# Protocolos de Transporte: TCP
(parte II)

Andres Barbieri

barbieri(at)cespi.unlp.edu.ar

Octubre 2014

# 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.
  - Control cuanto puede enviar una aplicación sabiendo que la receptora tiene capacidad de recibirlo.

## Servicios de TCP (Cont'd)

- Control de Flujo (Cont'd):
  - 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 el estado del receptor y 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.

## Servicios de TCP (Cont'd)

- Para realizar control de errores y control flujo se utilizan técnicas de ARQ.
- ARQ solo no hace control de flujo, requiere de otros mecanismos como RNR, o Dynamic Window.
- La capacidad de envío
  será $MIN(Congestion, Flujo, Errores)$.

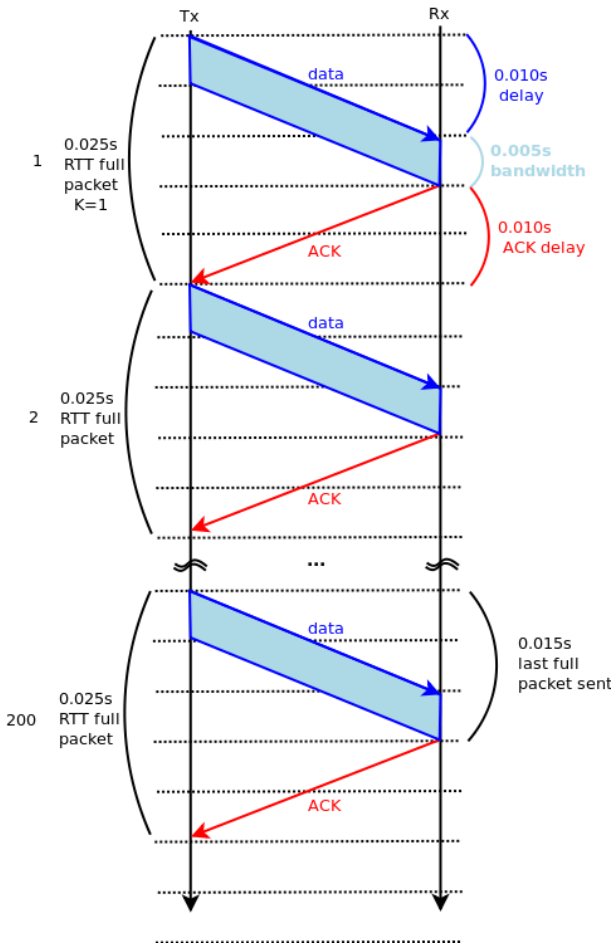
## Técnicas de ARQ (Automatic Repeat reQuest)

- Stop-and-Wait (Poco eficiente).
- Ventana Deslizante (Sliding Window).
  - Ventana Estática: Capacidad fijada en el emisor. Usado en L2 (Static Go Back-N). (Las técnicas estáticas realizan control de pérdidas, NO control de flujo por si solas, requieren mensajes de control extra para indicar sobrecarga).
  - Ventana Dinámica: Capacidad anunciada por el receptor (Dynamic Go Back-N). Sirve para hacer control de flujo.
- Ventana Selectiva (Selective Sliding Window) / SR (Selective Repeat).

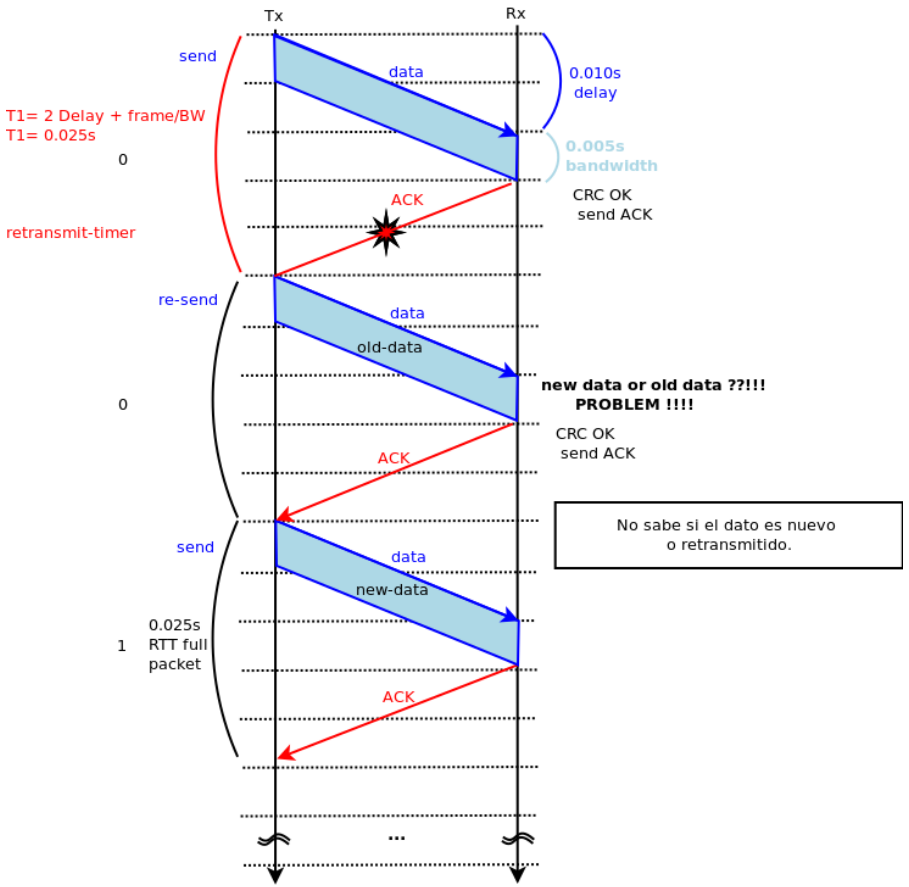
## Stop-and-Wait

- No se envía el próximo mensaje hasta que no se confirma el que se envió.
- Sistema simple y poco eficiente: no optimiza producto: Delay, Bandwidth: $D \times B$, $D = RTT$.
- Ventana de Tamaño $K = 1$.
- Cada vez que envía un segmento requiere arrancar un timer: $RTO$ o $T1$.
- Si no recibe confirmación se vence el timer y retransmite.
- Para ser libre de errores requiere un bit de secuencia 0..1: de forma de no confundir datos re-enviados de nuevos cuando se pierden ACK.

# Stop-and-Wait (Cont'd)



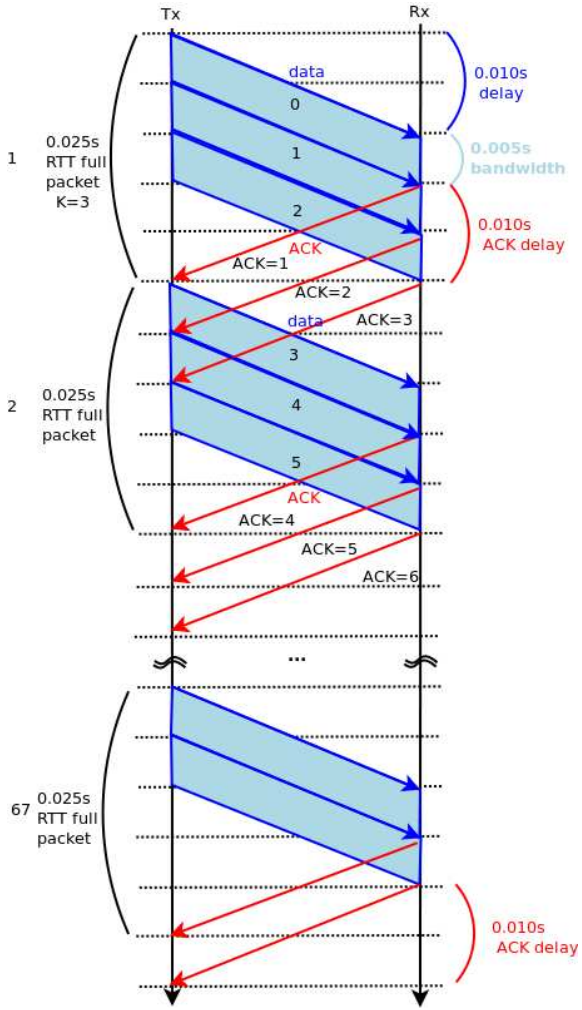
# Stop-and-Wait (Cont'd)

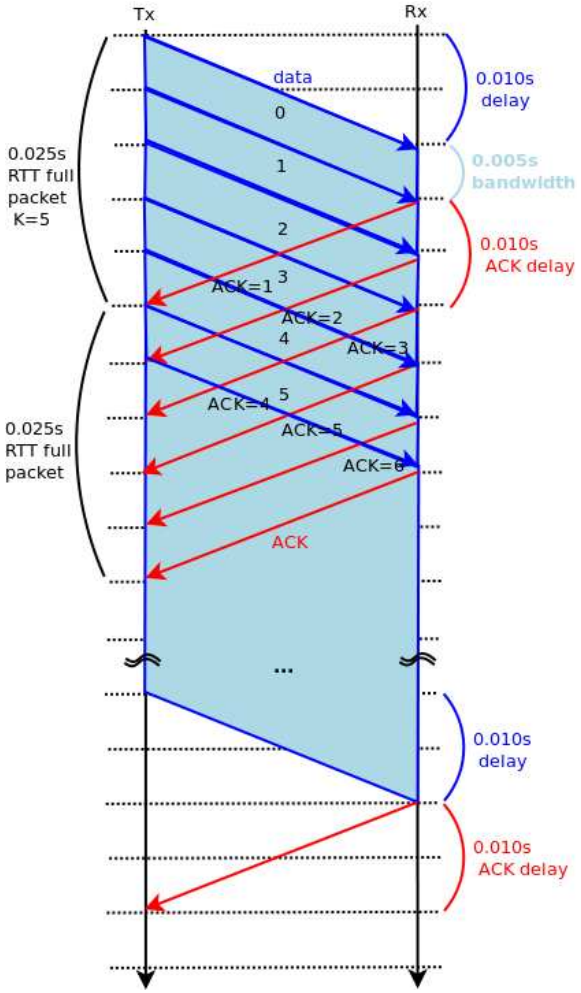


## Go-Back-N Estático

- Se tiene una ventana estática de tamaño $K = n, n > 1$.
- Se pueden enviar $n$ mensajes, segmentos sin recibir confirmación.
- Por cada mensaje enviado se inicia un timer de retransmisión: $T1$ o $RTO$.
- Más eficiente, si llena el pipe.
- Por cada confirmación se descarta/reinicia el timer $RTO$. Si no se recibe confirmación vence $RTO$ (timeout) y se retransmite.
- Podría generarse Confirmaciones Negativas: NAK (NO Acknowledge).

