

---

# Control de Congestión en TCP

**Control de Congestión en Redes TCP – Ing. Pablo Ronco**

Universidad de Buenos Aires  
Facultad de Ingeniería - Diciembre 2007



# Congestión

---

Qué es Congestión y por qué se produce?

Controlar / Evitar

Control de Flujo / Control de Congestion

TCP - Control de Congestion en TCP

TAHOE, RENO, NewRENO, D-SACK, VEGAS, BIC, CUBIC  
ECN, AQM

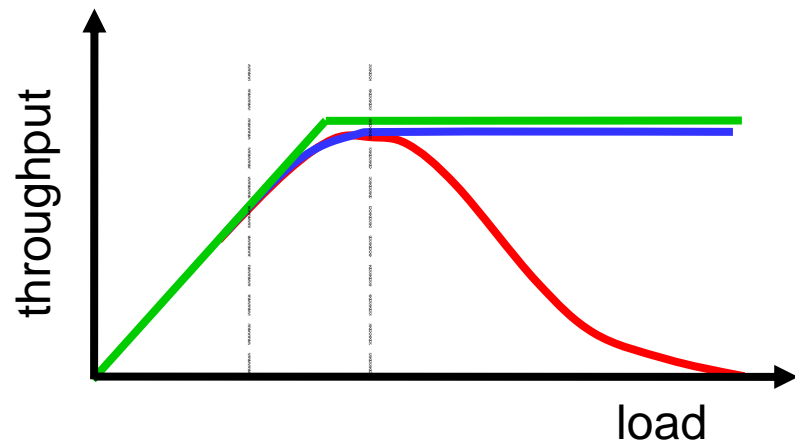
# Que es “Congestión”

**Congestión** es la condición que se produce cuando la cantidad de paquetes siendo transmitidos **se aproxima** a la capacidad de transporte de la red.

Congestión suave: Existe tiempo de espera en cola

Congestión severa: Existe pérdida de paquetes.

Control de Congestion es el mecanismo para prevenir estos problemas. Regulando la tasa de transferencia por debajo de un cierto nivel a partir del cual la performance cae estrepitosamente



# Por que se produce?

---

Baja capacidad de los enlaces: La tasa de arribo de paquetes excedo o al menos se aproxima a la capacidad de salida del enlace. Se encolan paquetes.

Procesador lento: La tasa de arribo de paquetes excede la capacidad de ruteo de la Red. Se encolan paquetes.

Memoria insuficiente. Los buffers de recepción no son lo suficientemente grandes para almacenar los paquetes que arriban. Se descartan paquetes.

Naturaleza estadística del tráfico: Tráfico de ráfagas de paquetes de tamaño variable.



# Controlar o Evitar la Congestión?

---

## Estrategia de TCP

Controlar congestión una vez que esta ocurre

Repetidamente incrementar la carga en un intento de encontrar el punto al cual la congestión ocurre y luego retroceder

## Estrategia Alternativa

Predecir cuando la congestión está a punto de ocurrir

Reducir la tasa antes que empiecen a descartarse paquetes



# Como se obtienen conclusiones?

---

No hay simulación ni medición exhaustiva – No se puede probar la eficiencia de un protocolo o mecanismo – Pero se puede demostrar los problemas de cada implementación.

