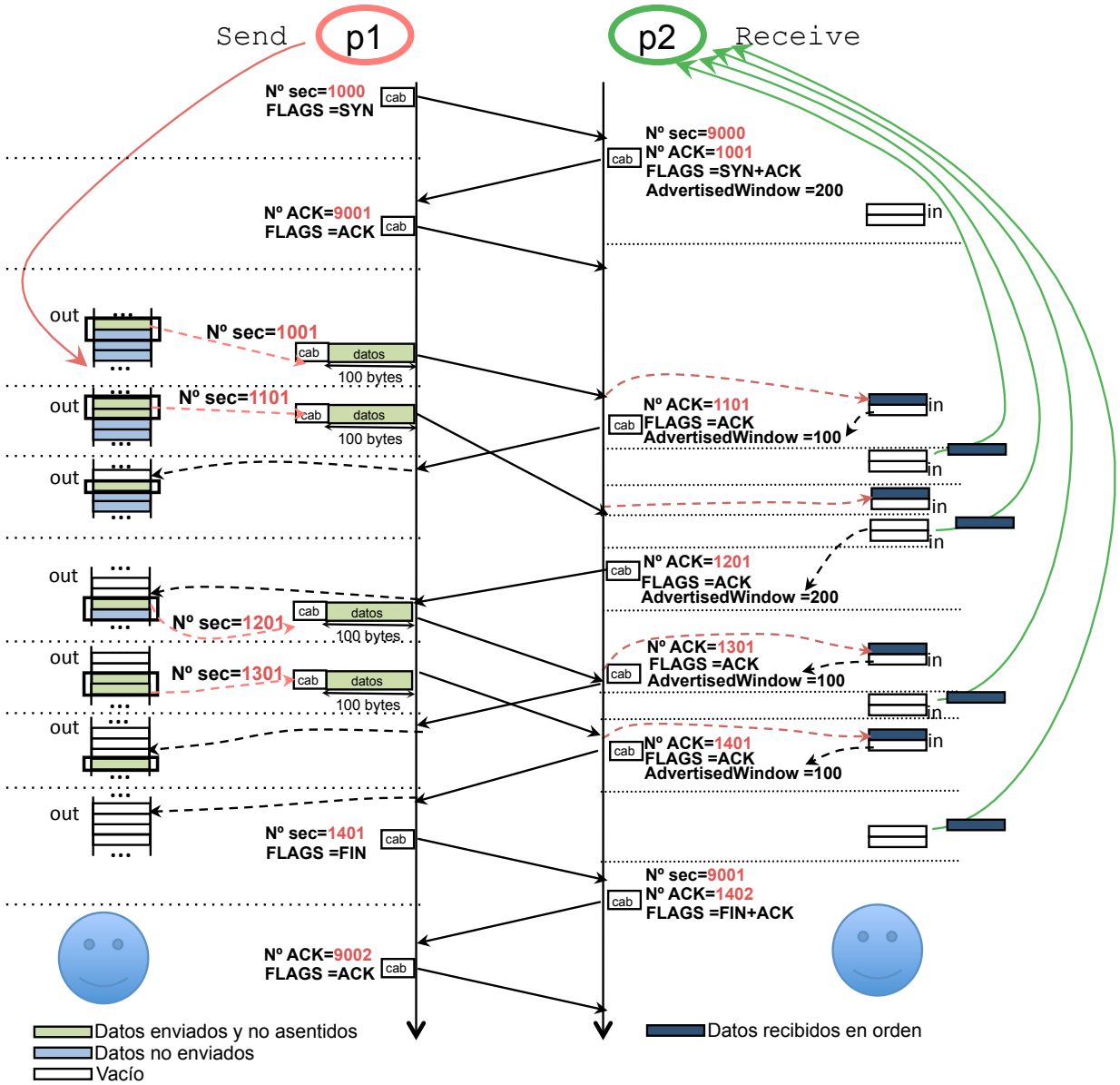
Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
in
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº sec=1301
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1401
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº sec=1101
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
in
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº sec=1301
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº ACK=1401
FLAGS =ACK
AdvertisedWindow =100
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

Ejemplo


Send
p1
p2
Receive
Nº sec=1000
FLAGS =SYN
cab
Nº sec=9000
Nº ACK=1001
FLAGS =SYN+ACK
AdvertisedWindow =200
in
Nº ACK=9001
FLAGS =ACK
out
Nº sec=1001
datos
100 bytes
Nº ACK=1101
FLAGS =ACK
AdvertisedWindow =100
Nº sec=1101
Nº ACK=1201
FLAGS =ACK
AdvertisedWindow =200
Nº sec=1201
Nº ACK=1301
FLAGS =ACK
AdvertisedWindow =100
Nº sec=1301
Nº ACK=1401
FLAGS =ACK
AdvertisedWindow =100
Nº sec=1401
FLAGS =FIN
Nº sec=9001
Nº ACK=1402
FLAGS =FIN+ACK
Nº ACK=9002
FLAGS =ACK
Datos enviados y no asentidos
Datos no enviados
Vacío
Datos recibidos en orden

