Scheduling

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

Redes
4º Ingeniería Informática

Hoy...

1. Introducción a las redes
2. Tecnologías para redes de área local
3. Conmutación de circuitos
4. Tecnologías para redes de área extensa y última milla
5. Encaminamiento
6. **Arquitectura de conmutadores de paquetes**
   - **Scheduling / planificación**

- Control de acceso al medio
- Transporte extremo a extremo

upna

REDES
Área de Ingeniería Telemática

Scheduling why?

- No es suficiente con elegir el primero?

  FCFS first come first served? cola unica elijo el primer paquete de la cola

  DropTail si voy a colocar un paquete en la cola y no hay sitio lo descarto

  – Puede no existir el concepto de el primero en llegar

    en el caso de colas a la entrada tenemos que elegir entre paquetes en varios puertos de entrada... cuál es el primero en llegar?

  – Podemos querer tratar de forma diferente a diferentes aplicaciones

    Aplicaciones elásticas (mail, web, transferencia de ficheros)

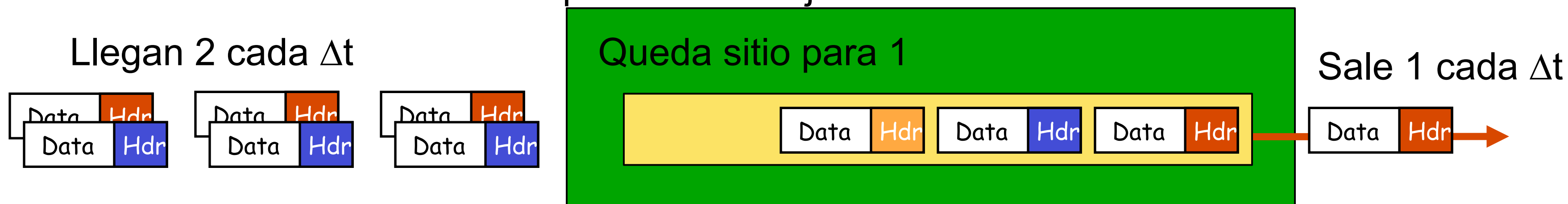
    No les importa mucho tardar unos pocos milisegundos mas en salir

    (best-effort)

    Aplicaciones con restricciones de tiempo (VoIP, videoconf, juegos)

    Milisegundos de mas marcan la diferencia entre funcionar o no

  – Podemos querer tratar a todos los paquetes por igual (Fairness) y si no tenemos cuidado FCFS puede crear injusticias



Principios de scheduling

- Requisitos
  - Facilidad de implementación

    La decisión debe tomarse por cada paquete

    Debe escalar con el numero de entradas/flujos/conexiónes

    Debe tener poco estado interno

    Normalmente se hará en hardware
  - Equidad(fairness) and protection

    Debe tratar por igual a las entradas

    incluso en condiciones en las que las entradas no se comportan correctamente no debe penalizar a las que si lo hacen

    fairness -> protection pero no al reves
  - Requisitos de prestaciónes

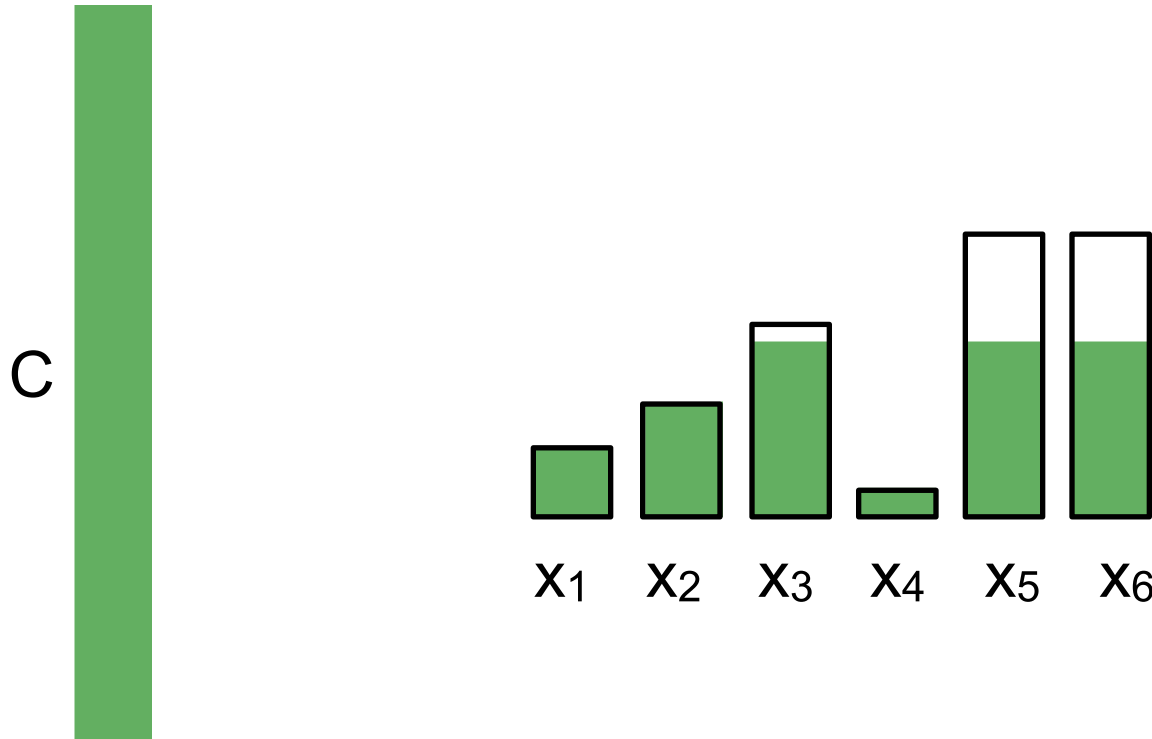
    Ser capaz de garantizar limites arbitrarios de prestaciones para algunas entradas/flujos

    Estos limites se garantizaran asignando recursos
  - Facilidad de control de admisión

    Ser capaz de decidir si permitir una nueva entrada/flujo permitira seguir cumpliendo los limites o no, para decidir si aceptarla

Equidad max-min (max-min fairness)

- Propiedad deseable para un reparto de un recurso escaso
- El recurso tiene una capacidad a repartir C
- Las demandas de recurso de los diferentes usuarios son $\{x_1,x_2,x_3...\}$ tales que en general $x_1+x_2+x_2+... > C$
- Intuitivamente querríamos conceder su petición a los usuarios que piden poco y repartir por igual entre los usuarios que piden mucho



Equidad max-min (max-min fairness)

- Algoritmo
- Ordenar las demandas de menor a mayor
- Supondremos que $\{x_1,x_2,x_3...\}$ están ordenadas de forma que $x_1<x_2<x_2<...$
- Asignamos C/N a cada usuario

  Si $x_1<C/N$ nos sobra $C/N-x_1$ para repartir entre el resto

  Sumamos $(C/N-x_1)/(N-1)$ a cada uno del resto

  x_1 ya tiene asignado la cantidad correcta tomamos el siguiente y repetimos el proceso

- Maximiza el mínimo asignado a las demandas que no son satisfechas

  No penaliza a los usuarios que se comporten bien y piden poco incluso en presencia de usuarios que piden mucho.