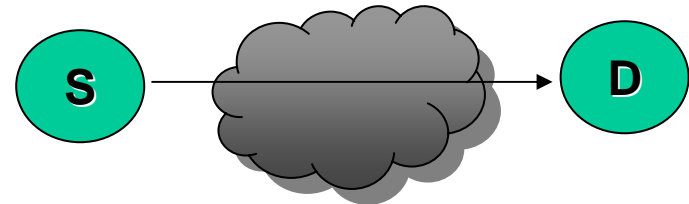
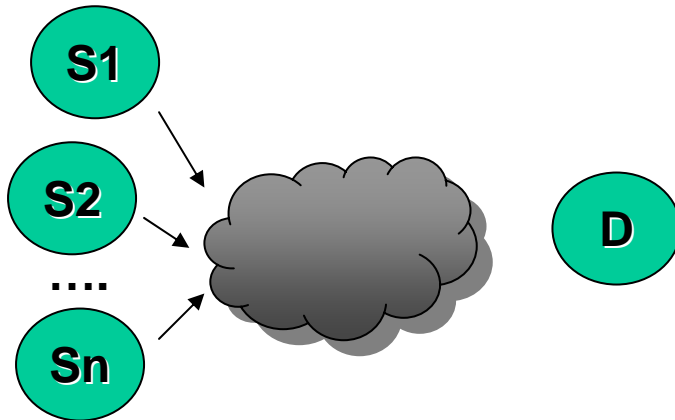
- **Fairness** (entre flujos similares)
- **Friendliness** (contra el TCP nativo)
- **Efficiency** (uso de la capacidad disponible de la Red).
- **Responsiveness** (Velocidad de respuesta a los cambios en las condiciones de la Red, ej. nuevos flujos iniciando o cerrando)



# Control de Congestión vs. Control de Flujo

- Control de congestión: interviene cuando la Red se encuentra congestionada o sobrecargada.
- Debido al flujo entre los dispositivos origen y la Red.
- Conocido como “Control de Flujo de Origen”

- Control de Flujo: Interviene cuando el receptor se encuentra saturado.
- Debido a no poder procesar los paquetes que arriban a la velocidad que arriban.
- Conocido como “Control de Flujo de Destino”



# TCP Transmission Control Protocol

---

TCP provee un servicio de transmisión confiable, orientado a la conexión, de flujo de bytes. TCP considera que las capas inferiores no ofrecen servicios confiables.

## Servicios de TCP

- Identificación unívoca de procesos que se comunican
- Esquema CLIENTE - SERVIDOR
- Protocolo orientado a la Conexión
- Transmisión Confiable
- Control de Flujo explícito
- Control de Congestión implícito



# Control de Congestión en TCP

---

- El origen determina el valor de **cwnd** a partir de indicios de congestión en la red
  - Usa reglamentación implícita
- Indicadores de Congestión
  - **Pérdidas**
  - Demoras
  - Marcas
- TCP asume red de “mejor esfuerzo” (routers FIFO o FQ) cada fuente determina por si misma la capacidad de la red.
- ACKs regulan el paso de transmisión. Cuando llega un ACK implica que un paquete salió de la red (*self-clocking*).
- Diversos algoritmos para calcular **cwnd**
  - Tahoe, Reno, Vegas, BIC-TCP, CUBIC



# Control de Congestión en TCP

---

## Control de Flujo en el Receptor

Evitar sobrecargar al destino

Explícitamente establecido por el destino

**awnd:** (advertised) window

## Control de Flujo en el Origen

Evitar sobrecargar a la red

Establecido por quien transmite

A partir de estimar la capacidad disponible de la red

**cwnd:** congestion window

sea  $W = \min(\text{cwnd}, \text{awnd})$

# Algoritmos de Control de Congestión

---

- Tahoe (Jacobson 1988)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- Reno (Jacobson 1990)
  - Fast Recovery
- NewReno (Jacobson 1990)
  - Fast Recovery Mejoras
- Vegas (Brakmo & Peterson 1994)
  - Evitar congestión a partir del RTT de la red
- SACK, D-SACK (M. Mathis 1996)
  - Detección de retransmisiones innecesarias.
- BIC-TCP (L. Xu 2004)
  - Binary Increase Congestion Control
- CUBIC (I. Rhee 2007)
  - Cubic function instead of a linear window increase



# TCP TAHOE

## Slow Start

Inicialmente

$cwnd = 1$  (slow start)

$ssthresh = \text{Max. } W$  (65535)

Por cada ACK exitoso

incrementar  $cwnd$  (MSS)

