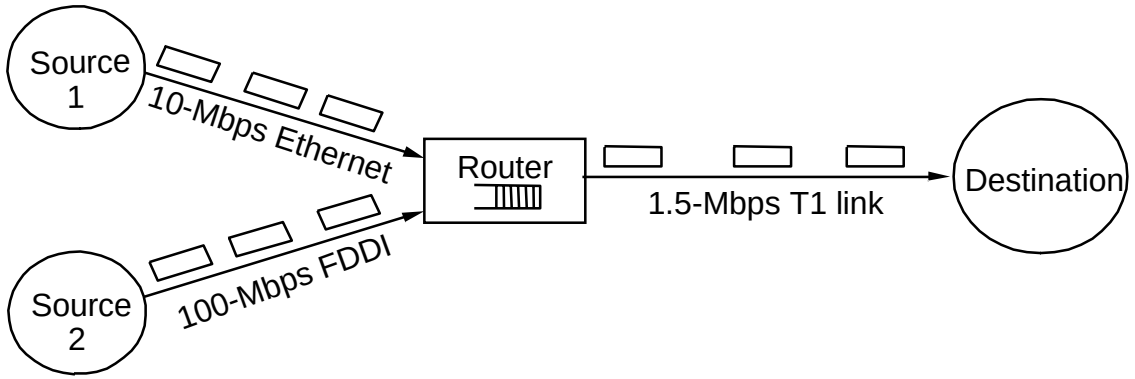
Control de Congestión

Contenidos

- Disciplina de encolado
- Reaccionando ante Congestión
- Abolición de Congestión

Problemas

- Control de Congestión: Es el esfuerzo hecho por los nodos de la red para prevenir o responder a sobrecargas de la red que conducen a perdidas de paquetes.
- Los dos lados de la moneda
  - Pre-asignar recursos (ancho de banda y espacio de buffers en routers y switches) para abolir congestión
  - Controlar la congestión si ocurre (y cuando ocurra)



- Objetivo: asignar los recursos de la red en forma “equitativa”; es decir cuando haya problemas compartir sus efectos entre todos los usuarios, en lugar de causar un gran problema a tan solo unos pocos.

Problemas (cont)

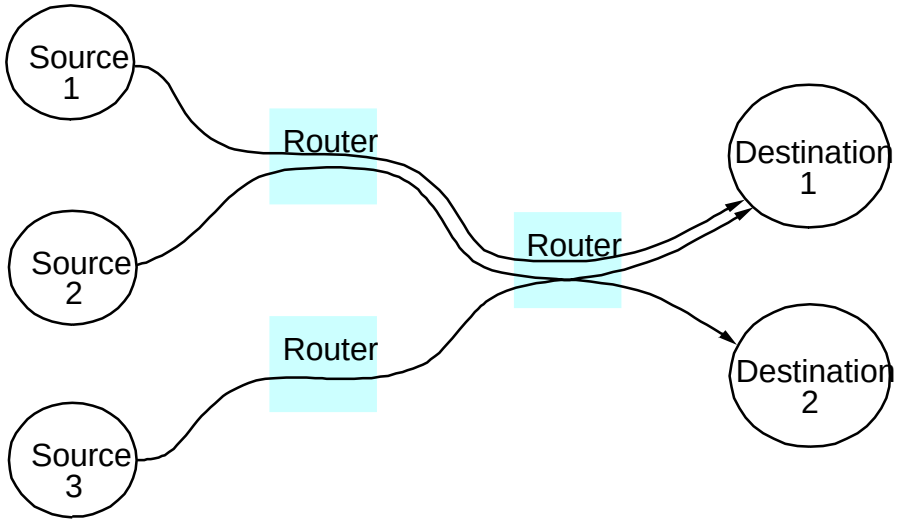
- Control de flujo v/s control de congestión: el primero previene que los transmisores sobrecarguen a receptores lentos. El segundo evita que los transmisores sobrecarguen el interior de la red.

- Dos puntos para su implementacion
  - maquinas en los entremos de la red (protocolo de transporte)
  - routers dentro de la red (disciplina de encolado)

- Modelo de servicio de los niveles inferiores
  - best-effort o mejor esfuerzo (lo asumimos por ahora). Es el servicio de Internet.
  - múltiples *calidades de servicio* QoS (mas adelante). Por ejemplo ancho de banda (para video streaming bajo) y retardo (para Voz sobre IP VoIP).