Scheduling más simple

- FCFS + DropTail
- No hacemos distinciones

  Primero en entrar primero en salir

  Al colocar un paquete en cola si no hay sitio se elimina

- Utilización media del servidor (independiente del scheduling utilizado)

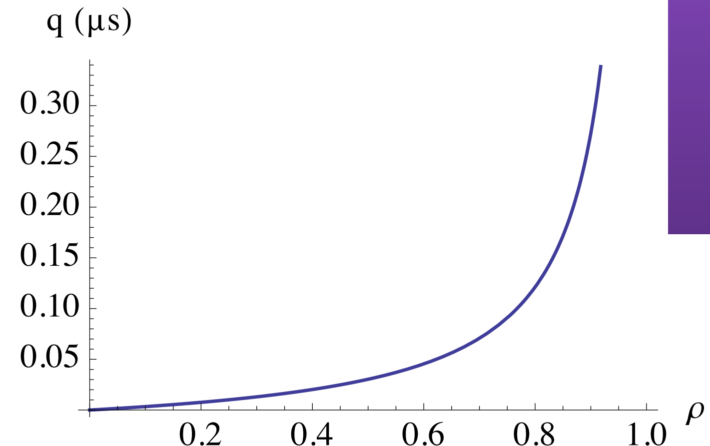
  ρ=λ x

  λ paquetes por unidad de tiempo

  x tiempo medio de servicio del paquete
- **q** : Tiempo medio de espera en la cola

  depende de la utilización

  cómo depende se discute en teoría de colas



- FCFS no hace distinción entre paquetes y todos tienen que esperar en media eso. ¿Podemos hacer otras disciplinas de scheduling que consigan que algunos paquetes esperen menos?

Ley de conservacion

- Si que se puede conseguir menos tiempo de espera si separamos las llegadas y las tratamos de forma diferente

  En las próximas clases se discutirán métodos

  Pero... hay un limite
- Ley de conservación

  Si tenemos clases de usuarios i=0,1,2...

  Cada clase genera una carga $\rho_i=\lambda_i x_i$

  Cada clase obtiene un tiempo medio de espera q_i

  Y el sistema es **conservativo de trabajo**=si hay paquetes para servir los sirve

  Se cumple que para cualquier disciplina de scheduling

$$\sum_{i=0}^{N} \rho_i q_i = constante$$

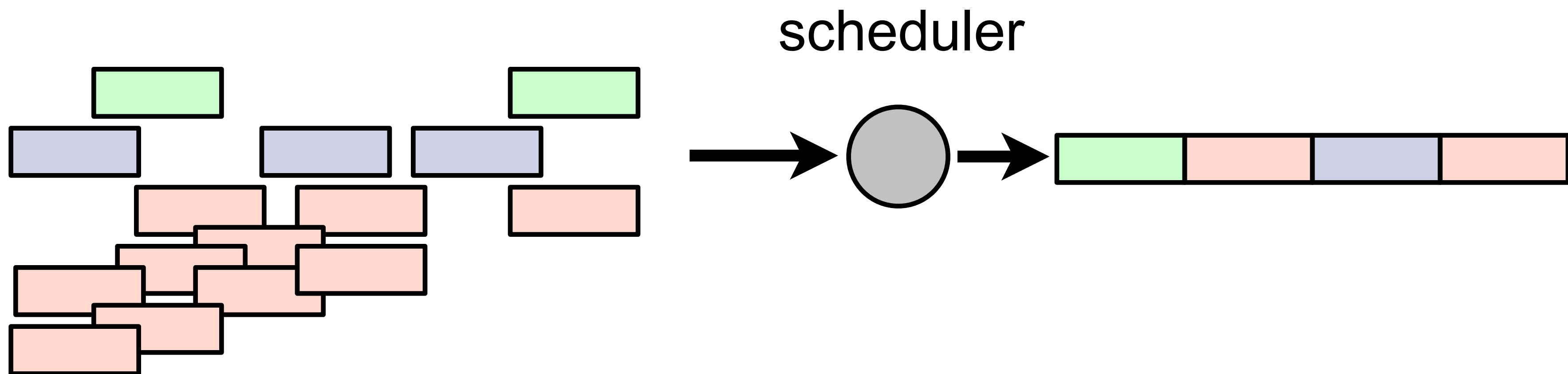
Para FCFS $$q \sum_{i=0}^{N} \rho_i = constante$$

- No podemos obtener un retardo medio menor que el que obtiene FCFS. Sólo podemos reducir el tiempo medio de espera de una clase aumentando el de otras

Scheduling: el escenario

- Servidor que no puede atender a todo lo que le llega

  En media si, pero en un momento de pico tiene que elegir
- Los clientes pertenecen a N entradas/clases/flujos diferentes
  - Entre estas clases debemos garantizar fairness
  - o requisitos arbitrarios de prestaciones (bandwidth, delay, jitter)



FCFS: first come first served

- Scheduler simple

  Una cola por orden de llegada

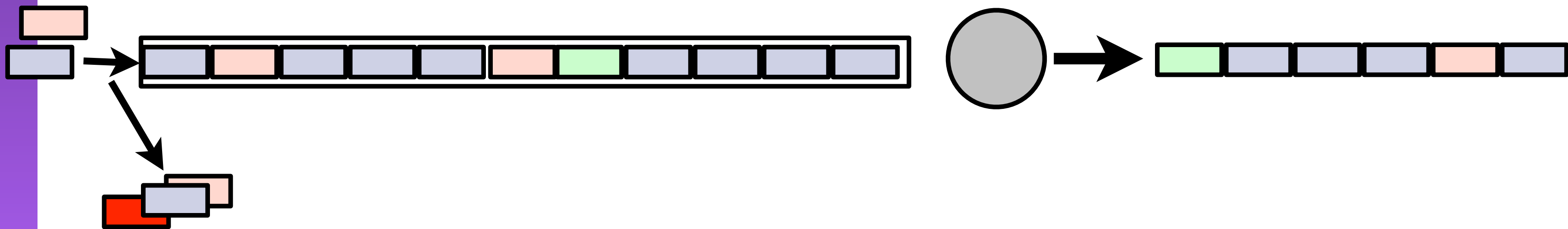
  Sirvo en orden

  No max-min fair

  Reparto proporcional a la demanda

  No hay protección contra flujos que envían mucho que consiguen mas recursos

  Los flujos que se comportan bien y envían poco tienen más probabilidades de perder todos los paquetes



