TCP:
Control de Congestión

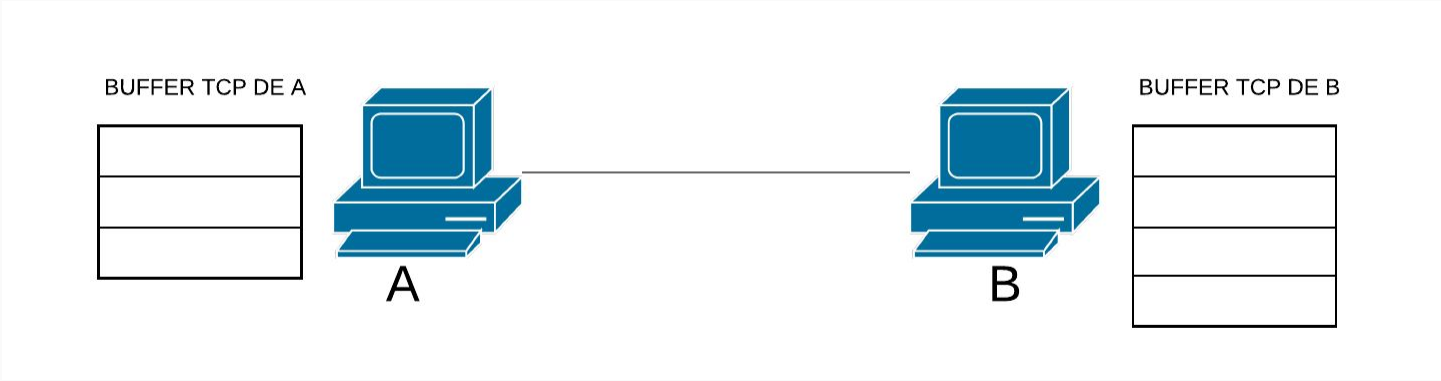
Introducción a los Sistemas Distribuidos (75.43)

Universidad de Buenos Aires, Facultad de Ingeniería

Control de flujo

Control de flujo

- Cada extremo de la transmisión TCP tiene su propio buffer.
- TCP almacena datos en el buffer para pasarlos en orden a la capa de aplicación.
- El buffer tiene un tamaño finito.



Control de flujo

¿Qué pasa si se reciben más datos de los que la capa de aplicación es capaz de consumir?

Control de flujo

¿Qué pasa si se reciben más datos de los que la capa de aplicación es capaz de consumir?

BUFFER OVERFLOW

Control de flujo

- Para evitar el buffer overflow, **cada extremo** de la transmisión tiene que comunicar la cantidad de datos que puede recibir.
- A esto se lo llama ***rwnd*** (Receiver Window o Advertised Window).
- Se utiliza el campo Window del TCP header para comunicar este valor.

Control de flujo

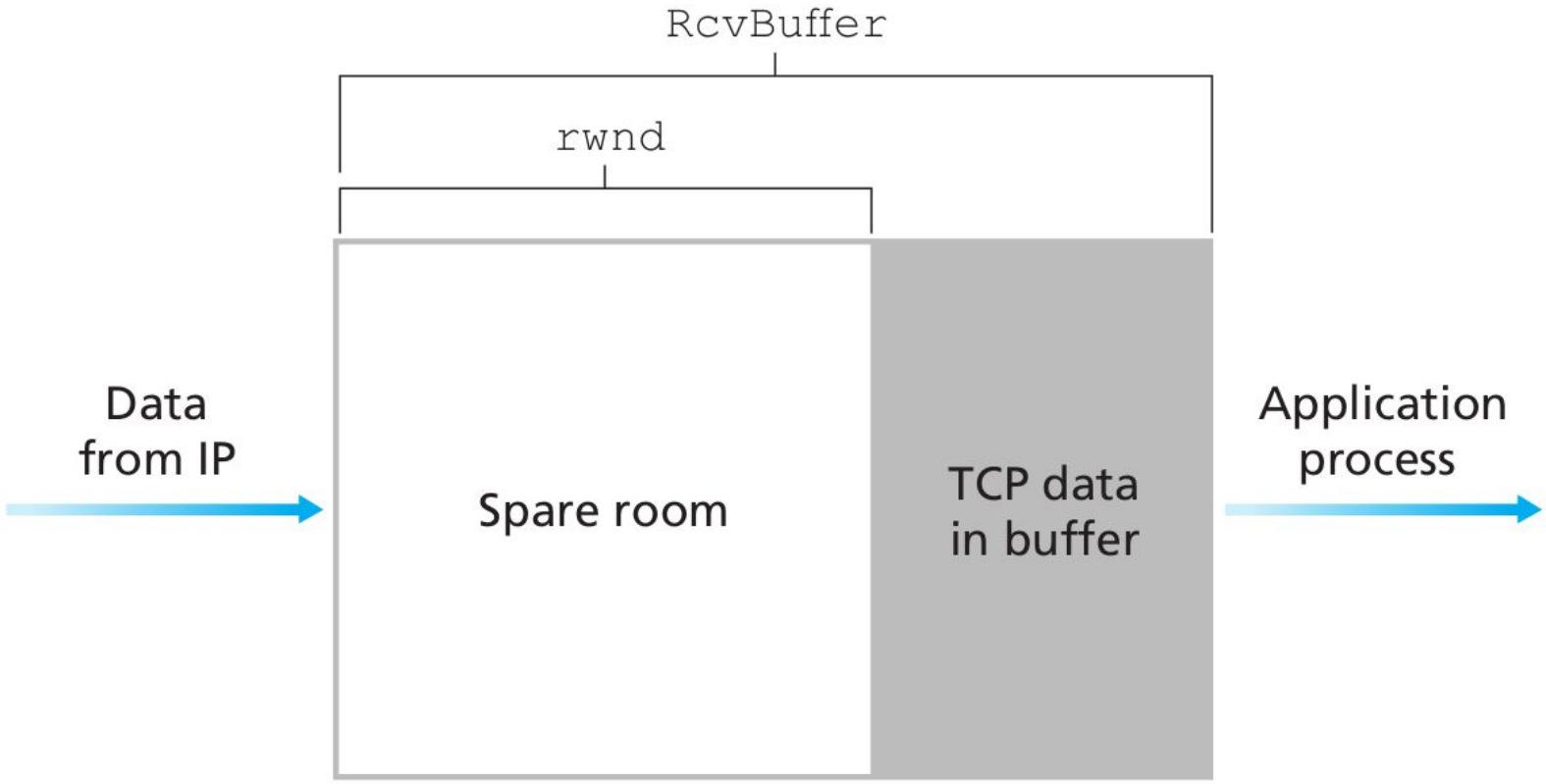


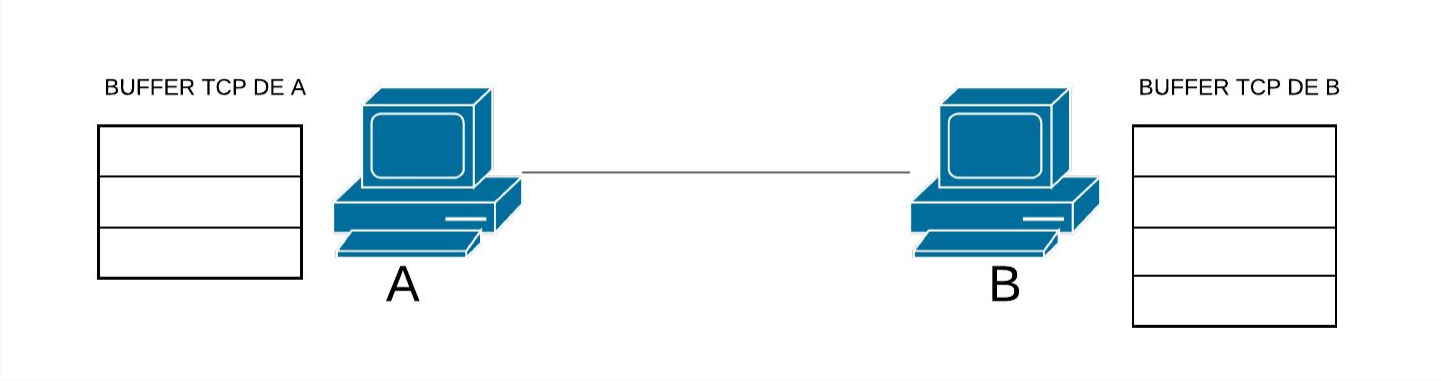
Figure 3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

Control de flujo

- Cuando se recibe un cambio en ***rwnd***, el emisor debe modificar su tasa de envío para no causar buffer overflow, pero tampoco subutilizar el buffer.
- Si se recibe ***rwnd*** = 0, el emisor debe dejar de enviar datos hasta recibir un nuevo valor de ***rwnd***

Control de flujo

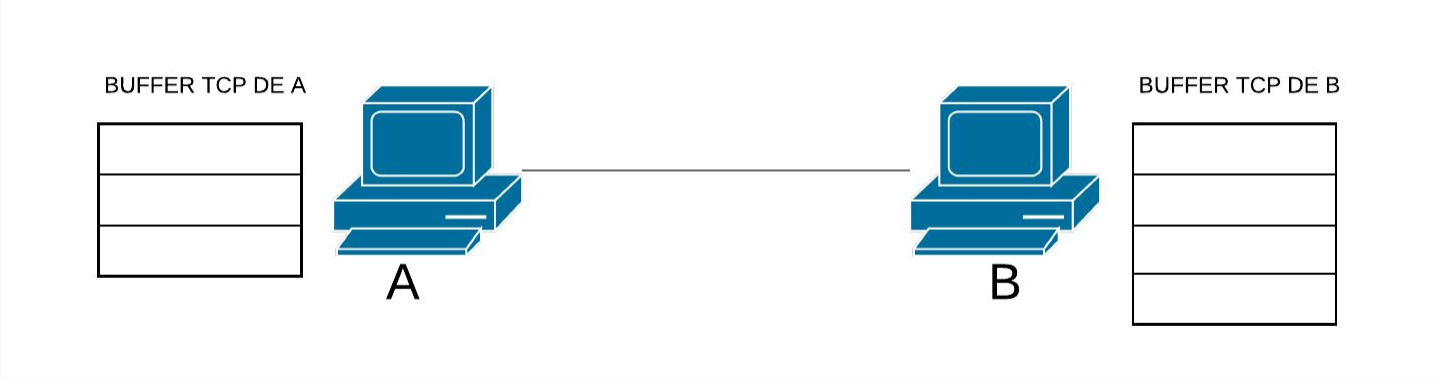
- Imaginemos el siguiente escenario
  - A envía un paquete a B
  - El buffer de B se llena, por lo que envía un ACK ***rwnd*** = 0 a A.
  - A deja de enviar datos.



Control de flujo

- La capa de aplicación en B lee del buffer y lo libera.

¿Cómo hace B para avisarle a A que puede recibir más datos?



Control de flujo

- Para lograr esto, luego de un período de inactividad A continuará enviando paquetes de 1 byte a B.
- Cuando B los recibe, indica el tamaño de ventana actual.
- Cuando el buffer se libere, B indicará ***rwnd*** > 0 y se reanudará la transmisión.

Control de congestión

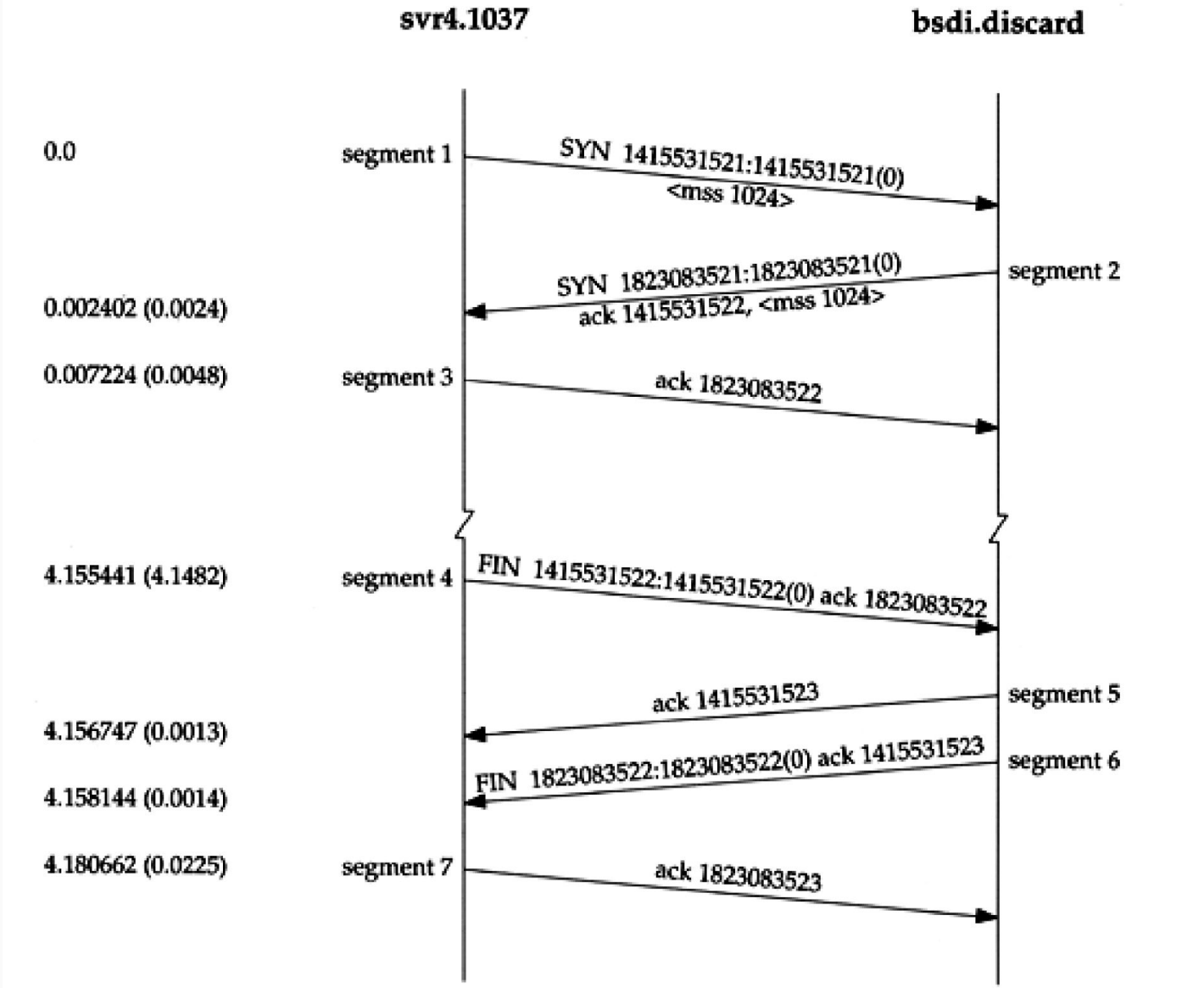
TCP : control de congestión

- El control de congestión **no formaba parte** del diseño original de TCP

TCP : control de congestión

- Fue una respuesta a la **congestión persistente** observada en la red
- La **congestión** es producto de múltiples flujos TCP en simultáneo sobre la misma red

TCP : control de congestión



Algoritmos de control de congestión

TCP : control de congestión

- La congestión aparece principalmente a causa del **tamaño finito** de los buffers de los **dispositivos intermedios** de capas inferiores.
- Cuando se llena un buffer, se empiezan a **descartar** los segmentos -> se dice que la red está **congestionada**

TCP : control de congestión

- **cwnd** : es la máxima cantidad de bytes que puede haber en vuelo

- **RECORDAR:**

  LastByteSent-LastByteACKed < **min(cwnd,rwnd)**

TCP : control de congestión

- Tiene distintas etapas:
  - Slow start
  - Congestion avoidance
  - Fast Retransmit
  - Fast Recovery

TCP : control de congestión

Slow start

- cwnd(n+1) = cwnd(n) + MSS * #(ACK)
- cwnd medido en Bytes

TCP : control de congestión

Slow start

- cwnd(n+1) = cwnd(n) + #(ACK)
- cwnd medido en **MSSs** (segmentos de tamaño máximo)

TCP : control de congestión

Slow start

- cwnd(n+1) = cwnd(n) + #(ACK)
- cwnd medido en **MSSs** (segmentos de tamaño máximo)
- Crecimiento exponencial

TCP : control de congestión

Slow start

cwnd(n+1) = cwnd(n) + #(ACK)

- Ejemplo: si cwnd(0) = 1
- cwnd(0) = 1, envío 1 segmento

TCP : control de congestión

Slow start

cwnd(n+1) = cwnd(n) + #(ACK)

- Ejemplo: si cwnd(0) = 1
- cwnd(0) = 1, envío 1 segmento
- Recibo 1 ACK
- cwnd(1) = ?

TCP : control de congestión

Slow start

cwnd(n+1) = cwnd(n) + #(ACK)

- Ejemplo: si cwnd(0) = 1
- cwnd(0) = 1, envío 1 segmento
- Recibo 1 ACK
- cwnd(1) = 1+ #(ACK) = 1 + 1 = 2

TCP : control de congestión

Slow start

cwnd(n+1) = cwnd(n) + #(ACK)

- Ejemplo: si cwnd(0) = 1
- cwnd(0) = 1, envío 1 segmento
- Recibo 1 ACK
- cwnd(1) = 1+ #(ACK) = 1 + 1 = 2
- Envío 2 segmentos. Recibo 2 ACK
- cwnd(2) = ?

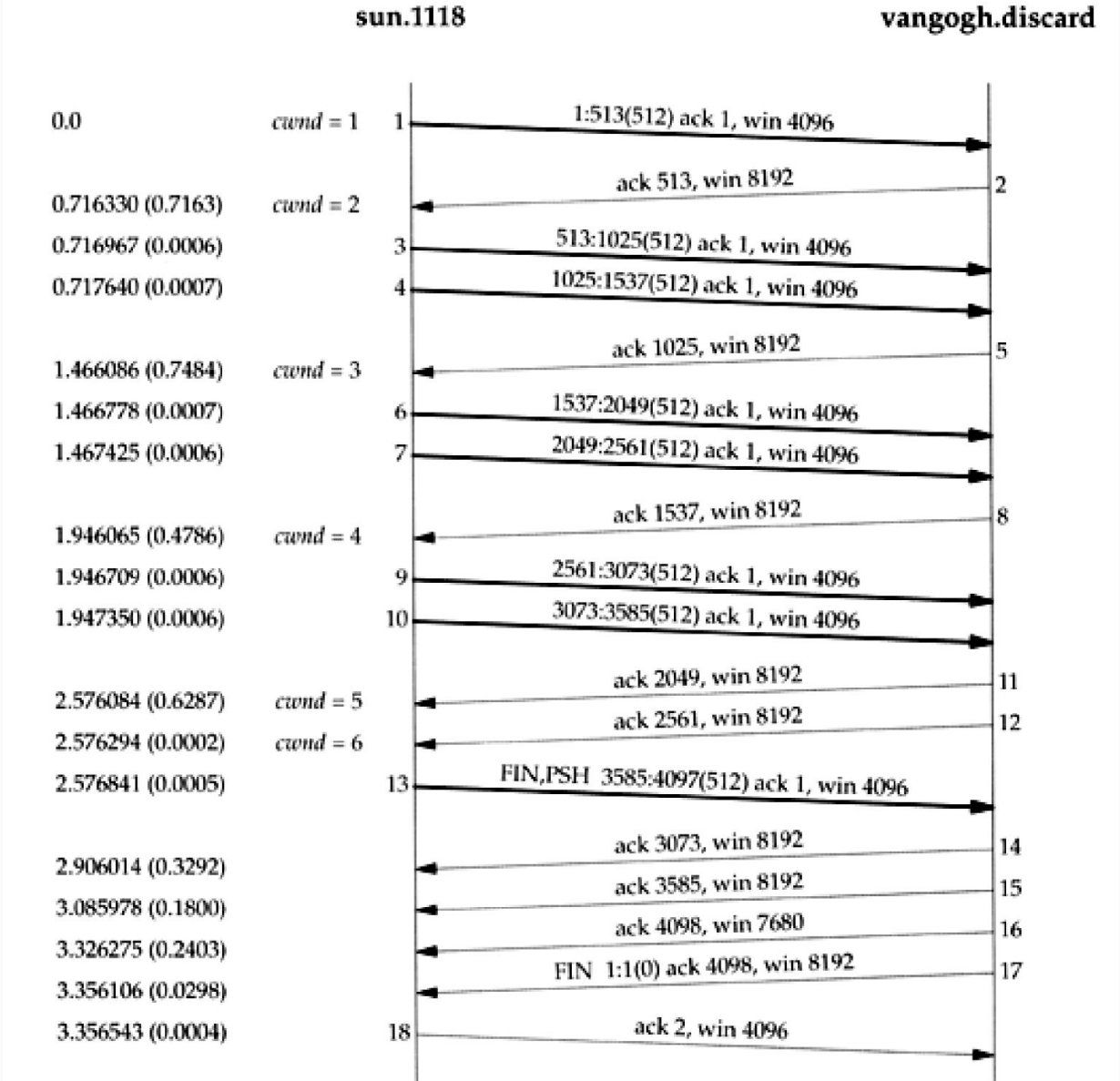
TCP : control de congestión

Slow start

cwnd(n+1) = cwnd(n) + #(ACK)

- Ejemplo: si cwnd(0) = 1
- cwnd(0) = 1, envío 1 segmento
- Recibo 1 ACK
- cwnd(1) = 1+ #(ACK) = 1 + 1 = 2
- Envío 2 segmentos. Recibo 2 ACK
- cwnd(2) = 2 + #(ACK) = 2 + 2 = 4

TCP : control de congestión



TCP : control de congestión

¿Seguimos incrementando la ventana exponencialmente para siempre?

TCP : control de congestión

Cambio de etapa

- En un determinado momento, se dará por terminada la etapa de **Slow Start** para pasar a la próxima, llamada **Congestion Avoidance**
- Esto viene dado por el valor de **sstresh** (slow start threshold size), qué es configurable.

TCP : control de congestión

Congestion avoidance

- cwnd(n+1) = cwnd(n) + #(ACK)/cwnd(n)
- Aumenta 1 cuando nos llegan los ACKs de toda la ráfaga
- cwnd siempre tiene que ser entero (la unidad es el **MSS**)
- Se redondea para abajo!

TCP : control de congestión

Congestion avoidance

cwnd(n+1) = cwnd(n) + #(ACK)/cwnd(n)

- Ejemplo: si cwnd(n) = 2
- cwnd(n) = 2, envío 2 segmentos

TCP : control de congestión

Congestion avoidance

cwnd(n+1) = cwnd(n) + #(ACK)/cwnd(n)

- Ejemplo: si cwnd(n) = 2
- cwnd(n) = 2, envío 2 segmentos
- Llega 1 ACK
- cwnd(n+1) = 2 + 1 / 2 = 2

TCP : control de congestión

Congestion avoidance

cwnd(n+1) = cwnd(n) + #(ACK)/cwnd(n)

- Ejemplo: si cwnd(n) = 2
- cwnd(n) = 2, envío 2 segmentos
- Llega 1 ACK
- cwnd(n+1) = 2 + 1 / 2 = 2
- Llega el 2do ACK
- cwnd(n+2) = 2 + 2 / 2 = 3

TCP : control de congestión

¿Seguimos en CA para siempre, felizmente incrementando en #(ACK)/cwnd(n) por toda la eternidad?

TCP : control de congestión

¿Qué pasa si se pierden paquetes?

TCP : control de congestión

Pérdida por timeout (RTO)

- ssthresh = cwnd(n) / 2
- cwnd(n+1) = 1 (1 es LW [loss window])
- Empieza slow start de nuevo
- Ver ejemplo

TCP : control de congestión

En una conexión TCP recién establecida (IW = 2 MSS, SSTHRESH = 64 KB) con RTT=200 ms y MSS=2 KB, el host receptor siempre anuncia una AdvertisedWindow de 16 KB. La red está cargada al punto que si una ráfaga fuera de 16 KB o más, se perderían todos los segmentos de la misma. **¿Cuántos rounds se demora en enviar un archivo de 36 KB?**

Este ejercicio fue tomado de la materia “Teoría de Comunicaciones”, de la carrera de Licenciatura en Ciencias de la Computación de la Facultad de Ciencias Exactas y Naturales de la UBA. Los autores del ejercicio son Esteban Lanzarotti y Guido Chiari. El ejercicio fue levemente modificado.

TCP : control de congestión

En una conexión TCP recién establecida (IW = 2 MSS, SSTHRESH = 64 KB) con RTT=200 ms y MSS=2 KB, el host receptor siempre anuncia una AdvertisedWindow de 16 KB. La red está cargada al punto que si una ráfaga fuera de 16 KB o más, se perderían todos los segmentos de la misma. **¿Cuántos rounds se demora en enviar un archivo de 36 KB?**

¿Qué datos son importantes?

Este ejercicio fue tomado de la materia “Teoría de Comunicaciones”, de la carrera de Licenciatura en Ciencias de la Computación de la Facultad de Ciencias Exactas y Naturales de la UBA. Los autores del ejercicio son Esteban Lanzarotti y Guido Chiari. El ejercicio fue levemente modificado.

TCP : control de congestión

En una conexión TCP recién establecida (IW = 2 MSS, SSTHRESH = 64 KB) con RTT=200 ms y MSS=2 KB, el host receptor siempre anuncia una AdvertisedWindow de 16 KB. La red está cargada al punto que si una ráfaga fuera de 16 KB o más, se perderían todos los segmentos de la misma. **¿Cuántos rounds se demora en enviar un archivo de 36 KB?**

Este ejercicio fue tomado de la materia “Teoría de Comunicaciones”, de la carrera de Licenciatura en Ciencias de la Computación de la Facultad de Ciencias Exactas y Naturales de la UBA. Los autores del ejercicio son Esteban Lanzarotti y Guido Chiari. El ejercicio fue levemente modificado.