$cwnd \leftarrow cwnd + 1$

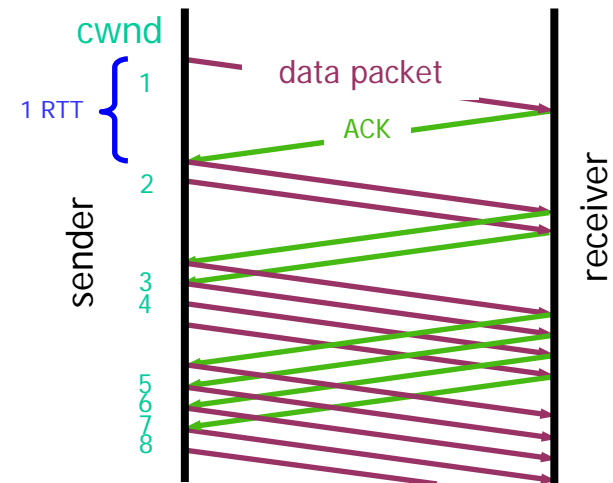
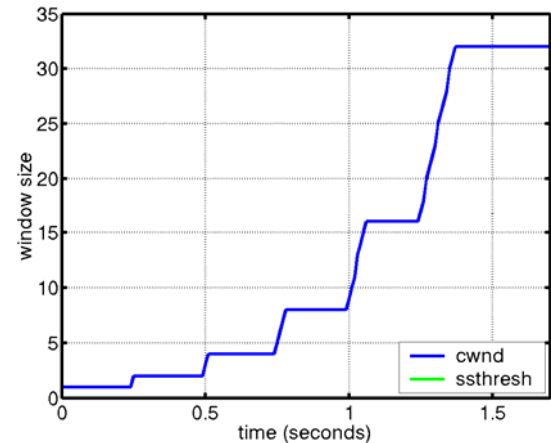
Crecimiento exponencial de  $cwnd$

por cada RTT:  $cwnd \leftarrow 2 \times cwnd$

Ingresar en **CA** cuando

$cwnd \geq ssthresh$

**CA = Congestion Avoidance**



# TCP TAHOE

## Congestion Avoidance

Comienza cuando

$$cwnd \geq ssthresh$$

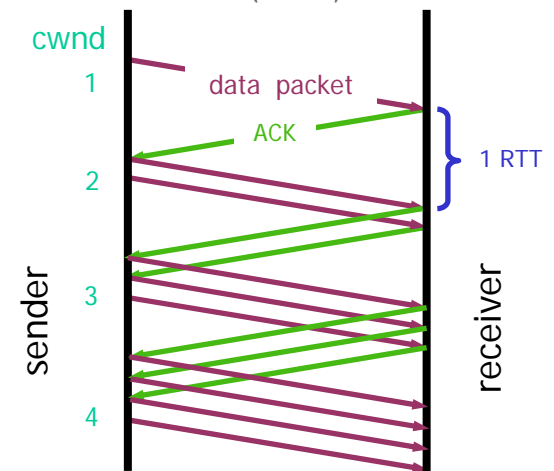
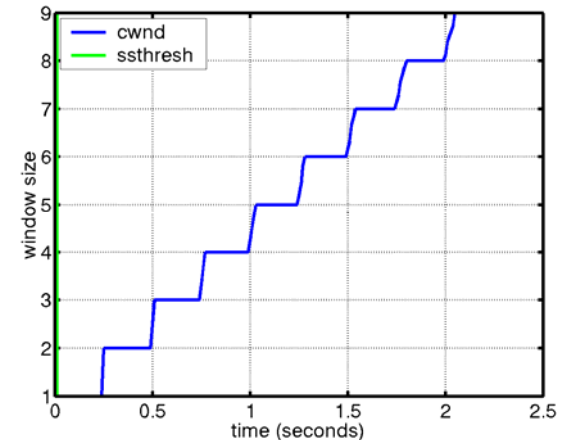
Por cada ACK correcto:

