TCP: Control de congestión

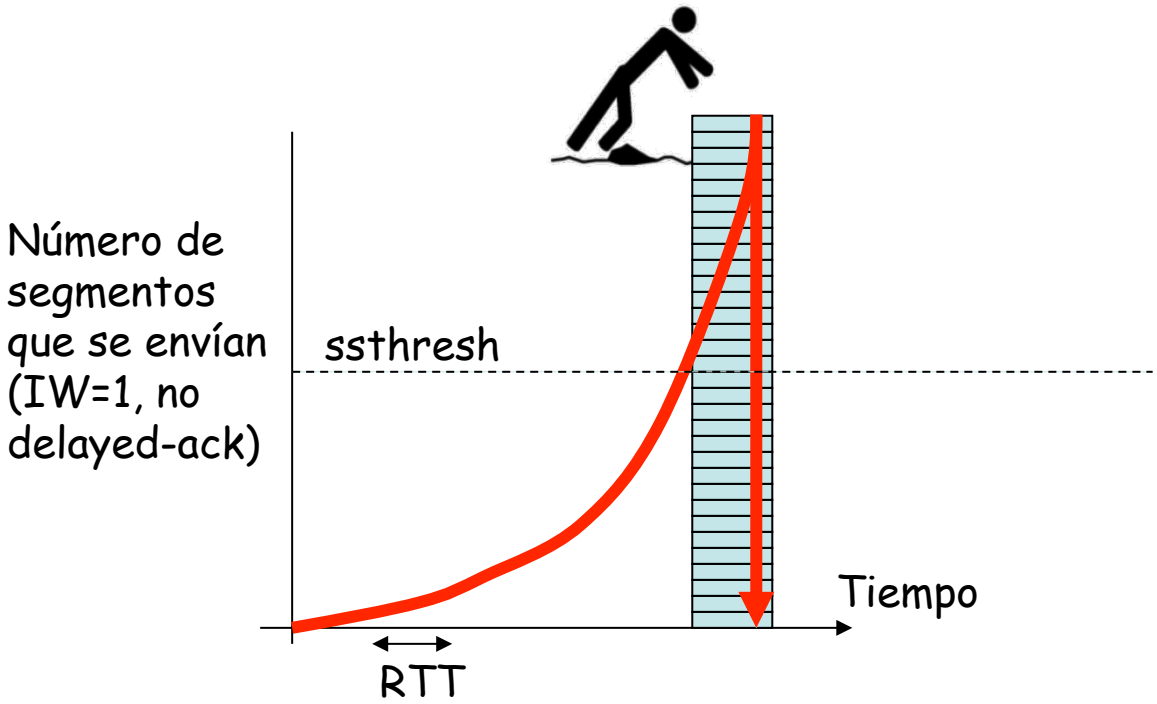
Area de Ingeniería Telemática
http://www.tlm.unavarra.es

Grado en Ingeniería en Tecnologías de Telecomunicación, 4º

TCP Tahoe

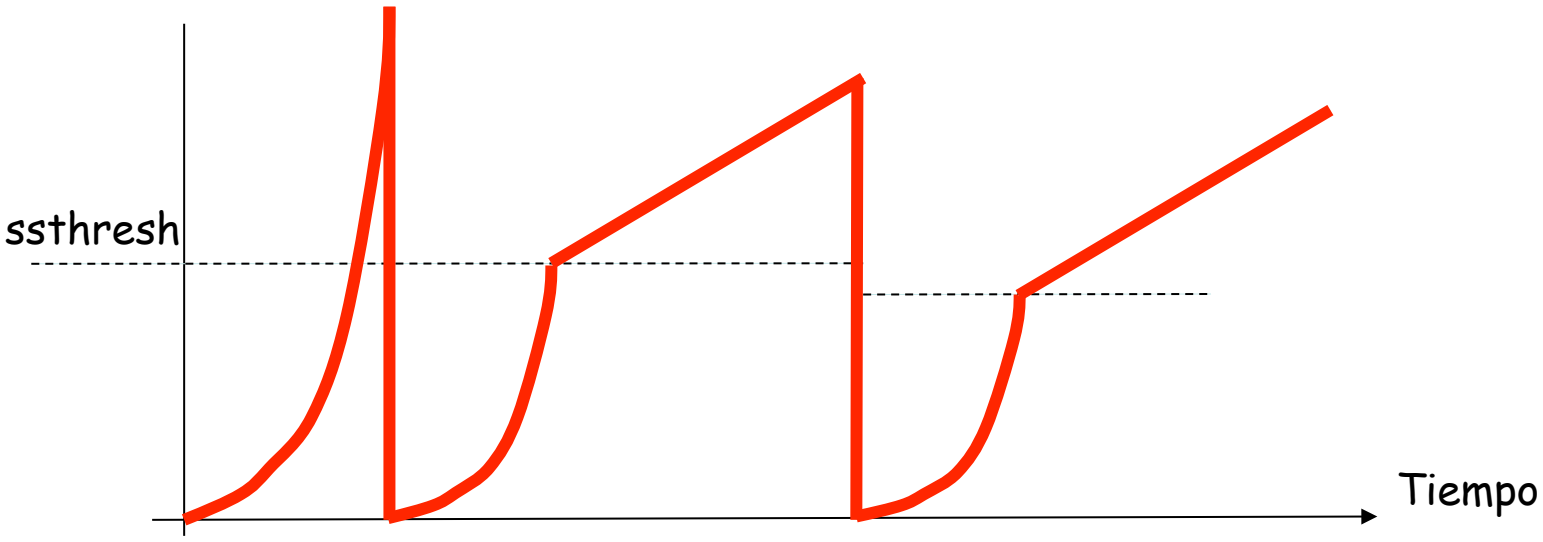
Pérdidas por timeout

- Al detectar pérdidas por timeout:
  - cwnd = SMSS (independiente de IW)
  - Si los datos no se había retransmitido ya por timeout:
    ssthresh = máx(FlightSize / 2, 2xSMSS)
  - Si ya se habían retransmitido no se cambia ssthresh