Contenidos

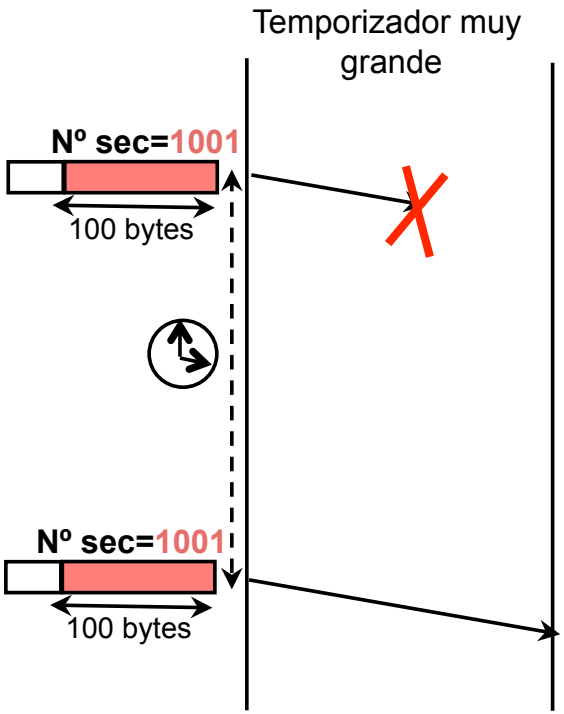
Sondas de ventana

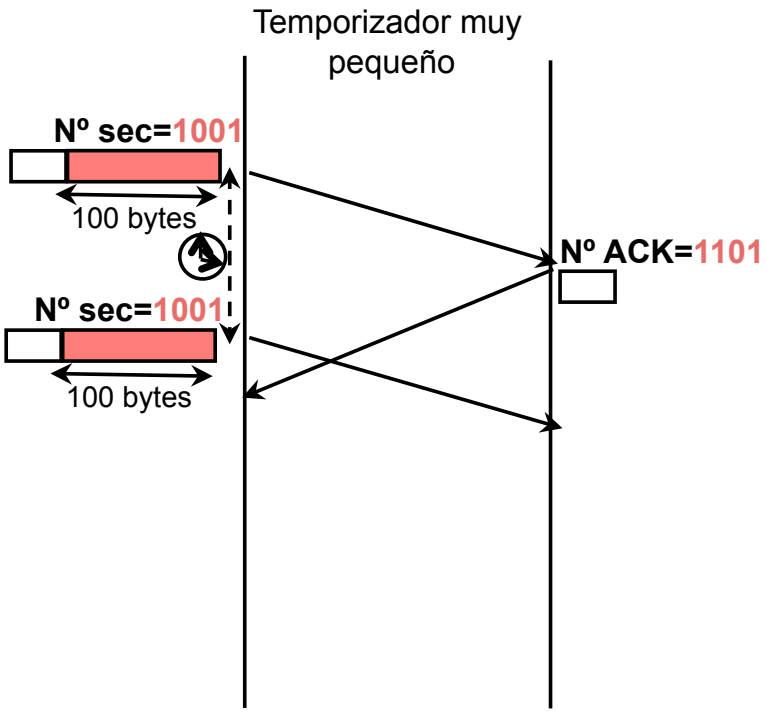
- Si la aplicación en el lado receptor no lee los datos que se están recibiendo, la implementación de TCP en el lado del receptor irá llenando su *buffer* de recepción e irá anunciando un valor de ventana (ventana de flujo) cada vez menor, incluso llegando a valer cero.
- Si el emisor recibe del receptor `AdvertisedWindow = 0`, el emisor no puede seguir enviando datos nuevos. El receptor "ha cerrado la ventana".
- En esta situación, el emisor va a enviar periódicamente un segmento con un número secuencia igual al último que tiene asentido y **longitud 0 bytes** para provocar el envío de asentimientos desde el receptor. Estos segmentos se denominan sondas de ventana.
- Cuando la aplicación en el lado receptor lea los datos que se están recibiendo el *buffer* irá vaciándose y la implementación de TCP en el lado receptor podrá guardar nuevos datos en dicho *buffer* y se lo comunicará al emisor rellenando el valor correspondiente en el campo `AdvertisedWindow` de los asentimientos que esté enviando como respuesta a las sondas de ventana.

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 ponemos?
  - Si el plazo es muy grande, puede tardarse mucho tiempo en retransmitir un segmento que se ha perdido.
  - Si el plazo es muy pequeño, puede que no de tiempo a que se reciba el ACK e innecesariamente se retransmita un segmento.

