TCP: Wireshark

Introducción a los Sistemas Distribuidos (75.43)

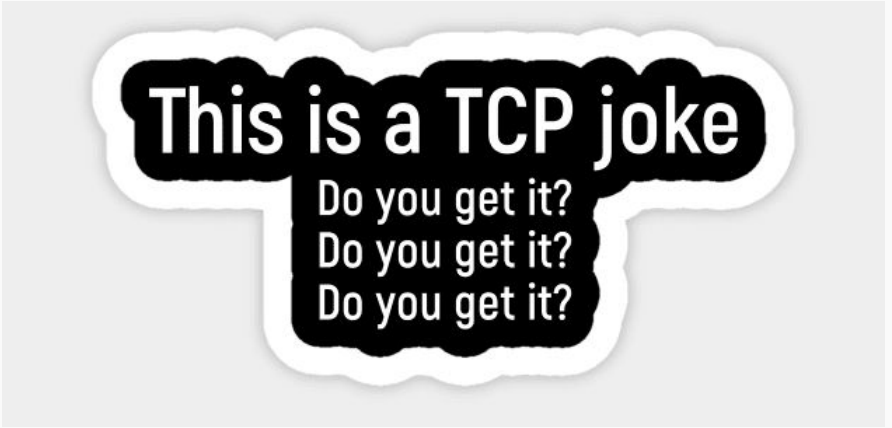
Universidad de Buenos Aires, Facultad de Ingeniería

TCP: Repaso

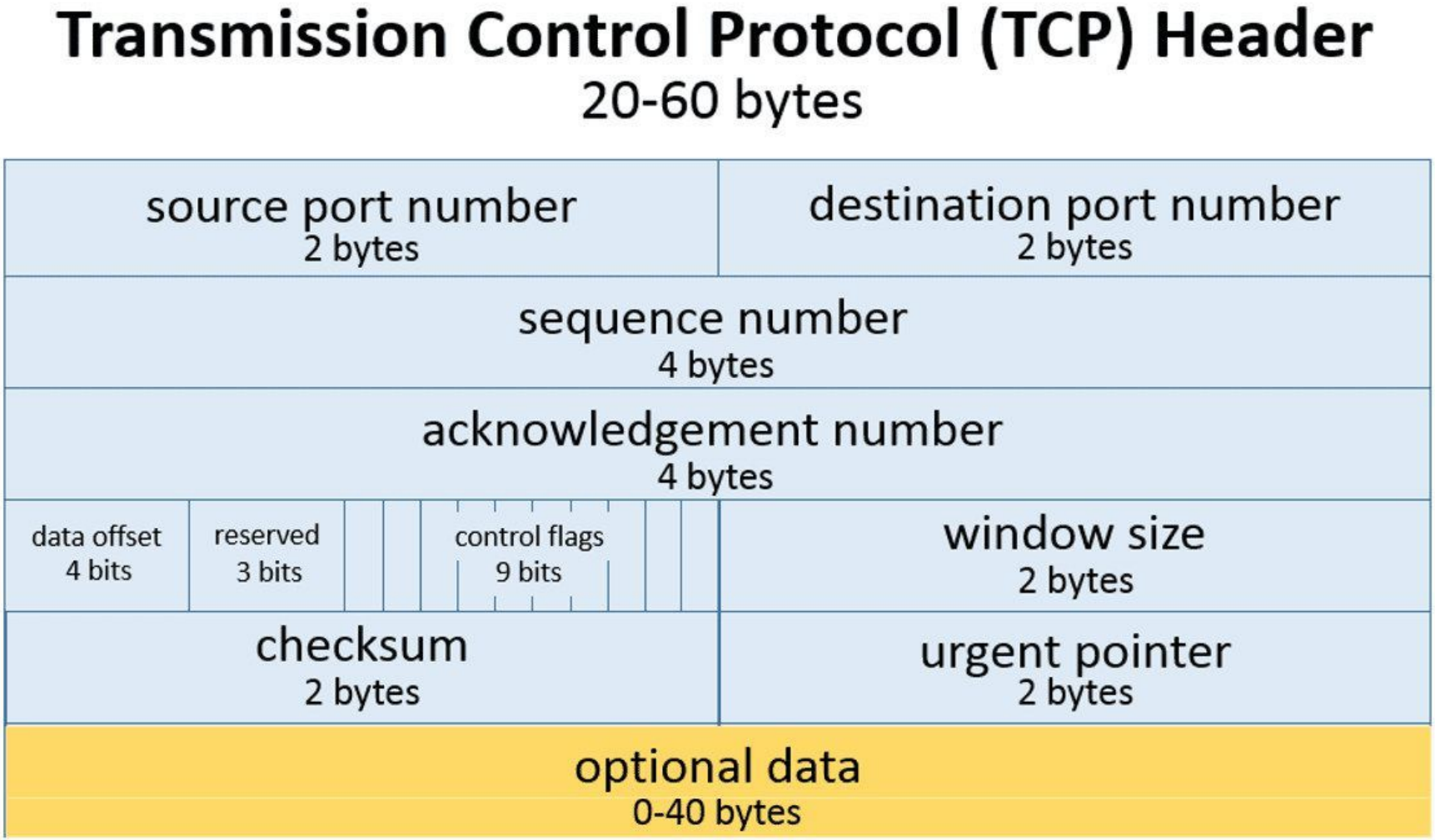
¿Qué características tiene TCP?

¿Qué nos ofrece?

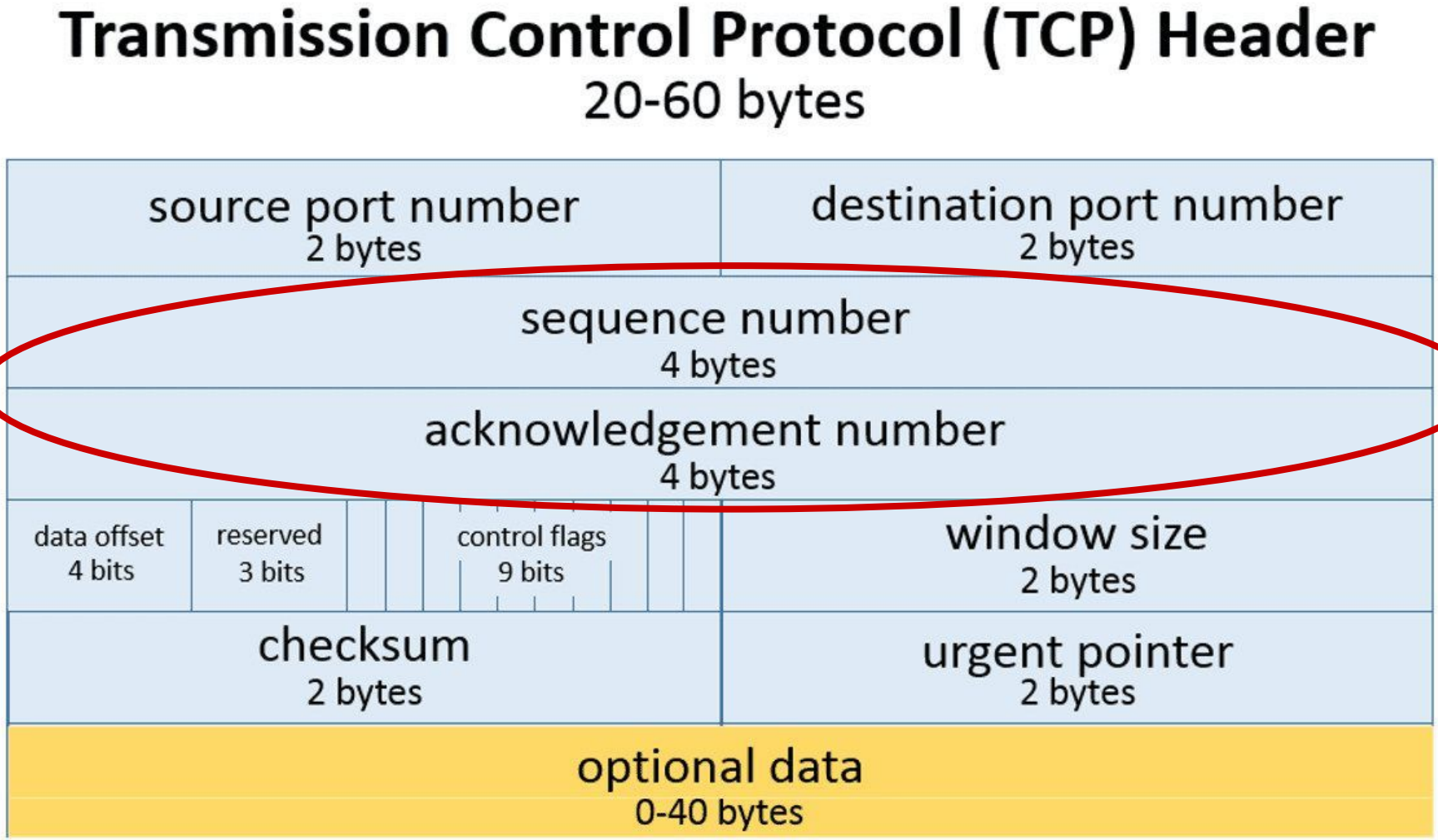
¿Ventajas y Desventajas?



TCP: Repaso


Transmission Control Protocol (TCP) Header
20-60 bytes
source port number
2 bytes
destination port number
2 bytes
sequence number
4 bytes
acknowledgement number
4 bytes
data offset
4 bits
reserved
3 bits
control flags
9 bits
window size
2 bytes
checksum
2 bytes
urgent pointer
2 bytes
optional data
0-40 bytes

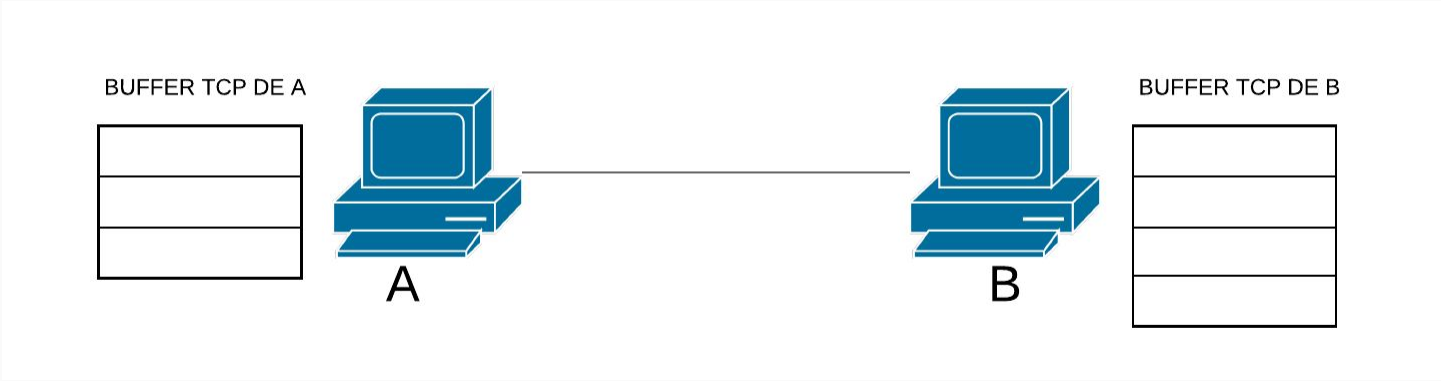
TCP: #Sequence y #ACK



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. Ojo 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

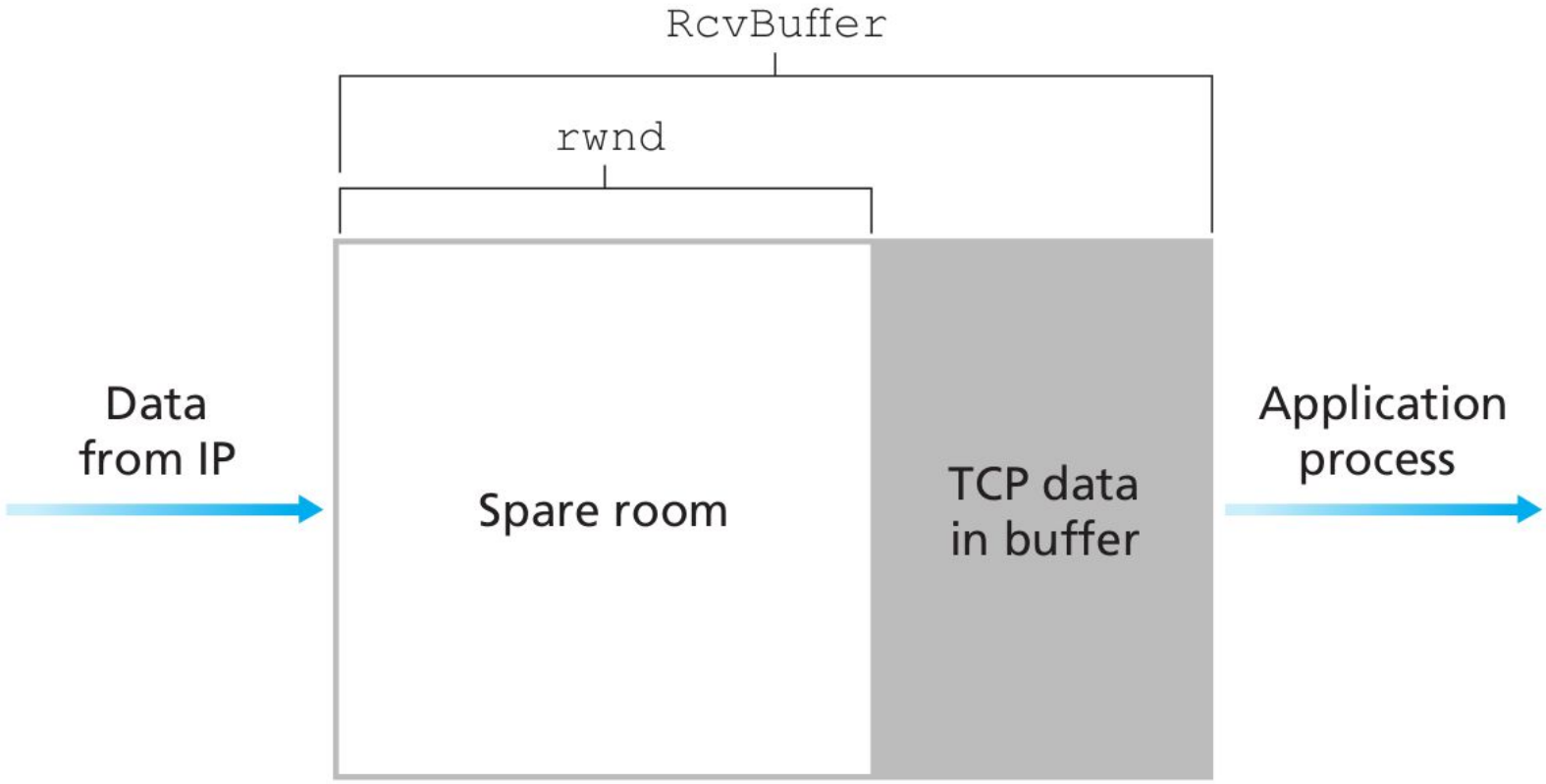
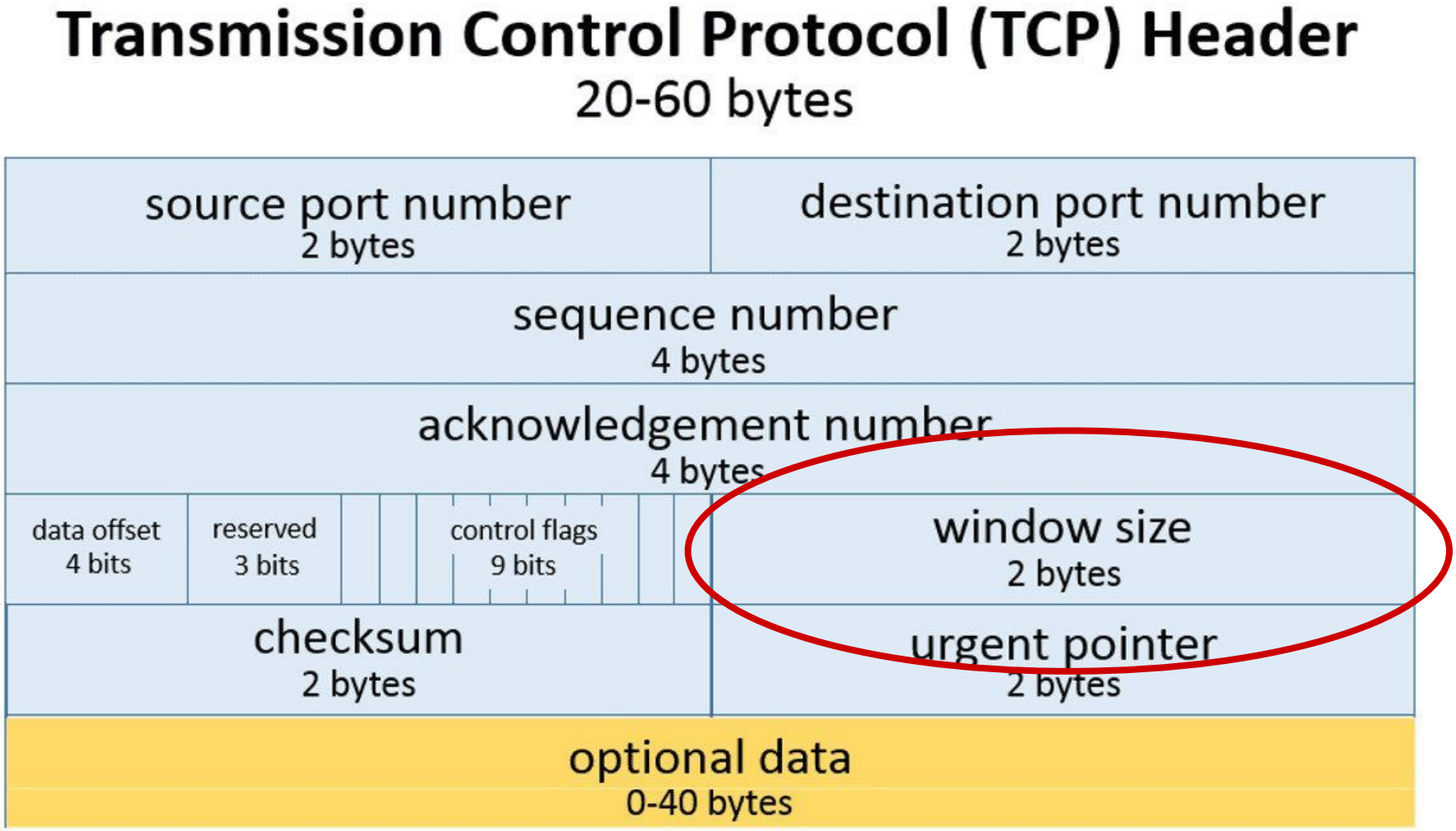