# Go-Back-N (Cont'd)



## Go-Back-N (Cont'd)

- Se requiere numerar los mensajes, segmentos, más de un bit. Se realiza en módulo $M$, $K <= (M-1)$
- El receptor puede usar: timer de ACK, $T2$, para confirmar. $T1 > T2$, $T1 > T2 + RTT$
- Se pueden aprovechar tramas de datos para confirmar: Piggy-backing.
- Se puede confirmar confirma desde $N$ hacia atrás (ACK acumulativos). No necesariamente se confirma individualmente.
- Si se re-envía se hace desde $N$ hacia adelante, los que ya se enviaron.

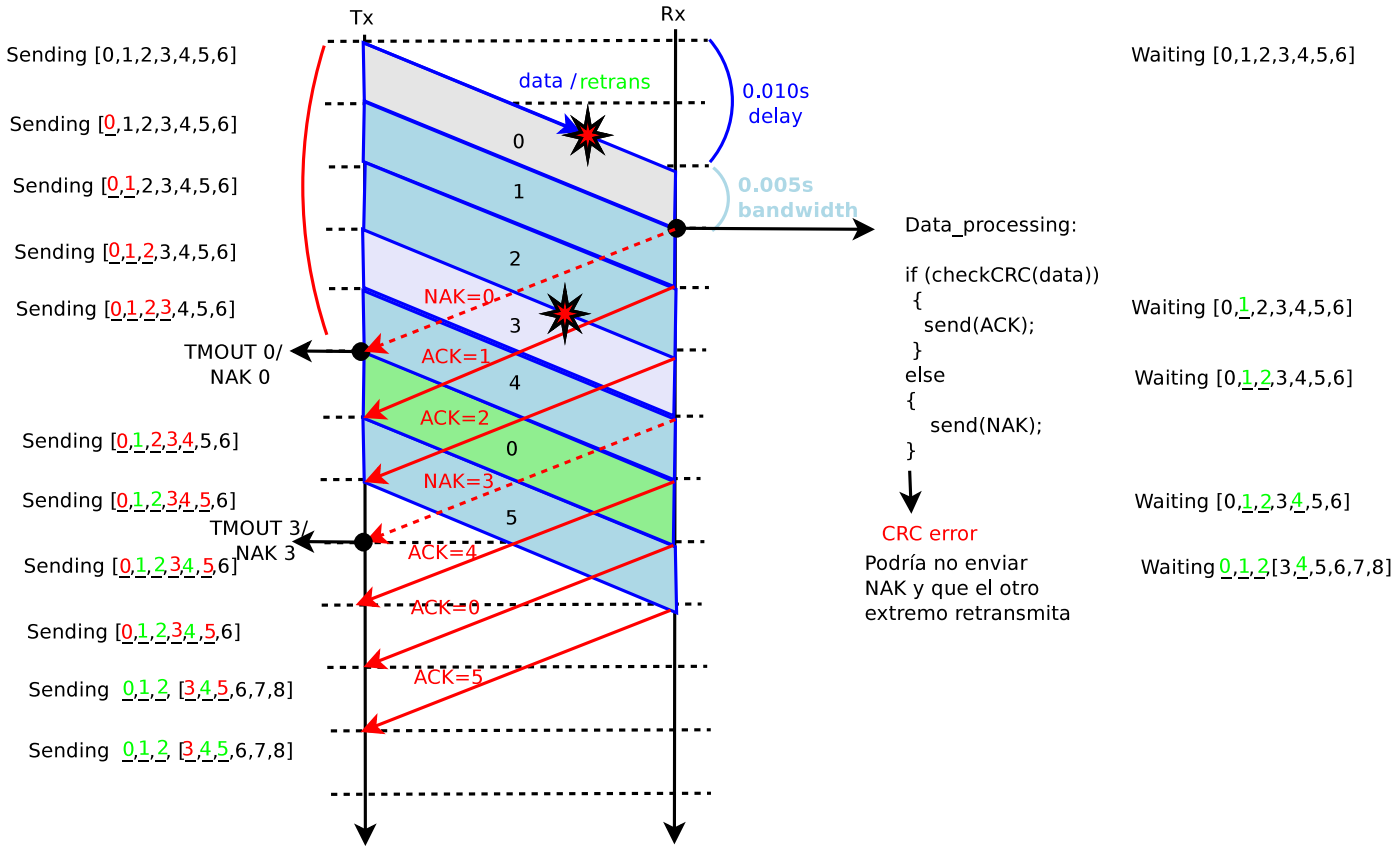
- $T2$ debe aprovechar piggy-backing, y confirmaciones acumulativas pero sin demorar demasiado tiempo el flujo de datos.
- Permite llenar el pipe y mantener el throughput, si se pierde uno se debe vaciar el pipe y volver atrás: Go-Back.
- No admite segmentos fuera de orden, ni confirmaciones fuera de orden.

## Selective Repeat (SR)

- Go-Back-N ante pérdidas retransmite segmentos innecesarios si se perdió uno del medio del stream de datos.
- Selective Sliding Window/Selective Repeat solo retransmite los que no se confirmaron.
- El receptor puede confirmar de a uno o usar bit vectors/intervalos de confirmaciones.
- No se debe confundir los segmentos de diferentes ráfagas. No se deben reusar #SEQ hasta asegurarse que tiene todos los mensajes previos o estos no están en la red.

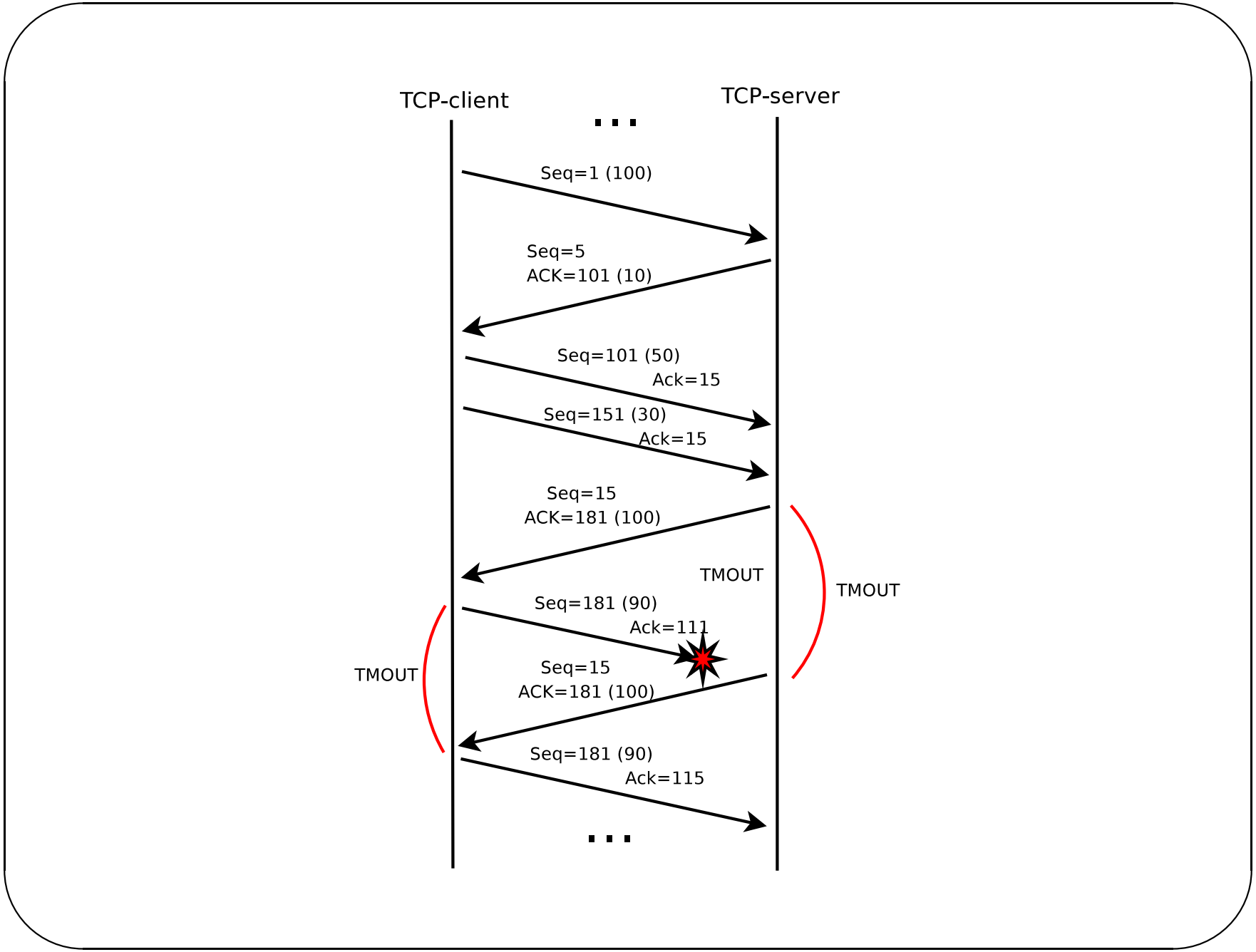
- Se realiza en módulo $M$, $K <= (M - 1/2)$, para evitar confundir los ACK de segmentos.
- La ventana se desliza sin dejar huecos, desde los confirmados más viejos.

# Selective Repeat (Cont'd)



## Control de Errores en TCP

- TCP hace el control de flujo por bytes (byte oriented), no por segmentos.
- Los segmentos se numeran de acuerdo a bytes enviados (nro. del primer byte).
- Las confirmaciones son “anticipativas”, indican el nro. de byte que esperan.
- Utiliza Go-back-N con ventana dinámica (flow-control), utiliza piggy-backing y permite negociar Ventana Selectiva con Opciones.
- TCP utiliza los campos: #SEQ, #ACK, flag ACK.