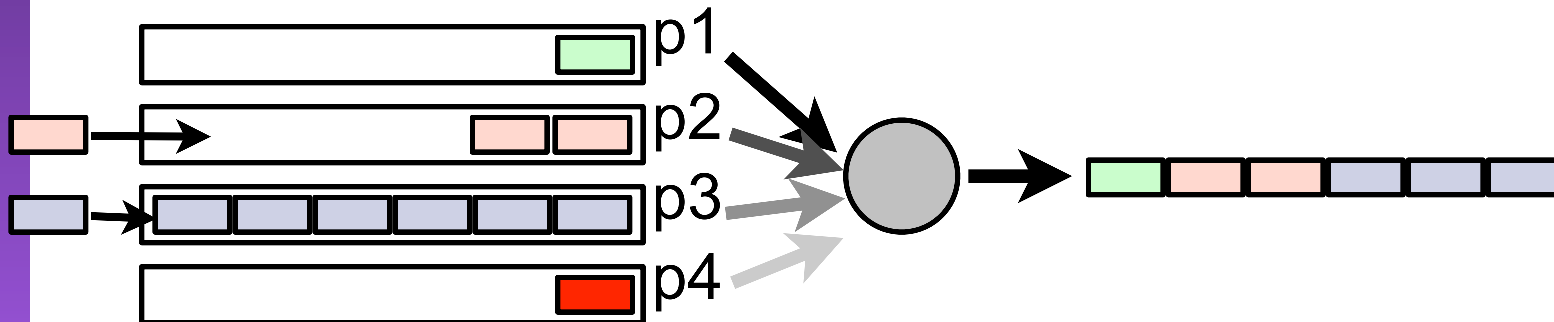
Priority queueing

- Usando prioridades
- Scheduler simple

  Una cola para cada clase

  Ordenadas por prioridad

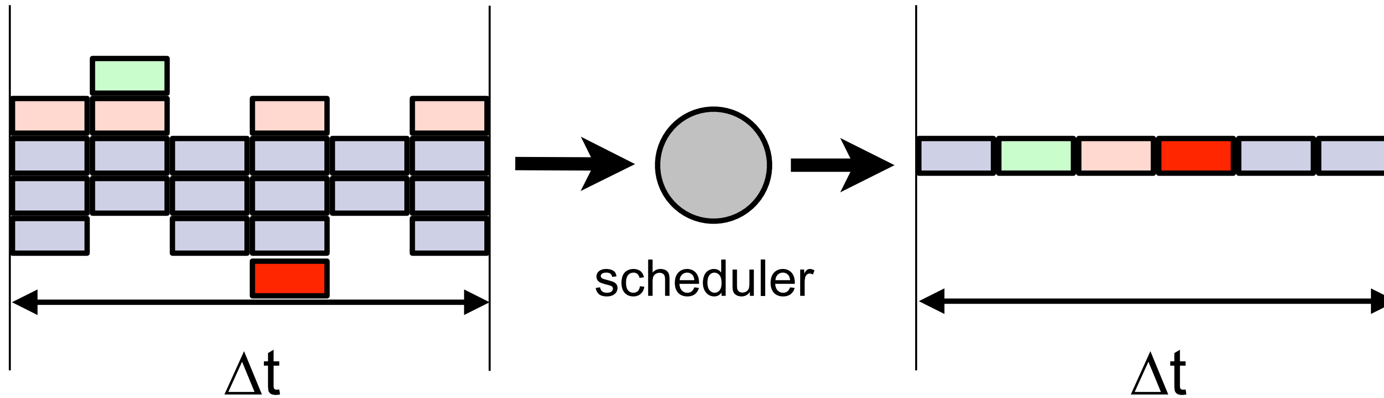
  Sirvo la cola de prioridad k solo si no hay nada que servir en las colas de prioridad <k
- Cierta protección: flujos de alta prioridad no entorpecidos por flujos de menos prioridad

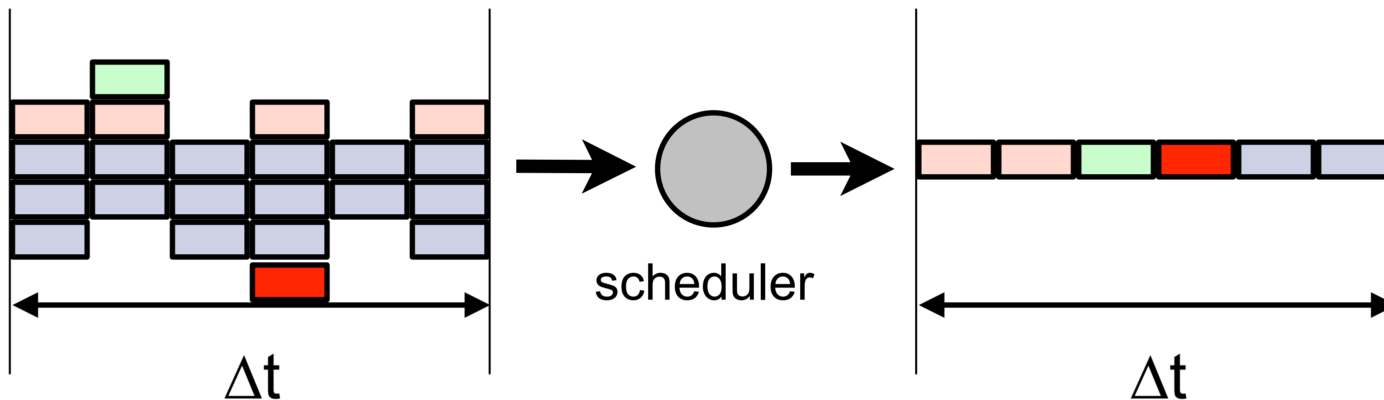


- Problemas:
  - Starvation: demasiado trafico en prioridad alta puede dejar sin servicio a los de menor prioridad
  - No siempre queremos prioridad de unos flujos sobre otros (max min fair)

Max-min fair

- No todos los repartos son max-min fair





- Y depende de a que escala de tiempo se observe
- Cómo construimos schedulers max-min fair