$$cwnd \leftarrow cwnd + 1/cwnd$$

Crecimiento Lineal de cwnd

1

$$\text{cada RTT: } cwnd \leftarrow cwnd + 1$$



# TCP TAHOE:

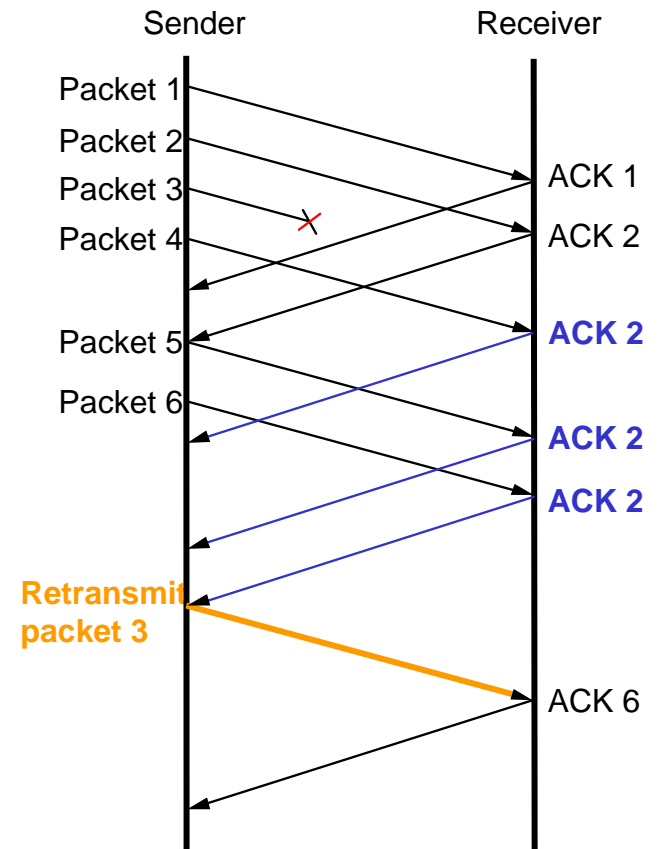
## Fast Retransmit - Detectando Congestión

**Se Asume:** Pérdida de paquetes indica congestión. Pérdida de paquetes debida a errores  $< 1\%$

La pérdida de paquetes puede ser detectado a partir:

TimeOuts de retransmisión (RTO timer)

ACKs Duplicados (al menos 3)



# TCP TAHOE:

## Fast Retransmit

---

Esperar por Timeout puede representar mucho tiempo

=> Retransmitir inmediatamente luego de recibir 3 dupACKs  
sin esperar por Timeout

Ajustar ssthresh

$\text{flightsize} = \min(\text{awnd}, \text{cwnd})$

$\text{ssthresh} \leftarrow \max(\text{flightsize}/2, 2)$

Pasar a Slow Start ( $\text{cwnd} = 1$ )

# TCP RENO:

## Fast Recovery

---

Evitar que `la tubería` se vacíe después de fast retransmit

La congestión no es tan severa: Cada dupACK indica que los paquetes continúan llegando al destino luego del paquete descartado

