

# TCP: Control de Congestión

2019



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

0-0

## Servicios de TCP

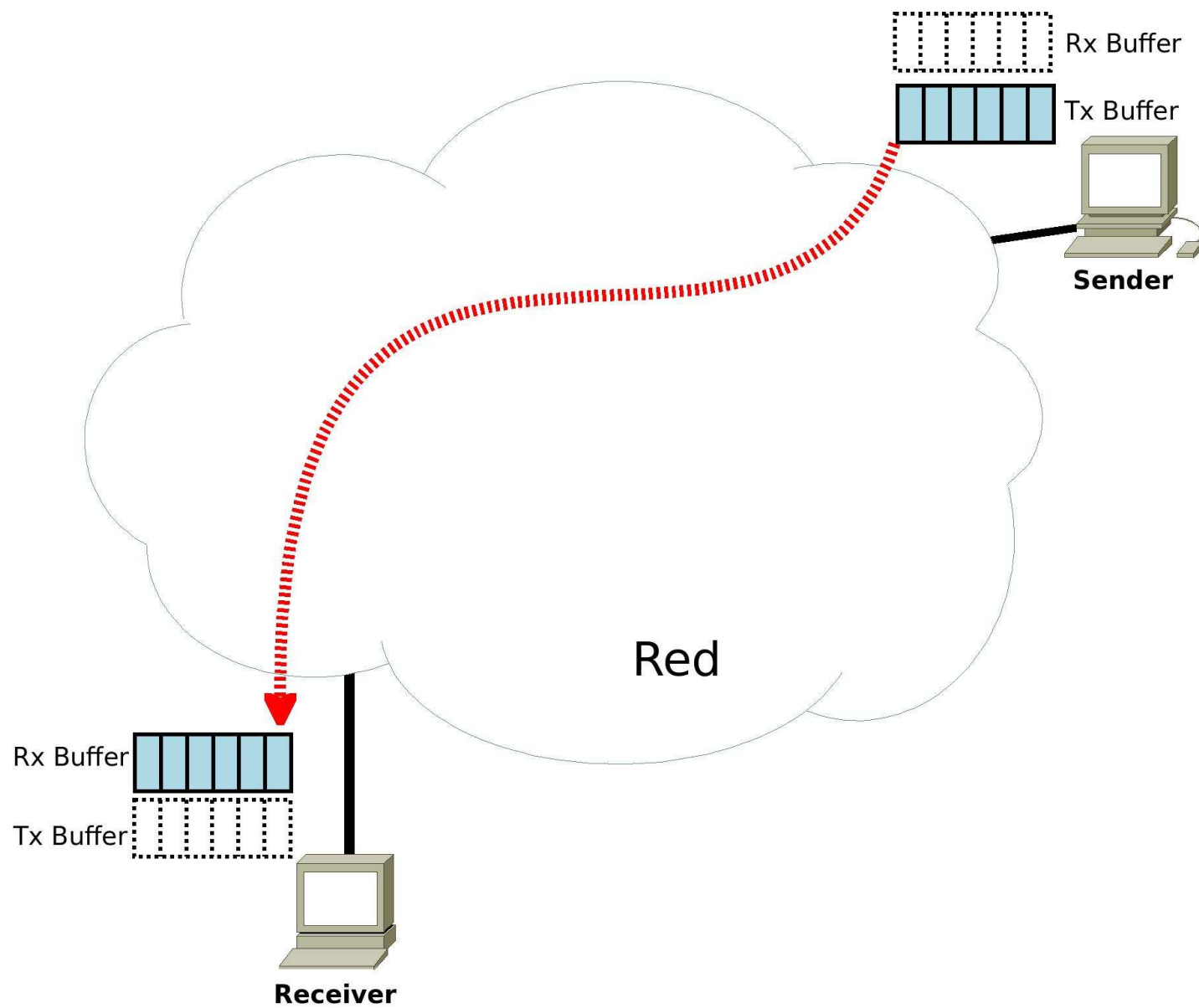
- Control de Errores:
  - Mecanismo protocolar que permite ordenar los segmentos que llegan fuera de orden y recuperarse mediante solicitudes y/o retransmisiones de aquellos segmentos perdidos o con errores.
  - Se realiza por cada conexión: End-to-End, App-to-App.
- Control de Flujo (Flow-Control):
  - Mecanismo protocolar que permite al receptor controlar la tasa a la que le envía datos el transmisor.
  - Controla cuanto puede enviar una aplicación sabiendo que la receptora tiene capacidad de recibirlo (espacio en RxBuf).

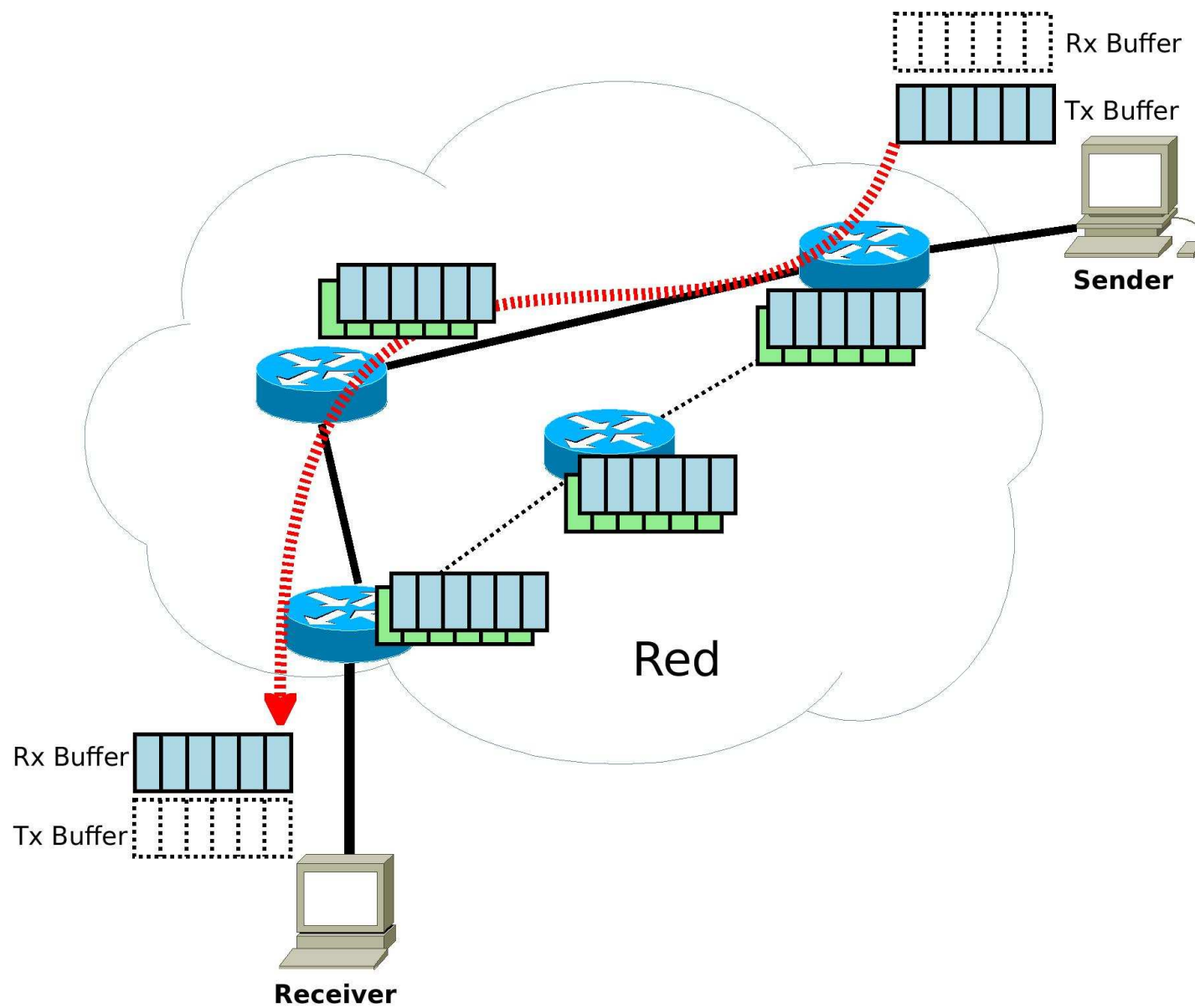
## Servicios de TCP (Cont.)

- Control de Flujo (Cont.):
  - Se realiza por cada conexión: End-to-End, App-to-App.
  - Permite que aplicaciones con diferentes capacidades dialoguen regulando la velocidad, tasa de transmisión.
  - Tiene en cuenta solo el estado del receptor, no el de la red.
- Control de Congestión:
  - Se realiza por cada conexión: End-to-End, App-to-App.
  - Permite que aplicaciones no saturen la capacidad de la red.
  - Tiene en cuenta el estado de la red a diferencia del control de flujo que solo ve el receptor.

## Control de Congestión TCP

- **Objetivo:** Controlar el tráfico que se envía evitando que se colapse la red y se descarte teniendo que retransmitirse.
- Se puede implementar End-to-End (caso TCP, RFC-5681 (ant. RFC-2581)).
- O tomando partida la red. Modelo basado en la Red: IP+TCP:RFC-3168 (similar a mecanismos L2/L3: Frame-Relay, ATM).
- **Congestión:** Problemas de delay en los routers, problemas de overflow y descarte.





