upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Gestión y Planificación de Redes y Servicios
Área de Ingeniería Telemática

TCP: Ventana de control de flujo y timers

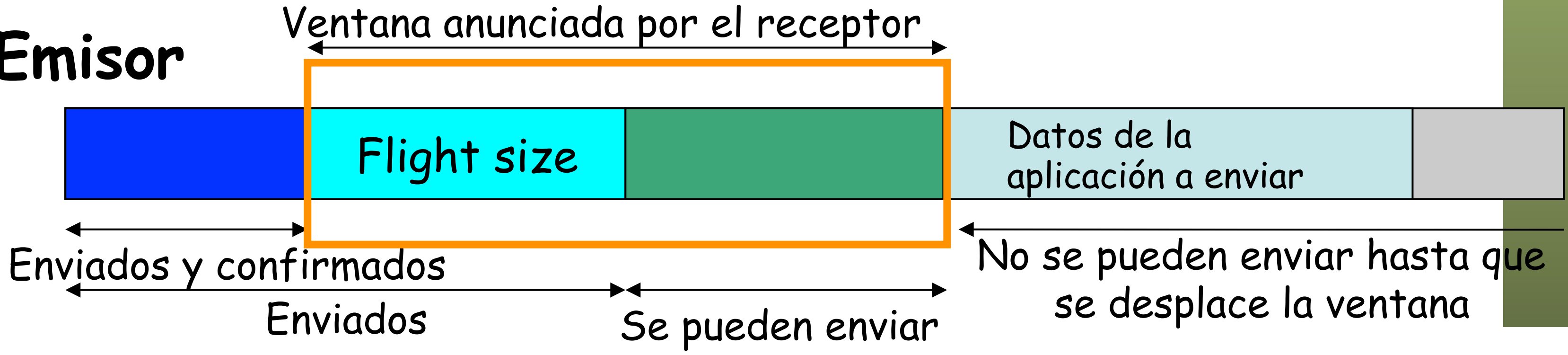
Area de Ingeniería Telemática
http://www.tlm.unavarra.es

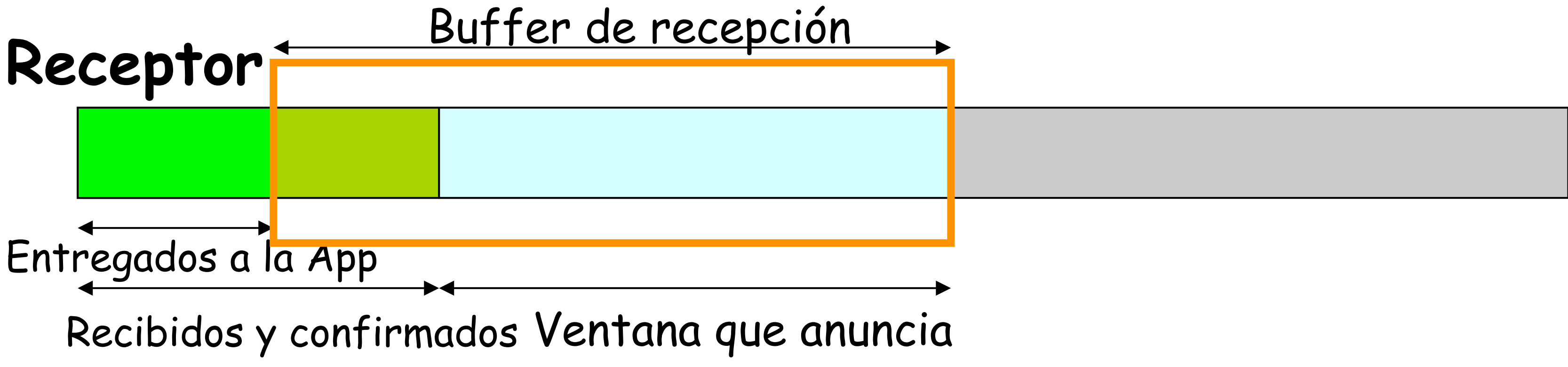
Grado en Ingeniería en Tecnologías de Telecomunicación, 4º

Control de flujo: Ventana deslizante

Ventana: Básico

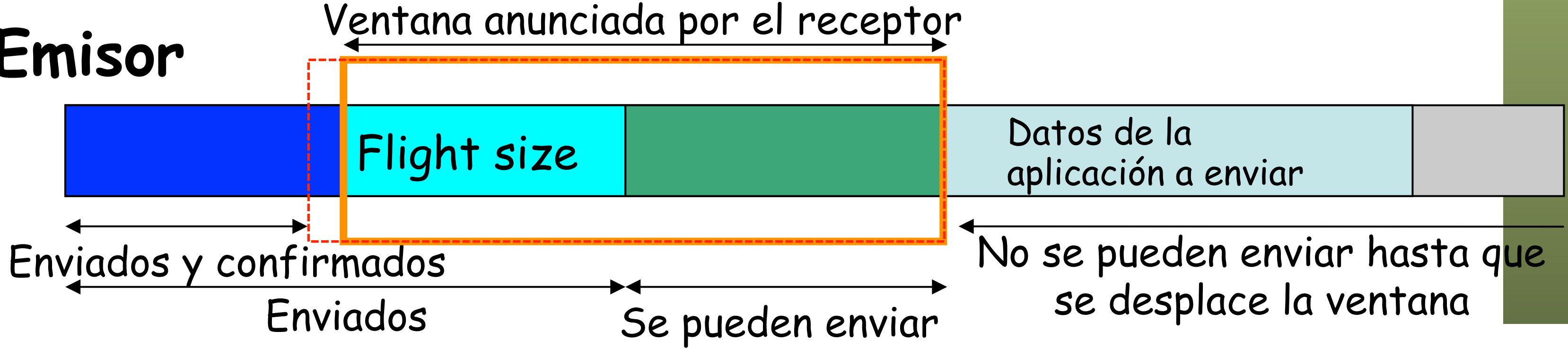
- Por simplicidad analicemos solo un sentido

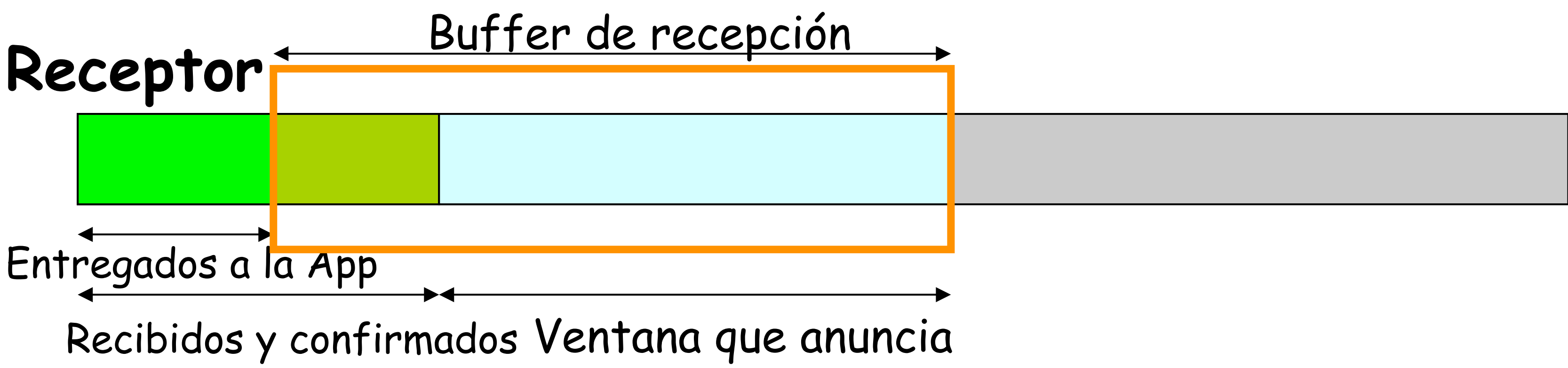




Ventana: Básico

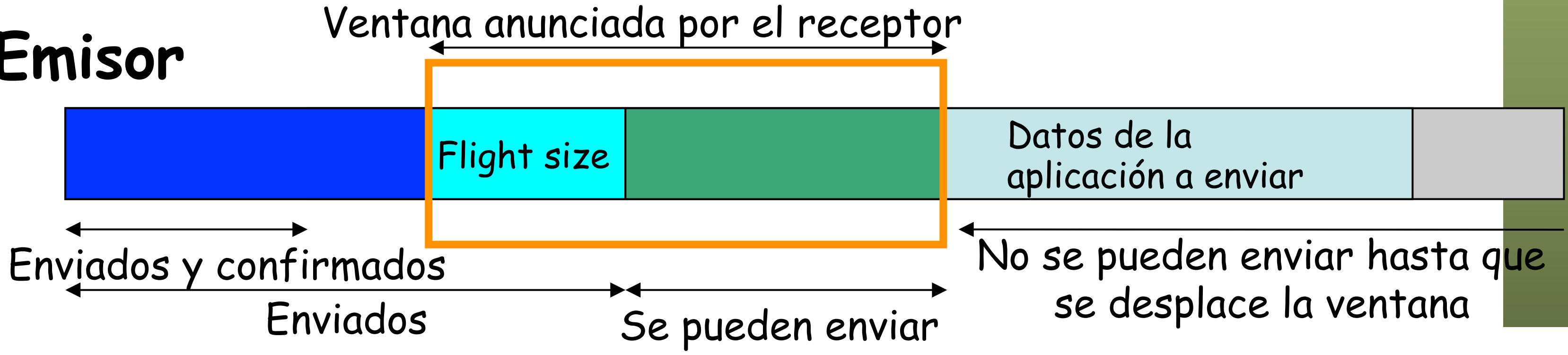
- Por simplicidad analicemos solo un sentido
- **Se reciben más confirmaciones**
- La ventana se desliza en el emisor
- Pero si la aplicación receptora no ha leído los datos es probable que anuncie una ventana menor