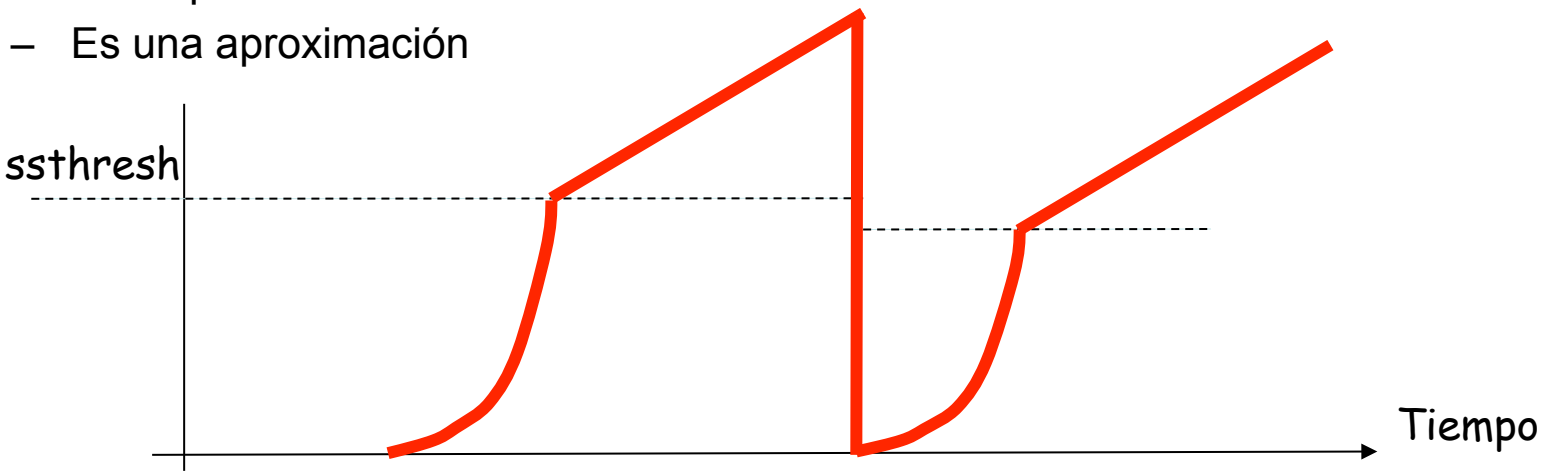
Congestion avoidance

- Cuando cwnd es mayor que ssthresh
- Se incrementa cwnd en 1 SMSS por cada RTT o en mín(bytes_ack'ed,SMSS)



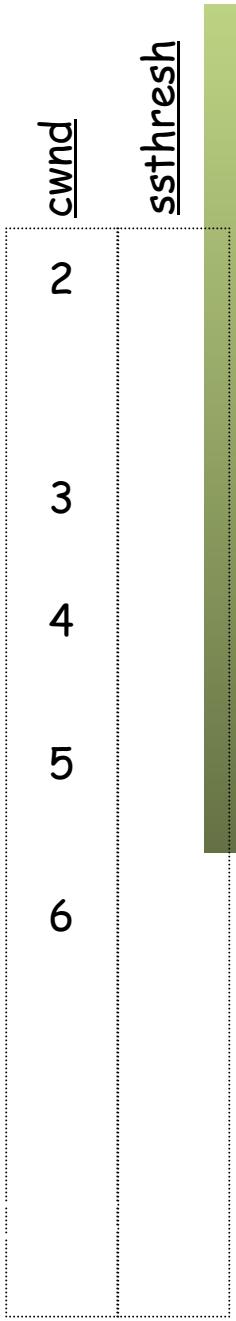
Congestion avoidance

- Cuando cwnd es mayor que ssthresh
- Se incrementa cwnd en 1 SMSS por cada RTT o en mín(bytes_ack'ed,SMSS)
- **Opción 1**
  - Contar los bytes confirmados
  - Cuando alcanza el valor de cwnd se incrementa la ventana
  - Requiere un contador adicional
- **Opción 2**
  - Con cada ACK que confirma nuevos datos actualizar
    cwnd += SMSS x SMSS / cwnd
  - No requiere contador adicional
  - Es una aproximación



Ejemplo Tahoe (I)

```
753.246362 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: S 0:0(0) win 32120
753.246536 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: S 0:0(0) ack 1

753.468594 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1:205(204) ack 1
753.468750 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 205:409(204) ack 1

754.291021 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 205
754.291137 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 409:613(204) ack 1
754.291257 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 613:817(204) ack 1
754.746127 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 409
754.746234 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 817:1021(204) ack 1
754.746353 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1021:1225(204) ack 1
755.832827 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 817
755.832948 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1225:1429(204) ack 1
755.833066 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1429:1633(204) ack 1
755.833182 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1633:1837(204) ack 1
756.327987 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1225
756.328105 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1837:2041(204) ack 1
756.328220 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2041:2245(204) ack 1
756.328333 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2245:2449(204) ack 1

(...)
```



Ejemplo Tahoe (I)

```
753.246362 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: S 0:0(0) win 32120
753.246536 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: S 0:0(0) ack 1

753.468594 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1:205(204) ack 1
753.468750 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 205:409(204) ack 1

754.291021 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 205
754.291137 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 409:613(204) ack 1
754.291257 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 613:817(204) ack 1
754.746127 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 409
754.746234 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 817:1021(204) ack 1
754.746353 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1021:1225(204) ack 1
755.832827 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 817
755.832948 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1225:1429(204) ack 1
755.833066 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1429:1633(204) ack 1
755.833182 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1633:1837(204) ack 1
756.327987 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1225
756.328105 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1837:2041(204) ack 1
756.328220 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2041:2245(204) ack 1
756.328333 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2245:2449(204) ack 1

760.697798 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1225:1429(204) ack 1

761.796804 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1429
```

cwnd	ssthresh
2	
3	
4	
5	
6	
1	3
2	

Ejemplo Tahoe (II)

```
761.796932 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1429:1633(204) ack 1
761.797054 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1633:1837(204) ack 1
762.838579 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1837

762.838691 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1837:2041(204) ack 1
762.838807 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2041:2245(204) ack 1
762.838923 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2245:2449(204) ack 1

764.101234 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2245

764.827096 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2449

764.827200 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2449:2653(204) ack 1
764.827315 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2653:2857(204) ack 1
764.827428 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2857:3061(204) ack 1
764.827541 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3061:3265(204) ack 1
766.320277 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2857

766.320406 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3265:3469(204) ack 1
766.320523 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3469:3673(204) ack 1
766.758919 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 3265

766.759024 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3673:3877(204) ack 1
766.759141 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3877:4081(204) ack 1
767.859805 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 3673
```

cwnd	ssthresh
3	3
4	
4 (1)	
4 (2)	
4 (3)	
4 (4)	

Ejemplo Tahoe (III)

```
767.859926 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4081:4285(204) ack 1
767.860043 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4285:4489(204) ack 1
768.359065 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4081

768.359188 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4489:4693(204) ack 1
768.359305 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4693:4897(204) ack 1
768.359418 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4897:5101(204) ack 1
768.899165 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4489

768.899311 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5101:5305(204) ack 1
768.899429 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5305:5509(204) ack 1
770.122698 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4897

770.122858 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5509:5713(204) ack 1
770.122975 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5713:5917(204) ack 1
770.614382 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5305

770.614528 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5917:6121(204) ack 1
770.614644 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6121:6325(204) ack 1
771.347724 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5509

771.347856 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6325:6529(204) ack 1
771.659393 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5917
```

cwnd	ssthresh
	3
5	
5 (1)	
5 (2)	
5 (3)	
5 (4)	
5 (5)	

Ejemplo Tahoe (y IV)

```
771.659497 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6529:6733(204) ack 1
771.659613 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6733:6937(204) ack 1
772.159445 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6325

772.159599 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6937:7141(204) ack 1
772.159715 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7141:7345(204) ack 1
772.159829 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7345:7549(204) ack 1
772.699491 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6733

772.699660 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7549:7753(204) ack 1
772.699781 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7753:7957(204) ack 1
1773.397932 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6937

773.398089 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7957:8161(204) ack 1
773.919587 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 7345

773.919734 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8161:8365(204) ack 1
773.919851 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8365:8569(204) ack 1
774.409619 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 7753

774.409770 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8569:8773(204) ack 1
774.409886 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8773:8977(204) ack 1
774.909675 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 8161
```