## Causas de Congestión en la Red

- Límite de la capacidad de la red:
  - Velocidad de los Routers/Switches (CPU).
  - Capacidad de los Buffers de los Routers/Switches (Memoria).
  - Velocidad de los Enlaces (Interfaces).
- Utilización de la red:
  - Demasiado tráfico en la red (modelo de red compartida).
  - Se detecta por los nodos intermedios (router/switches) por ejemplo: cuando las colas sobrepasan un umbral. Se utiliza simple umbral o doble umbral (min,max).

## Control de Congestión, Modelo End-to-End

- Modelo en el cual no participa la red (más implementado).
- Se utilizan nuevos parámetros a los de Control de Flujo (variables locales):
  - *cwnd* Ventana de congestión. tiene en cuenta el estado de la red.
  - *ssthresh* Slow Start Threshold (Umbral).
  - Se calcula:  $MaxWin = Min(rwnd, cwnd)$ . *rwnd* era la ventana de recepción, usada para el control de flujo.

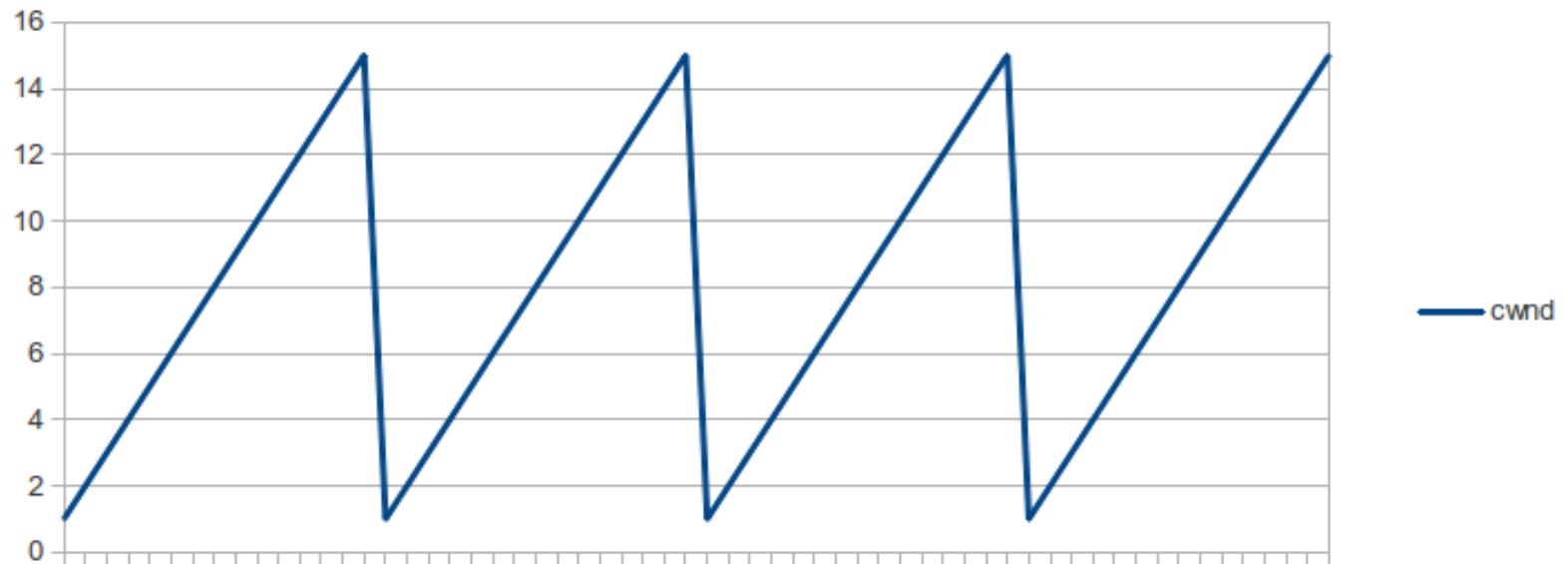


- $FlightSize = (LastByteSent - LastByteACKed)$   
segmentos en vuelo, enviados y aún no confirmados  
( $Sent.No.ACKed$ )
  - Puede enviar:  
 $Effective\_Win = MaxWin - FlightSize.$
  - $MaxWin = Min(rwnd, cwnd)$  se calcula en base a quién  
esta más cargado, el receptor o la red.
- Diferentes Fases:
- Si  $cwnd < ssthresh$ : Fase de crecimiento inicial: SS (Slow Start).
  - Si  $cwnd \geq ssthresh$  Fase de mantenimiento: CA (Congestion Avoidance).

## Evolución Control de Congestión TCP (Old Version)

- 1974, 1era. Versión TCP, Vint Cerf, Bob Kahn, TCP incluía también la capa de red (IP).
- Se divide en capas, TCP/IP, No considera control de Congestión inteligente.
  - “Ventana de congestión: *cwnd*” (no definida), tráfico crece hasta que se resetea, buffer overflow o error.
  - El destino podría limitarla con *rwnd*: válido para LAN.
  - No utiliza Slow Start (SS), se supone: incrementa  $cwnd++$  (en MSS) por cada RTT.
  - ACK perdido o segmento erróneo deriva en arrancar de 0.
  - Congestión/Límite de la red detectada por solo un tipo de evento, expira RTO.
  - Algoritmo no válido para entorno WAN, Internet.

## Control de Congestión TCP (Old Version)



## Control de Congestión TCP (Old Tahoe/Tahoe)

- Old Tahoe (BSD 4.2 - 1984 ?).
- TCP Tahoe (BSD 4.3 - 1988). Van Jacobson.
  - Utiliza Slow Start (SS), ventana crece exponencialmente, inicio  $cwnd = IW = 1 * MSS$ , o similar,  $IW$  (Init Window) puede ser 2 o 3 segmentos.
  - Una vez que se alcanza  $ssthresh$  se trabaja con Congestion Avoidance (CA).
  - Valor inicial  $ssthresh = \infty$  (un valor alto).
  - Old-Tahoe sin Fast Retransmit, solo SS y CA.

- Tahoe agrega Fast Retransmit, aunque no esta bien implementado.
- Congestión detectada por RTO o 3DUP ACKs, derivaba en ambos casos en: Slow Start:  
 $ssthresh = \text{Min}(cwnd/2, 2) * MSS$  ,  
 $cwnd = LW = 1 * MSS$  (Loss Window). (mejor implementado en TCP Reno).
- Puede suceder que MSS sea diferente entre emisor y receptor, para este caso se considera  $SMSS$  y  $RMSS$ . Los cálculos se hacen en base a  $SMSS$ .

## Control de Congestión TCP (SS)

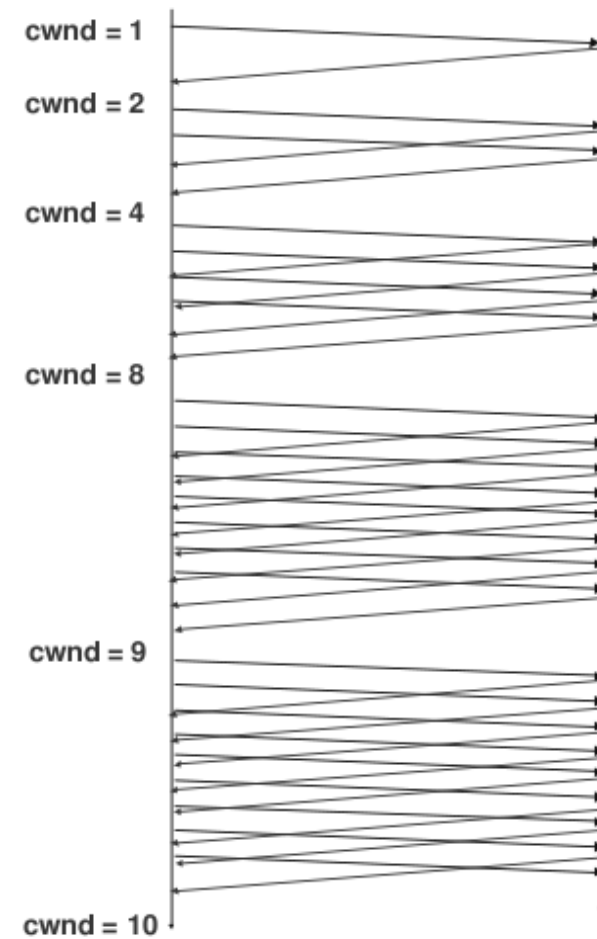
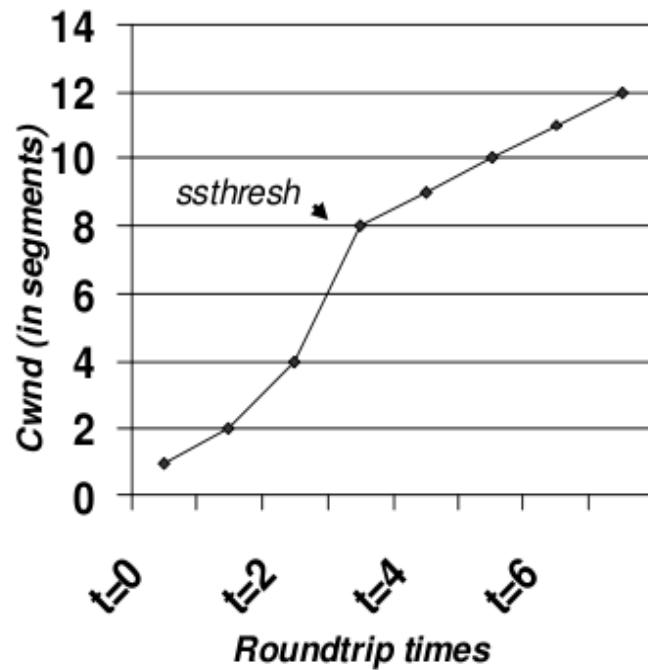
- Slow Start:
  - Crece exponencialmente, de forma rápida, no es slow (lento).
  - Se le llama Slow Start porque comienza a probar con pocos paquete, menos agresivo que el enfoque de enviar tanto como la la ventana de recepción permita.
  - Inicia  $cwnd = IW = 1 * MSS$  (a veces se usa 2 o 3). Transmite y espera ACK.
  - ACK recibido,  $cwnd++$ :  $cwnd = 2 * MSS$ , nuevos ACKs:  $cwnd = 4 * MSS \dots$
  - Si destino retarda ACK no se cumple.
  - Incrementa  $cwnd++$  (en MSS) por cada ACK (varios por RTT, ráfaga).