Marco de trabajo

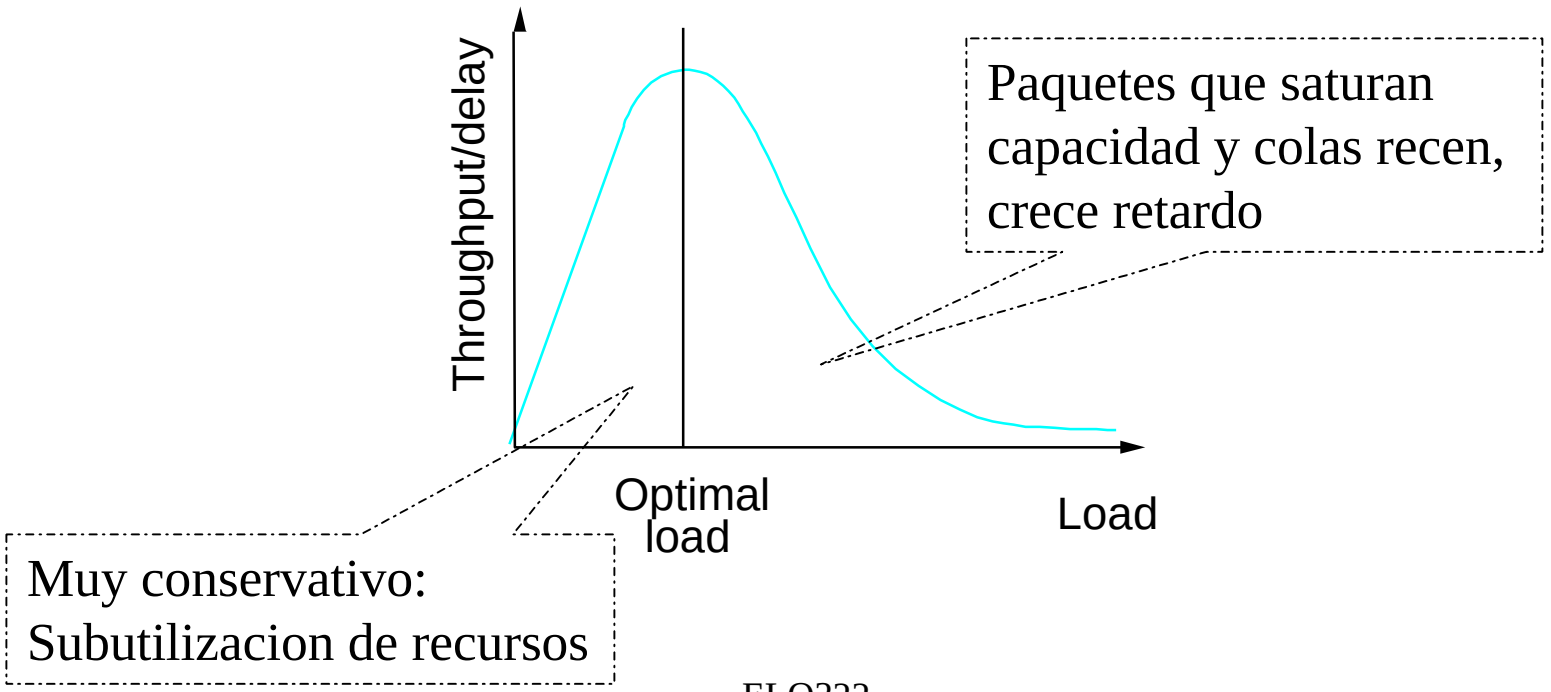
- Caso no internet (como ATM, X.25) hay servicios orientados a la conexión. Se reserva ancho de banda y espacio al establecer la conexión. => subutilizacion de recursos.
- Flujos de datos no orientados a la conexión (Caso Internet)
  - secuencia de paquetes enviados entre el par fuente/destino
  - mantenemos estado transciente en el router



- Taxonomía
  - Centrado en router versus centrado en los hosts
  - basados en reservación versus los ***basados en realimentacion***
  - basados en ventanas versus los basados en tasa de transferencia

Criterios de Evaluación

- La idea es que la red sea ***utilizada eficientemente*** y al mismo tiempo en forma ***equitativa***
- Buen indicador para eficiencia: Potencia =throughput / retardo



Criterios de Evaluación

- Equidad: los recursos sean compartidos equitativamente.
- Indicador de equidad de Jain: Dados flujos por un enlace $(x_1, x_2, ...x_n)$

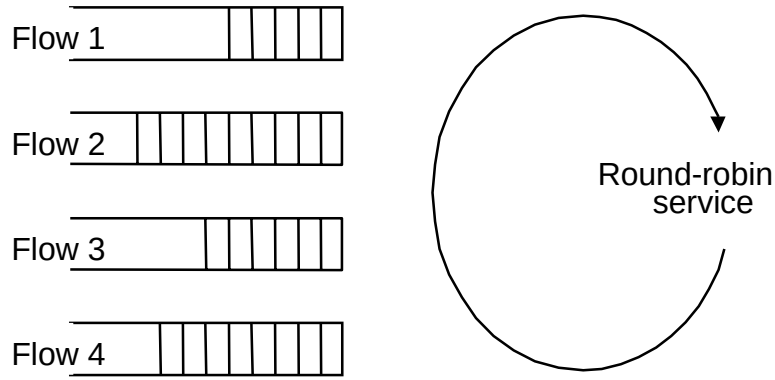
$$f(x_1, x_2, ..., x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}$$

- $0 \leq f \leq 1$

Disciplinas de Encolado

- Hay que distinguir entre disciplina para sacar paquetes del buffer (programación) y disciplina de descarte de paquetes
- Disciplinas de programación: FIFO y Encolado equitativo

- First-In-First-Out (FIFO)
  - No discrimina entre fuentes de tráfico
  - Problema: no discrimina entre diferentes fuentes de tráfico.
- Fair Queuing (FQ) (encolado equitativo)
  - explícitamente segrega tráfico basado en los flujos (varias colas)
  - asegura que ningún flujo tomará más de su cuota de capacidad
  - variación: weighted fair queuing (WFQ) (encolado equitativo ponderado)
- Problema?:
  - paquetes de distinto tamaño