Figure 3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

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***

TCP: #Sequence y #ACK



Control de congestión

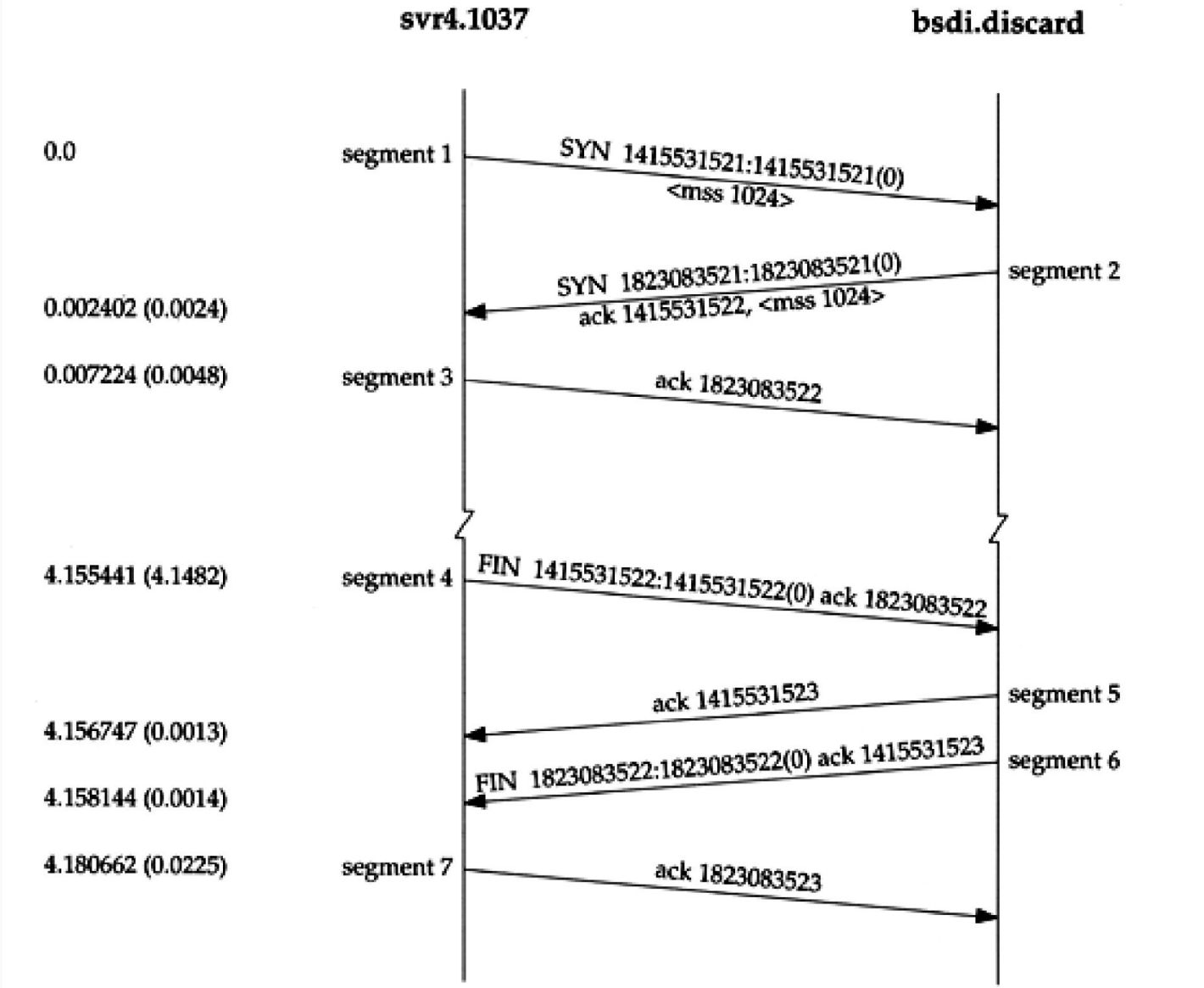
TCP : control de congestión

- 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



TCP : control de congestión

Disclaimer: Modelo aproximado a la realidad utilizado para entender el funcionamiento de estos algoritmos.

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
- **IW**: *initial window* es el valor inicial de cwnd. Tiene que ser <= 2 segmentos.

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
- cwnd medido en **MSSs** (segmentos de tamaño máximo)

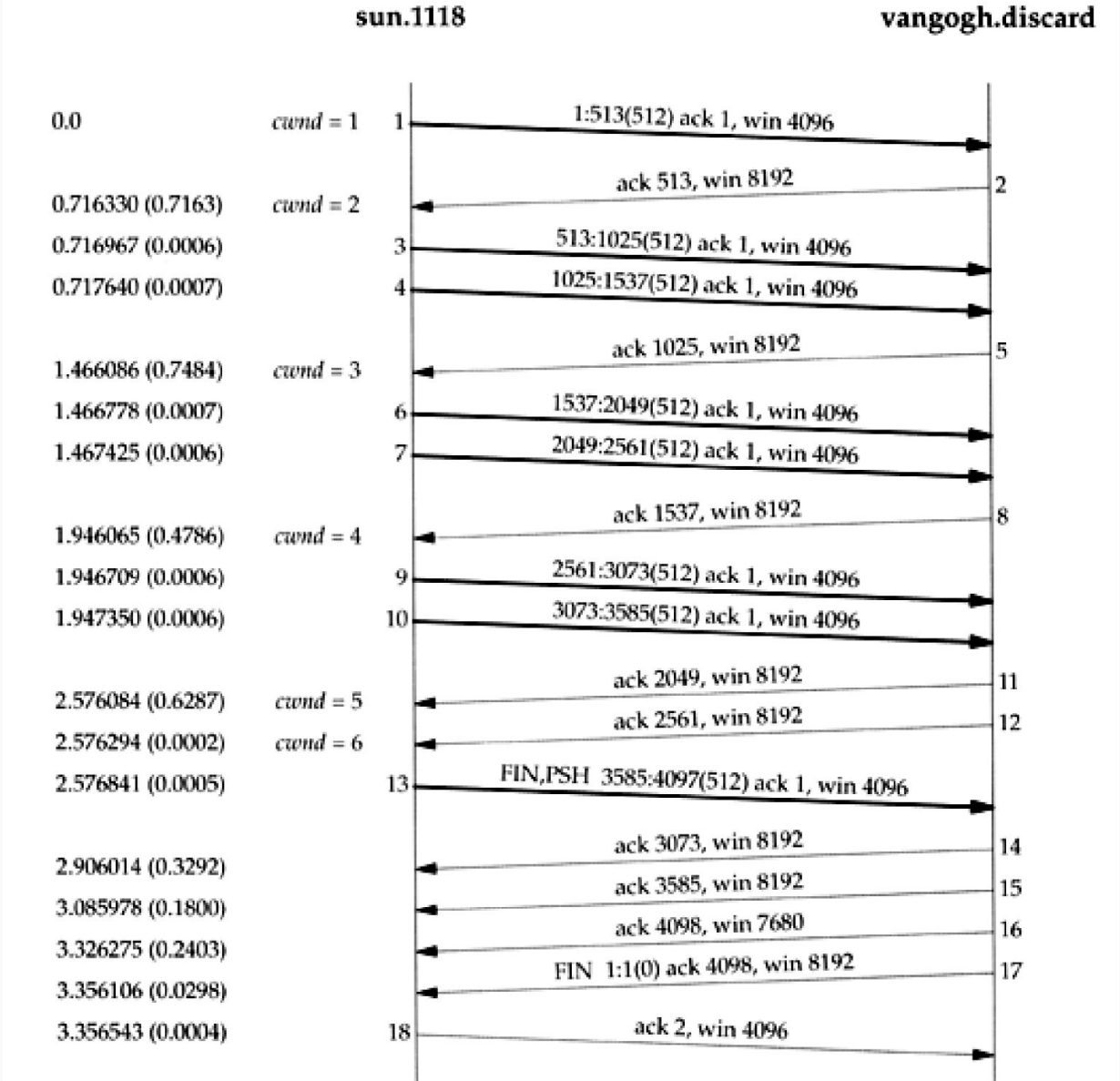
- Crecimiento exponencial

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



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

¿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

Pérdida de paquetes

- **¿Qué pasaría si se pierde un paquete?**

- Supongamos que se pierde **UN SOLO** en una ráfaga.