## Control de Congestión TCP (CA)

- Congestion Avoidance:
  - Ante el primer evento de congestión se calcula el *ssthresh*, primer ráfaga SS puro.
  - Una vez que  $cwnd \geq ssthresh$  crece de forma lineal.
  - Incrementa  $cwnd++$  por cada RTT.
- Fast Retransmit: objetivo, recuperarse más rápido que un timeout. En Tahoe, FRT seguido por Slow Start. Vuelve al inicio  $cwnd = 1 * MSS$ .

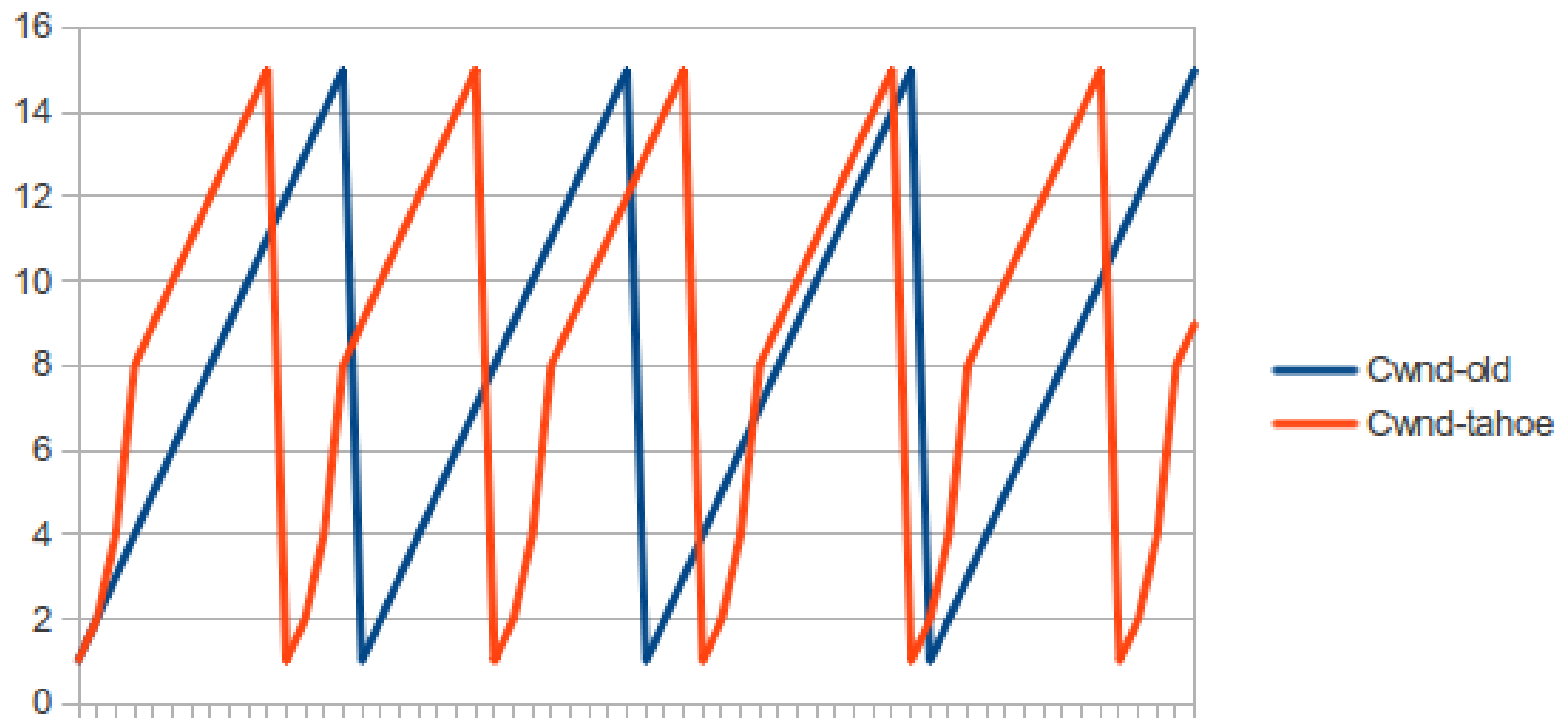
## Fases Control de Congestión TCP (SS y CA)

Assume that  $ssthresh = 8$





## Control de Congestión TCP (Old Version vs. Tahoe)



## Control de Congestión TCP (Reno)

- TCP Reno (BSD 4.3 - 1990) Van Jacobson.  
RFC-2001/RFC-2581/RFC-5681.
- Se implementan de forma correcta Fast Retransmit y Fast Recovery.
  - Slow Start (SS)
  - Congestion Avoidance (CA).
  - Fast Retransmit (FRT)
  - Fast Recovery (FR).
- $FlightSize = cwnd$  si siempre tuvo datos para enviar, aunque en general  $cwnd > FlightSize$ , donde  $cwnd$  crece sin enviar realmente datos, por lo tanto no reflejaría en ese caso el límite de la red.

## Control de Congestión TCP (Reno FRT)

### ■ Fast Retransmit:

- Intenta recuperarse más rápido que un timeout (expira RTO).
- El receptor inmediatamente al recibir un segmento fuera de orden no debe esperar RTO del emisor sino generar un ACK (DUP ACK). No se debe retardar.
- Debido que el emisor no sabe si se perdió o llegó fuera de secuencia: espera por más ACK duplicados.
- El emisor, si recibe 3 ACK duplicados (4 ACK para el mismo segmento) considera que se perdió (no es re-ordenamiento) y re-envía el segmento solicitado sin esperar RTO.

## Control de Congestión TCP (Reno FR)

- Fast Recovery:
  - Si recibe 3 ACK duplicados (4 ACK para el mismo segmento) entro en Fast Retransmit y se reenvió el segmento.
  - Luego de Fast Retransmit entra en fase Fast Recovery, crece lineal.
  - Incrementa de forma lineal la ventana por cada ACK recibido, considera que es un espacio nuevo en la red (incremento inicialmente en 3, por los 3ACK duplicados)
  - Luego de Fast Recovery (se ACKed el segmento perdido) se realiza CA(Reno), no SS(Tahoe).

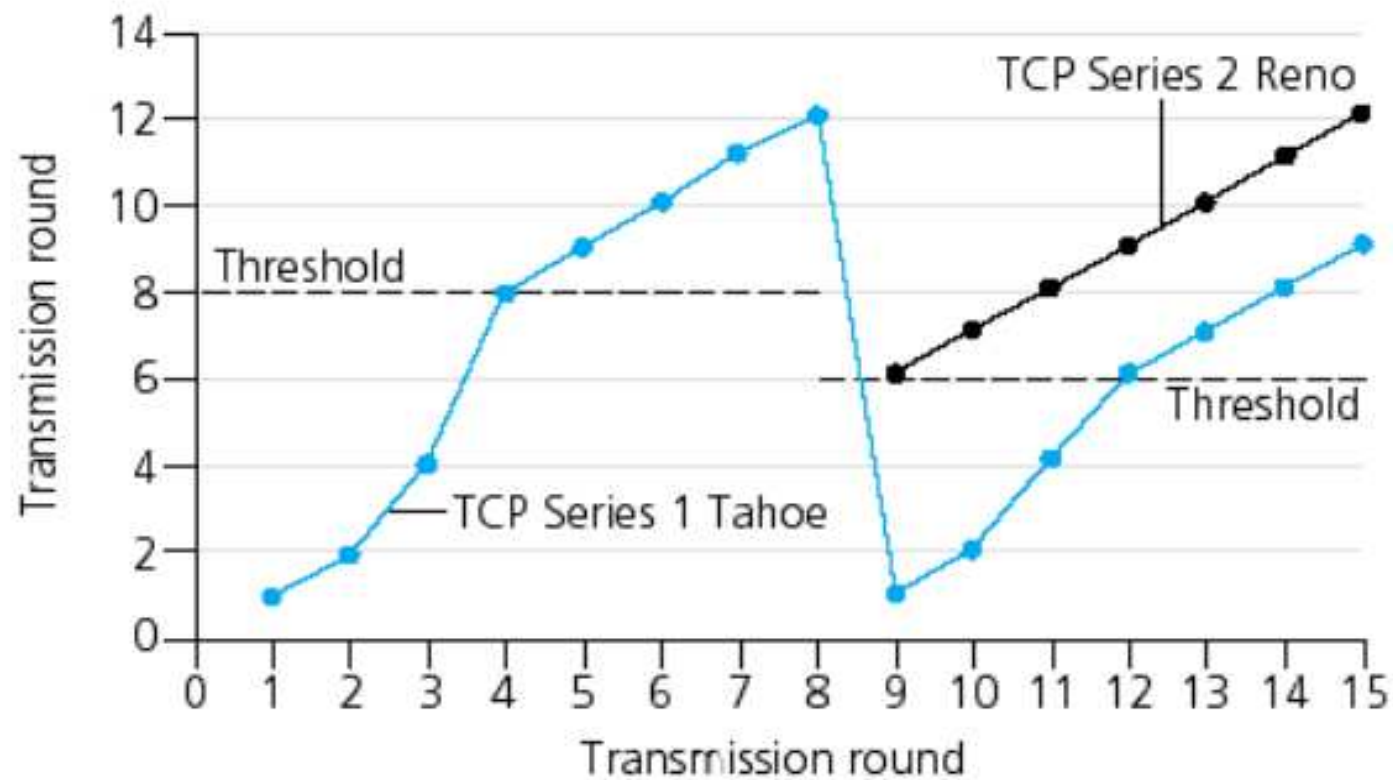
## Control de Congestión TCP (Reno)