GPS (Generalized processor sharing)

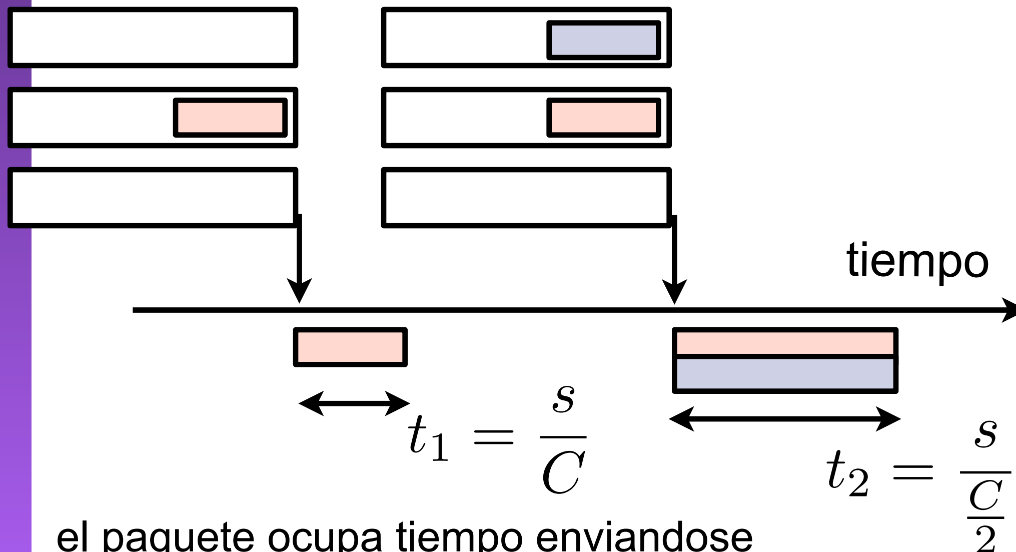
- Hay un algoritmo max-min fair ideal

  Supongamos que el servidor es capaz de enviar por separado trozos infinitamente pequeños de los paquetes (aproximación de trafico como fluido ideal :-))

  Una cola para cada clase

Si dos clases tiene paquetes de tamaño *s* ...
el servidor envía alternativamente trozos infinitamente pequeños de cada uno

Es como si se enviaran los dos a la vez entremezclados pero tardan el doble porque cada uno va a la mitad de la velocidad



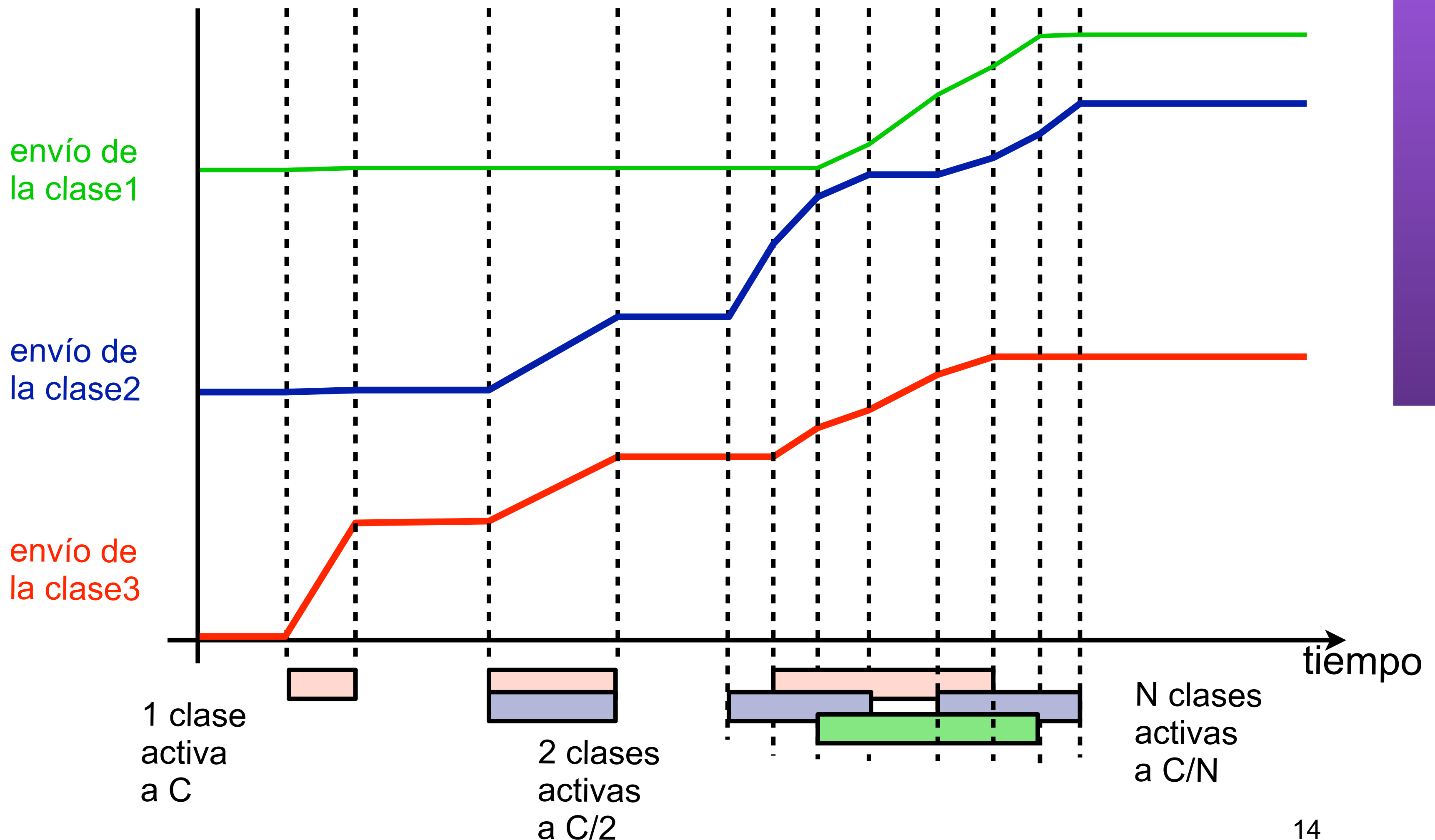
$$t_1 = \frac{s}{C}$$

$$t_2 = \frac{s}{\frac{C}{2}} = 2\frac{s}{C}$$

el paquete ocupa tiempo enviandose
si solo una clase tiene un paquete
de tamaño *s*
se envia durante s/C

GPS (Generalized processor sharing)

- Es equivalente a pensar que los paquetes se envían como un fluido que se envía en total a C unidades por unidad de tiempo



GPS (Generalized processor sharing)

- El reparto puede ser a partes iguales entre las clases

$$c_i = \frac{C}{N}$$

- O proporcional a un conjunto de pesos $\{\phi(1), \phi(2), \ldots\}$

$$c_i = C \frac{\phi(i)}{\sum \phi(i)}$$

- El problema es que GPS es un algoritmo ideal
  - No podemos repartir el servidor entre paquetes en tiempos infinitesimalmente pequeños
  - Tenemos que elegir uno de los paquetes para servir

- ¿Podemos construir aproximaciones a GPS?