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**

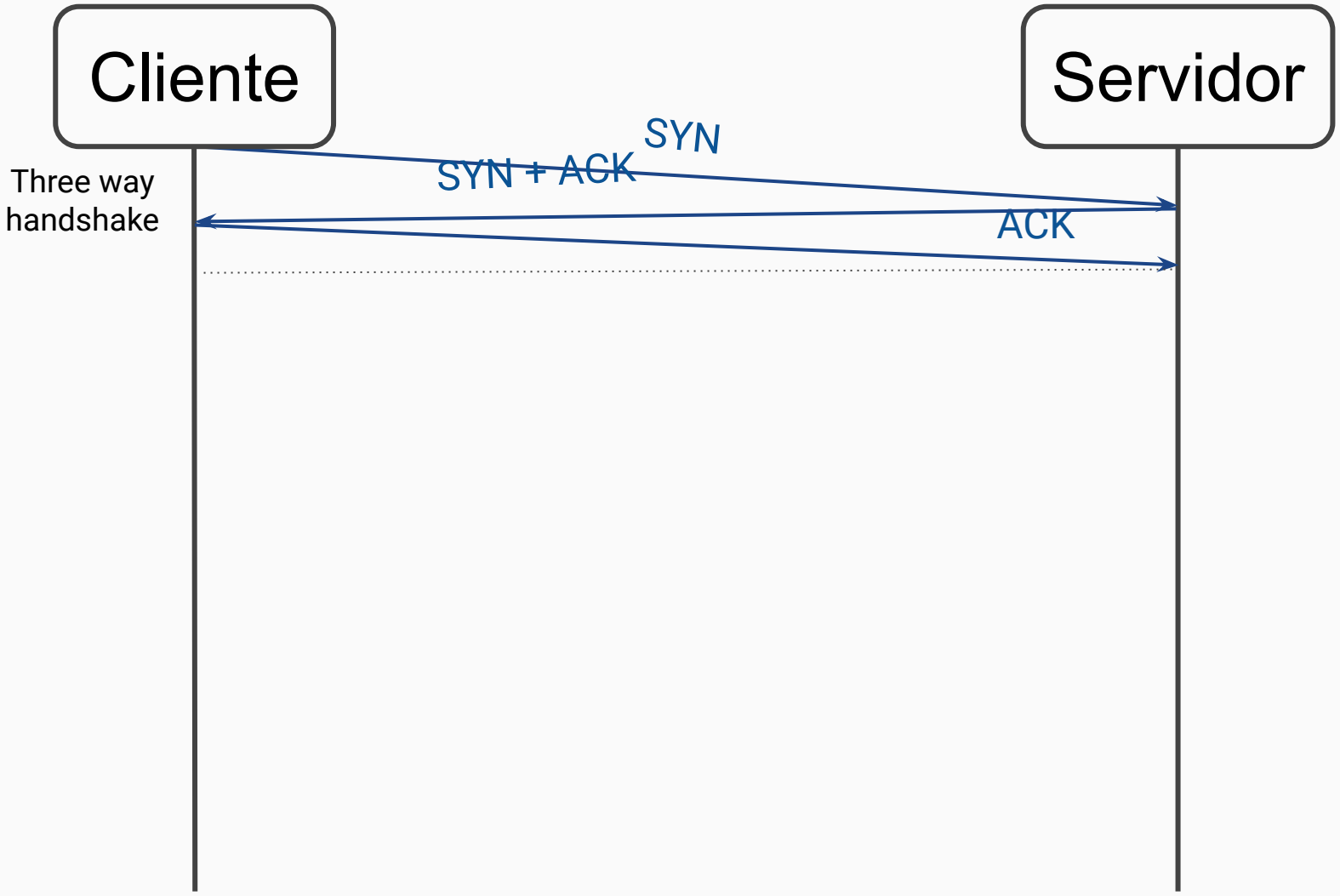
TCP: Tahoe

TCP : control de congestión

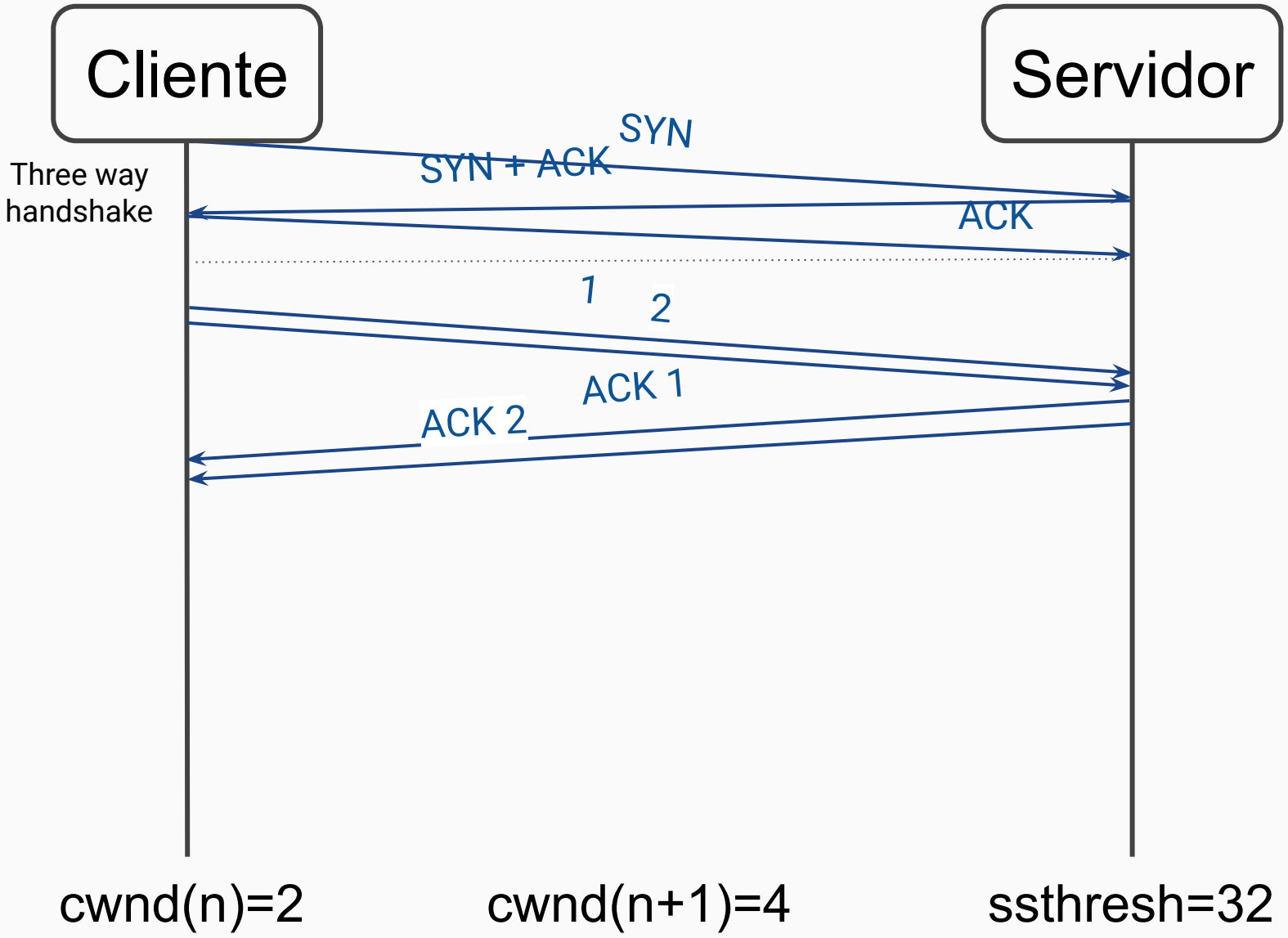
Algoritmo TCP Tahoe

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

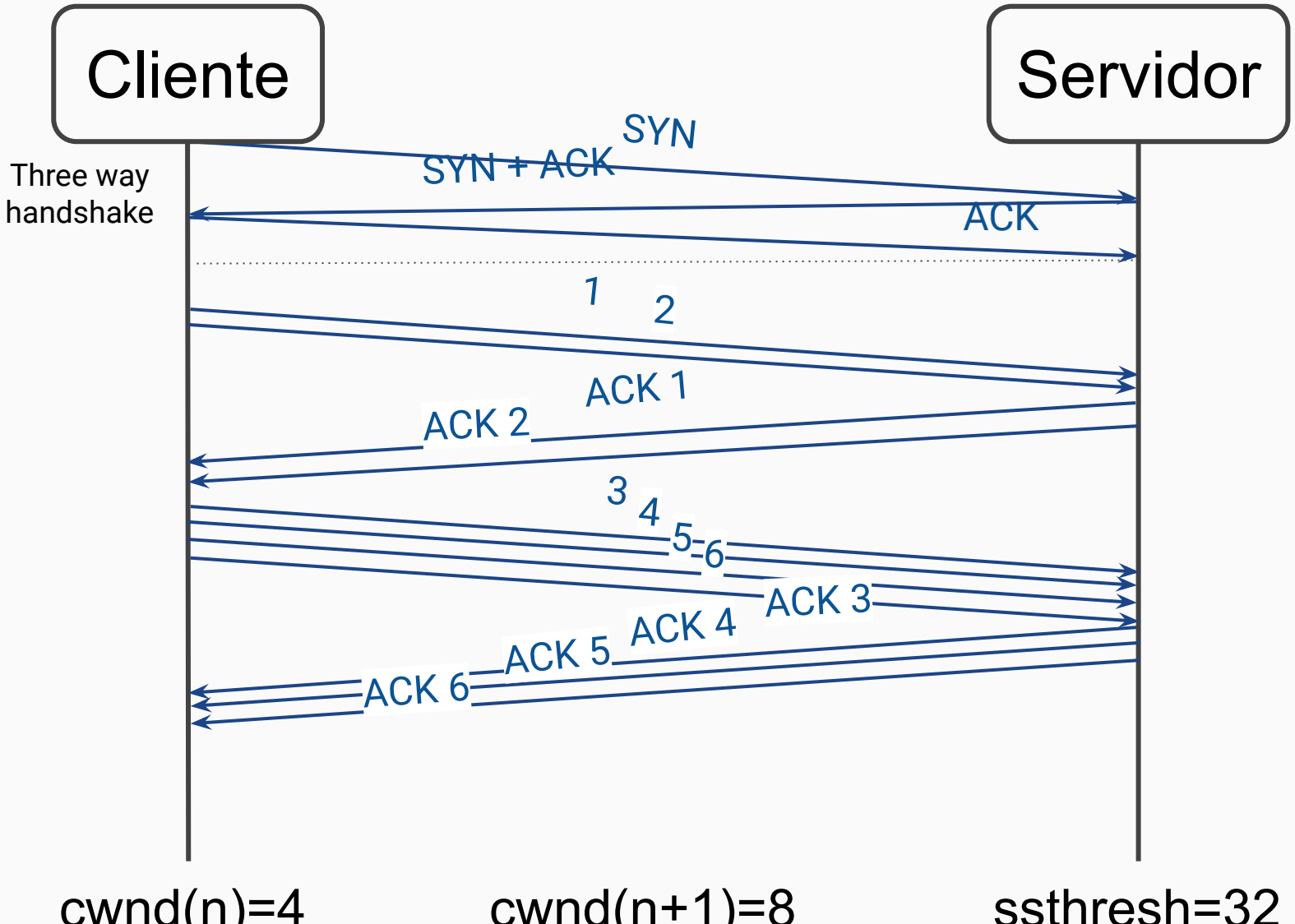
Three way handshake



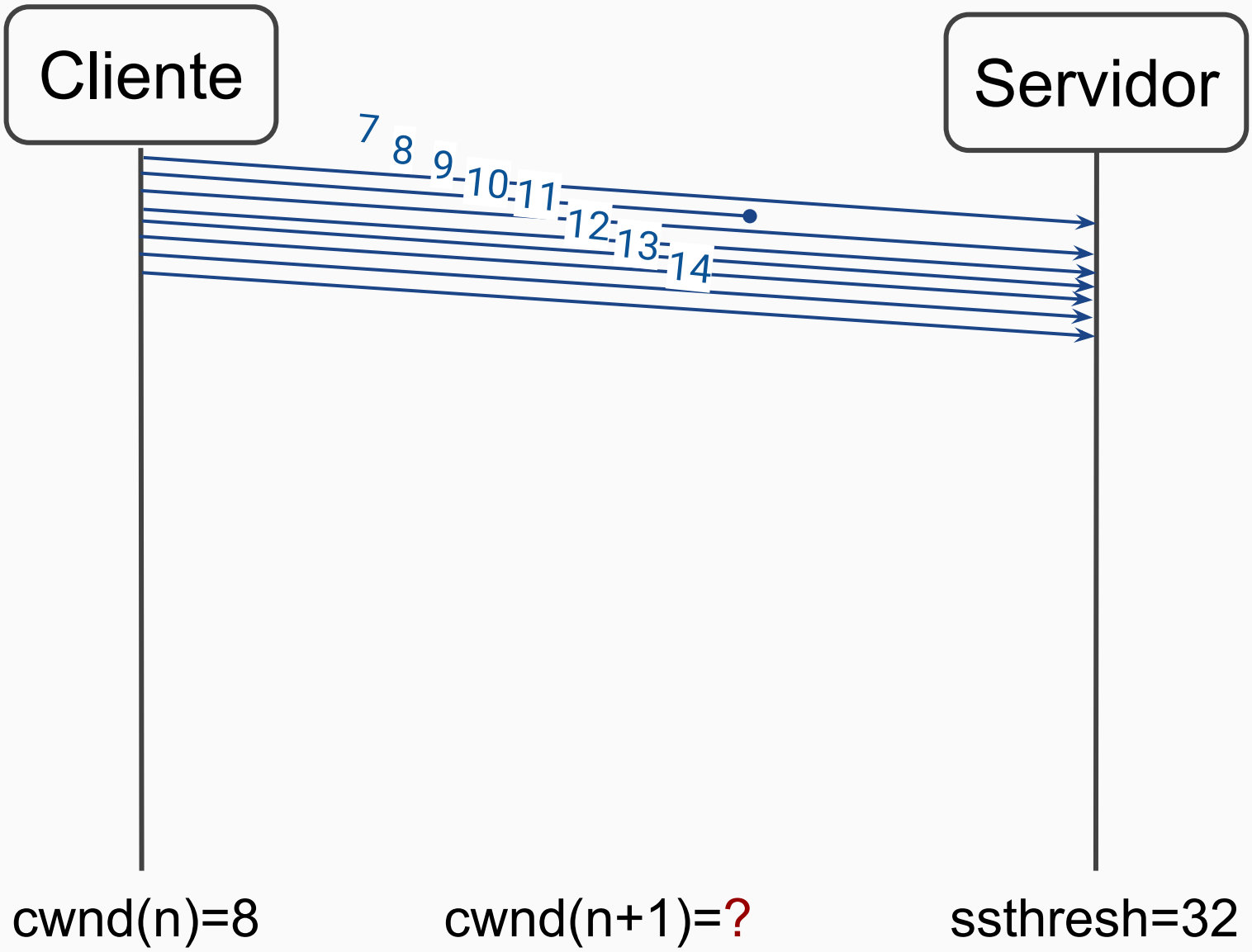
Slow start


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

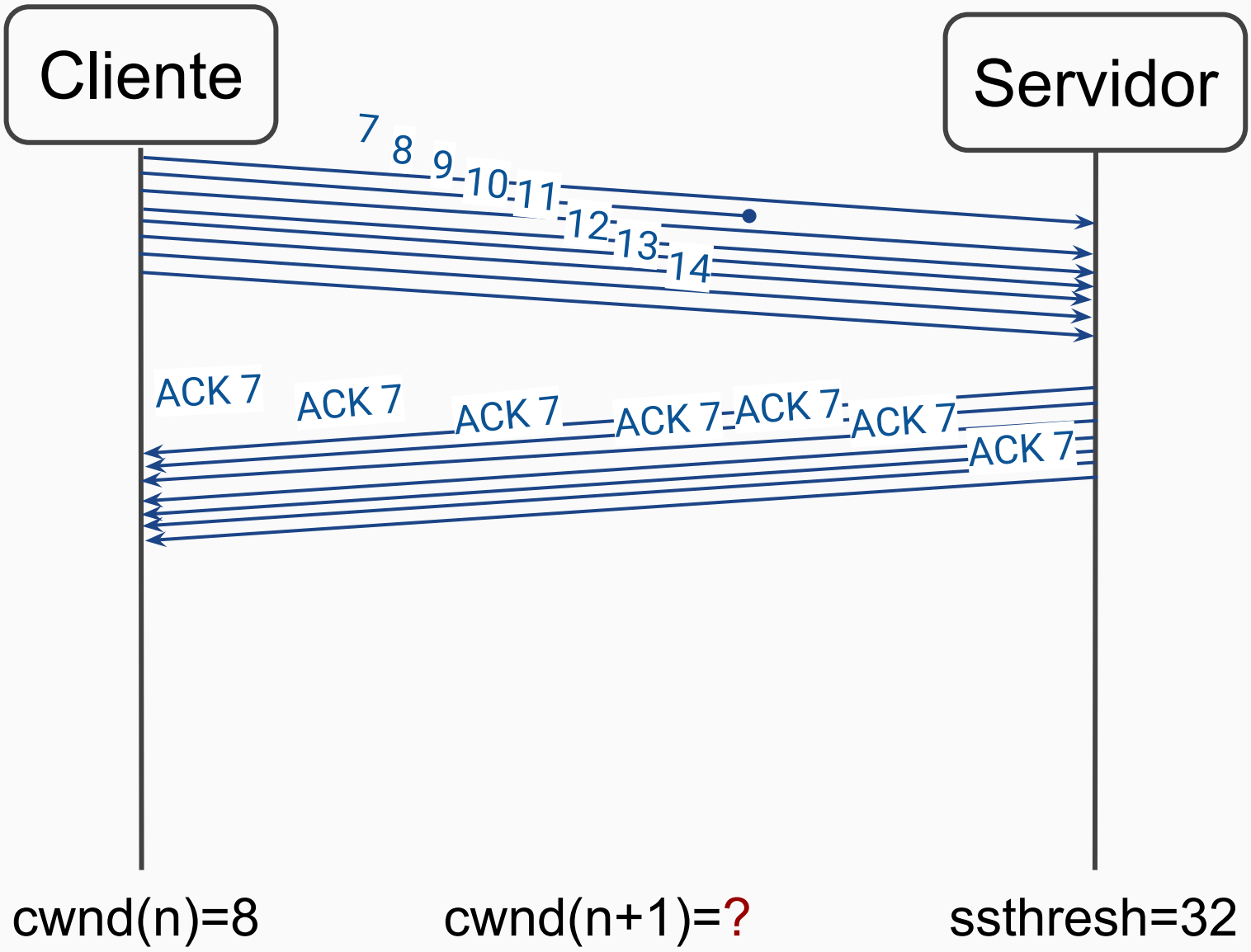
Slow start


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