⇒ No entrar en Slow Start, sino que pasar a CA

Entrar en FR/FR Luego de 3 dupACKs

$$ssthresh \leftarrow \max(\text{flightsize}/2, 2)$$

Retransmitir el paquete perdido

$$cwnd \leftarrow ssthresh + ndup \text{ (agrandar la ventana)}$$

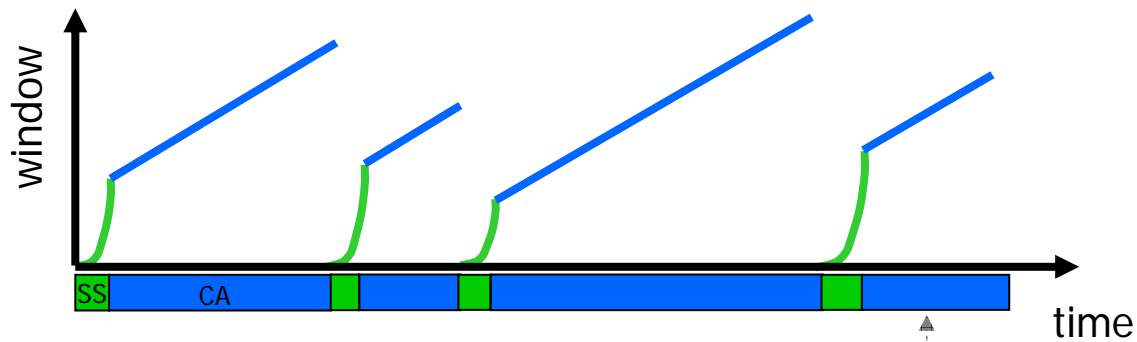
Esperar hasta que  $W = \min(\text{awnd}, cwnd)$  sea suficientemente grande y transmitir **nuevos** paquetes

Cuando no se reciban más dupACKs (1 RTT después),

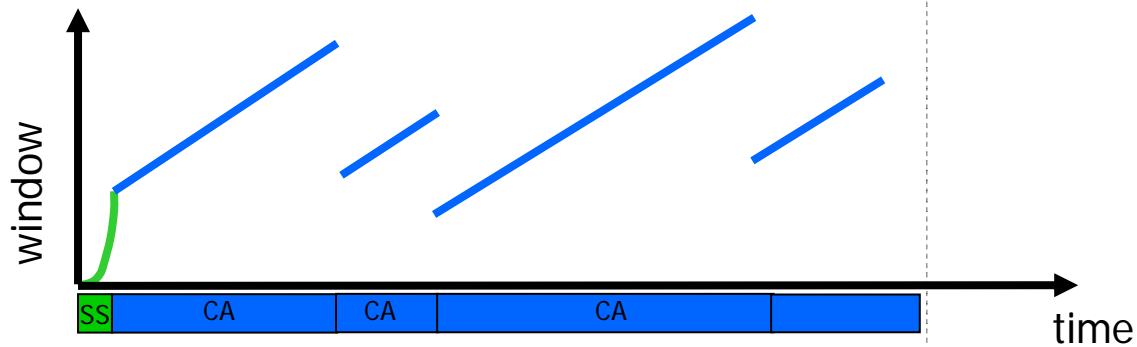
$$cwnd \leftarrow ssthresh \text{ (Reducir la ventana)}$$

Pasar a **CA**

# Tahoe vs. Reno



TCP Tahoe



TCP Reno

SS: Slow Start

CA: Congestion Avoidance

# Mejoras

---

## TCP NewReno

Partial Ack: Infiere pérdida múltiple de paquetes en una ventana.

## TCP SACK, D-SACK

SACK (Selective Ack): Usar las opciones del header TCP para informar bloques de segmentos recibidos.

Dual SACK: Extensión de SACK que indica que un bloque de datos ha sido recibido dos veces.



# TCP NewReno

---

Detectar pérdidas múltiples dentro de una ventana de transmisión

**Partial ACK** confirman la recepción de algunos pero no de todos los paquetes pendientes desde el inicio de FR

**Partial ACK** hace que TCP Reno salga de FR, y reduzca su ventana

El origen deberá esperar un timeout antes de retransmitir

⇒ Cuando se reciben ACK parciales:

Se permanece en FR/FR y se retransmite inmediatamente

Se retransmite 1 paquete perdido por RTT hasta que **todos** los paquetes perdidos se hayan retransmitido

⇒ De esta forma se eliminan los timeouts

Nota: La implementación de NewReno solamente aplica si las extensiones SACK no han sido negociadas.

# TCP SACK, D-SACK

---

## **SACK:**

Que paquetes han sido recibidos?

Que es lo que falta?

SACK permite desacoplar el **cúando** y el **qué** enviar.