Round robin (RR)

- Scheduler simple (aproximación más simple de GPS)

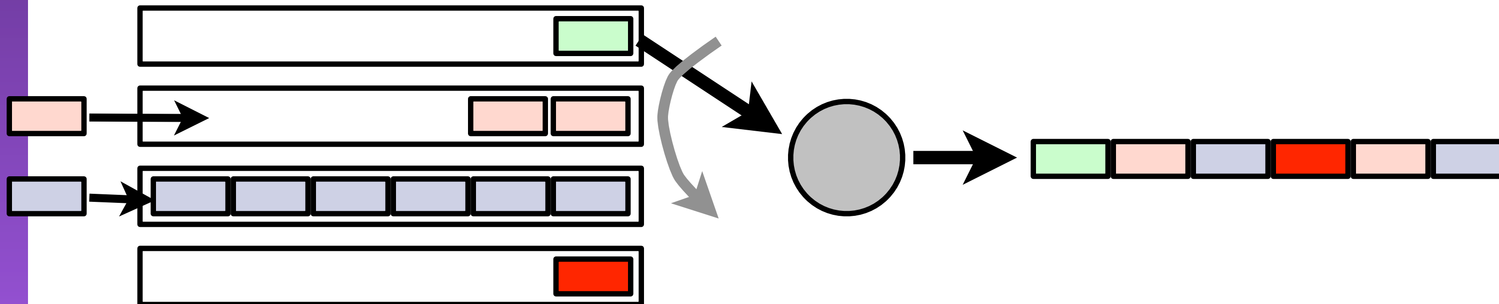
  Una cola para cada clase

  El scheduler coge cada paquete de una cola y pasa a la siguiente

  Si una cola esta vacia pasa a la siguiente sin esperar (work conserving)

  Si se llena la cola de una clase se tiran paquetes de esa clase, no afecta a otras.

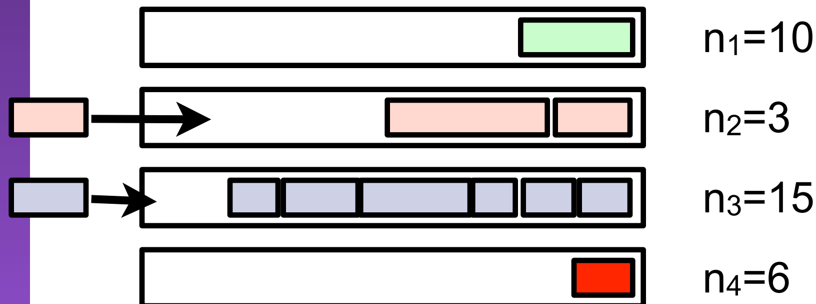
  Esto parece justo, no?

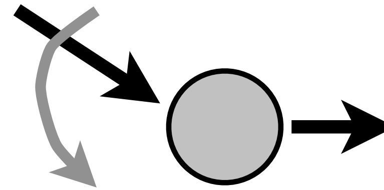


- Es una aproximación a GPS... suficientemente buena?
  - Necesitamos soportar pesos
  - En la realidad los paquetes no son de longitud constante

Weighted round robin (WRR)

- Asignamos pesos a las clases $\phi(i)$
- Normalizamos por el tamaño medio de paquete en cada clase $\frac{\phi(i)}{s_i}$
- Normalizamos los pesos para que sean números enteros
- Con pesos enteros $\{n_i\}$ el algoritmo es fácil
  - Enviar n_i paquetes de la cola i
  - Pasar a la cola i+1





- Es una buena aproximación de GPS...

  Si la miramos en periodos de tiempo mayores que el ciclo (de 34 paquetes en el ejemplo)
- Necesita saber la media del tamaño de paquete de una clase: no es facil
- Es útil en algunos enlaces de muy alta velocidad (con tamaño fijo de paquete y ciclos cortos)

Deficit round robin (DRR)

- Algoritmo para hacer round robin con pesos sin saber previamente el tamaño medio de paquete en cada clase
  - Cada clase tiene un contador de deficit inicializado a 0
  - Hay un quanto q de servicio (i.e. q=1000bytes)
  - Cuando es el turno de servir una clase

    si el paquete siguiente es menor que q+deficit

    se envia ese paquete y el deficit=0

    en caso contrario

    deficit+=q

    no se envia y se pasa a la siguiente cola

- Eso no tenia pesos... esta es la version con pesos
  - Cada clase tiene un contador de deficit inicializado a 0
  - Hay un quanto q de servicio (i.e. q=1000bytes) la idea es dar a cada cola q*peso
  - Cuando es el turno de servir una clase

    si el paquete siguiente es menor que q*peso+deficit

    se envia ese paquete y el deficit=0

    en caso contrario

    deficit+=q*peso

    no se envia y se pasa a la siguiente cola

-

(weighted) fair queueing (WFQ)

- Algoritmo con comportamiento aproximado a GPS

  No asume paquetes infinitesimales

  Soporta tamaño de paquete variable

  Y no necesita tamaño medio de paquete de la clase

  Nombres: WFQ (weighted fair queueing) y PGPS (packet-by-packet GPS)

- Idea:
  - Calcular el tiempo en el que saldría cada un paquete con un planificador GPS (simular GPS)
  - Servir los paquetes en orden de ese tiempo de finalización (le llamaremos numero de finalización)

- Idea para implementarlo:
  - Tiempo virtual que pasa mas despacio cuanto más clases hay activas (con paquetes que enviar)

    Le llamaremos turno
  - Cada clase lleva la cuenta del tiempo en el que saldra el siguiente paquete de esa cola.