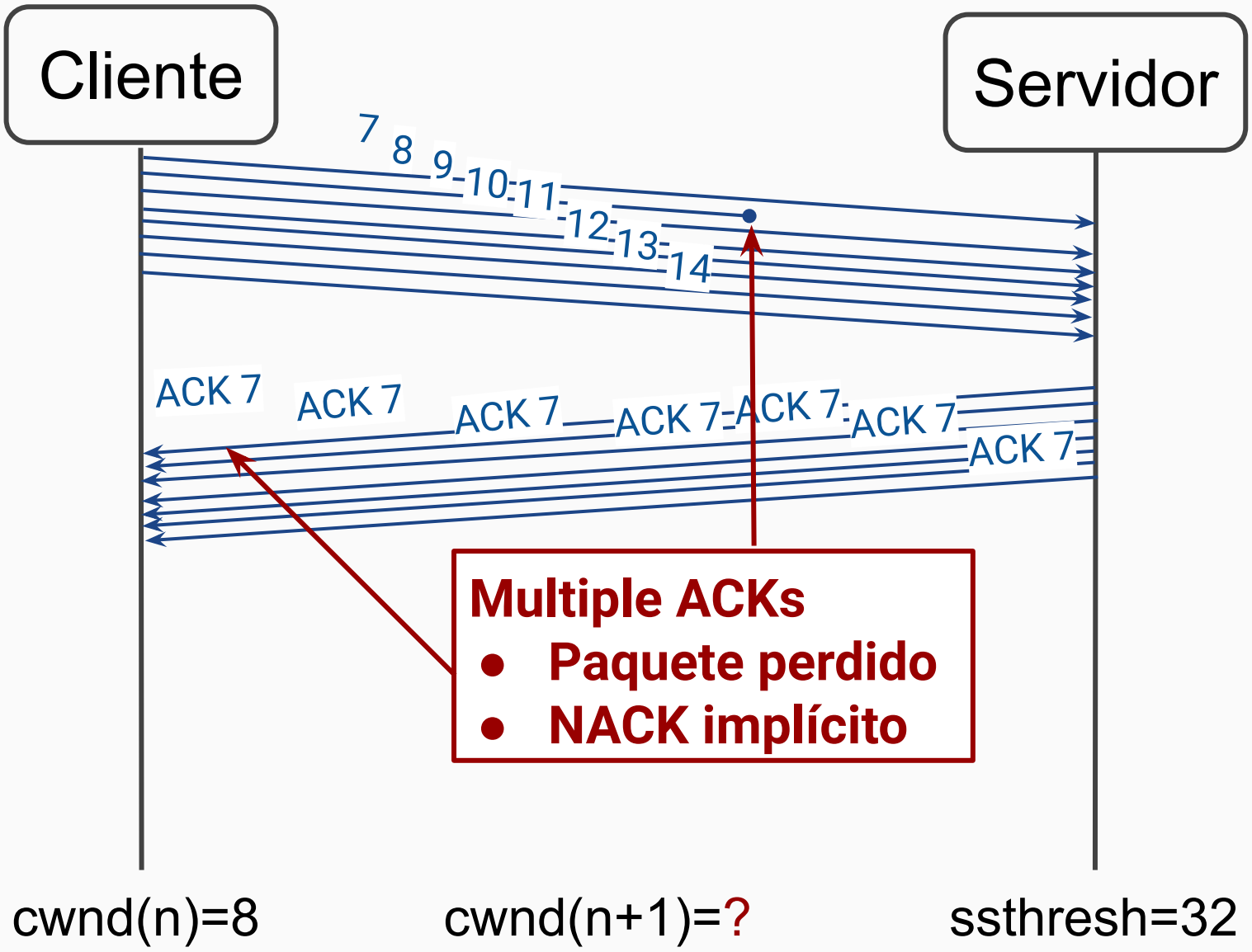
Slow start


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

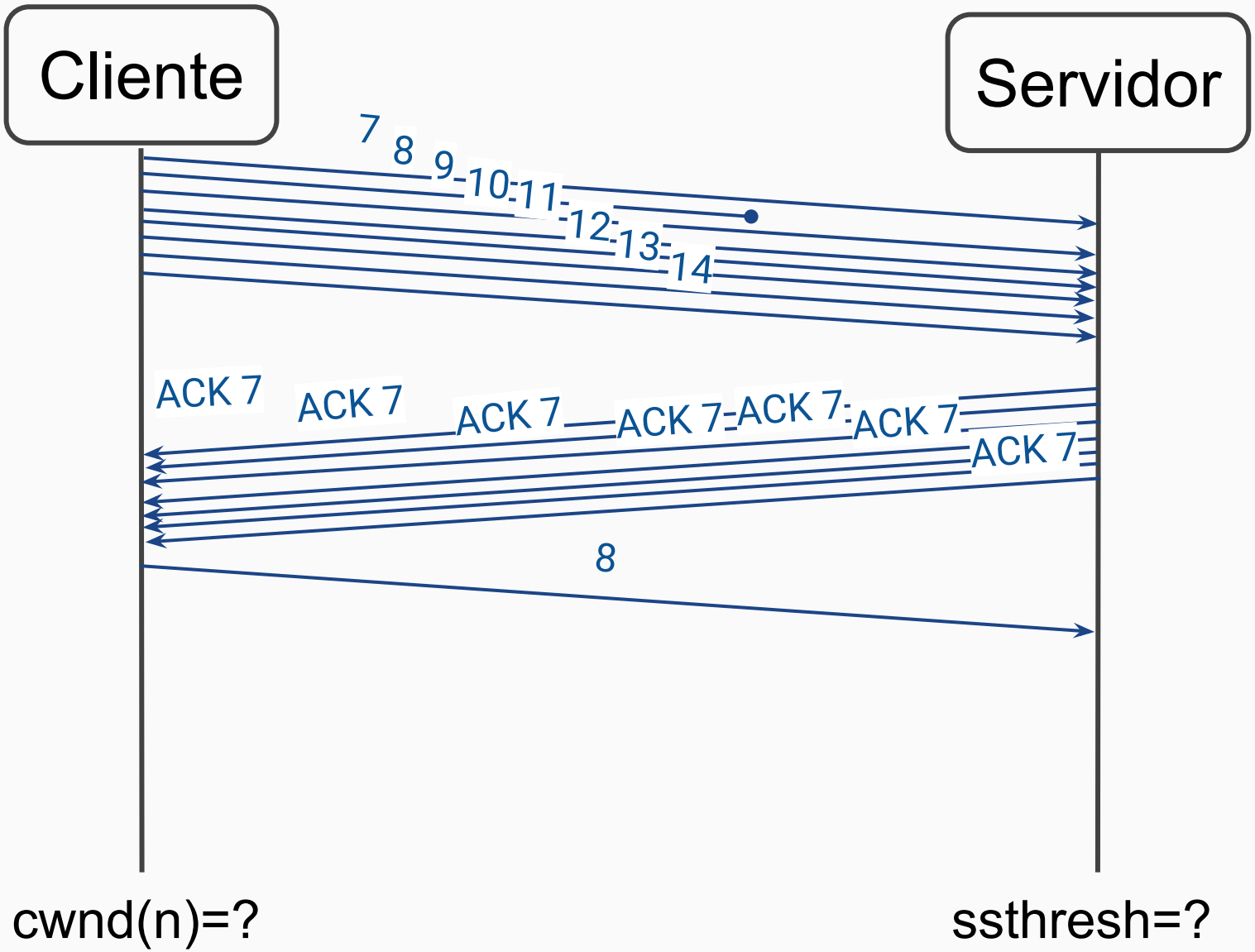
Slow start


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

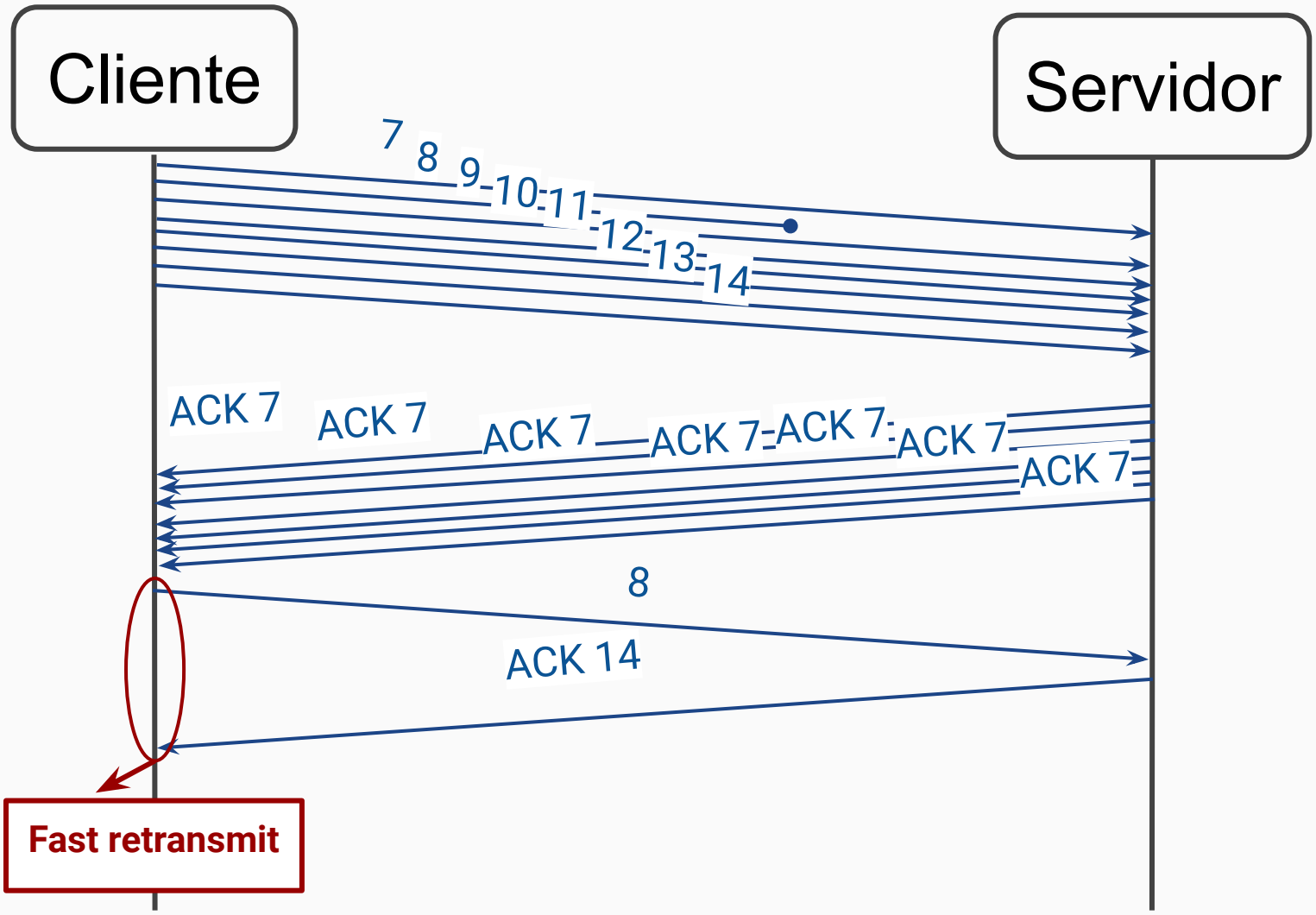
Slow start


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

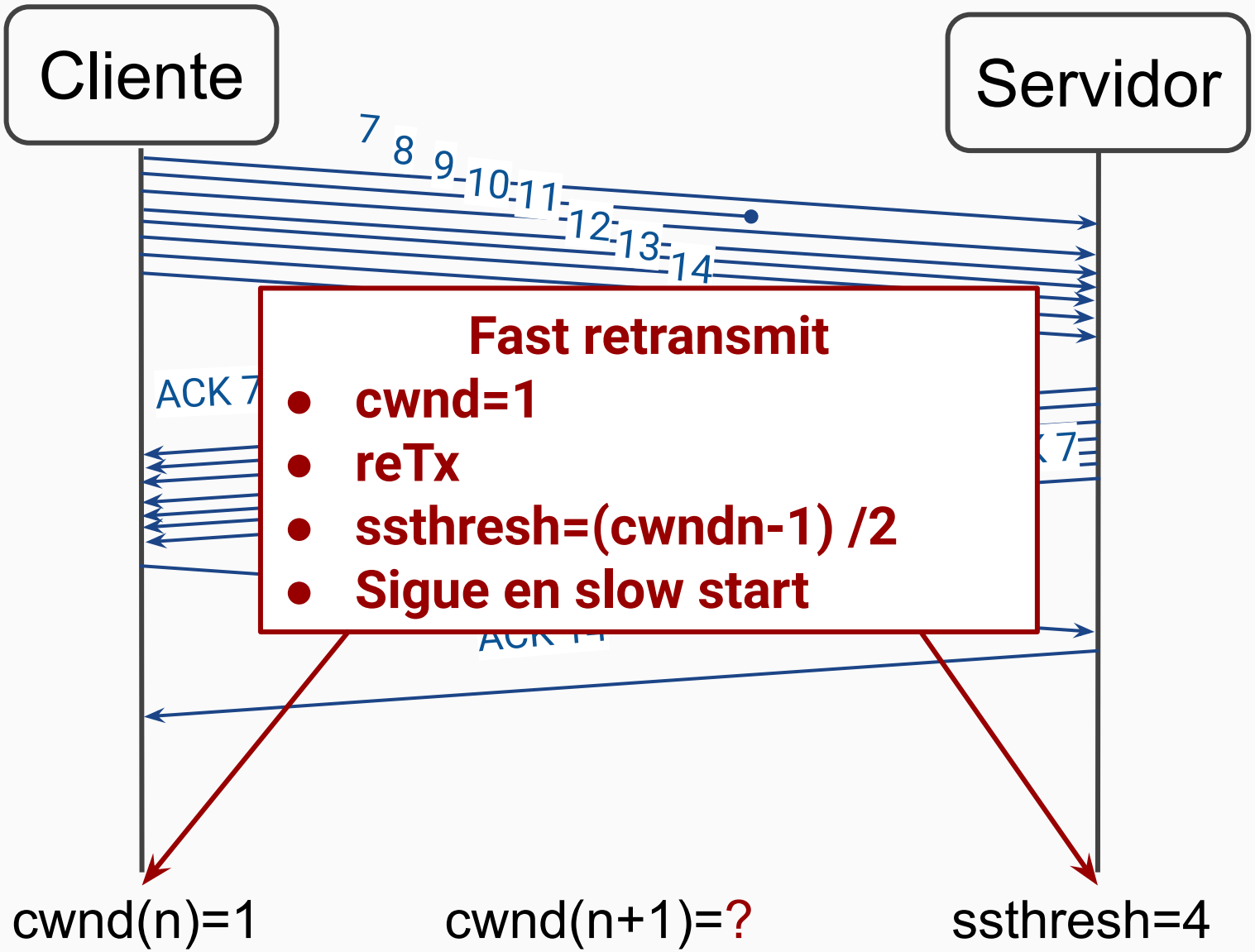

Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
8
cwnd(n)=?
ssthresh=?

Fast Retransmit

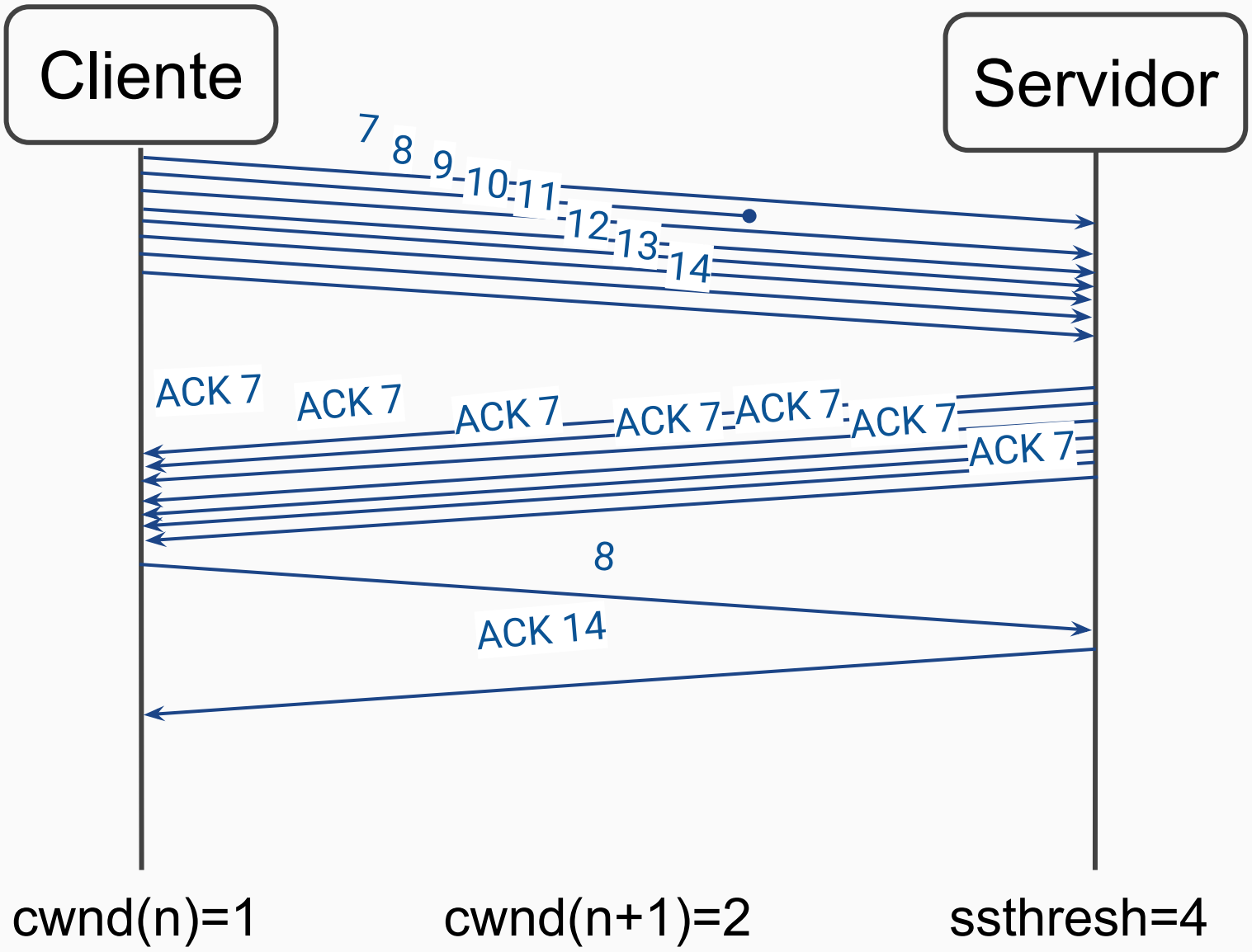


cwnd(n)=? cwnd(n+1)=? ssthresh=?