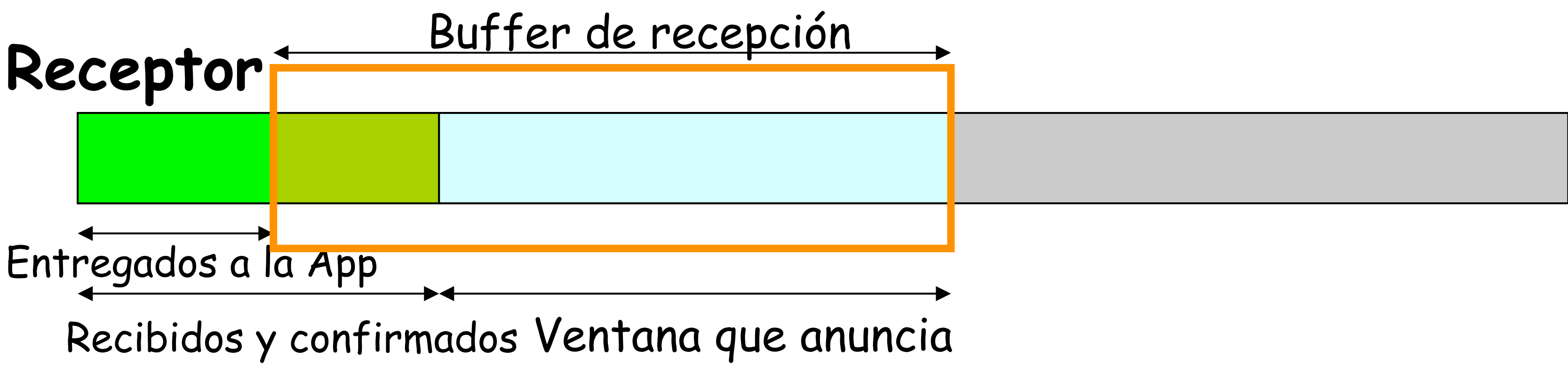




Ventana: Básico

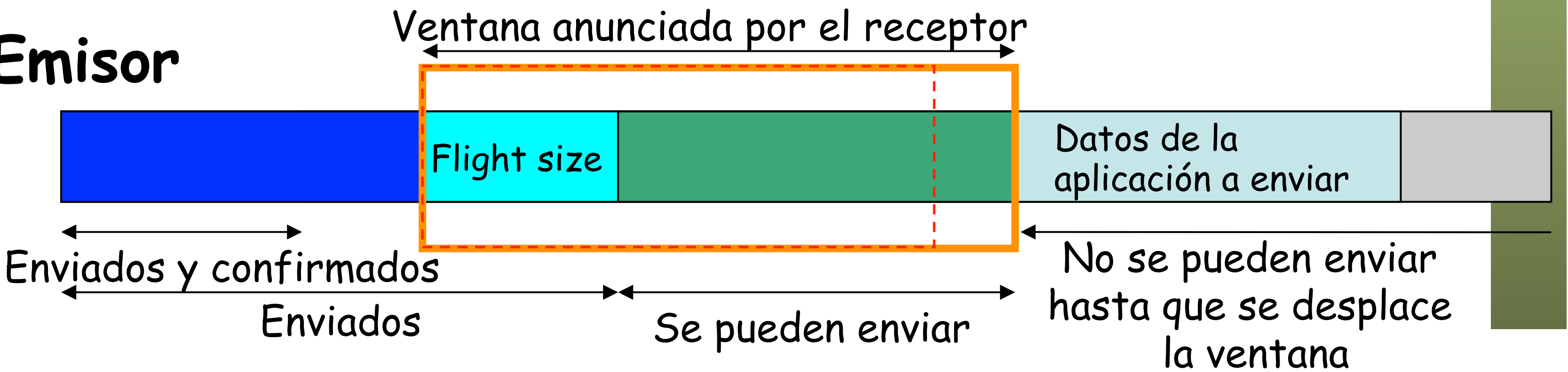
- Por simplicidad analicemos solo un sentido
- **Se reciben más confirmaciones**
- En este caso se han recibido ya todas las confirmaciones
- Pero aún hay datos en vuelo que no han llegado al receptor

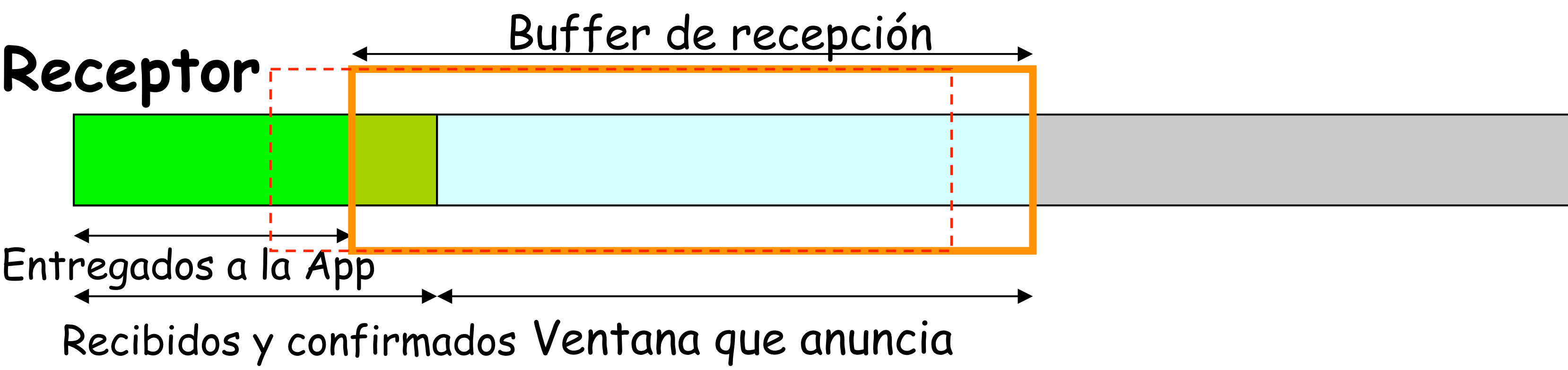




Ventana: Básico

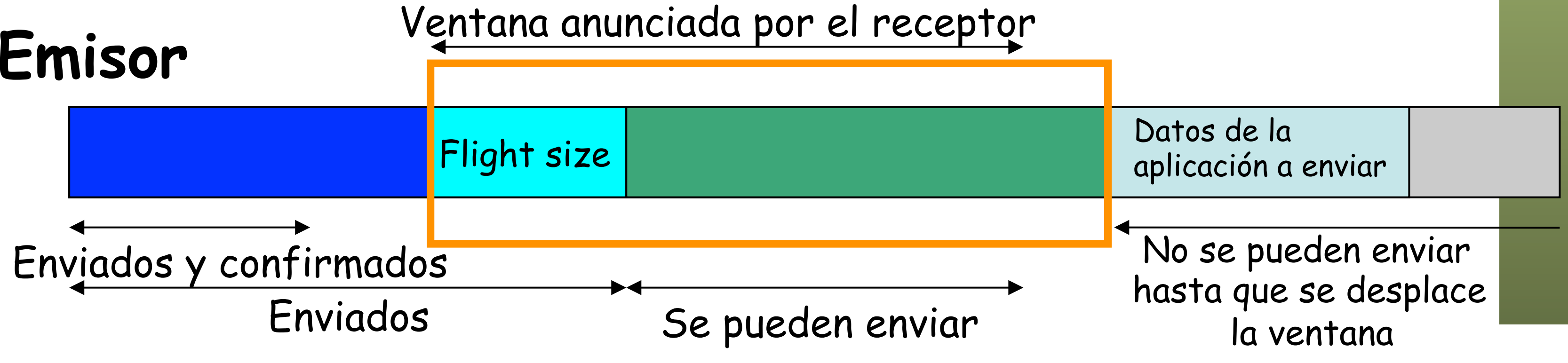
- Por simplicidad analicemos solo un sentido
- La aplicación **receptora lee bytes** del stream
  - La ventana se desliza en el receptor
  - Y se abre en el emisor (*window update*) (…)

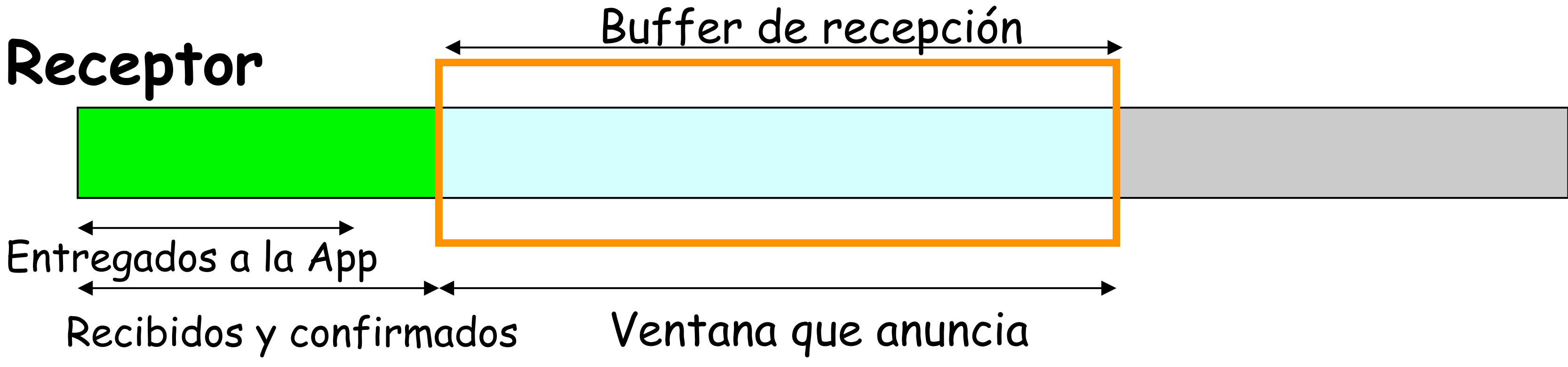




Ventana: Básico

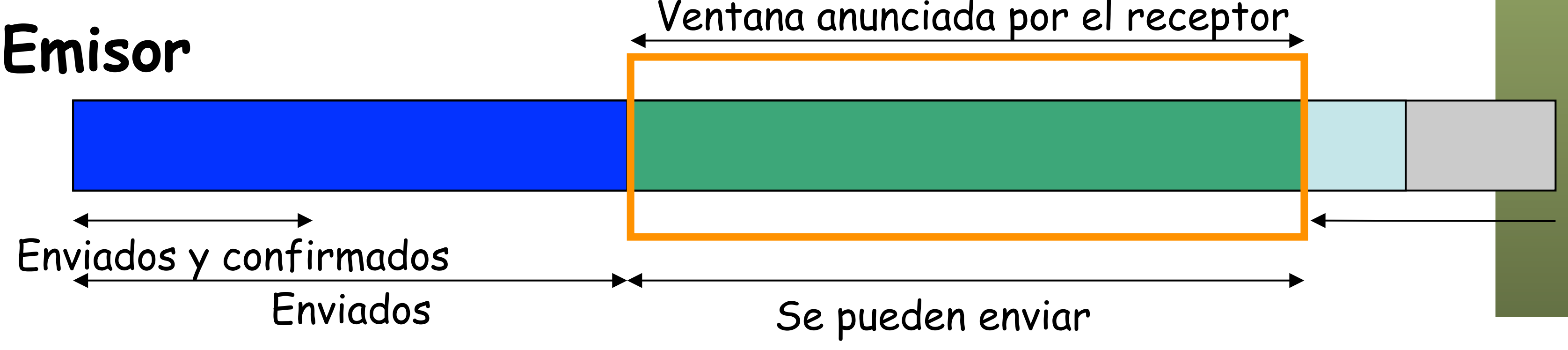
- Por simplicidad analicemos solo un sentido
- En este caso la aplicación receptora ha sacado todo del buffer