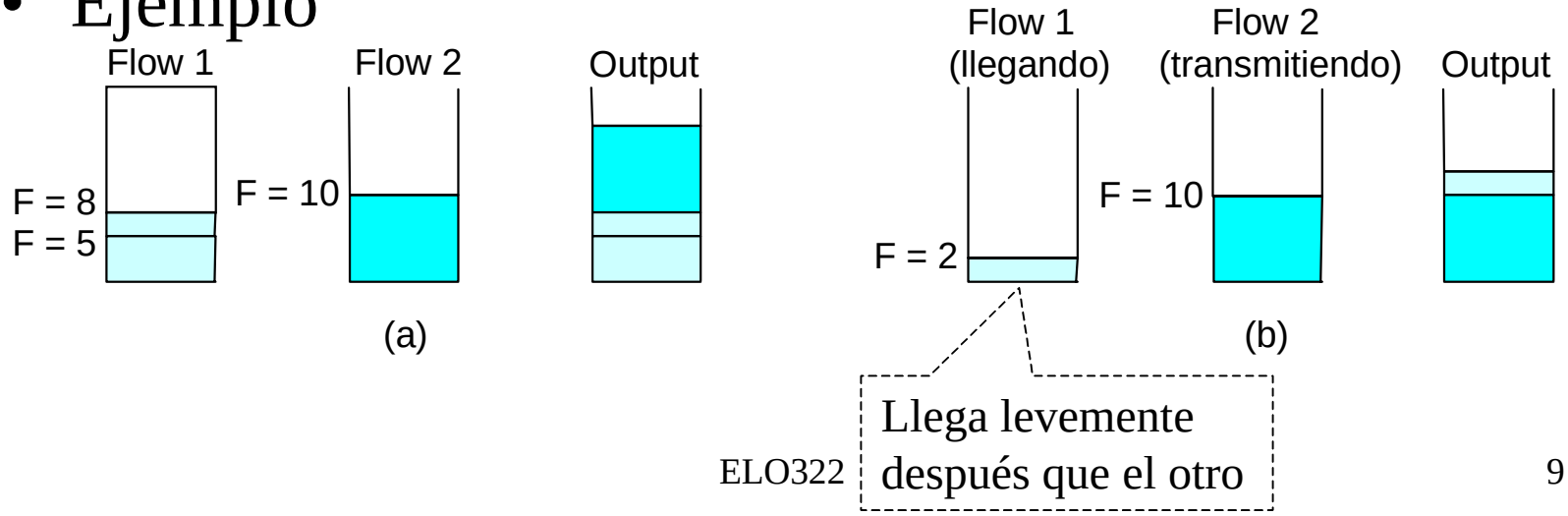


Algoritmo FQ

- Idealmente queremos round-robin bit a bit.
- Supongamos que un reloj hace tic cada vez que un bit es transmitido
- Sea P_i el largo del paquete i medido en tics
- Sea S_i el tiempo cuando comenzamos a transmitir el paquete i
- Sea F_i el tiempo cuando terminamos de transmitir el i
- $F_i = S_i + P_i$
- Cuando el router comienza a transmitir el paquete i?
  - Si esto es antes que el router termine de transmitir el paquete i - 1 de este flujo, entonces i es transmitido inmediatamente después del ultimo bit de i - 1 (F_{i-1})
  - Si no hay paquetes actuales de este flujo, entonces comenzar a transmitir cuando el paquete llegue (digamos tiempo A_i)
- Así: $F_i = MAX\ (F_{i\ -\ 1}, A_i) + P_i$

Algoritmo FQ (cont)

- Para múltiples flujos
  - calcula F_i para cada paquete que llega en cada flujo
  - Tratar todos los F_i's como marcas de tiempo
  - el próximo paquete a transmitir es el con marca te tiempo mínima
- No es perfecto: no puede retrasar el paquete actual
- Ejemplo



Comentarios sobre FQ

- Puede ser combinado con varias políticas de descarte.
- Existen esquemas ponderados. En estos se asigna distinta tasa de salida a cada flujo. Por ejemplo, para acceso internet: red de alumnos 2, profesores 2 y memoristas 1 (2/5, 2/5 y 1/5 del ancho de banda).
- Se puede considerar también FQ en términos de clases de tráfico (usando campo Type of service de IP, se usan tres bits 0= best effort, 7 mayor prioridad).

Control de Congestión en TCP

- Idea
  - TCP asume red de “mejor esfuerzo” (routers FIFO o FQ) cada fuente determina por si misma la capacidad de la red.
  - Usa reglamentación implícita
  - ACKs regulan el paso transmisiones. Cuando llega un ACK implica que un paquete salió de la red (*self-clocking*).
- Desafíos
  - determinación de la capacidad disponible en primer lugar
  - ajustes a los cambios en la capacidad disponible

Incremento aditivo/Decremento multiplicativo

- Objetivo: ajustarse a los cambios en la capacidad disponible
- Nueva variable de estado por conexión: **`CongestionWindow`**
  - limita cuantos datos la fuente tiene en transito (en la red)

```
MaxWin = MIN(CongestionWindow,
             AdvertisedWindow)
EffWin = MaxWin - (LastByteSent -
                   LastByteAcked)
```

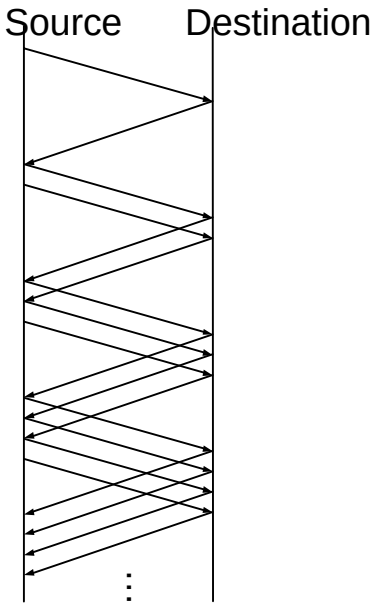
  - Así TCP no envía lo mas restrictivo entre lo que el destino y la red pueden aceptar
- Como TCP llega a un buen valor para **`ContentionWindow`** ?
- Idea:
  - aumentar **`CongestionWindow`** cuando la congestión se reduce
  - reducir **`CongestionWindow`** cuando la congestión aumenta

Incremento aditivo/Decremento multiplicativo (cont)

- Pregunta: como la fuente determina si la red está o no congestionada?