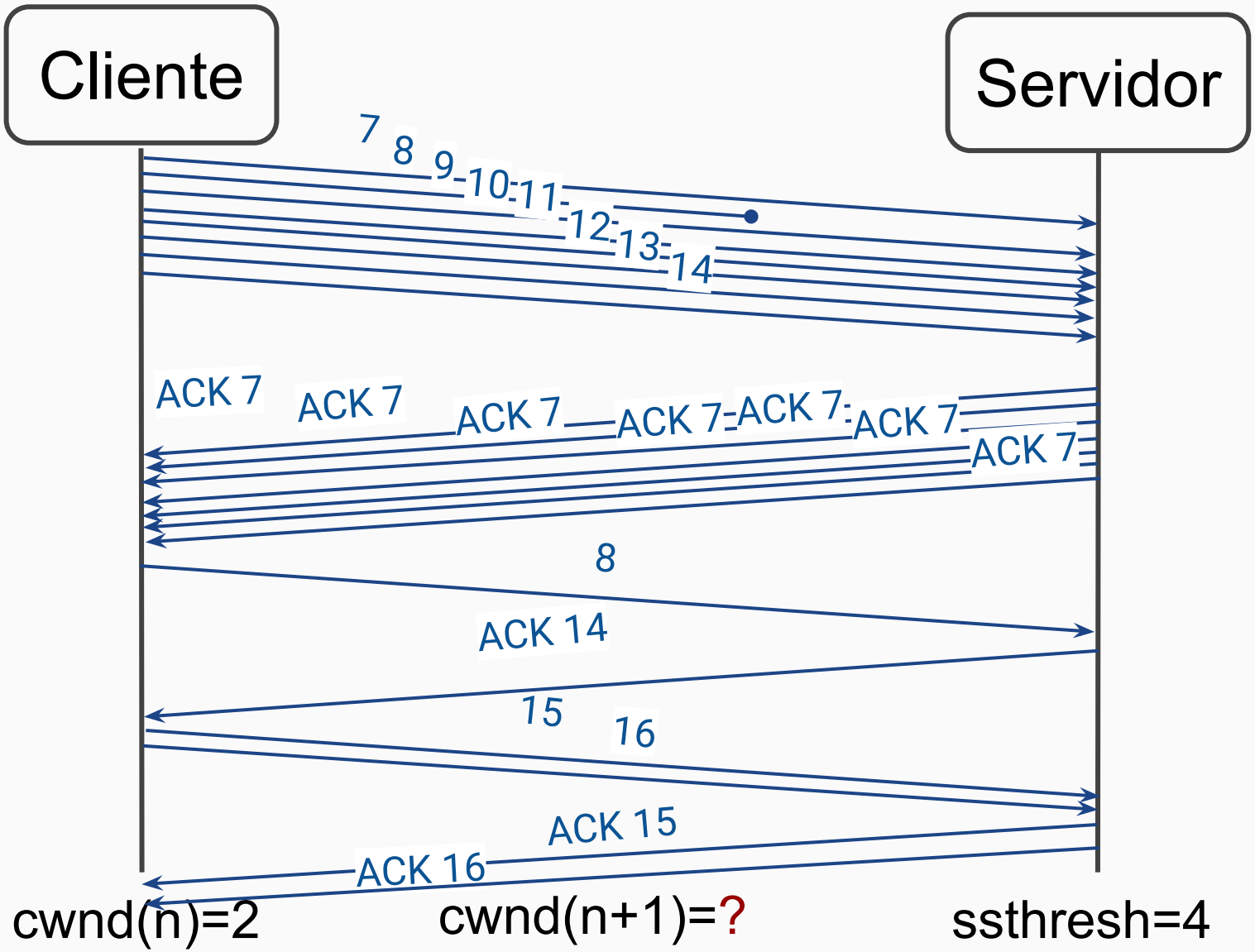
Fast Retransmit



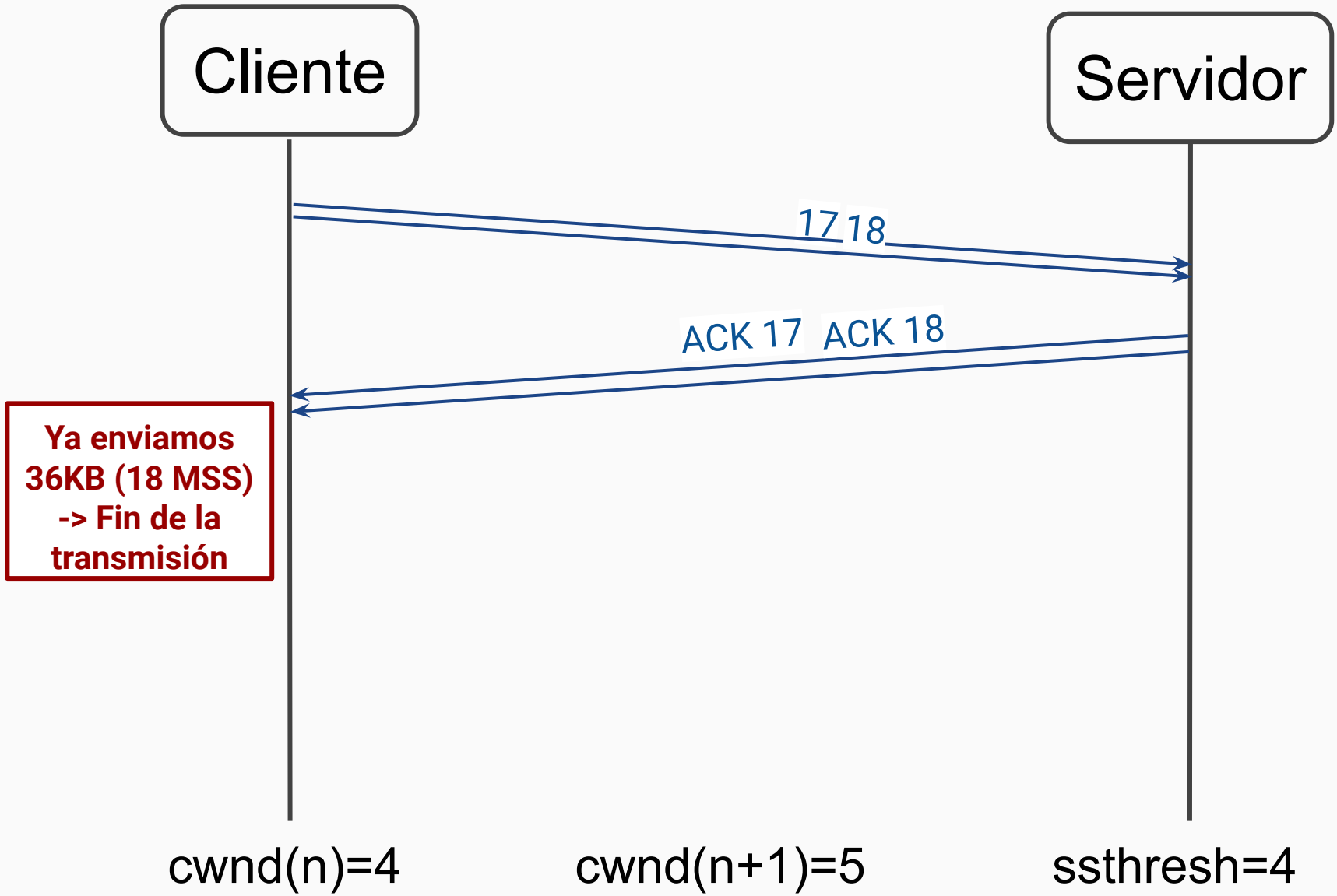
Slow start


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

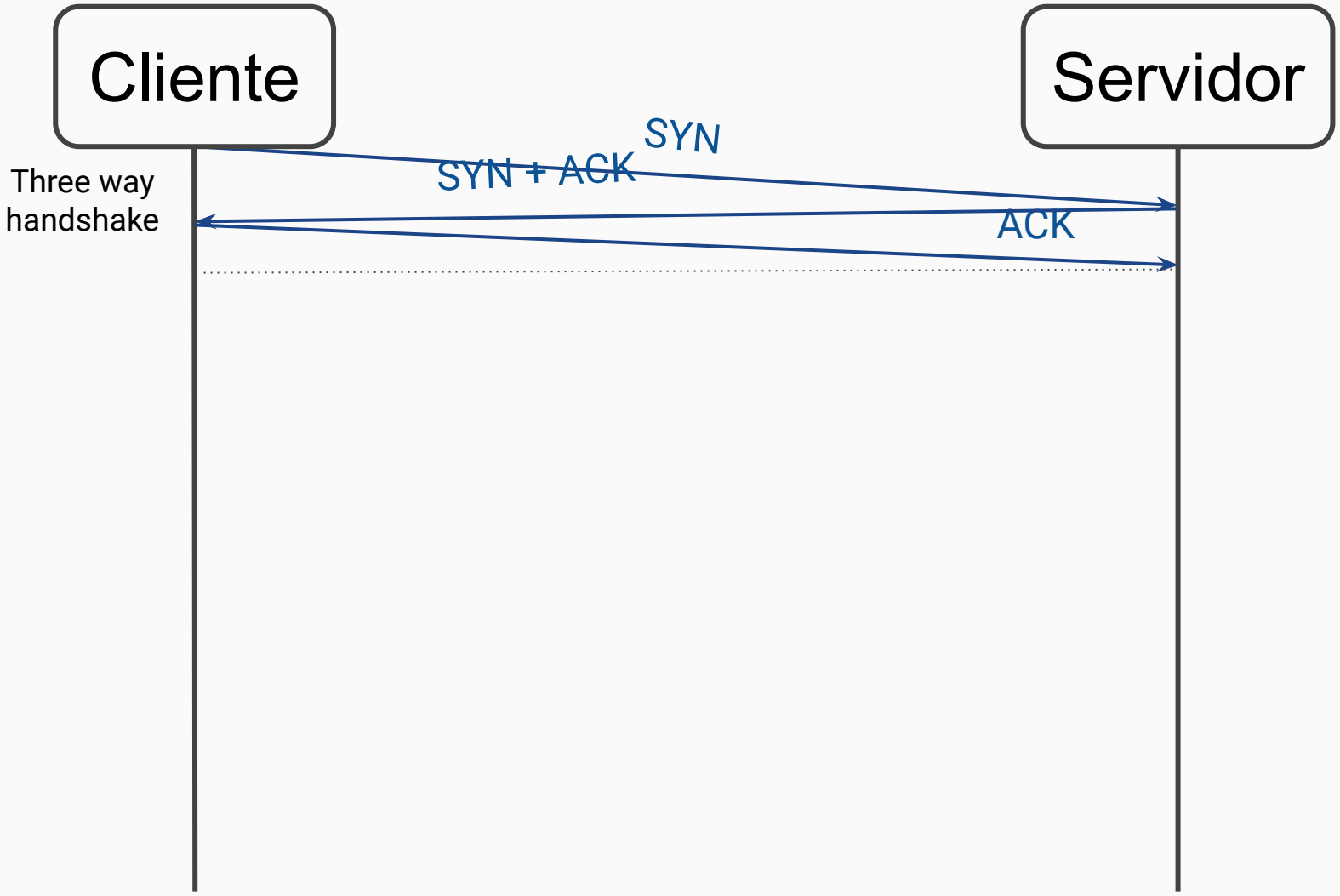
- **Ahora, veamos el caso de Reno**
- Supongamos que se pierde **UN SOLO** segmento de la ráfaga en cuestión.
- Y además, que estamos usando el algoritmo de control de congestión de **Reno**

TCP : control de congestión

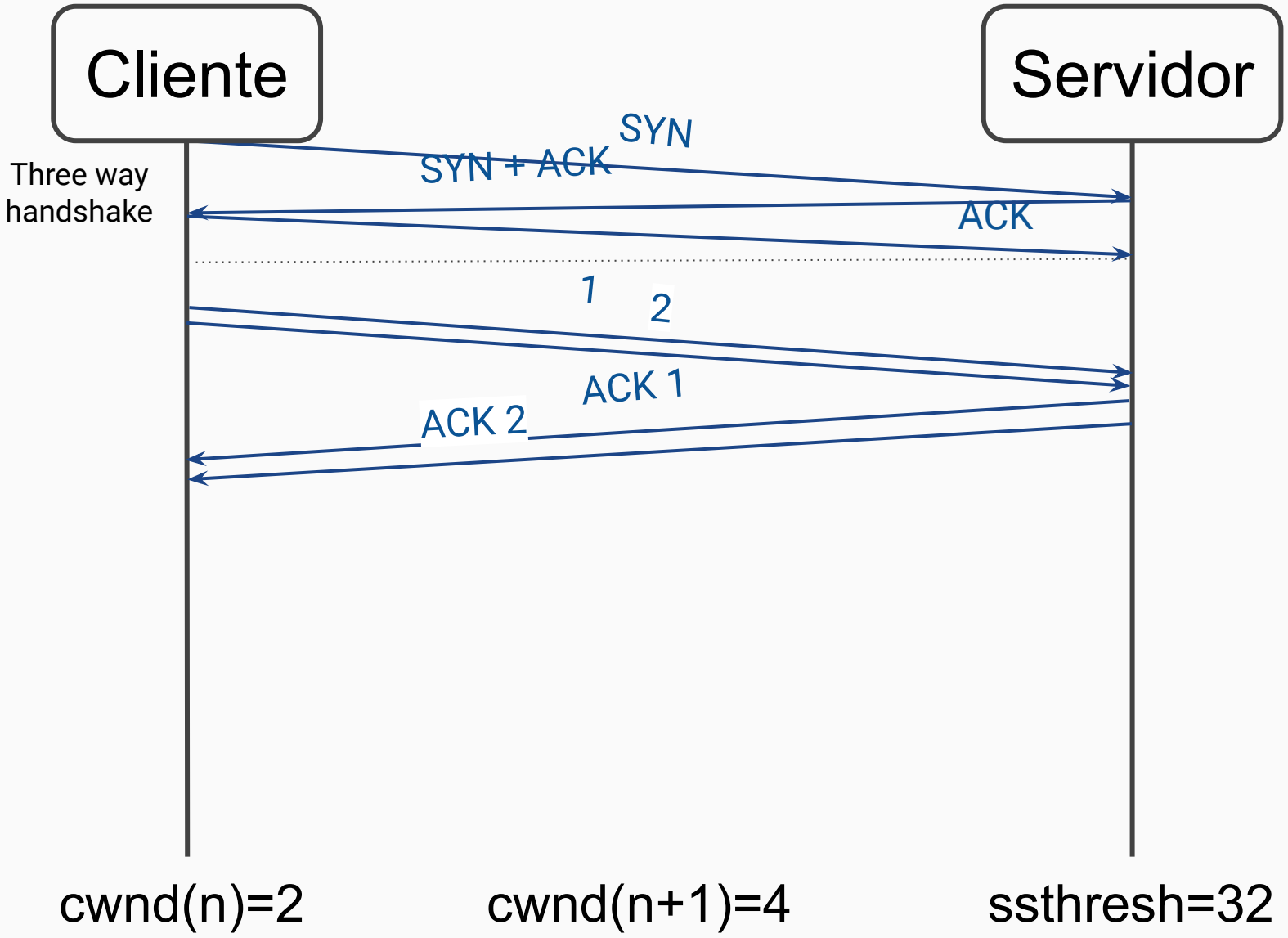
Algoritmo TCP Reno

- 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**

Three way handshake

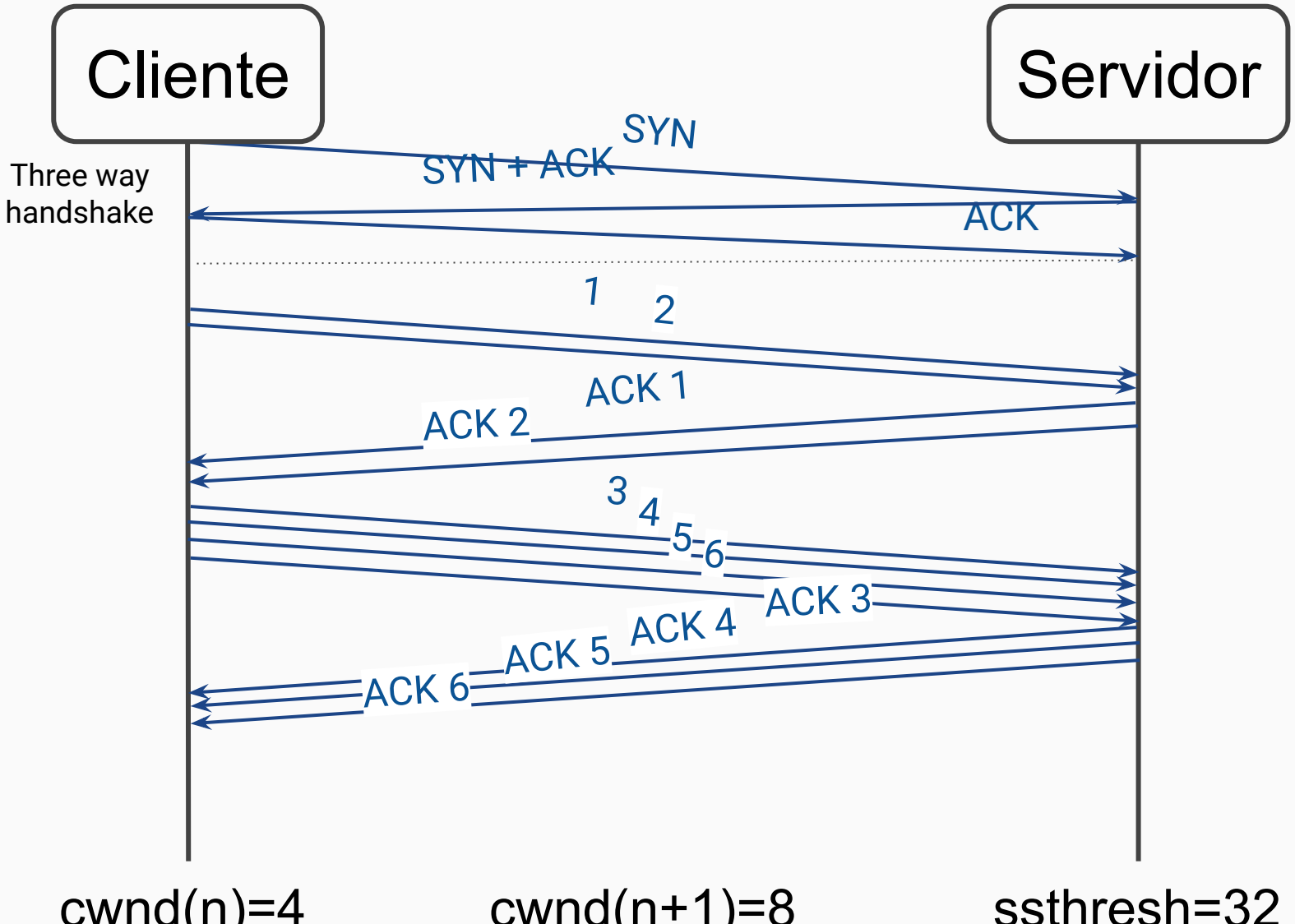


Slow start

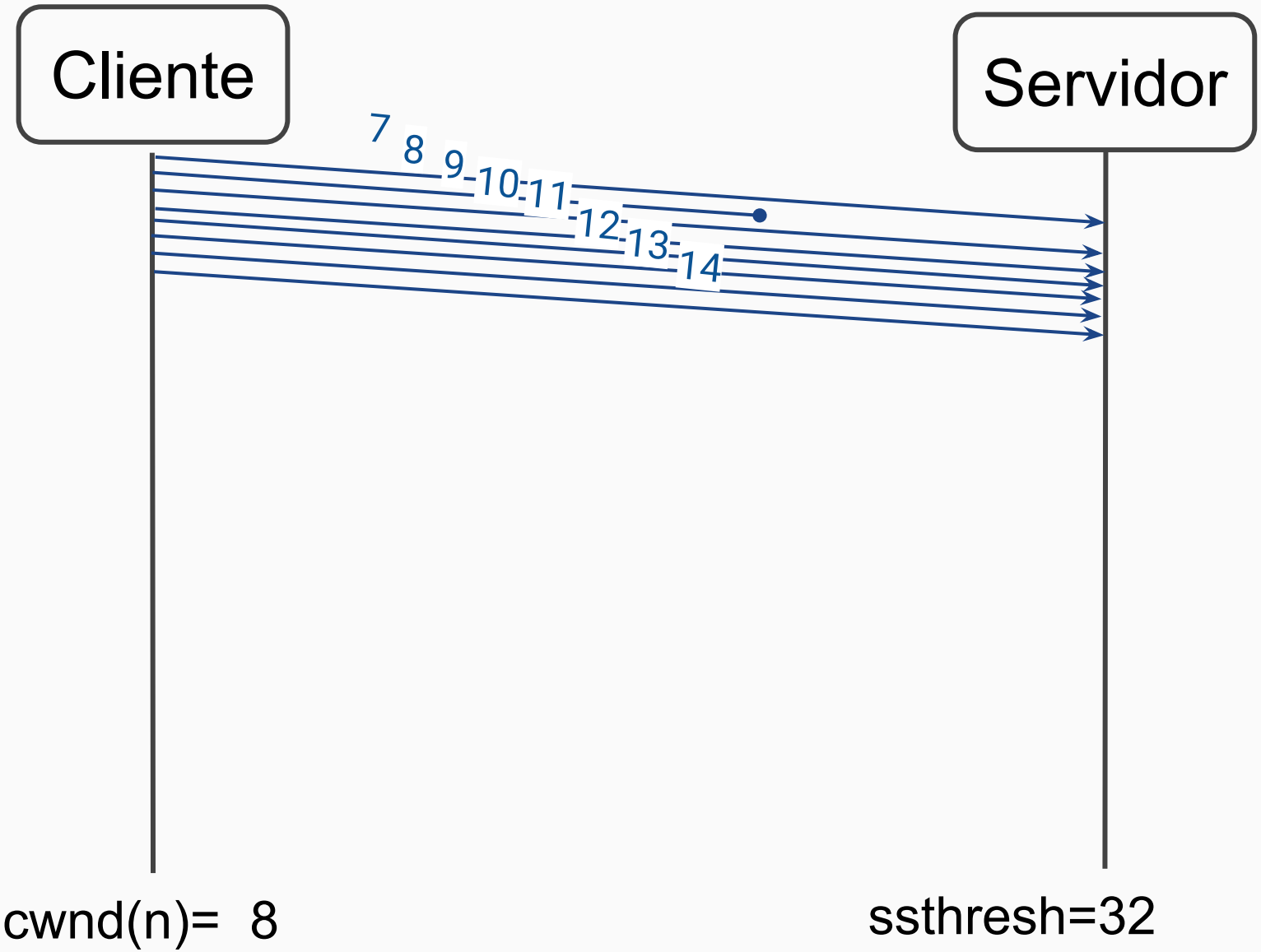

Cliente
Servidor
Three way handshake
SYN
SYN + ACK
ACK
1
2
ACK 1
ACK 2
cwnd(n)=2
cwnd(n+1)=4
ssthresh=32