TCP-client
TCP-server
Seq=1 (100)
Seq=5
ACK=101 (10)
Seq=101 (50)
Ack=15
Seq=151 (30)
Ack=15
Seq=15
ACK=181 (100)
TMOUT
TMOUT
Seq=181 (90)
Ack=111
TMOUT
Seq=15
ACK=181 (100)
Seq=181 (90)
Ack=115

## Control de Errores TCP (Cont'd)

- Por cada segmento TCP (con datos) que envía TCP inicia un Timer local, $RTO$ (TMOUT) y pone copia del segmento en cola local (RFC-793) $TxBuf$.
- Por cada segmento ACKed descarta el timer asociado y descarta la copia del segmento (RFC-793) del $TxBuf$. Permite hacer lugar para nuevos segmentos a Tx.
- Si $RTO$ expira antes que se confirme el segmento TCP lo copia del $TxBuf$ y retransmite (RFC-793).
- Segmentos ACKed no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el $RxBuf$ del receptor).

- Si el receptor detecta error en el segmento simplemente descarta y espera que expire $RTO$ en el emisor (podría envía un NAK, re-enviar ACK para el último recibido en orden, forma de solicitar lo que falta).
- Receptor con segmentos fuera de orden descarta directamente y podrá re-enviar ACK (podría dejar en $RxBuf$ pero no entregar a la aplicación, tiene huecos).

## Control de Errores TCP (Cont'd)

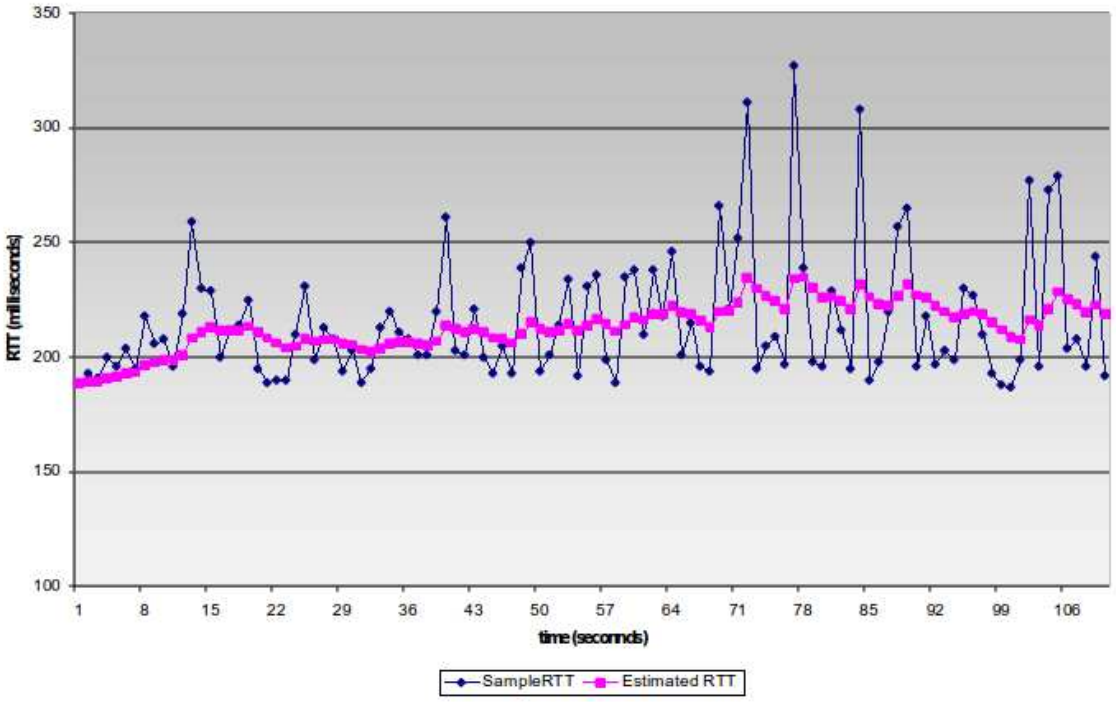
- Se puede confirmar con ACK acumulativos.
- TCP NO arrancar un $RTO$ por cada segmento, solo mantiene un por el más viejo enviado y no ACked y arranca uno nuevo solo si no hay $RTO$ activo.
- Si se confirman (ACked) datos, se inicia un nuevo $RTO$ (RFC-6298) recomendado.
- El nuevo $RTO$ le esta dando más tiempo al segmento más viejo aún no confirmado.
- Si vence un RTO se debe retransmitir el segmento más viejo no ACKed y se debe doblar: Back-off timer $RTO = RTO * 2$ $RTO_{MAX} = 60s$ (RFC-6298) recomendado.

## Cálculo de RTO

- $RTO$ debe ser dinámico, debe contemplar estado de la red.
- $RTO$ estático solo sirve para L2 (directamente conectados).
- Para calcular $RTO$ se estima RTT (Round Trip Time). RTT inicial RFC-2988(2000), 3seg - RFC-6298(2011), 1 seg. Cambio en las redes.
- $RTO = SRTT + (4 * DevRTT)$ (RFC-6298).
- $SRTT_i = (1 - \alpha) * SRTT_{i-1} + \alpha * RTT, \alpha = 1/8$
- Influencia de las muestras pasadas decrece exponencialmente.
- $DevRTT_i = (1 - \beta) * DevRTT_{i-1} + \beta * |RTT - SRTT_i|, \beta = 1/4$
- Si hay gran variación en $SRTT_i$ se usa un mayor margen.

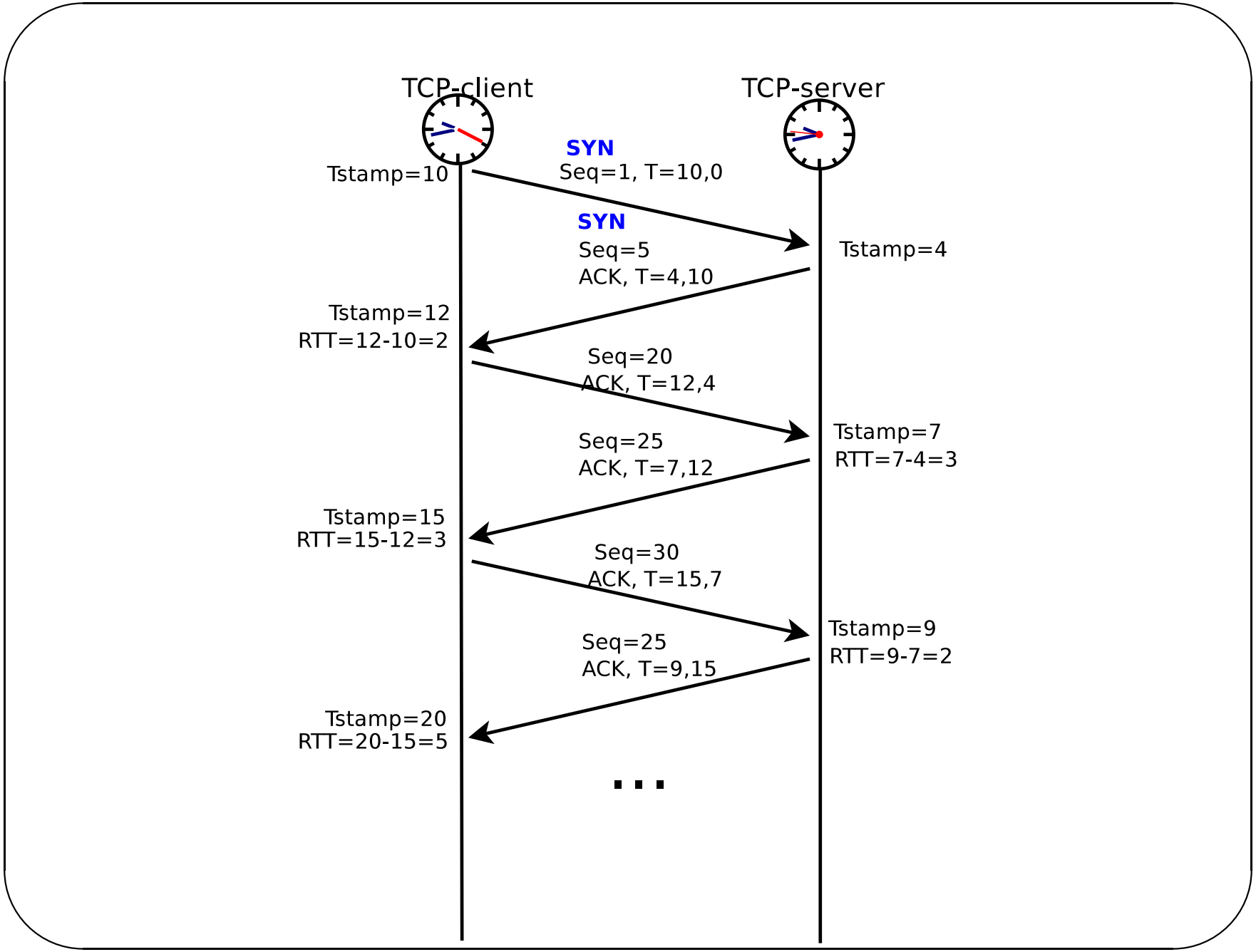
- $RTO < 1seg$ : redondeado a 1 seg (RFC-6298).
- Se mide por cada RTT. Se puede utilizar la opción TimeStamp.

# SRTT


350
300
250
200
150
100
RTT (milliseconds)
1
8
15
22
29
36
43
50
57
64
71
78
85
92
99
106
time (seconnds)
SampleRTT
Estimated RTT

## TimeStamp

- RFC-1323.
- Se envía en el primer SYN el timeStamp local.
- En cada mensaje TCP con esta opción, se copia el timeStamp local y se hace echo del último timeStamp recibido desde otro extremo.
- Con el valor recibido como echo y el valor del reloj local se calcula el RTT.
- Si el mensaje no es un ACK válido no se actualiza la estimación del RTT $SRTT$.
- Relaja la necesidad de usar timer por cada segmento para estimar RTT.