- Si se detectan 3DUP ACKs (Fast Retransmit):
  1.  $ssthresh = \text{Min}(cwnd/2, 2) * MSS$ , en RFC  
 $ssthresh = \text{Min}(\text{FlightSize}/2, 2 * SMSS)$
  2. Fast Retransmit.
  3. Fast Recovery:  $cwnd = (ssthresh + 3) * MSS$  (3 segments cached in receiver).
  4. Por cada ACK de segmento distinto que recibe incrementa la ventana  $cwnd++$  (crece linealmente).
- Una vez recuperado (ACKed segmento perdido) vuelve “a la mitad”,  $cwnd = ssthresh$ , y comienza con CA.
- Los incrementos lineales realizados en FR previos a la recuperación se vuelven atrás (se achica un poco  $cwnd$ ).

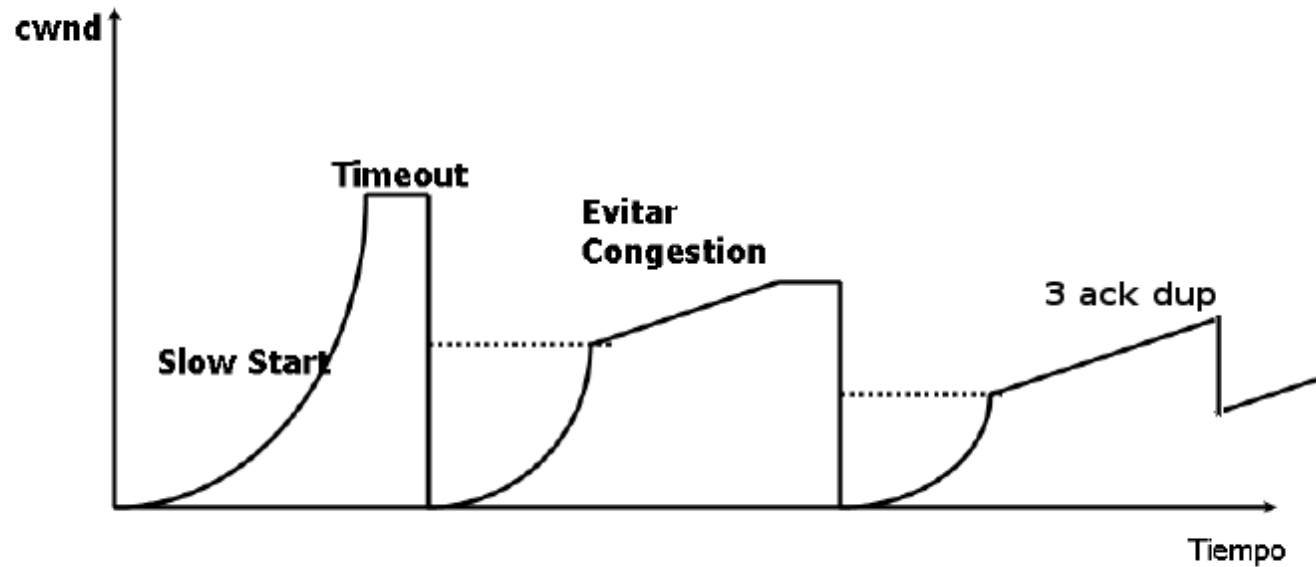
## Control de Congestión TCP (Reno)

- Si se timeout, RTO expira.
  1. Retransmite el segmento perdido.
  2.  $ssthresh = \text{Min}(cwnd/2, 2) * MSS$ , en RFC  
 $ssthresh = \text{Min}(\text{FlightSize}/2, 2 * SMSS)$
  3.  $cwnd = LW = 1 * MSS$ , en RFC  $LW = 1 * SMSS$ .
  4. Inicia con Slow Start como en los algoritmos anteriores.
  5. Comienza a retransmitir como Go-Back-N a partir del segmento que dio timeout de acuerdo a lo que le permite la ventana.