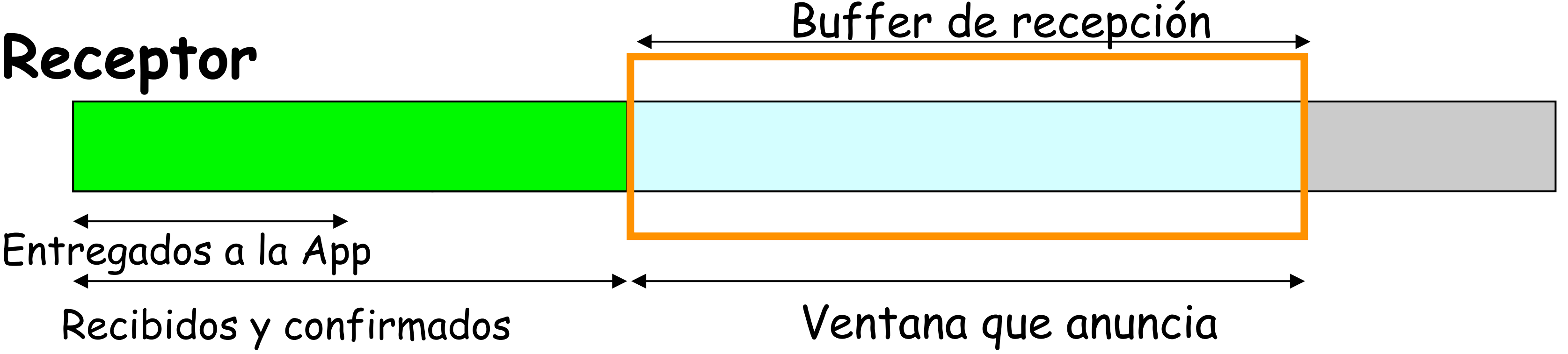




Ventana: Básico

- En muchas ocasiones se dan varios de los fenómenos de forma simultánea
- Por ejemplo llegan datos al receptor y antes de que se envíe la confirmación los lee la aplicación
- Así, se manda la confirmación manteniendo el valor de la ventana
- (…)

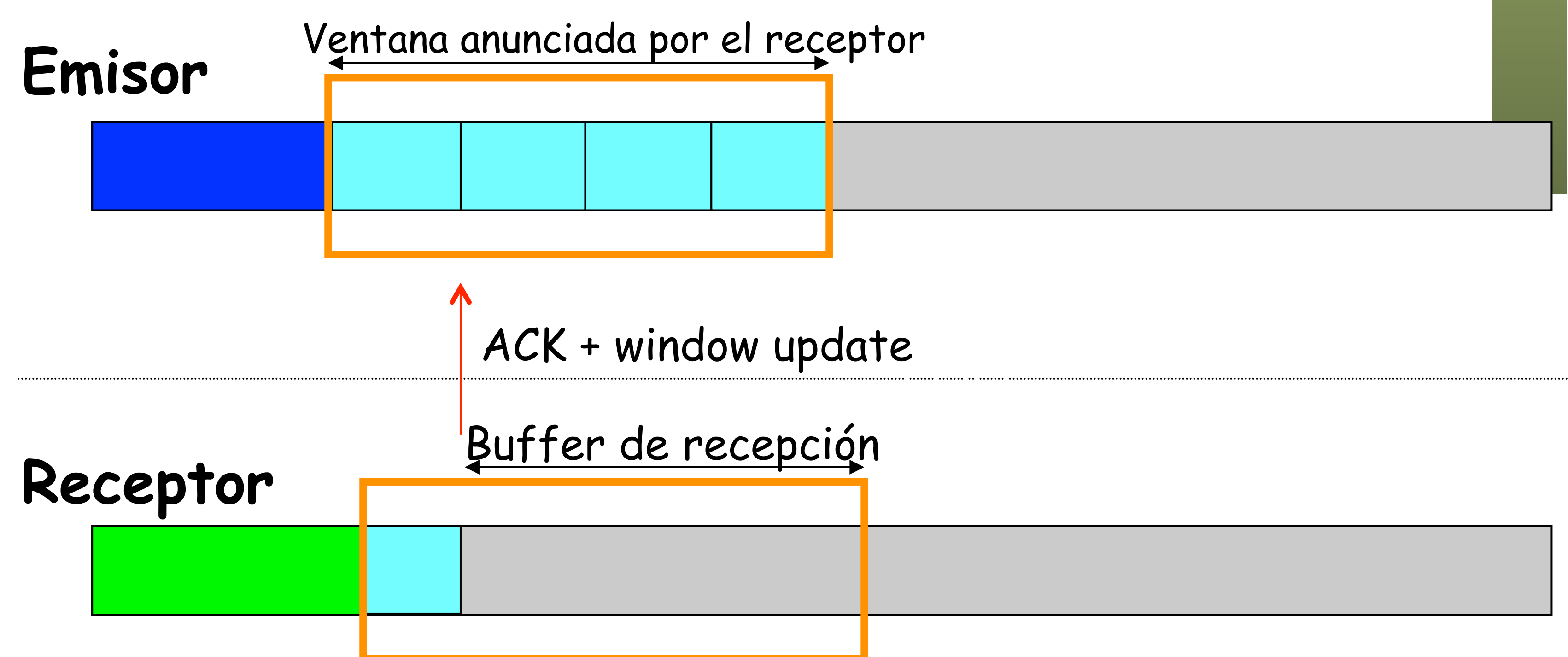




Silly Window Syndrome

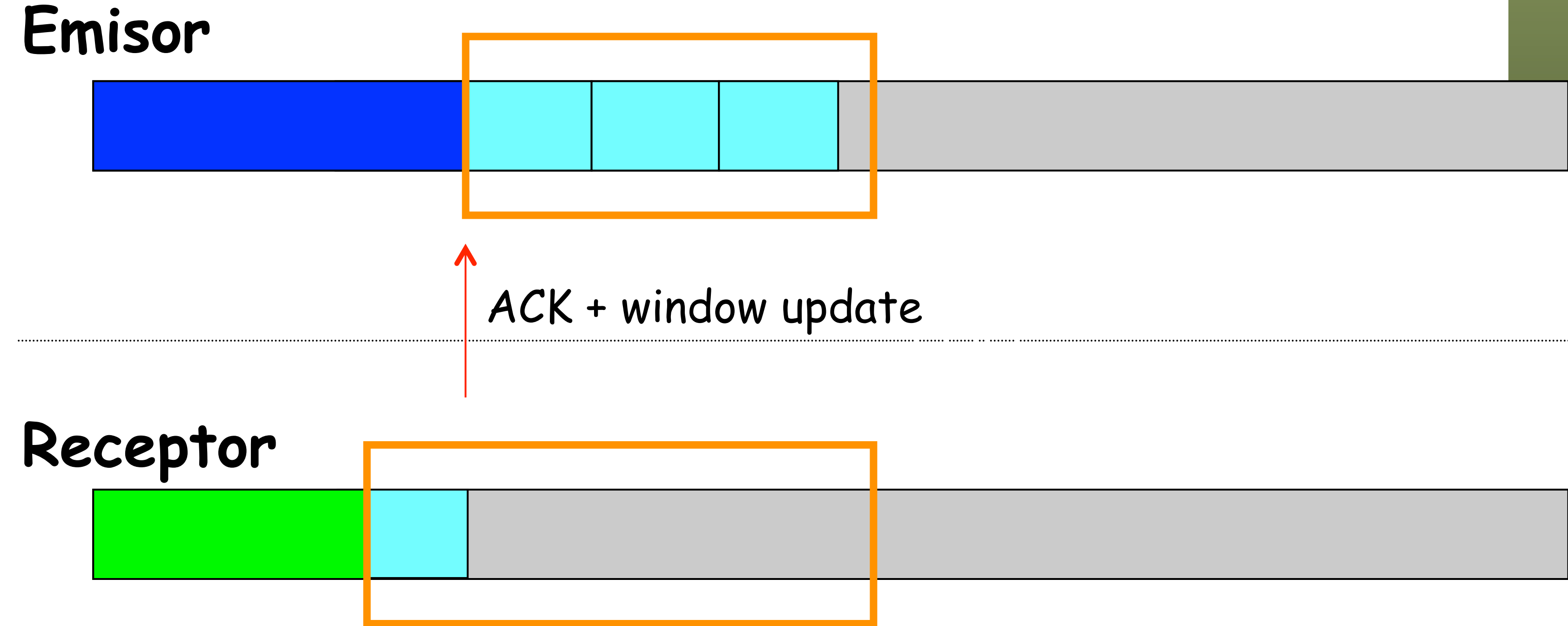
Ventana: SWS

- *Silly Window Syndrome*, David D. Clark, RFC 813
- Ejemplo:
  - Emisor envía la ventana en 4 segmentos
  - Llega el primero y lo confirma, pero la aplicación solo lee parte (...)



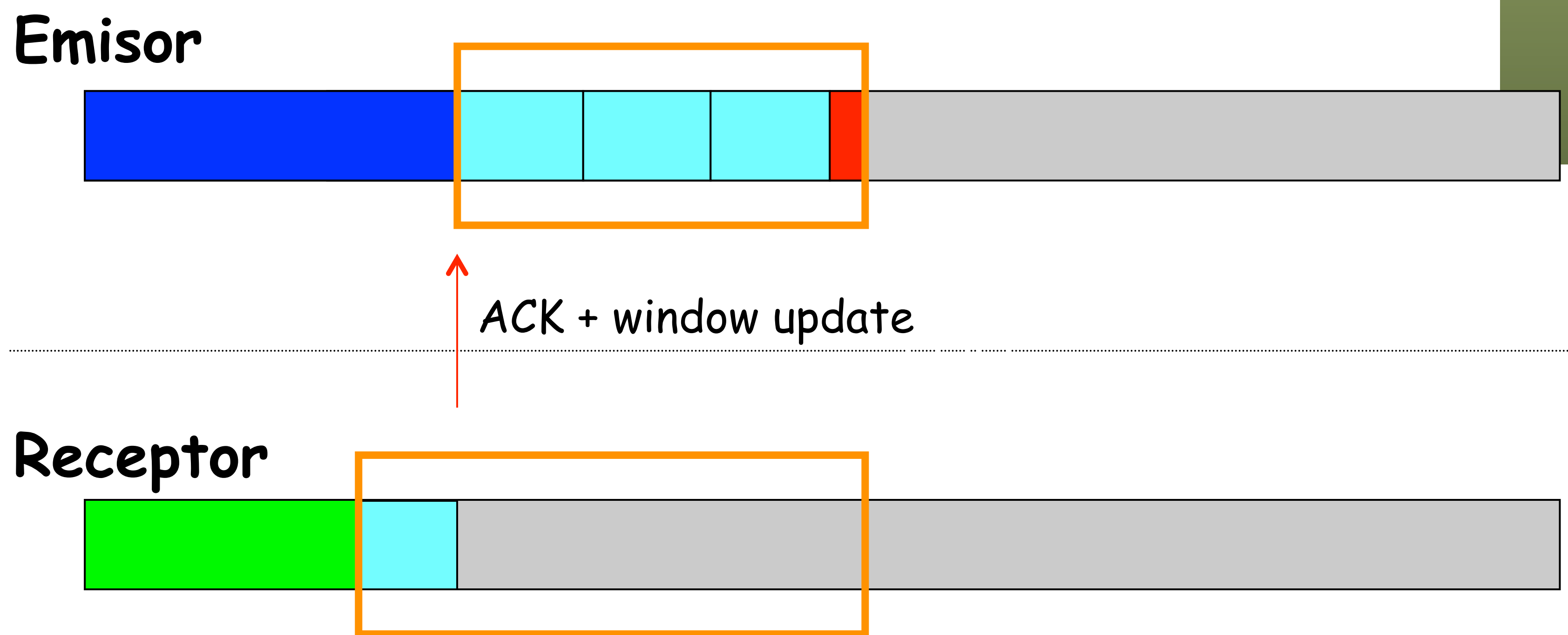
Ventana: SWS

- *Silly Window Syndrome*, David D. Clark, RFC 813
- Ejemplo:
  - Emisor envía la ventana en 4 segmentos
  - Llega el primero y lo confirma, pero la aplicación solo lee parte
  - Se desplaza y cierra la ventana en el emisor (...)