TCP: Reno

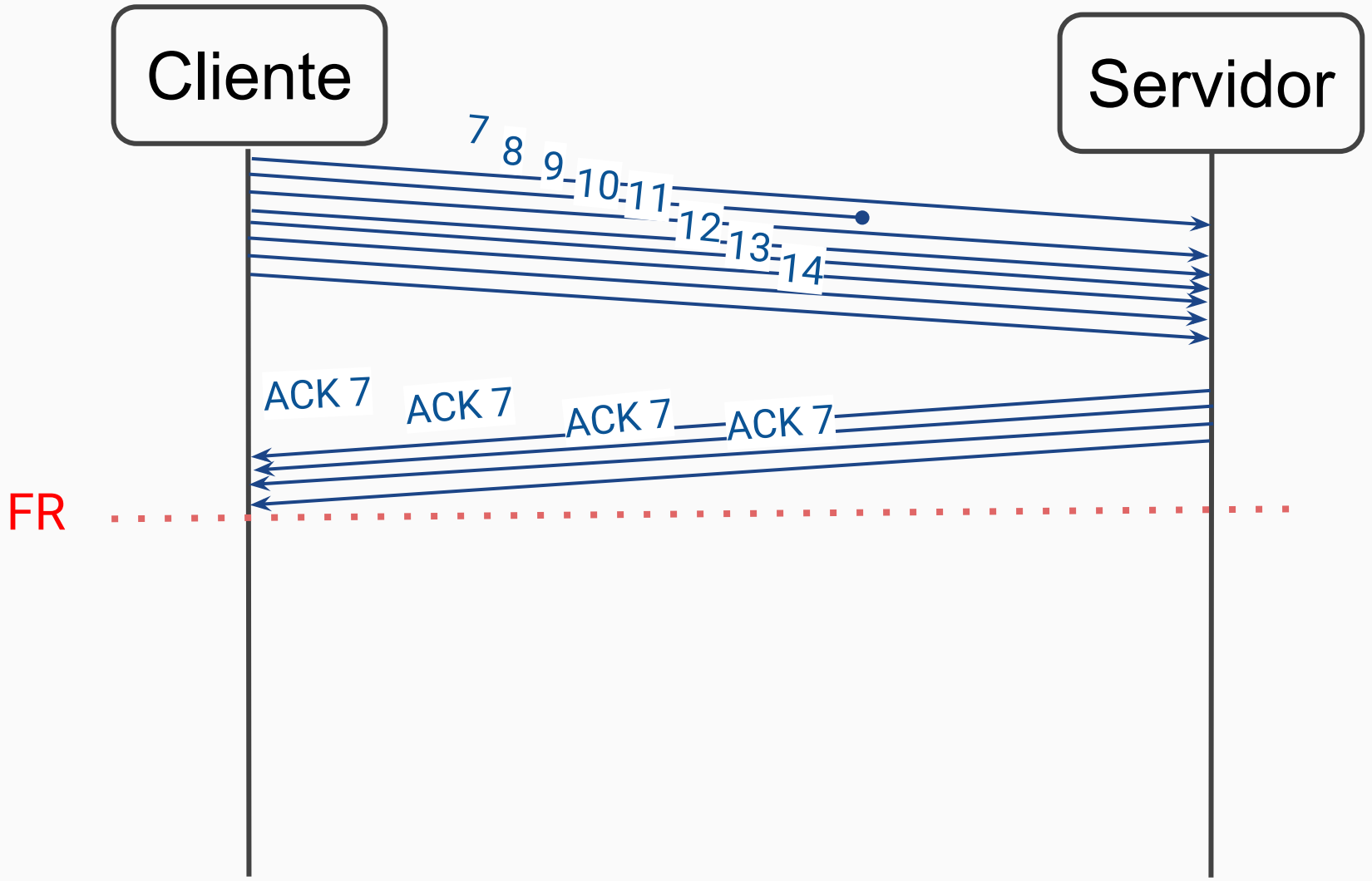
Slow start


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



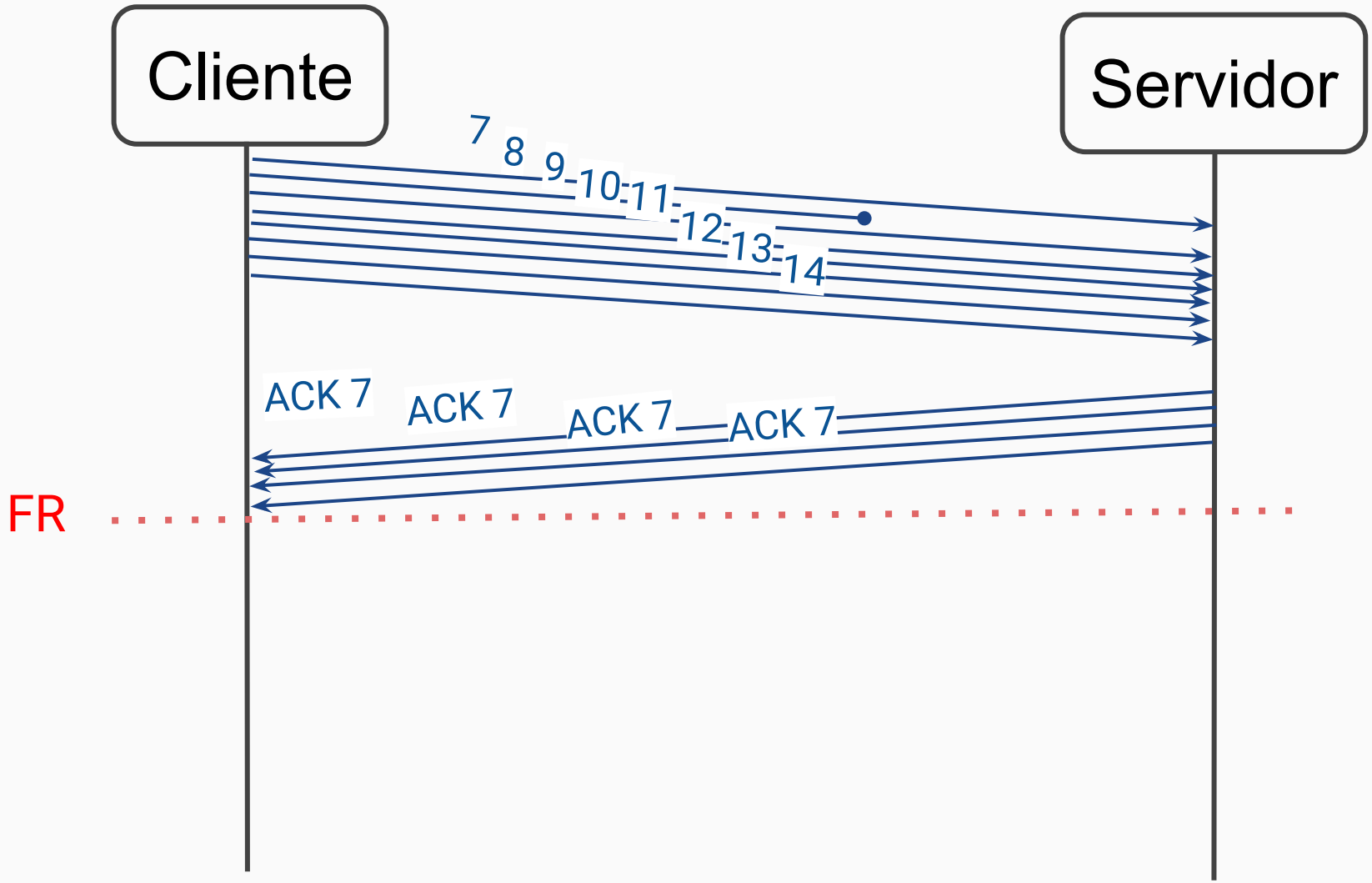
Slow start



cwnd(n)= ssthresh

ssthresh=cwnd/2

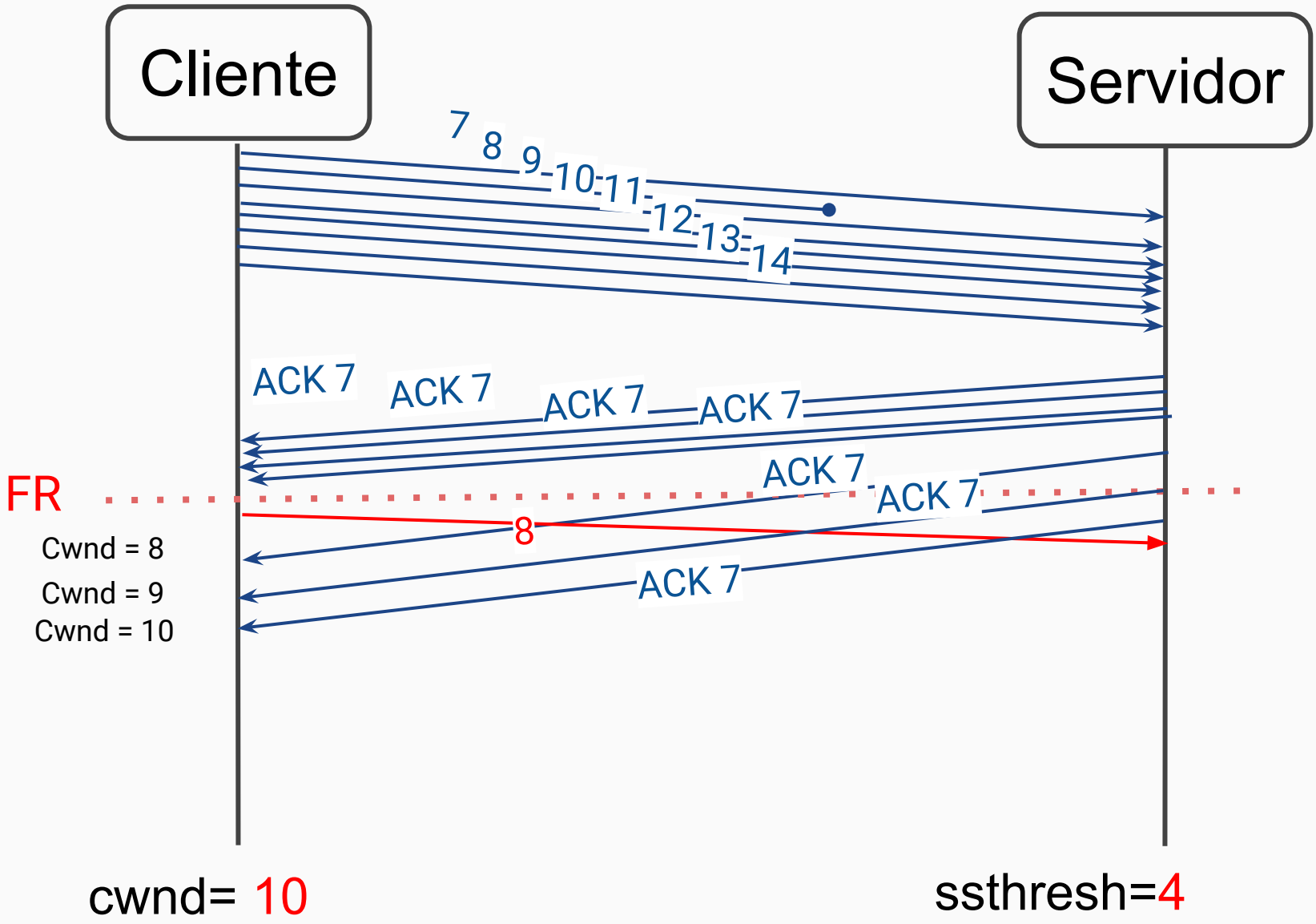
Slow start



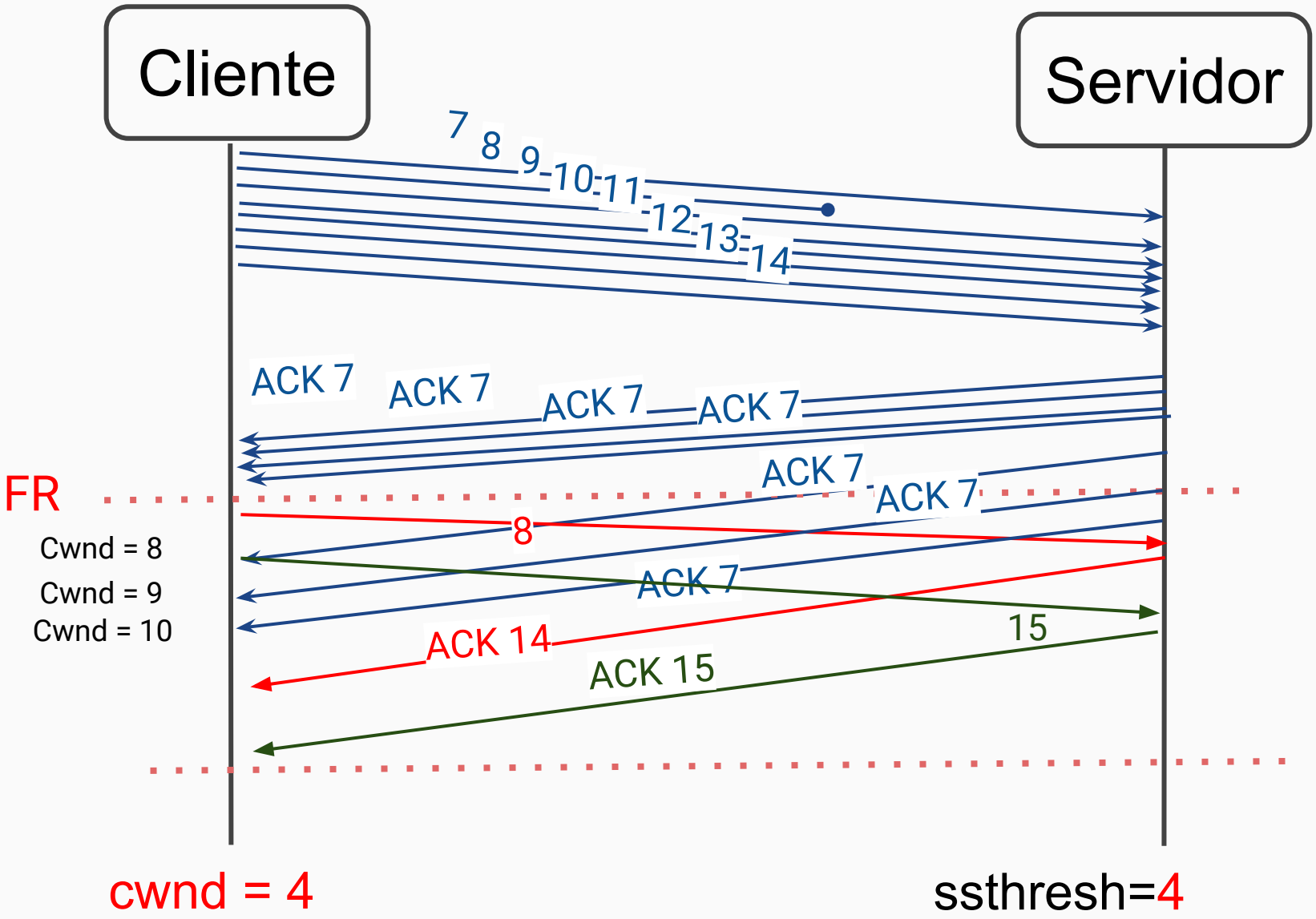
cwnd(n)= 4 + 3 (Dup ACK) = 7

ssthresh=4

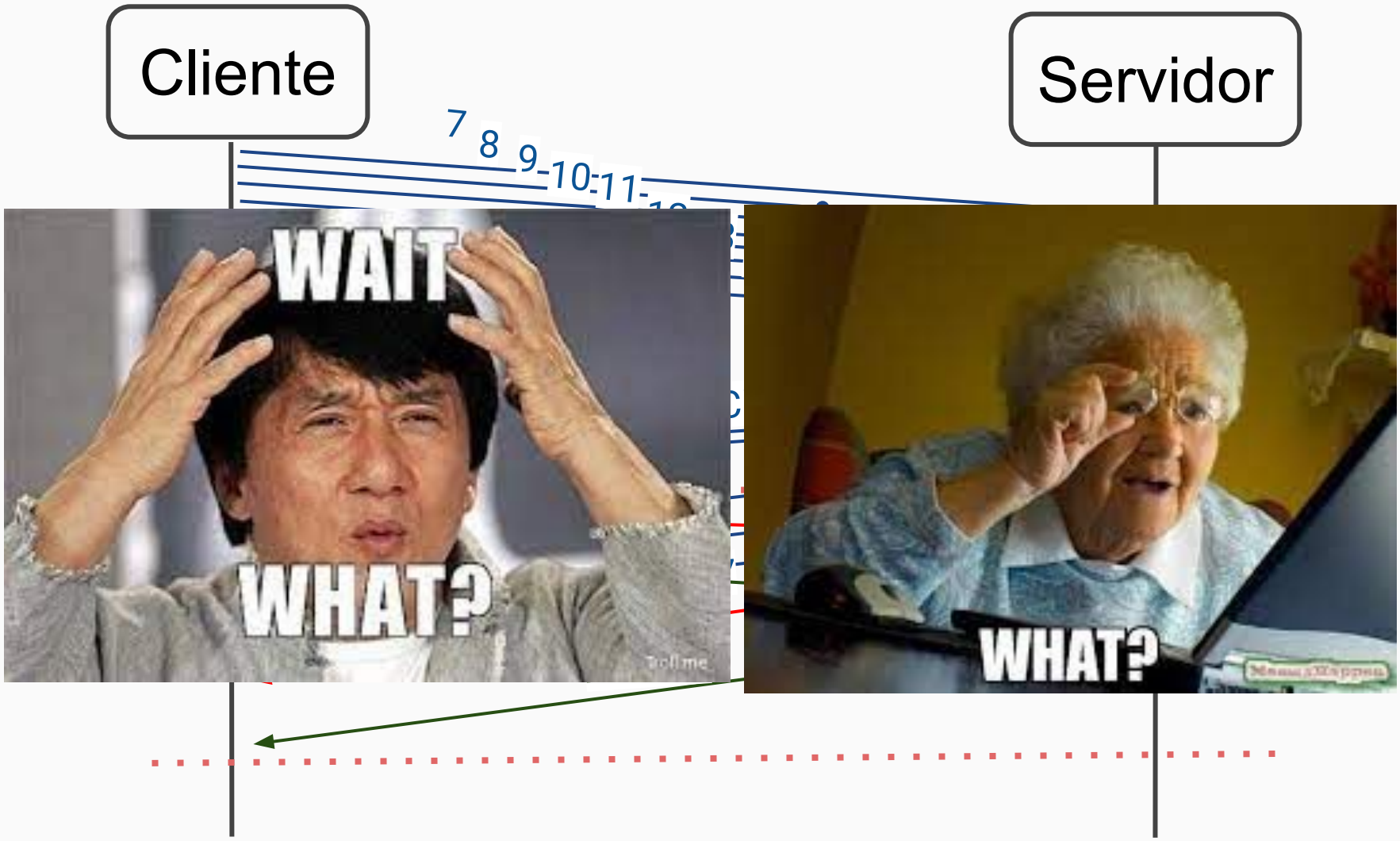
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
FR
8
Cwnd = 8
Cwnd = 9
Cwnd = 10
cwnd= 10
ssthresh=4