- Respuesta: a timeout ocurre
  - timeout indica que un paquete se perdió
  - Paquetes son raramente perdidos debido a errores de transmisión (ruido en la líneas etc..)
  - Paquetes perdidos implican congestión

Incremento aditivo/Decremento multiplicativo (cont)



- Algoritmo
  - Incrementar **`CongestionWindow`** en un paquete por RTT (*incremento lineal*)
  - dividir **`CongestionWindow`** por dos cada vez que ocurre un timeout (*decremento multiplicativo*)

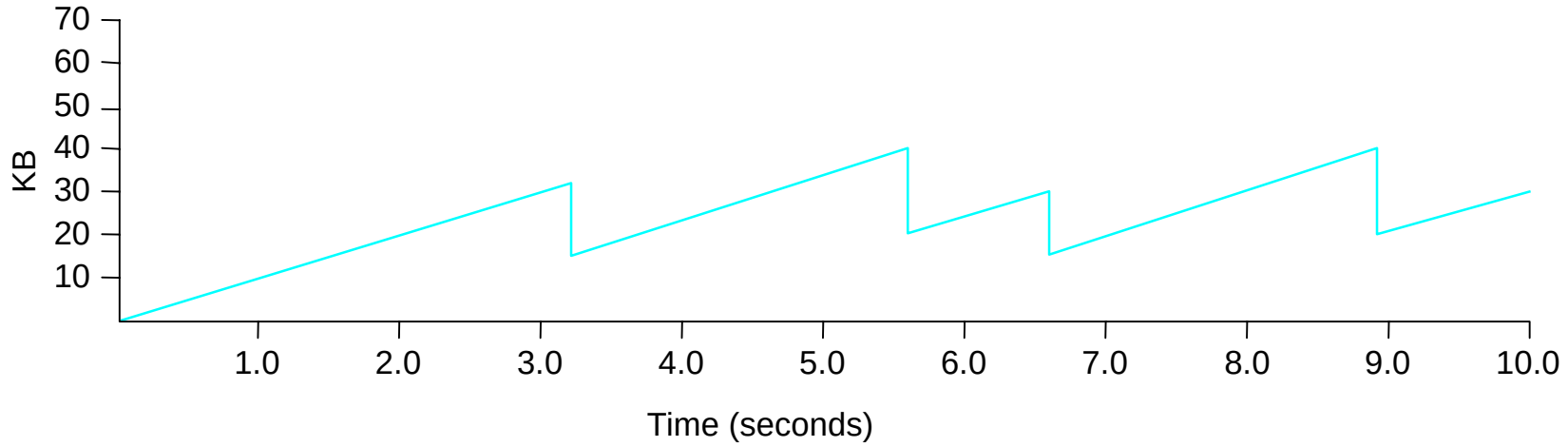
- En práctica: incrementar un poco por cada ACK

```
Increment = (MSS * MSS)/CongestionWindow
CongestionWindow += Increment
```

MSS: Maximum Segment Size

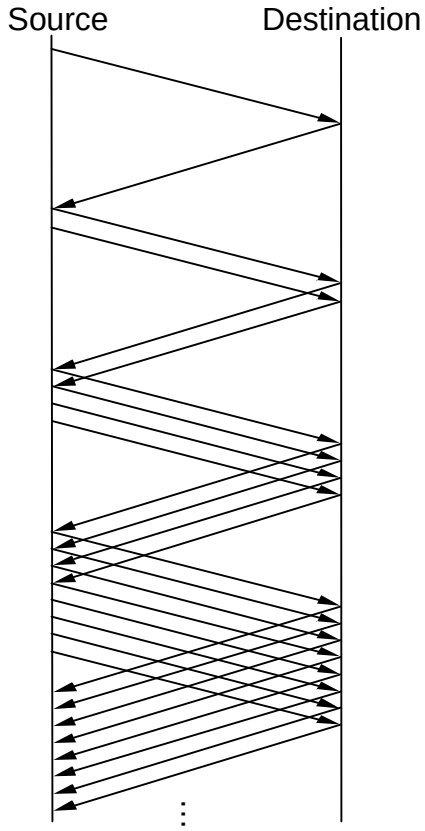
Incremento aditivo/Decremento multiplicativo (cont)

- Traza: comportamiento diente de cierra



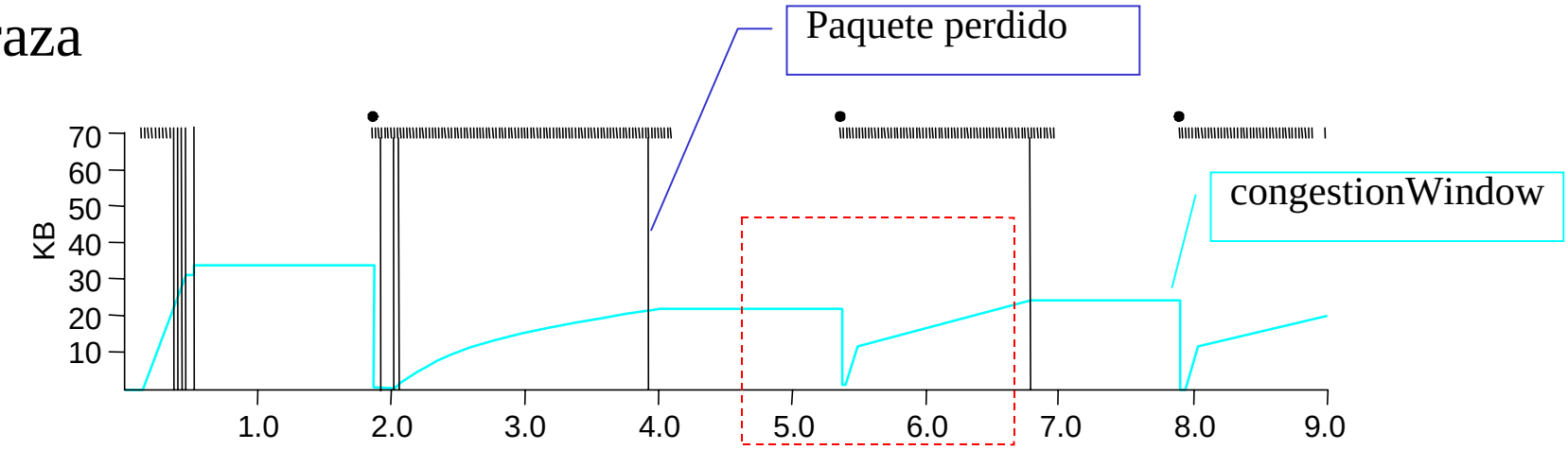
Partida lenta (Slow Start)