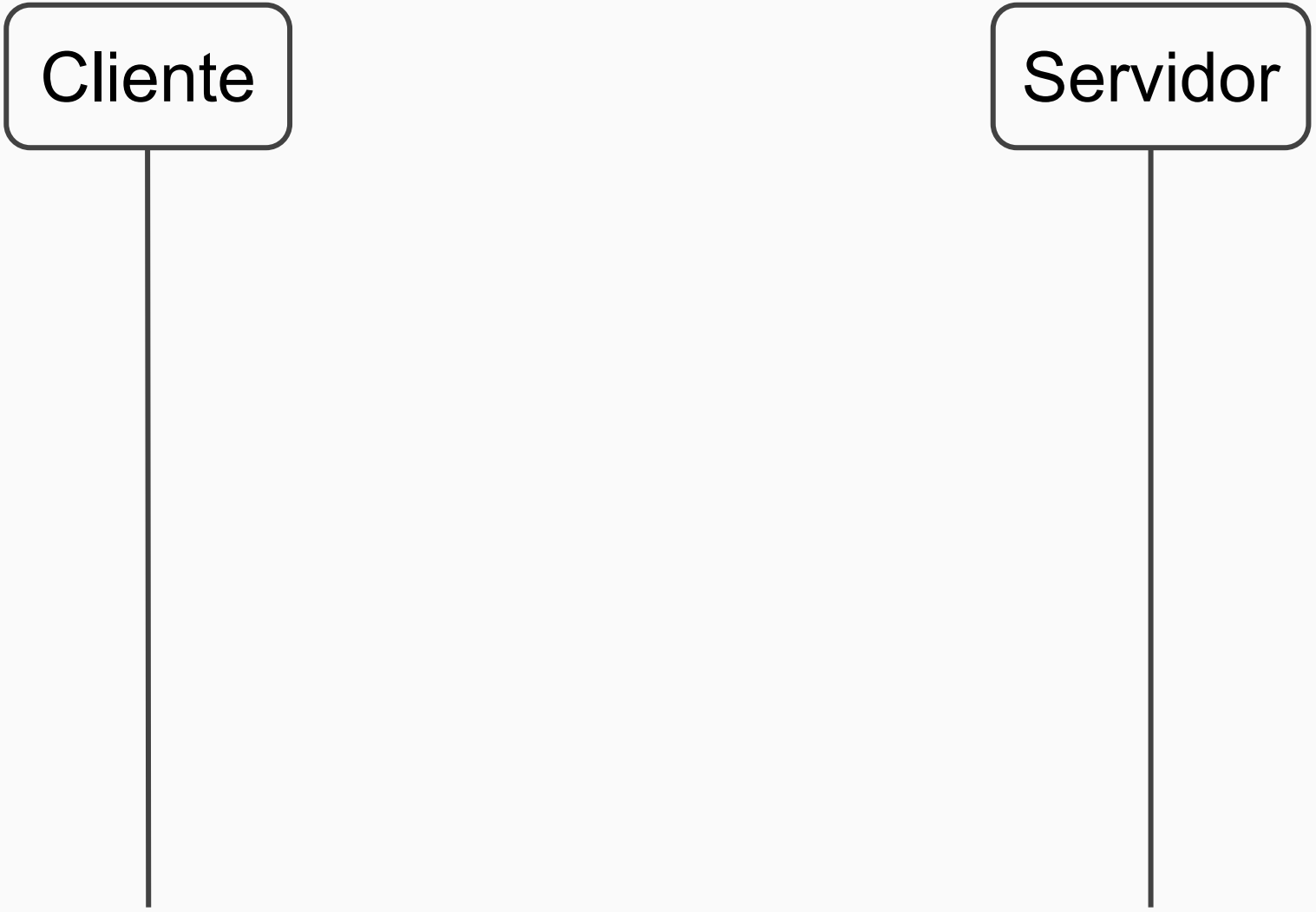
TCP : control de congestión

En una conexión TCP recién establecida (IW = **2 MSS**, SSTHRESH = **32 MSS**) con RTT=200 ms y MSS=2 KB, el host receptor siempre anuncia una AdvertisedWindow de **8 MSS**. La red está cargada al punto que si una ráfaga fuera de **8 MSS** o más, se perderían todos los segmentos de la misma. **¿Cuántos rounds se demora en enviar un archivo de 18 MSS?**

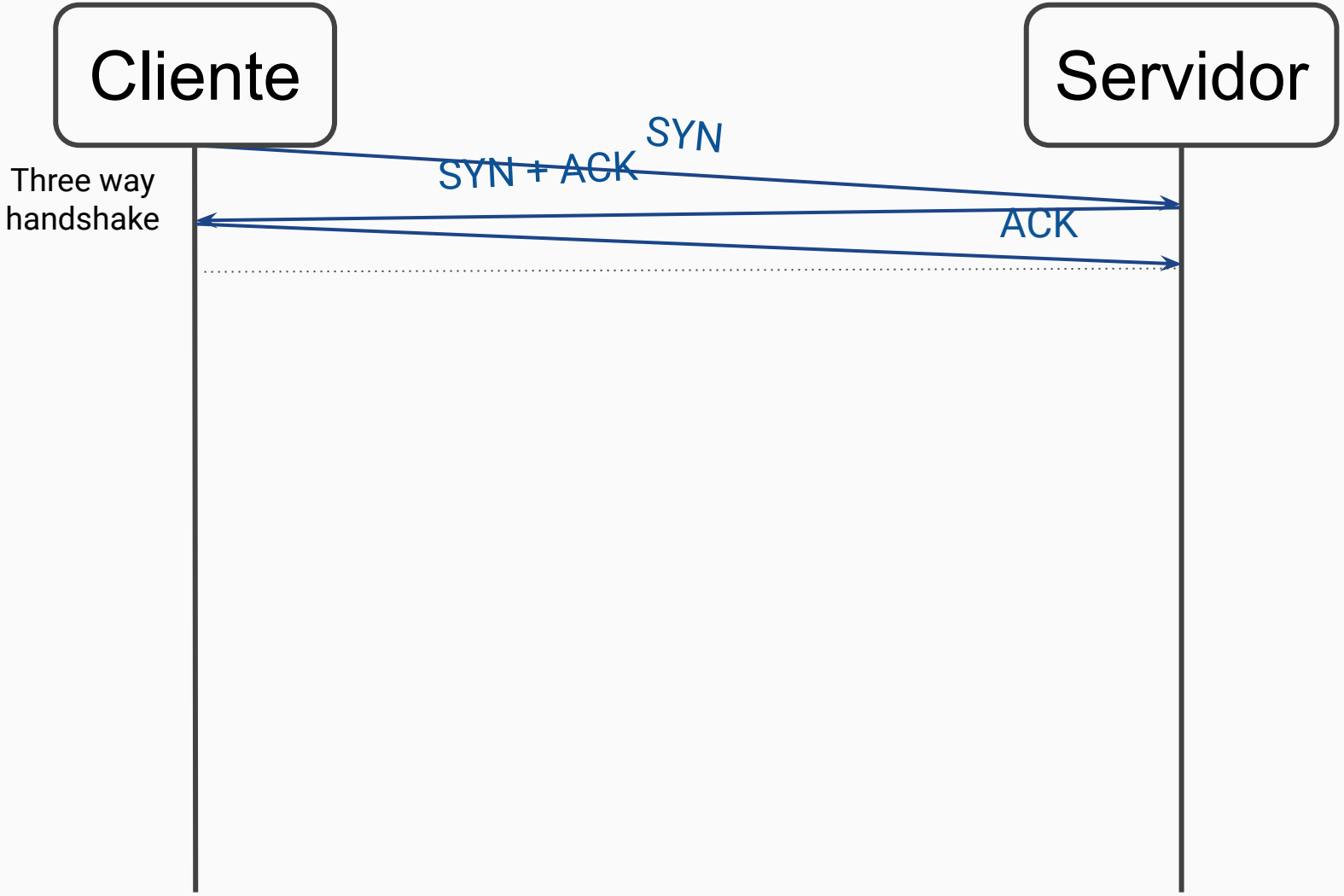
Convertimos los datos a MSSs

Este ejercicio fue tomado de la materia “Teoría de Comunicaciones”, de la carrera de Licenciatura en Ciencias de la Computación de la Facultad de Ciencias Exactas y Naturales de la UBA. Los autores del ejercicio son Esteban Lanzarotti y Guido Chiari. El ejercicio fue levemente modificado.

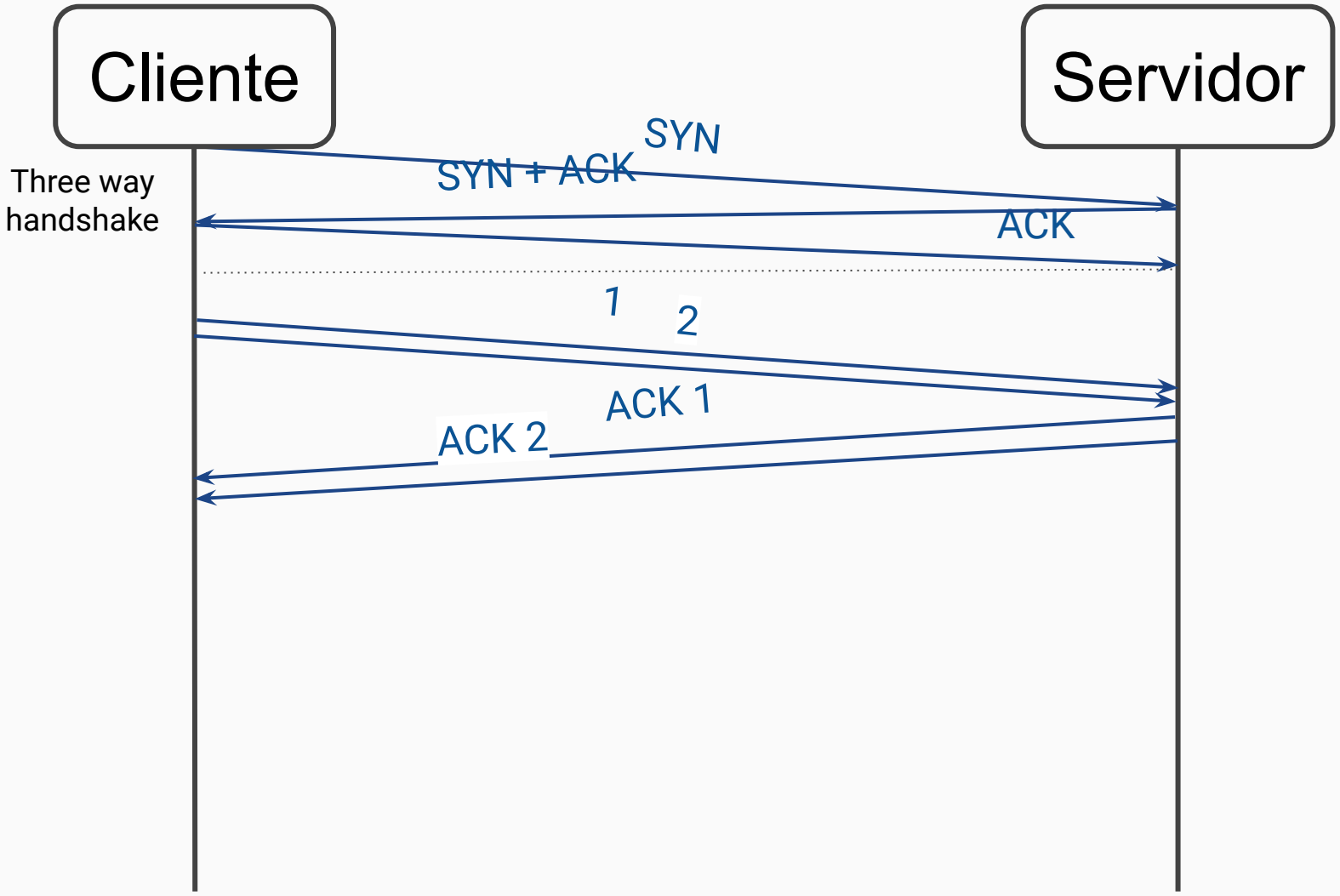
Resolución



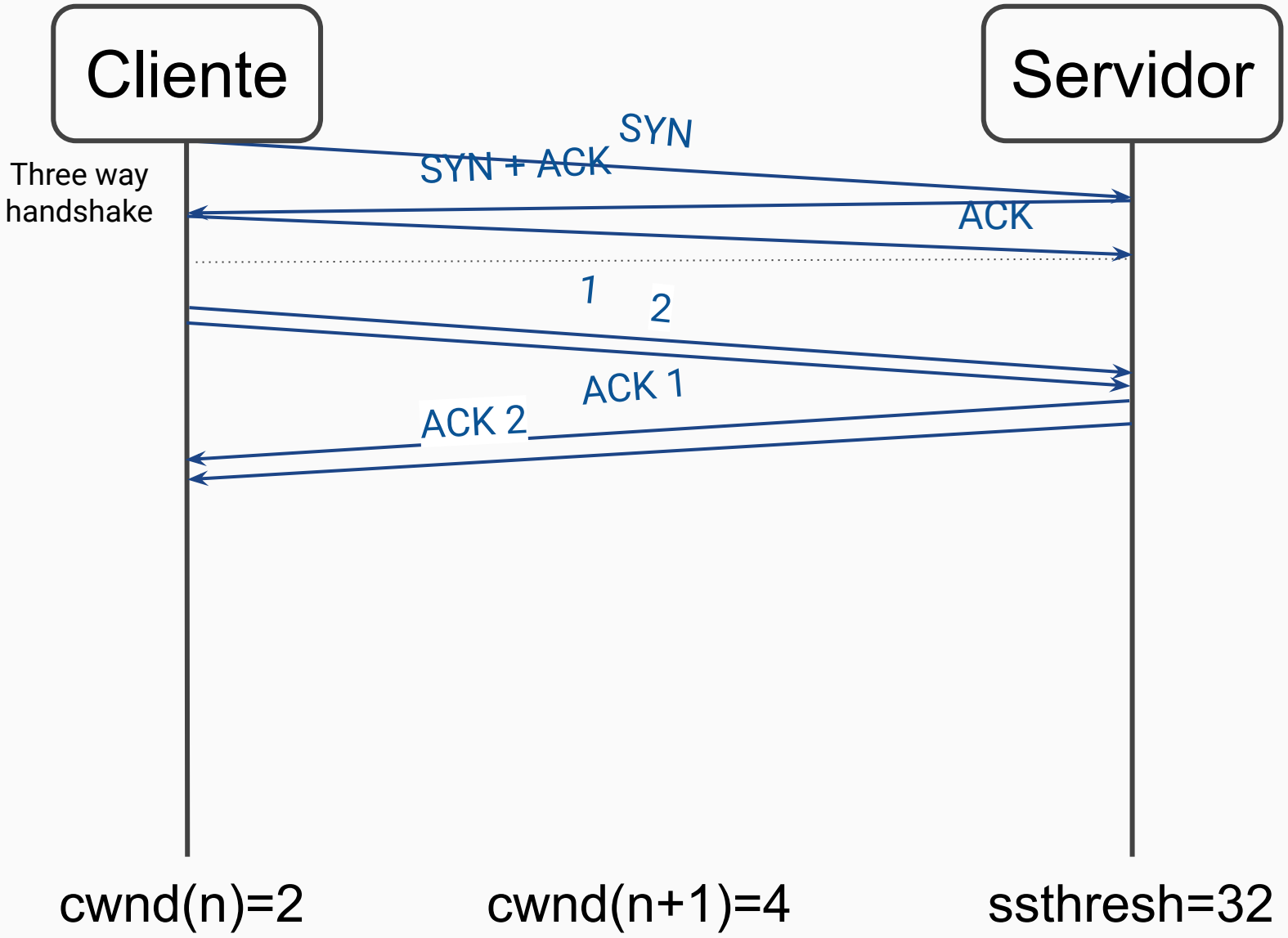
Resolución



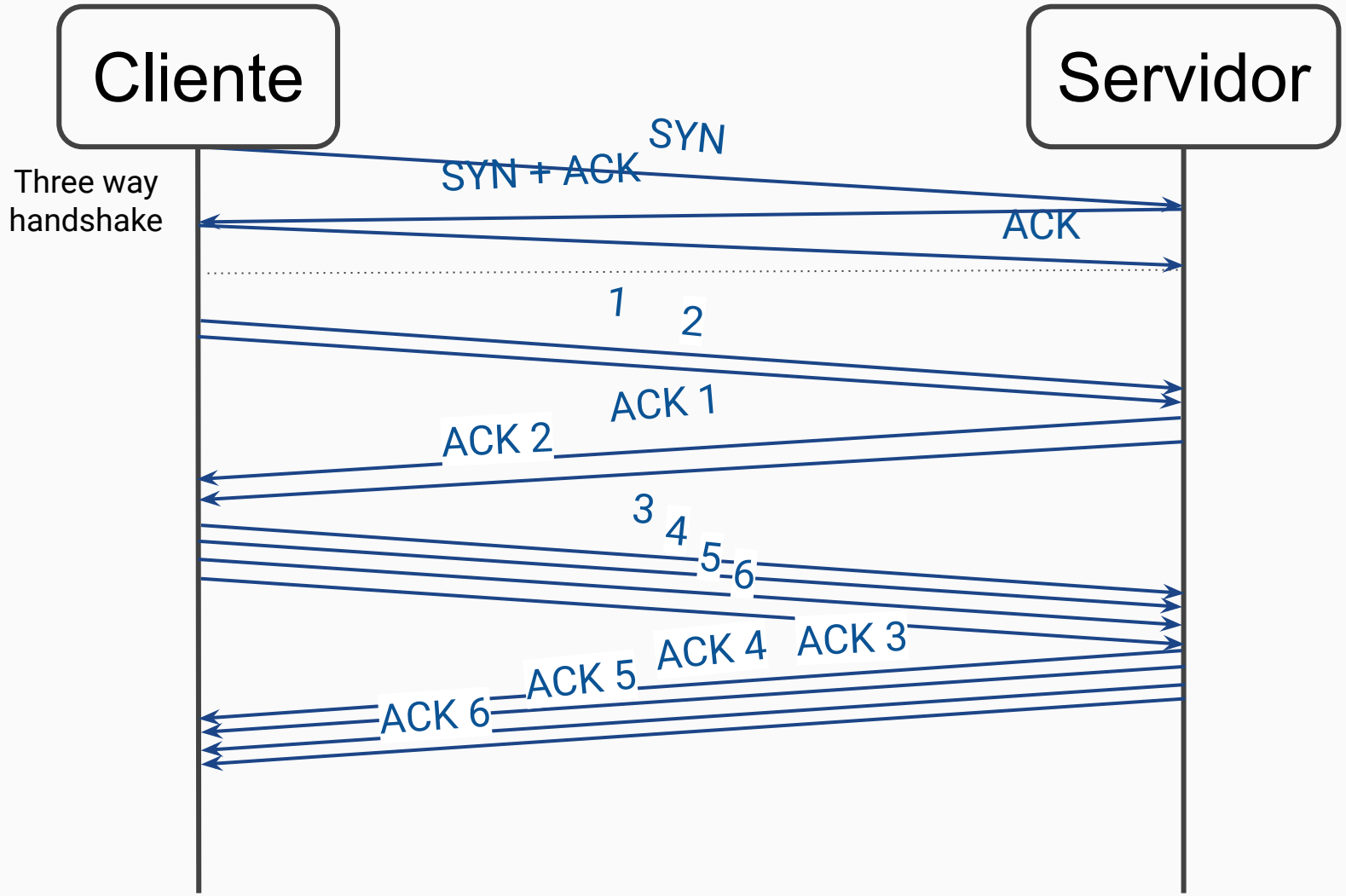
Resolución


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
cwnd(n)=2
cwnd(n+1)=?
ssthresh=32

Resolución

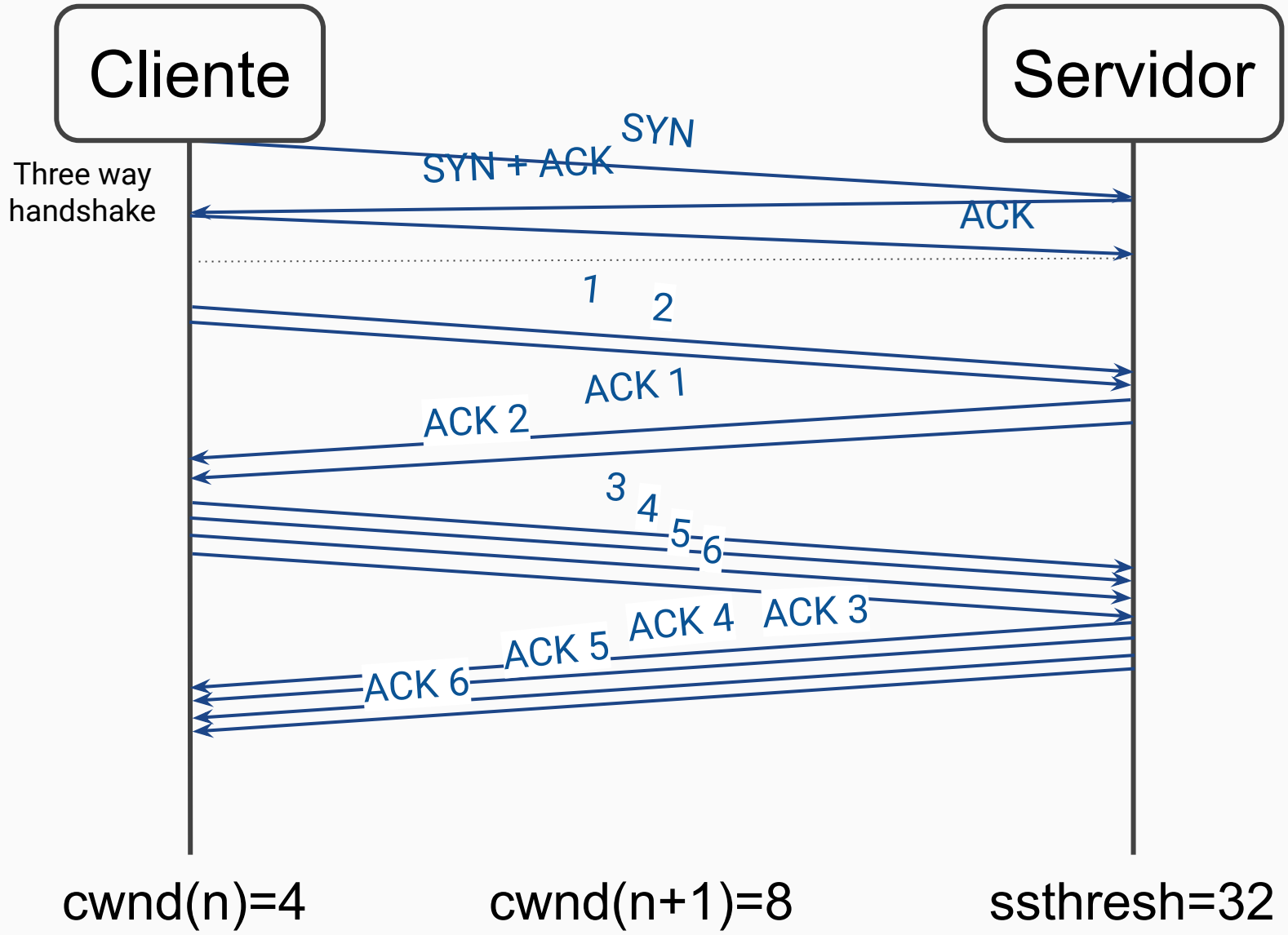

Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
cwnd(n)=2
cwnd(n+1)=4
ssthresh=32

Resolución

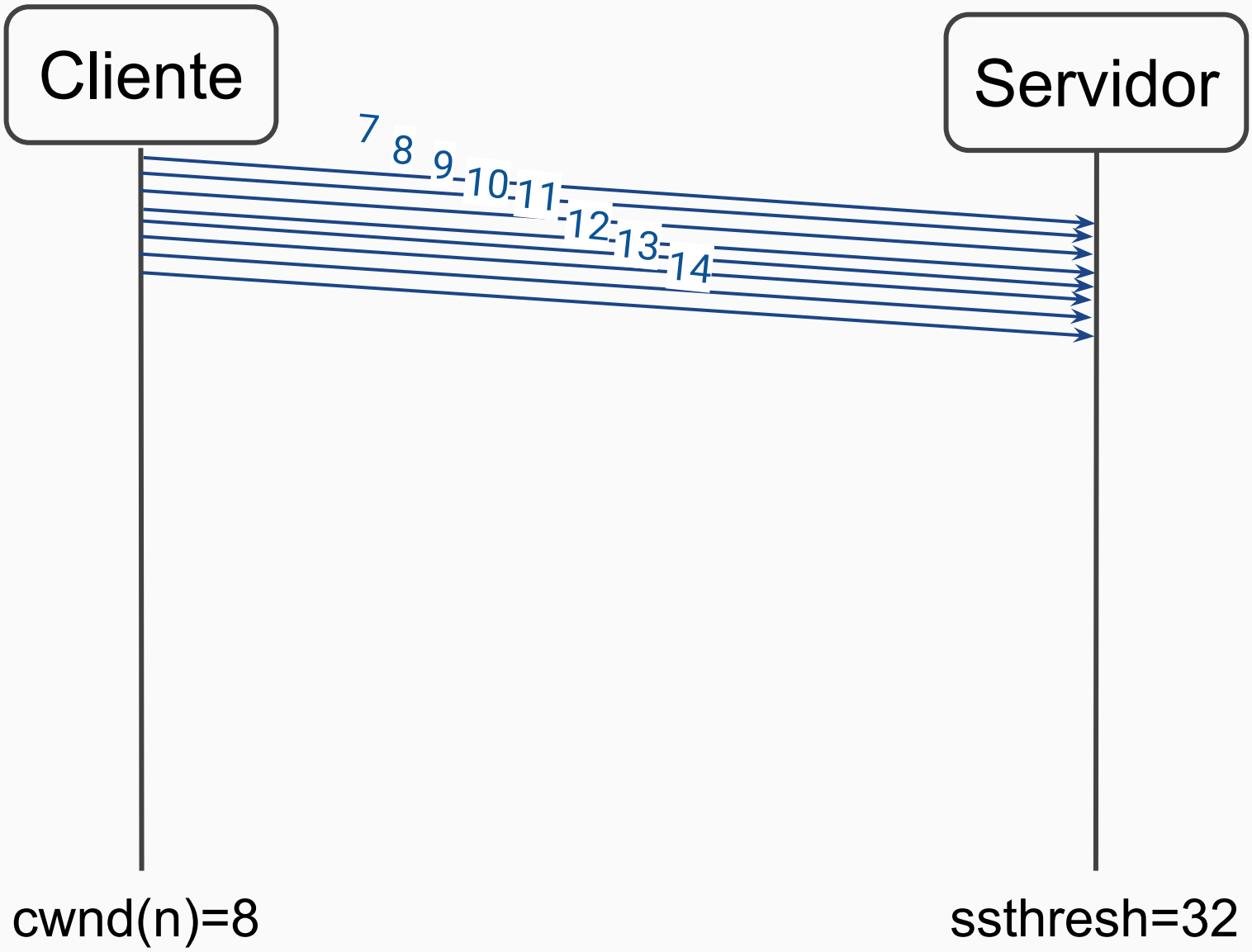


cwnd(n)=4 cwnd(n+1)=? ssthresh=32

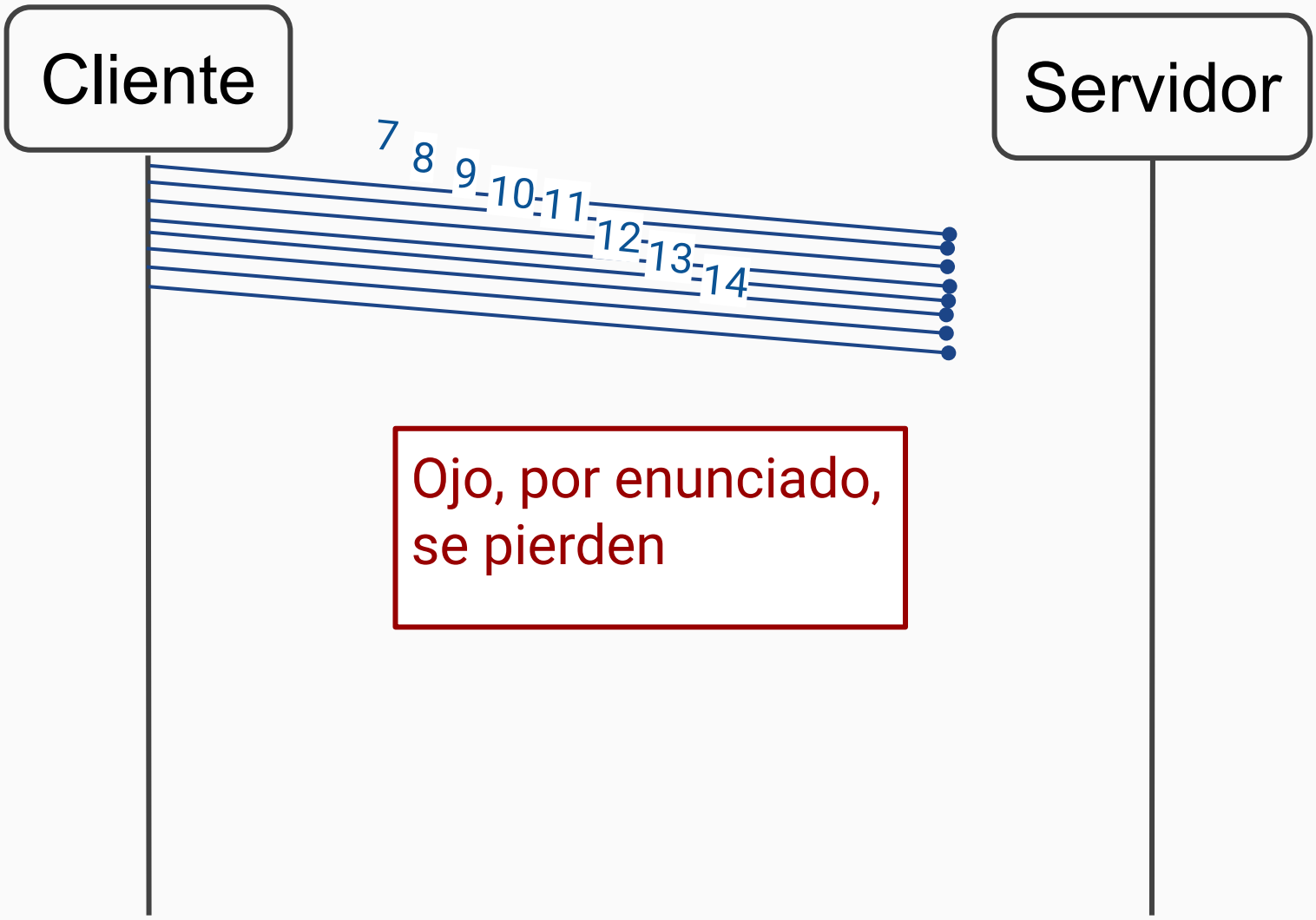
Resolución


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
3
4
5
6
ACK 3
ACK 4
ACK 5
ACK 6
cwnd(n)=4
cwnd(n+1)=8
ssthresh=32