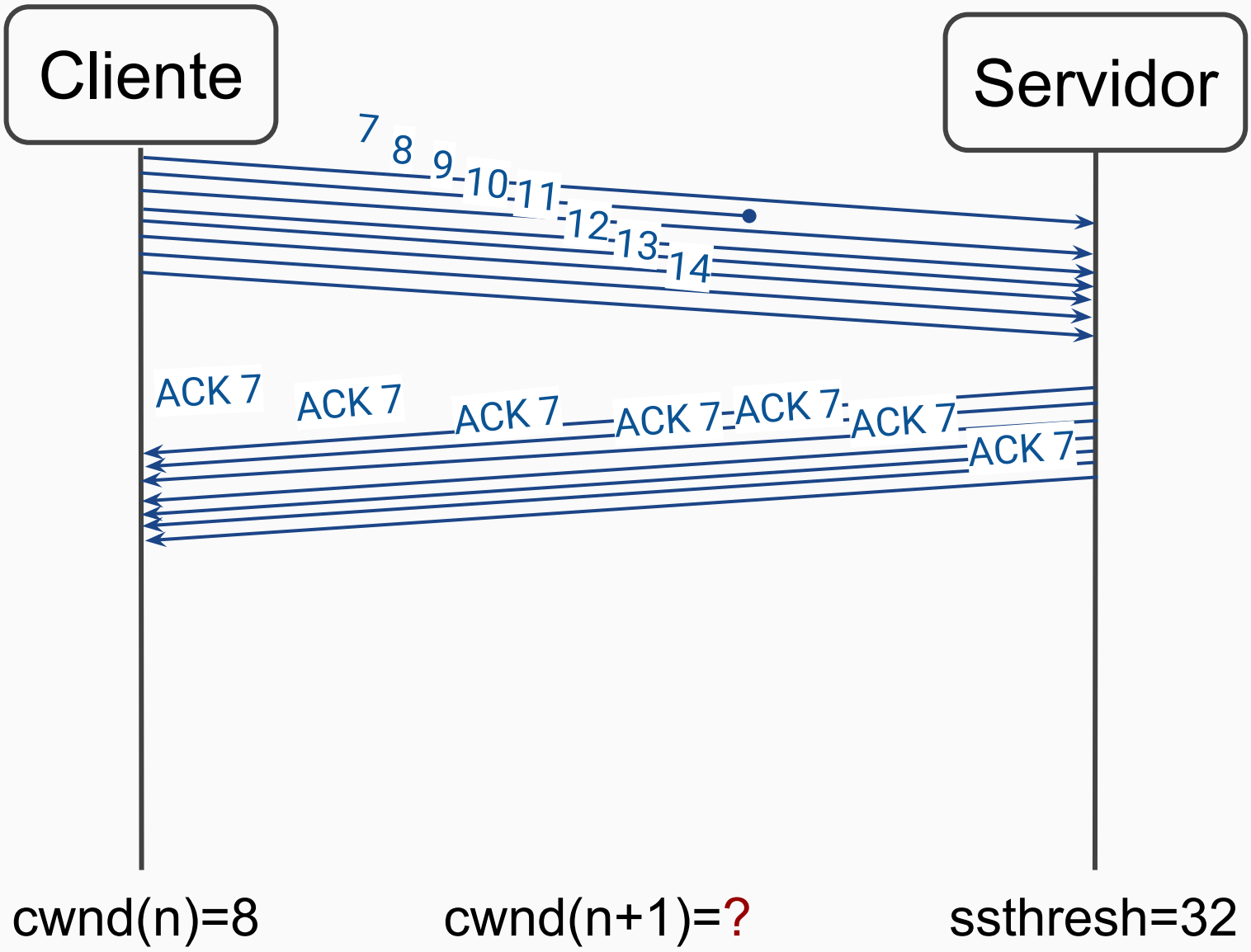
Slow start


Cliente
Servidor
7
8
9
10
11
12
13
14
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
ACK 7
FR
8
Cwnd = 8
Cwnd = 9
Cwnd = 10
ACK 7
ACK 14
15
ACK 15
cwnd = 4
ssthresh=4

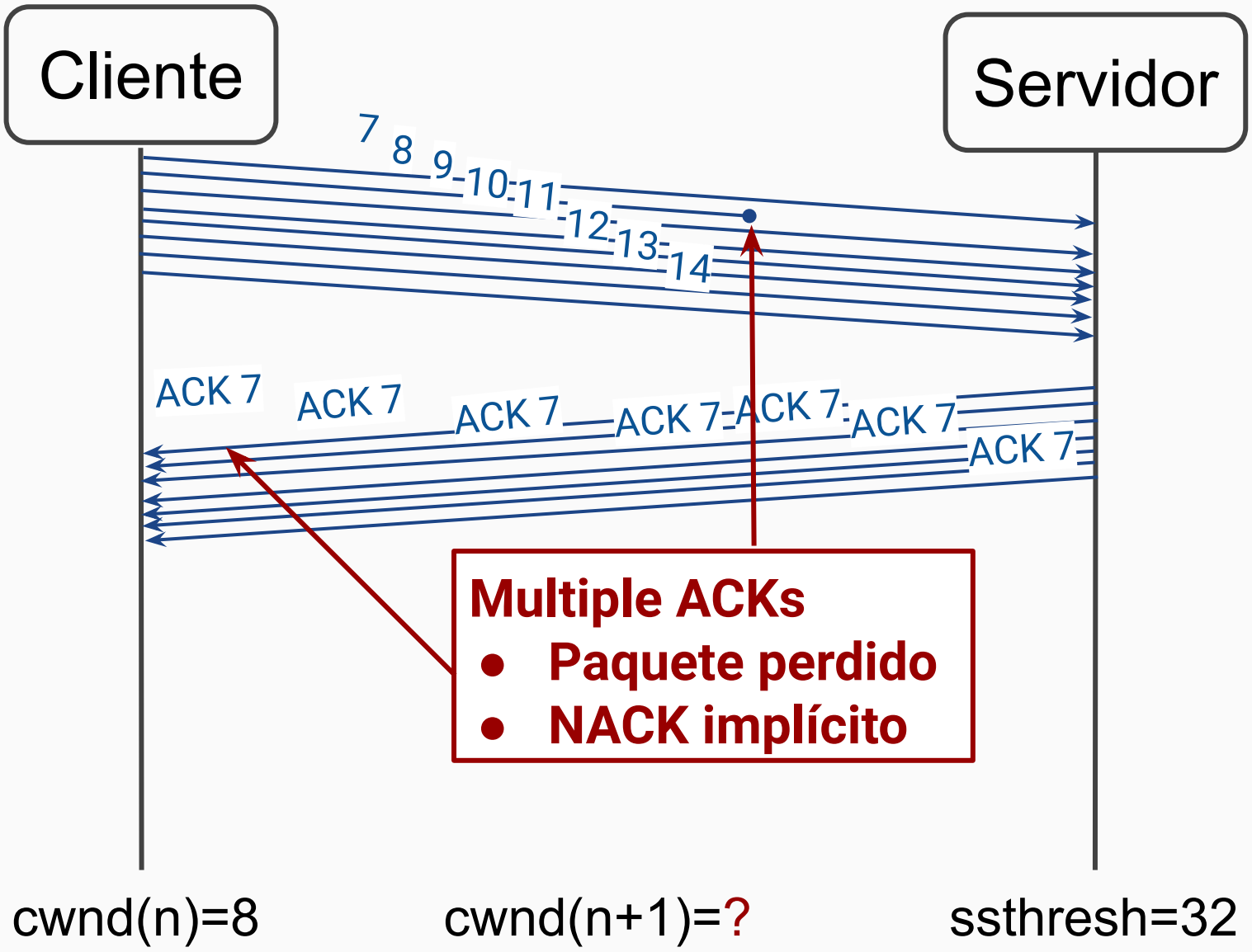
Slow start



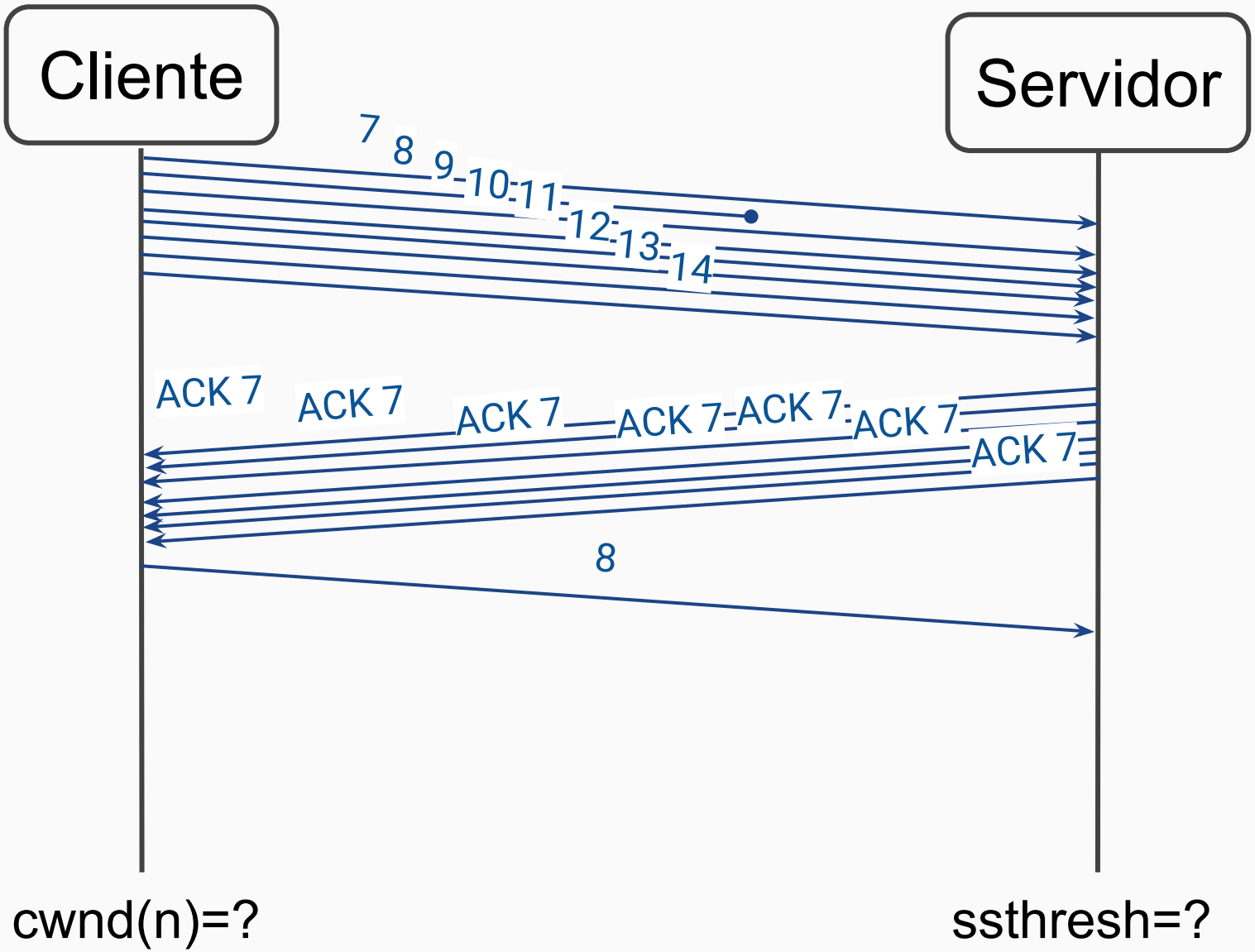
Slow start


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

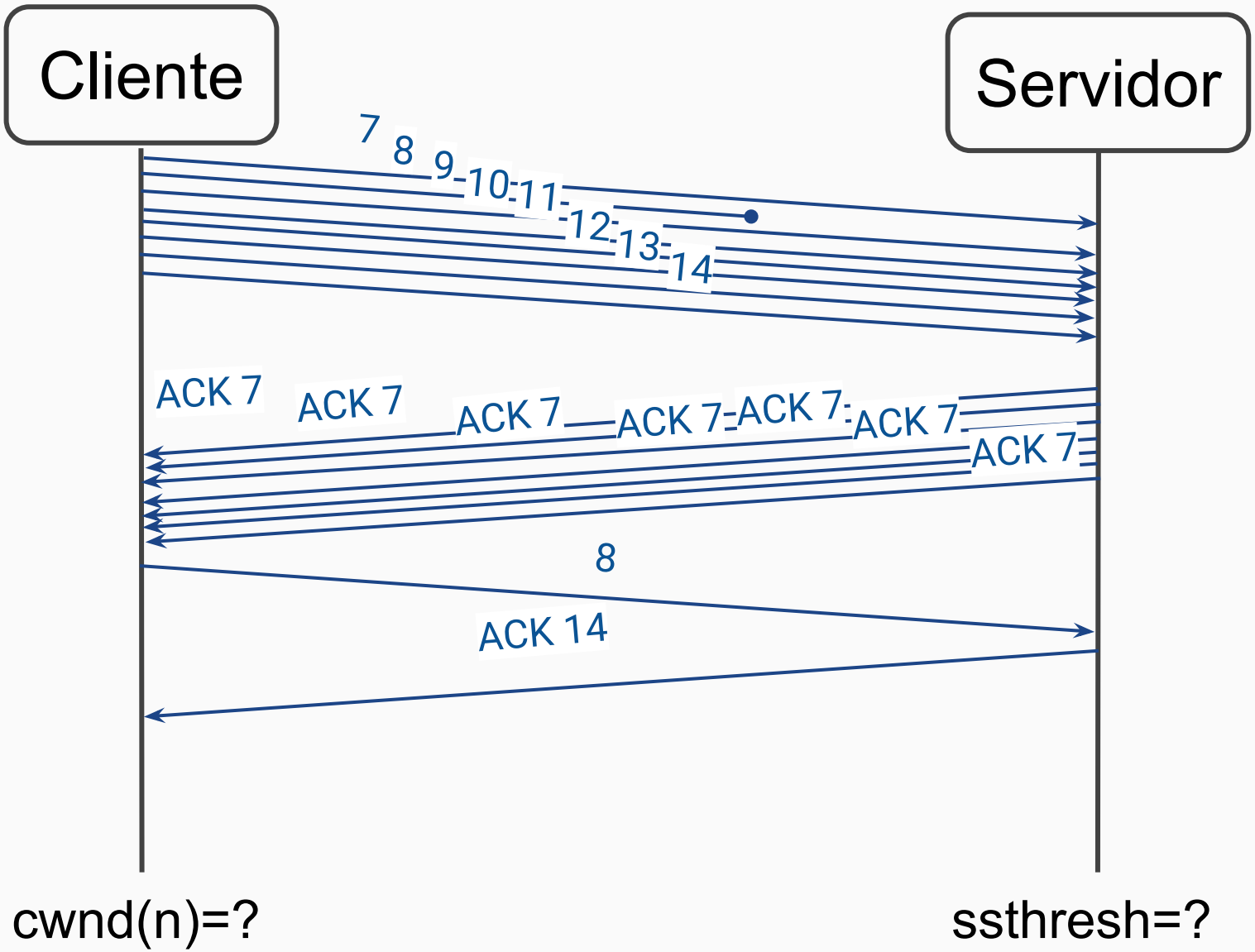
Slow start


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

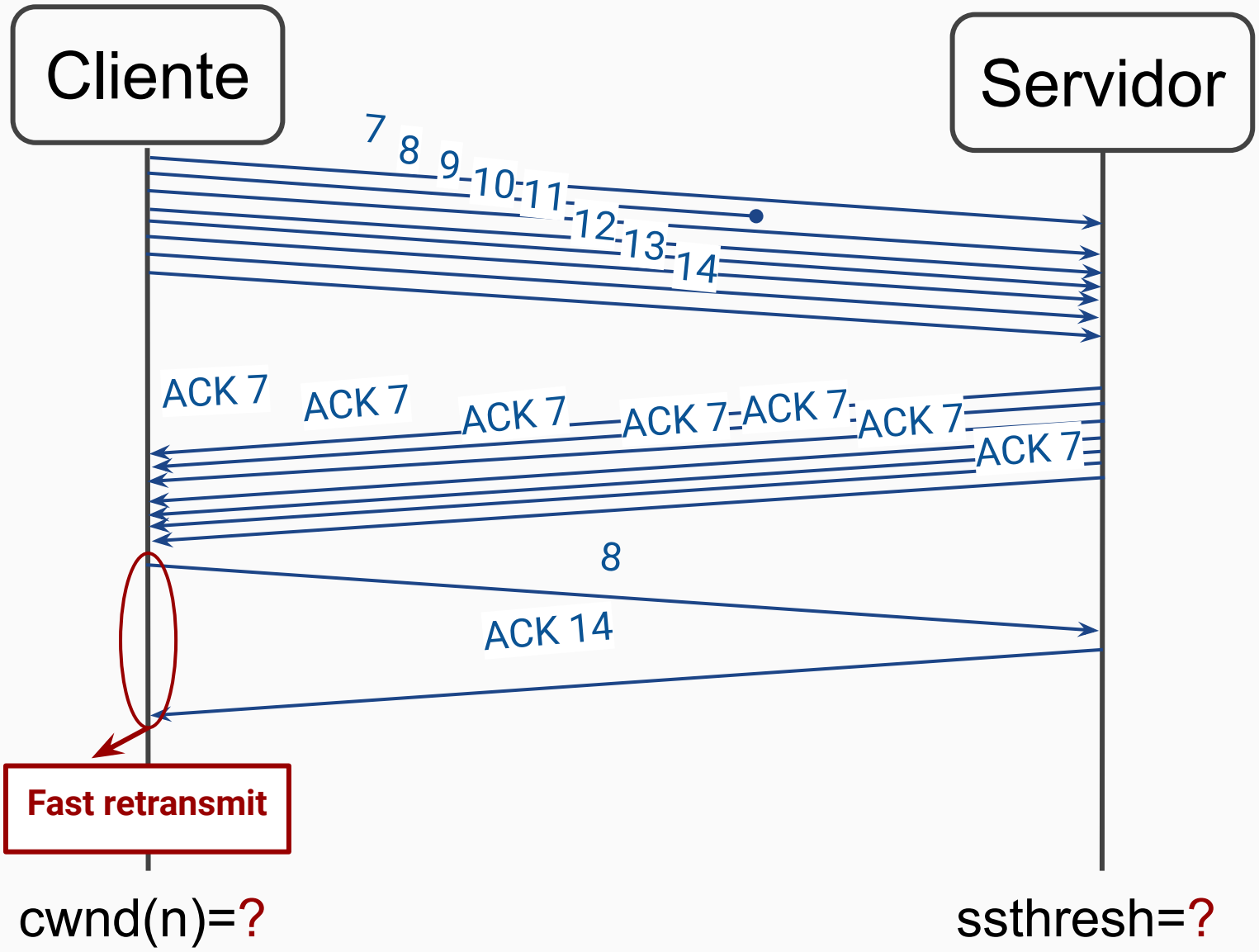
Slow start


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=?

