Reparto justo y protección

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

Grado en Ingeniería en Tecnologías de Telecomunicación, 3º

Reparto justo y protección

- Flujos con requerimientos estrictos deben tenerlos garantizados independiente de esta “justicia”
- Reparto justo es importante para flujos best-effort
- Reparto decimos que es justo si satisface un *max-min fair share*

- Scheduling es normalmente una decisión local al nodo de conmutación pero la “justicia” para un flujo es un objetivo global
- (...)

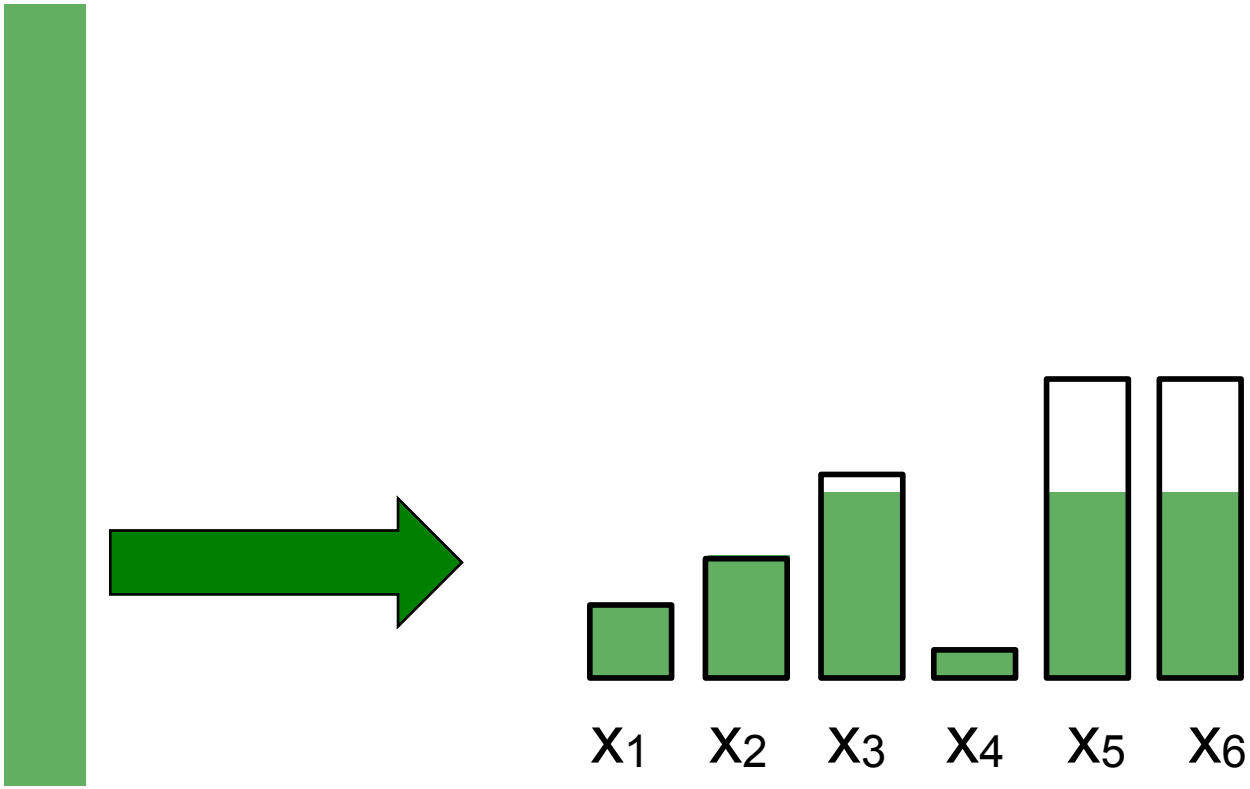
Reparto justo y protección

- Lograr justicia global con flujos cambiantes no es tan sencillo
- La protección implica que un flujo que envíe más que su asignación justa no afecte al resto
- Un planificador que haga un reparto justo ofrece protección
- La justicia es una característica del resultado del reparto, independientemente de cómo se llegue a él
- (...)