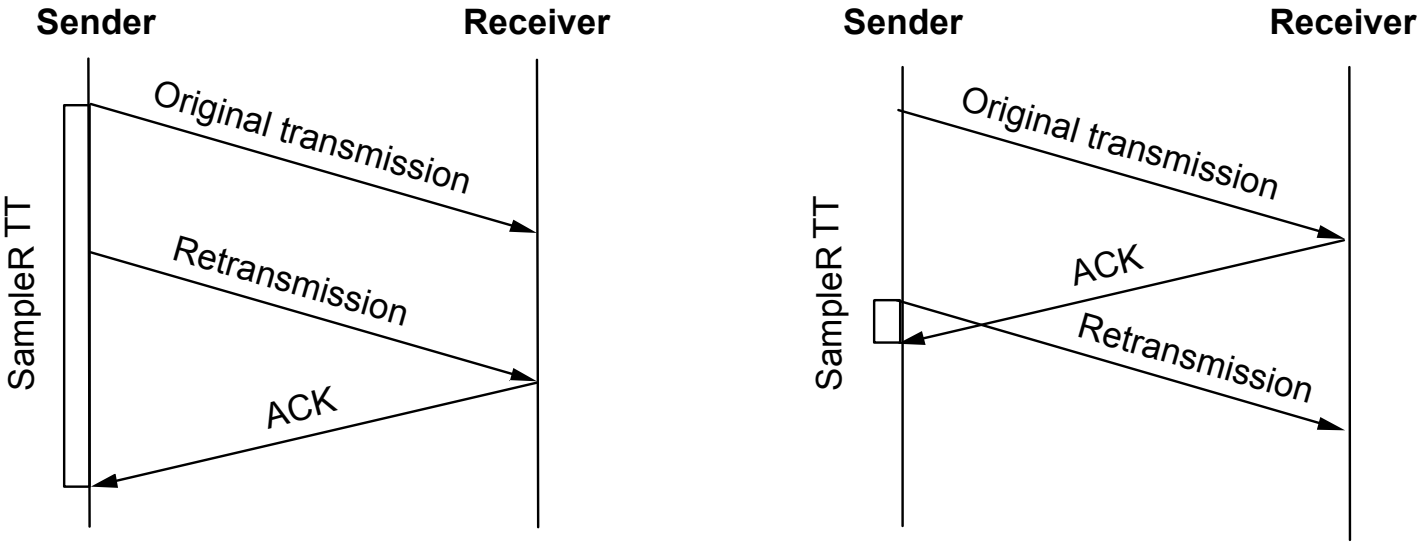




Retransmisión adaptativa

- Para cada segmento se calcula el tiempo de ronda (Round-Trip-Time, RTT): tiempo entre que se envía el segmento y se recibe el asentimiento. Se va tomando su media en el tiempo.
- Estas medidas de RTT se calculan para cada pareja (segmento / ACK)
- El Timeout se calcula en función de cada medida de RTT y la varianza de dichas medidas. Suele ser aproximadamente igual al doble del RTT.

Retransmisión adaptativa: algoritmo de Karn/Partridge



- No se toman medidas del RTT cuando se retransmite, pues no sabemos si el ACK ha venido muy rápido (dcha) o muy lento (izqda).
- *Exponential Backoff*: se dobla el *timeout* cuando se retransmite. Se supone que sólo se pierden paquetes por congestión $\rightarrow$ doblando timeout contribuimos a que decrezca.
- Es el algoritmo usado por defecto en la mayoría de implementaciones (p.ej., en Linux).

Contenidos

Extensiones de TCP

- Se implementan como opciones de la cabecera, que van en el segmento detrás 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 granularidad fina en la medición del RTT
  - 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

- 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 sería 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:
  - En el segmento `SYN` se incluye esta opción, junto con un *factor*
  - Si el receptor está dispuesto a usar esta opción, en su segmento `SYN+ACK` incluirá también esta opción junto con un *factor*.
  - A partir de entonces:
    Ventana Anunciada Real $= 2^{factor} \times AdvertisedWindow$
    - 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:

    MSS = 1500 - 20 (cabecera IP sin opciones) - 20 (cabecera TCP sin opciones) = 1460 bytes

- Cada lado de una conexión, por su parte, hace segmentos de tamaño menor o igual a su MSS.
- Si se usa la opción de TCP, en el segmento SYN cada lado incluye su MSS para indicar al otro lado que haga segmentos de tamaño menor o igual a ese valor.
- Si ambos lados de la conexión usan esta opción, la consecuencia es que ambos lados usarán segmentos con la parte de datos de tamaño igual al menor de los dos MSS.

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