## Control de Congestión TCP (Fases)

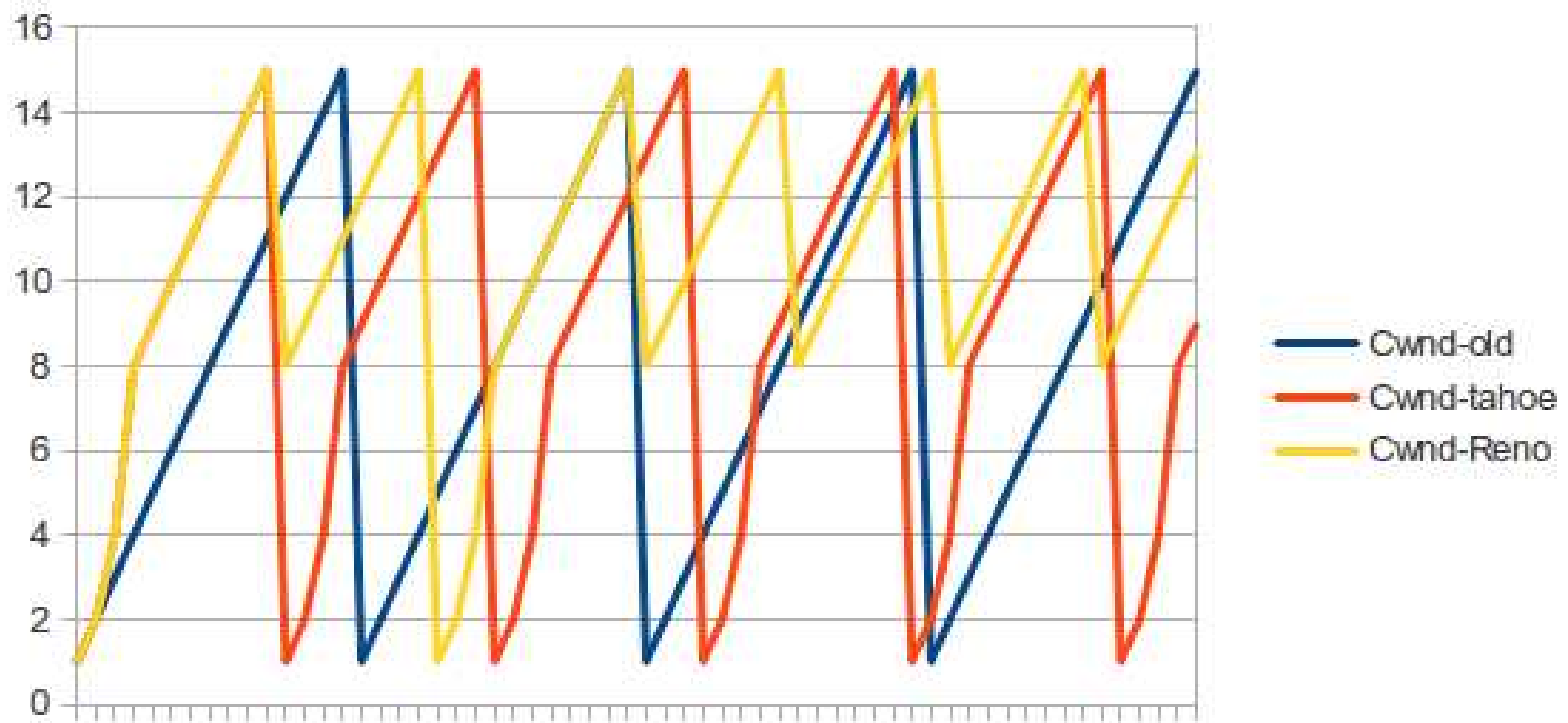


## Ejemplo: Evolución Ventana de Congestión

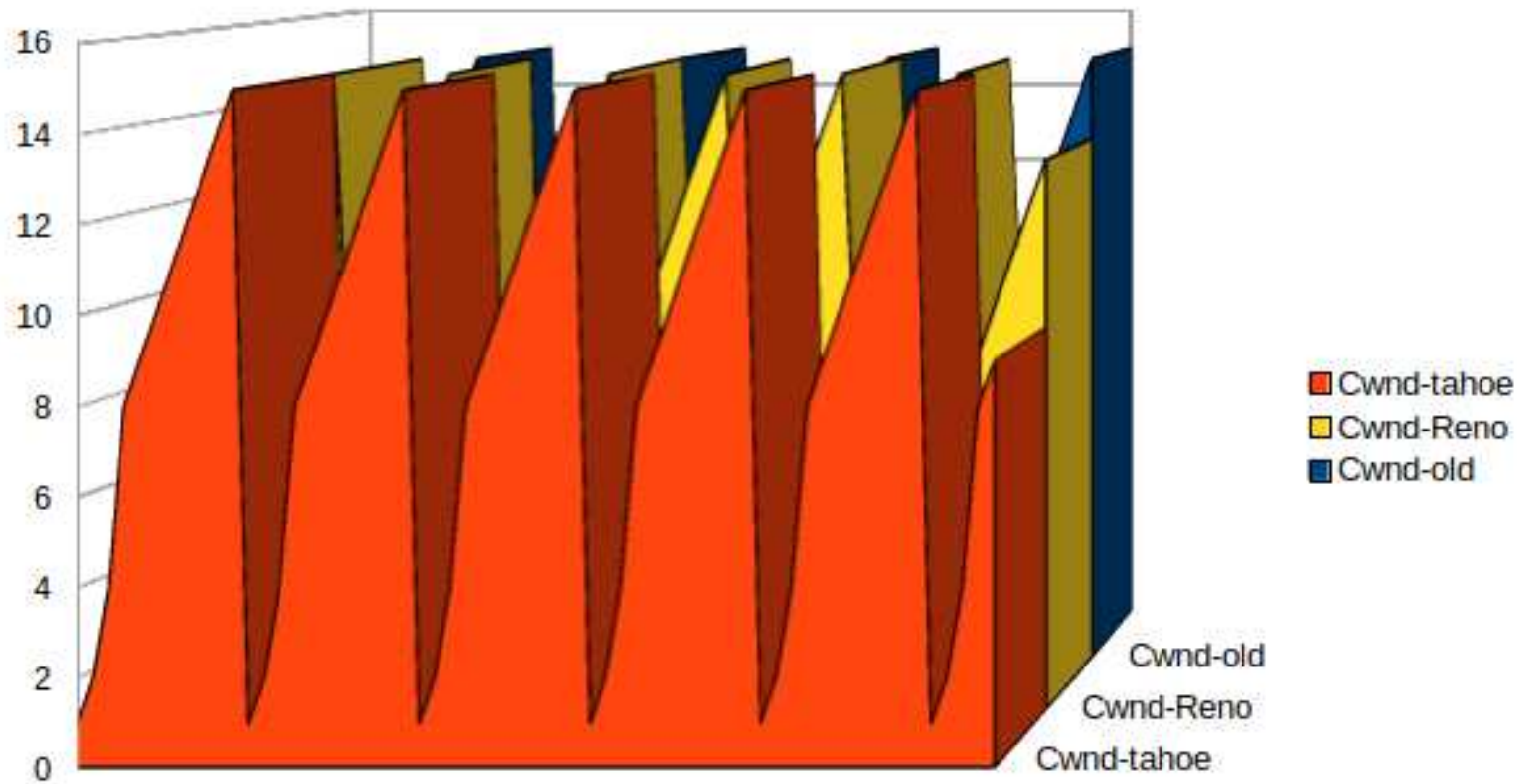




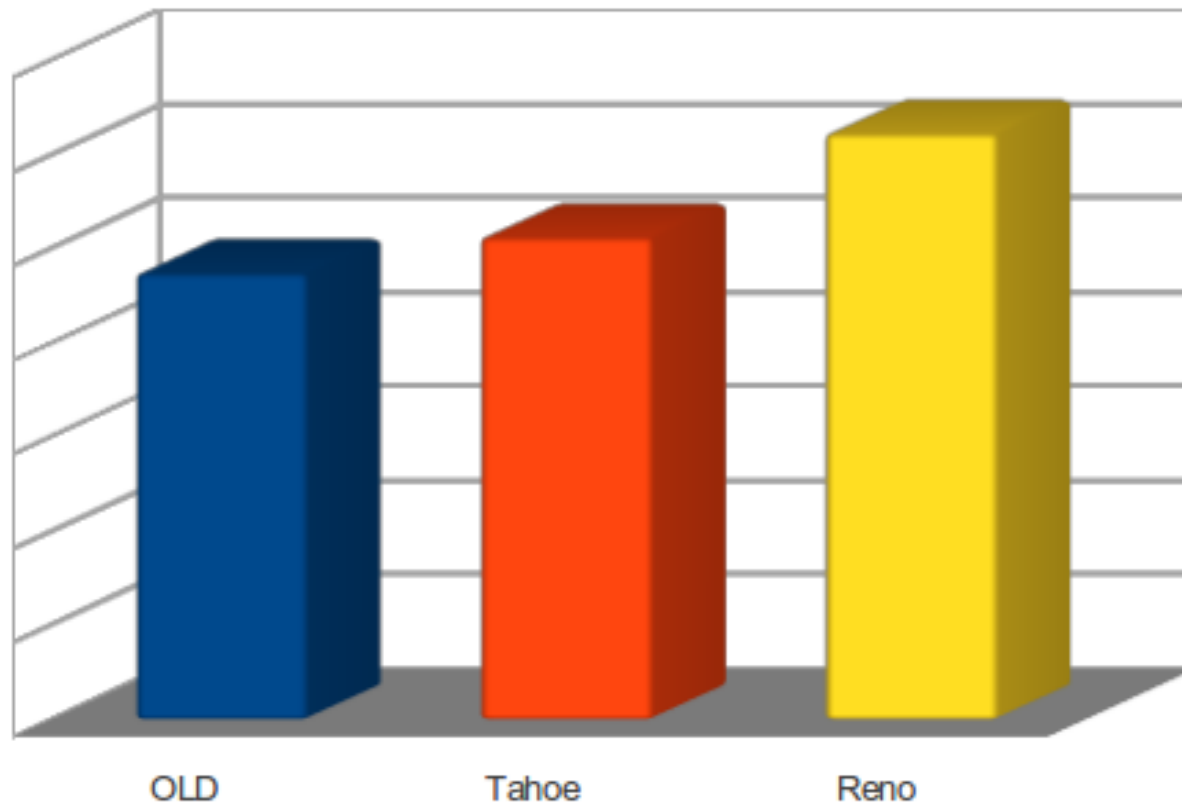
## Control de Congestión TCP (Old vs. Tahoe vs. Reno)



## Control de Congestión TCP (Old vs. Tahoe vs. Reno) 3D



## Control de Congestión TCP (Old vs. Tahoe vs. Reno)



## Modelo de Algoritmo TCP CC

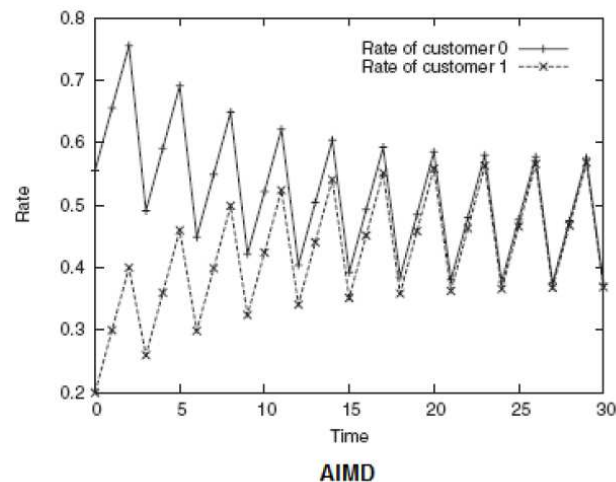
```
void tcp_snd(event_t ev)
{
    switch (ev) {
        case (init):
            cwnd = 1;
            ssthresh = INF;
            ...
            break;
        case (newack):
            if (cwnd < ssthresh) {
                /* Slow Start */
                /* 1 MSS for each ACK */
                cwnd++; // cwnd = cwnd + 1
            } else {
                /* Congestion Avoidance */
                /* 1 MSS for each RTT */
                cwnd = cwnd + 1/cwnd;
            }
            ...
            break;
        ...
    }

    ...
    case (timeout):
        ssthresh = cwnd / 2;
        cwnd = 1;
        ...
        break;
        case (3ackdup):
            fast_retrans_fast_recover();
            ssthresh = cwnd / 2;
            cwnd = ssthresh + 3;
            ...
    }
}
```