- Objetivo: determinar la capacidad disponible, i.e. llegar a la ventana de congestión mas rápidamente que incrementando de a un paquete.
- Idea:
  - comenzar con **`CongestionWindow`** = 1 paquete
  - duplicar **`CongestionWindow`** cada RTT (incrementar en 1 paquete por cada ACK)
- Su nombre se debe a que originalmente TCP enviaba toda la ventana avisada.



Partida lenta (cont)

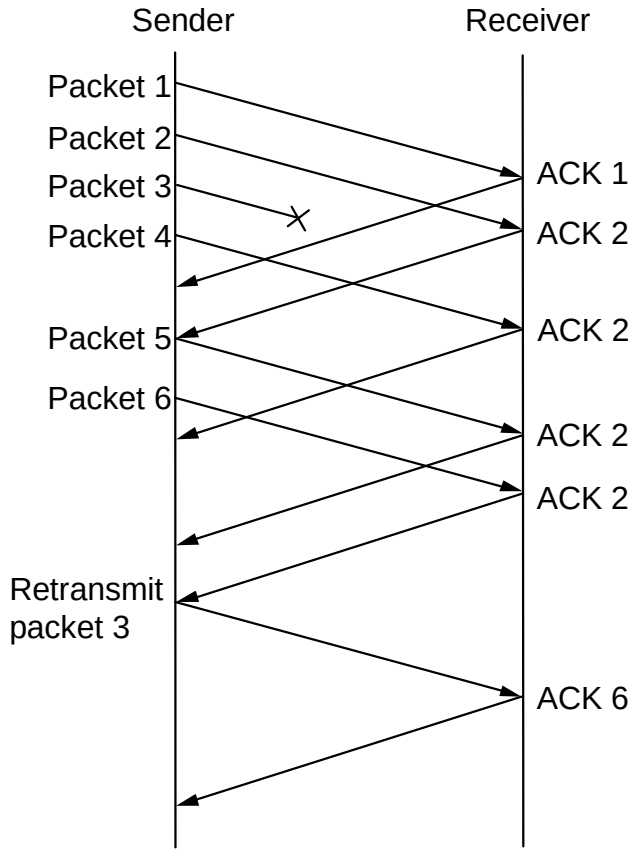
- Crecimiento exponencial, pero más lenta que toda la ventana avisada de una vez
- Cuando es usado…
  - recién iniciada la conexión
  - cuando la conexión se cuelga esperando por un timeout. Al reiniciarse tendría toda la ventana libre para enviar como al partir.
- Traza



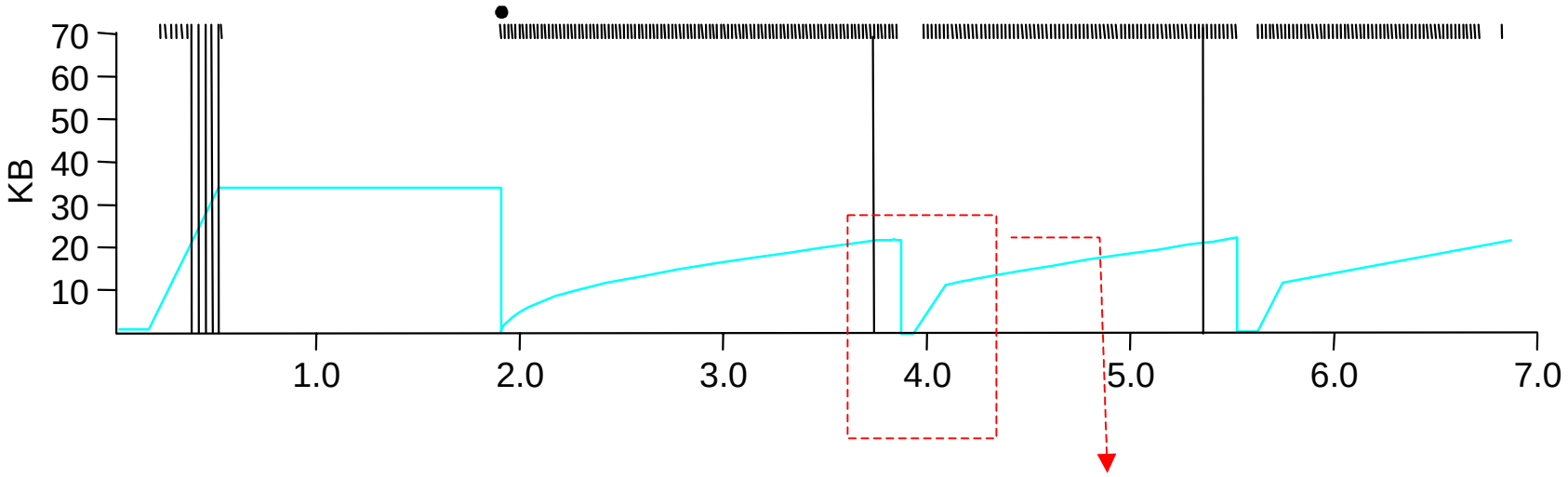
- Problema:al comienzo puede llegar a perder hasta la mitad de una **`CongestionWindow`** en datos (duplica la ventana justo antes que un paquete se pierde en el camino)