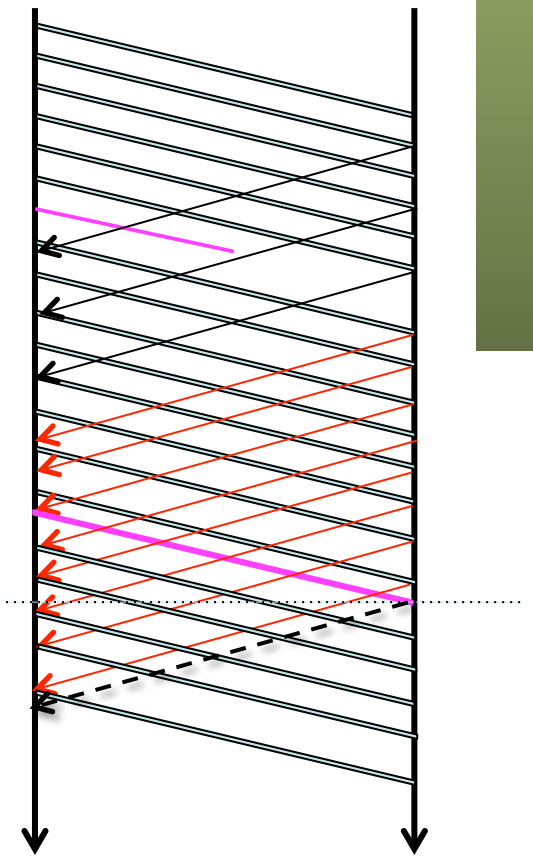
cwnd	ssthresh
	3
6	
6 (1)	
6 (2)	
6 (3)	
6 (4)	
6 (5)	

TCP Reno

Fast recovery

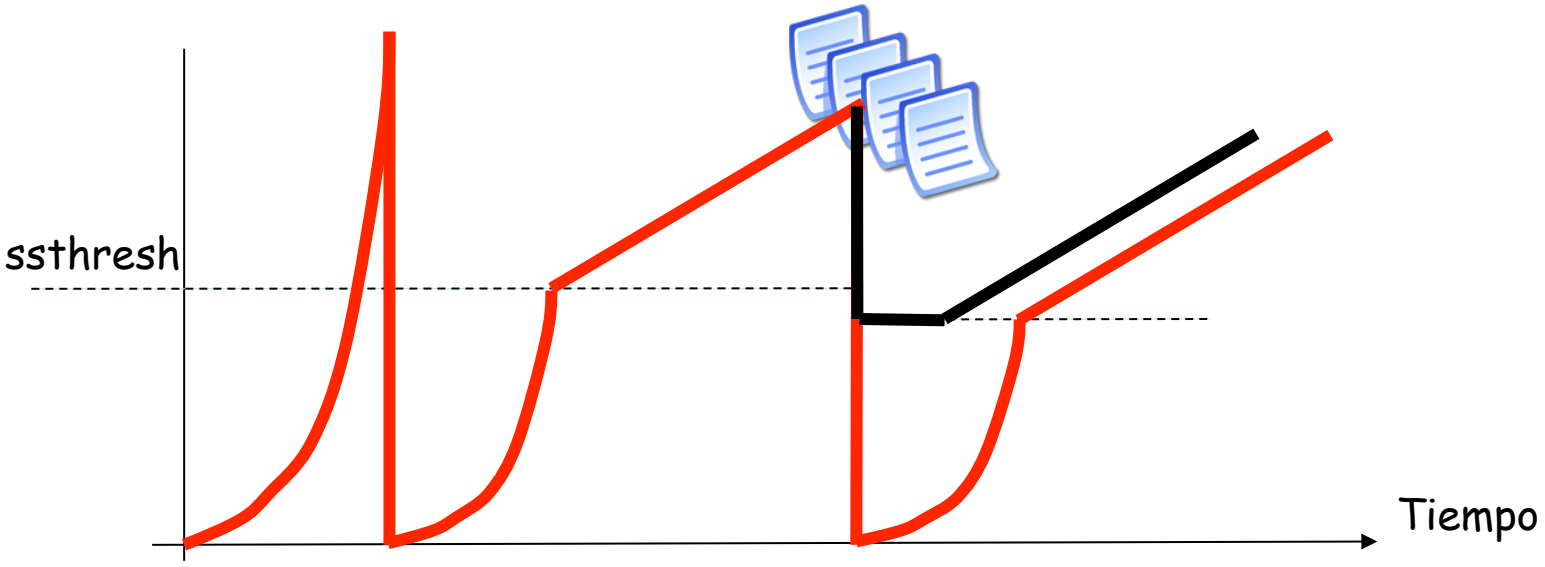
- Pueden seguir llegando ACKs duplicados según el *FlightSize*
- No pasa a slow start pues el que lleguen los duplicados implica que los paquetes están llegando al destino
- Seguimos teniendo el "reloj" que dan los ACKs, puede seguir transmitiendo
- Se reduce ssthresh a la mitad

  ssthresh = máx(FlightSize / 2, 2xSMSS)
- cwnd = ssthresh + 3xSMSS
- Esto añade a la ventana los 3 segmentos que han debido llegar (eso dicen los dup ACKs)
- Por cada nuevo ACK duplicado se incrementa cwnd += SMSS
- Esto puede permitir un envío con nuevos datos
- Cuando llegue el segmento perdido se genera confirmación para todo lo recibido
- Finalmente llegará ACK para nuevos datos
- Pone cwnd = ssthresh (*deflating* la ventana)
- Debería confirmar todo lo recibido tras el hueco



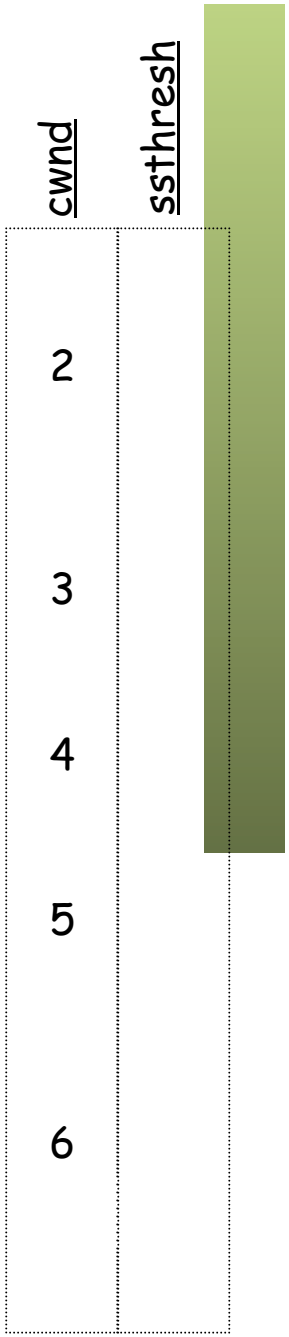
Fast retransmit/recovery

- Si durante *fast recovery* caduca el timer de retransmisión se pasa a slow start
- Si la pérdida es al final de un bloque de datos y no hay más para enviar no se pueden generar los dup-ACKs y caduca el timer