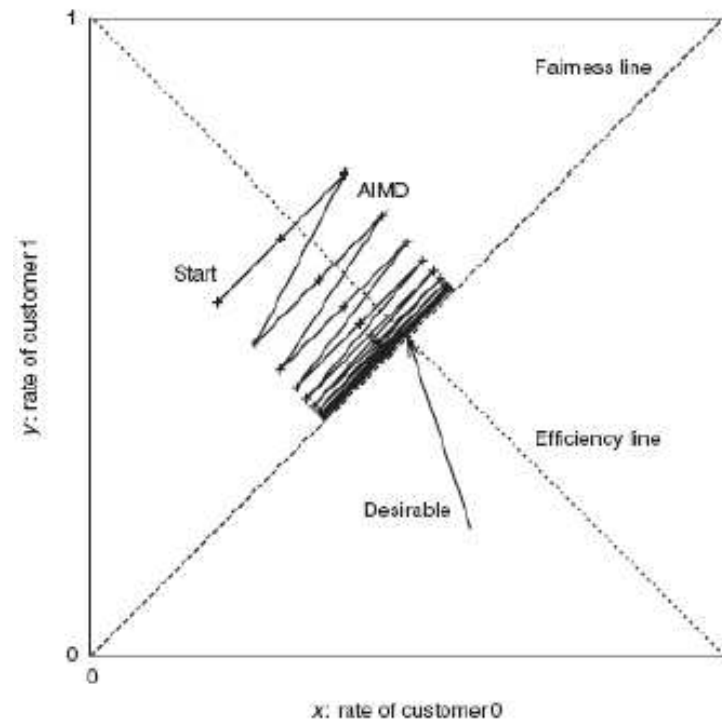
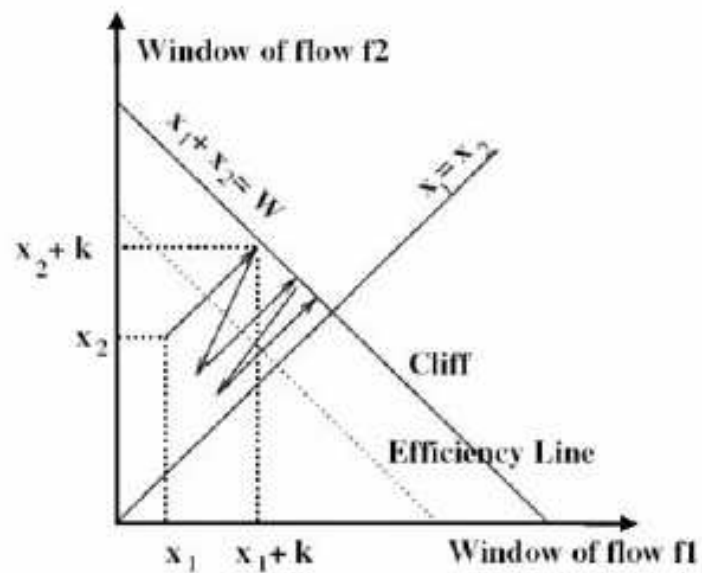
## AIMD (Additive Increase/Multiplicative Decrease)

- De acuerdo al modelo de TCP parece ser estable y justo.
- Utiliza un enfoque AIMD, crece de forma aditiva con ACK positivos y decrece de forma multiplicativa ante 3 ACK DUP (a la mitad).
- Se tratan de autoregular entre flujos.
- $a_i > 0; a_d = 0$
- $b_i = 1; 0 < b_d < 1$

$$w(t+1) = \begin{cases} a_i + b_i w(t) & \text{si } (+) \\ a_d + b_d w(t) & \text{si } (-) \end{cases}$$



# AIMD (Additive Increase/Multiplicative Decrease)



## Control de Congestión TCP (New Reno)

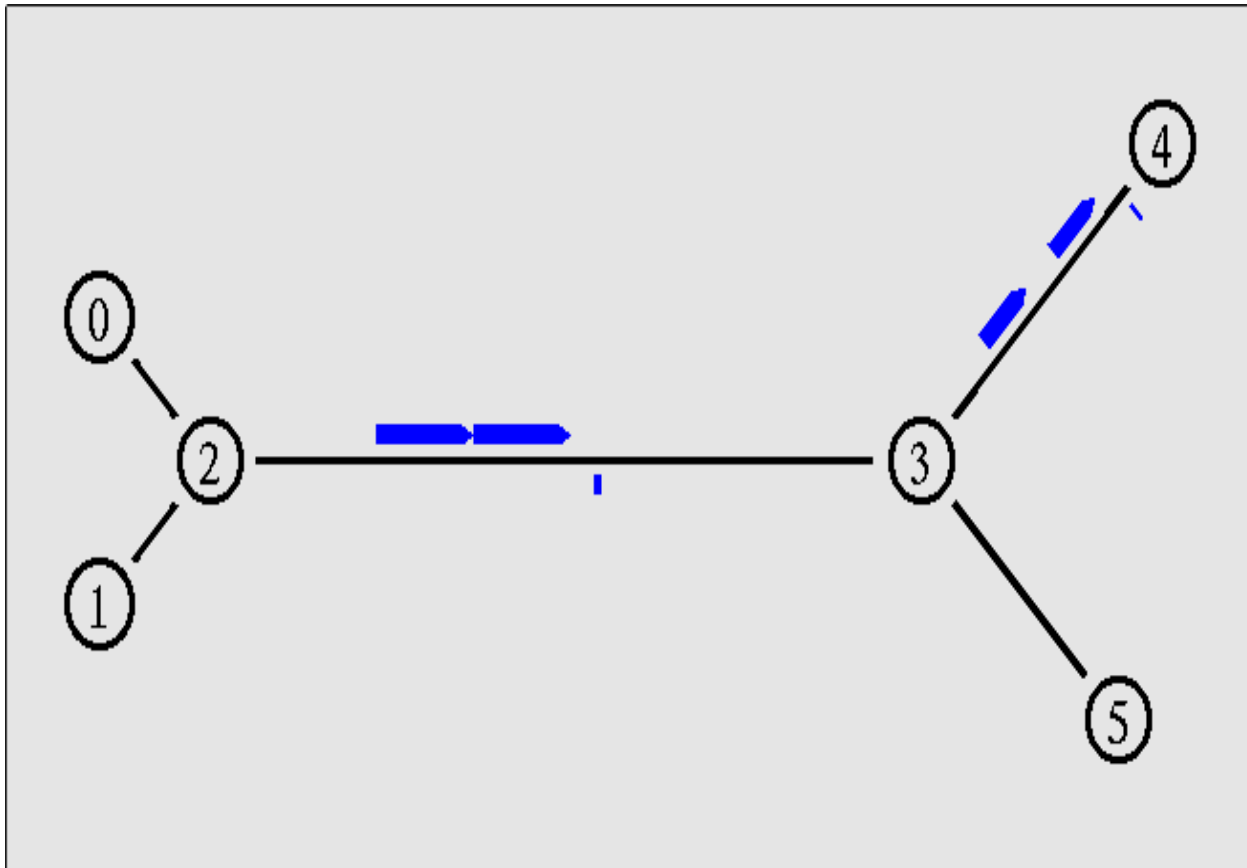
- TCP Reno ante la pérdida de múltiples segmentos probablemente termine en timeout (RTO expira) y vuelve a Slow Start.
- TCP New Reno, última versión RFC 6582(2012) hace obsoletas RFC 3782 (2004), RFC 2582(1999) S. Floyd, T. Henderson.
- Una vez que esta en Fast Recovery, permite enviar varios segmentos perdidos y recuperase de ACK parciales.
- New Reno obtiene buen rendimiento con pérdida en varios segmentos y sin SACK.

## Otras Implementaciones TCP CC

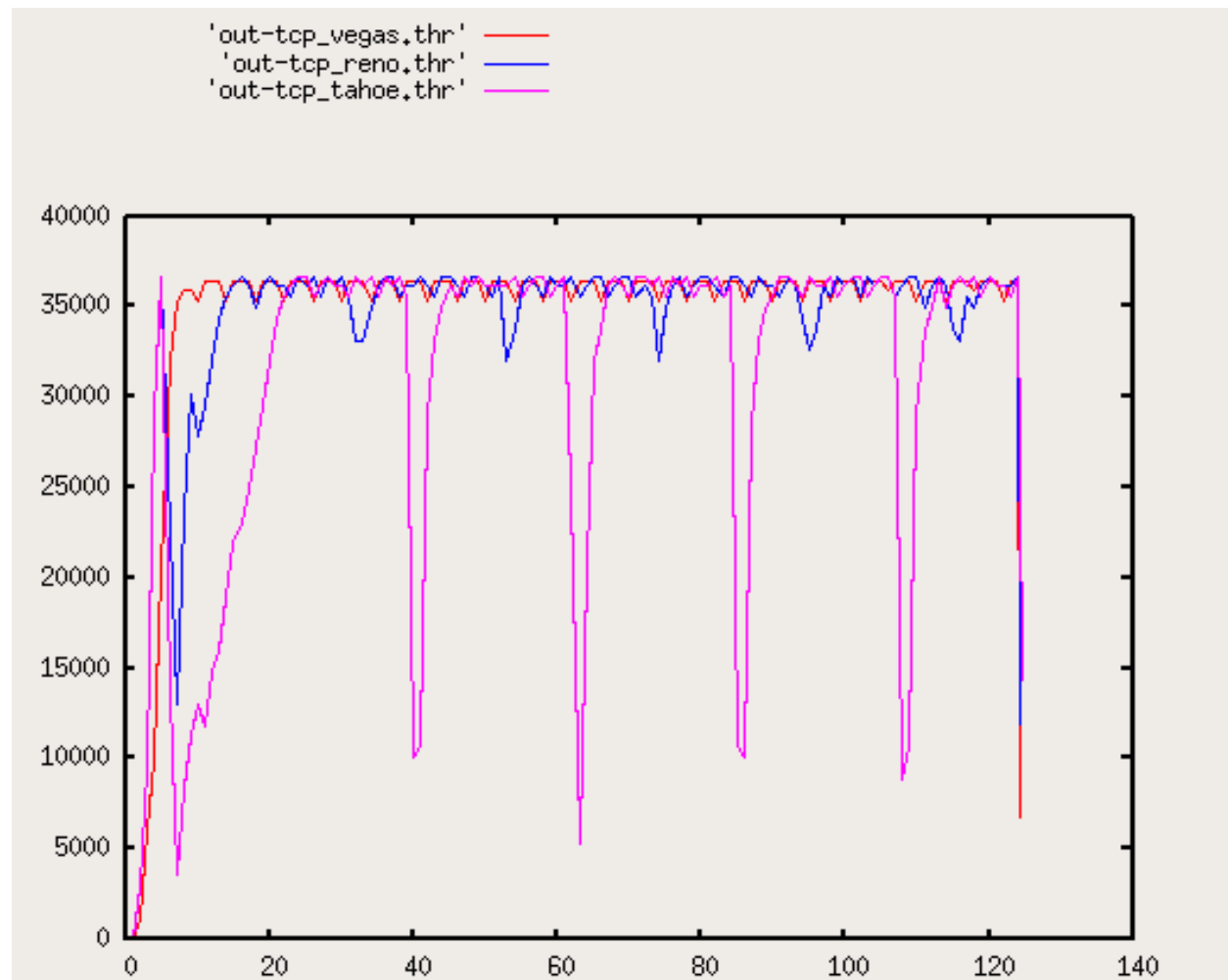
- TCP New Reno (1996), mejora ante la pérdida de más segmentos.
- SACK (1996).
- TCP Vegas.
- TCP BIC.
- TCP Cubic.
- TCP Westwood.