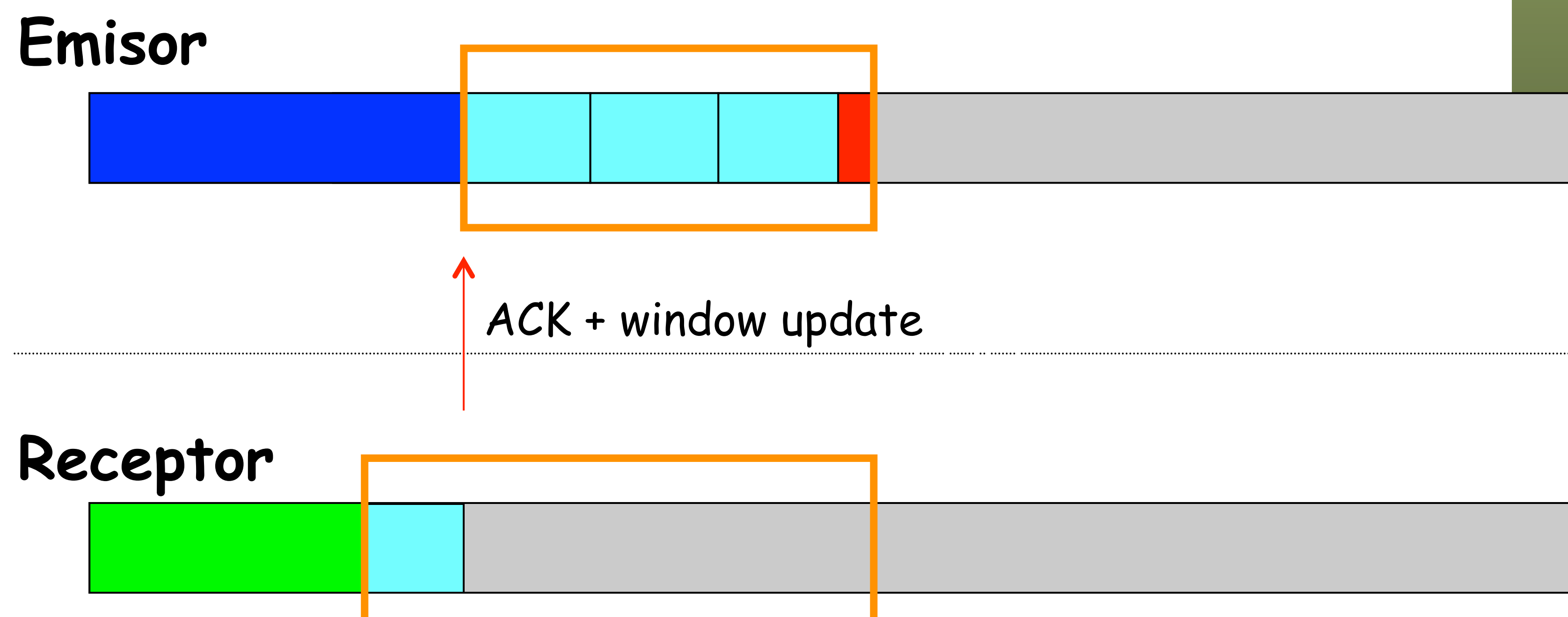
Ventana: SWS

- *Silly Window Syndrome*, David D. Clark, RFC 813
- Ejemplo:
  - Emisor envía la ventana en 4 segmentos
  - Llega el primero y lo confirma, pero la aplicación solo lee parte
  - Se desplaza y cierra la ventana en el emisor
  - Ahora puede enviar un segmento nuevo, pero es pequeño



Ventana: SWS

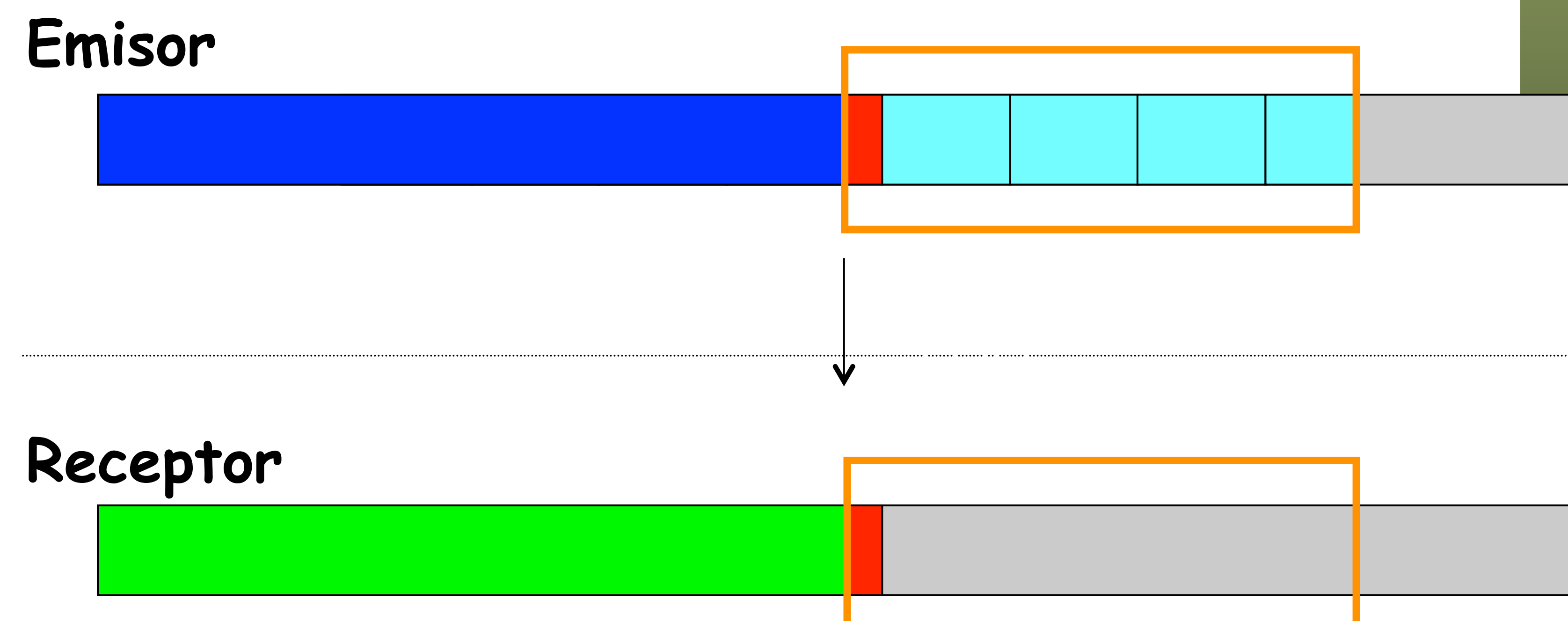
- Ejemplo:
  - Tiempo más tarde (...)



Gestión y Planif. Redes y Servs.
Área de Ingeniería Telemática

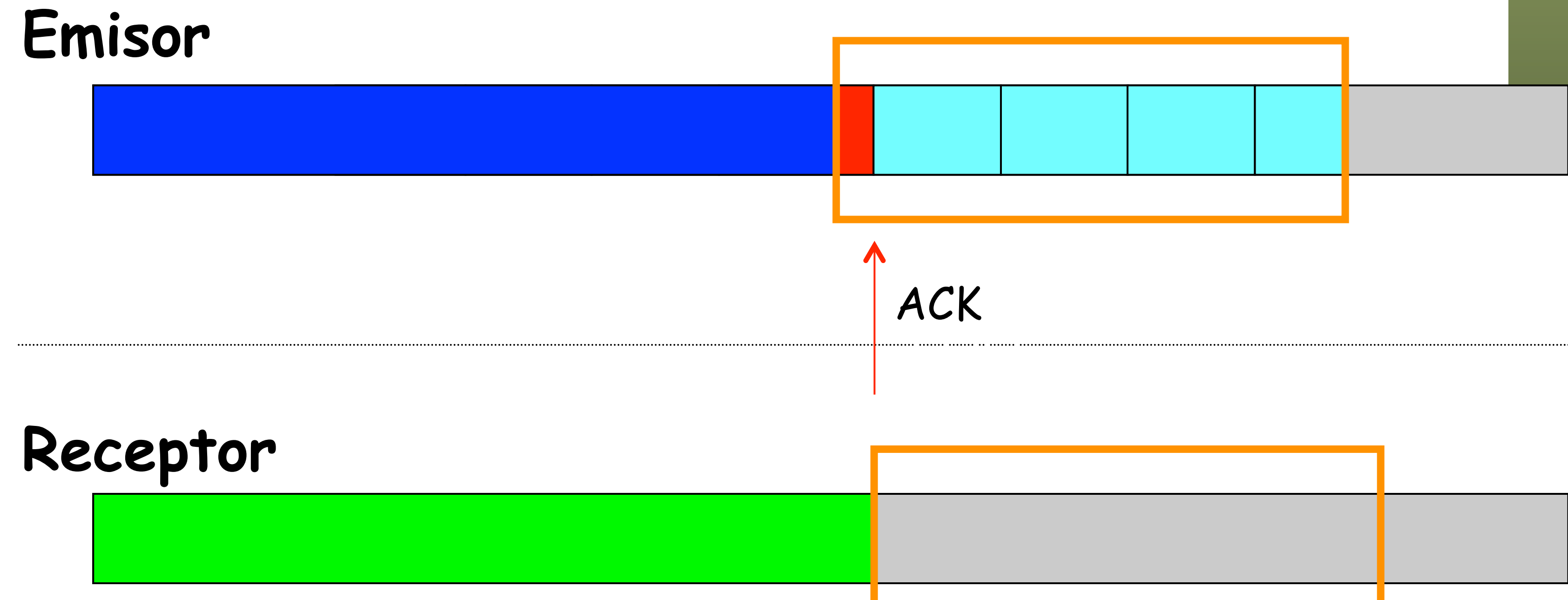
Ventana: SWS

- Ejemplo:
  - Tiempo más tarde ese segmento pequeño llega (...)