TCP-client
TCP-server
SYN
Seq=1, T=10,0
Tstamp=10
SYN
Seq=5
ACK, T=4,10
Tstamp=4
Tstamp=12
RTT=12-10=2
Seq=20
ACK, T=12,4
Tstamp=7
RTT=7-4=3
Seq=25
ACK, T=7,12
Tstamp=15
RTT=15-12=3
Seq=30
ACK, T=15,7
Tstamp=9
RTT=9-7=2
Seq=25
ACK, T=9,15
Tstamp=20
RTT=20-15=5
. . .

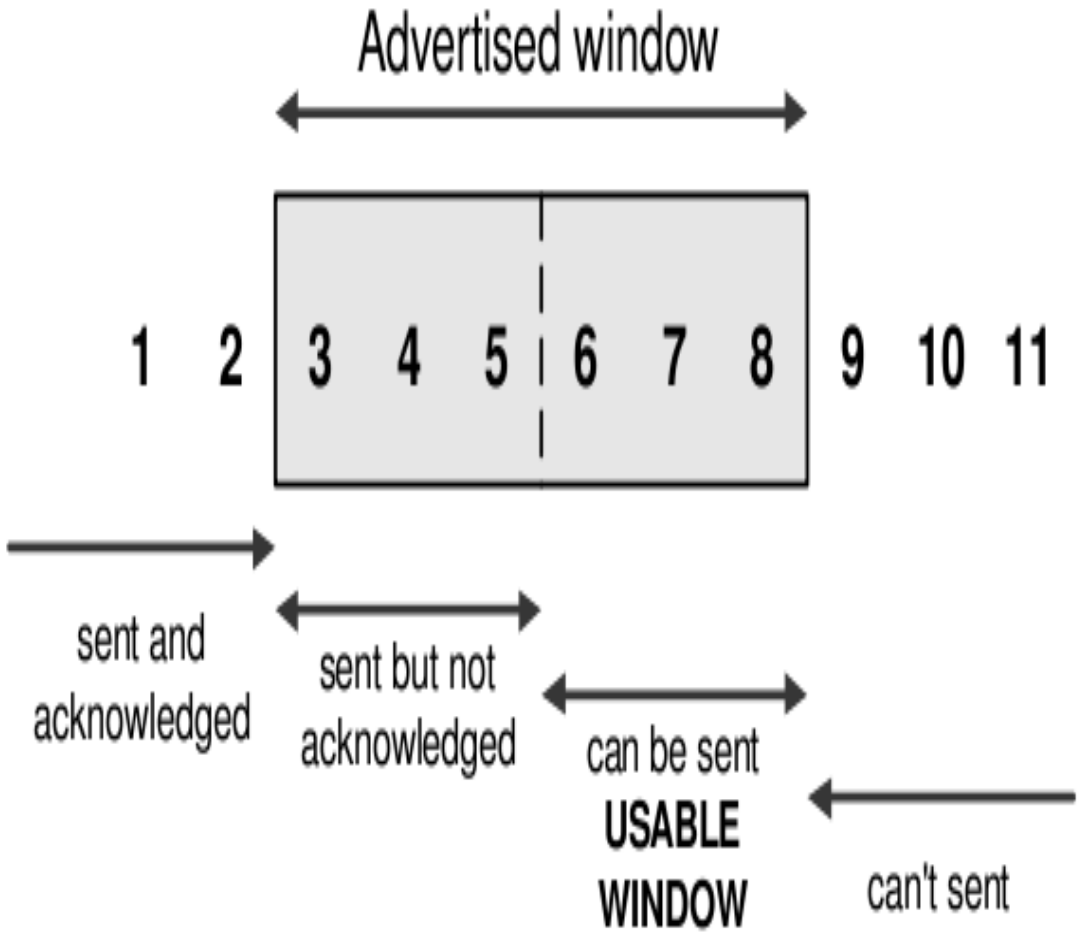
## Control de Flujo TCP

- Por cada segmento que envía indica el tamaño del buffer de recepción RxBuf (mbufs). Cada conexión mantiene su propio buffer) en espacio del kernel (TCP): WIN (Ventana).
- WIN (Ventana) indica la cantidad de datos que el emisor le puede enviar sin esperar confirmación (mejora notablemente contra Stop & Wait.
- La ventana de recepción de cada extremo es independiente.
- Cada vez que llega un segmento es puesto por TCP en el $RxBuf$, TCP lo debe confirmar.
- Cada vez que la aplicación lee se hace espacio en el $RxBuf$. Se va modificando el tamaño de la ventana.

# Ventana Deslizante TCP

- Ventana de Recepción $Rcv\_win = Win - Sent.No.ACked$.
- El receptor "ofrece" la ventana $Win$ en los segmentos TCP.
- El transmisor no puede enviar más de la cantidad de bytes en $Win - Sent.NO.ACKed$.
  - Al recibir ACKs de TCP (App. no lee aún) se cierra ventana.
  - Al recibir ACKs y $Win$ fijo desliza ventana (App. lee a rate fijo).
  - Al achicarse $Win$ se reduce ventana (App, no lee).
  - Al agrandarse $Win$ tiene posibilidad de enviar más (App. lee más rápido).
  - Tamaño de ventana seleccionado por el kernel o por aplicación `setsockopt()`.

# Ventana Deslizante TCP (Cont'd)



## TCP Bulk y TCP Interactivo

- Delayed ACKs: No enviar ACK sin esperar de enviar datos antes: piggy-back (200ms, MAX=500ms).
- Algoritmo Nagle: No enviar datos en chunks pequeños, esperar juntar información.
- Perjudica aplicaciones interactivas.
- Tinygrams: segmentos chicos: App. interactivas, $Win$ casi vacía.
- Silly Window: $Win$ casi llena se “ofrecen” pequeños incrementos. Solución: Muestra incrementos de $Min(MSS, RecvBuf)/2$.

# Control de Congestión TCP

- El Control de Congestión es otro servicio ofrecido por TCP.
- Controlar el tráfico que se envía evitando que se colapse la red y se descarte teniendo que retransmitirse. Tiene en cuenta el estado de la red, no solo el buffer del receptor.
- Se puede implementar End-to-End (caso TCP, RFC-5681).
- O tomando partida la red, Modelo basado en la Red (IP+TCP:RFC-3168, Frame-Relay, ATM).
- Problemas de Delay en los routers, problemas de overflow y descarte.

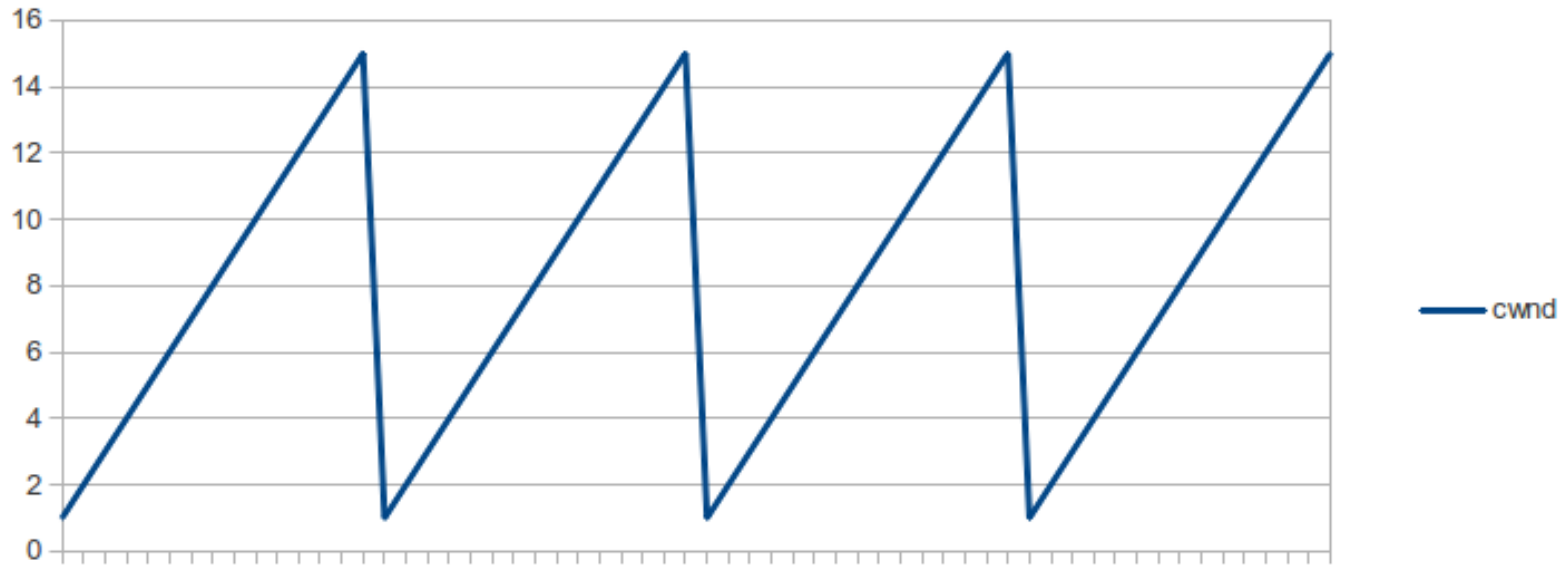
## Control de Congestión TCP (Cont'd)

- Se utilizan nuevas variables:
  - $cwnd$ Ventana de congestión. tiene en cuenta el estado de la red.
  - $ssthresh$ Slow Start Threshold (Umbral).
  - $can\_send = Min(Rcv\_win, cwnd)$.
- Diferentes Fases:
  - Si $cwnd < ssthresh$: Fase de crecimiento inicial: SS (Slow Start).
  - Si $cwnd >= ssthresh$ Fase de mantenimiento: CA (Congestion Avoidance).

## Control de Congestión TCP (Cont'd)

- 1era. Versión TCP CC: Old Tahoe TCP (BSD 4.3).
  - Ventana de congestión: $cwnd$ crece hasta que se resetea, buffer overflow o error.
  - El destino podría limitarla con $Win$: válido para LAN.
  - No utiliza Slow Start (SS), se incrementa $cwnd++$ (en MSS) por cada RTT.
  - ACK perdido o segmento erróneo deriva en $cwnd = 1$.
  - Congestión detectada por RTO.