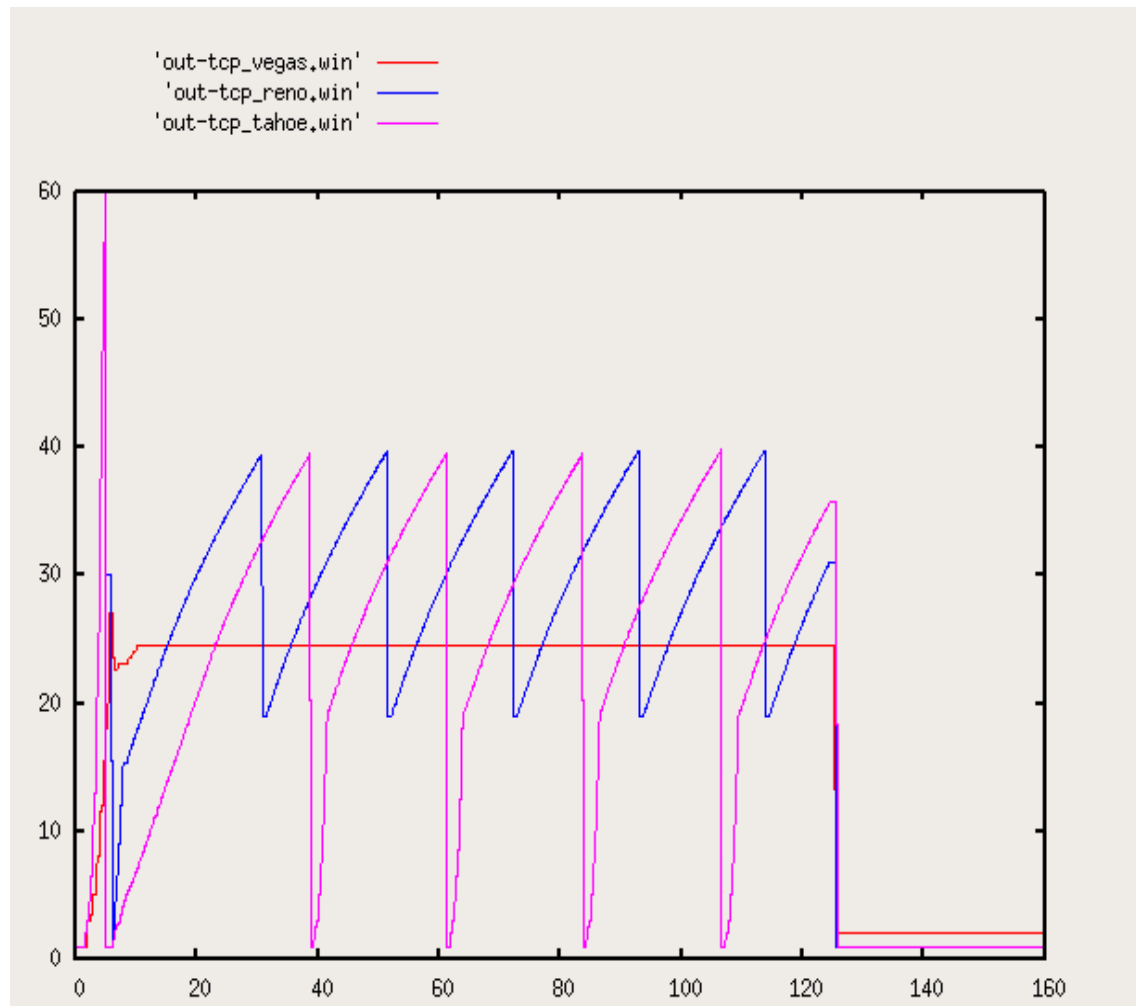
## Implementaciones Modelo



## Implementaciones (Comparar Throughput)



## Implementaciones (Comparar cwnd)



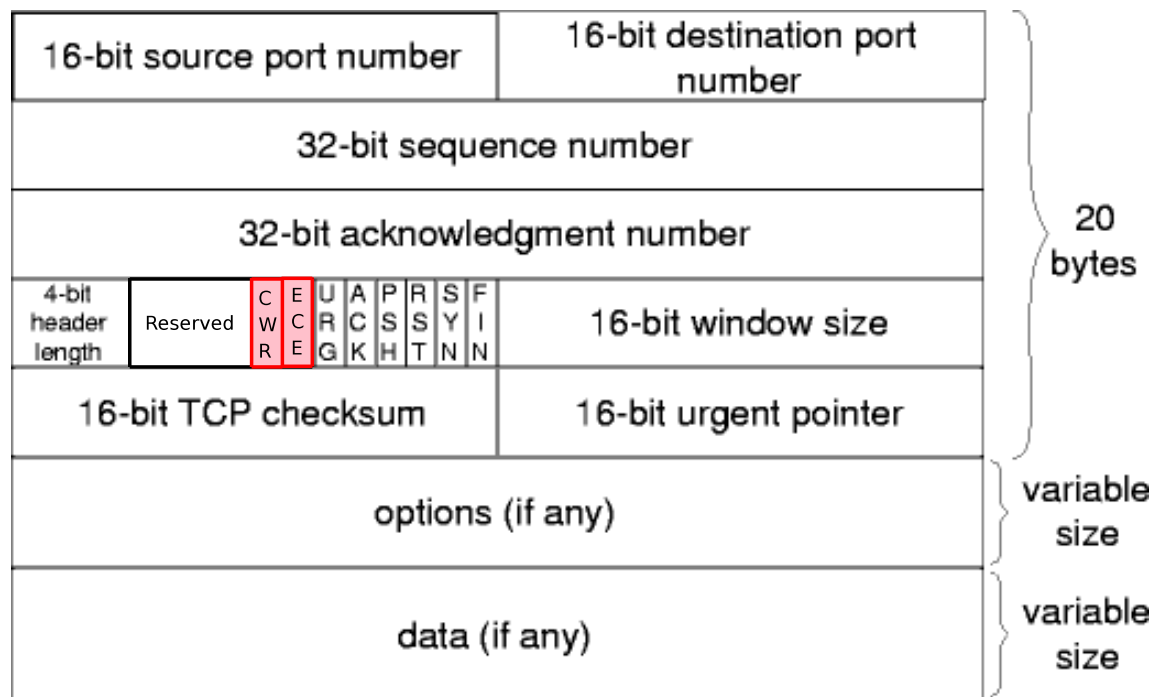
## Control de Congestión por la Red

- TCP se monta sobre IP, best effort protocol, no considera en su origen QoS.
- Luego RFC-1349(ToS), obsoleta por RFC-2474(DSCP).
- Luego con RFC-3168 se agrega funcionalidad de CC por la red.
- Se utilizan campos reservados para marcar tráfico.

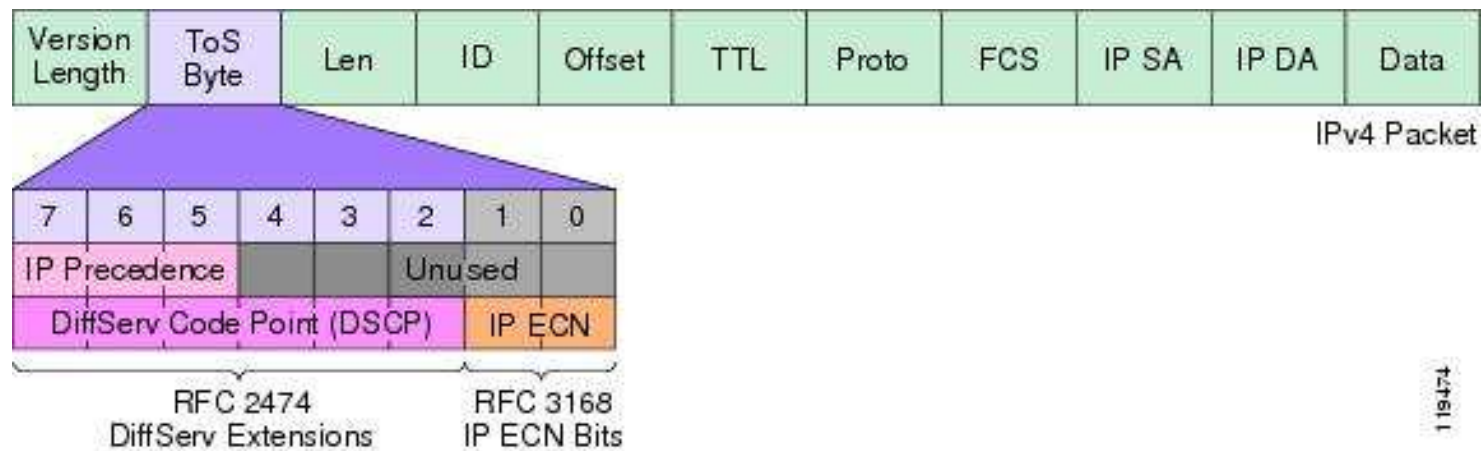
## Control de Congestión por la Red (Cont'd)

- Los destinos congestionados, o routers intermedio pueden generar ICMP: Source Quench.
- Estos mensajes deberían ser considerados y bajar el data rate.
- No solo funcionan con UDP, es a nivel IP.
- Habitualmente son mensajes ignorados.
- TCP+IP con ECN es un mecanismo más adecuado.