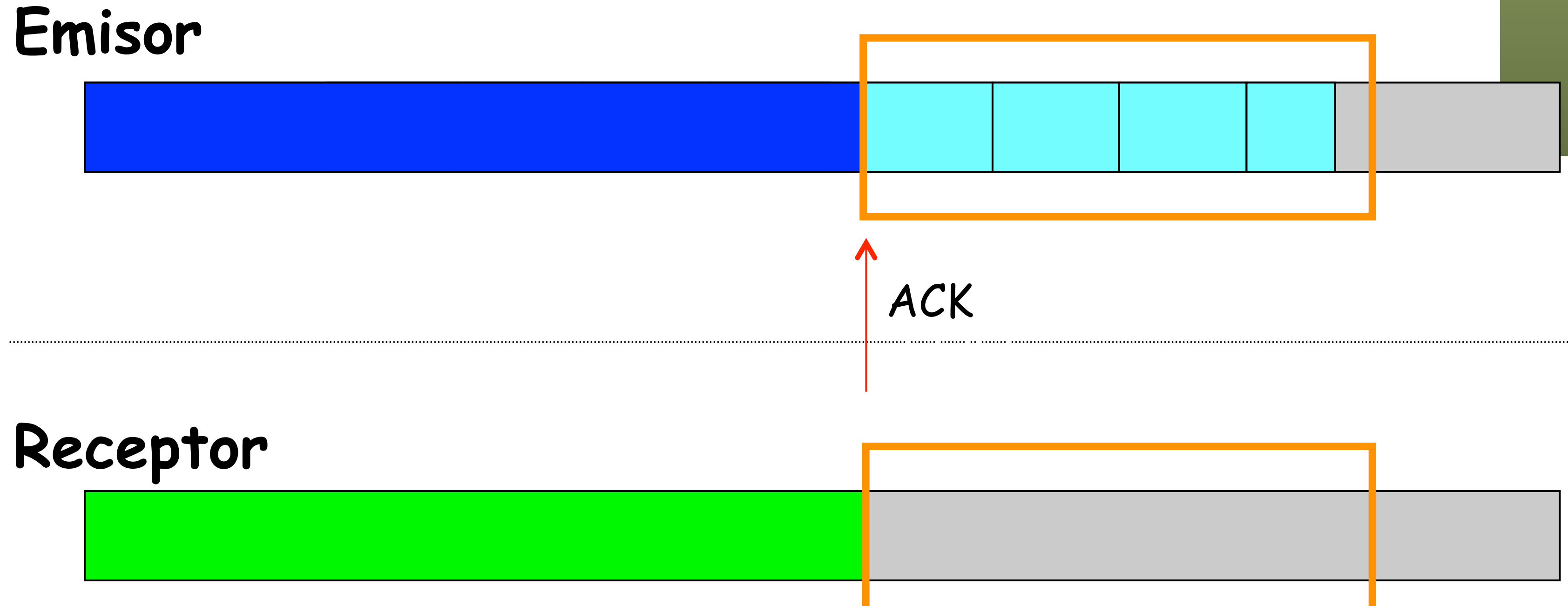
Ventana: SWS

- Ejemplo:
  - Tiempo más tarde ese segmento pequeño llega y se confirma
  - (...)



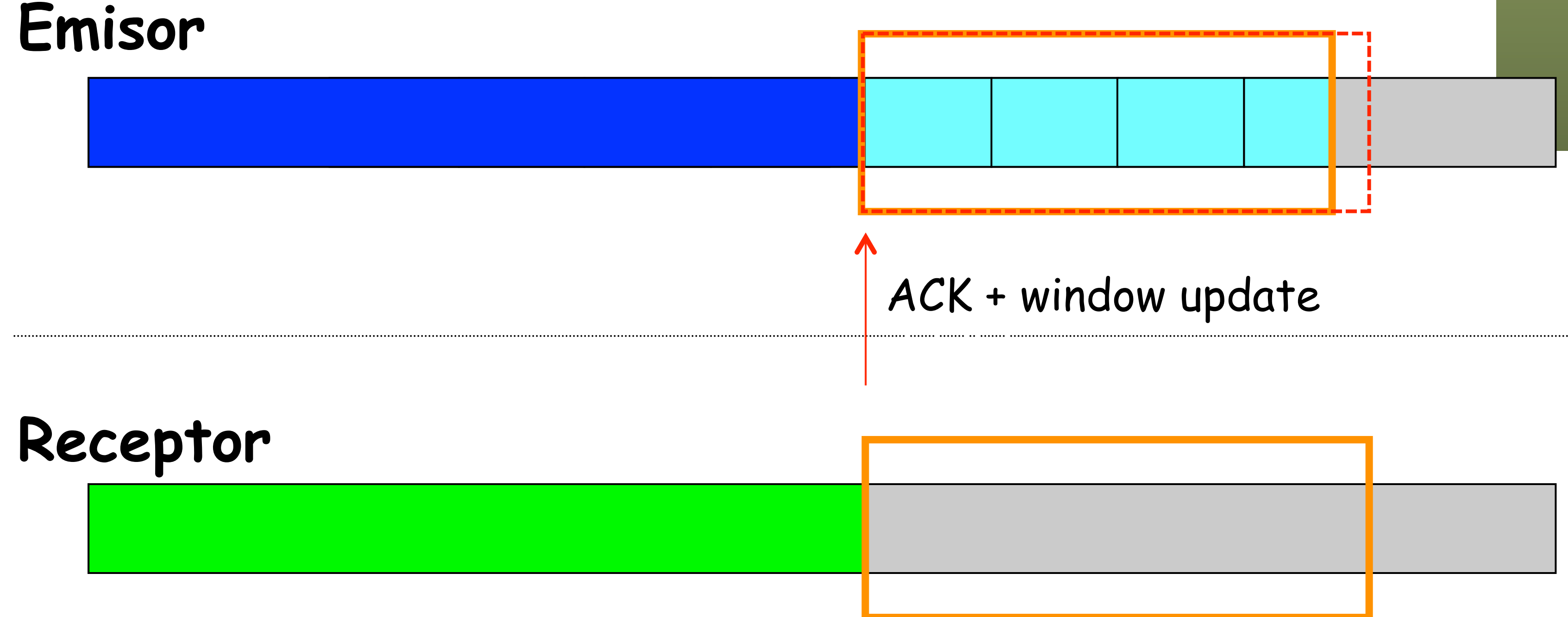
Ventana: SWS

- Ejemplo:
  - Tiempo más tarde ese segmento pequeño llega y se confirma
  - Eso desplaza la ventana solo ese poco
  - Y ahora, si hay nuevos datos, solo se pueden enviar en un segmento pequeño
  - (...)



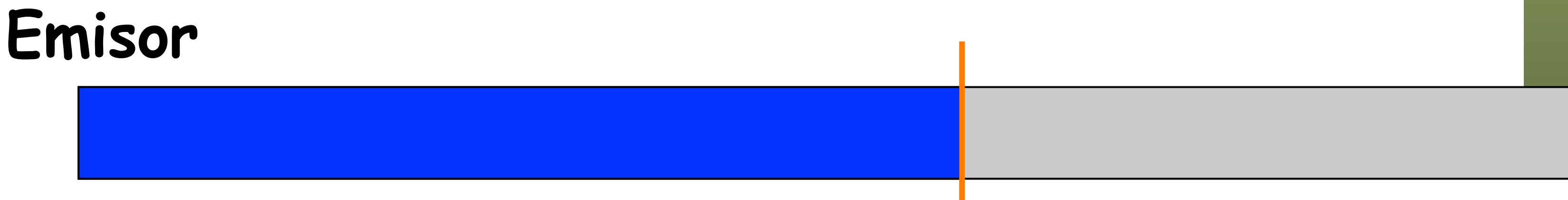
Ventana: SWS

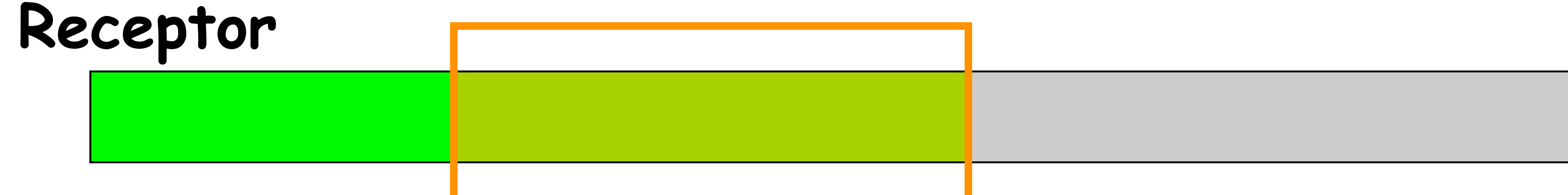
- Ejemplo:
  - Tiempo más tarde ese segmento pequeño llega y se confirma
  - Eso desplaza la ventana solo ese poco
  - Y ahora, si hay nuevos datos, solo se pueden enviar en un segmento pequeño
  - Para evitar esto, el anuncio de ventana en la confirmación podría reducirla para que no deje hueco hasta que sea al menos del MSS



Ventana: SWS

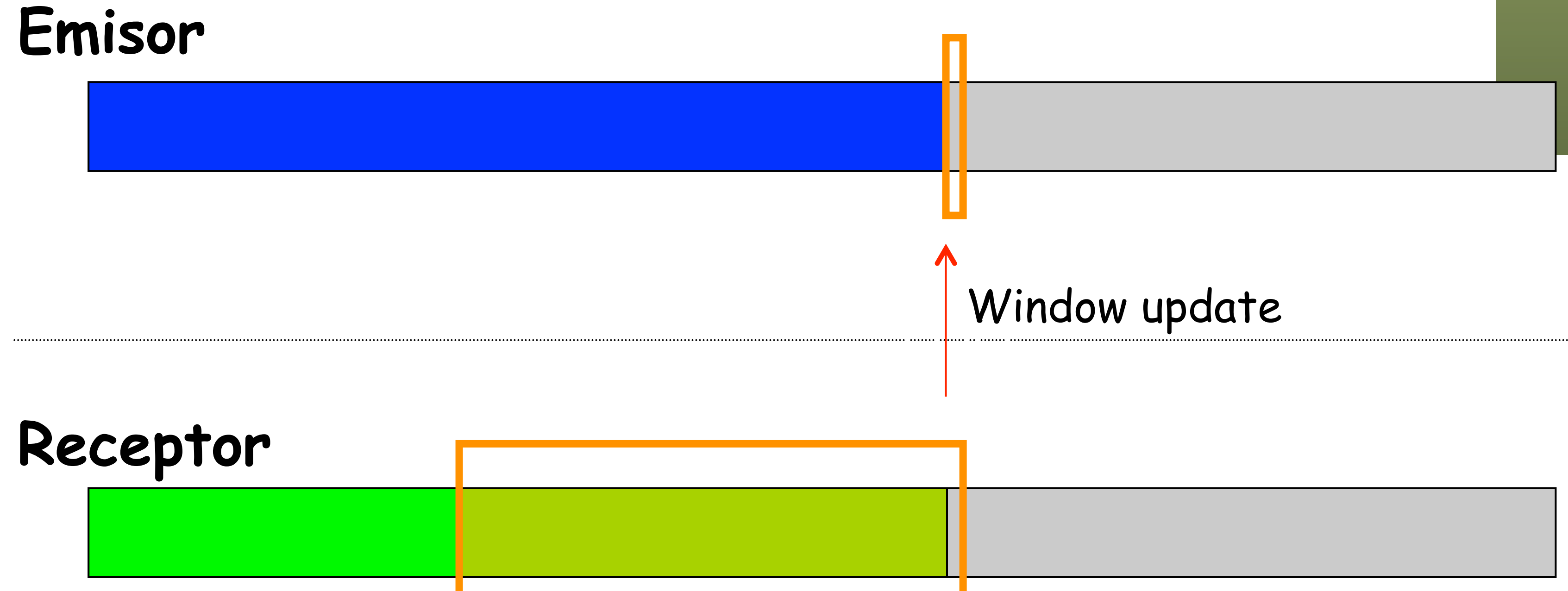
- Otro ejemplo:
  - El receptor está muy ocupado y la ventana cerrada
  - Receptor solo lee del buffer una pequeña cantidad de bytes (...)