# Control de Congestión TCP (Old Tahoe)



# Control de Congestión TCP (Cont'd)

- TCP Tahoe (BSD 4.3 - 1988).
  - Utiliza Slow Start (SS), ventana crece exponencialmente, inicio $cwnd = 1 * MSS$.
  - Una vez que se alcanza $sstresh$ se trabaja con Congestion Avoidance (CA).
  - Valor inicial $ssthresh = \infty$ (un valor alto).
  - También agrega Fast Retransmit, Mejor implementada en Reno.
  - Congestión detectada por RTO o 3DUP ACKs, derivaba en ambos casos $ssthresh = Min(cwnd/2, 2) * MSS$ y Slow Start.

## Control de Congestión TCP (Cont'd)

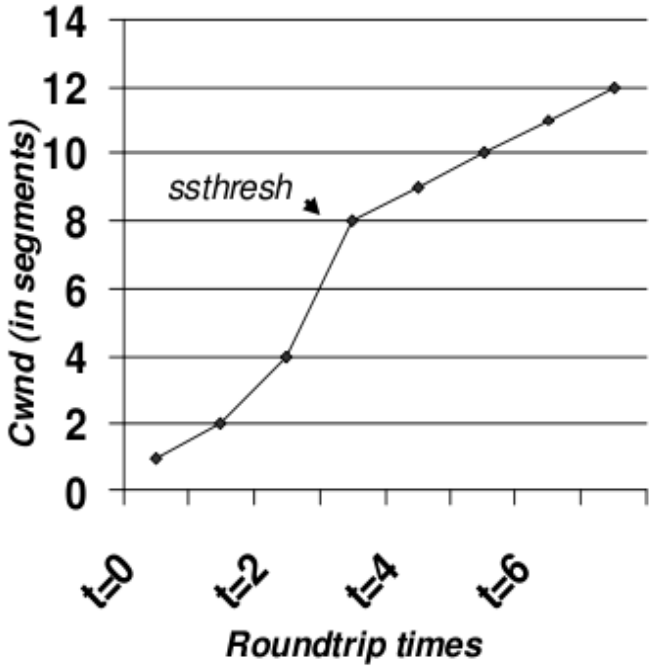
- Slow Start:
  - Crece exponencialmente, de forma rápida.
  - Inicia $cwnd = 1 * MSS$ (a veces se usa 2 o 3). Transmite y espera ACK.
  - ACK recibido $cwnd++$ (en MSS) $cwnd = 2 * MSS$, nuevos ACKs recibidos $cwnd = 4 * MSS$ ...
  - Si destino retarda ACK no se cumple.
  - Incrementa $cwnd++$ (en MSS) por cada ACK (varios por RTT).

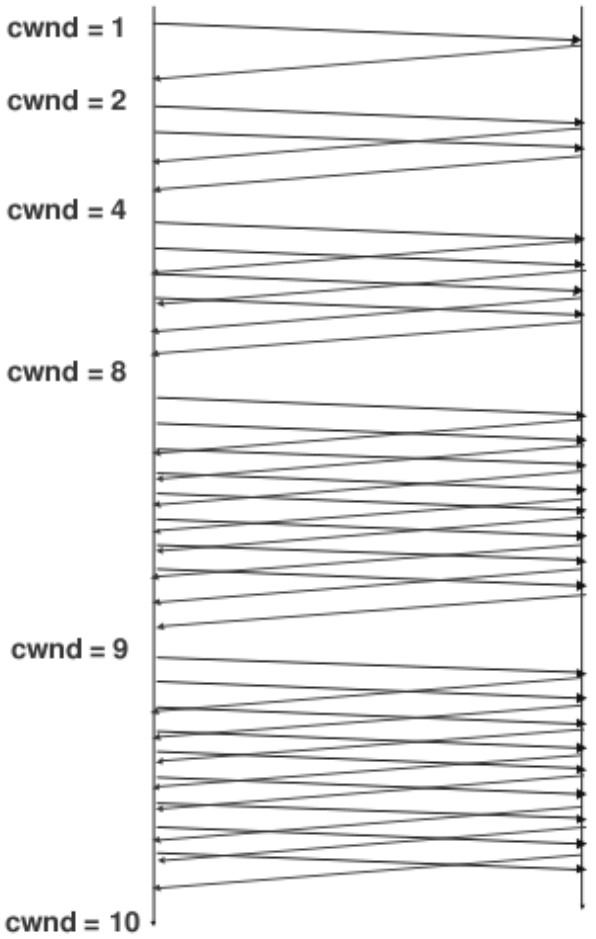
## Control de Congestión TCP (Cont'd)

- Congestion Avoidance:
  - Una vez que $cwnd >= ssthresh$ crece de forma lineal.
  - Incrementa $cnwd++$ por cada RTT.
- Fast Retransmit: en Tahoe seguido por Slow Start. Vuelve a 0 (cero).

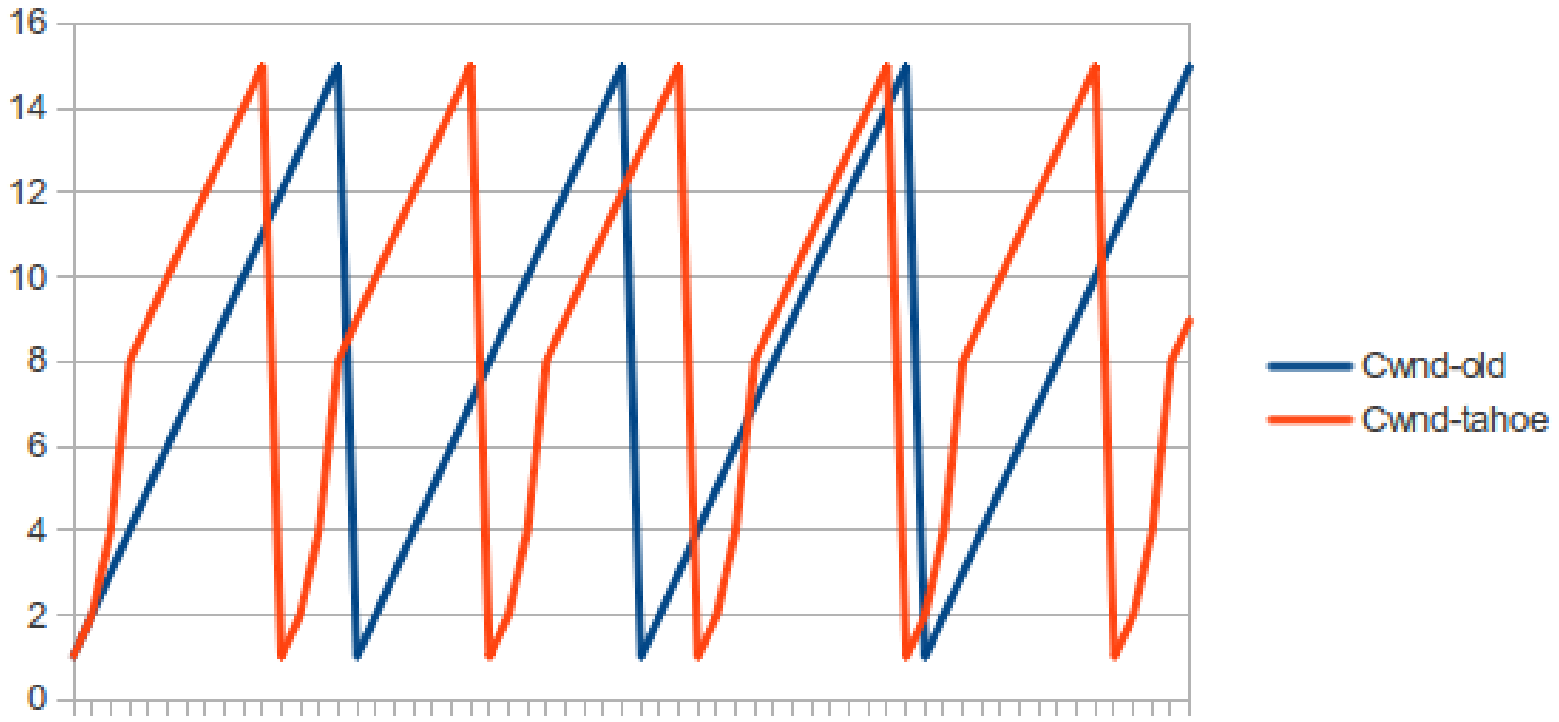
# Control de Congestión TCP (Cont'd)

Assume that *ssthresh = 8*



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



## Control de Congestión TCP (Cont'd)

- TCP Reno (BSD 4.3 - 1990) RFC-2581/RFC-5681.
  - Slow Start.
  - Congestion Avoidance.
  - Fast Retransmit.
  - Fast Recovery.

## Control de Congestión TCP (Cont'd)

- Fast Retransmit:
  - 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 llego 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 sin esperar RTO.