- TCP estandar de ACK acumulativo permite retransmitir solamente un paquete por cada RTT

La extensión *Duplicate-SACK (D-SACK)* permite al receptor indicar la recepción de segmentos duplicados mediante el uso de los bloques de SACK.

El origen puede detectar retrasmisiones innecesarias por haber considerado pérdidas de paquetes que no existieron. ej.: Reordenado de paquetes es una razón potencial para el reenvío no necesario de paquetes.



# Performance en enlaces de alto BDP

---

BDP (Bandwidth Delay Product)

Los algoritmos de control de congestión presentados presentan un pobre desempeño en redes de alta Velocidad y alto RTT, donde la cwnd es muy grande.

La ventana de congestión se reduce a la mitad y solamente es incrementada un MSS por cada RTT

=> Toma mucho tiempo recuperar el tamaño de la ventana luego de una condición de congestión.

TCP BIC

TCP CUBIC



# TCP BIC

La mejora en rendimiento de BIC se obtiene de:

- Crecimiento lento en la meseta
- Crecimiento lineal en “Additive Increase” y “Max Probing”

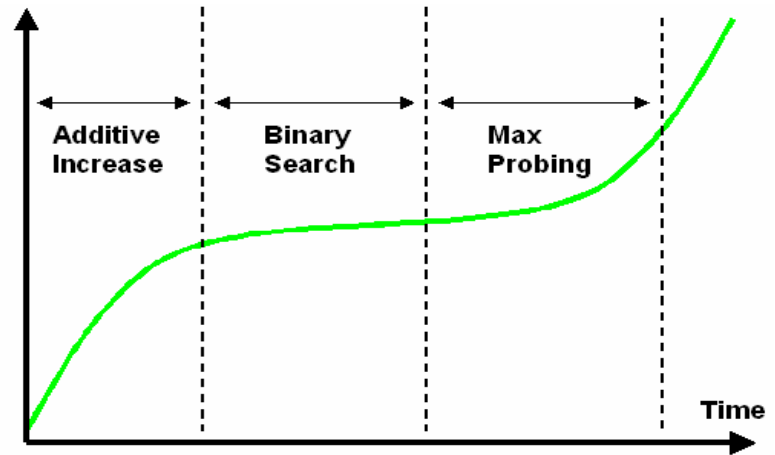
En la meseta:

TCP friendliness. BIC es más lento que TCP original.

**AIMD (additive increase multiplicative decrease)**

TCP Fairness: Reparto justo del ancho de banda entre otros flujos BIC.

**Incrementa la utilización de la Red.**



$W_{max}$ ,  $W_{min}$  -> Límites de búsqueda

$S_{max}$  -> define incremento lineal

$S_{min}$  -> Condición de Convergencia

# TCP CUBIC

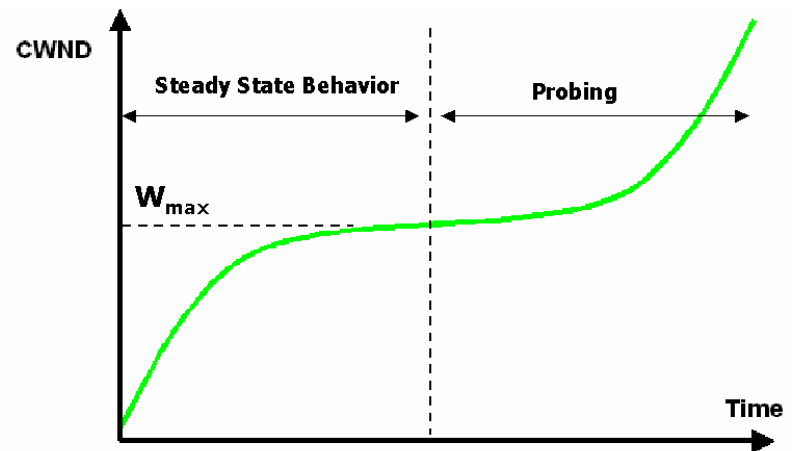
BIC es muy agresivo, sobretodo en redes de bajo RTT