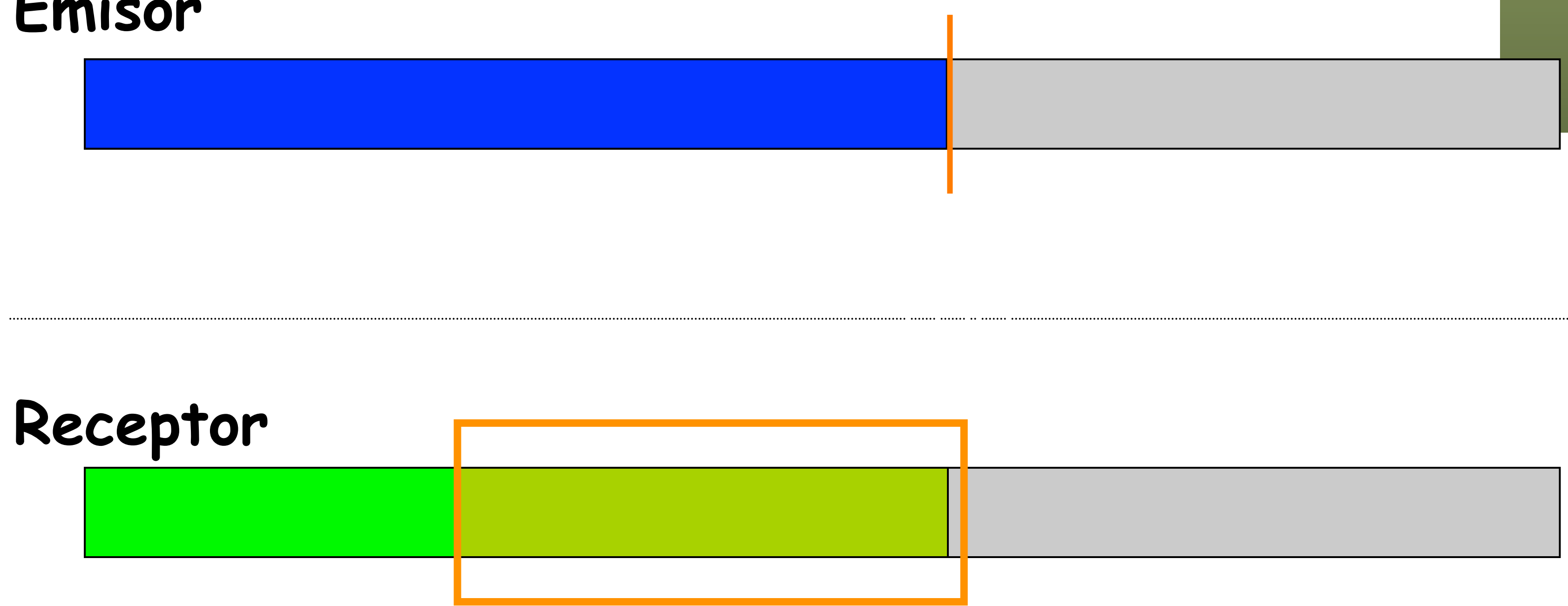
Ventana: SWS

- Otro ejemplo:
  - El receptor está muy ocupado y la ventana cerrada
  - Receptor solo lee del buffer una pequeña cantidad de bytes
  - Abre la ventana solo en esa cantidad con lo que solo se puede enviar un paquete pequeño
  - (...)



Ventana: SWS

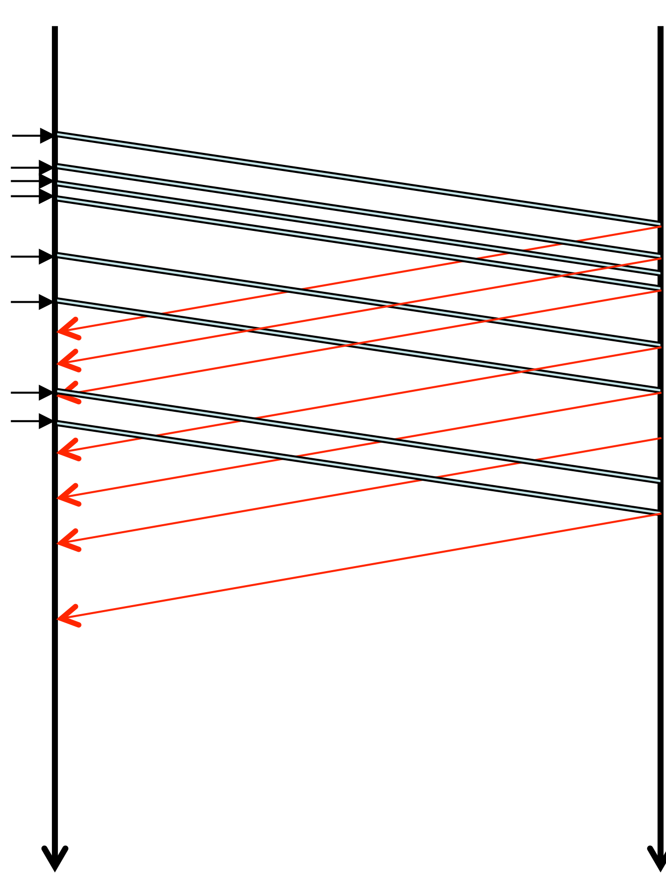
- Otro ejemplo:
  - El receptor está muy ocupado y la ventana cerrada
  - Receptor solo lee del buffer una pequeña cantidad de bytes
  - Abre la ventana solo en esa cantidad con lo que solo se puede enviar un paquete pequeño
  - Para evitar esto puede retener el window update hasta poder abrir la ventana en todo un MSS



Algoritmo de Nagle

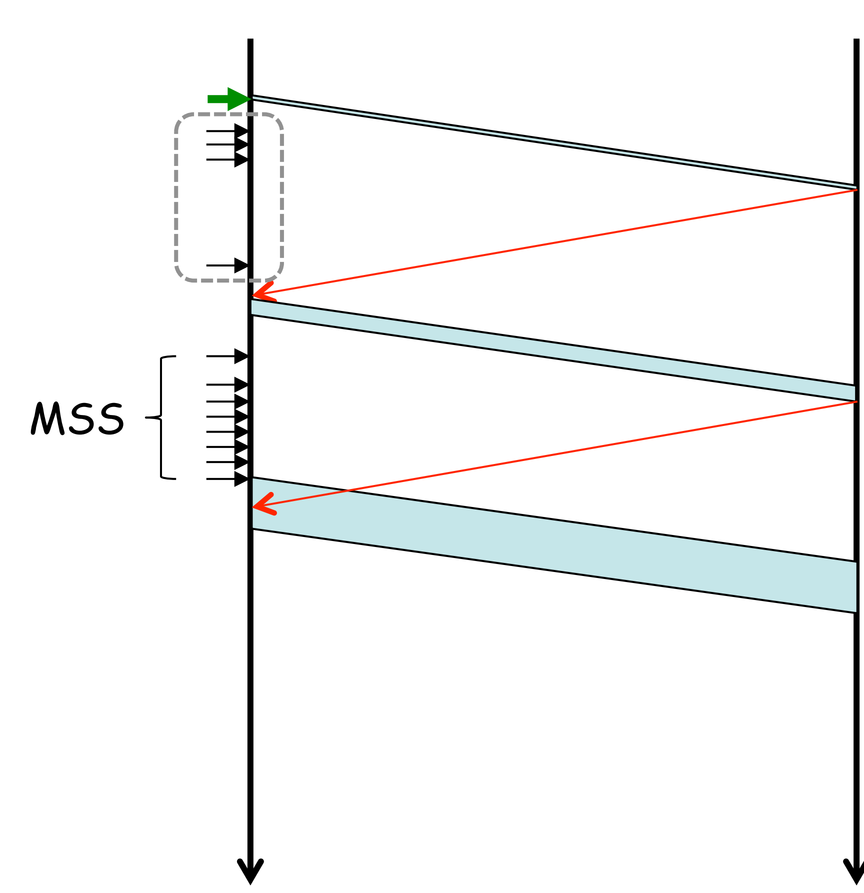
Algoritmo de Nagle

- Fuentes lentas o que generan datos en bloques pequeños
- Dan lugar a una gran cantidad de paquetes pequeños
- Mucha cabecera y pocos datos (1 byte de datos TCP + 20 de cabecera IP + 20 de cabecera TCP en el caso peor)
- (…)



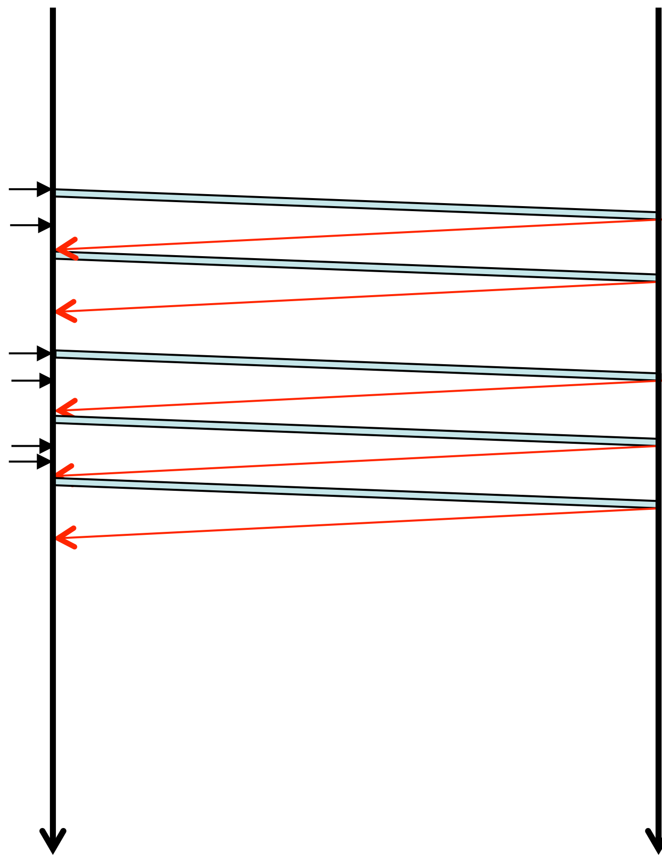
Algoritmo de Nagle

- Mientras haya datos en vuelo sin confirmar: acumular datos
- Una vez que están confirmados todos se puede enviar lo acumulado
- O si se alcanza el MSS
- (...)