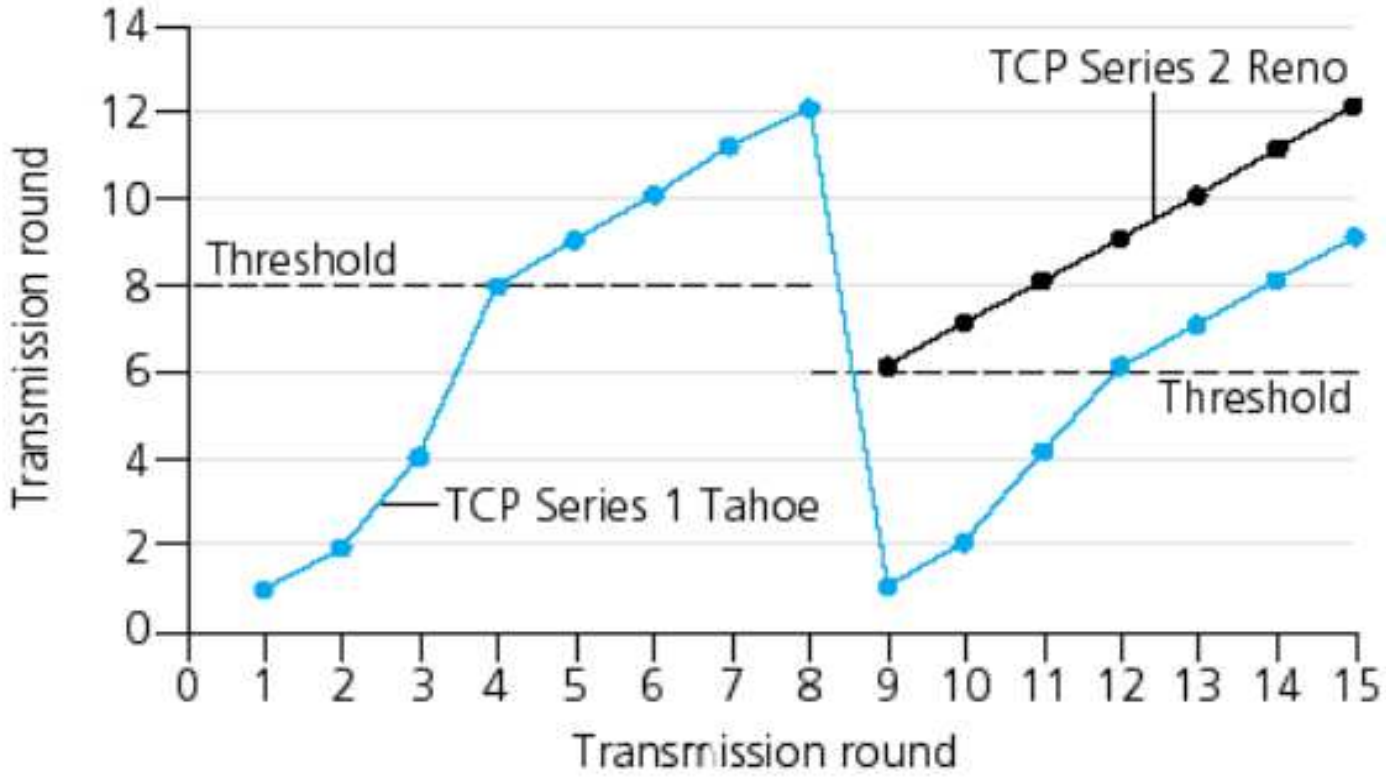
## Control de Congestión TCP (Cont'd)

- 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 Fast Recovery, crece lineal.
  - Luego de Fast Recovery (se ACked el segmento perdido) se realiza CA(Reno), no SS(Tahoe).

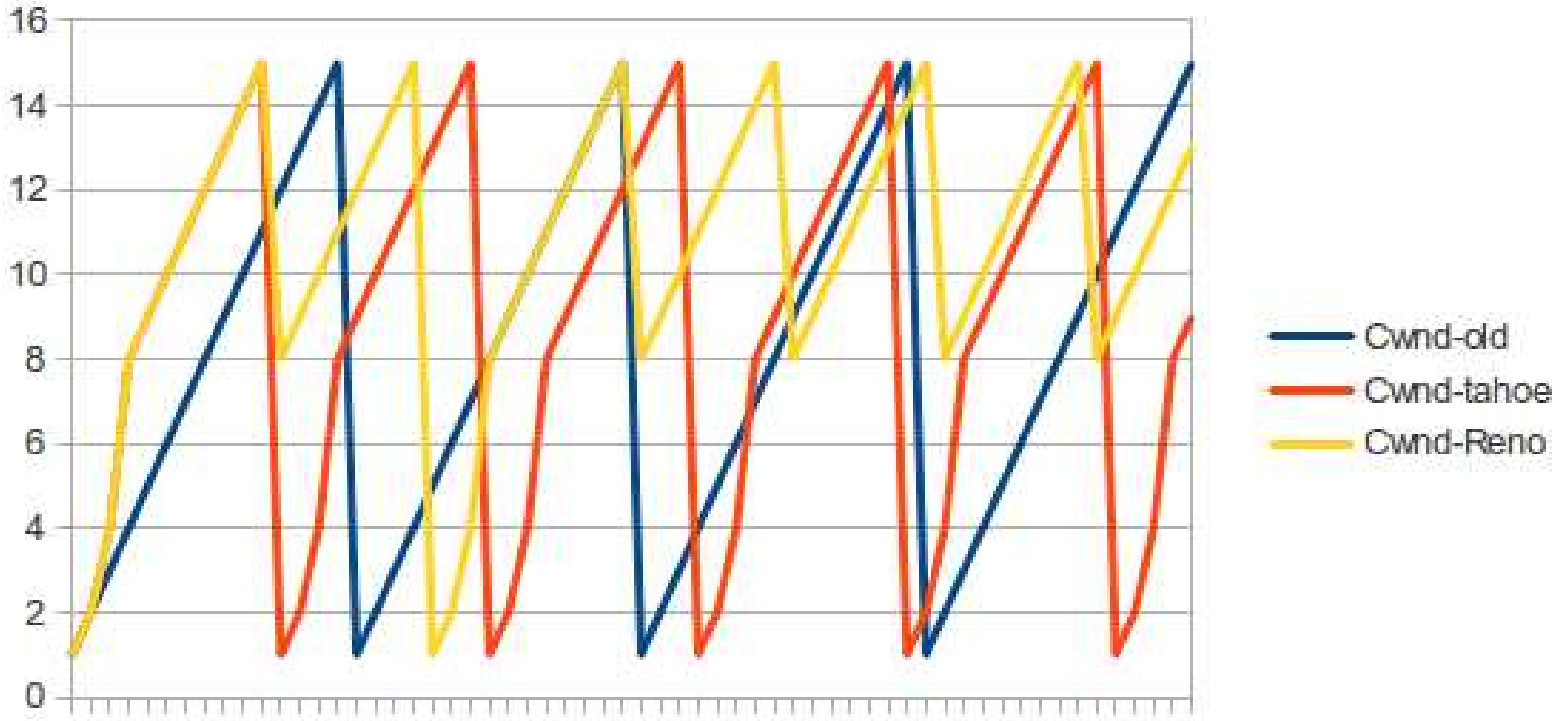
## Control de Congestión TCP (Cont'd)

- Si se detectan 3DUP ACKs (Fast Retransmit):
  1. $ssthresh = Min(cwnd/2, 2) * MSS$
  2. 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 $cwnd$ y comienza con CA.

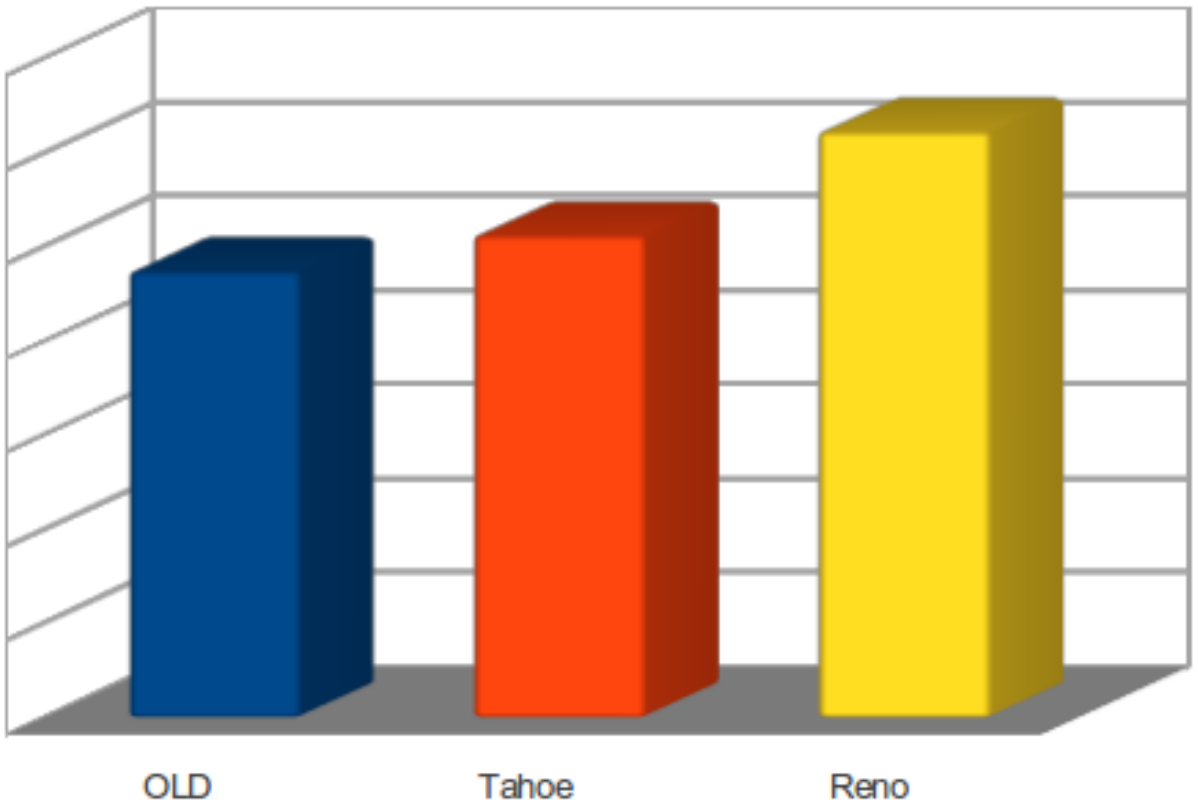
# Control de Congestión TCP (Cont'd)