Retransmisiones rápidas y Recuperaciones Rápidas

- Problema: granularidad gruesa del timeout de TCP conduce a periodos de no actividad
- Retransmisiones rápidas: usa ACKs duplicados para gatillar retransmisión



Resultados



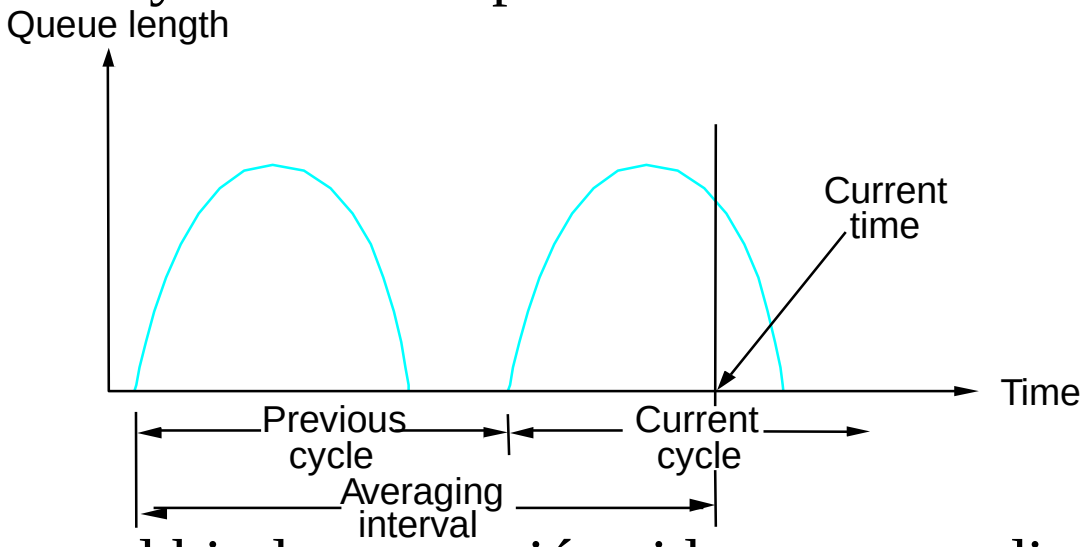
- Recuperación rápida cuando usamos retransmisiones rápidas
  - saltar la fase de partida lenta
  - ir directamente a la mitad de la ultima **`CongestionWindow (ssthresh)` exitosa**

Abolición de la 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
  - llamamos a esto *abolición* de la congestión en lugar de *control* de congestión
- Dos posibilidades
  - Centradas en router: DECbit y RED Gateways
  - Centradas en host: TCP Vegas

DECbit

- Agreda un bit de congestión a cada encabezado de paquete
- Router
  - monitorea largo promedio de cola sobre último ciclo ocupado/libre y el ciclo ocupado actual



  - pone en uno el bit de congestión si largo promedio de cola > 1
  - intenta balancear throughout contra retardo (<1 => poco retardo, router idle. >1 => cola permanente y mayor throughput)

Hosts extremos

- Destino envía eco del bit hacia la fuente
- Fuente registra cuantos paquetes resultaron con el bit marcado
- Si menos que 50% de la última ventana tenía bit marcado
  - incrementa **`CongestionWindow`** en un paquete
- Si 50% o más de la última ventana tenía bit marcado
  - decrementa **`CongestionWindow`** a 0.875 del su valor

Detección aleatoria temprana (Random Early Detection, RED)

- Notificación es implícita
  - solo descarta el paquete (en TCP habrá timeout)
  - podría hacerse explícita marcando el paquete
- Descarte aleatorio temprano
  - en lugar de esperar por que se llene la cola, descarta cada paquete de entrada con alguna *probabilidad de descarte* cada vez que la cola excede algún *nivel de descarte*