Algoritmo de Nagle

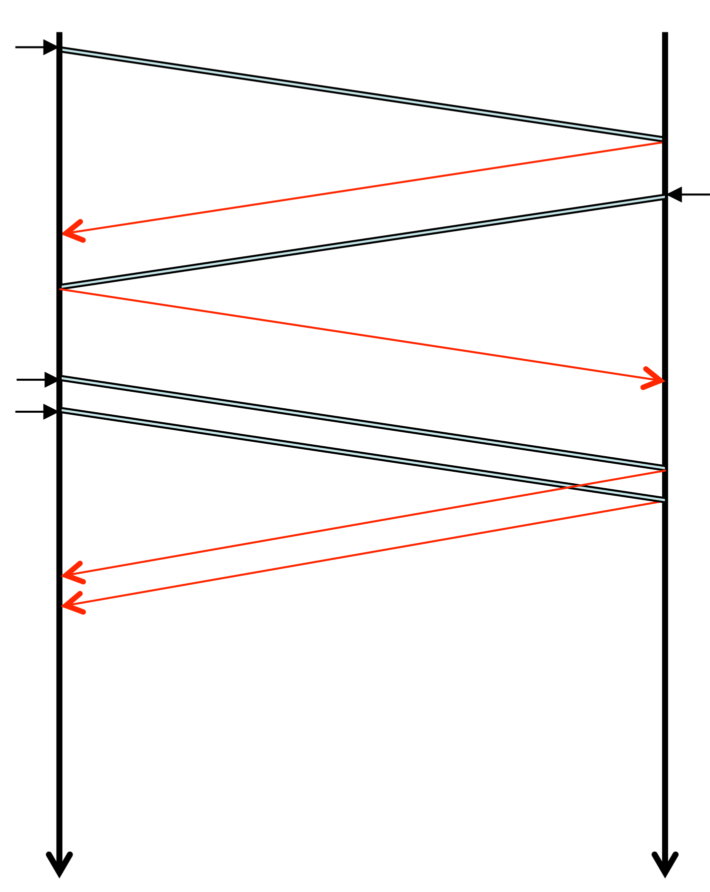
- *Self-clocking*: si los ACKs llegan pronto se envían paquetes que pueden ser pequeños
- Ahorra capacidad en los enlaces
- Aumenta el retardo para la aplicación por el buffering
- No afecta si se envían bloques grandes
- Se puede desactivar si compensa el menor RTT frente al *overhead*



Delayed ACK

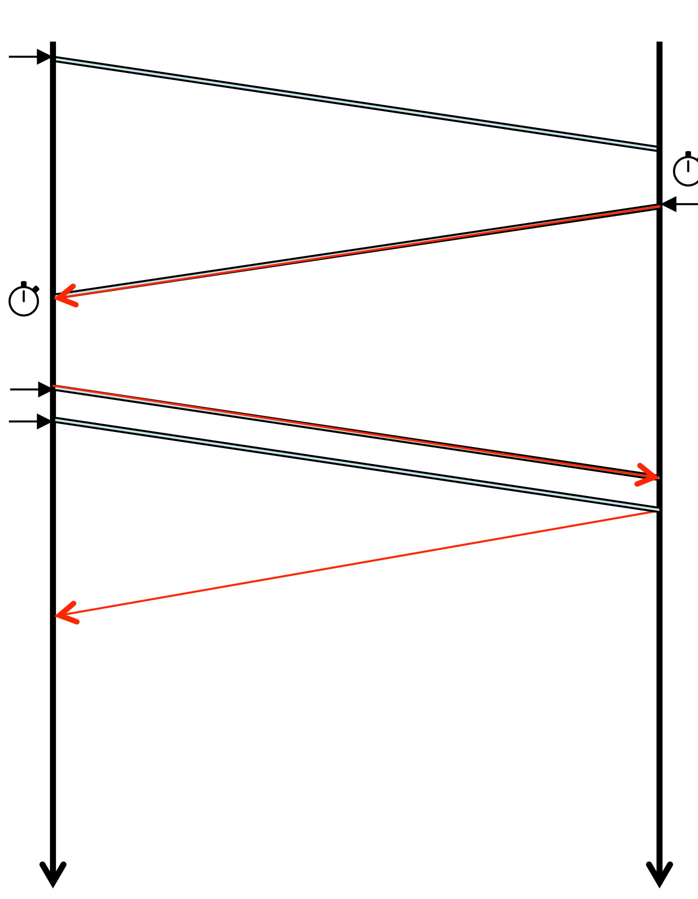
Delayed ACK

- Se envía un ACK por cada paquete de datos
- Los ACK son acumulativos, uno posterior repite la confirmación del anterior, luego no hacen falta tantos
- (...)



Delayed ACK

- Con *delayed ACK* al recibir datos espera antes de mandar ACK
- Así tal vez pueda hacerles *piggybacking* con datos en sentido contrario
- O reciba más datos y confirme todos de una vez
- Confirma si recibe 2xRMSS bytes
- O si caduca un timer
- Timer debe ser < 500ms
- Típico 200ms o dinámico
- (Implementaciones con valores mucho menores)



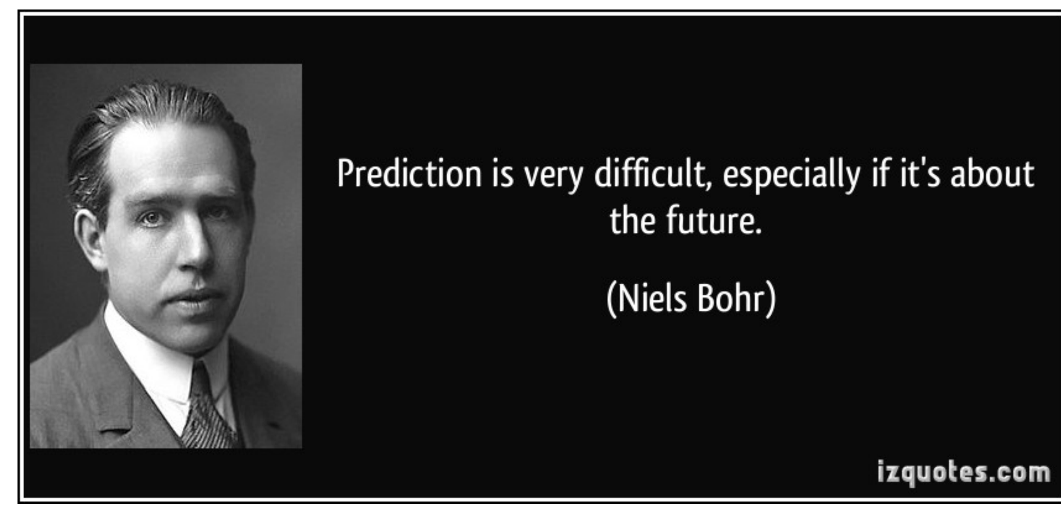
Retransmission Timer

RTT para calcular RTO

- *Retransmission Timeout*
- Esquema ARQ (Automatic Repeat Request)
- Tras un tiempo sin recibir confirmación se retransmite
- ¿Valor de ese tiempo?
- Si sobre-estimamos tarda en reconocer la pérdida
  - Tarda en retransmitir
  - Tarda en darse cuenta de que puede haber congestión
- (...)

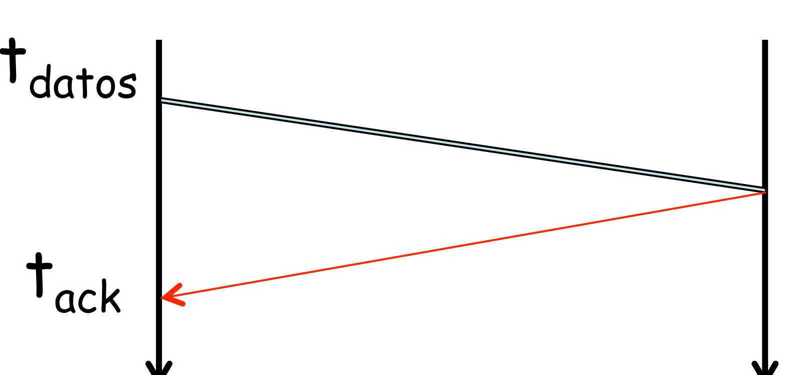
RTT para calcular RTO

- Si sub-estimamos retransmite sin necesidad y considera que hay congestión cuando no la hay
- Idealmente debería ser un RTT o algo relacionado
- Cuidado que el RTT medido es “viejo”
- Se hace una predicción



RTO

- Retransmission TimeOut
- Timeout es el último recurso pues vacía la red de segmentos
- Se basa en estimar el RTT con el tiempo entre datos y ACK
- Incorporarlo a medidas anteriores mediante un EWMA (*Exponentially Weighted Moving Average*)
- Y aumentar ese valor por un factor
- Bastantes cambios desde la RFC 793
- (...)



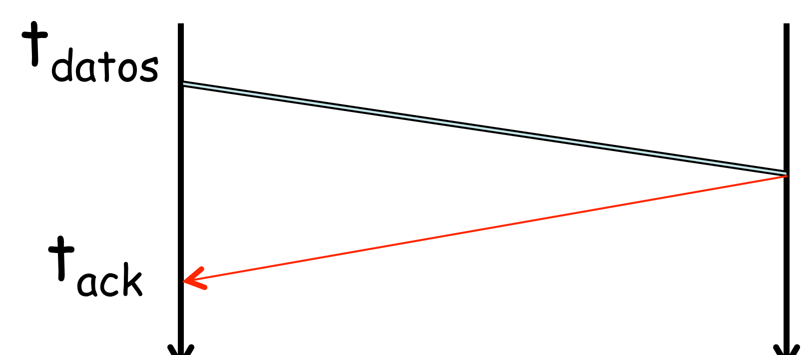
RFC 793

$$t_{ack} - t_{datos} = RTT$$
$$SRTT' = \alpha\ SRTT + (1-\alpha)\ RTT$$
$$RTO = \beta\ SRTT'$$

α = 0.8 a 0.9 (*smoothing factor*)
β = 1.3 a 2 (*delay variance factor*)
SRTT = “RTT suavizado”

RTO

- Acotado por un mínimo y máximo (típico entre 1s y 4min)
- Inicio (SYNs) RTO = 3s
- Al expirar: retx y RTO'=2xRTT (exponential backoff)



RFC 793

$$t_{ack} - t_{datos} = RTT$$

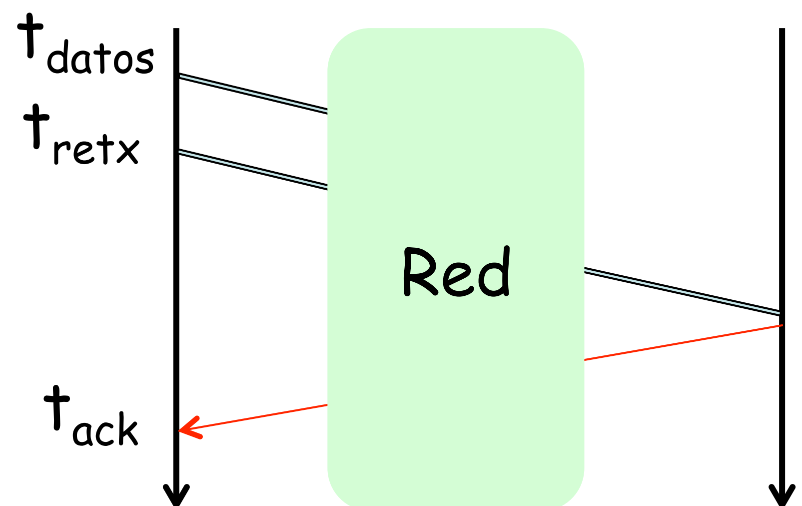
$$SRTT' = \alpha\ SRTT + (1-\alpha)\ RTT$$

$$RTO = \beta\ SRTT'$$

α = 0.8 a 0.9 (*smoothing factor*)
β = 1.3 a 2 (*delay variance factor*)
SRTT = “RTT suavizado”

RTO: Algoritmo de Karn

- Si hay retransmisión no se sabe a qué paquete corresponde el ACK
- El timer con *backoff* debido a la retransmisión se emplea para la próxima transmisión
- Se propone ignorar medida de RTT para paquetes que se han transmitido más de una vez
- Recalcular el RTO solo cuando llegue un ACK sin retransmisión
- Si el segmento TCP emplea la opción timestamp sí se recalcula
- Es un algoritmo obligatorio ("MUST" en RFC 6298)