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



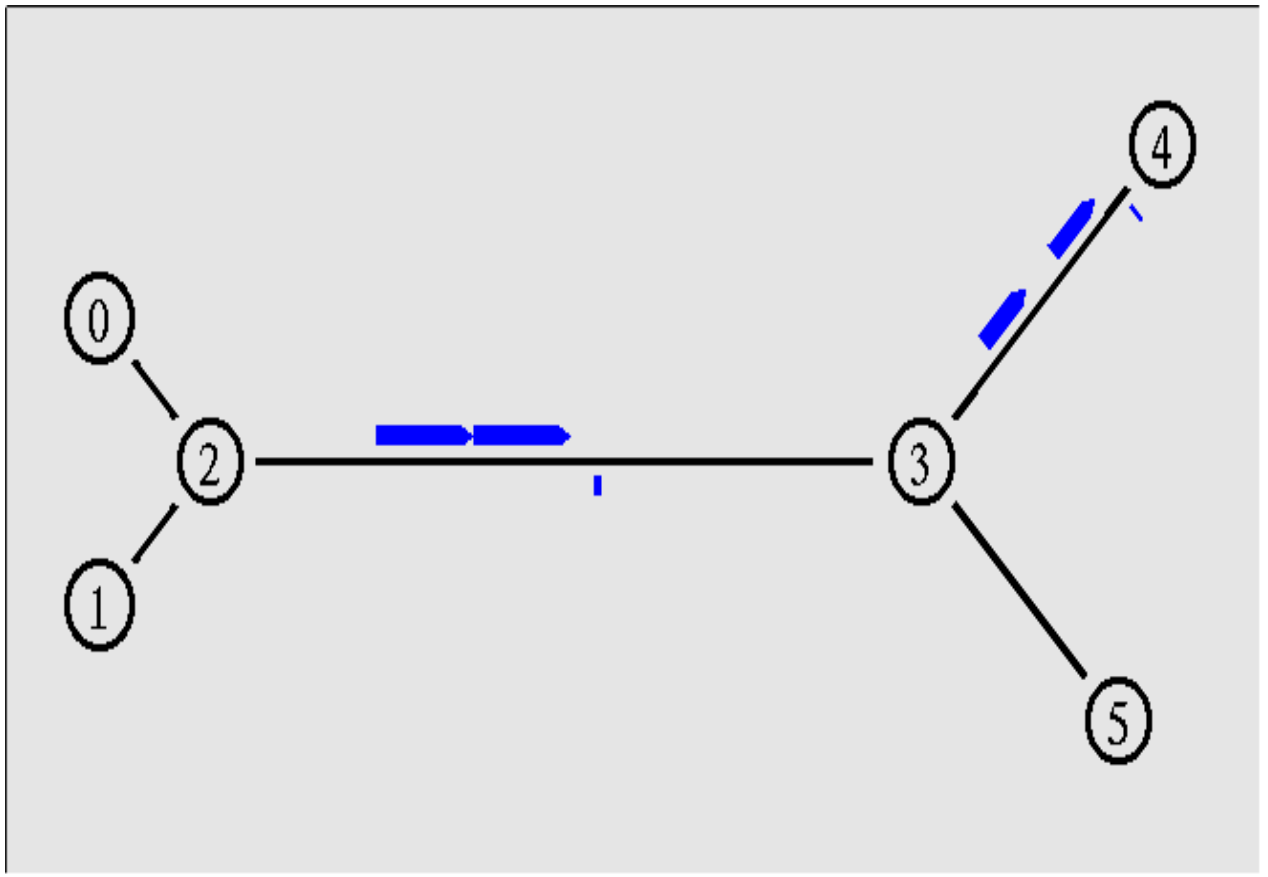
# Control de Congestión TCP (Old vs. Tahoe vs. Reno)