Ejemplo Reno (I)

```
1.1.1.15.1069 > 10.1.8.251.5000: S 0:0(0) win 32120
10.1.8.251.5000 > 1.1.1.15.1069: S 0:0(0) ack 1 win 31680 (= 33MSS)
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 1 win 32120


10.1.8.251.5000 > 1.1.1.15.1069: P 1:949(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 949:1897(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 949 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: P 1897:2845(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 2845:3793(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 1897 win 32232

10.1.8.251.5000 > 1.1.1.15.1069: P 3793:4741(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 4741:5689(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 3793 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: P 5689:6637(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 6637:7585(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 7585:8533(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 5689 win 31284

...
```



Ejemplo Reno (II)

```
10.1.8.251.5000 > 1.1.1.15.1069: P 75841:76789(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 76789:77737(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 77737:78685(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: . 78685:79633(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 79633:80581(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 80581:81529(948) ack 1 win 3223
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284   <- DUP 1

1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284   <- DUP 2

1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284   <- DUP 3
10.1.8.251.5000 > 1.1.1.15.1069: P 52141:53089(948) ack 1 win 32232

1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284
```

FastReTx

cwnd	ssthresh
31	
	15
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	

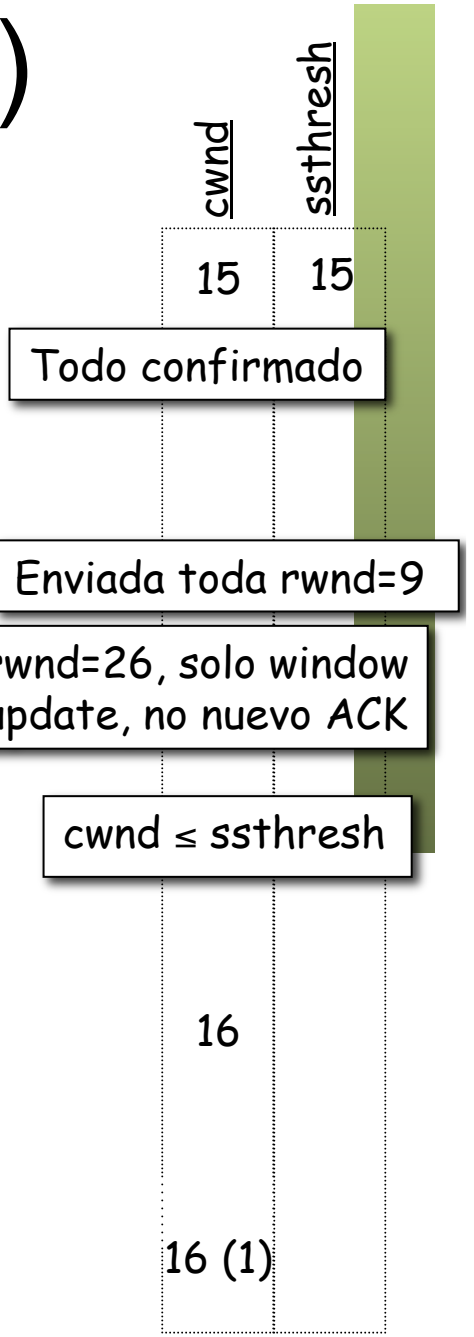
Ejemplo Reno (III)

Trace	cwnd	ssthresh
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	28	15
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	29	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	30	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	31	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	32	
10.1.8.251.5000 > 1.1.1.15.1069: . 81529:82477(948) ack 1 win 32232		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284 (rwnd = 33)	33	
10.1.8.251.5000 > 1.1.1.15.1069: . 82477:83425(948) ack 1 win 32232		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284 (rwnd<cwnd)	34	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	35	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	36	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	37	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	38	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	39	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	40	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	41	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	42	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	43	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	44	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 81529 win 8532 (rwnd = 9)	**15**	

Fin de fast recovery

Ejemplo Reno (y IV)

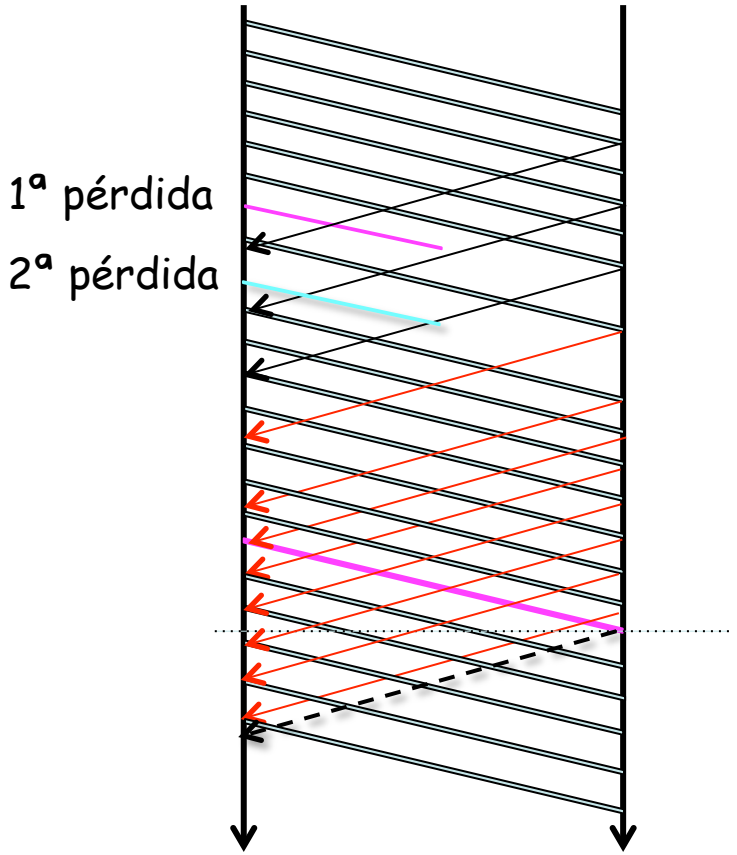
```
10.1.8.251.5000 > 1.1.1.15.1069: . 83425:84373(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 84373:85321(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 85321:86269(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 86269:87217(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 87217:88165(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 88165:89113(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 89113:90061(948) ack 1 win 32232

1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 81529 win 24648

10.1.8.251.5000 > 1.1.1.15.1069: . 90061:91009(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 91009:91957(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 91957:92905(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 92905:93853(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 93853:94801(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 94801:95749(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 83425 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: . 95749:96697(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 96697:97645(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 97645:98593(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 85321 win 31284

...
```