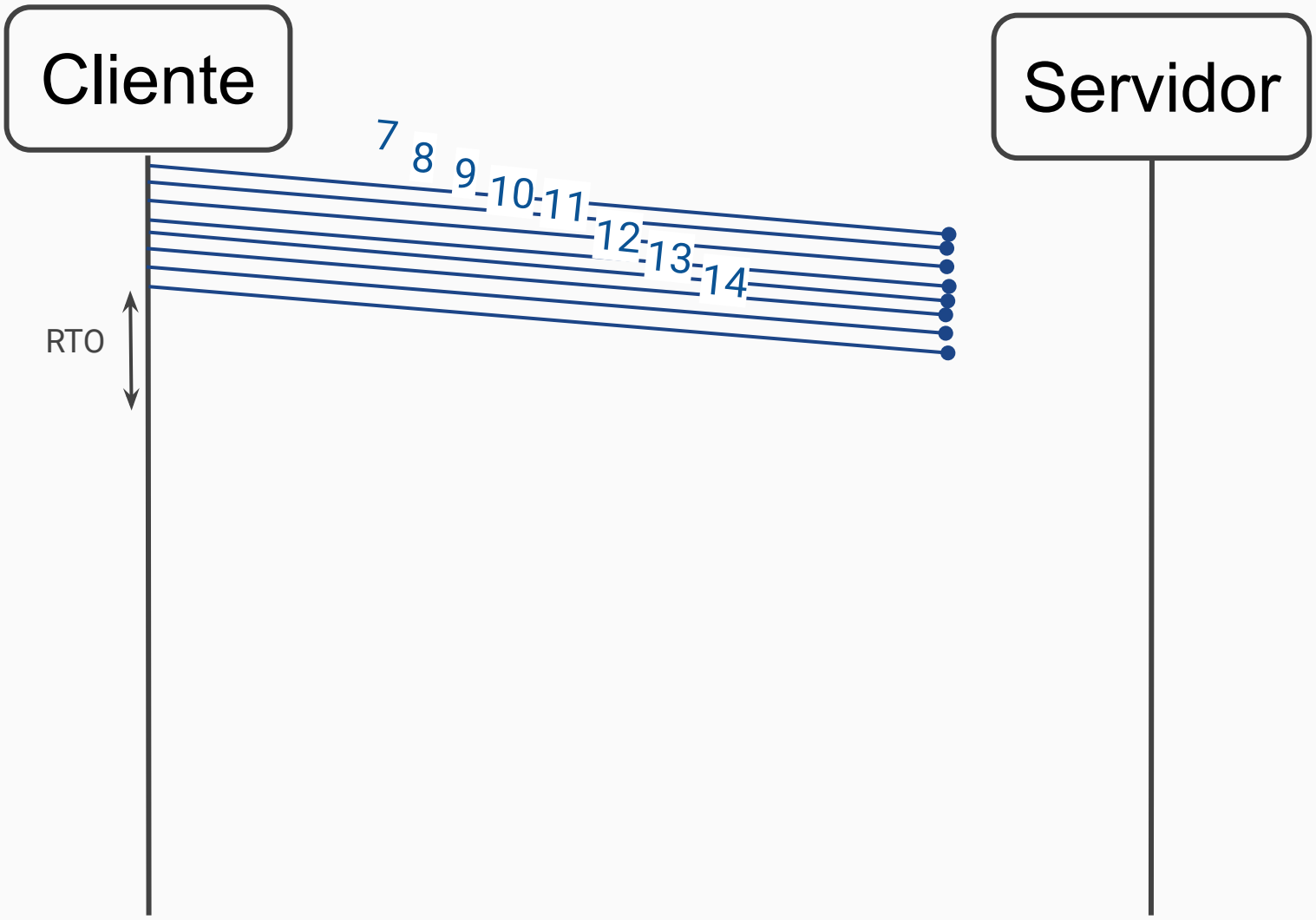
Slow start



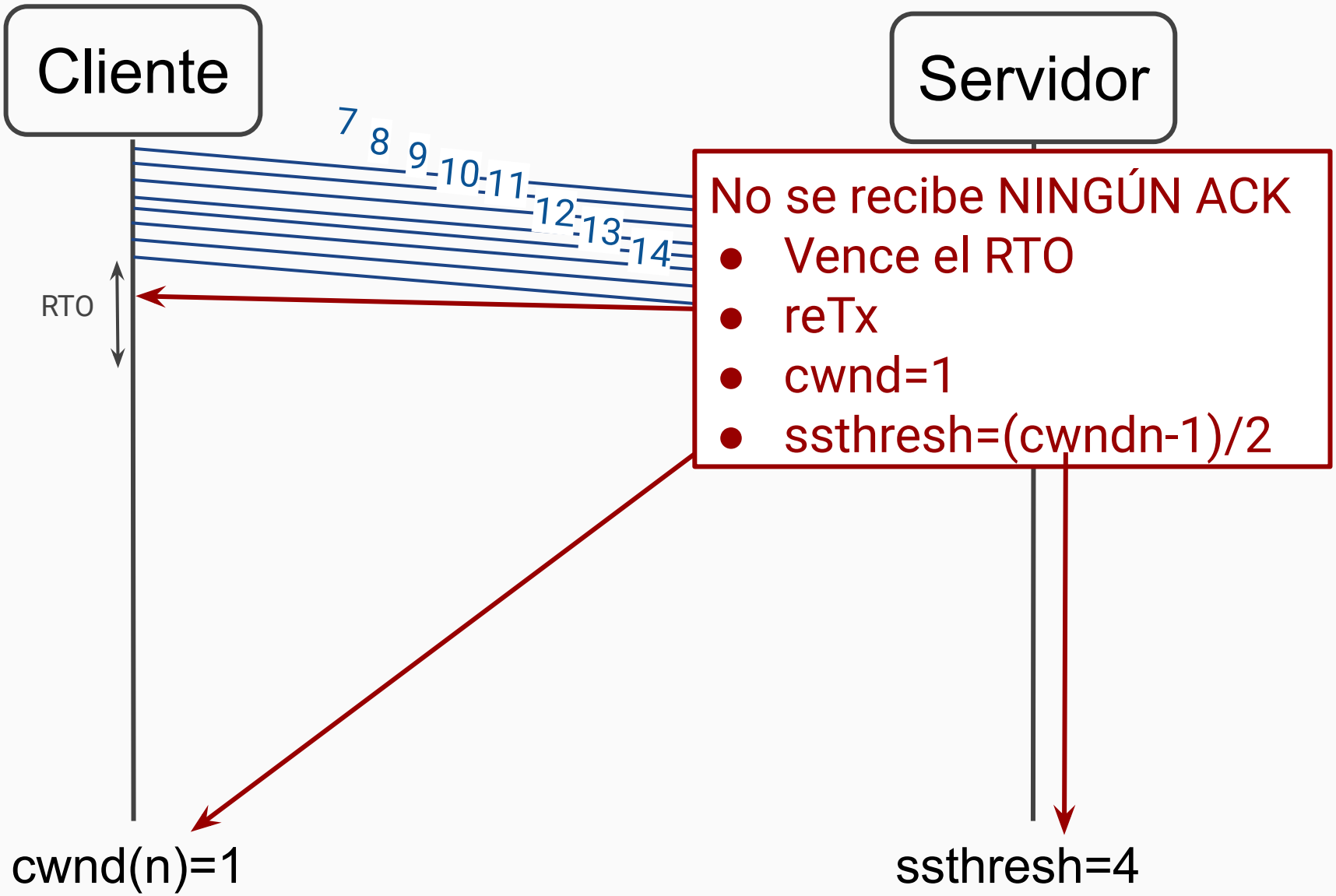
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
Ojo, por enunciado, se pierden
cwnd(n)=8
ssthresh=32

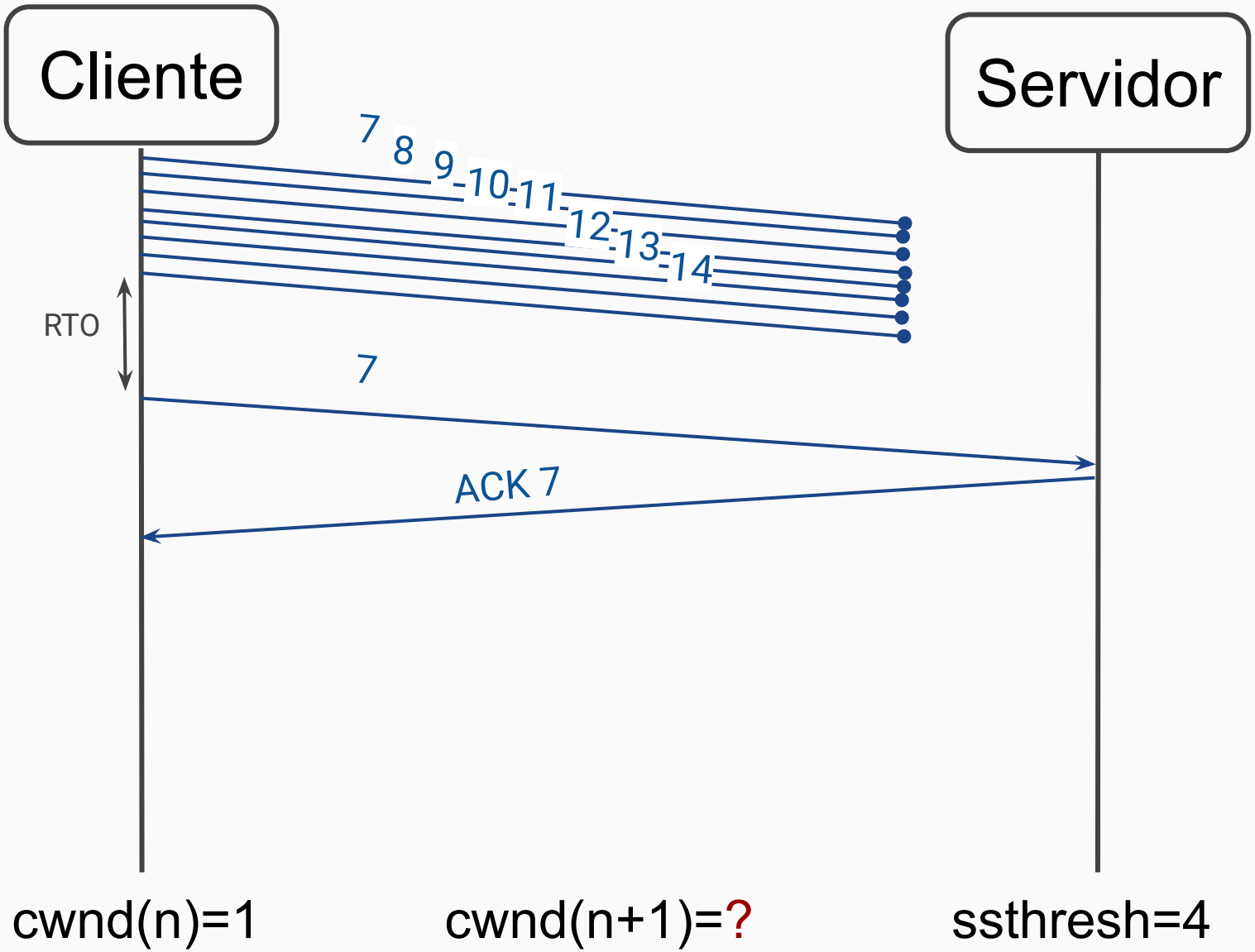
Slow start



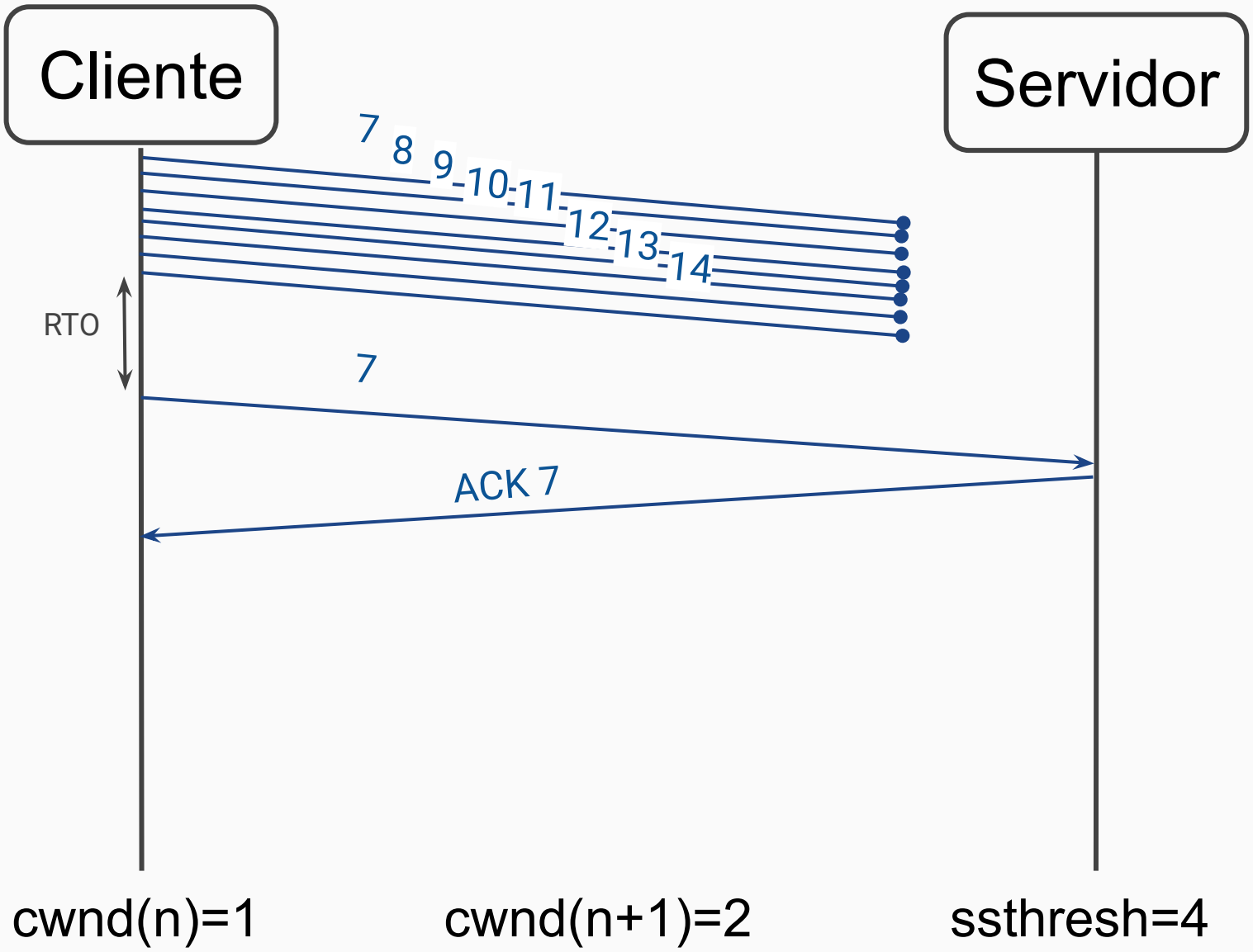
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
No se recibe NINGÚN ACK
Vence el RTO
reTx
cwnd=1
ssthresh=(cwndn-1)/2
cwnd(n)=1
ssthresh=4

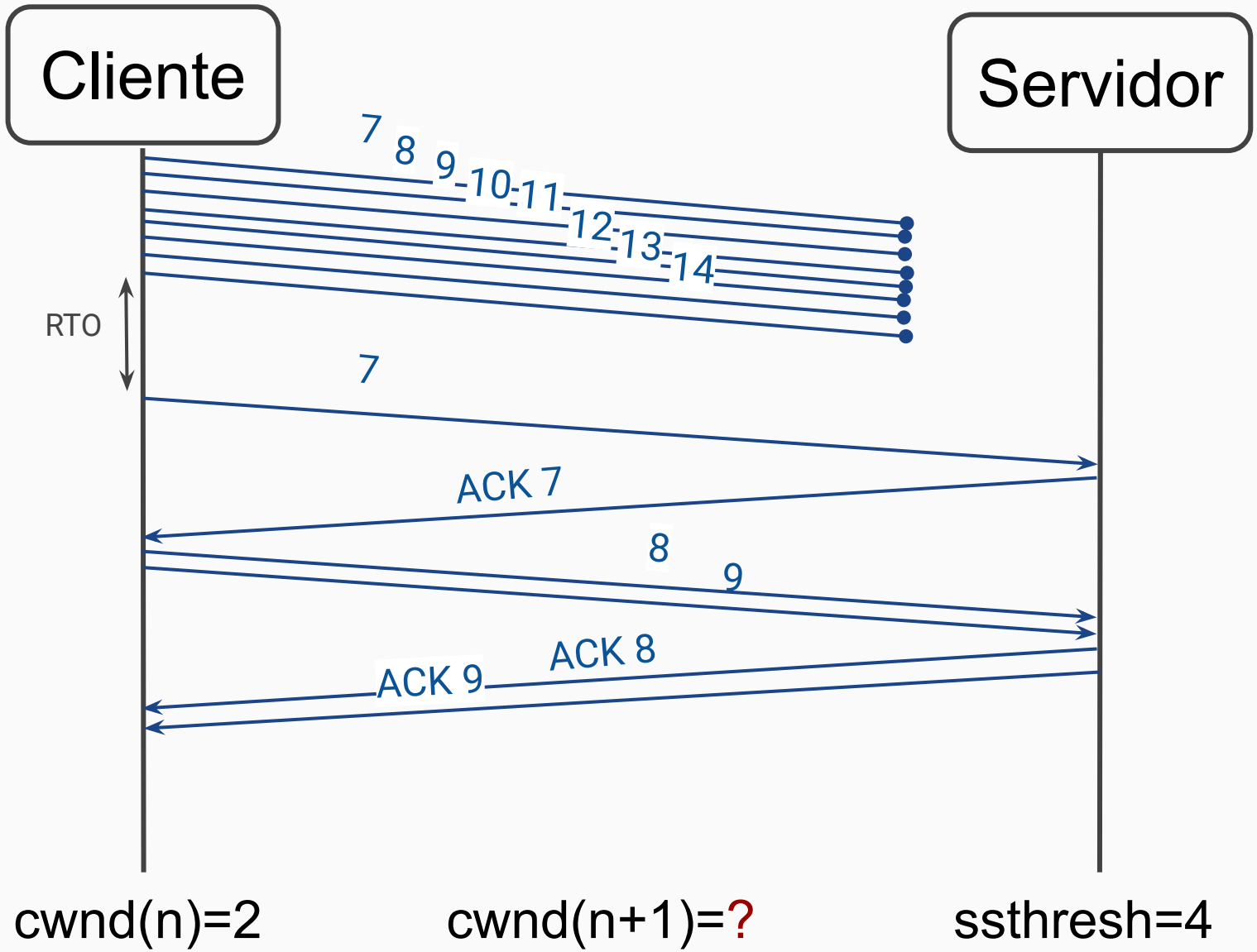
RTO Loss


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
7
ACK 7
cwnd(n)=1
cwnd(n+1)=?
ssthresh=4

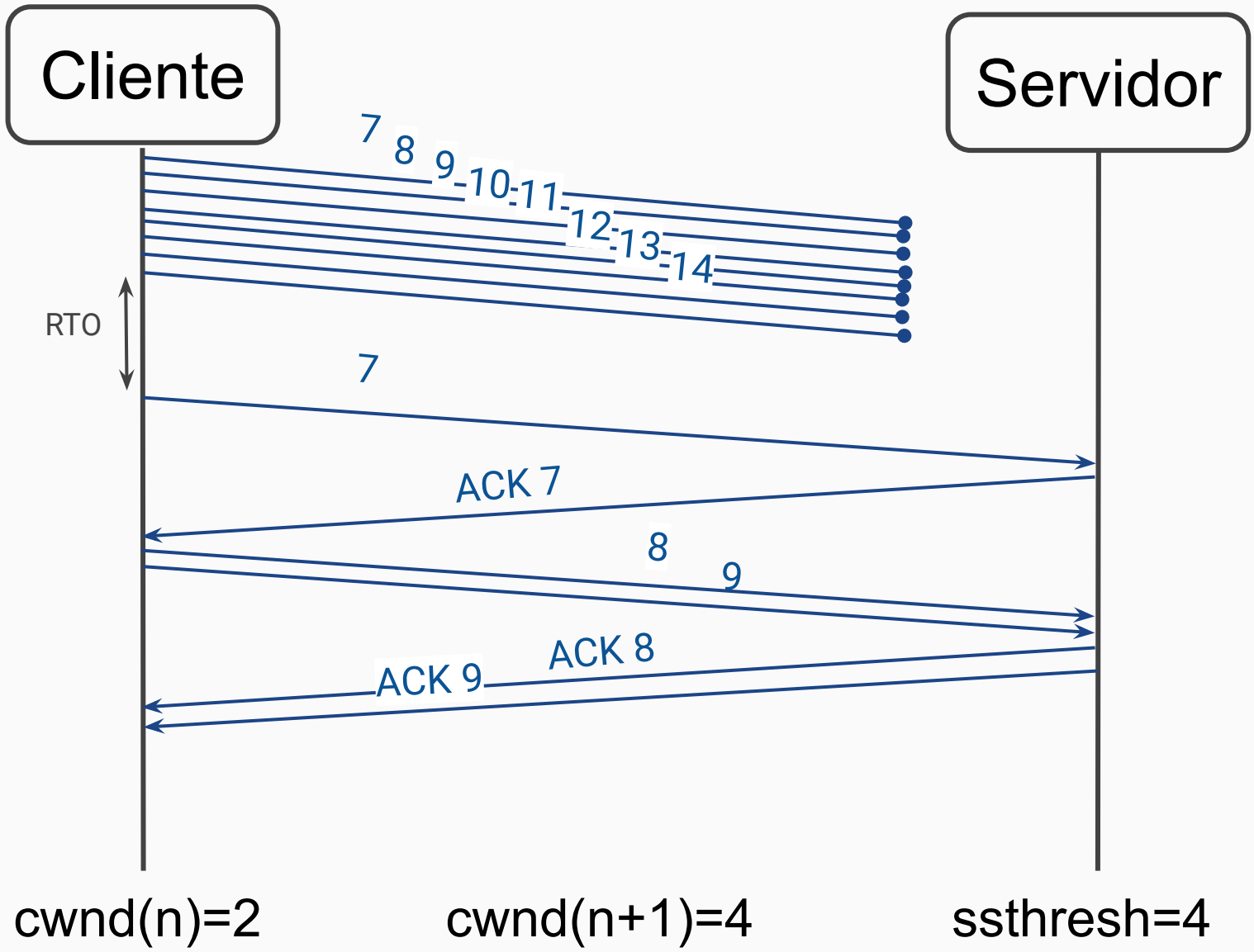
RTO Loss


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
7
ACK 7
cwnd(n)=1
cwnd(n+1)=2
ssthresh=4

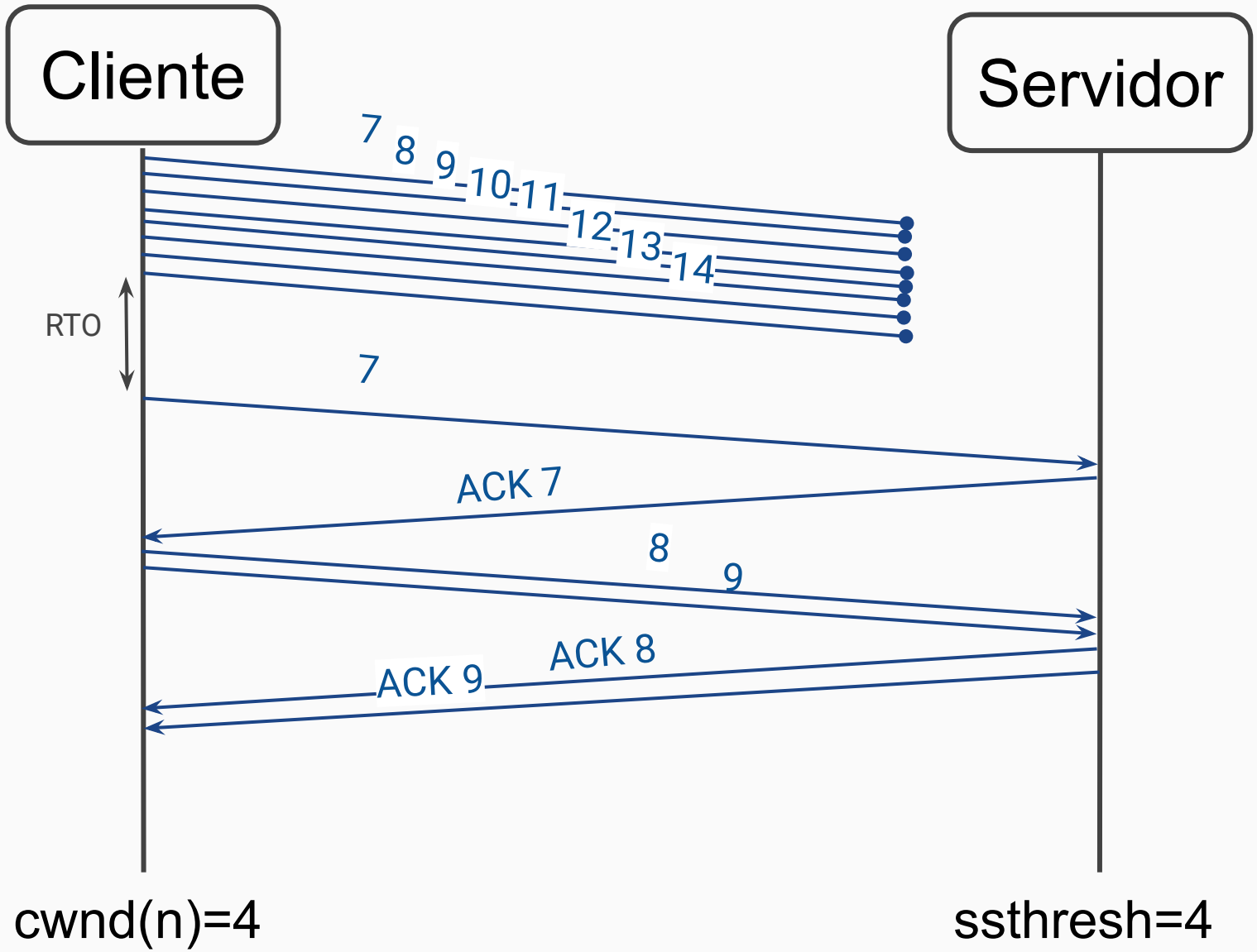
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
7
ACK 7
8
9
ACK 8
ACK 9
cwnd(n)=2
cwnd(n+1)=?
ssthresh=4

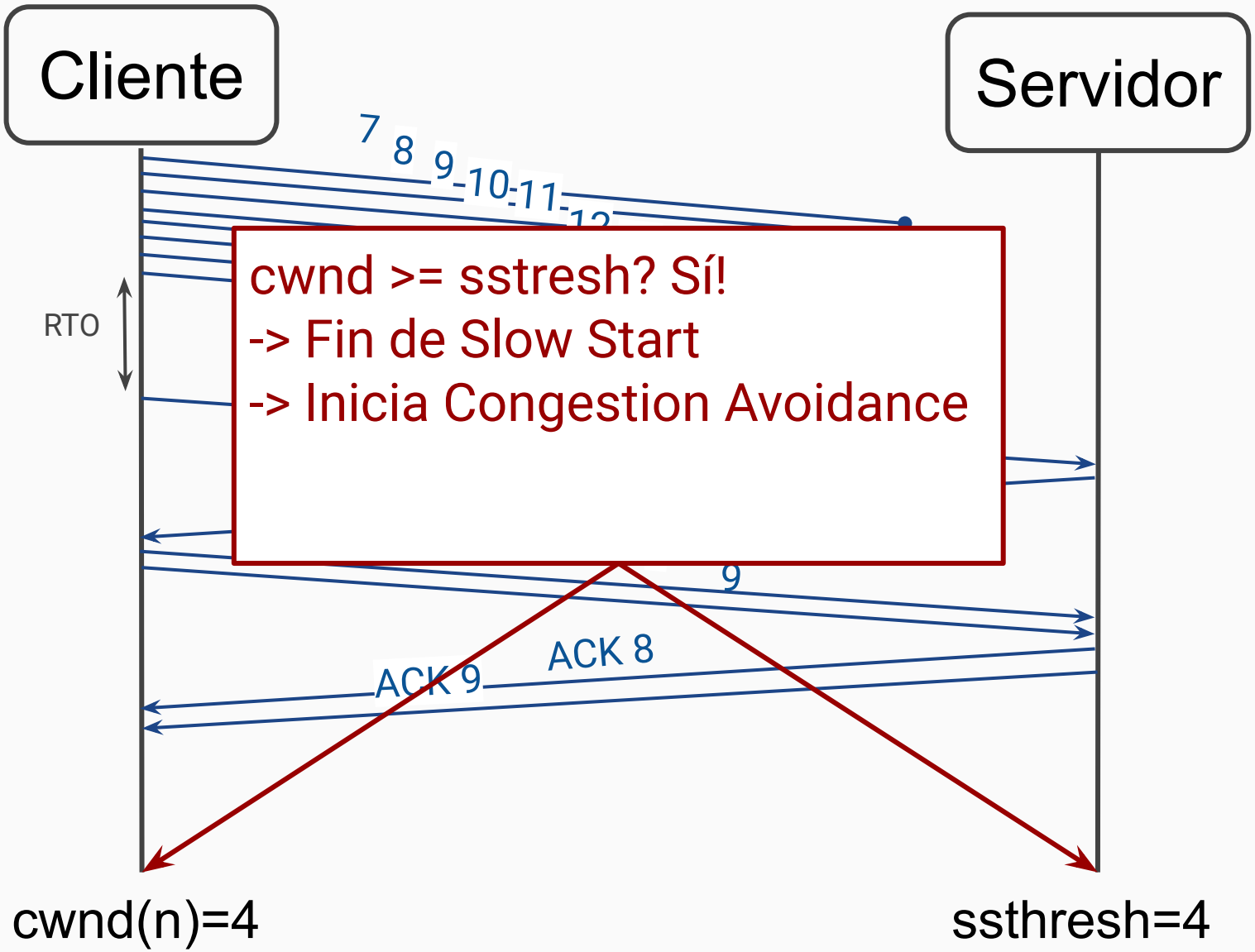
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
7
ACK 7
8
9
ACK 8
ACK 9
cwnd(n)=2
cwnd(n+1)=4
ssthresh=4