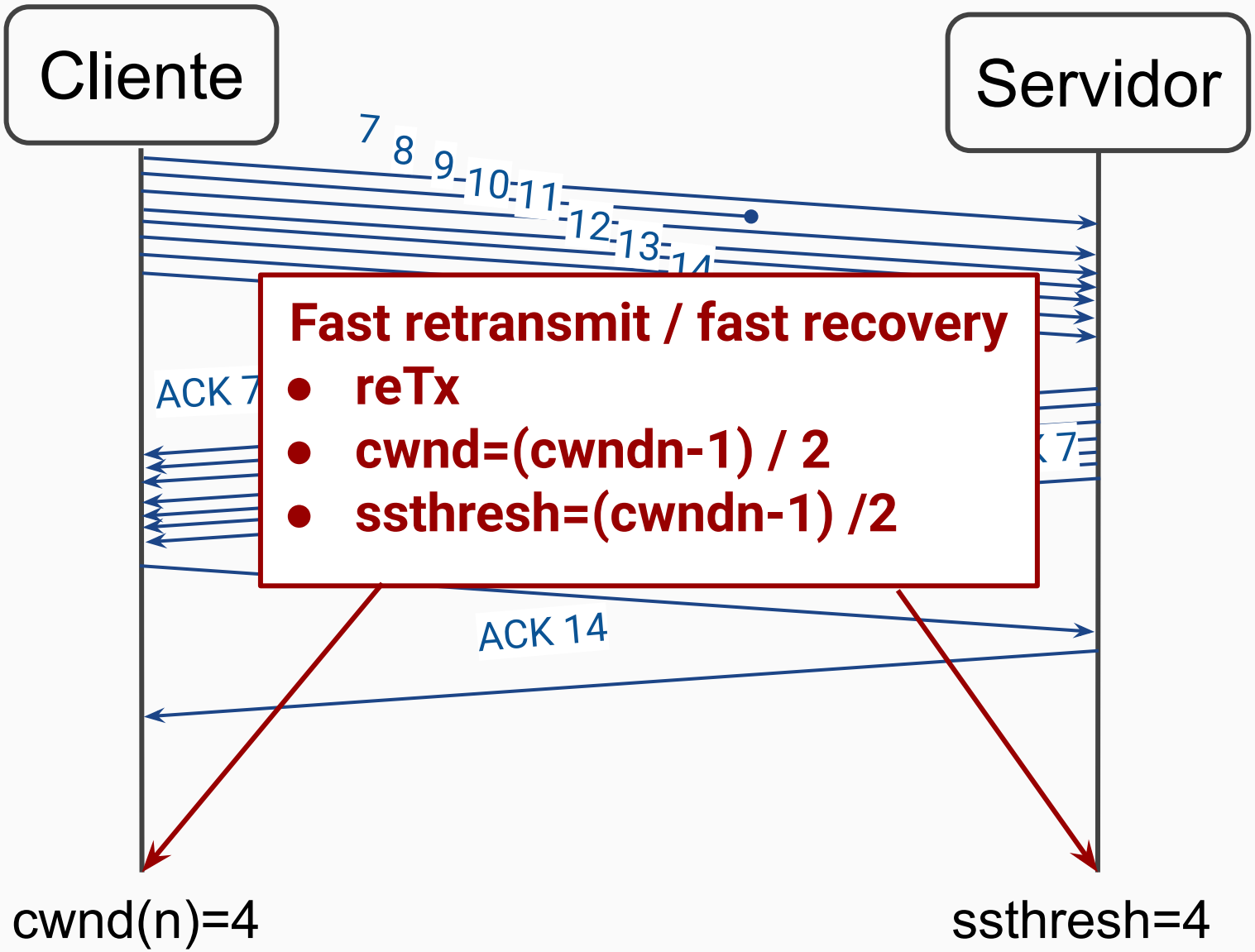
Slow start


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=?

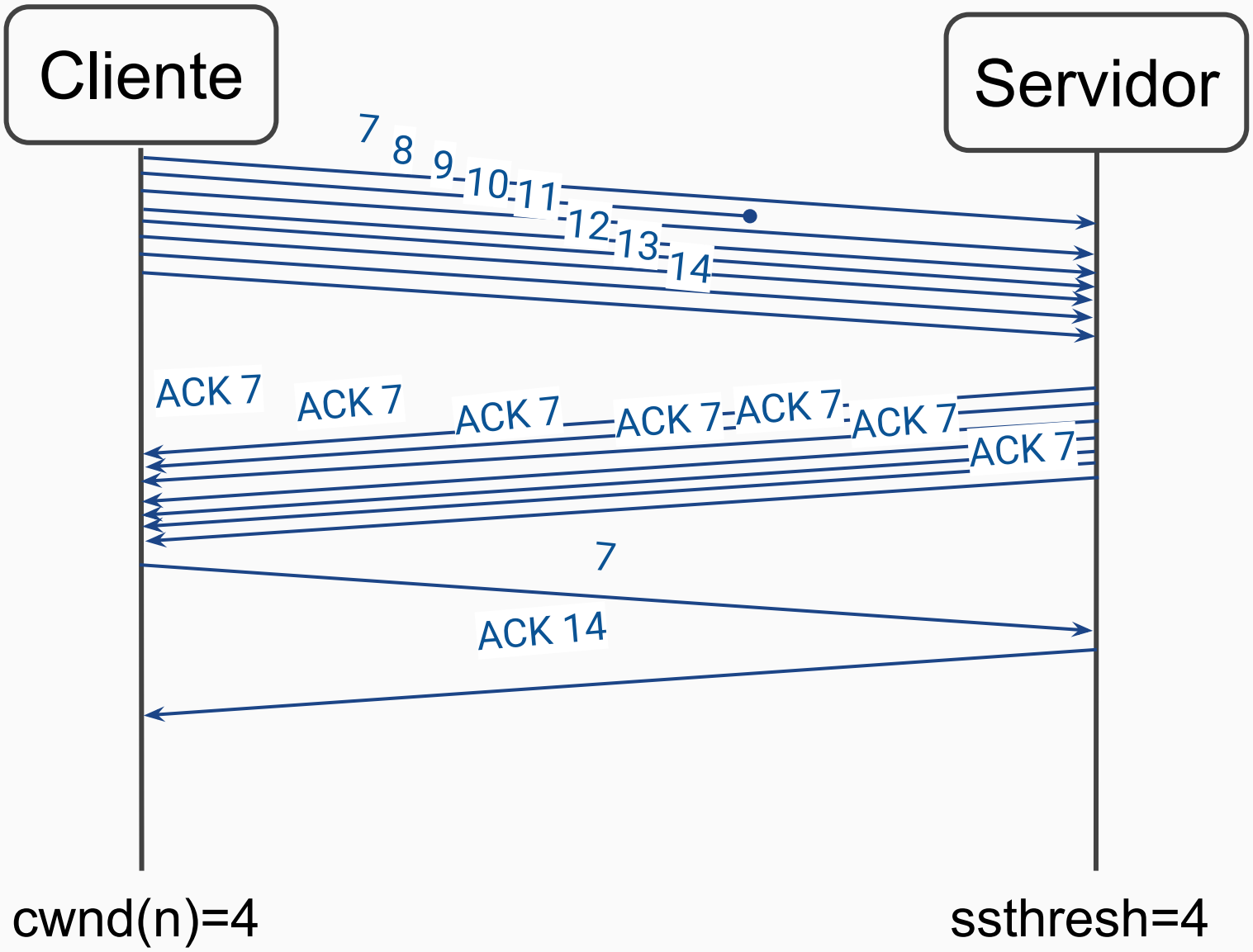
Fast retransmit



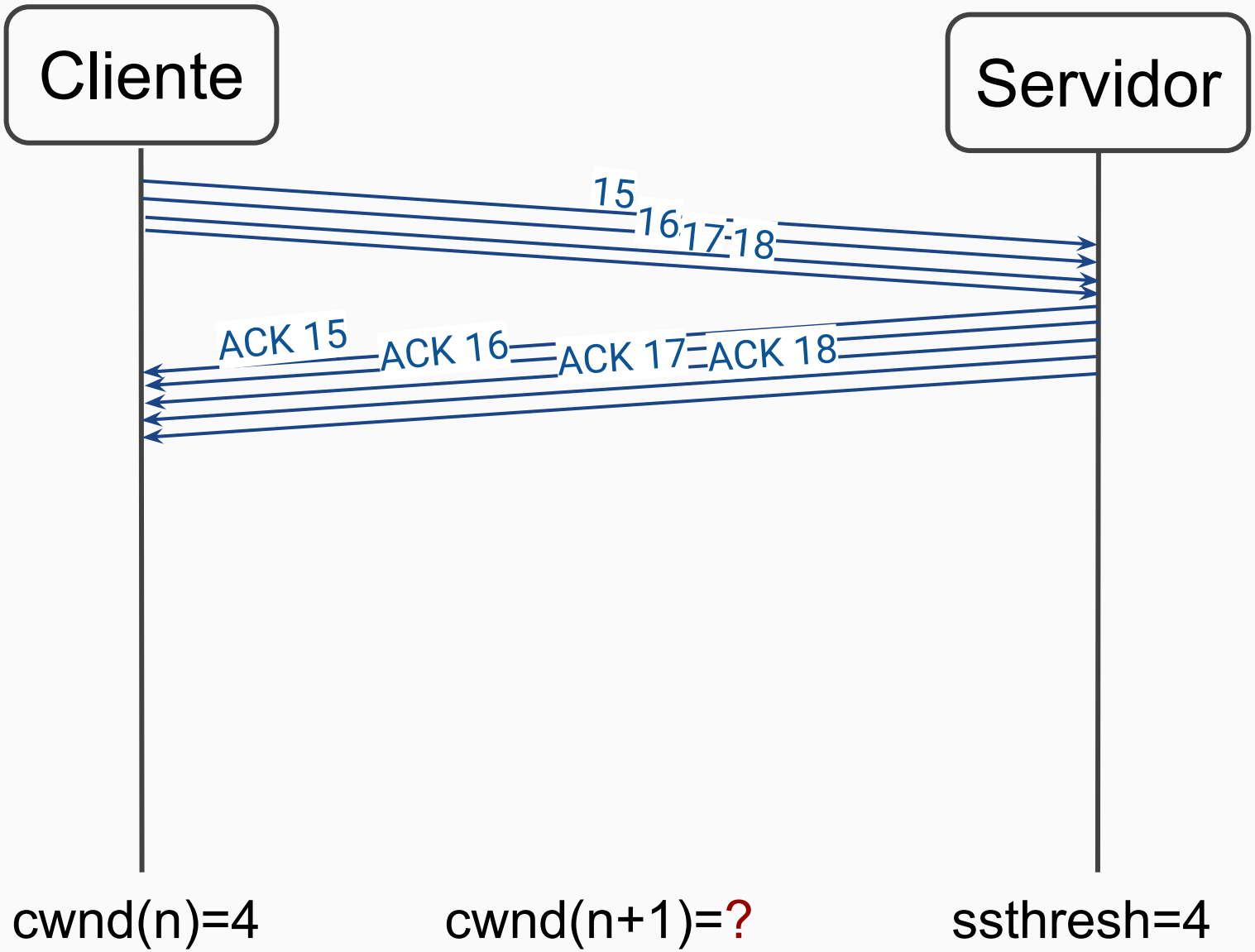
Fast retransmit



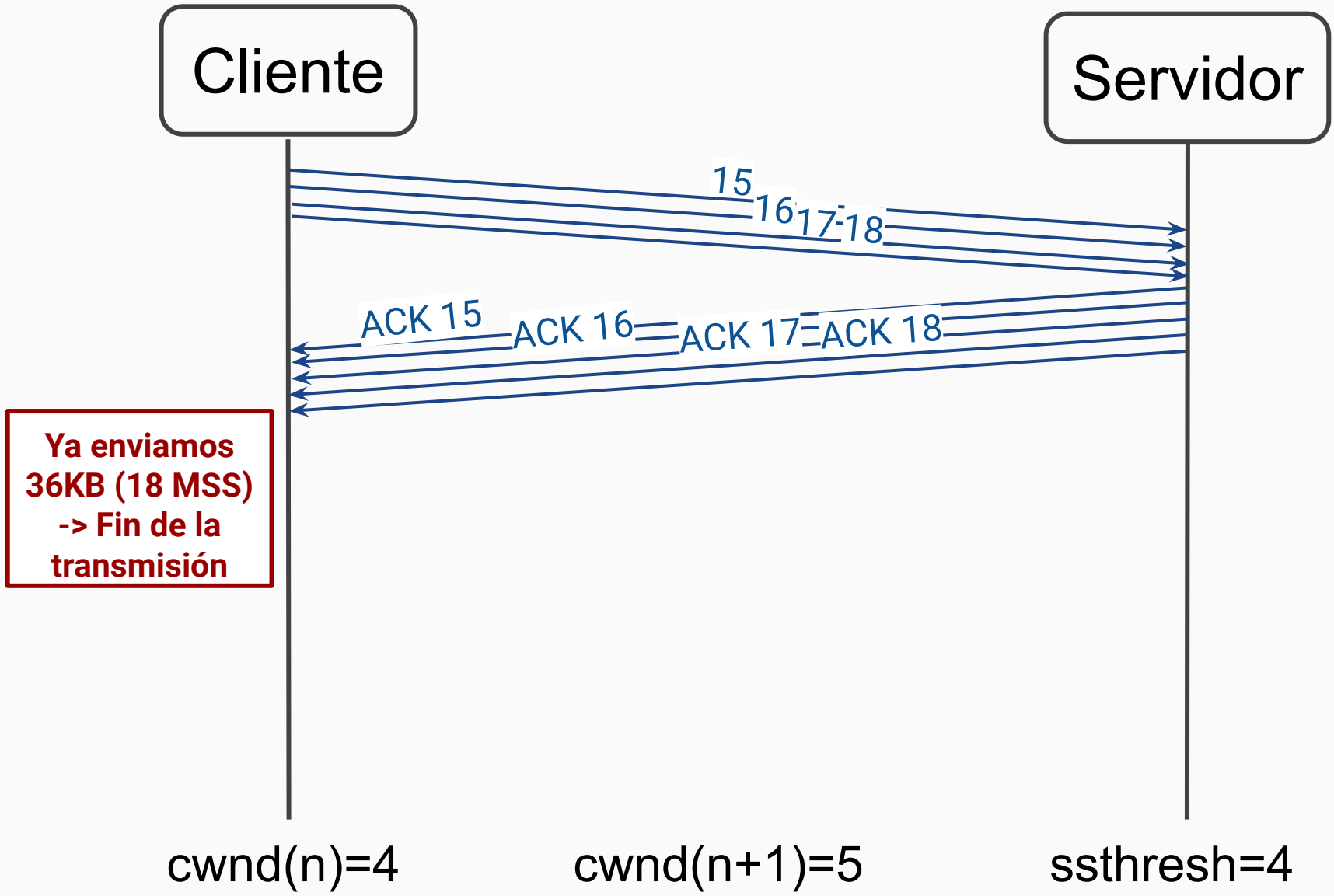
Fast retransmit



Fast retransmit



Fast retransmit



TCP : control de congestión

Otros algoritmos

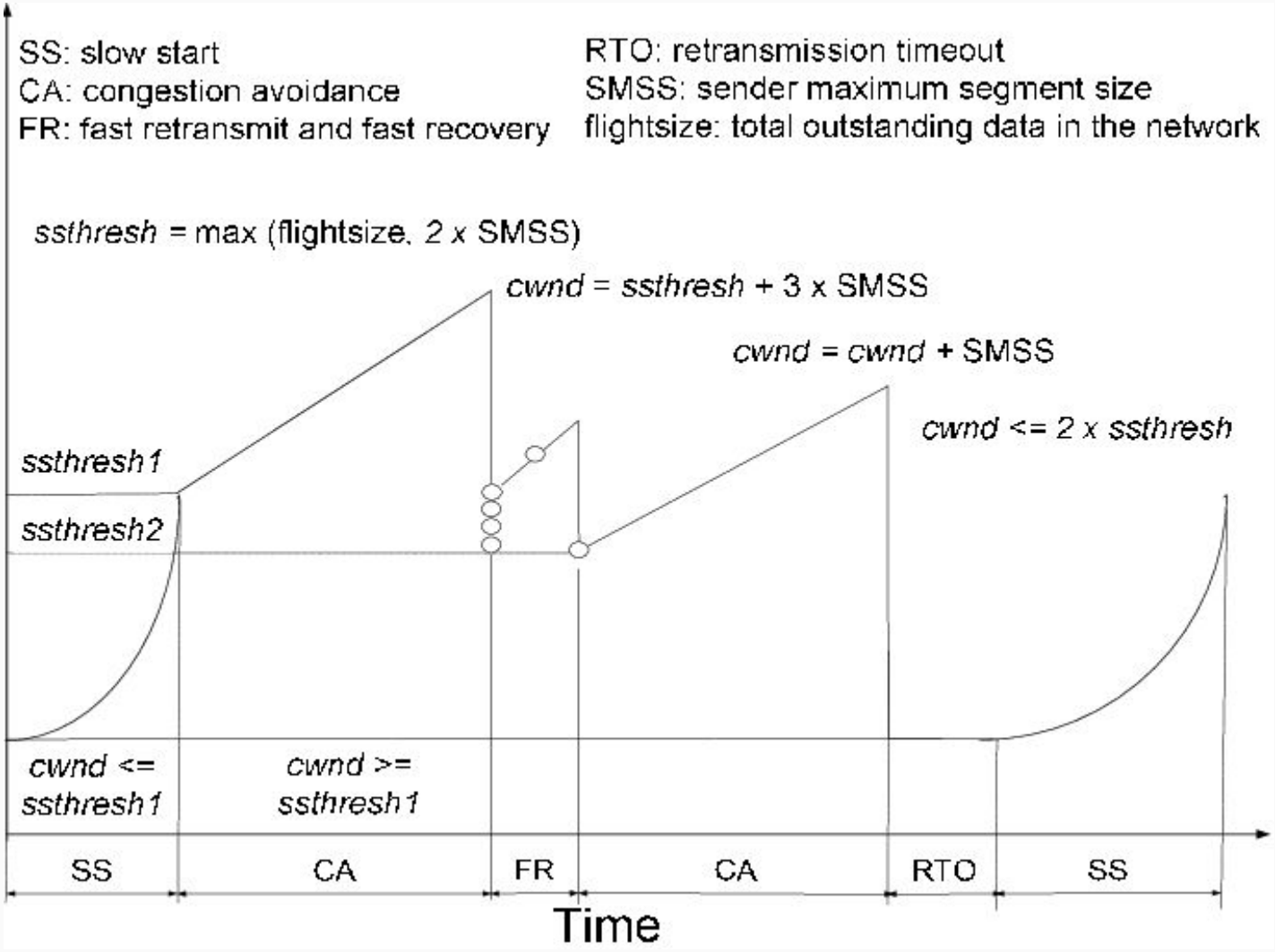
- TCP Tahoe y Reno son los más simples a nivel didáctico
- En la práctica ya no se utilizan. Hoy en día se usan algoritmos más modernos y complicados

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)

TCP : control de congestión



¿Preguntas?

Referencias

- TCP: RFC 793
- TCP - Control de Congestion: RFC 5681
- Captura TCP: https://github.com/packetpioneer/youtube/blob/main/TCP%20Congestion%20Control.pcapng