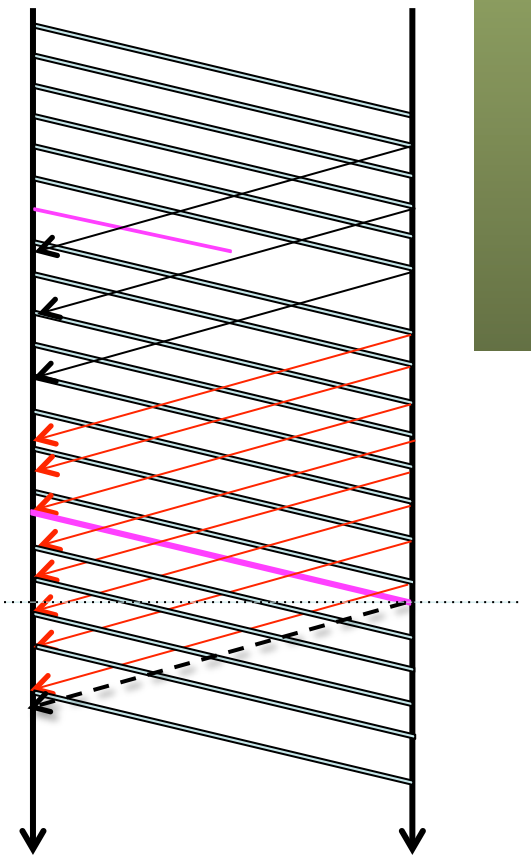


TCP New Reno

Pérdidas múltiples

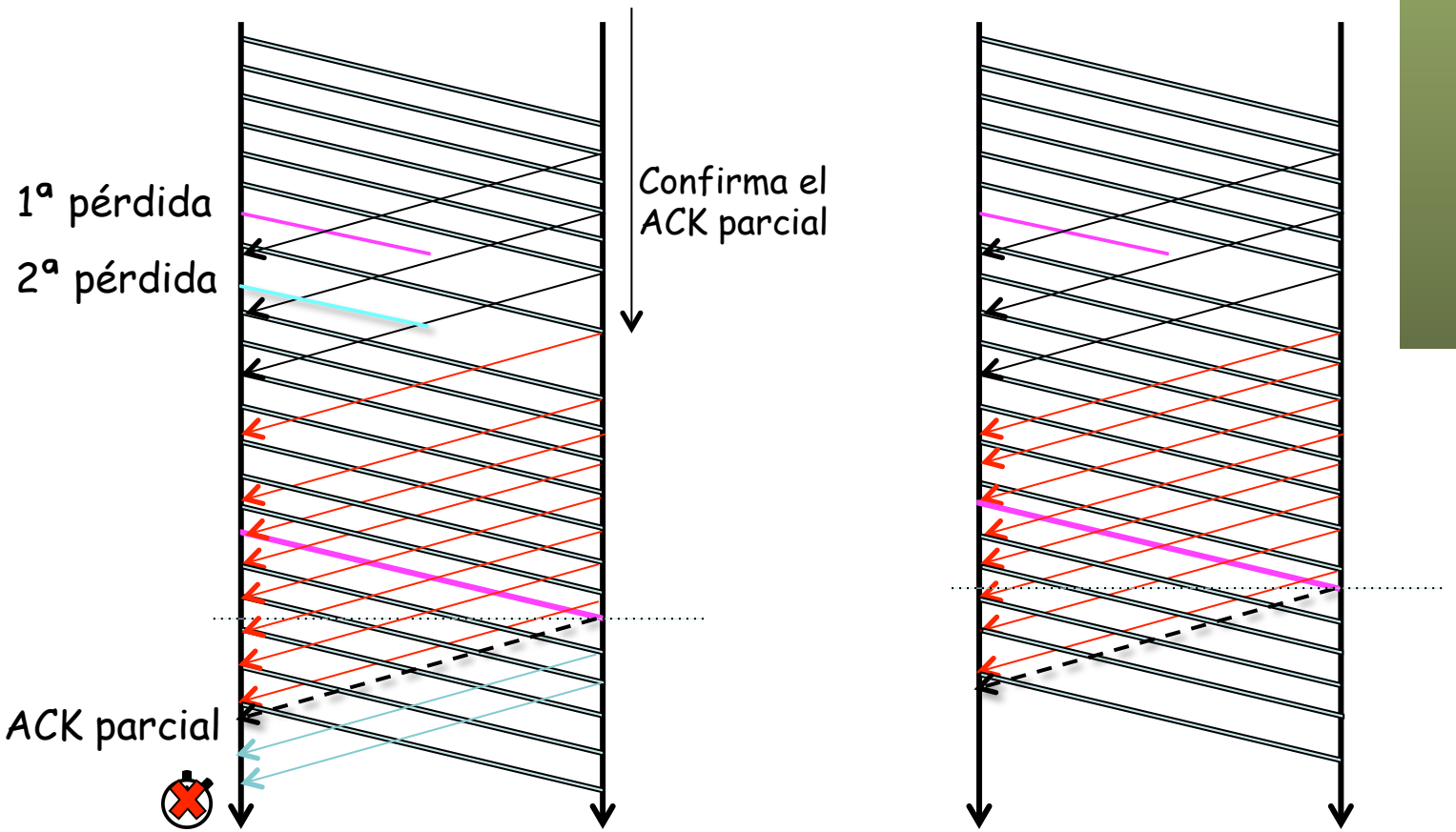
- Fast retransmit/fast recovery no recupera tan bien pérdidas múltiples en el *FlightSize* dado que solo retransmite 1 segmento
- En este caso por ejemplo se pierden 2
- Ahora el tercer dup ACK es más tarde (falta el ACK del 2º perdido)
- (…)