Reparto justo (*max-min fair*)

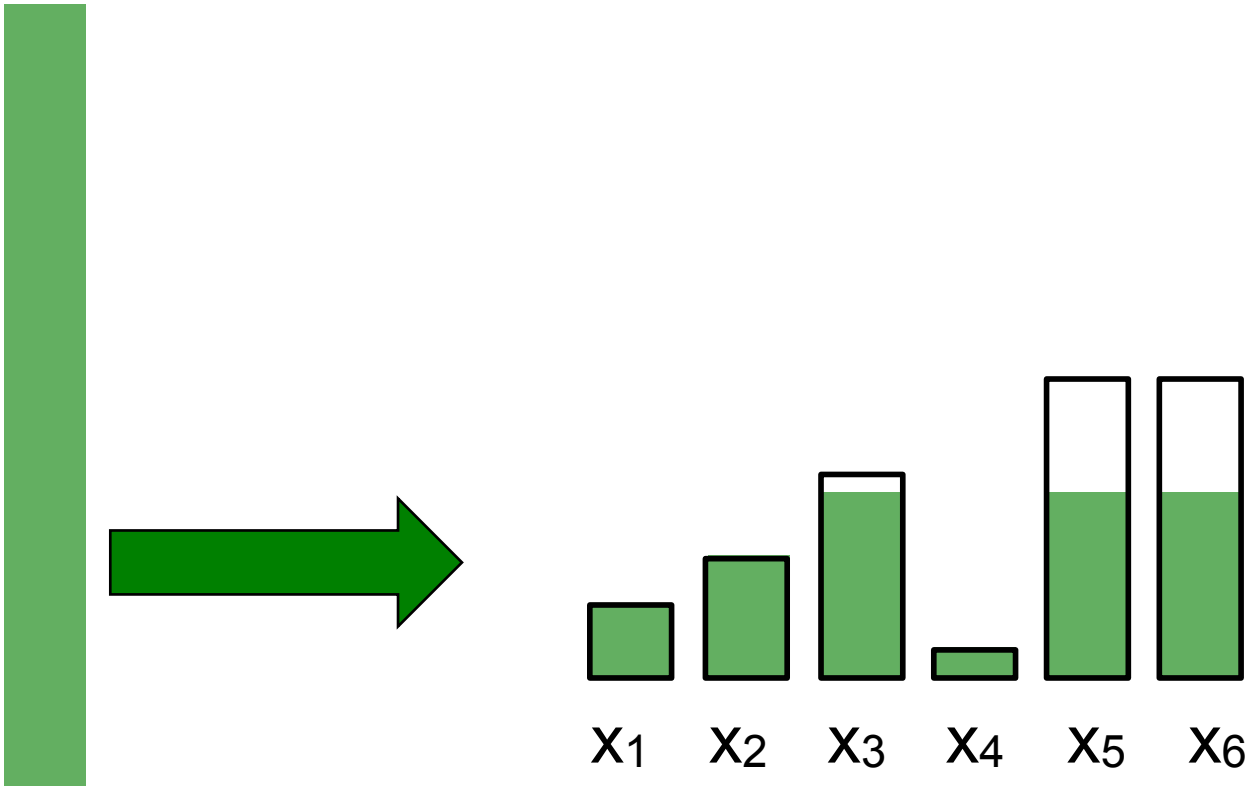
Reparto justo (max-min fair)

- Para dividir recursos entre un conjunto de usuarios, todos con iguales “derechos” pero diferentes demandas
- De forma simple: a los que piden “poco” se les da lo que piden y lo que sobra se reparte entre los que piden “mucho”
- En una situación de reparto max-min-fair, para mejorar la asignación a un flujo que no consigue todo lo que solicita hay que quitarle a otro que tiene asignado tanto como él o menos

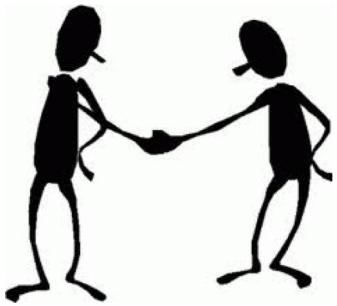


Reparto justo (max-min fair)

- Todos reciben al menos su parte proporcional, salvo que hayan pedido menos
- Asignar recursos en orden creciente de demanda
- Ningún cliente recibe más de lo que solicita
- Aquellos cuya demanda no se pueda satisfacer se reparten el remanente del recurso



Reparto justo (max-min fair)