## CUBIC

Nueva función de crecimiento de ventana.  
(Cúbica)

Nuevo modo “friendly”

Detección de bajo uso



$$W_{cubic} = C(T - K)^3 + W_{max} \quad (1)$$

$C$  is a scaling constant, and  $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$

$W_{max}$  es el tamaño de la ventana antes de la reducción  
 $T$  es el tiempo transcurrido desde la ultima reducción de ventana.,  
 $\beta$  es el factor de decrecimiento multiplicativo luego de un evento de pérdida de paquetes.

# Evitar la Congestión

---

Dos posibilidades

Centradas en Host:

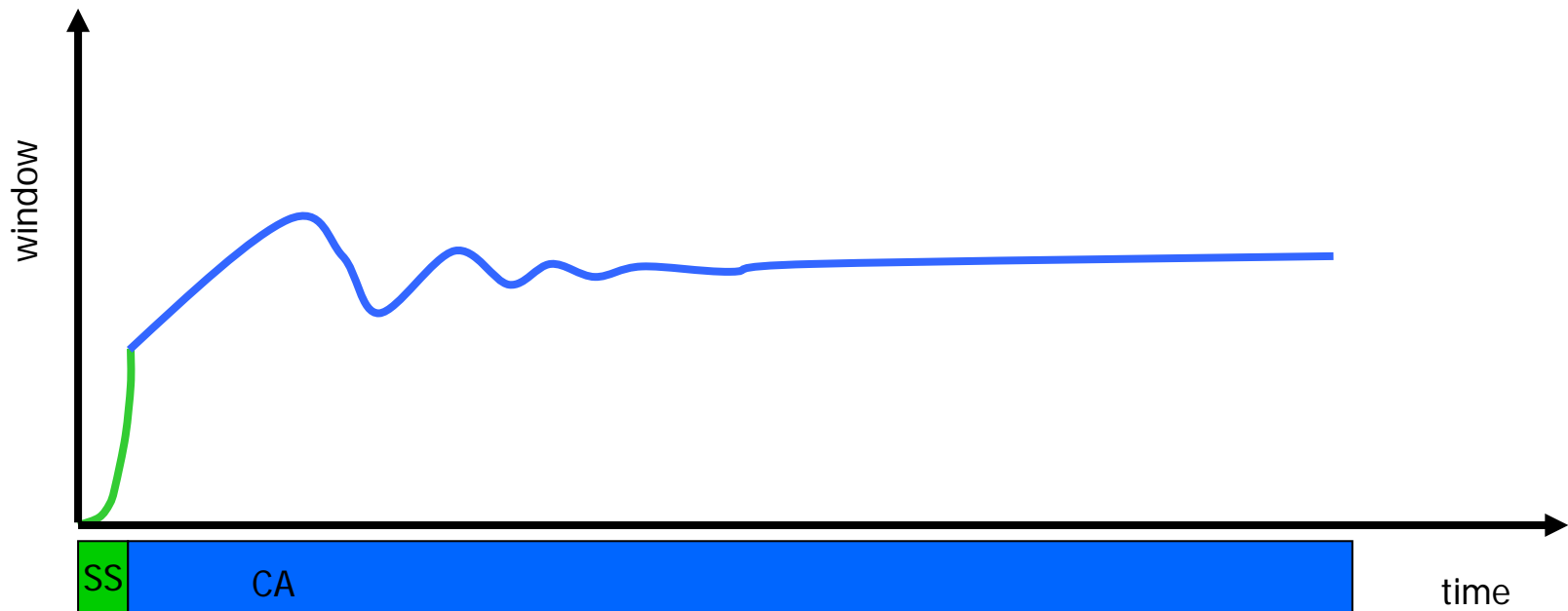
TCP Vegas

Centradas en Red:

AQM (Active Queue Management)

ECN (Explicit Congestion Notification)