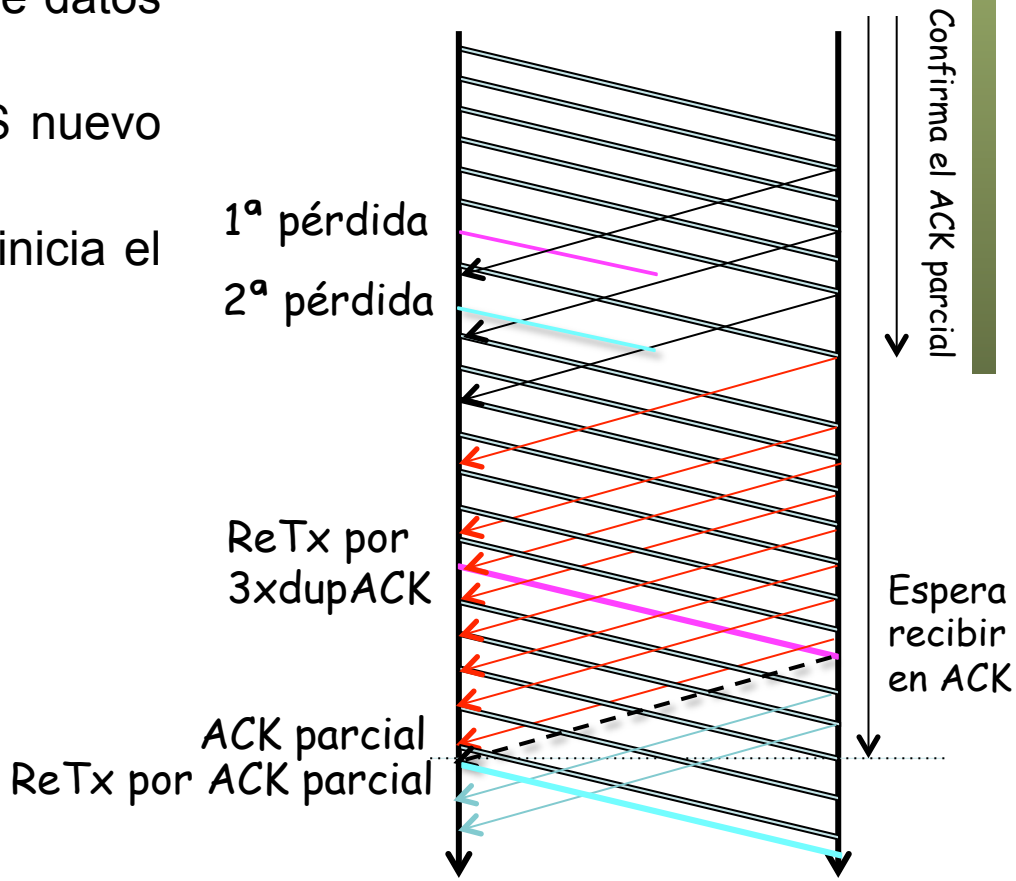
Pérdidas múltiples

- El ACK que llega al llenarse el hueco es un **ACK *parcial***
- La secuencia está interrumpida por la segunda pérdida
- Deberían llegar nuevos *dup* ACKs para reactivar *fast retransmit*
- Pero es improbable que haya tanto *FlightSize* y suele caducar el timer



NewReno

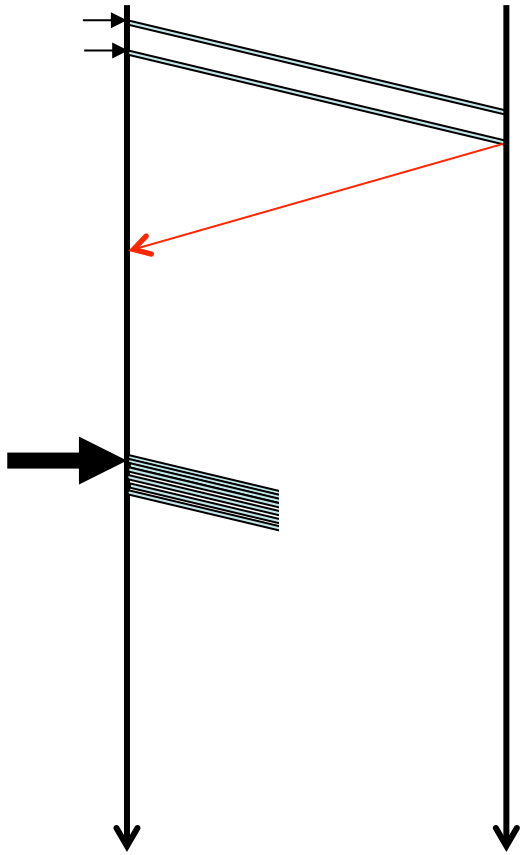
- Modifica *Fast recovery* (RFC 6582, año 2012, 1ª RFC 1999)
- Con el 3º dup-ACK apunta número de secuencia máximo enviado (hasta ahí debería recibir confirmación)
- Al recibir un **ACK parcial** retransmite el primer segmento sin confirmar
- Reduce cwnd en la cantidad de datos confirmados
- Si confirma al menos 1 SMSS nuevo incrementa cwnd en 1 SMSS
- Si es el primer ACK parcial reinicia el timer de retransmisión
- Sigue en fast recovery



Conexiones inactivas

Restarting idle connections

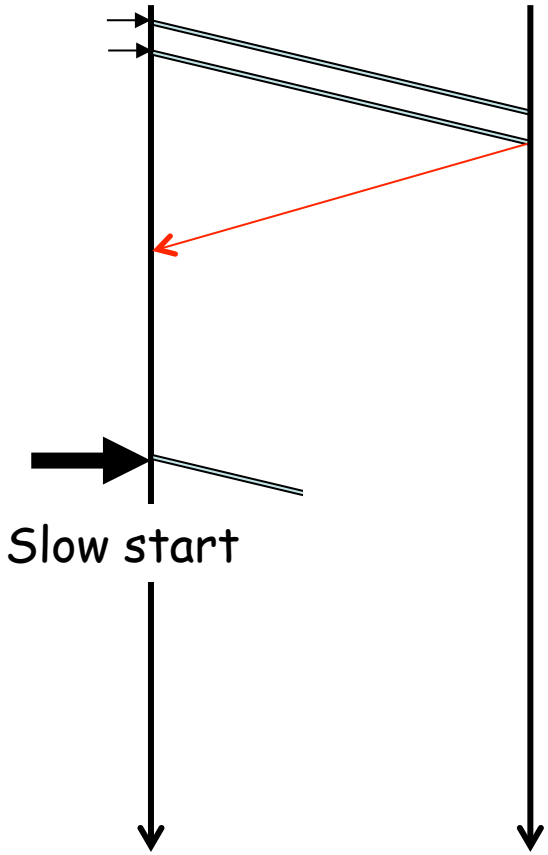
- ¿Qué sucede si el flujo queda inactivo un tiempo (no hay datos a enviar)?
- Ha perdido el reloj que dan los ACKs
- Puede tener una cwnd grande
- Si llega una gran cantidad de datos se pueden enviar en ráfaga
- Se supone que esos paquetes “caben” en la red (en buffers) pues lo dice cwnd
- Pero pueden haber cambiado las condiciones de la red
- Ahora cwnd puede ser una mala estimación
- Y de todos modos era una estimación de buffer para todo el trayecto
- Estamos soltando esos paquetes en ráfaga (*line rate*) al primer conmutador



Restarting idle connections

- La recomendación es volver a slow start tras un periodo largo de inactividad
- RFC 5681 sugiere hacerlo cuando TCP no ha recibido un segmento durante más de un RTO
- O cuando no ha tenido datos para transmitir durante al menos un RTO
- Se reduce cwnd a

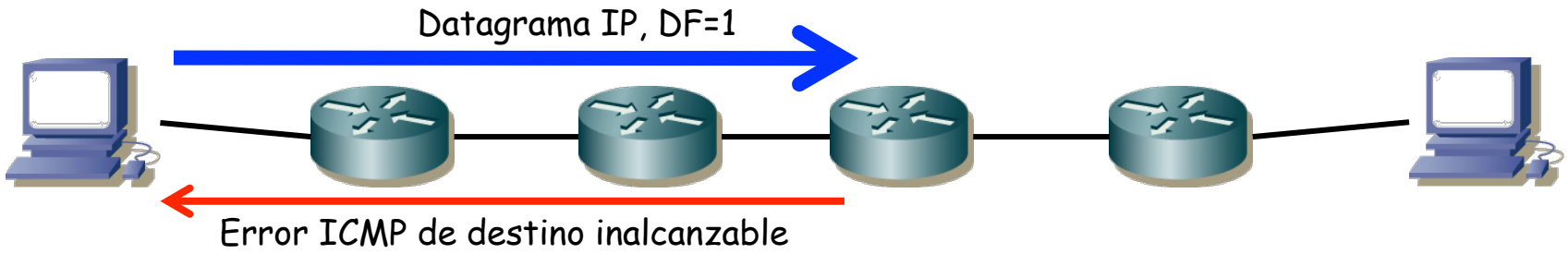
  cwnd' = RW = mín(IW, cwnd)
- No es obligatorio implementarlo