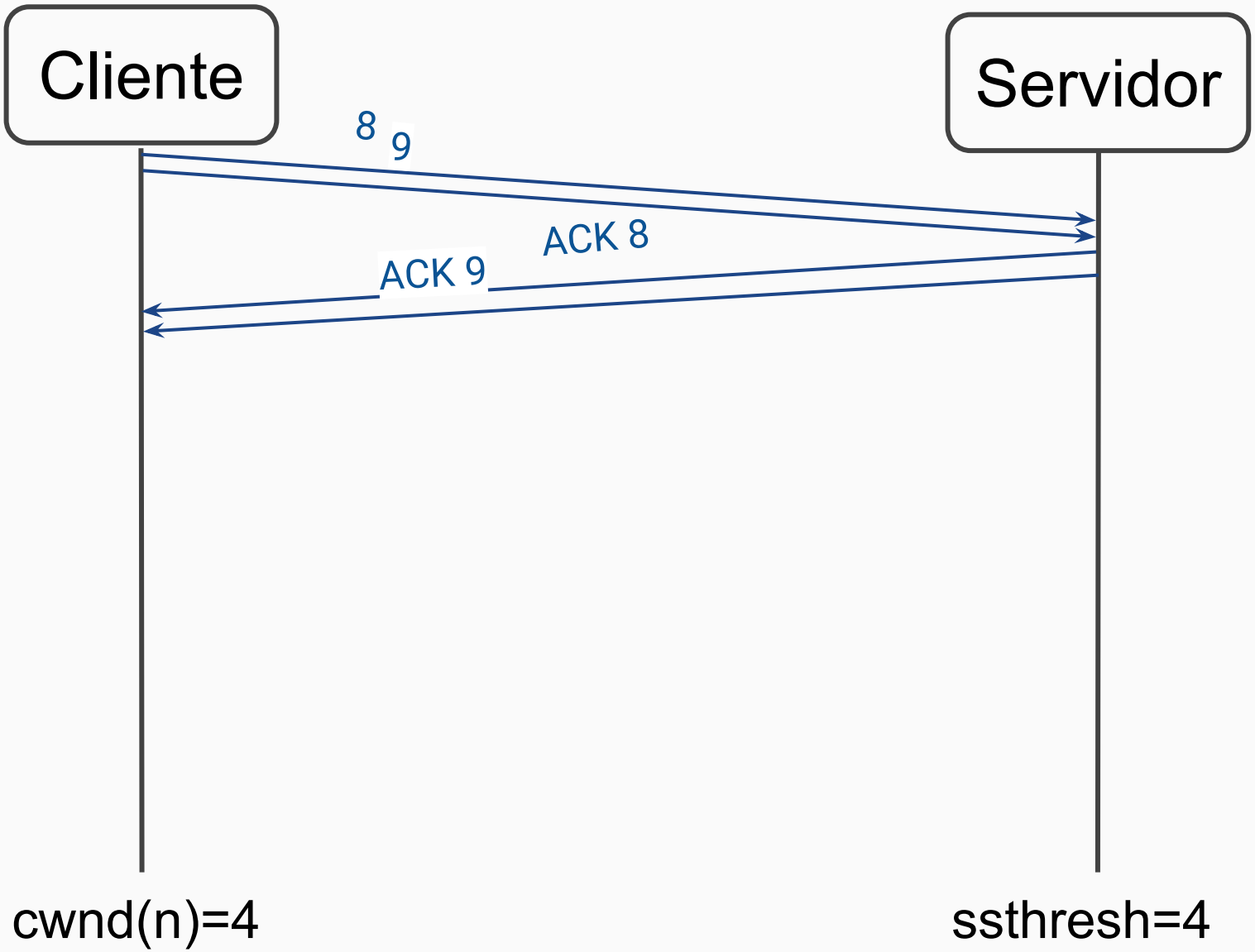
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
RTO
7
ACK 7
8
9
ACK 8
ACK 9
cwnd(n)=4
ssthresh=4

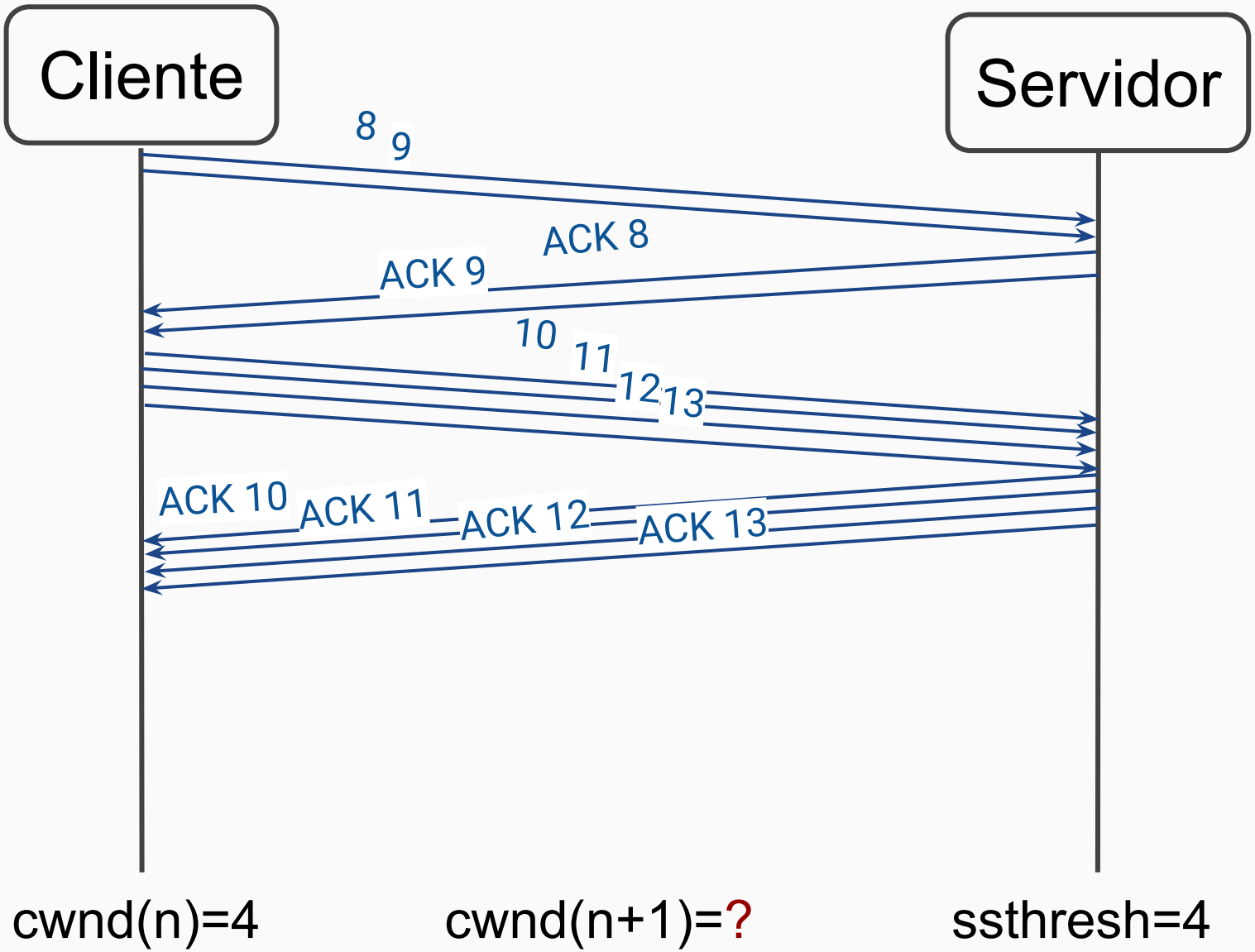
Slow start


Cliente
Servidor
7
8
9
10
11
12
RTO
cwnd >= sstresh? Sí!
-> Fin de Slow Start
-> Inicia Congestion Avoidance
9
ACK 8
ACK 9
cwnd(n)=4
ssthresh=4

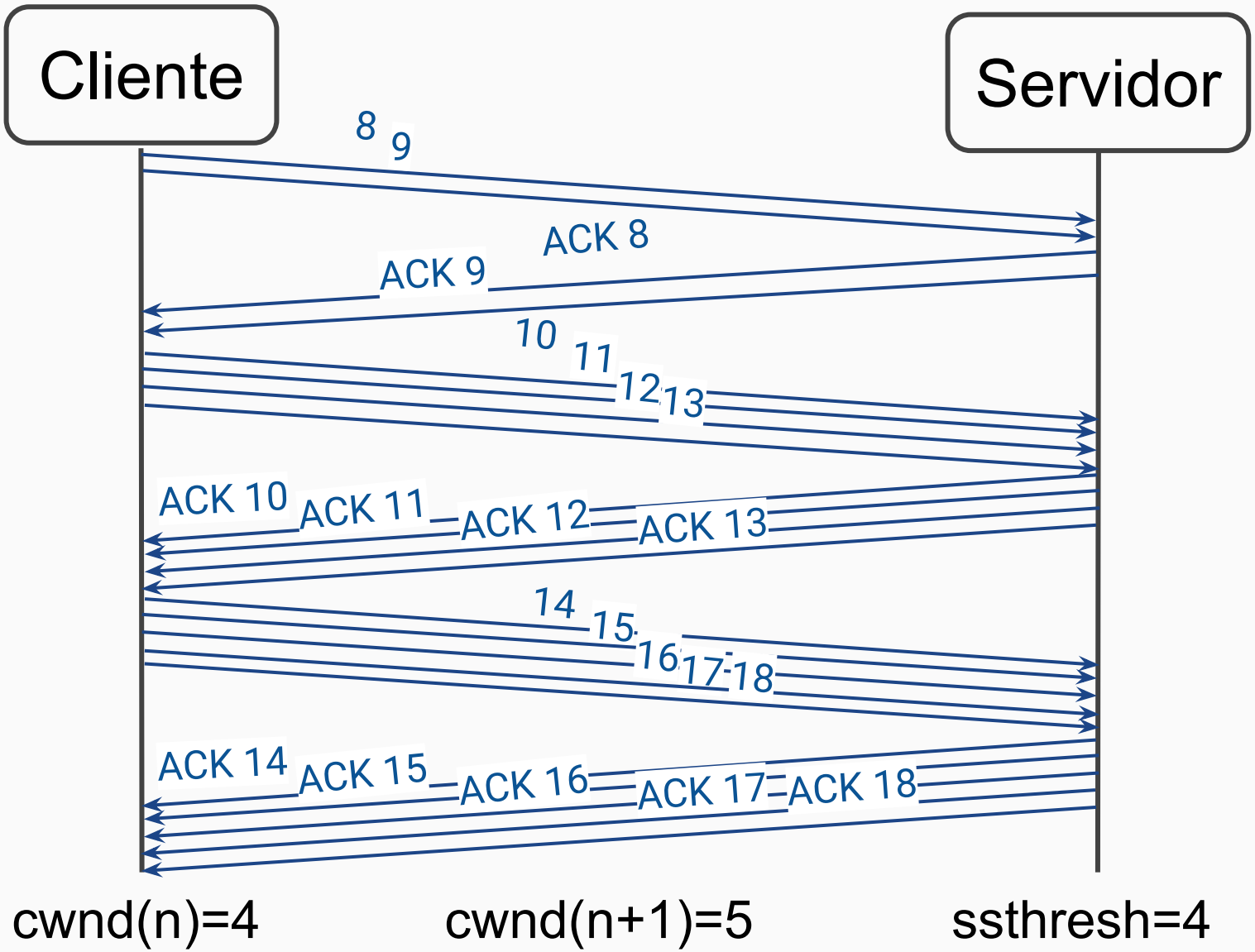
Slow start



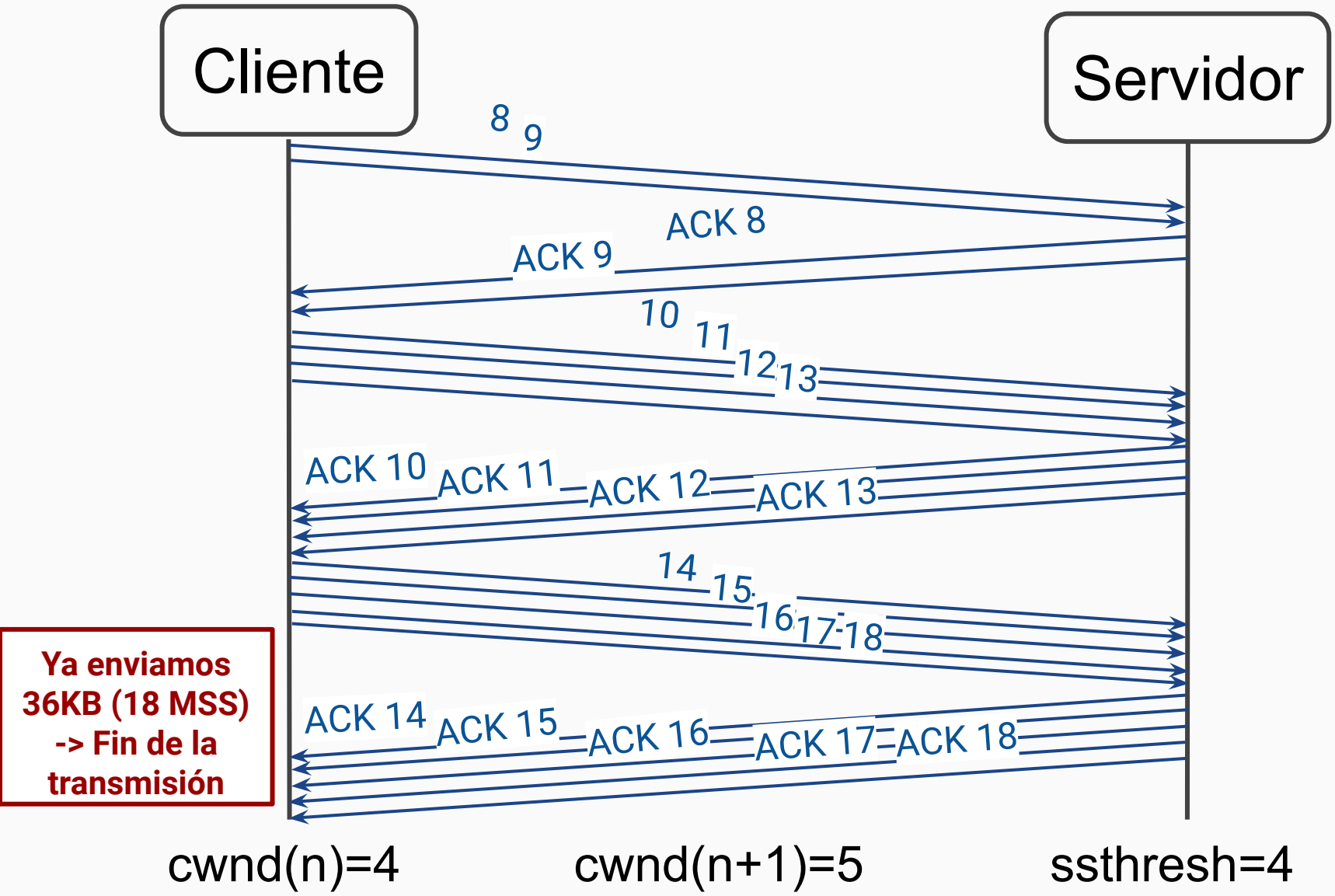
Slow start


Cliente
Servidor
8
9
ACK 8
ACK 9
10
11
12
13
ACK 10
ACK 11
ACK 12
ACK 13
cwnd(n)=4
cwnd(n+1)=?
ssthresh=4

Congestion avoidance



Congestion avoidance



Pérdida de paquetes

- **¿Qué pasaría si se pierde un paquete?**
- Supongamos que se pierde **UN SOLO** segmento de la ráfaga en cuestión.
- Y además, que estamos usando el algoritmo de control de congestión de Tahoe

TCP : control de congestión

ACK duplicados (llegan algunos paquetes)

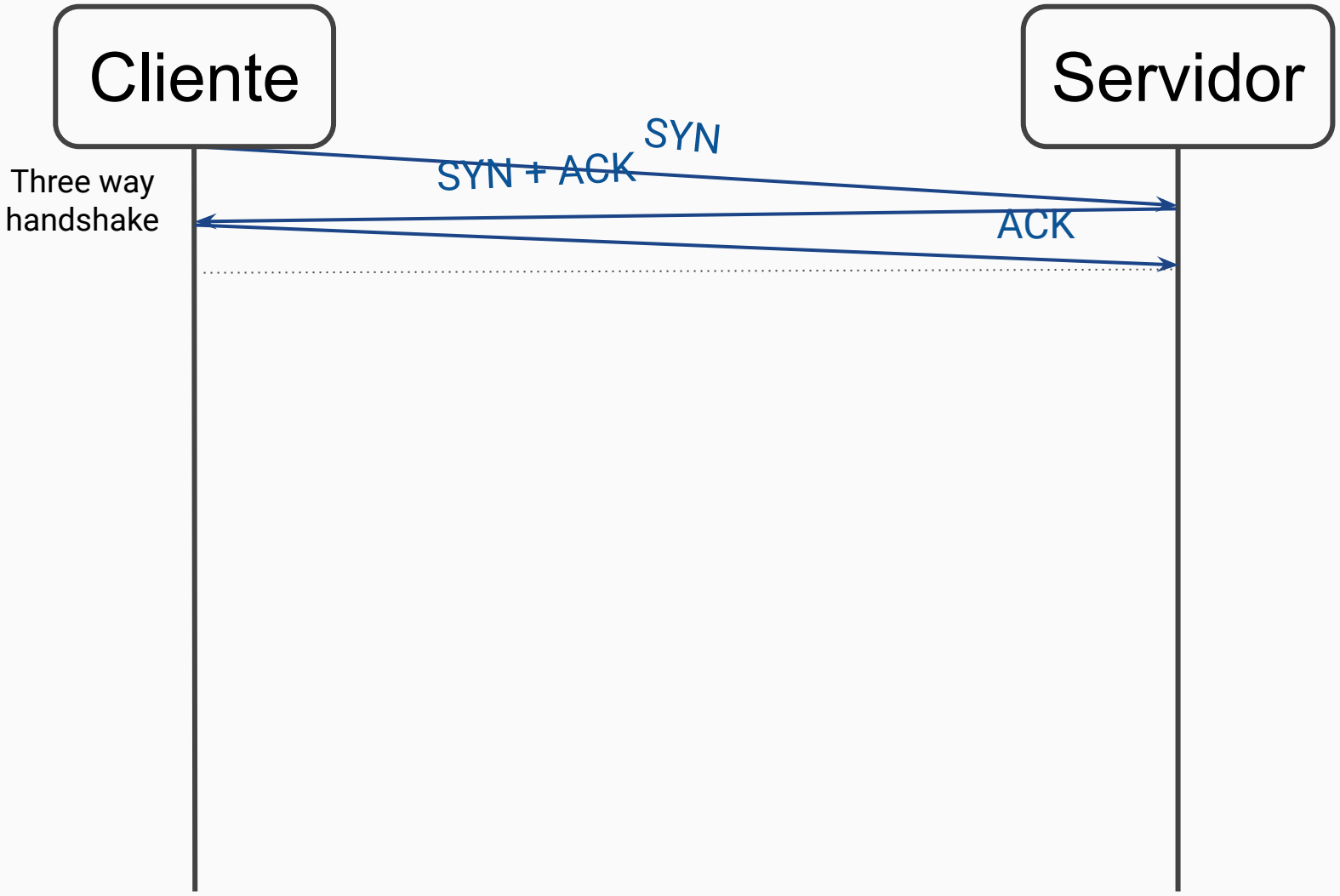
- Ocurre cuando se reciben 4 ACKs para el mismo segmento (3 dupACKs)
- Se inicia **Fast Retransmit**
- Depende del algoritmo implementado
  - **Fast Recovery**
  - **Slow Start**

TCP : control de congestión

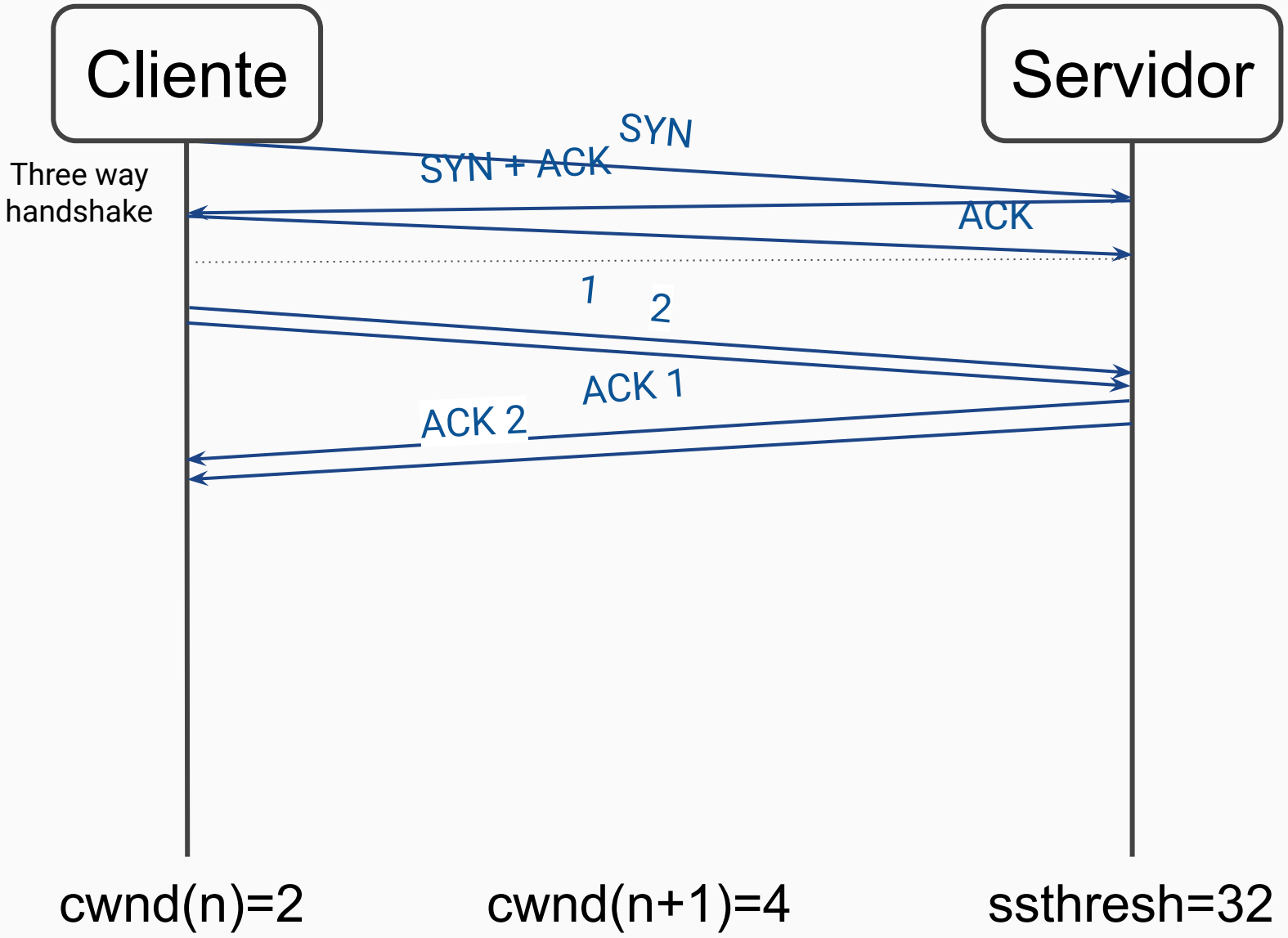
Algoritmo TCP Tahoe

- Le da al caso de los ACKs duplicados el mismo tratamiento que a un RTO (**Fast Retransmit** y luego **Slow Start**)
- Ver ejemplo