# TCP Vegas



Convergencia, no hay retransmisiones

... con la condición de que los buffers sean suficientemente grandes

# TCP Vegas

por cada RTT  
{ si  $W/\text{BaseRTT} - W/\text{RTT} < a \Rightarrow W$   
++  
si  $W/\text{BaseRTT} - W/\text{RTT} > b \Rightarrow W$  --  
}

por cada paquete perdido

$$W := W/2$$

## Expected Sending Rate

$$E = \text{cwnd}/\text{BaseRTT}$$

**BaseRTT**: RTT en condición de no congestión  
(en la práctica = min RTT)

**alpha**: valor pequeño no-cero para obtener  
ventaja del ancho de banda disponible  
inmediatamente. ( $= 1/\text{BaseRTT}$ )

**beta**: beta-alpha no debe ser muy pequeño a fin  
de evitar oscilaciones. ( $= 3/\text{BaseRTT}$ )

Indicador de Congestión = retardo de  
extremo a extremo

En equilibrio

No hay pérdida de paquetes

Ventana estable con utilización  
maximizada

Ocupación no-cero de colas

Convergencia

Converge si hay suficiente buffer

En caso contrario oscila como Reno

# TCP Vegas

---

Problemas comunes de algoritmos basados en retardo:

- Es el retardo un indicio confiable de congestión:
- Mediciones de retardo:
  - Problemas de muestro:
  - Baja correlación cuando hay pérdida de paquetes
- Influencia del retardo de camino inverso
  - Muy difícil de medir el retardo en un solo sentido
- Variaciones en el RTT de referencia. Ej.: Enlaces wireless

Específicos a TCP Vegas y algoritmos relacionados:

- Retardo no optimizado para ser pequeño
  - Ocupación de colas crece significativamente con la cantidad de flujos
  - Sobrecarga inevitable.
- El propio algoritmo hace difícil la medición del retardo (esp. baseRTT).



# AQM (Active Queue Management)

---

Routers en Internet deben implementar políticas de “active queue management” a fin de:

- Administrar el tamaño de las colas
- Reducir el retardo de extremo a extremo
- Reducir el descarte de paquetes
- Evitar lock-out (Sincronización Global) en Internet.

# Control de Congestión en TCP

## Implementaciones en Linux

---

- TCP Vegas se incluyó en la distribución oficial del Kernel Linux 2.6.6
- A partir de Linux 2.6.13 se incluyó TCP BIC; y a partir de 2.6.19 TCP CUBIC es el estándar en la distribución Linux
- Actual Linux Implementation use the TCP protocol defined in RFC 793, RFC 1122 and RFC 2001 with the NewReno and SACK extensions.



# Conclusiones

---

- Aplicar control de flujo es crítico para el desempeño de las redes.
- Fundamentalmente se utilizan técnicas de detección de congestión basadas en pérdida de paquetes.
- AQM ha demostrado ser un muy buen complemento a los algoritmos de Control de Congestión de TCP.
- Control de Flujo en TCP, área de investigación y desarrollo muy activa.
  - Especialmente en Redes de Alto BDP



# Referencias y Bibliografía

---

- W. Richard Stevens. TCP/IP Illustrated Volume 1
- Jon Postel. Transmission Control Protocol, September 1981, RFC 793.
- W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997, RFC 2001.
- M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options, October 1996, RFC 2018.
- B. Braden. Recommendations on Queue Management and Congestion Avoidance in the Internet, April 1998, RFC 2309
- M. Allman, V. Paxson, W. Stevens. TCP Congestion Control, April 1999, RFC 2581
- S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Mechanism, April 1999, RFC 2582.
- Steven Low. Equilibrium & Dynamics of TCP/AQM, Sigcomm August 2001
- Reordering basics and how TCP handle it using SACK, D-SACK - *Rajesh Patel*
- BIC & CUBIC, <http://netsrv.cse.ncsu.edu/twiki/bin/view/Main/BIC>