PMTUD

PMTU Discovery

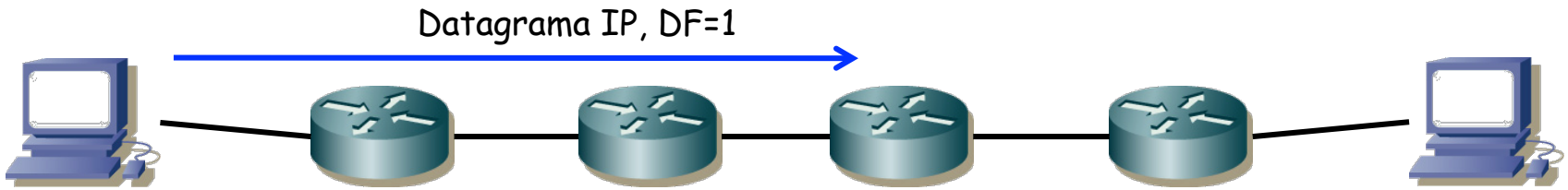
- RFC 1191 “Path MTU Discovery” (año 1990)
- Pretende averiguar la menor MTU en un trayecto
- Esto permitiría enviar datagramas IP del tamaño máximo que no se fragmenten
- Para cualquier protocolo aunque lo suele emplear solo TCP
- El host asume que la PMTU es la MTU de su enlace
- Envía un datagrama de ese tamaño con DF=1
- Si es demasiado grande será descartado y se recibirá un ICMP de destino inalcanzable (“fragmentation needed and DF set”)
- Ese error no notifica de la MTU (en IPv6 sí hay uno que lo hace) aunque se proponen modificaciones para que sí lo haga



PMTU Discovery

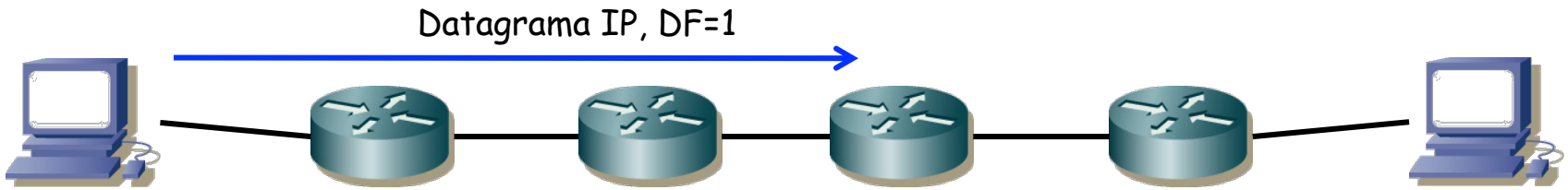
- Host origen reduce su estimación de PMTU y reintenta
- Se recomiendan unos valores de MTU típicos para la reducción
- (...)

MTU aprox.	¿Por qué?
65535	Máximo
32000	
17914	IBM Token Ring 16Mbps
8166	IEEE 802.4
4352	IEEE 802.5 4464 máximo, FDDI 4352
2002	IEEE 802.5 (recomendado)
1492	Ethernet 1500, 802.3 1492
1006	SLIP, ARPANET
508	X.25 576, NETBIOS 512
296	PPP low delay
68	Mínimo



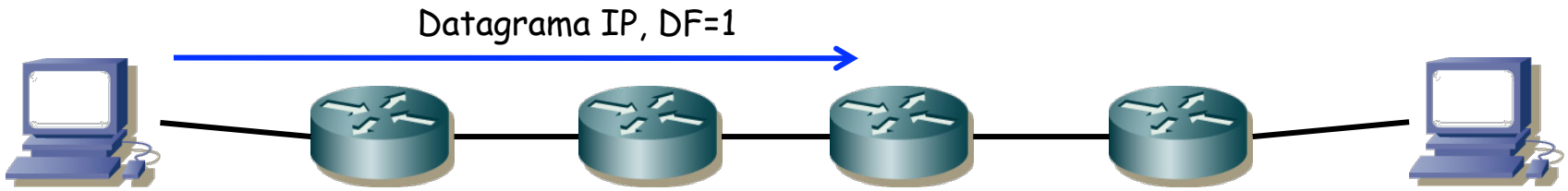
PMTU Discovery

- La PMTU permite calcular un MSS para las conexiones TCP
- La PMTU se puede guardar en una cache de la tabla de rutas que tenga una entrada por destino (la tabla de rutas no la tiene)
- De hecho es lo que se hace con los ICMP Redirect
- (...)



PMTU Discovery

- Aunque averigüe una MTU que le valga mantiene DF=1 por si hay cambios de ruta que la hayan reducido
- Para reconocer cambios de ruta que aumenten la PMTU puede reiniciar periódicamente su estimación de la PMTU
- O puede incrementarla en base a la tabla
- Esto llevará normalmente a paquetes descartados así que se debe hacer infrecuentemente (>10min)



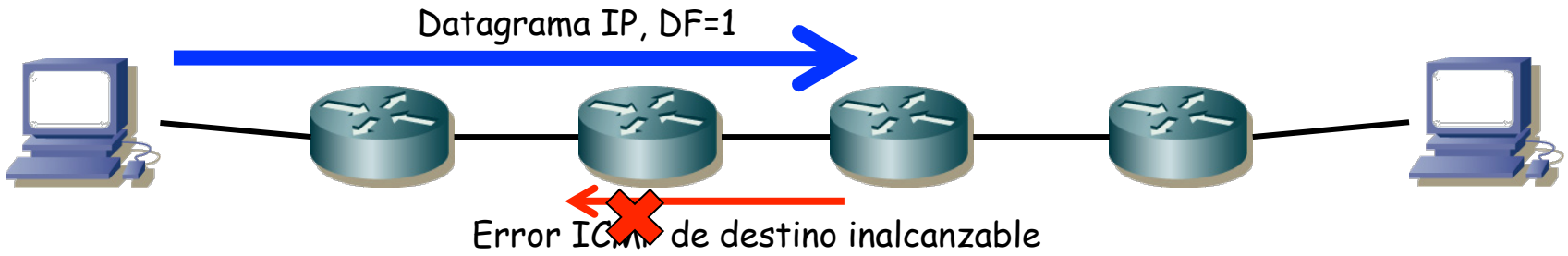
Problemas con PMTUD

- RFC 2923 “TCP Problems with Path MTU Discovery” (2000)
- (…)

Problemas con PMTUD

Black Hole Detection

- No se recibe el mensaje ICMP de destino inalcanzable
- Por bug en el router, problema de configuración, firewalls, etc
- El origen no reacciona, sigue mandando paquetes grandes que nunca llegan
- Difícil de depurar pues por ejemplo pings suelen funcionar
- Para resolverlo la RFC sugiere que si TCP detecta varios timeouts desactive el flag DF o envíe paquetes más pequeños