    Cuando un paquete llega a la cola su tiempo de salida será el tiempo en el que llega mas el tiempo que dura el envio

    Cuando hay más clases activas no hace falta recalcular el tiempo en el que saldra el paquete sino que el tiempo virtual pasa mas despacio

(weighted) fair queueing (WFQ)

- Algoritmo
- Entrada
  - clases i=0...N   no todas tienen un paquete que enviar en cada momento
  - todas las clases tienen el mismo peso (repartir por igual)
- Variables
  - round R(t) es el tiempo virtual actual
- Acciones:
  - Cuando un paquete i de tamaño s_i llega a la cola de una clase en el tiempo t

    El scheduler calcula R(t) que tiempo virtual es ahora

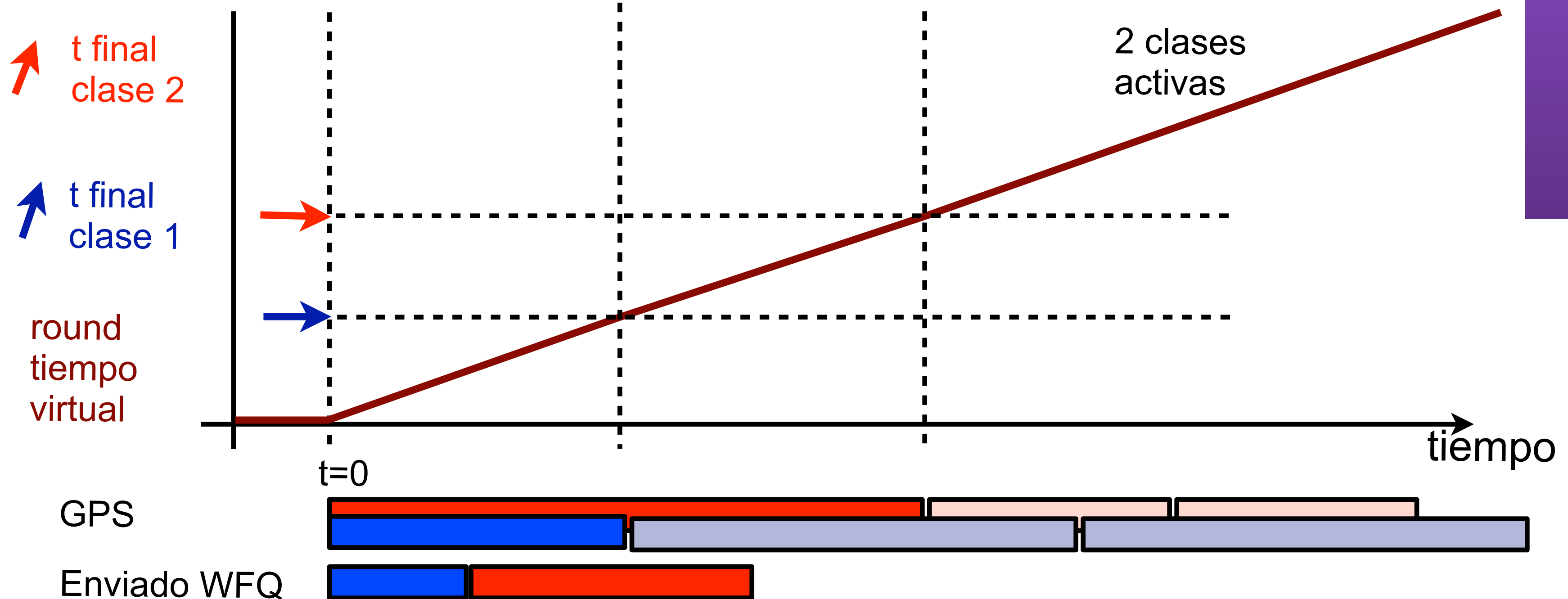
    Si esta vacia su tiempo de finalización con GPS es $F(i)=R(t)+s_i$

    Si el paquete anterior i-1 esta en la cola su tiempo de finalización es $F(i)=F(i-1)+s_i$

WFQ ejemplo

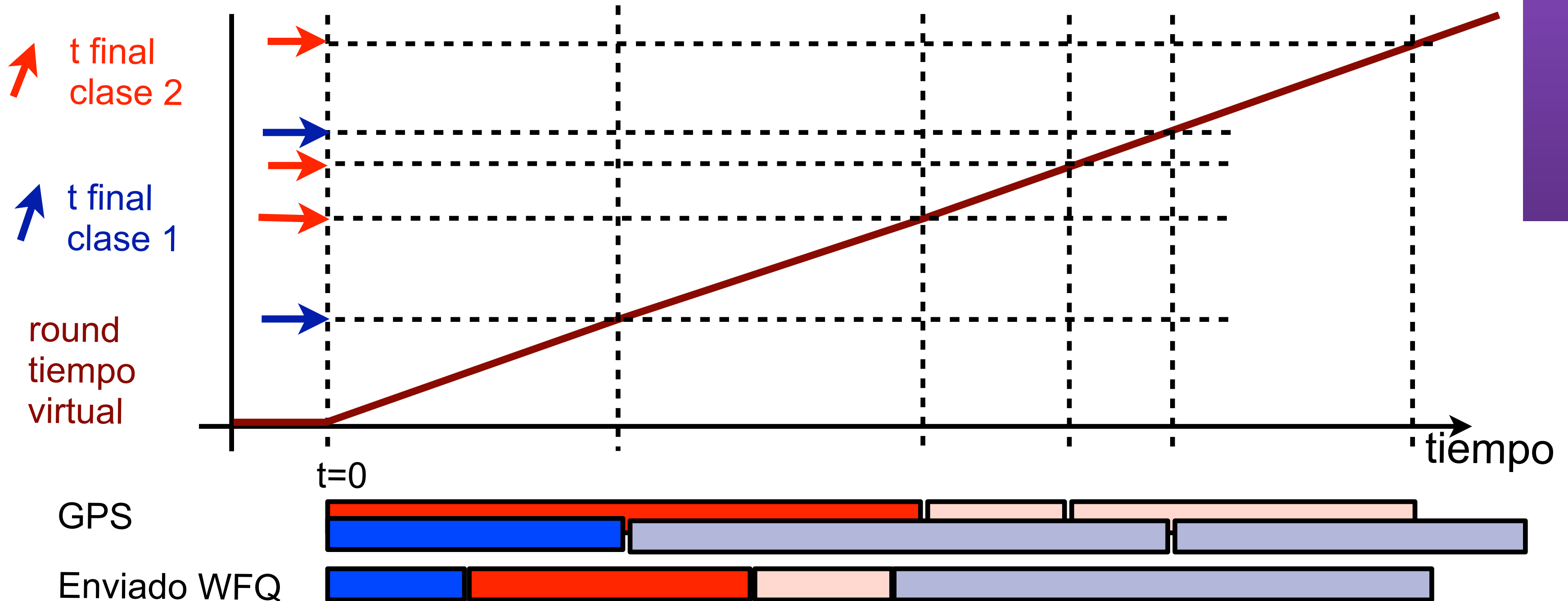
- Ejemplo dos paquetes de tamaño s_1=1 y s_2=2 son los primeros en las clases 1 y 2 del scheduler de WFQ

  y son las unicas clases activas
- El tiempo virtual va a velocidad R(t)=t k/2
- En el instante t=0 que llegan R(0)=0
- Tiempo de finalización del primero R(0) + s_1
- Tiempo de finalización del segundo R(0) + s_2