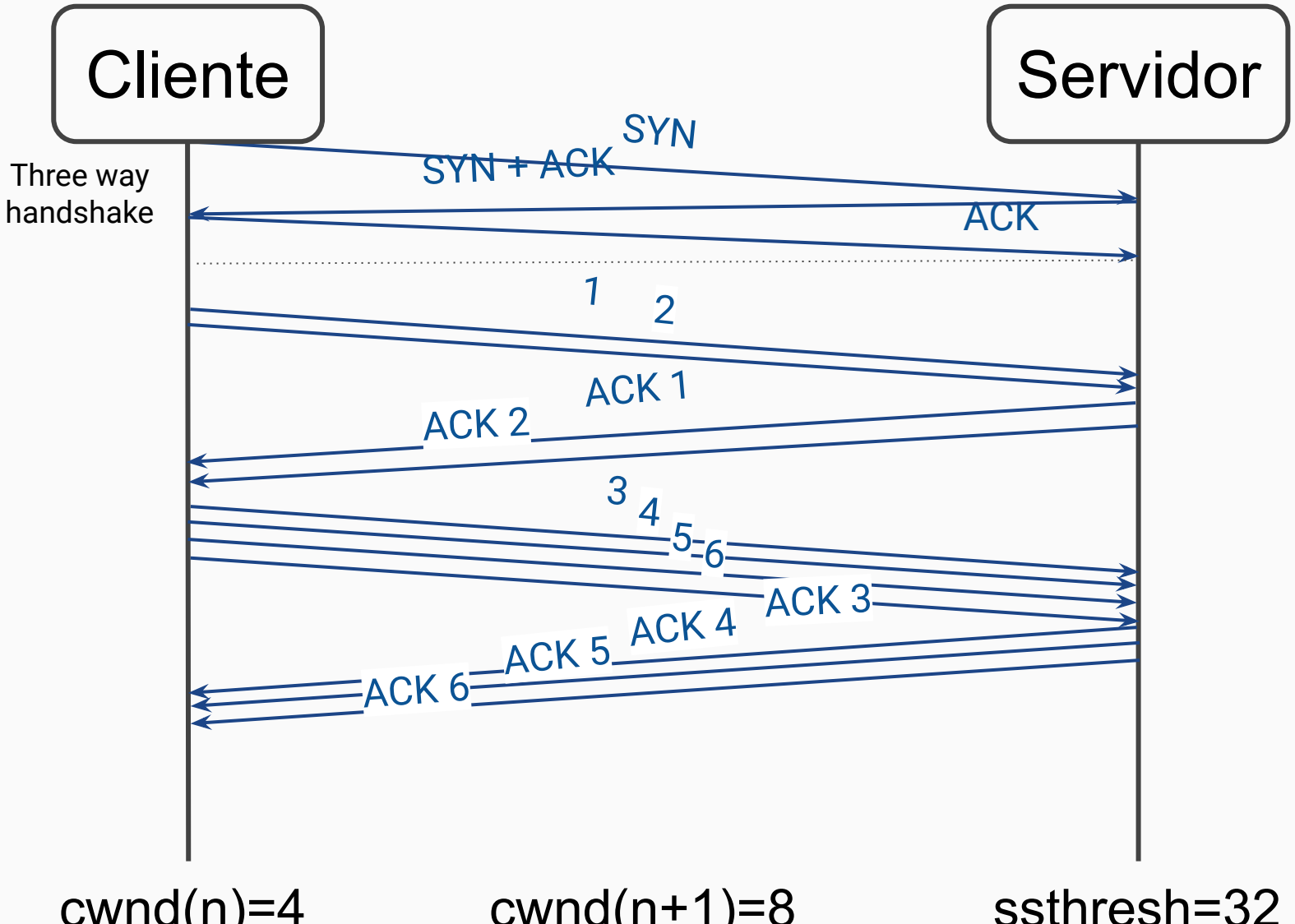
Three way handshake



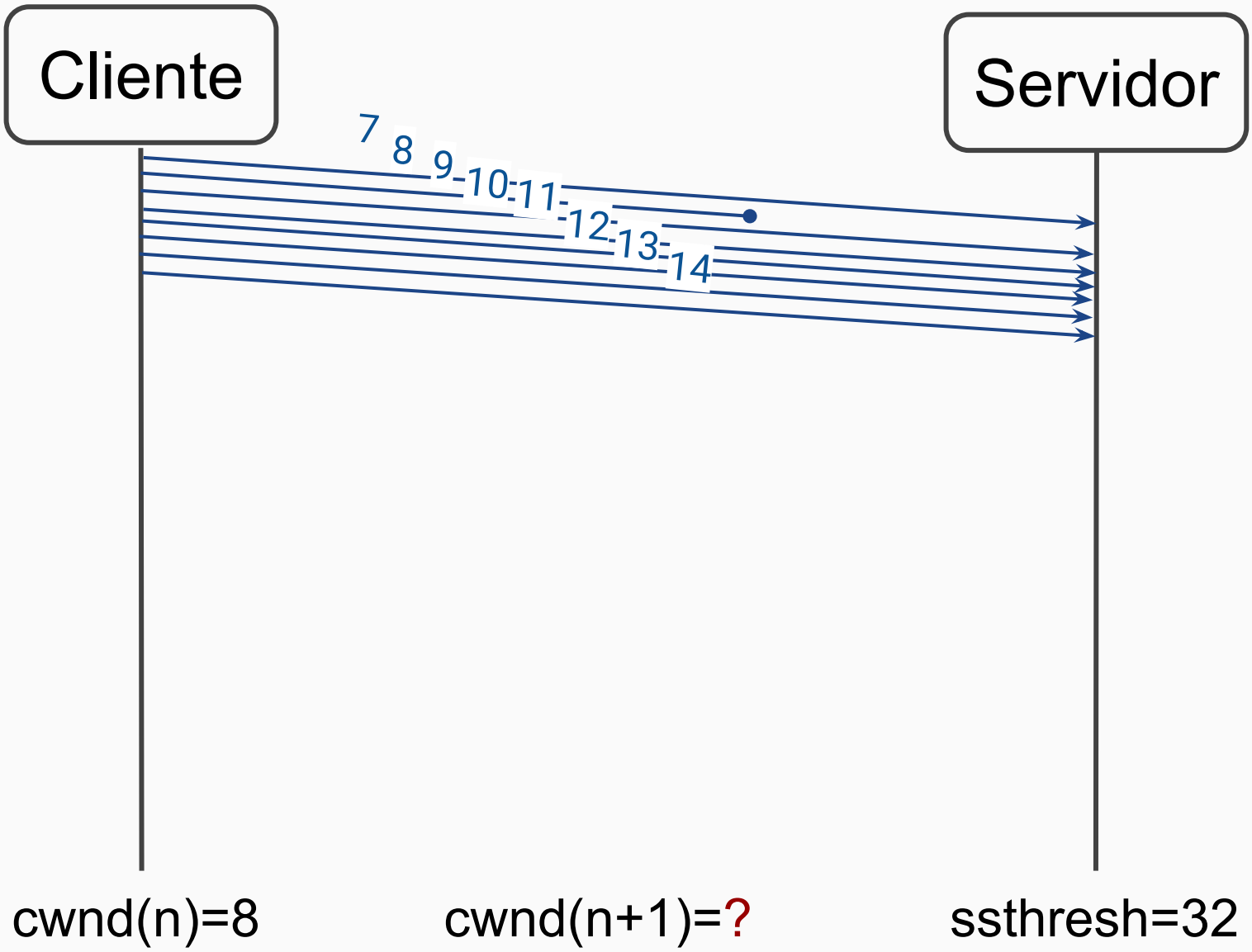
Slow start


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
cwnd(n)=2
cwnd(n+1)=4
ssthresh=32

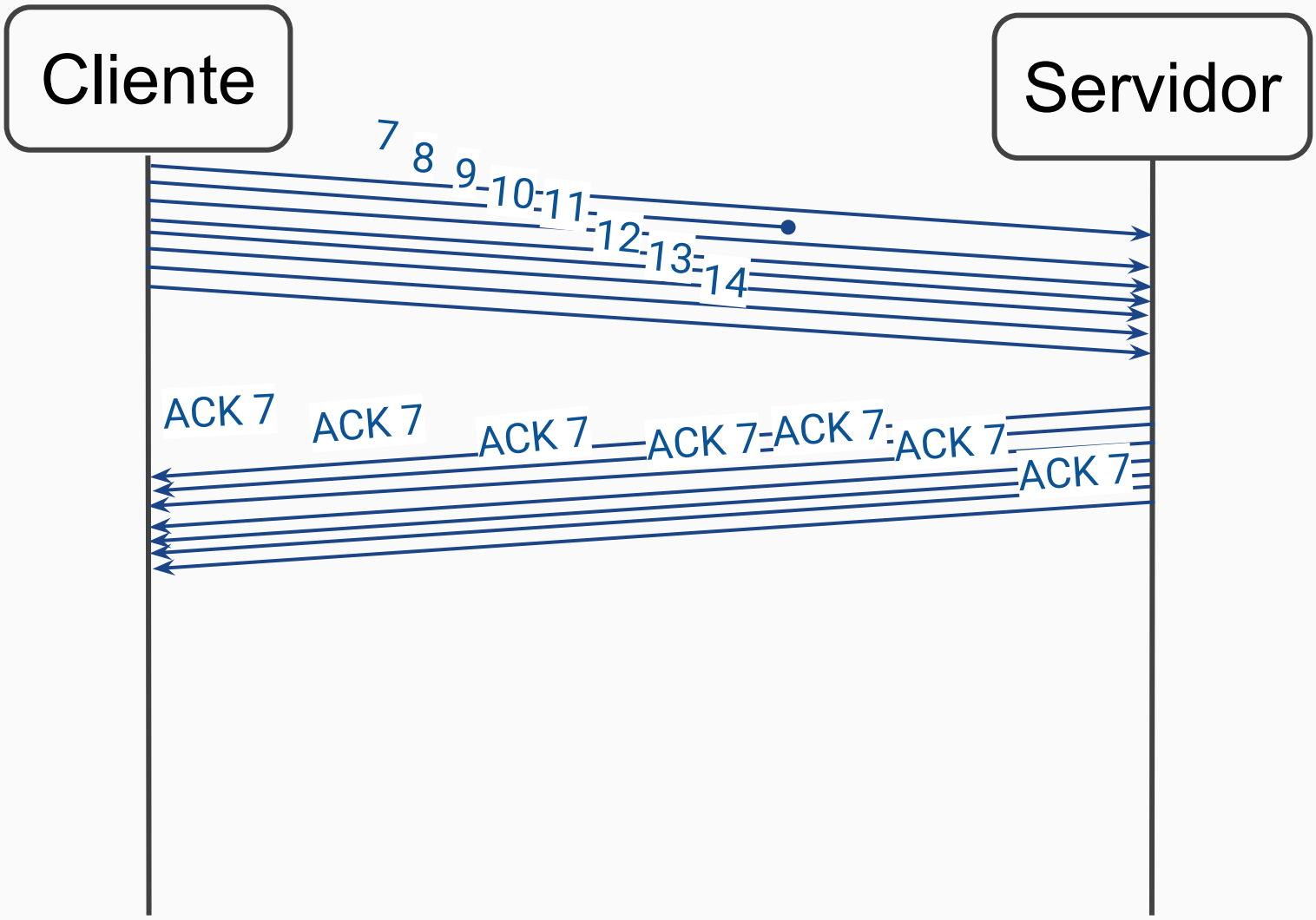
Slow start


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
3
4
5
6
ACK 3
ACK 4
ACK 5
ACK 6
cwnd(n)=4
cwnd(n+1)=8
ssthresh=32

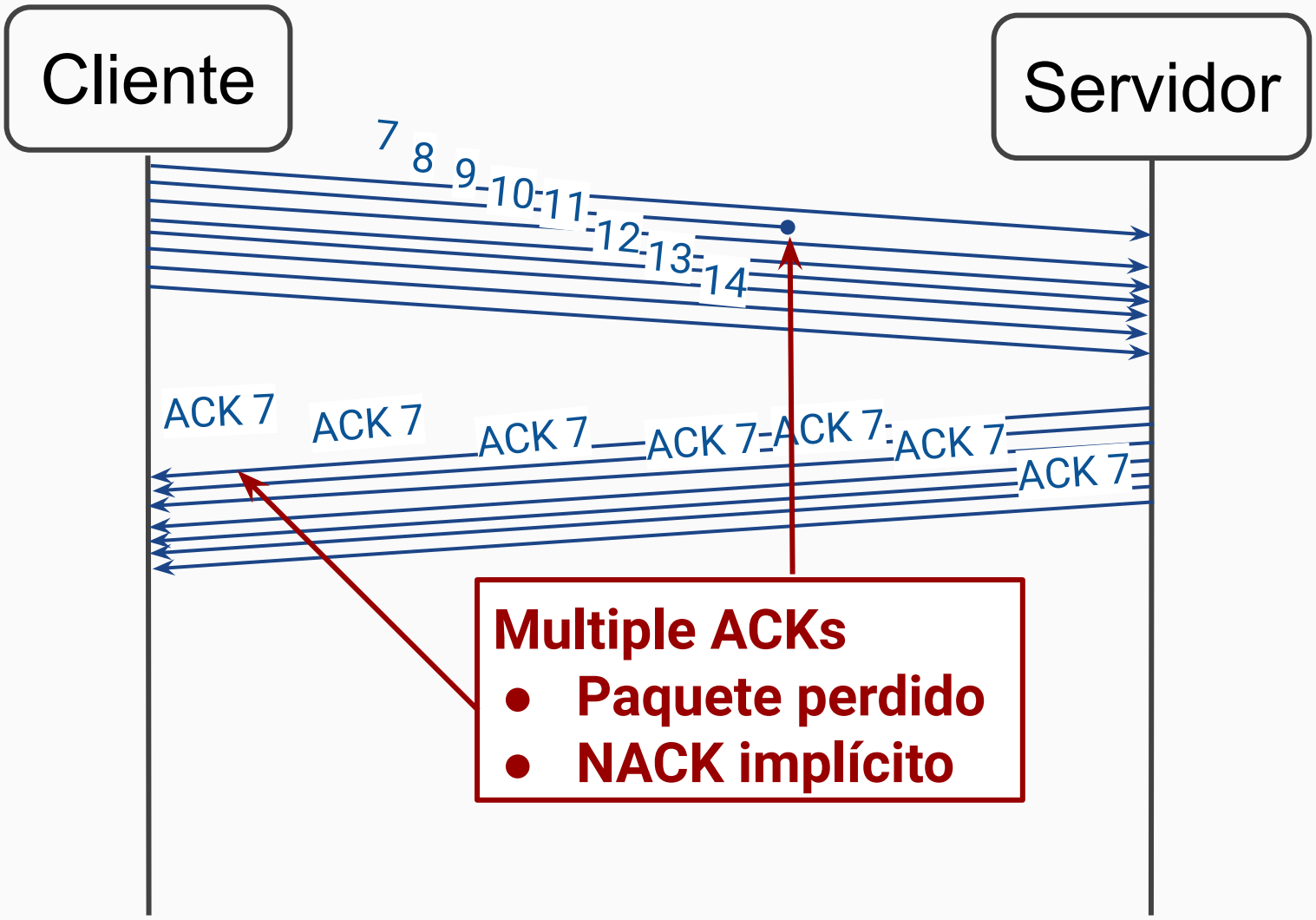
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
cwnd(n)=8
cwnd(n+1)=?
ssthresh=32

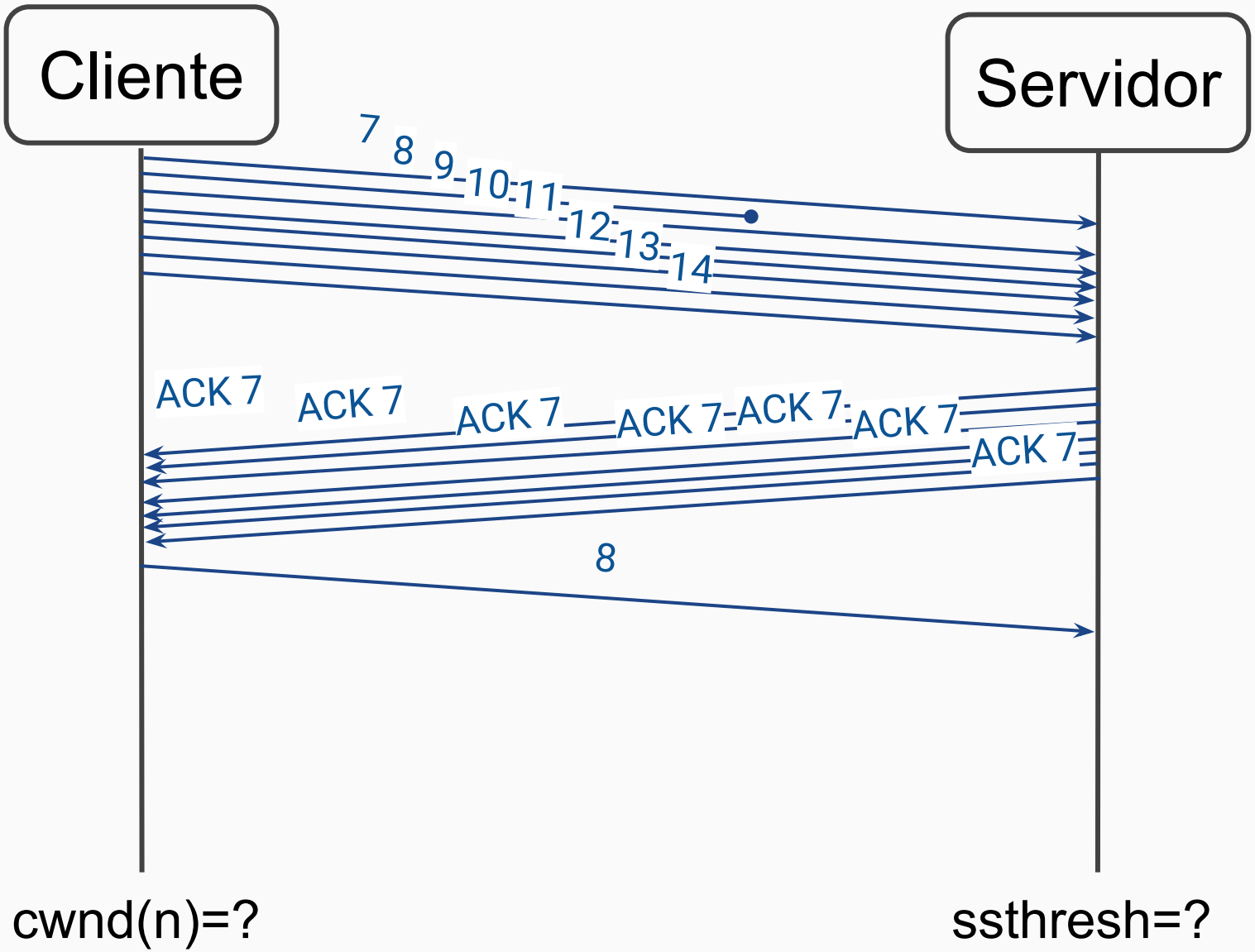
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
cwnd(n)=8
cwnd(n+1)=?
ssthresh=32

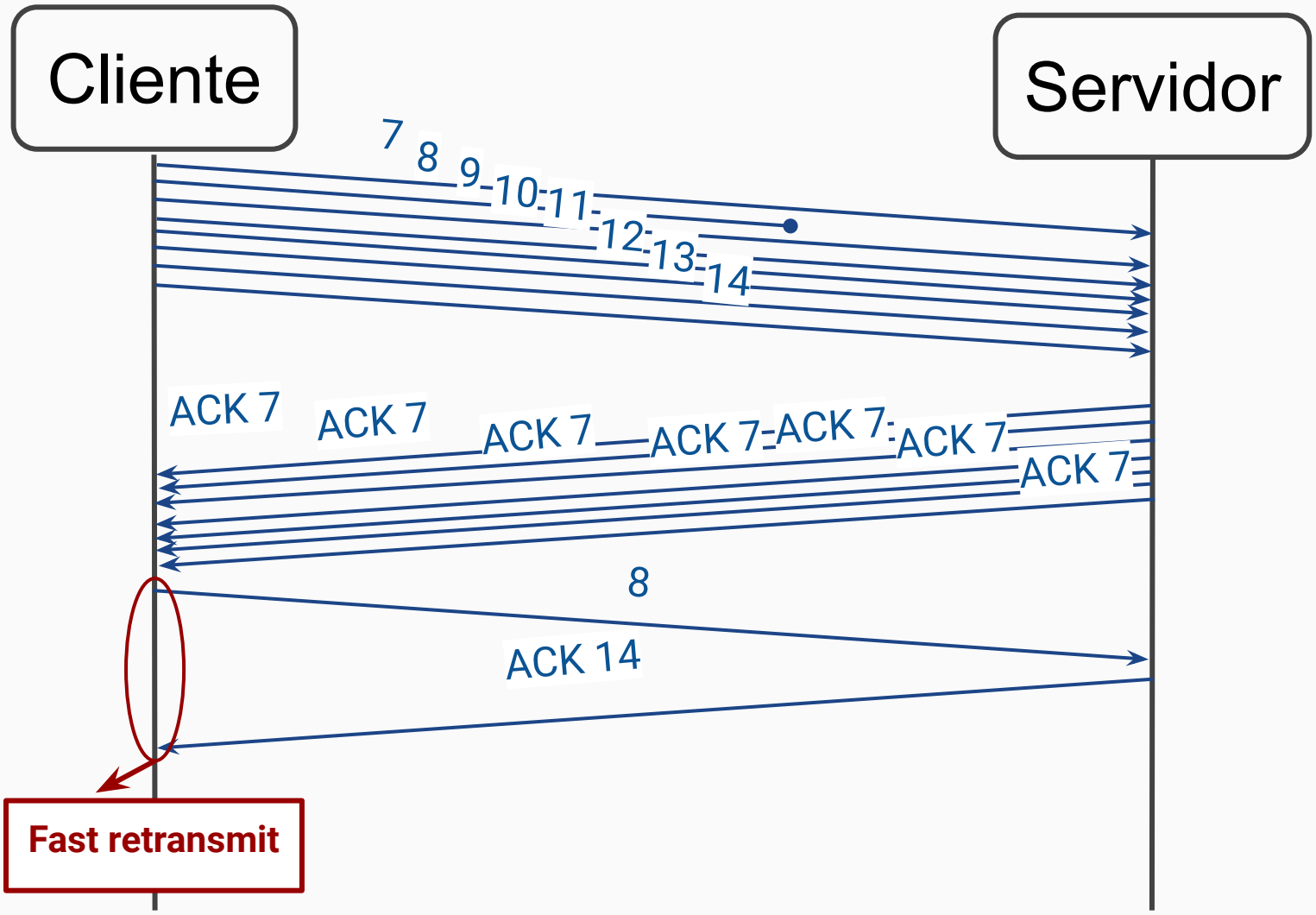
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
Multiple ACKs
Paquete perdido
NACK implícito
cwnd(n)=8
cwnd(n+1)=?
ssthresh=32

Slow start

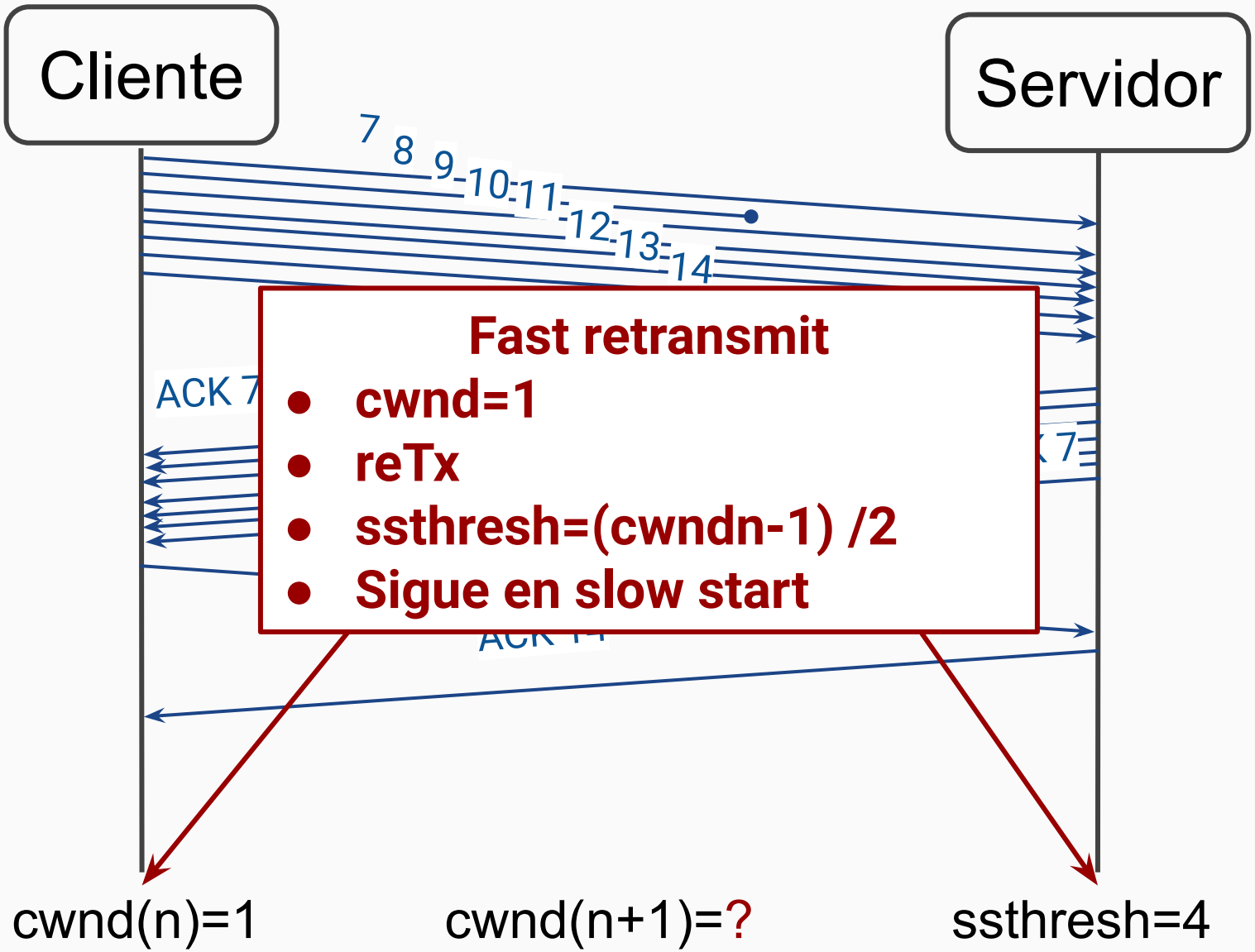

Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
cwnd(n)=?
ssthresh=?

Fast Retransmit

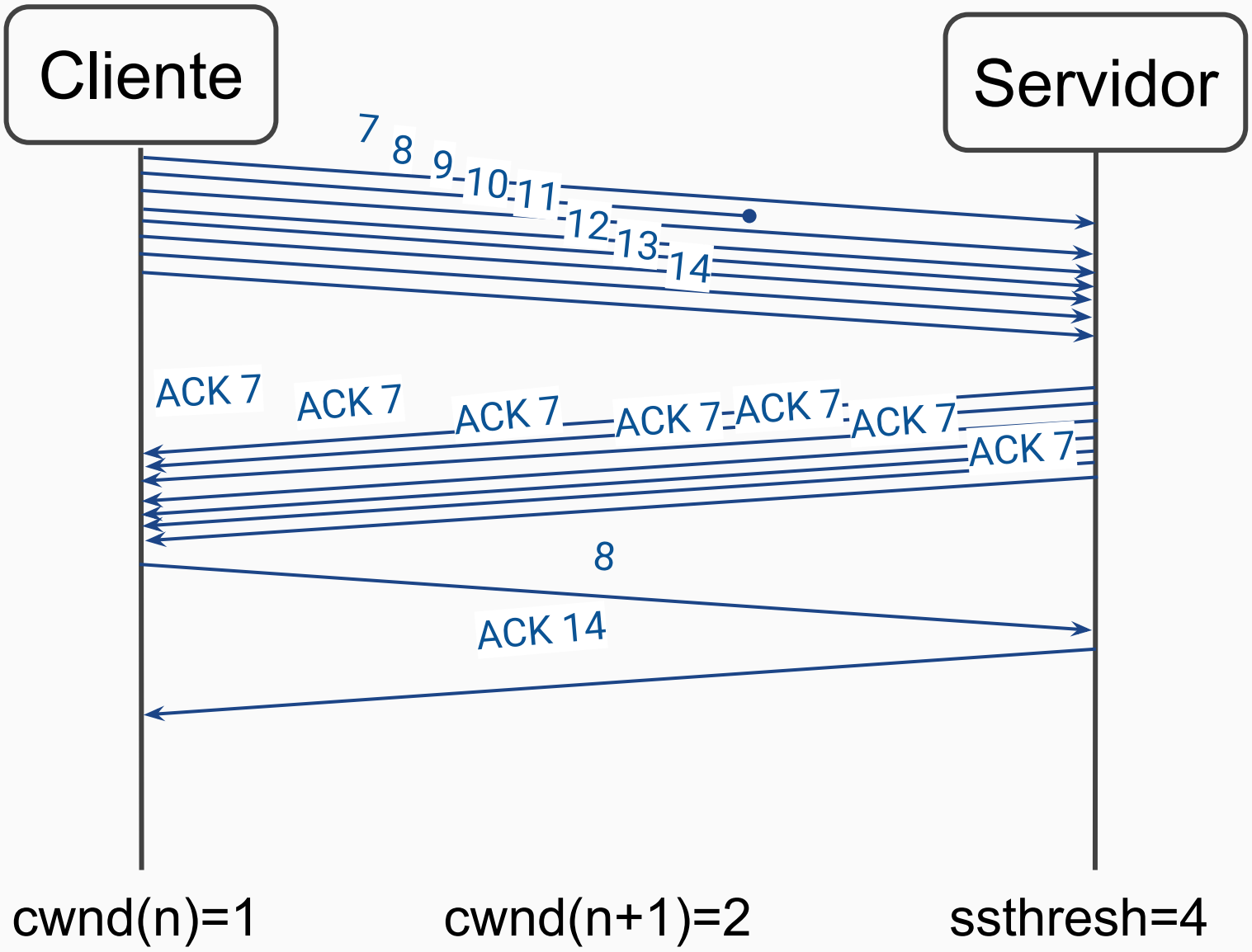


cwnd(n)=? cwnd(n+1)=? ssthresh=?