Phil Karn

RTO: Nuevo cálculo

- V. Jacobson introduce estimación de la varianza
- Versión actual recomendada: RFC 6298 “Computing TCP’s Retransmission Timer” (2011)
- Inicialmente RTO = 1s (SHOULD)
- Ante la primera medida de RTT:
  - SRTT = RTT
  - RTTVAR’ = RTT / 2
  - RTO = SRTT + 4 RTTVAR’ = RTT + 2 RTT = 3 RTT
- RTO mínimo de 1s (SHOULD, hay implementaciones a 200ms)
- Puede haber RTO máximo si es de al menos 60s
- Emplea algoritmo de Karn
- Si 4xRTTVAR’=0 usar la resolución del reloj empleado en timer

$$RTTVAR' = (1-\beta)\ RTTVAR + \beta\ (SRTT - RTT)$$

$$SRTT' = (1-\alpha)\ SRTT + \alpha\ RTT$$

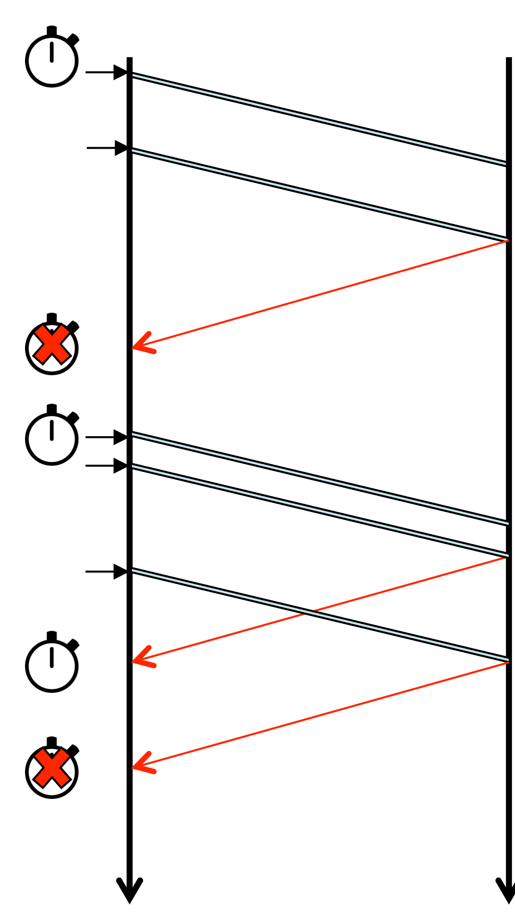
$$RTO = SRTT' + 4\ RTTVAR'$$

Ojo, en este orden

$\alpha = 1/8$
$\beta = 1/4$

RTO: ¿Por paquete?

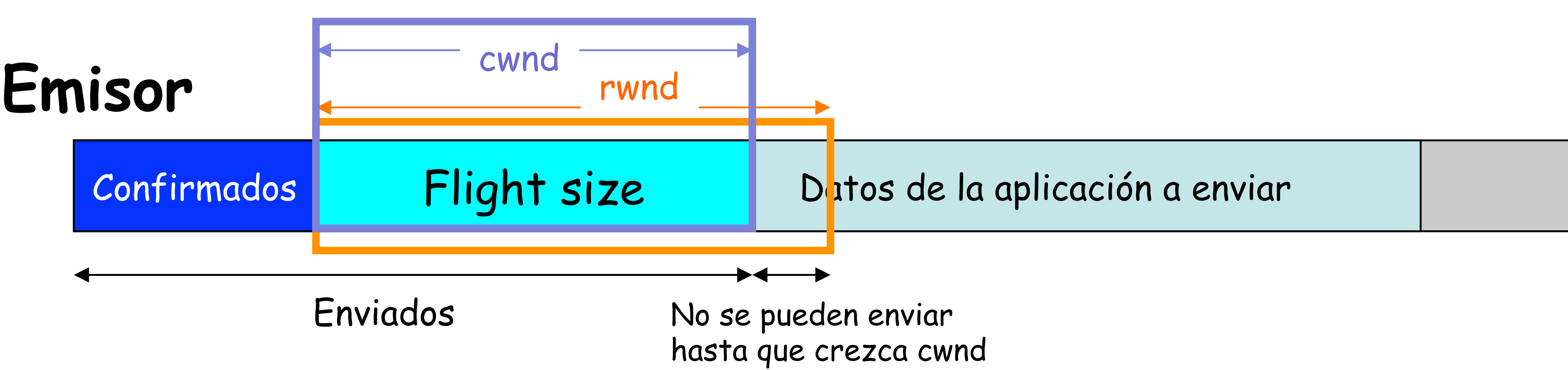
- Tradicionalmente las implementaciones de TCP toman una sola medida de RTT a la vez (es decir, una por RTT)
- Deben tomar al menos una por RTT (si lo permite el algoritmo de Karn) aunque con ventanas grandes se recomiendan más
- Para ventanas no muy grandes no mejora la estimación del RTT hacerla por paquete
- Recomendación para un solo timer (RFC 6298)
  - Si se envía un paquete de datos (nuevos o retransmitidos) y no hay timer activo activarlo
  - Cuando se confirman todos los datos enviados desactivar timer
  - Cuando un ACK confirma datos reiniciar el timer con nuevo RTO



Slow Start

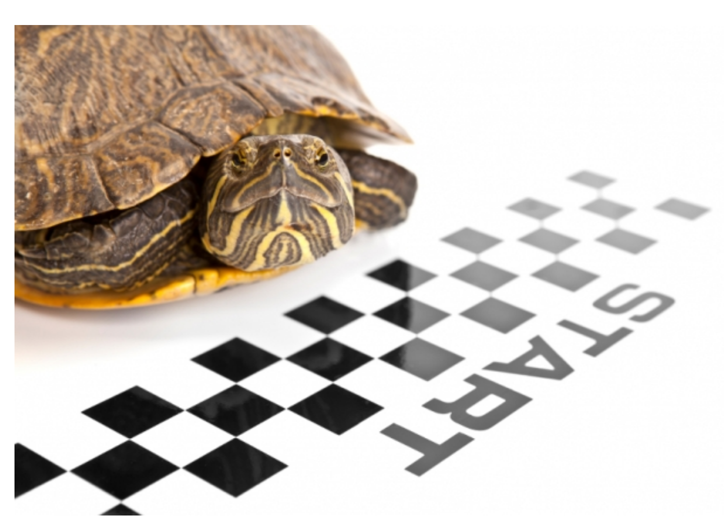
Control de congestión en TCP

- Última versión en RFC 5681 "TCP Congestion Control" (año 2009)
- Fundamentalmente 4 algoritmos:
  - Slow start
  - Congestion avoidance
  - Fast retransmit
  - Fast recovery
- Se puede ser más conservador pero no más agresivo enviando
- Los datos enviados sin confirmar (*flight size*) deben ser menos del mínimo entre:
  - *rwnd* : ventana de control de flujo anunciada por receptor
  - *cwnd* : ventana de congestión calculada por el emisor
- *ssthresh* : umbral entre slow start y congestion avoidance



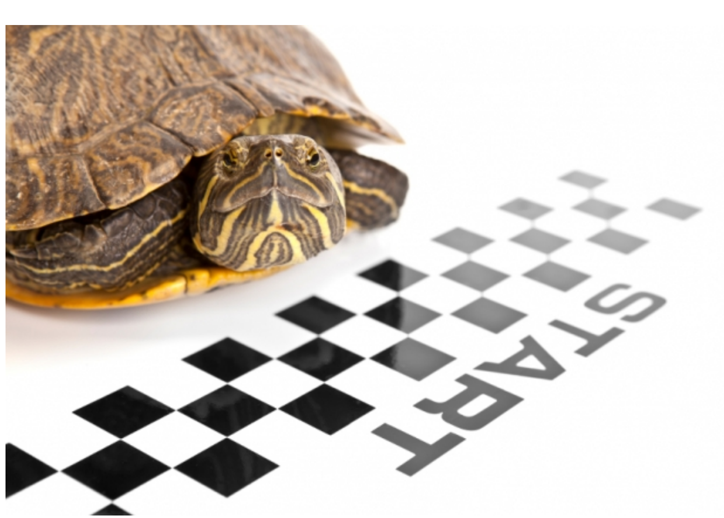
Slow start

- Inicialmente TCP no conoce las condiciones de la red
- No tiene el *self-clocking* que dan las confirmaciones
- Comienza enviando en modo muy conservador (1 paquete?)
- Si ve que los datos llegan (recibe ACK) prueba a enviar más de golpe (tener más “en vuelo”)
- De esa forma va consiguiendo ese reloj



Slow start

- IW = *Initial Window*, valor inicial máximo de cwnd (MUST)
  - Tradicionalmente IW = 1 ó 2 segmentos pero cambiado en 2002:
    - Si (SMSS > 2190) : IW = 2xSMSS y no más de 2 segmentos
    - Si (2190 >= SMSS > 1095) : IW = 3xSMSS y no más de 3 sgmts.
    - Si (1095 >= SMSS) : IW = 4xSMSS y no más de 4 segmentos
  - Con Ethernet end2end SMSS puede ser 1460 ➔ IW=3 segmentos
  - Evita un *delayed ack* con un primer segmento único
- ssthresh comienza en valor muy grande



Gestión y Planif. Redes y Servs.
Área de Ingeniería Telemática

Slow start

- TCP incrementa cwnd en como mucho el SMSS por cada ACK que confirma nuevos datos
- Hasta que alcance/supere ssthresh o se detecte una pérdida
- Tradicionalmente se incrementaba en SMSS
- Ahora se recomienda incrementar en mín(bytes_ack'ed,SMSS)
- Esto protege contra "ACK Division"
- RFC 6928 (Google, experimental, 2013) propone IW=10

Ejemplo de *slow start*

```
1.1.1.11.2823 > 1.1.1.12.1509: S 0:0(0) win 32120 <mss 1460>
1.1.1.12.1509 > 1.1.1.11.2823: S 0:0(0) ack 1 win 32120  <mss 1460>
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 1 win 32120

1.1.1.12.1509 > 1.1.1.11.2823: P 1:1449(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 1449:2897(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 1449 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 2897:4345(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 4345:5793(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 2897 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 5793:7241(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 7241:8689(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 5793 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 8689:10137(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 10137:11585(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 11585:13033(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 8689 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 13033:14481(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 14481:15929(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 15929:17377(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 10137 win 31856
```

Ejemplo de *slow start*

```
1.1.1.12.1509 > 1.1.1.11.2823: P 17377:18825(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 18825:20001(1176) ack 1 win 32120
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 13033 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 15929 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 18825 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 20001 win 31856

1.1.1.12.1509 > 1.1.1.11.2823: P 20001:21449(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 21449:22897(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 22897:24345(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 24345:25793(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 25793:27241(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 27241:28689(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 28689:30137(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 30137:31585(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 31585:33033(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 33033:34481(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 34481:35929(1448) ack 1 win 32120
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 22897 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: . 35929:37377(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: . 37377:38825(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: . 38825:40273(1448) ack 1 win 32120
```

cwnd

8
9
10
11
12