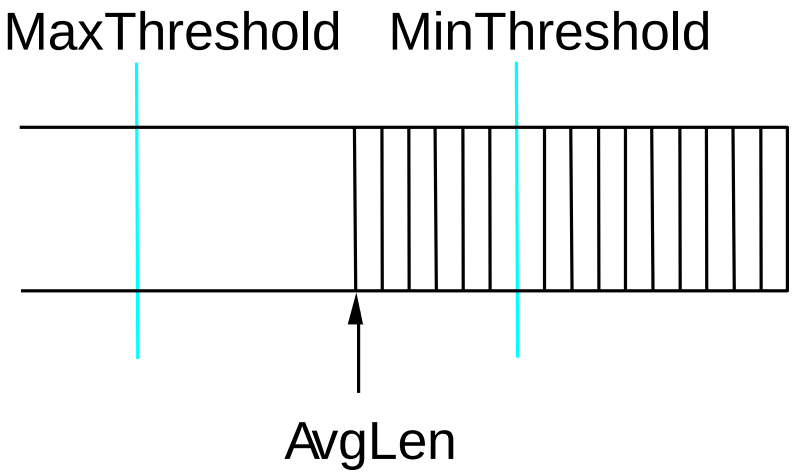
Detalles de RED

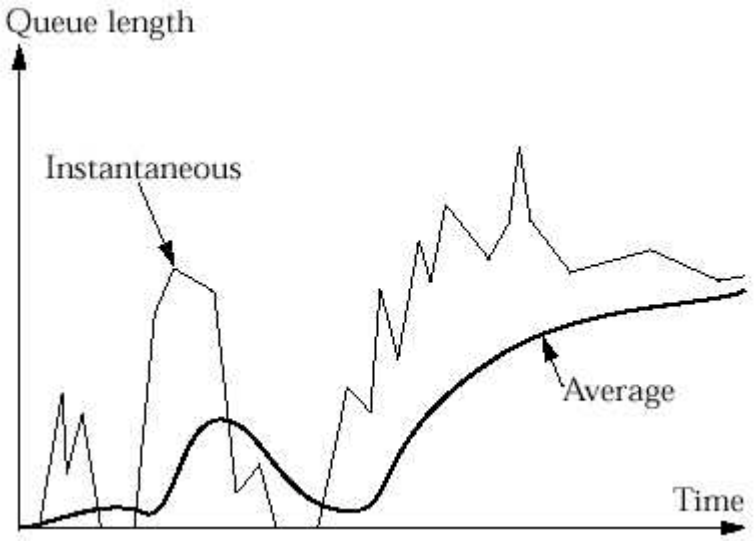
- Calcula largo de cola promedio

```
AvgLen = (1 - Weight) * AvgLen +
         Weight * SampleLen
```

0 < **`Weight`** < 1 (usualmente 0.002)

`SampleLen` es le largo de la cola cada vez que un paquete llega





Detalles RED (cont)

- Dos umbrales de largo de cola

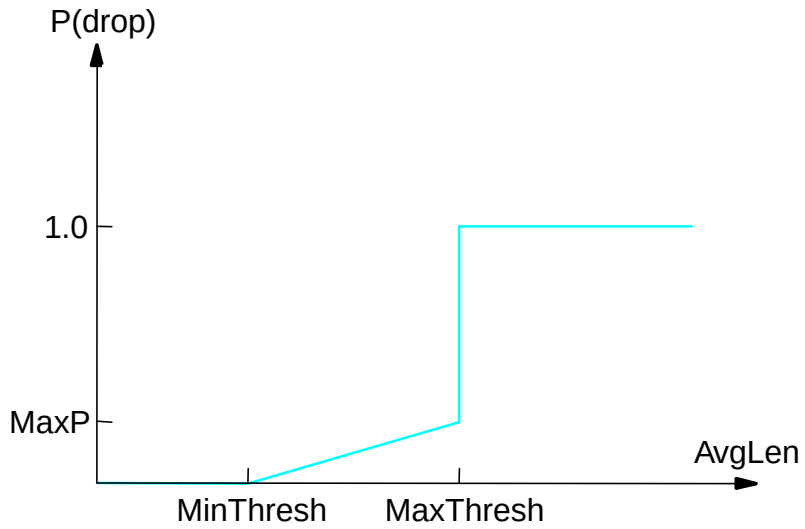
```
if AvgLen <= MinThreshold then
   encole el paquete
if MinThreshold < AvgLen < MaxThreshold then
   calcule probabilidad P
   descarte paquete entrante con probabilidad P
if ManThreshold <= AvgLen then
   descartar paquete entrante
```

Detalles RED (cont)

- Computo de probabilidad P

```
TempP = MaxP * (AvgLen - MinThreshold)/
        (MaxThreshold - MinThreshold)
P = TempP/(1 - count * TempP)
```

- Count cuneta el número de paquetes encolados mientras el AvgLen está entre los dos umbrales
- Curva de probabilidad de descarte