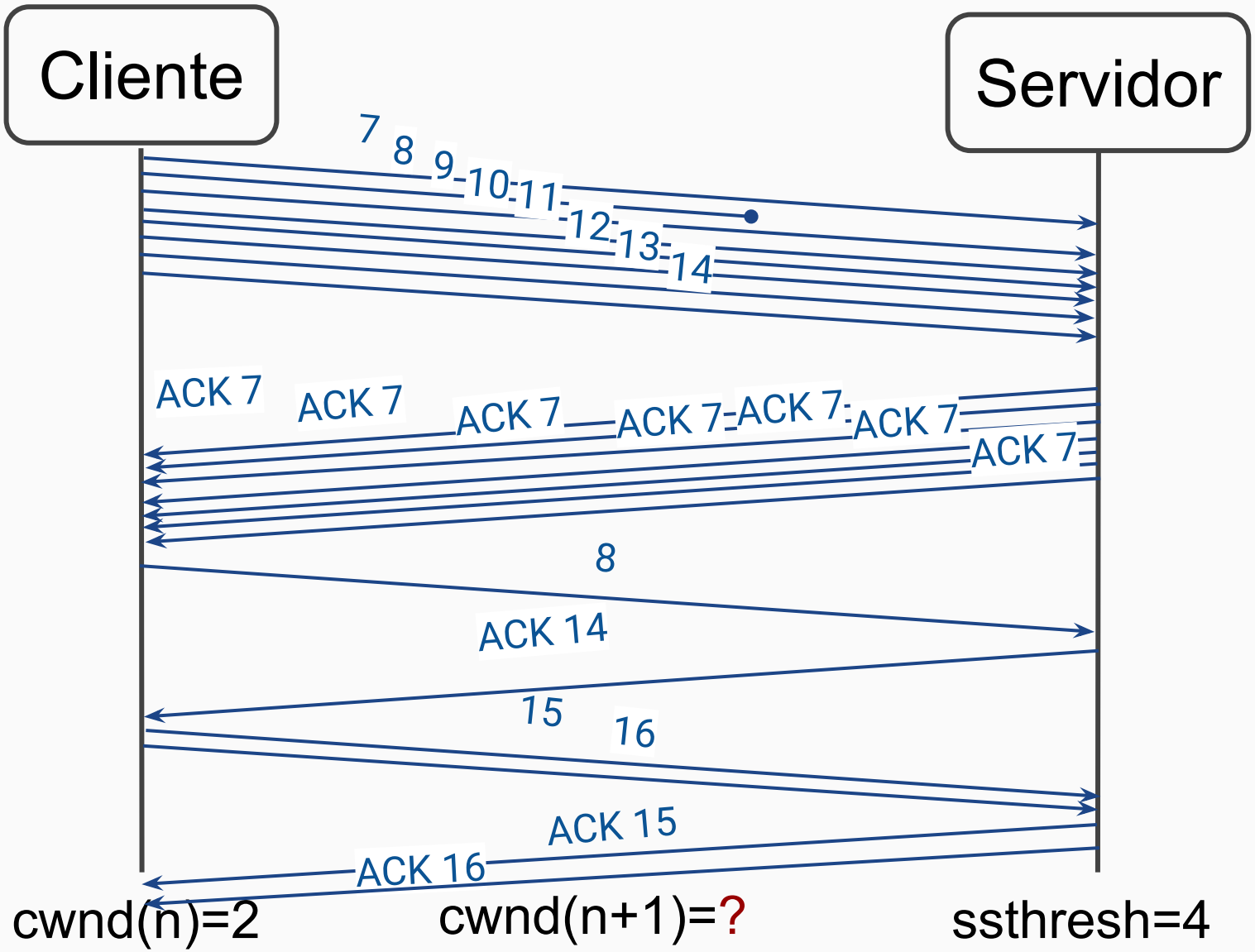
Fast Retransmit



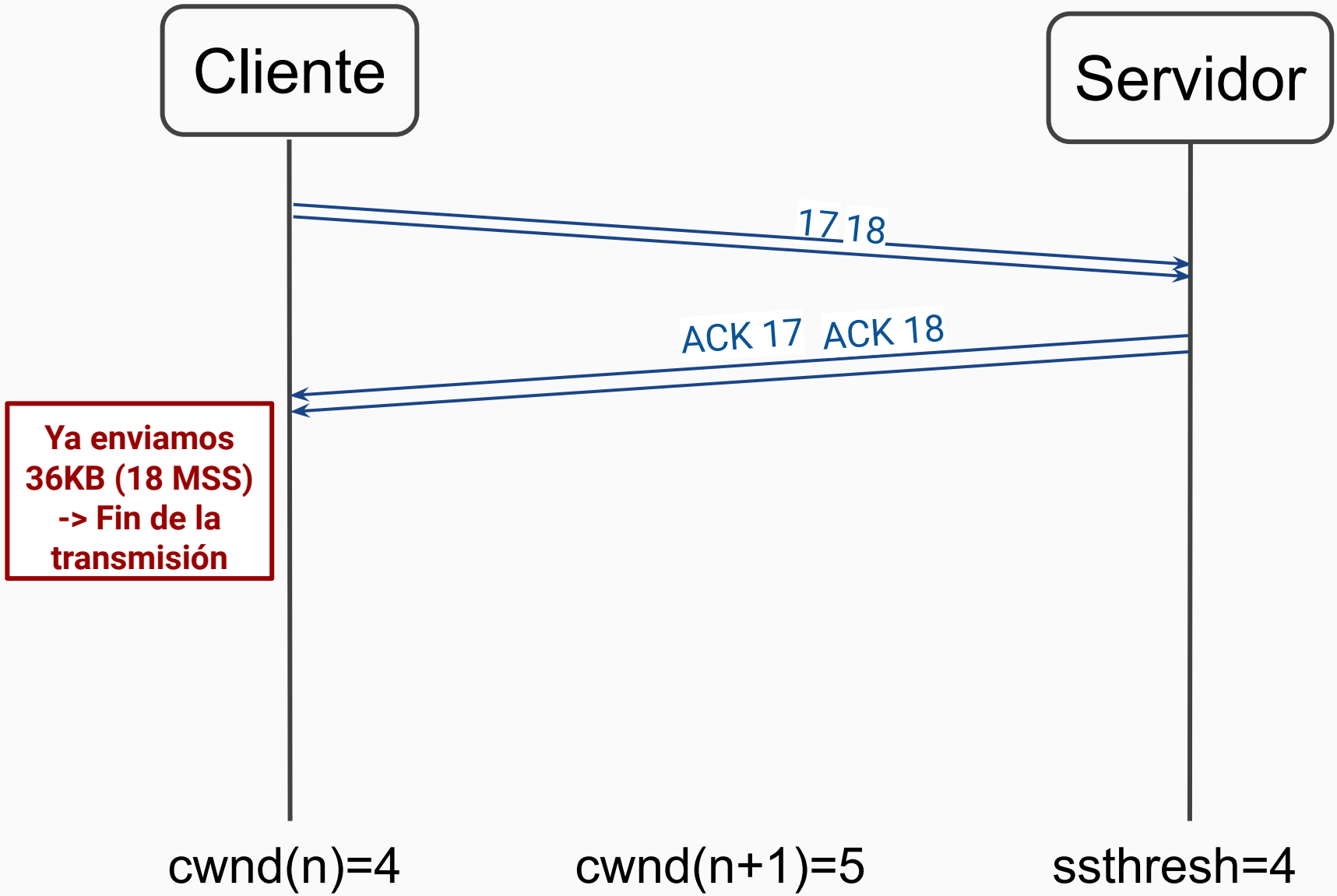
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
ACK 14
cwnd(n)=1
cwnd(n+1)=2
ssthresh=4

Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
ACK 14
15
16
ACK 15
ACK 16
cwnd(n)=2
cwnd(n+1)=?
ssthresh=4

Slow start


Cliente
Servidor
17 18
ACK 17 ACK 18
Ya enviamos 36KB (18 MSS) -> Fin de la transmisión
cwnd(n)=4
cwnd(n+1)=5
ssthresh=4

Pérdida de paquetes

- **Ahora, veamos el caso de Reno**
- Supongamos que se pierde **UN SOLO** segmento de la ráfaga en cuestión.
- Y además, que estamos usando el algoritmo de control de congestión de Reno

TCP : control de congestión

Algoritmo TCP Reno

- Cuando se reciben 4 ACKs iguales, se retransmite el siguiente segmento
- cwnd(n+1) = cwnd(n) / 2
- sshtresh = cwnd(n) / 2
- Se continúa con la fase de CA en vez de pasar a **Slow Start**
- Esto se denomina **Fast Recovery**

TCP : control de congestión

Otros algoritmos

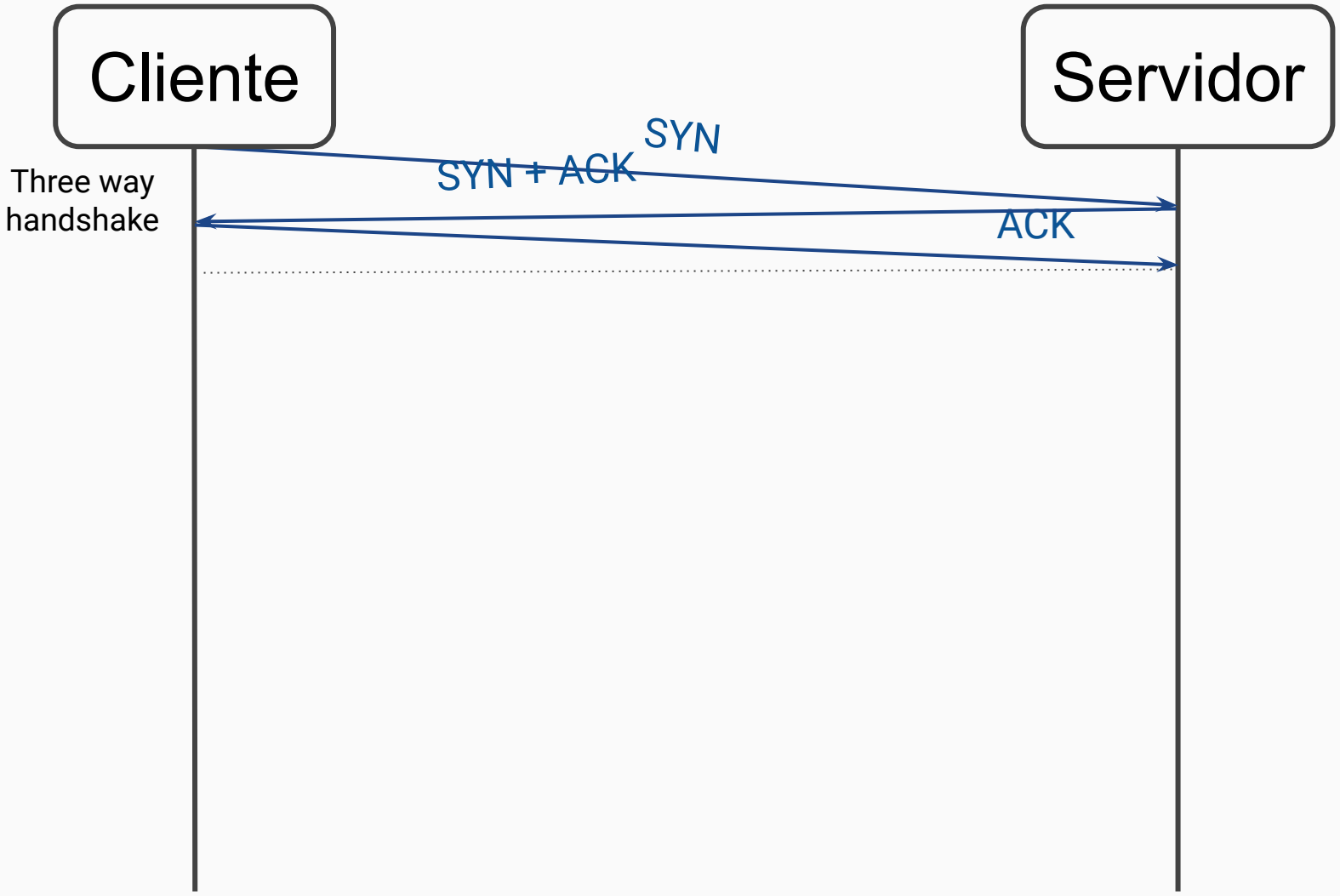
- TCP Tahoe y Reno son los más simples a nivel didáctico
- En la práctica ya no se utilizan. Hoy en día se usan algoritmos más modernos y complicados

TCP : control de congestión

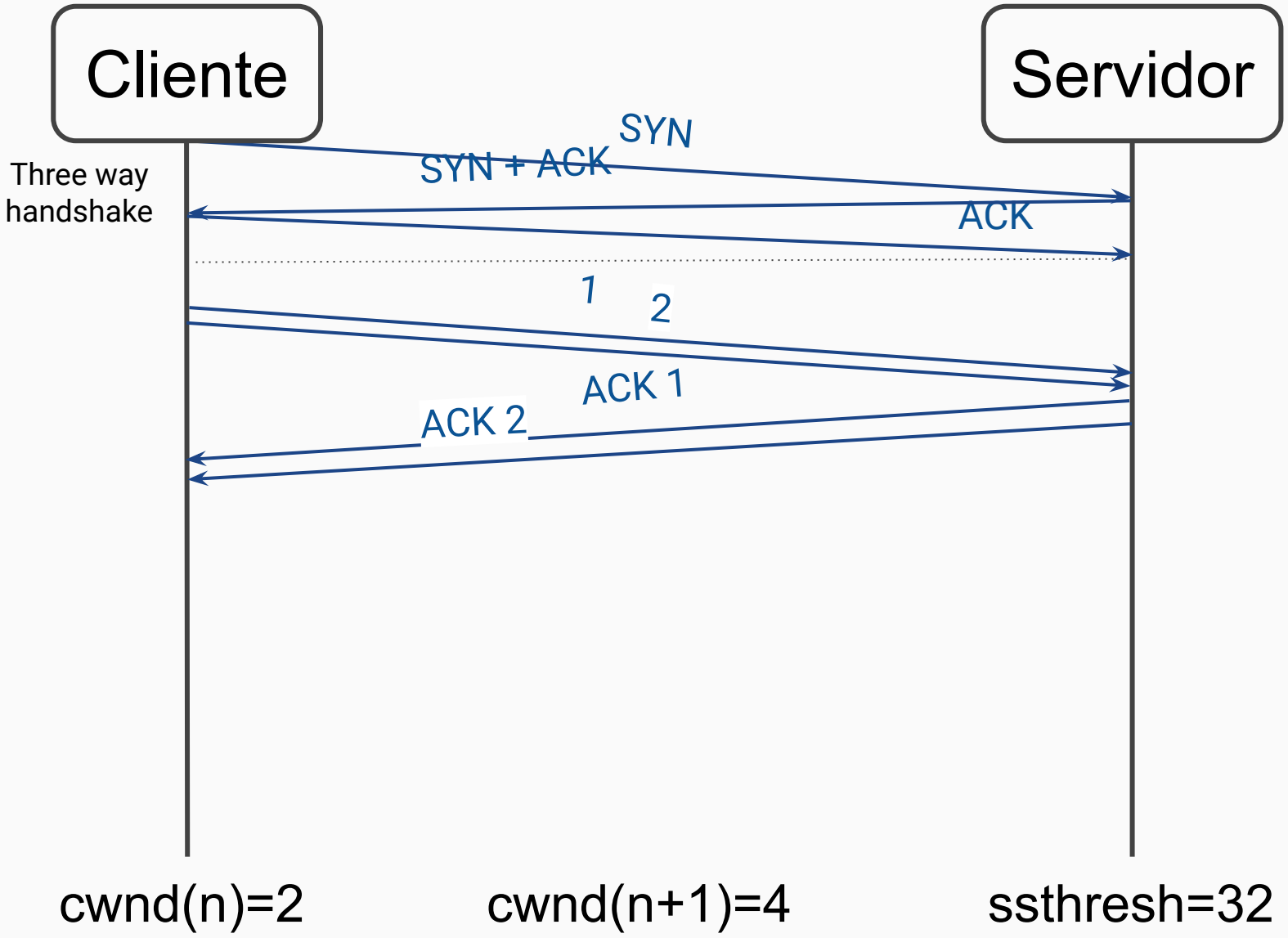
Otros algoritmos

- Algunos ejemplos
  - TCP Vegas
  - TCP New Reno (viene por default en FreeBSD)
  - TCP CUBIC (viene por default en Linux)

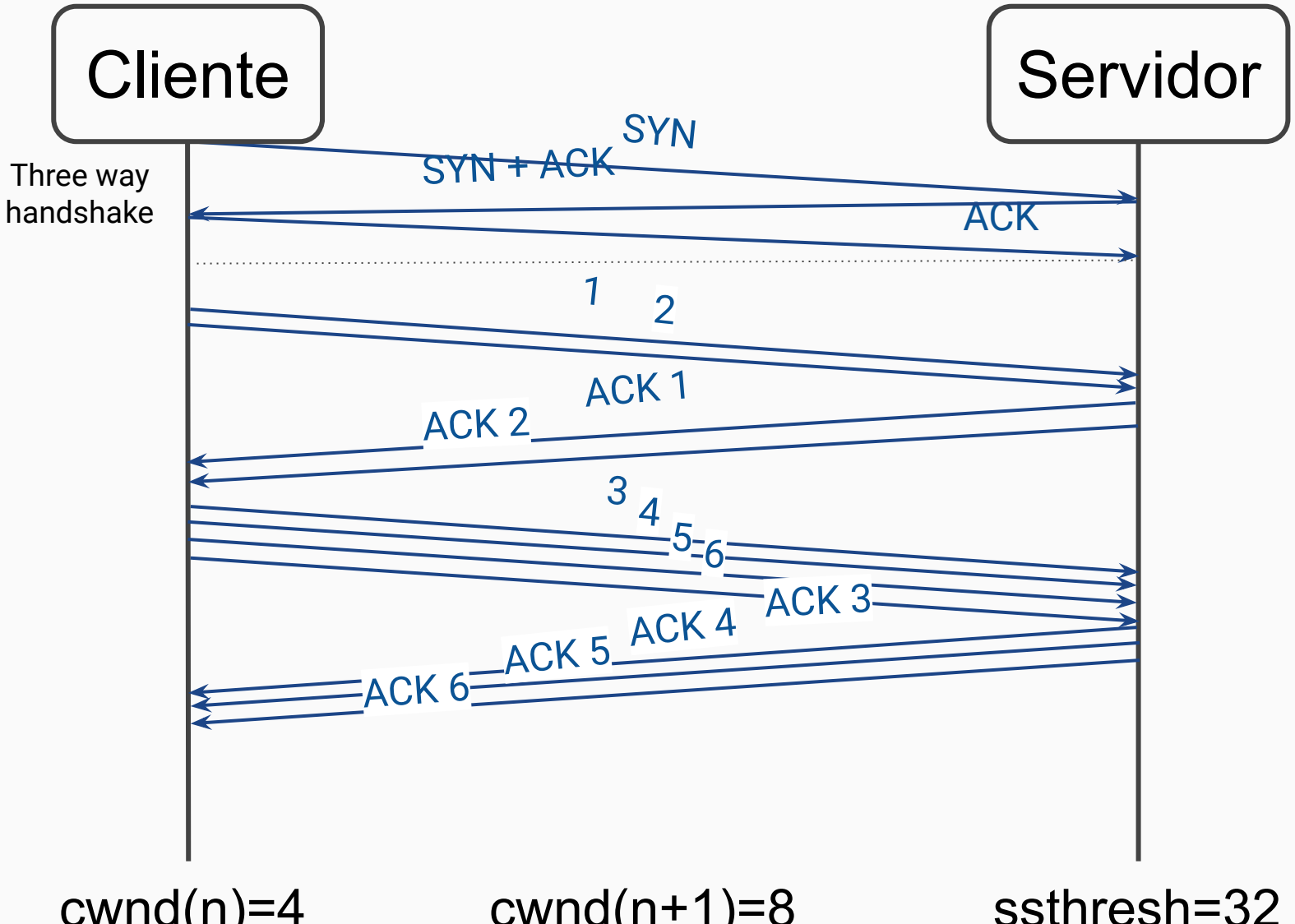
Three way handshake



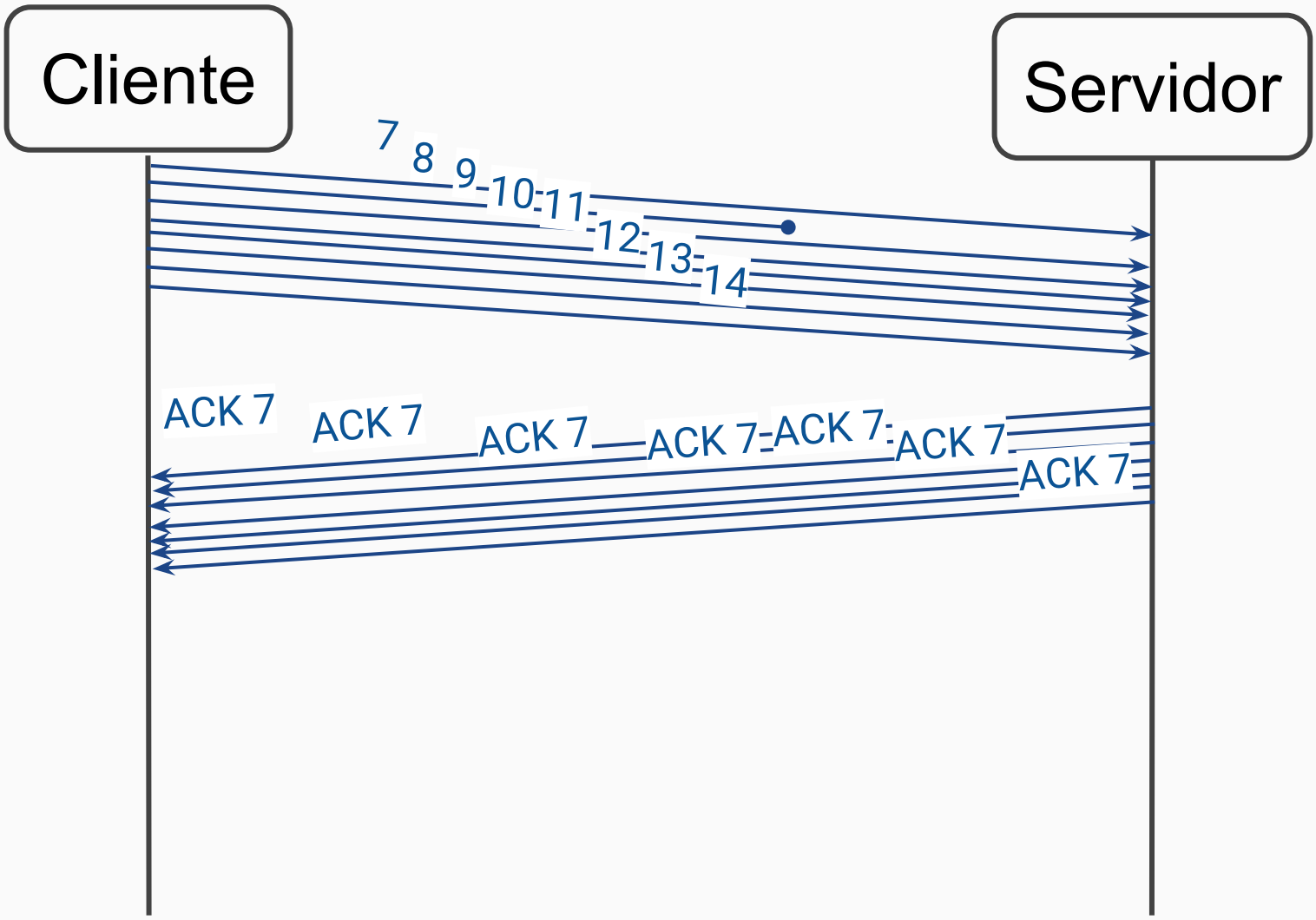
Slow start


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
cwnd(n)=2
cwnd(n+1)=4
ssthresh=32

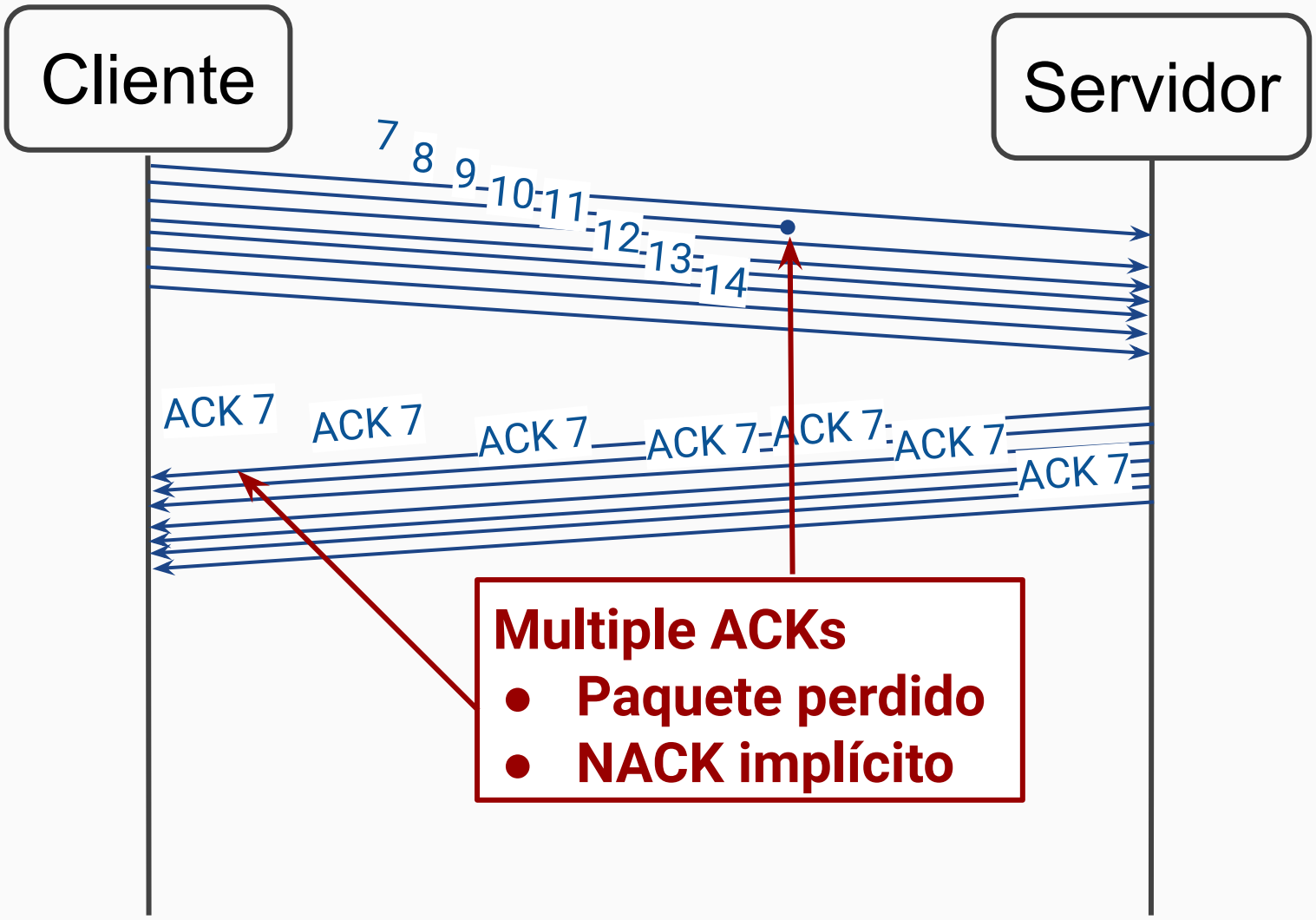
Slow start


Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
3
4
5
6
ACK 3
ACK 4
ACK 5
ACK 6
cwnd(n)=4
cwnd(n+1)=8
ssthresh=32

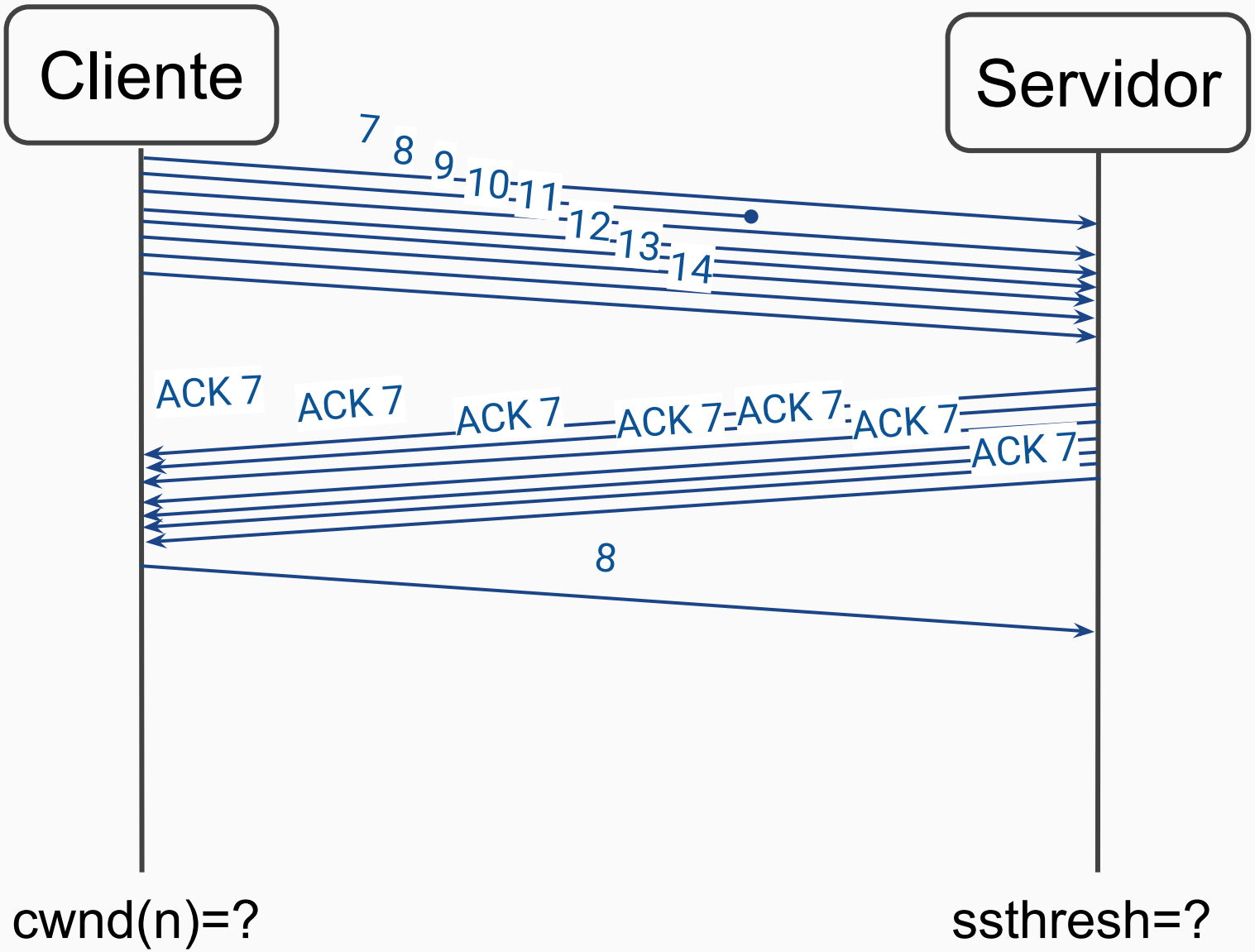
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
cwnd(n)=8
cwnd(n+1)=?
ssthresh=32