Problemas con PMTUD

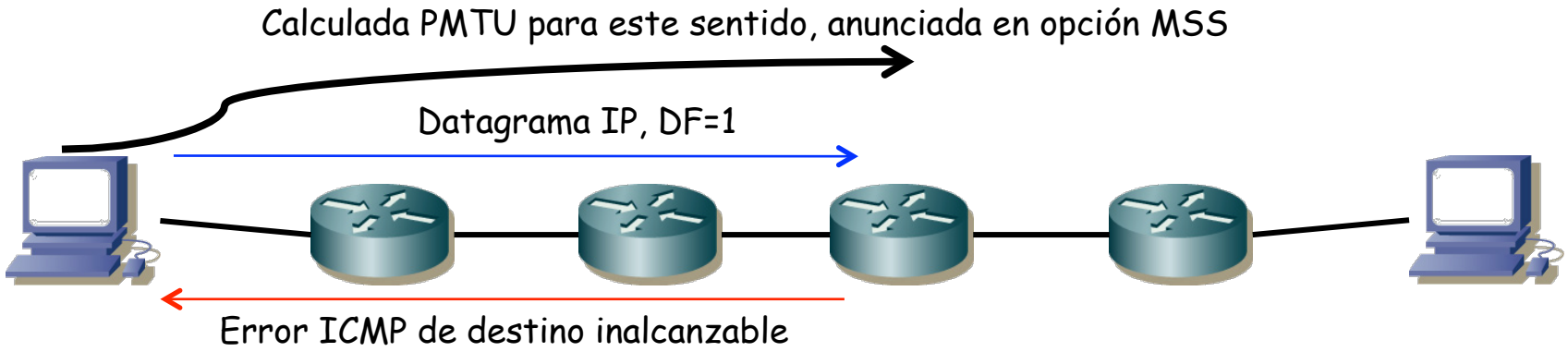
Stretch ACK

- Delayed-ACK lleva al menos a 1 ACK por cada dos segmentos completos
- ¿Cuál es el tamaño de esos “segmentos completos”?
- Puede que la implementación lo obtenga del anuncio de MSS que puede ser más grande que el PMTU
- Por ejemplo, que anuncie un MSS de 4312 bytes
- Y luego PMTU lo reduzca a 1460 bytes
- Debería generar 1 ACK al menos cada vez que acumule 2x1460 bytes
- Pero lo va a hacer cada 2x4312
- Muchos menos ACKs, crece más lento la cwnd y se dan ráfagas más grandes
- Soluciones:
  - Mandar siempre confirmación cada 2 paquetes aunque sean pequeños
  - Monitorizar el tamaño de los segmentos recibidos para determinar el tamaño que está empleando

Problemas con PMTUD

MSS a partir de PMTU

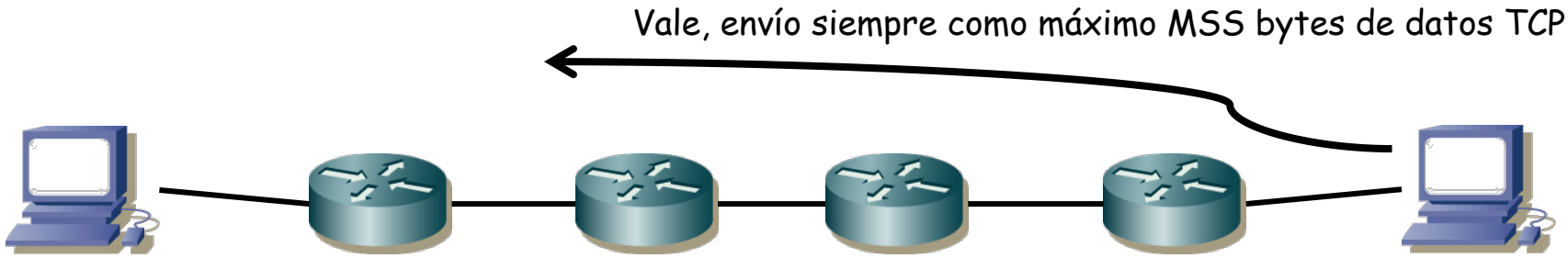
- Algunas implementaciones emplean el valor obtenido mediante PMTUD para decidir el MSS a anunciar
- PMTUD calcula la PMTU en un sentido pero el MSS se anuncia para el tráfico en el sentido contrario
- La opción MSS es del máximo que se desea "recibir"
- Si la ruta es asimétrica puede no tener sentido emplear la PMTU de un sentido para el tráfico en el otro
- (...)



Problemas con PMTUD

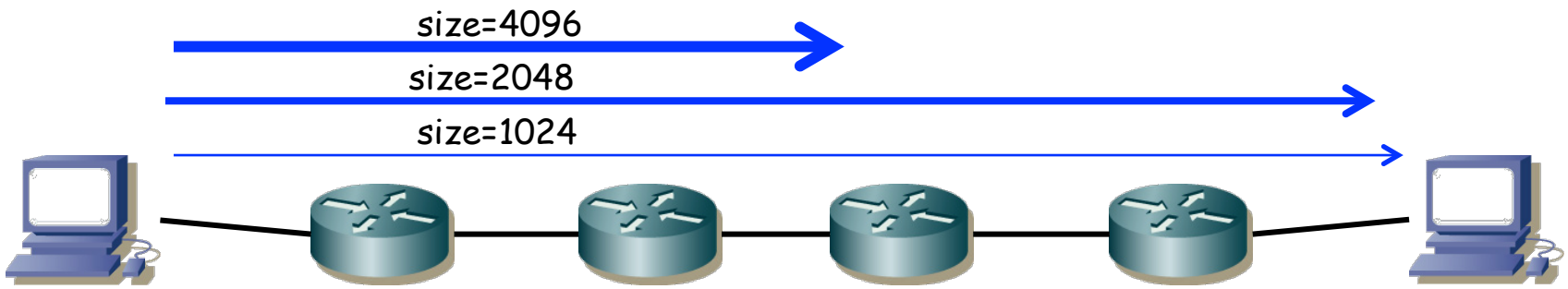
MSS a partir de PMTU

- Por otro lado, al imponer un MSS al otro sentido ya no hará pruebas de PMTUD con valores mayores
- Solución: decidir el MSS a partir de la MTU del interfaz



PLPMTUD

- RFC 4821 “Packetization Layer Path MTU Discovery” (2007)
- Pretende hacer PMTUD sin depender de mensajes ICMP pues suelen estar filtrados (*black hole*)
- “Packetization Layer” (PL) = Protocolo por encima de IP que decide las fronteras de los paquetes (tamaño de los segmentos)
- En algunas implementaciones se emplea esta técnica solo cuando PMTUD cae en un *black hole*
- La PL sondea el camino, probando con paquetes cada vez más grandes
- Si el paquete llega, se sube la PMTU a ésta
- Cuánto incrementar o decrementar no está fijado en la RFC
- (...)



PLPMTUD

- Una pérdida se considera una indicación de haber superado el límite de la MTU, no una indicación de congestión
- Se le permitiría al protocolo retransmitir un segmento sin ajustar cwnd
- Debe reimplementarse en cada PL
- Más discusión sobre la MTU en Internet:
  http://staff.psc.edu/mathis/MTU/