## Segmento TCP Marcado con ECE y CWR



## Datagrama IP Marcado con ECN: ECT, CE

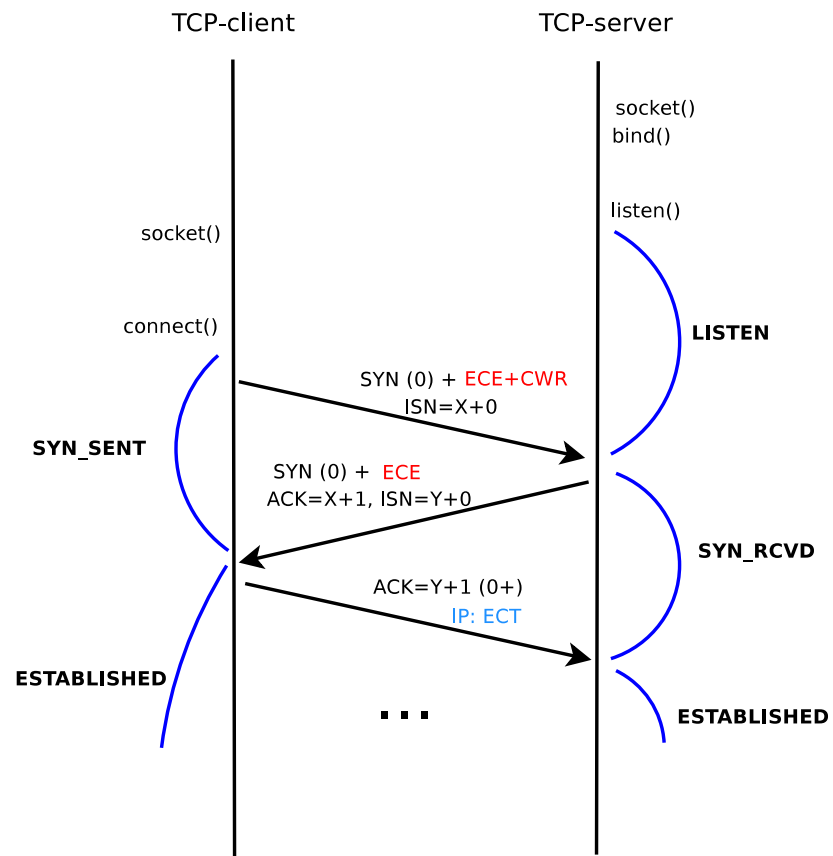


## Control de Congestión TCP+IP (Cont'd)

- Al establecerse la conexión TCP, se negocia capacidad de ECN.
- Configuran en el Setup, SYN el  $ECE = CWR = 1$  (ECN-Echo). “ECN-setup SYN packet”.
- SYN+ACK configuran  $ECE = 1$  y  $CWR = 0$ .
- Luego en datagramas IP se configuran  $ECT(0) = 01$  o  $ECT(1) = 10$  (ECN-Capable Transport).
- Si router (nodo intermedio) detecta congestión, tamaño de cola por encima de umbral aplica RED (Random Early Detection).
- Al aplicar RED, en lugar de descartar marca  $CE = 11$  (Congestion Experienced).



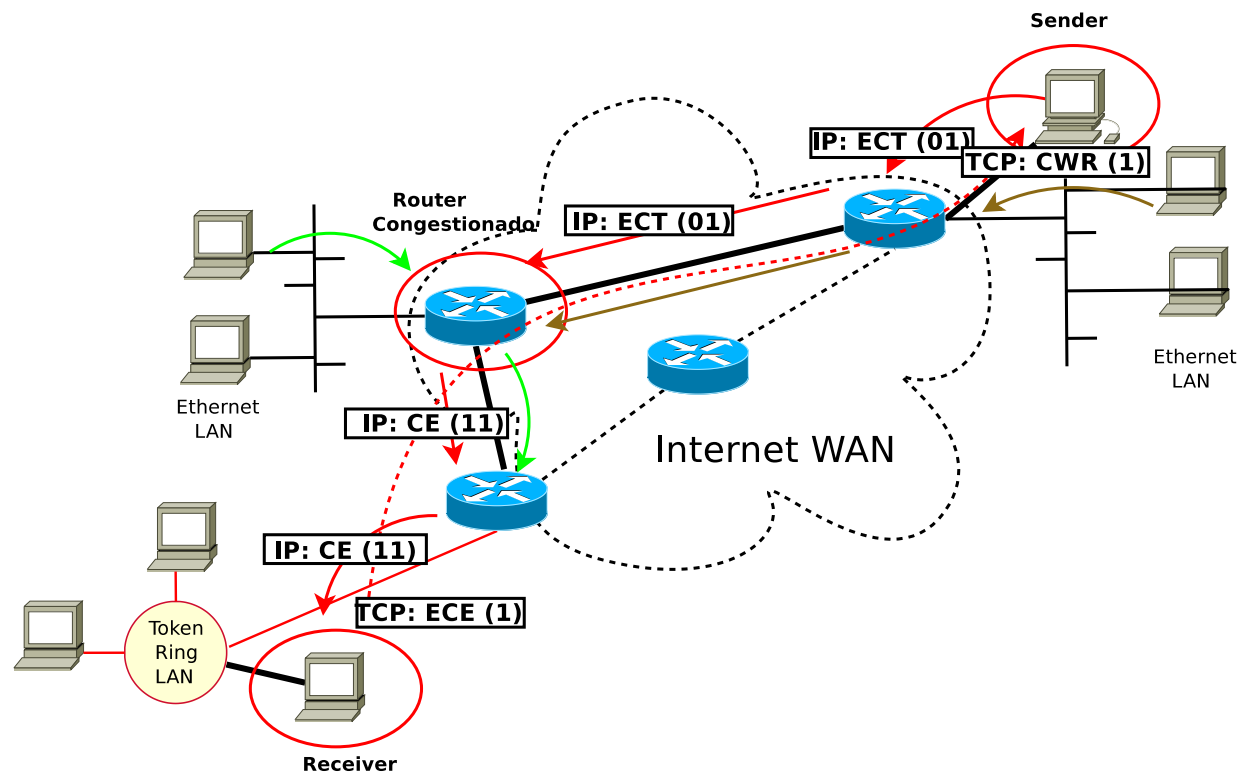
# TCP ECN Setup



## Control de Congestión TCP+IP (Cont'd)

- El receptor al recibir  $CE = 11$ , setea ECE en el próximo segmento de ACK.
- El emisor detecta  $ACK == ECE == 1$  y aplica TCP CC clásico como si el mensaje hubiese sido descartado.
- El emisor también configura  $CWR = 1$  (Congestion Window Reduced) para notificar que reacciono.

# TCP+IP ECT+CE, ECE+CWR



## Fuentes de Información

- Kurose/Ross: Computer Networking (5th Edition).
- TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.
- TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall.
- A Protocol for Packet Network Intercommunication. Cerf, Vinton G. & Kahn, Robert E. (1974).
- RFC 675, Specification of Internet Transmission Control Protocol, V. Cerf et al. (December 1974).
- RFCs: <http://www.faqs.org/rfcs/> RFC-793, ... RFC-791, RFC-1323, RFC-2001, RFC-2018, RFC-2581, RFC-5681, RFC-2582, RFC-6582, RFC-3168, RFC-3649, RFC-2988, RFC-6298.

- Jaco88: Van Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM '88.
- CJ89: Chiu, D. and R. Jain, “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks”, Journal of Computer Networks and ISDN Systems, vol. 17, no. 1, pp. 1-14, June 1989.
- Jaco90: Van Jacobson, “Modified TCP Congestion Avoidance Algorithm”, email to end2end-interest@ISI.EDU, April 1990.
- FF96: Fall, K. and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno and SACK TCP”, Computer Communication Review, July 1996. <ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z>.
- Wikipedia <http://www.wikipedia.org>.
- Slides del Prof. Luis Marrone del curso Ing. de Tráfico.
- TCP/IP Guide: <http://www.tcpipguide.com/>.

- Transport Layer: <http://people.westminstercollege.edu/faculty/ggagne/spring2007/352/notes/unit4/index.html>
- Internet ...