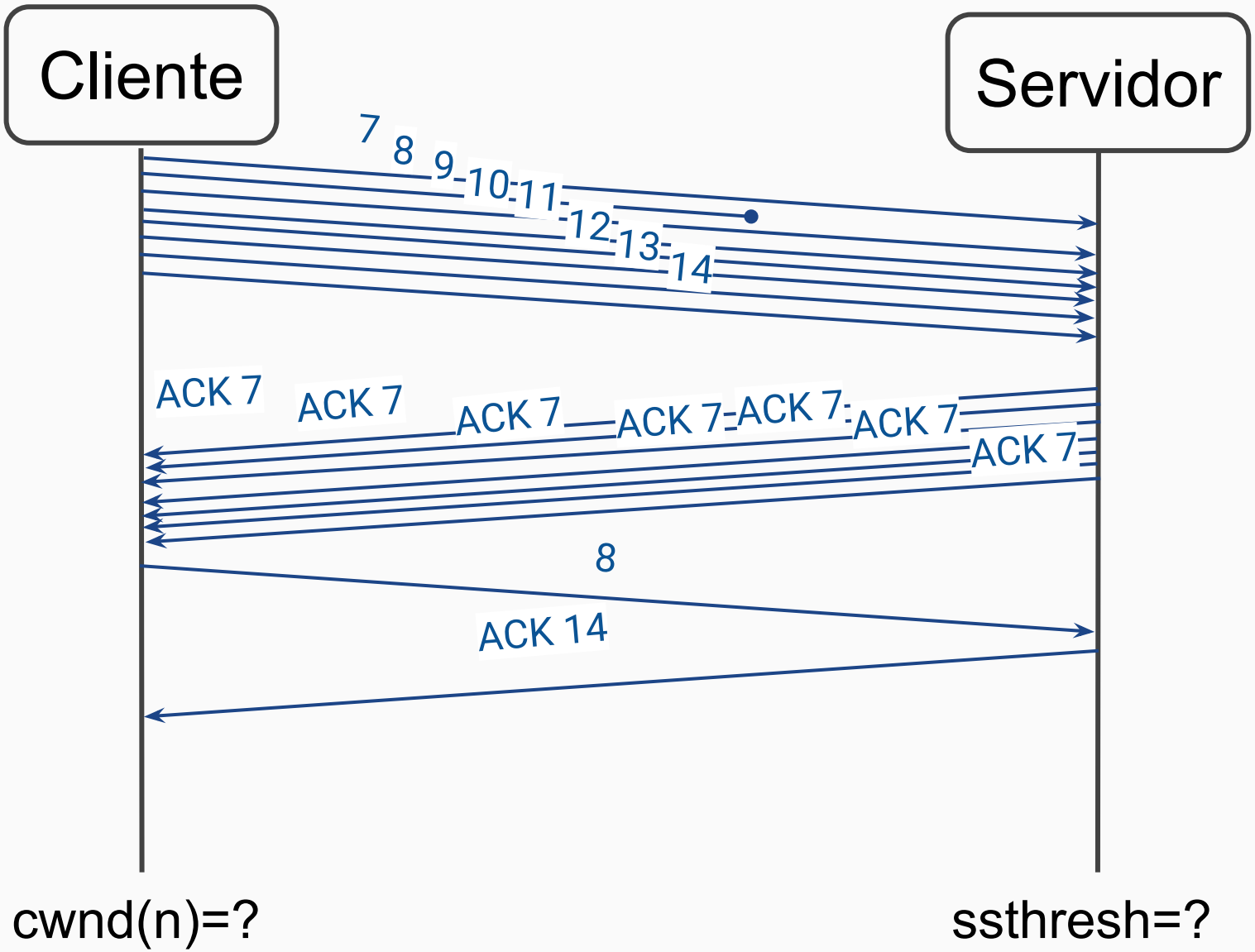
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
Multiple ACKs
Paquete perdido
NACK implícito
cwnd(n)=8
cwnd(n+1)=?
ssthresh=32

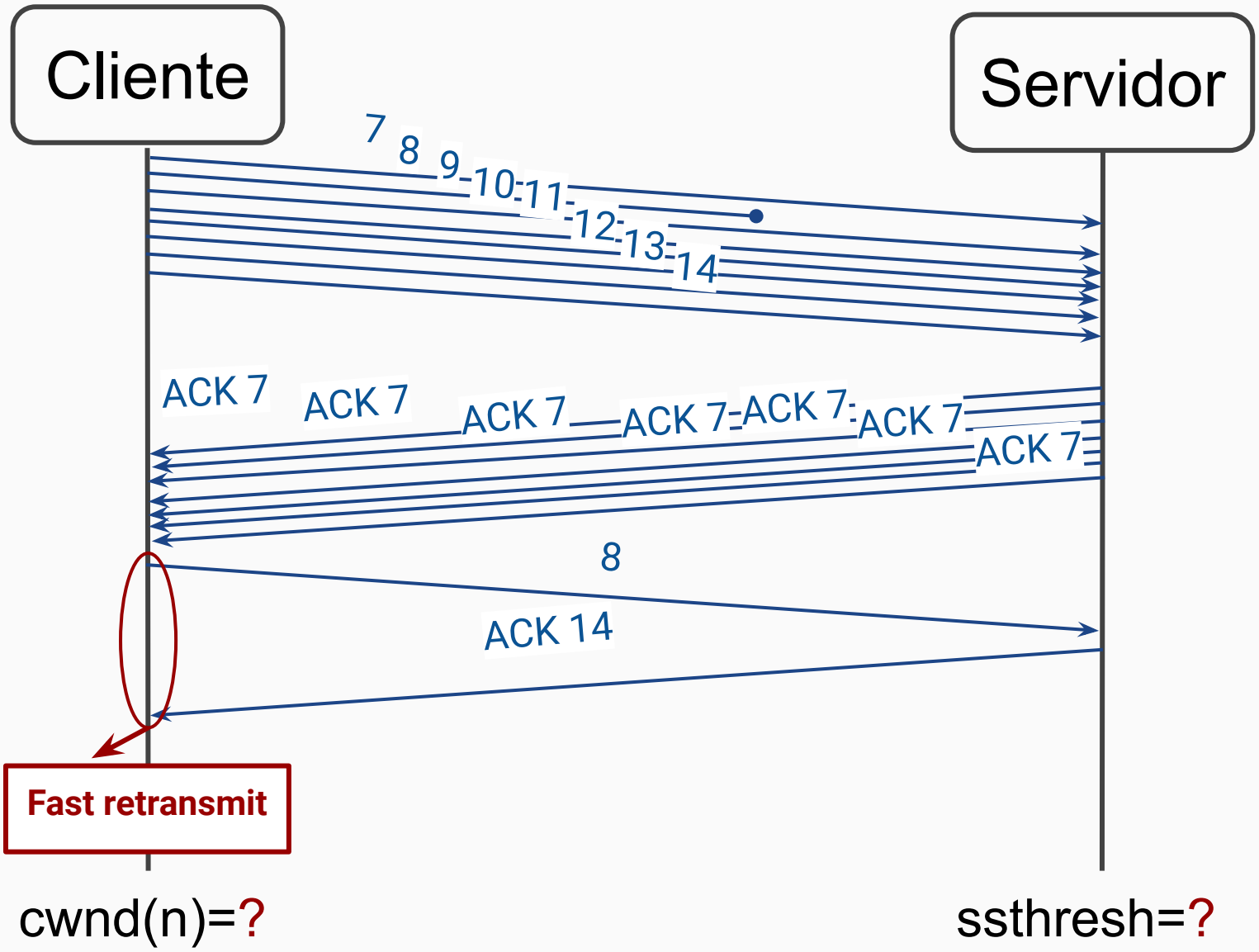
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
cwnd(n)=?
ssthresh=?

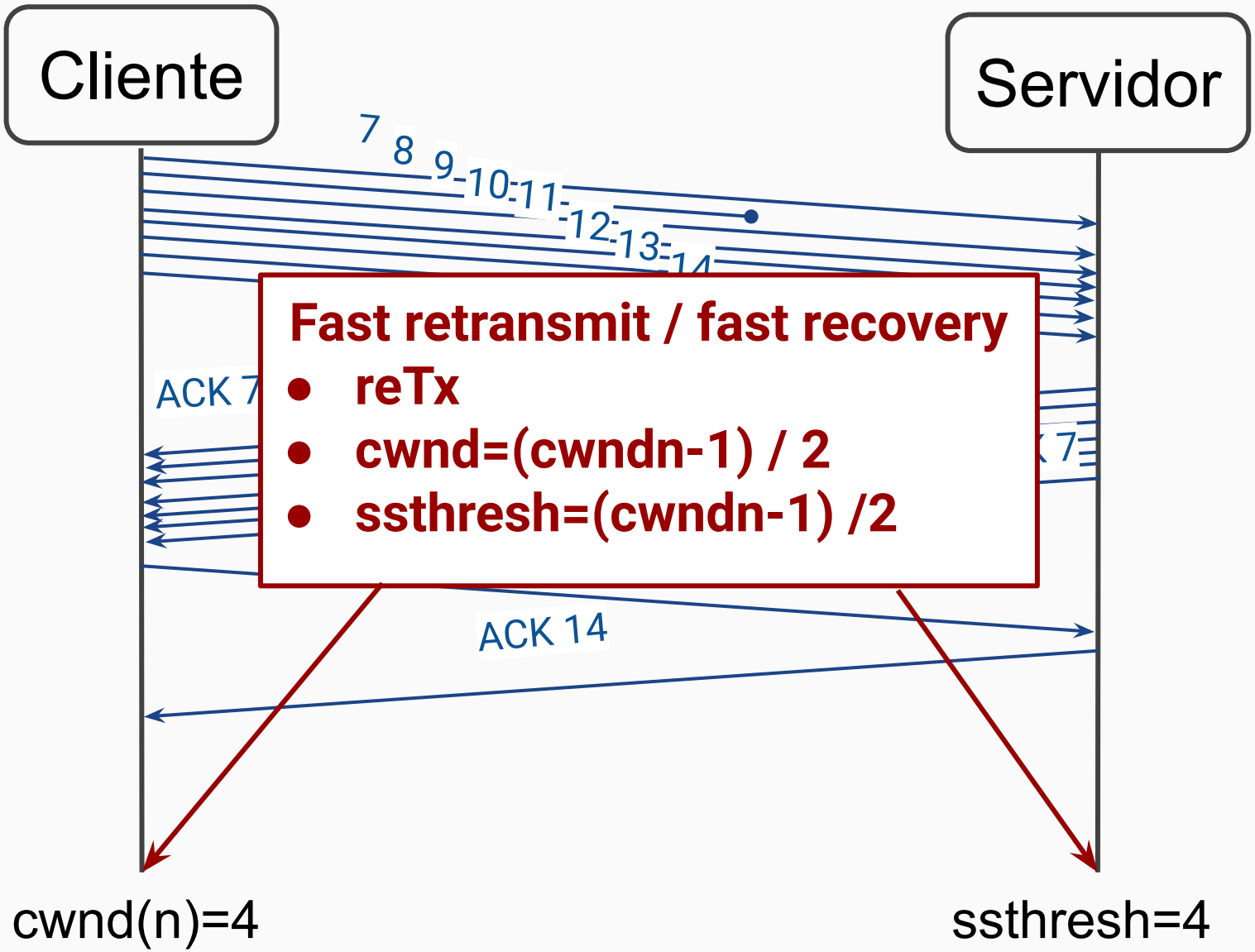
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
ACK 14
cwnd(n)=?
ssthresh=?

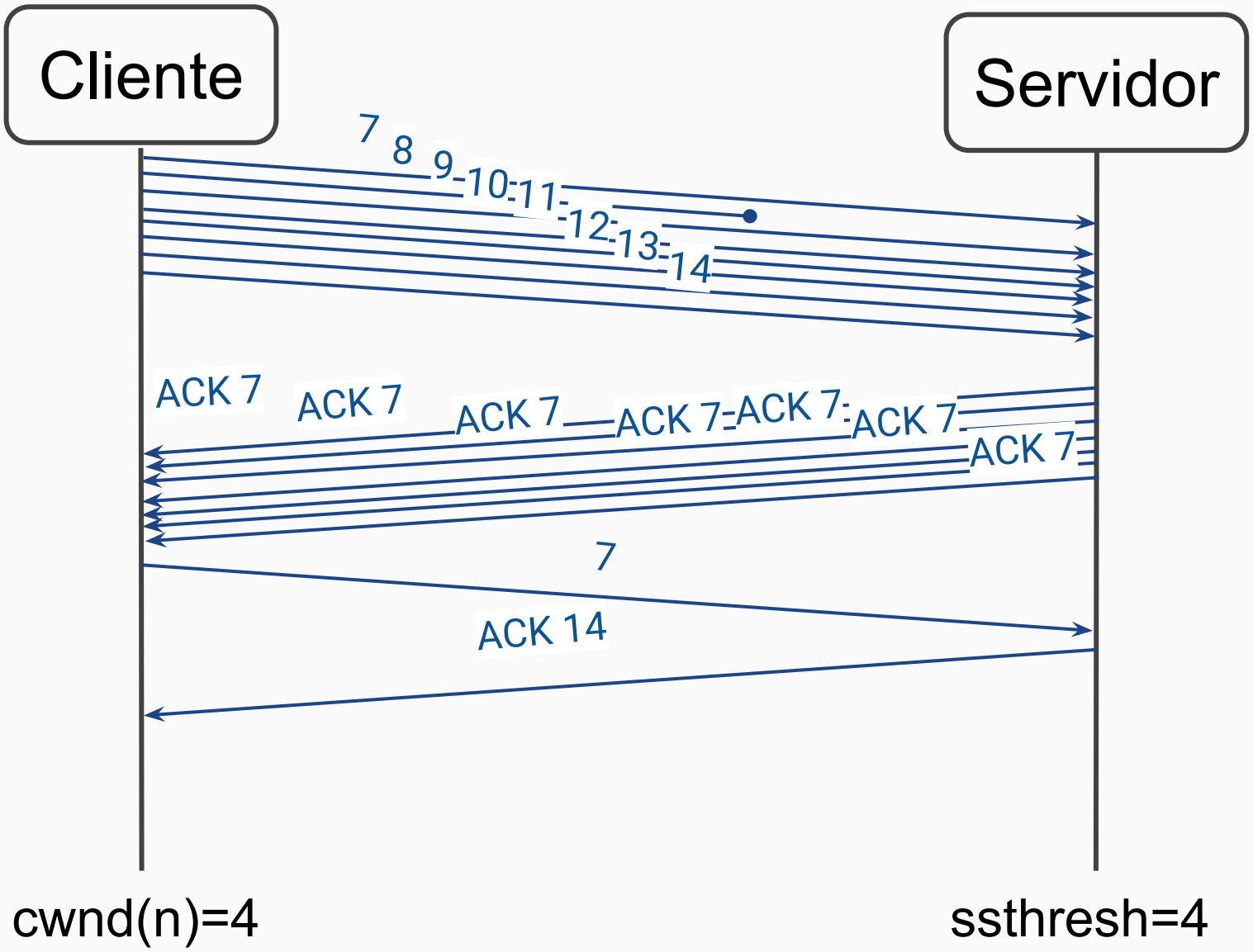
Fast retransmit



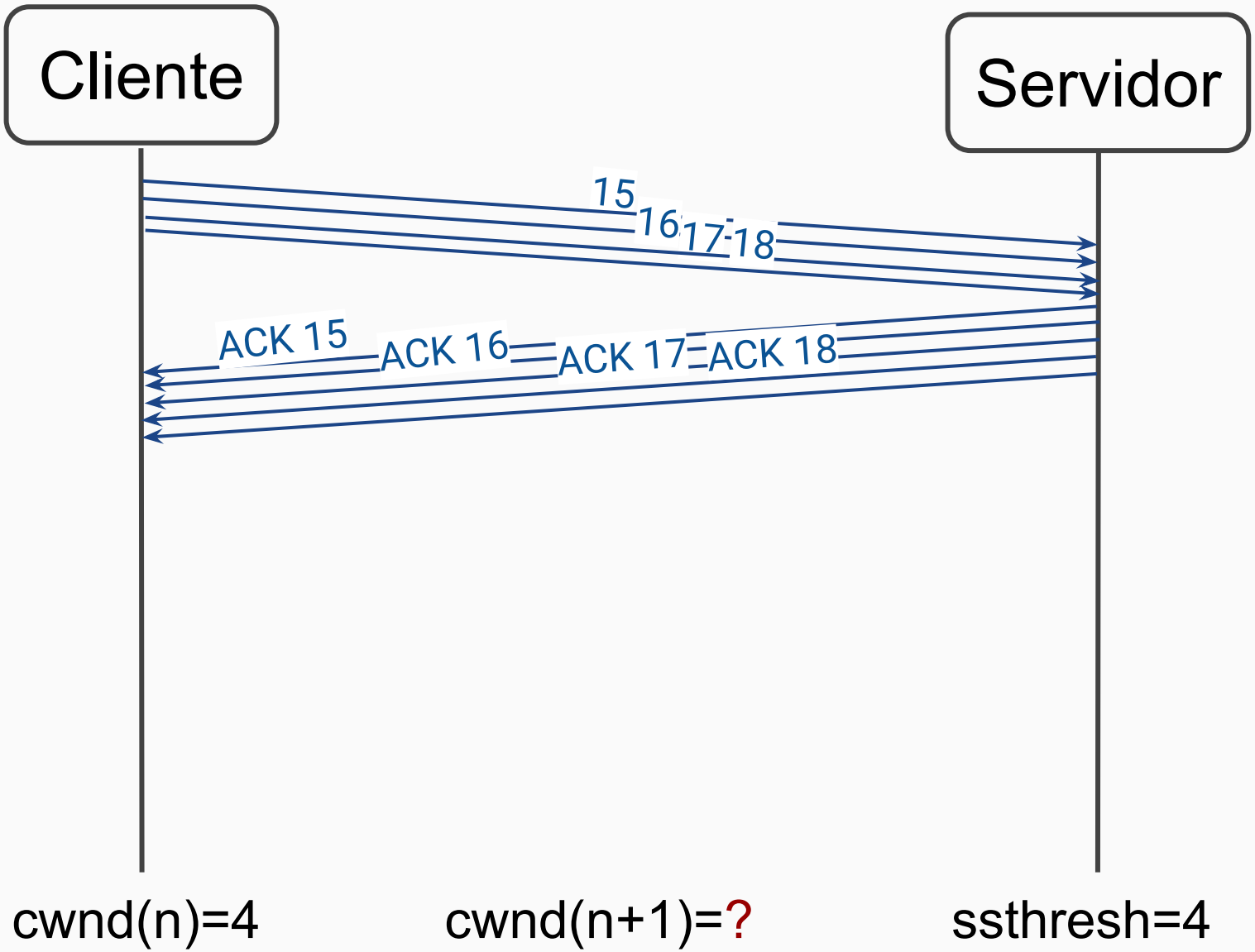
Fast retransmit



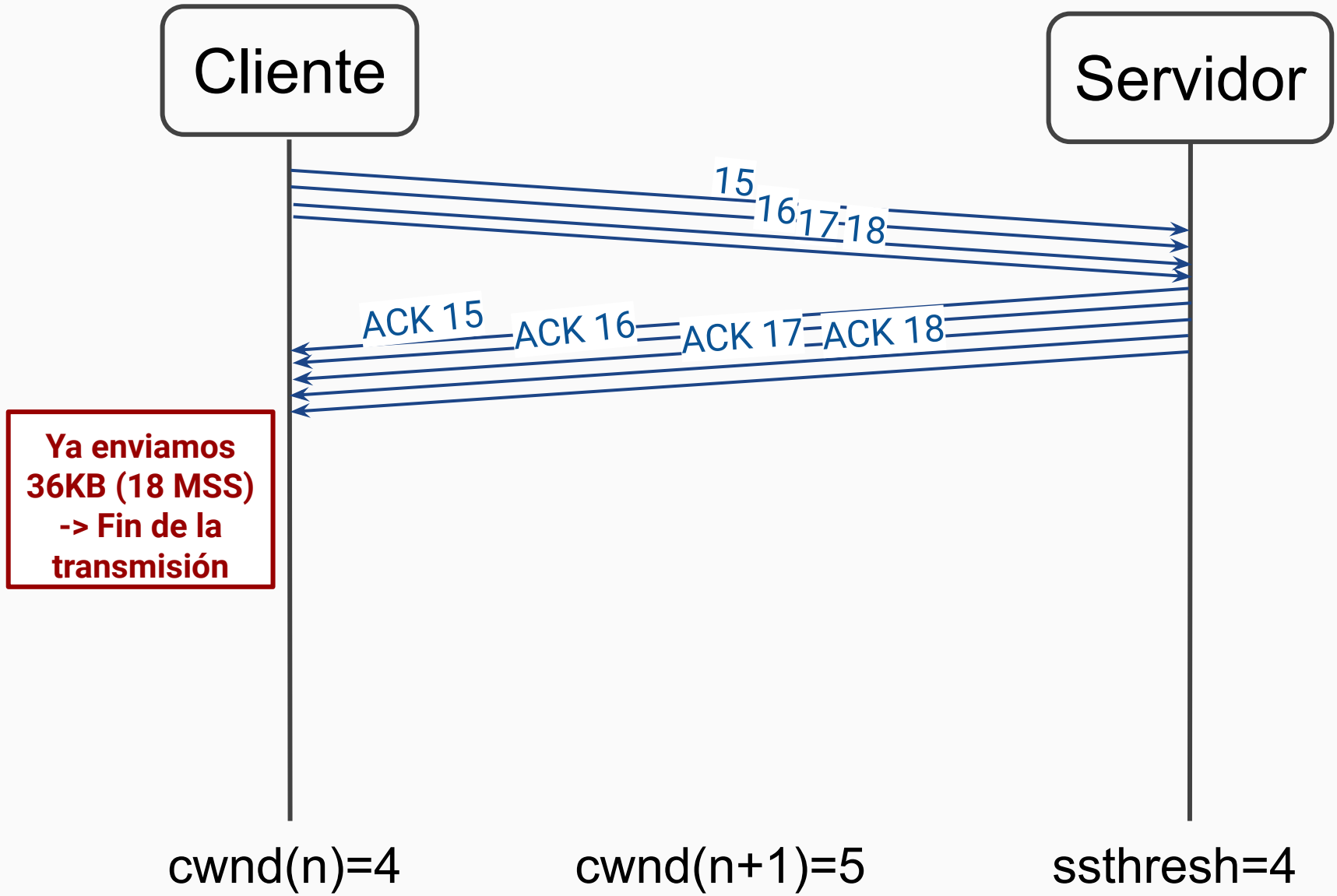
Fast retransmit



Fast retransmit



Fast retransmit



¿Preguntas?