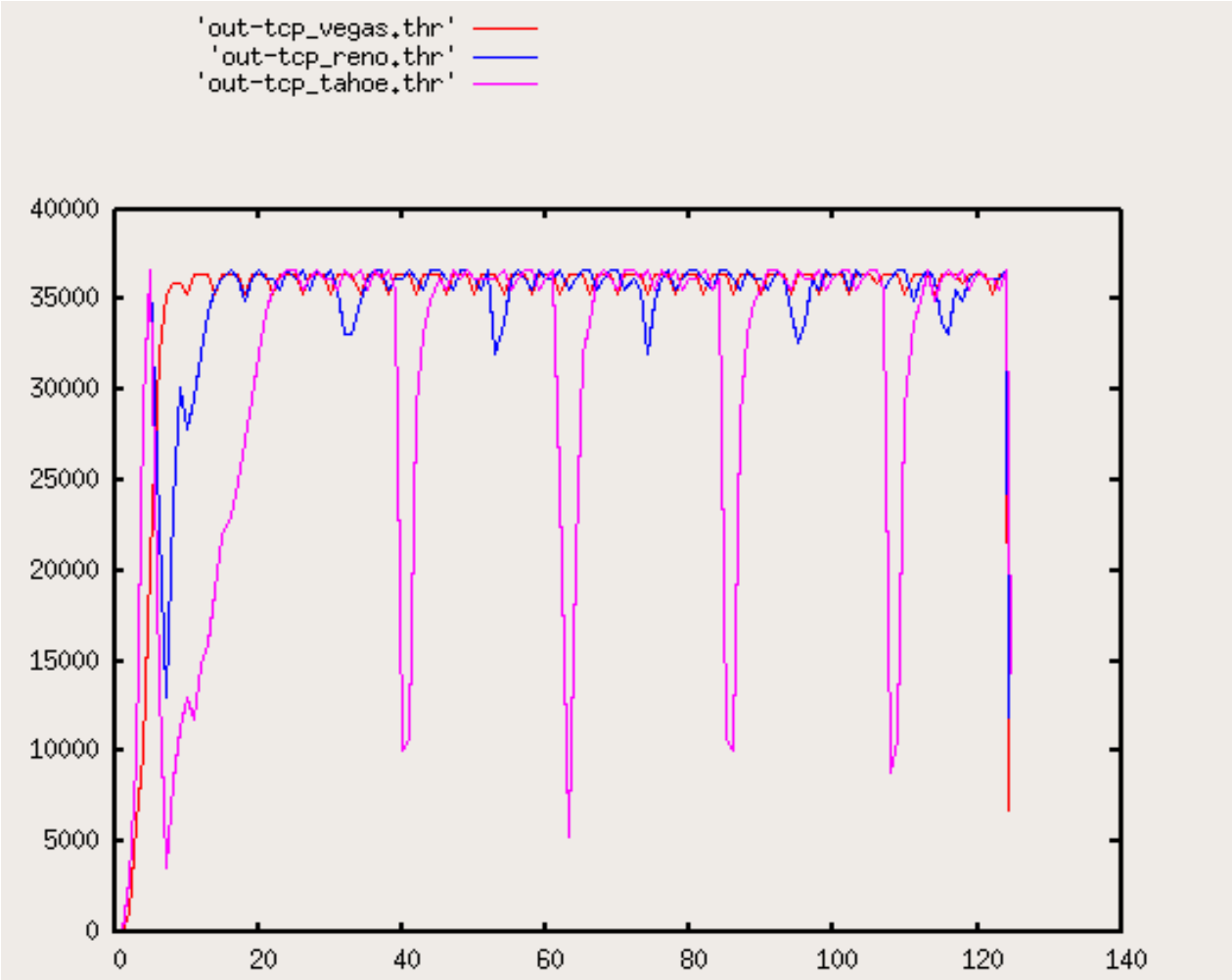
## Otras Implementaciones TCP

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

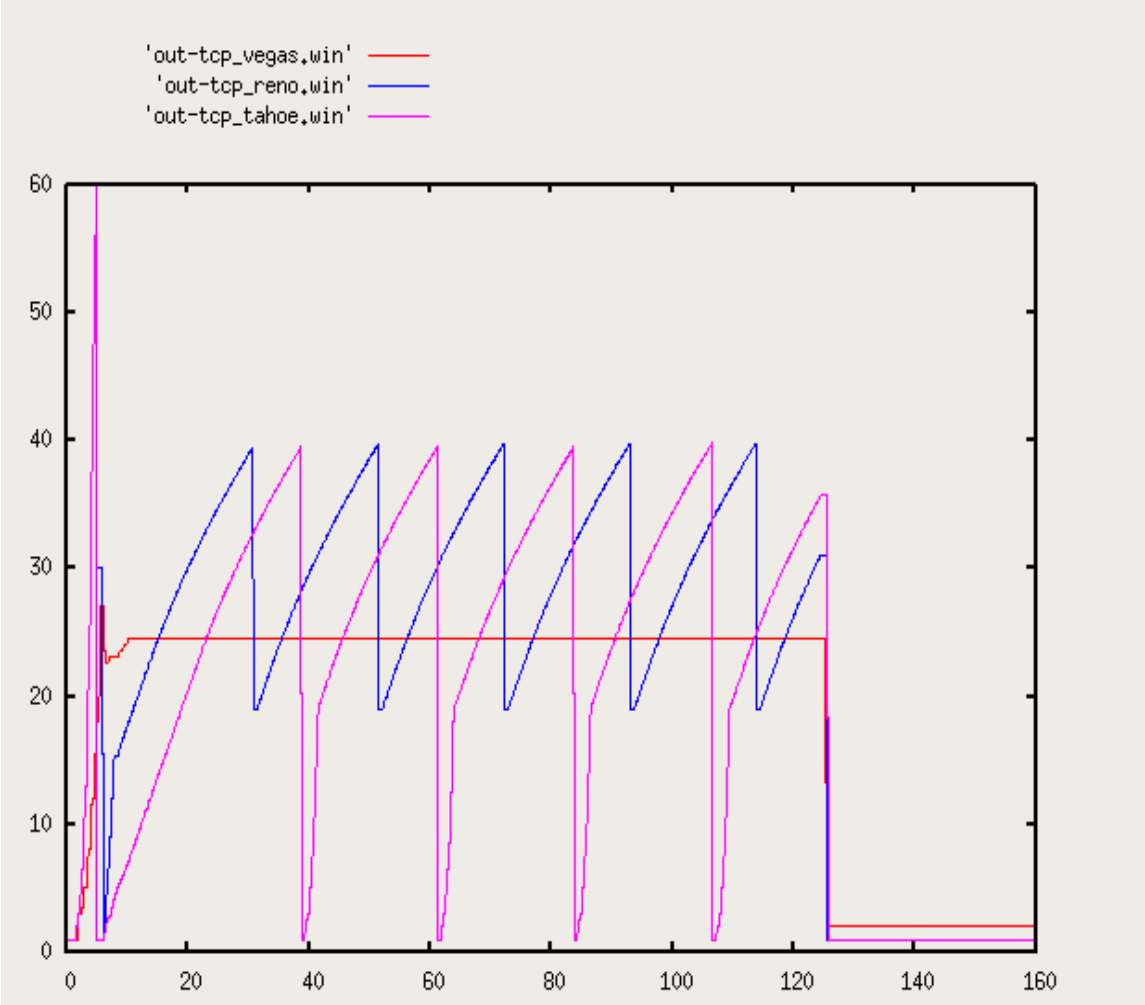
# Implementaciones Modelo



# Implementaciones (Comparar Throughput)



# Implementaciones (Comparar cwnd)



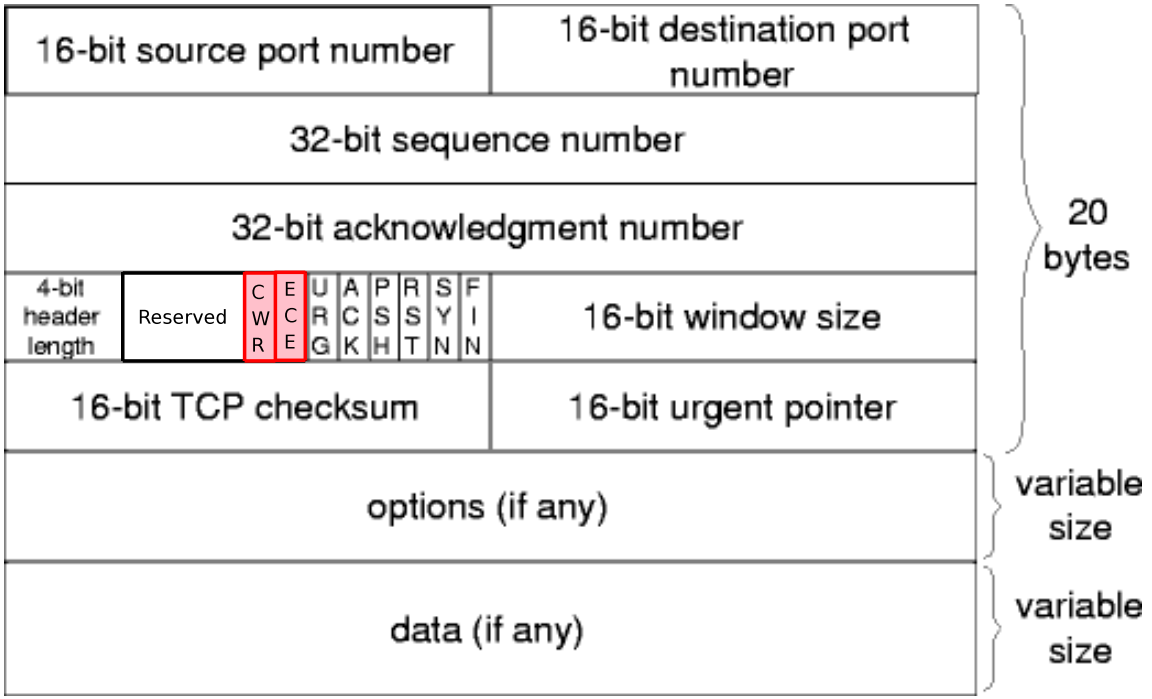
## Control de Congestión por la Red

- 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.

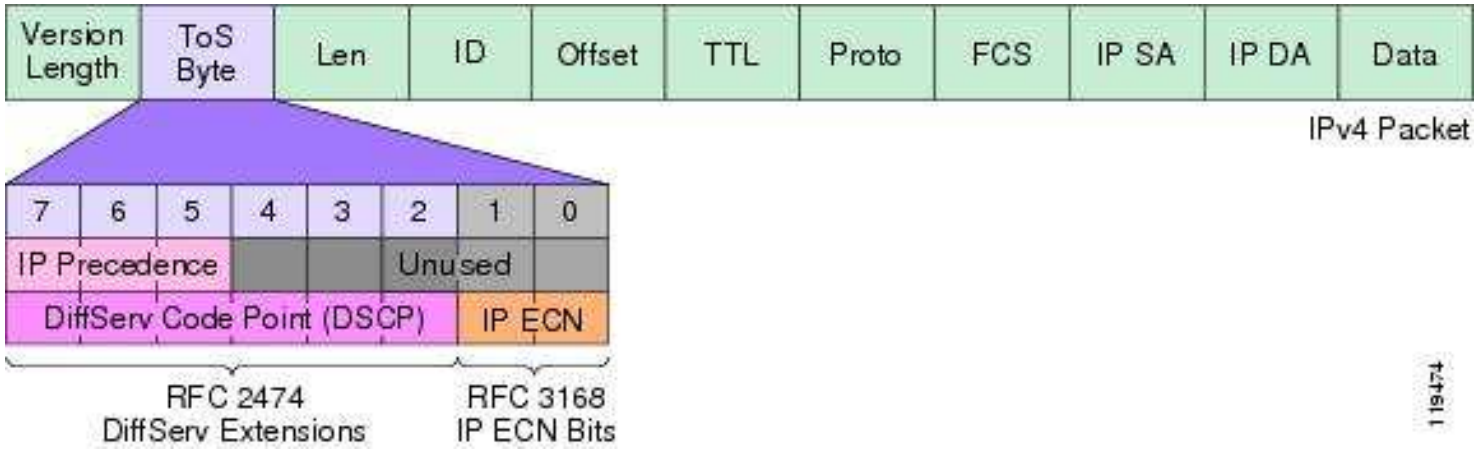
## Control de Congestión por la Red (Cont'd)

- 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



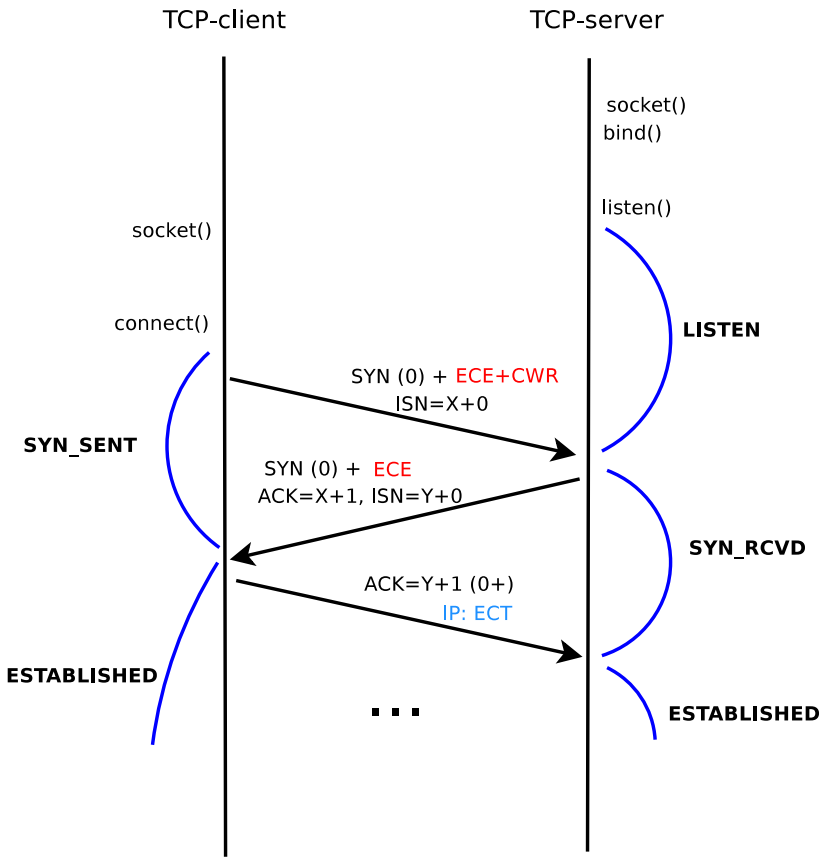
# Datagrama IP Marcado con ECN: ECT, CE



## 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).

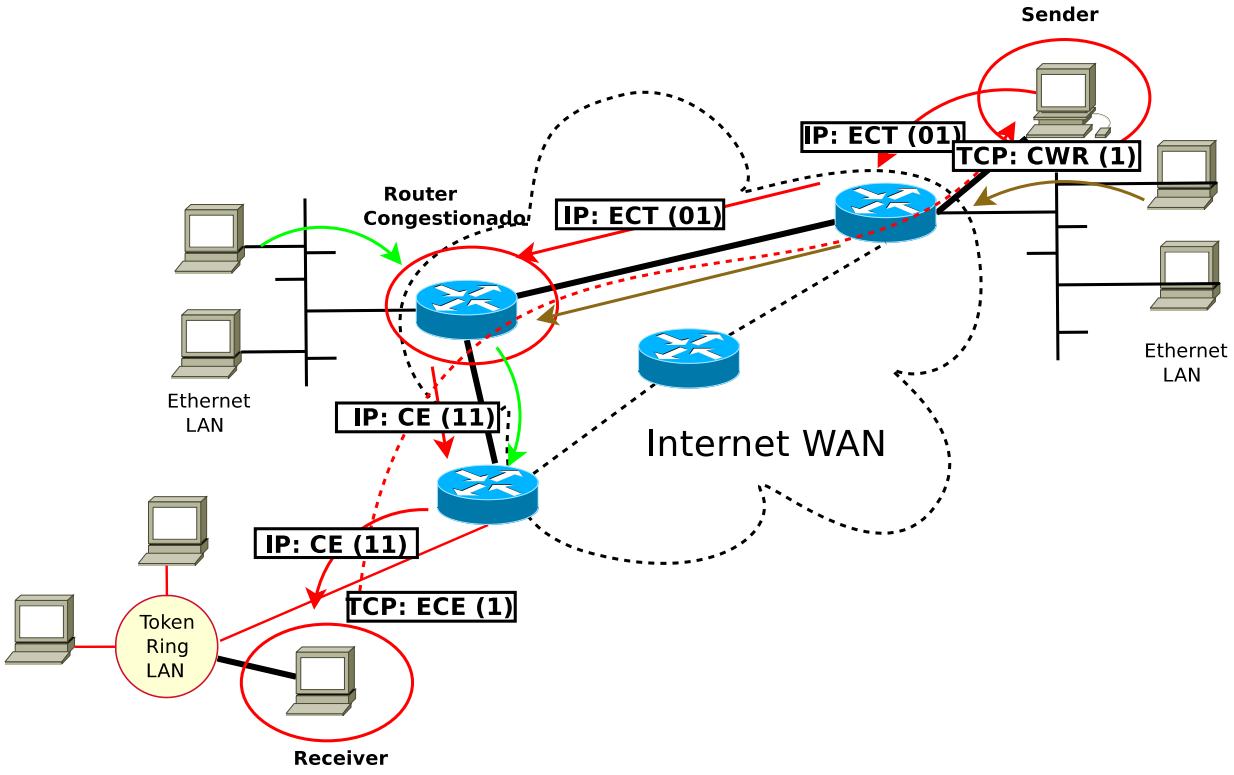
# TCP ECN Setup



## 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.
- 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.
- Wikipedia http://www.wikipedia.org.
- Slides de la Prof. Paula Venosa.
- TCP/IP Guide: http://www.tcpipguide.com/.

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