WFQ ejemplo

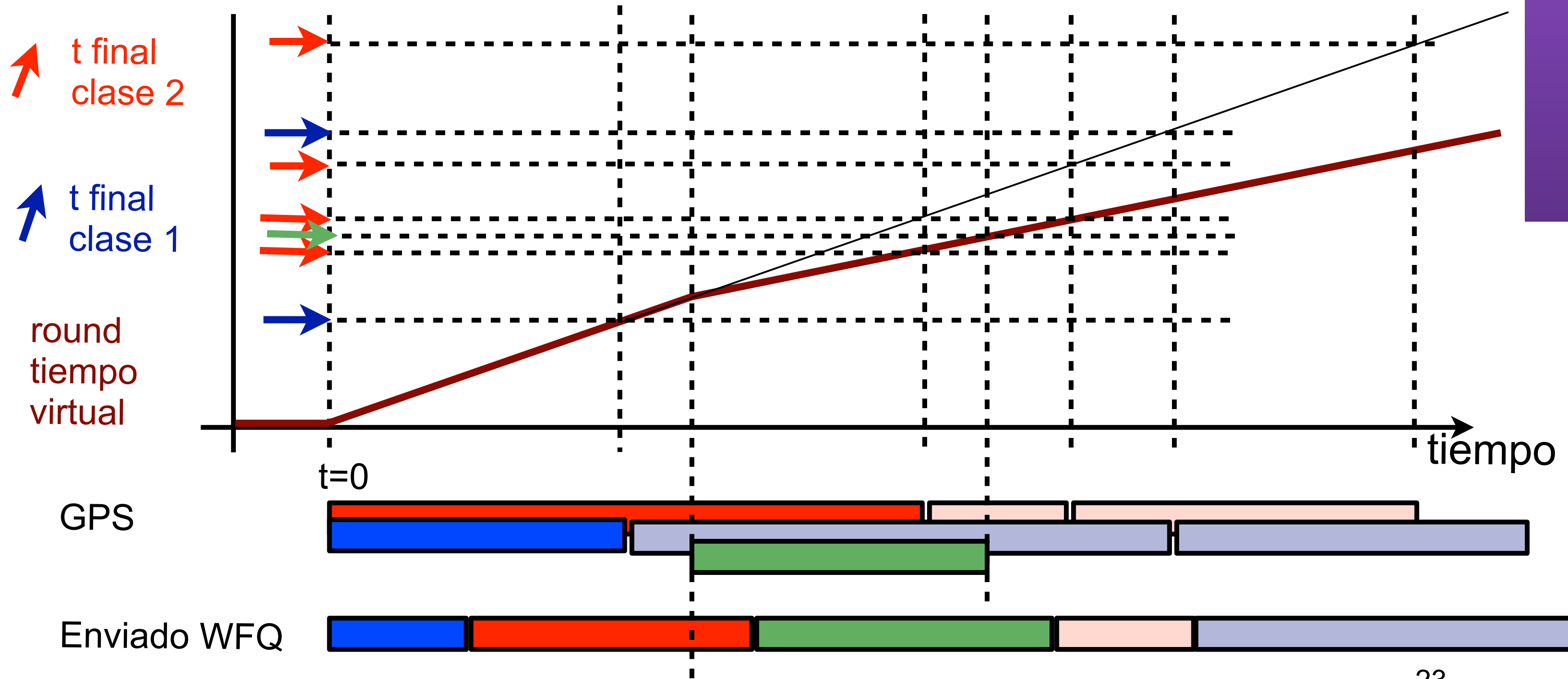
- Si han ido llegando mas paquetes despues de los primeros sus tiempos de finalizacion se han calculado y los paquetes se van sirviendo en ese orden
- La aproximacion nos dice como repartir los paquetes mas o menos igual que en GPS pero enviando solo de uno cada vez



WFQ ejemplo

- Si aparece un paquete de una clase nueva que se vuelve activa
- El tiempo virtual empieza a pasar mas despacio

  Pero los instantes de finalización siguen siendo validos, solo que al nuevo ritmo del tiempo tardarán más en alcanzarse
- Aparecen tiempos de finalización para los paquetes de la nueva clase



WFQ dificultades de implementacion

- En realidad la parte complicada es calcular el nuevo R(t) cada vez que llega un paquete

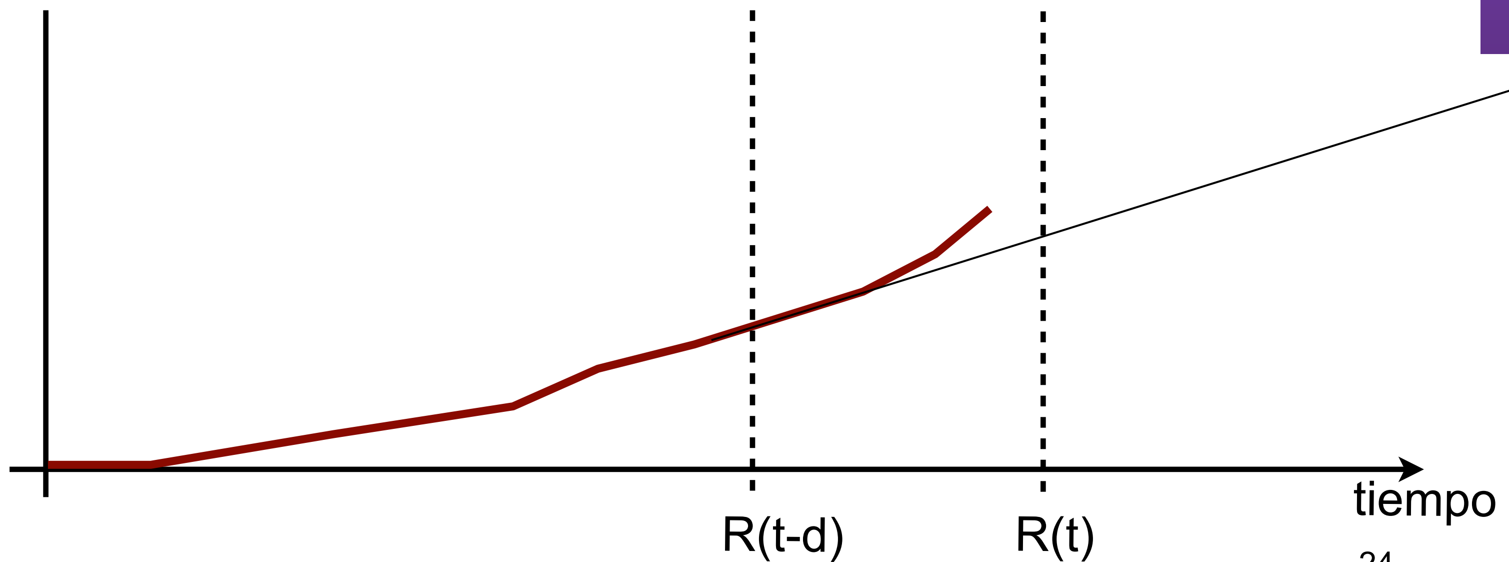
  Porque el scheduler no sabe los tiempos en los que las clases se quedan sin paquetes
- El calculo del R(t) es iterativo y no una operación simple
- Si tengo el R(t-d) cuando llego el paquete anterior... y las conexiones activas n que habia entonces