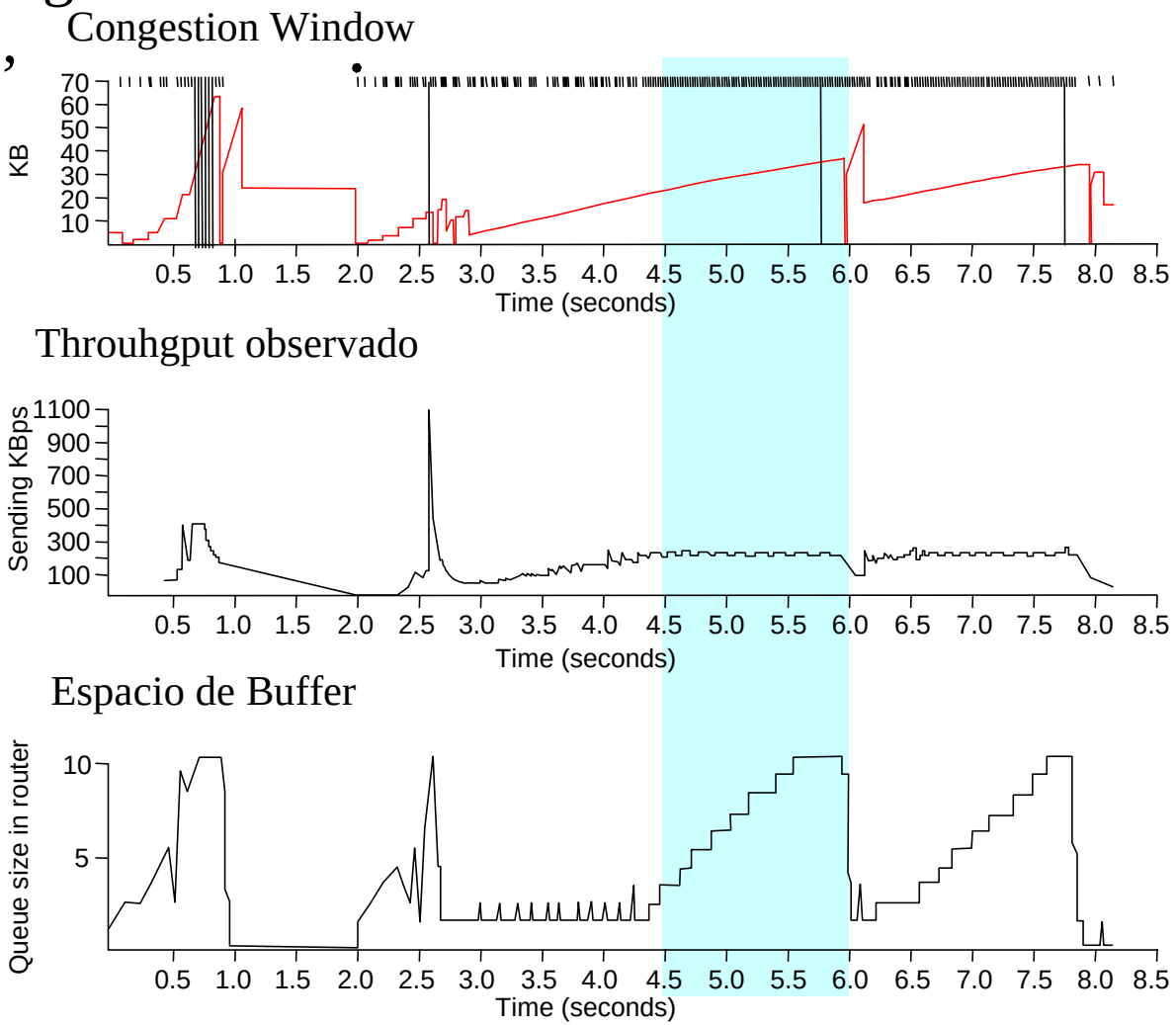


Sintonía en RED

- Probabilidad de descartar un flujo particular de paquetes es aproximadamente proporcional a parte del ancho de banda que el flujo está obteniendo
- **`MaxP`** es típicamente fijada en 0.02, es decir cuando el tamaño promedio de la cola es la mitad entre los dos umbrales, el gateway descarta +o- uno de cada 50 paquetes.
- Si el tráfico es rafagoso, entonces **`MinThreshold`** debería ser suficientemente grande para permitir que la utilización del enlace sea mantenida a un nivel aceptablemente alto
- Diferencia entre los dos umbrales debería ser más grande que el incremento típico en el largo de cola promedio calculado en un RTT; fijar **`MaxThreshold`** a dos veces **`MinThreshold`** es razonable para el tráfico de hoy en Internet

TCP Vegas

- Idea: la fuente observa signos de aumentos de colas y congestión; e.g.,
  - Crecimiento de RTT
  - tasa de envío se aplana



Algoritmo

- Sea **`BaseRTT`** el mínimo de todos los RTTs medidos (comúnmente el RTT de los primeros paquetes)
- Si no hay overflow de la conexión, entonces

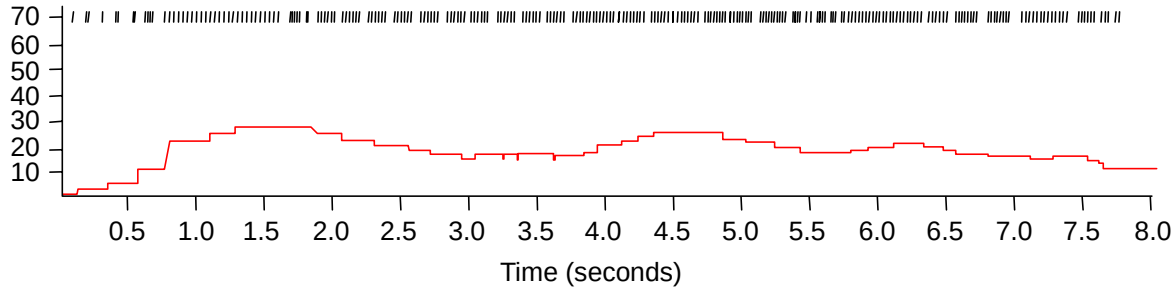
  `ExpectRate = CongestionWindow/BaseRTT`
- La fuente calcula la tasa de envío (**`ActualRate`**) por cada RTT
- La fuente compara **`ActualRate`** con **`ExpectRate`**

```
Diff = ExpectedRate - ActualRate  /* >= 0  */
if Diff < α   /* 1 paquete */
    incrementar CongestionWindow linealmente
else if Diff > β  /* 3 paquete */
    decrementar CongestionWindow linealmente
else
    dejar CongestionWindow sin cambio
```

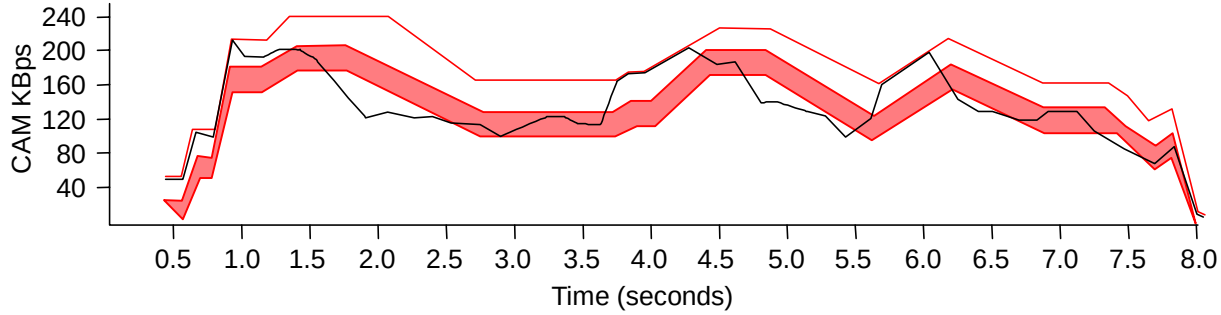
Algoritmo (cont)

- Parametros
  - α = 1 paquete
  - β = 3 paquetes

Congestion Window



Throughput esperado (rojo) y medido (negro) (sombra: región entre β y α)



- Retransmisión aún más rápida
  - mantener marcas de tiempo de granularidad fina para cada paquete
  - chequear timeout del primer ACK duplicado (no el tercero)