Close TCP

TCP states

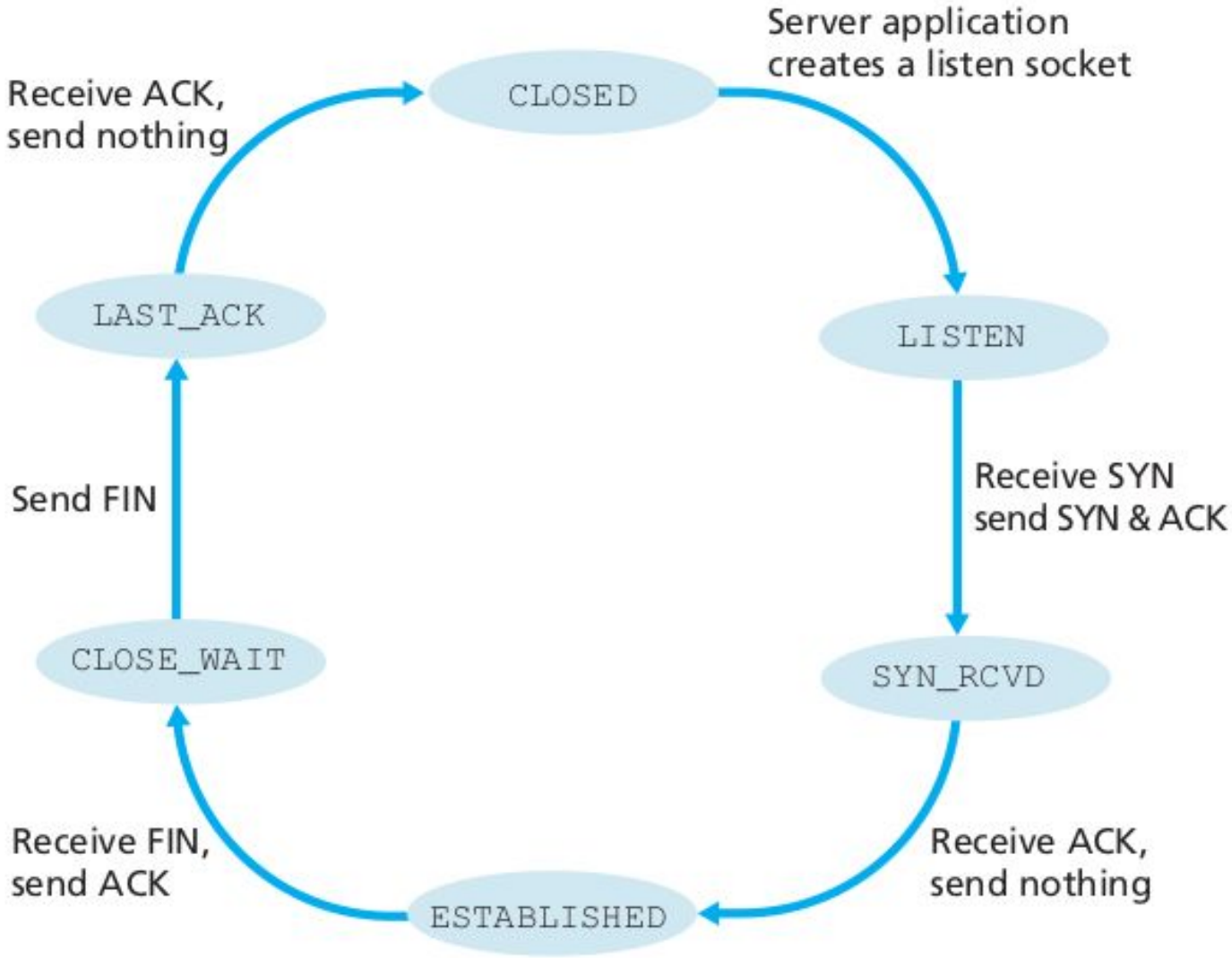


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

TCP states

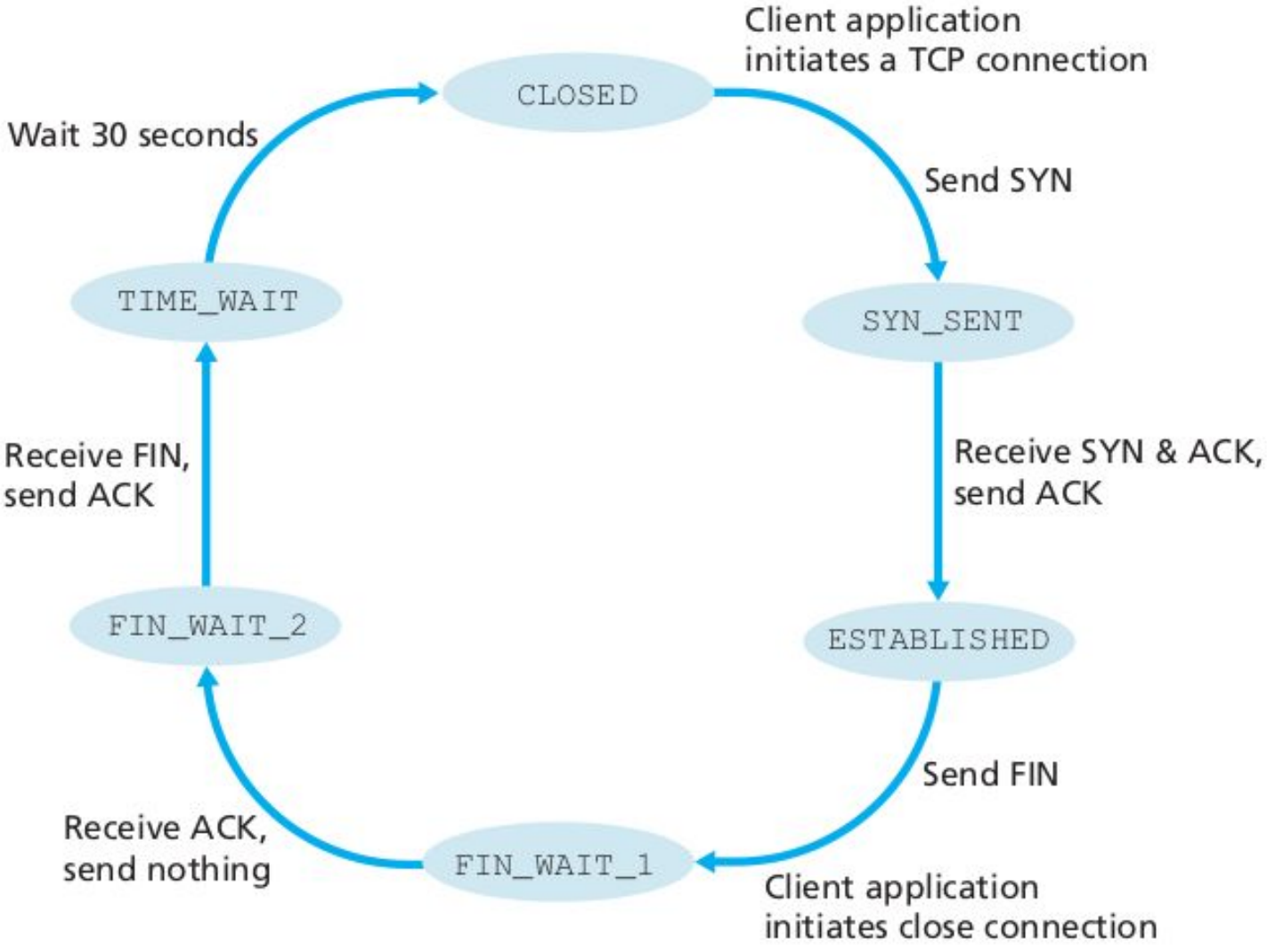
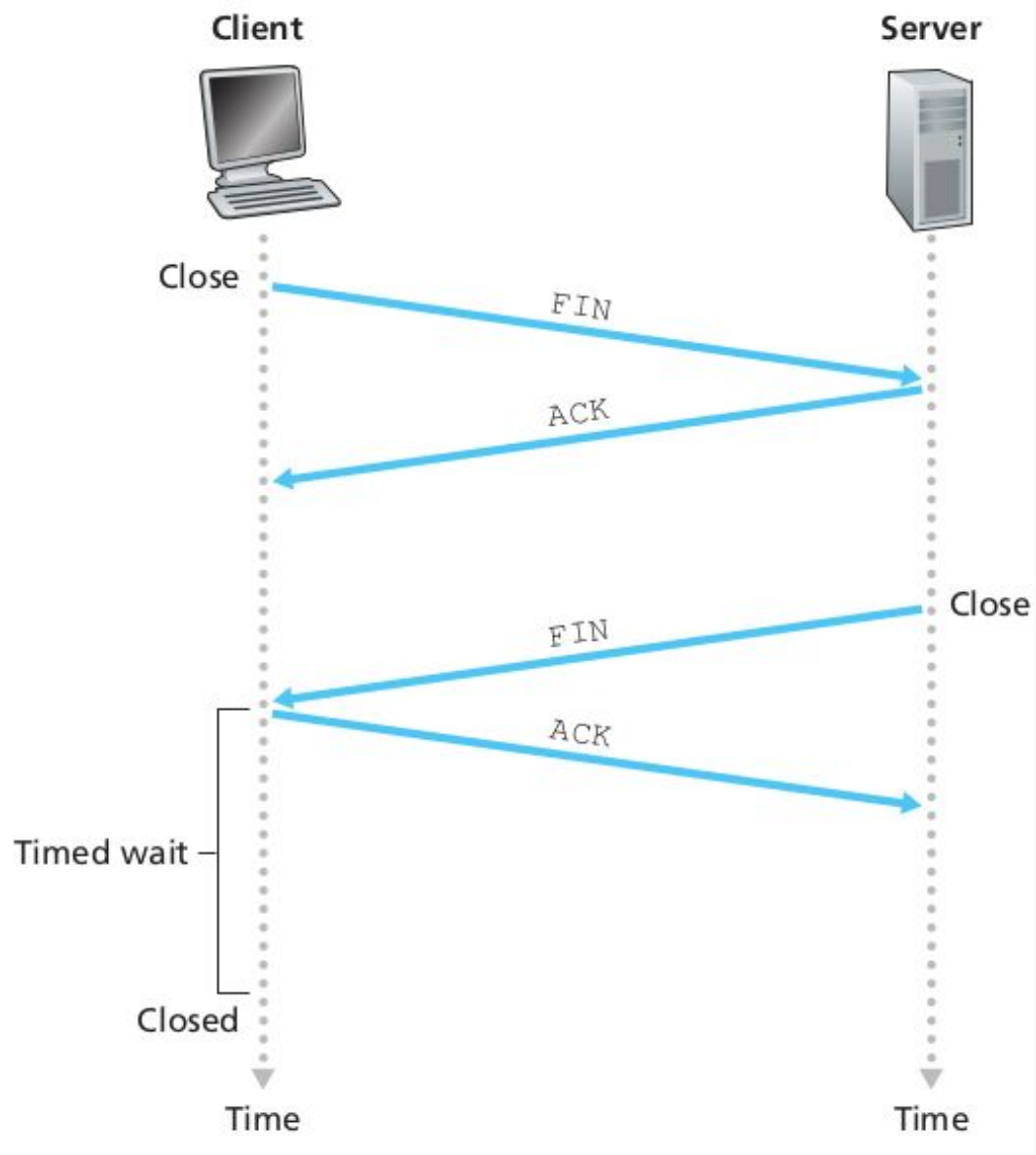


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

Close messages



Split TCP

Split TCP

- **Problema**
  - Latencia alta.

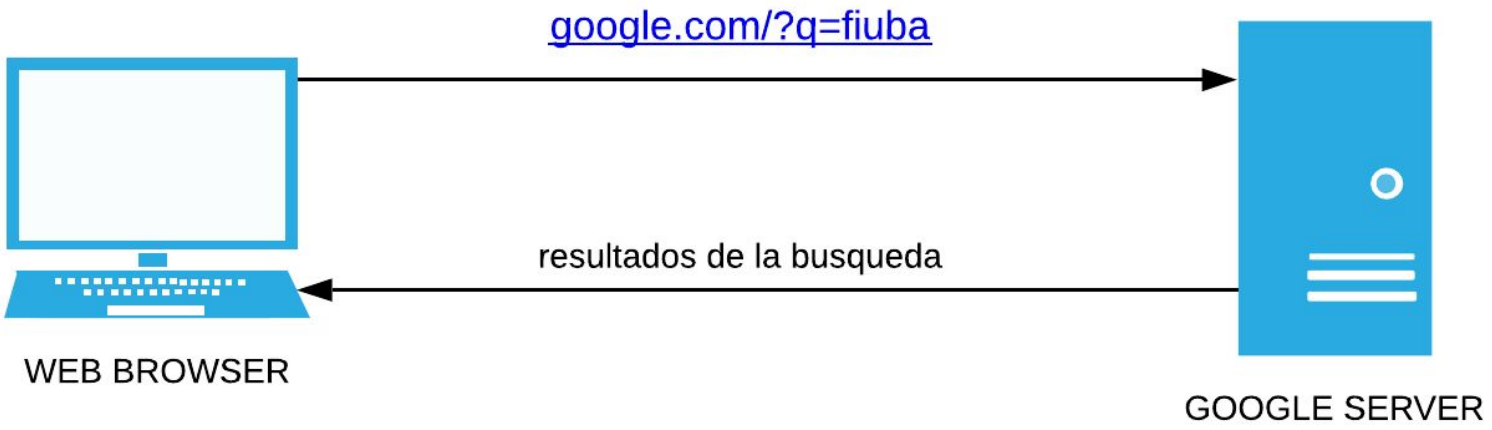
Split TCP

- **Problema**
  - Latencia alta.
  - Recursos dinamicos.

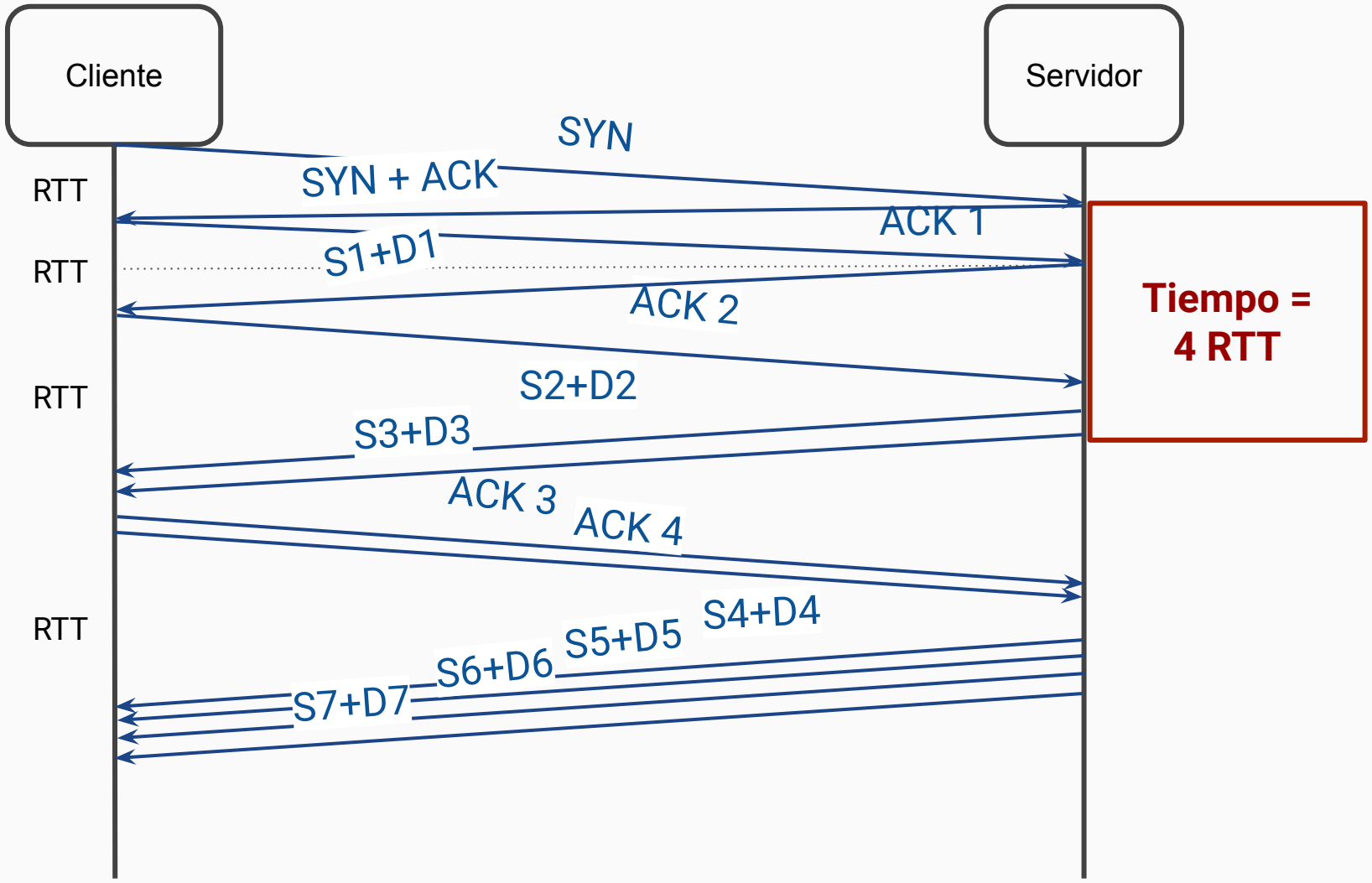
Split TCP - Ejemplo

- Ejemplo Kurose: HTTP search query
- “Typically, the server requires three TCP windows during slow start to deliver the response [Pathak 2010]”
- A estos 3 RTT tenemos que sumar el RTT para el three way handshake

Split TCP - Ejemplo



Split TCP - Ejemplo



Split TCP - Ejemplo

- Tardamos 4 RTTs en enviar el resultado de la query
- Se podría mejorar?

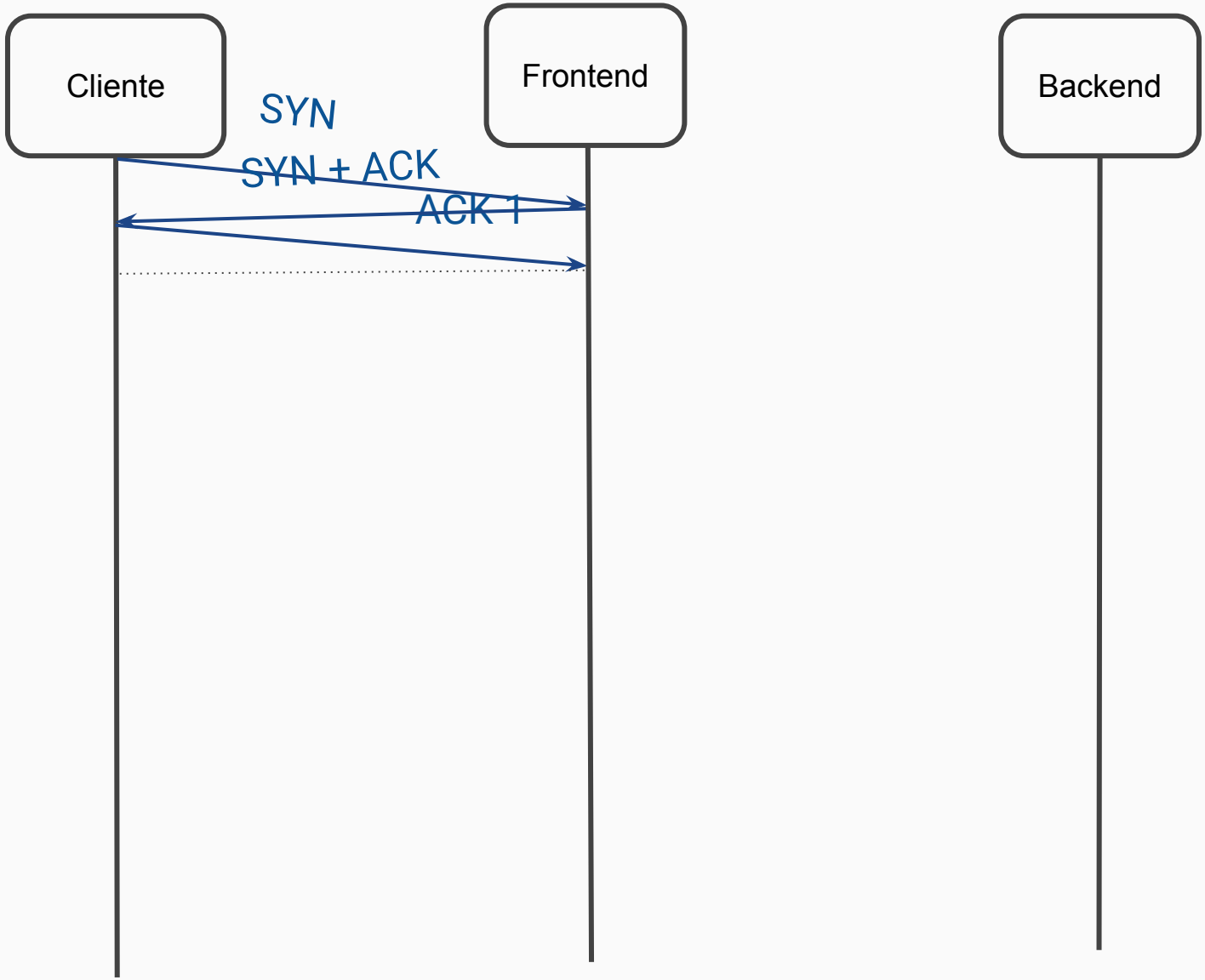
Split TCP

- Split TCP:
  - **Frontend**: cercano al cliente, ventanas más chicas

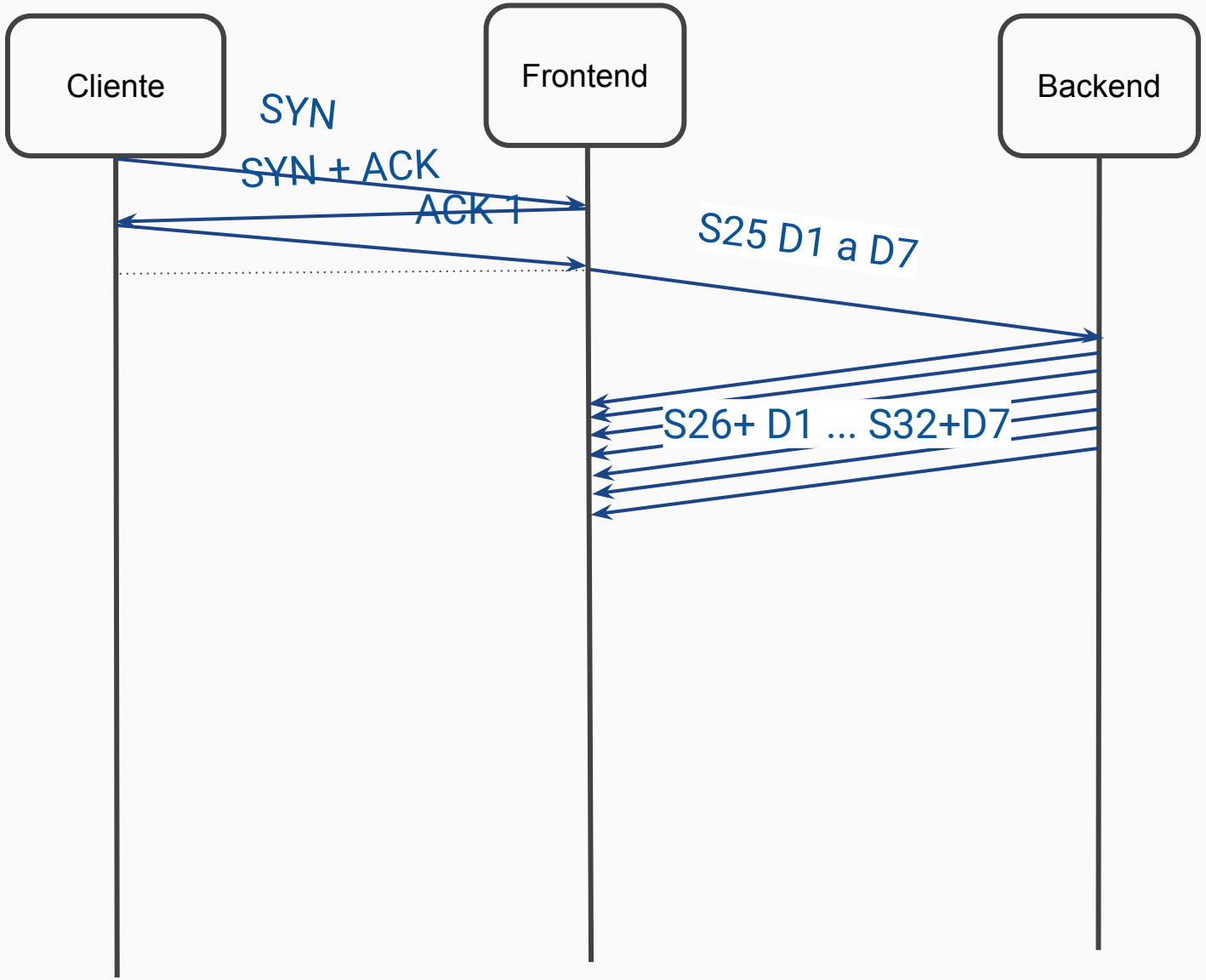
Split TCP

- Split TCP:
  - **Backend**: lejano al cliente
    - Tiene todo el tiempo una conexión activa con el **FE**
    - Ventanas más grandes

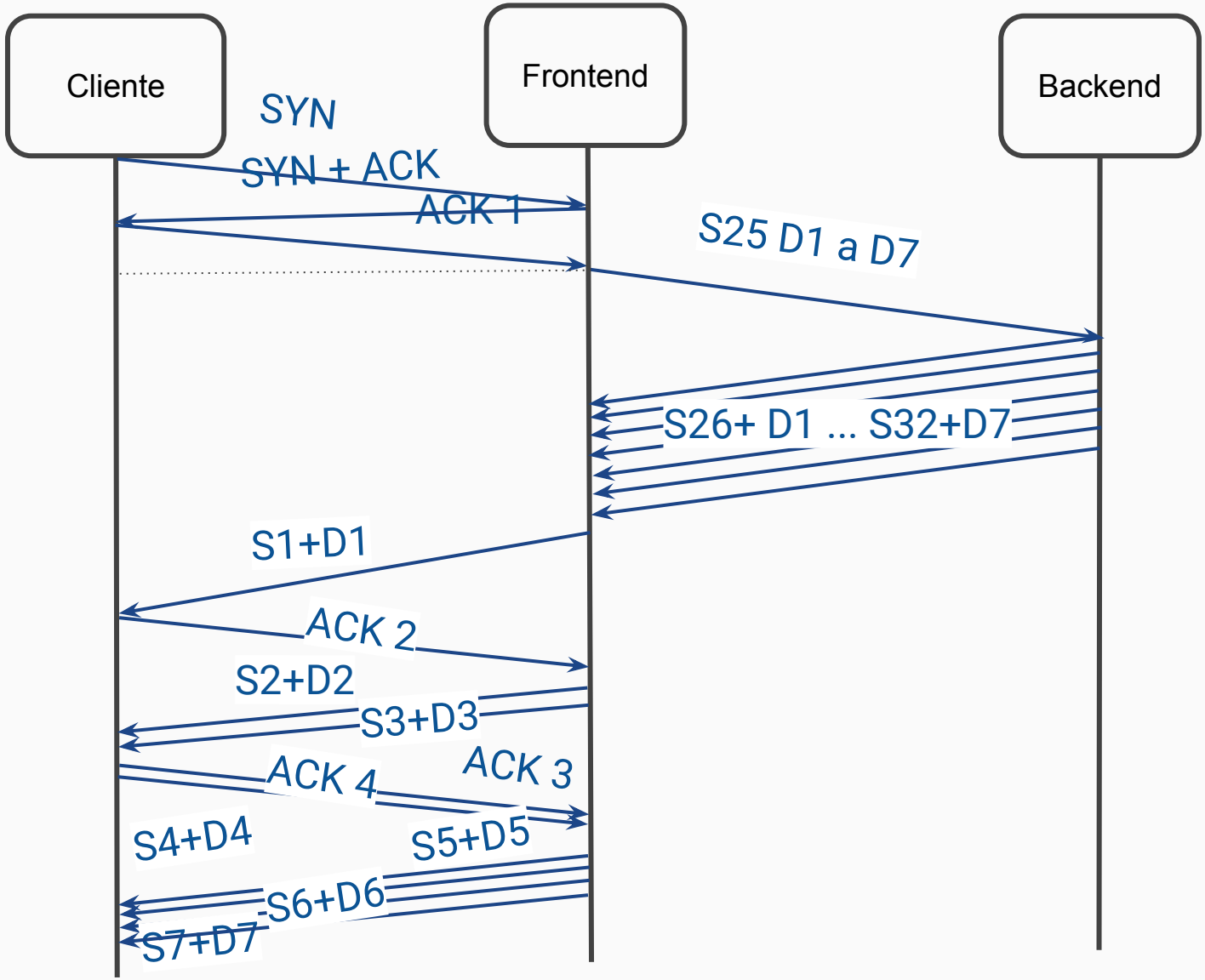
Split TCP - Ejemplo



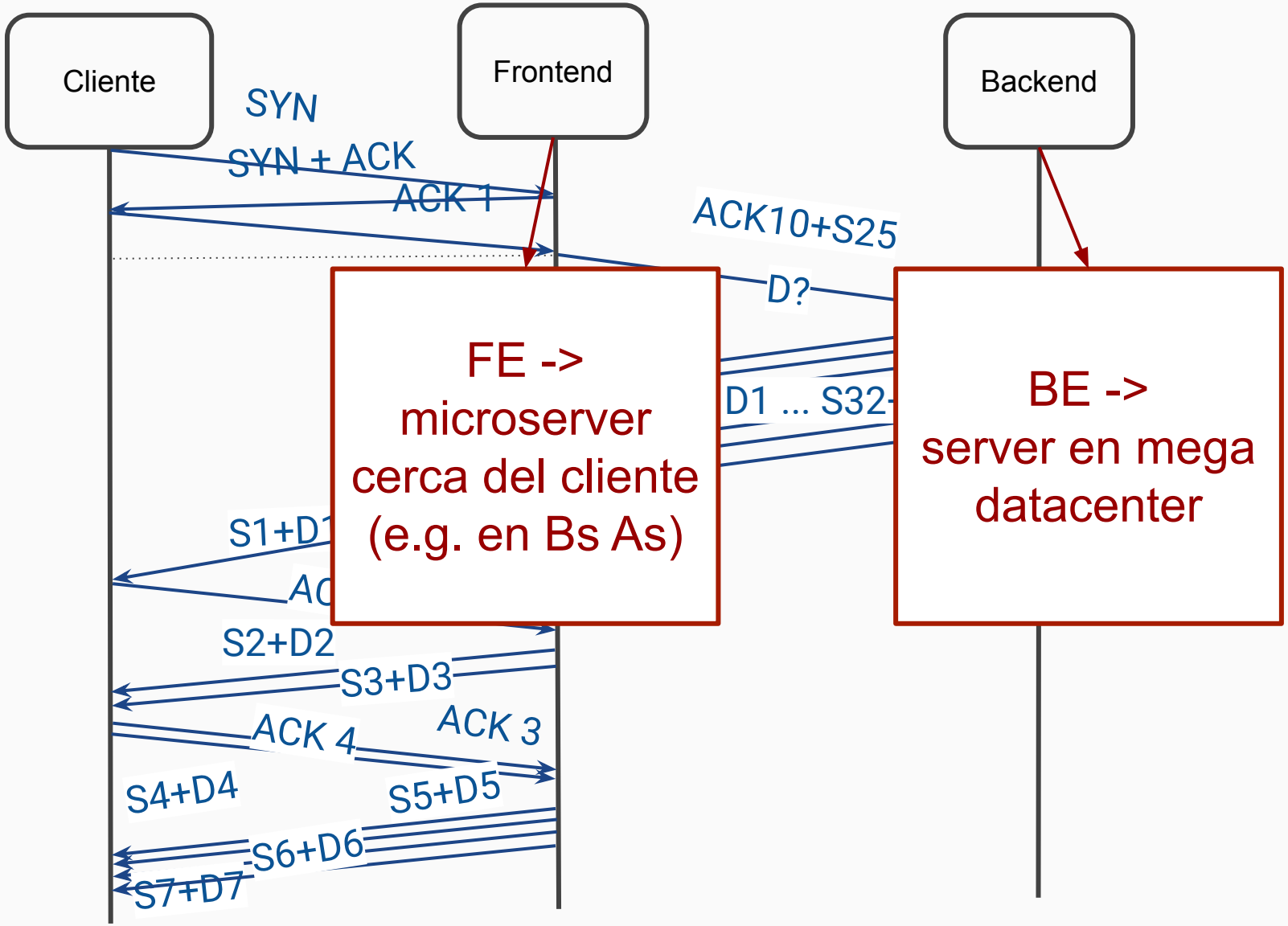
Split TCP - Ejemplo



Split TCP - Ejemplo



Split TCP - Ejemplo



Split TCP - Ejemplo