  R(t)=R(t-d)*n

  Salvo que el paquete que estaba transmitiendose entonces haya acabado

  en cuyo caso a partir del tiempo que acabo (y que calculo) si la conexion se volvió inactiva pudo subir la velocidad de R

  y entonces puede que le dio tiempo a acabar a otro paquete que pudo dejar una clase inactiva que pudo subir la velocidad...



Variantes WFQ

- La version con pesos se hace calculando el tiempo de finalización con $R(0) + s_1/peso$

- Hay variantes del WFQ que simplifican la implementación aunque hacen mas dificil demostrar que funcionan
  - SCFQ self clocking fair queueing

    Usa el instante de finalización del paquete en servicio como aproximacion de R
  - Start-time Fair Queueing

    Hace cálculos de los tiempos inicial y final de cada paquete
  - ...

Garantizando parámetros (beyond WFQ)

- WFQ permite dividir el ancho de banda de forma justa entre clases
- Eso implica protección de los flujos que envían lo que deben frente a los que envían de más
- Permite también calcular tiempos máximos de retraso por clase y con un control de admisión apropiado dar garantias en el ancho de banda y los retardos

Más schedulers...

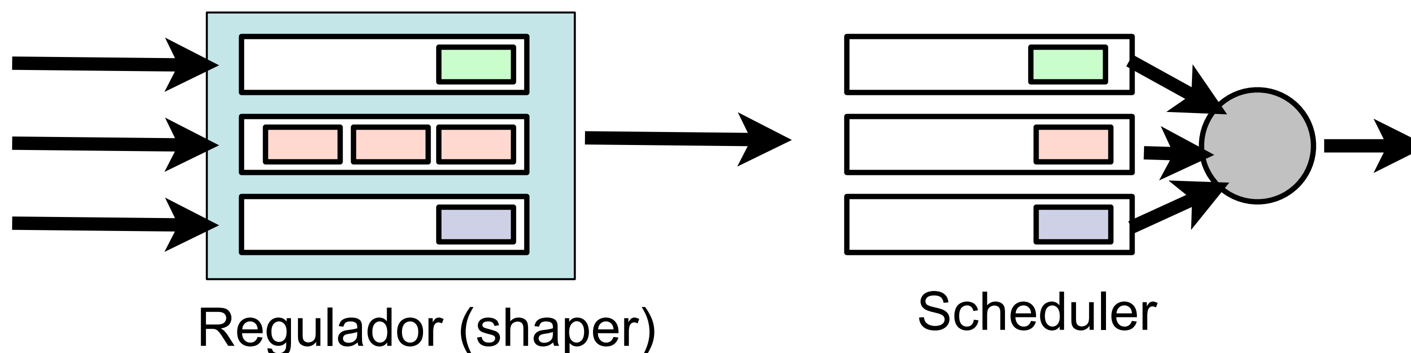
- Schedulers más sofisticados
- EDD earliest due date

  Los paquetes se etiquetan y ordenan segun un tiempo limite dado por su tiempo de llegada y un limite de retardo

  Permiten garantizar limites de retardo (Delay Earliest-due-date)
- Schedulers con regulador

  Antes del scheduler un regulador adapta el trafico para que no pase de unos limites de ancho de banda y tamaño de ráfaga

  El scheduler es atacado por trafico suavizado



Estos permiten propiedades más complejas como poner limtes al jitter (JEDD)

No son conservativos de trabajo (pueden tener un paquete y no servirlo hasta que llegue su tiempo)

Low latency queueing (LLQ)

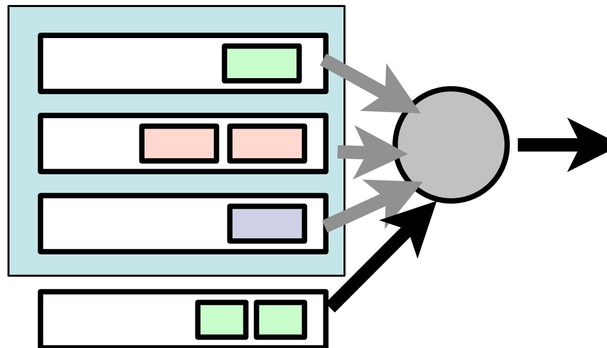
- En los sistemas reales es útil utilizar varias filosofías combinadas
- Low Latency Queuing

  Implementada en sistemas Cisco reales

  WFQ esta bien para repartir trafico entre clases por igual. Pero el trafico no elastico (i.e. voz) necesita prioridad total o suena mal
- Combinado WFQ con una cola de prioridad

  Si hay paquetes en la cola de prioridad se sirven

  Si no hay paquetes en la cola de prioridad se hace WFQ entre las colas sin prioridad



Conclusiones

- La planificación(scheduling) es un problema fundamental en arquitectura de routers y en muchos campos de redes
- Schedulers simples
  - FCFS
  - PQ Priority queueing
- Schedulers para conseguir max-min fairness
  - GPS (algoritmo ideal)
  - RR, WRR, DRR ... aproximaciones a GPS
  - WFQ/PGPS aproximaciones buenas a GPS
    - SCFQ, STFQ variantes de WFQ, implementaciones
- Schedulers para conseguir requisitos de BW, delay, jitter
  - DEDD
  - Schedulers con reguladores: JEDD ... (no conservativos)
- Schedulers combinando filosofias
  - LLQ