- Flujos 1,...,n
- Demandas $x_1,...,x_n$
- Demandas ordenadas $x_1 \leq x_2 \leq ... \leq x_n$
- Capacidad a repartir C
- Inicialmente asignar C/n al flujo 1
- Si esto es más que lo que necesita ($C/n > x_1$) lo que sobra se repartirá entre el resto
- Asignar al flujo 2: C/n más la parte que le corresponde de lo que sobró del flujo 1, es decir:

$$C/n + \frac{C/n - x_1}{n-1}$$

- Esto puede ser más que lo que el flujo 2 necesita, así que lo que sobra se puede repartir entre el resto
- *Al final todos tienen lo que han pedido o si no había suficiente para eso no tienen menos que cualquier otra fuente que ha pedido más*

Max-min Fair (Ejemplo)

- Recurso: 10
- Demandas: 2, 2.6, 4 y 5
- 10/4 = 2.5
  - Demasiado para el primer cliente
  - Asignarle 2 y queda 0.5
- Ese 0.5 repartirlo entre los otros 3:
  - 0.5/3 = 0.16666…
  - Asignaciones [2, 2.66, 2.66, 2.66]
  - Demasiado para el segundo cliente
  - Asignarle 2.6 y quedan 0.0666….
- Repartir ese 0.0666… entre los otros 2:
  - (2.5+0.5/3-2.6)/2 = 0.03333…
  - Asignaciones [2, 2.6, 2.7, 2.7]

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Tecnologías Avanzadas de Red
Área de Ingeniería Telemática

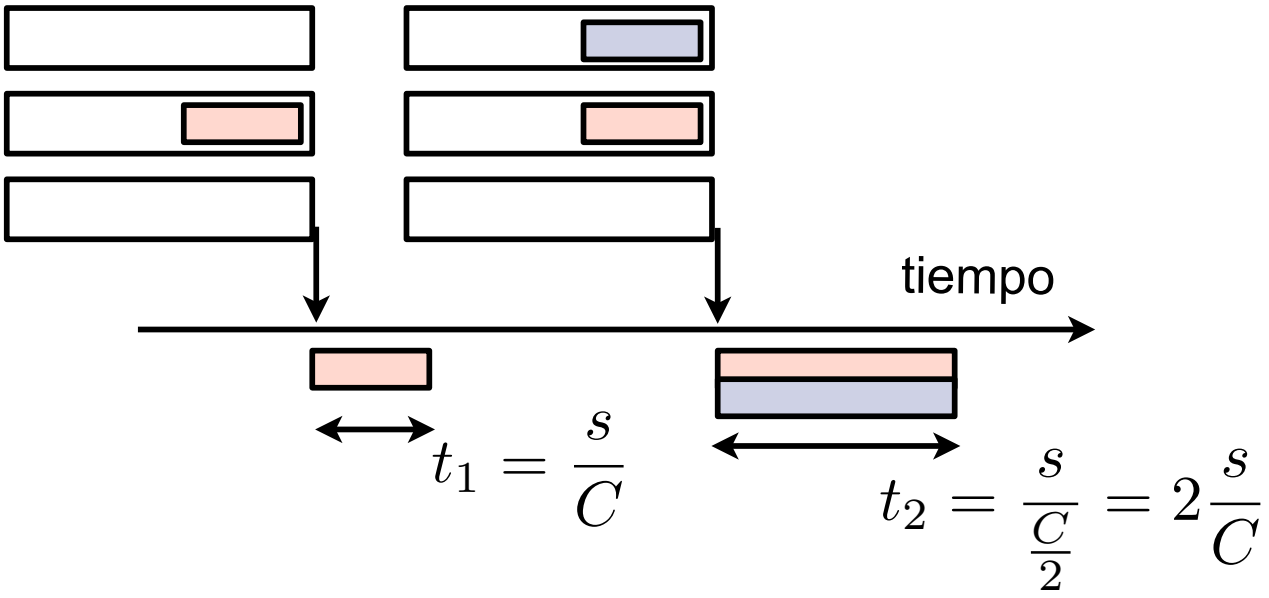
Processor Sharing

PS

- Para best-effort querríamos un reparto *max-min fair*
- Esto se puede lograr con un scheduler llamado *Processor Sharing*
- Es un planificador *work-conserving*
- Sirve de forma simultanea todas las colas, repartiendo la capacidad
- O se puede decir que las sirve por turnos (round robin) pero sirviendo una cantidad infinitesimal de cada una
- (…)

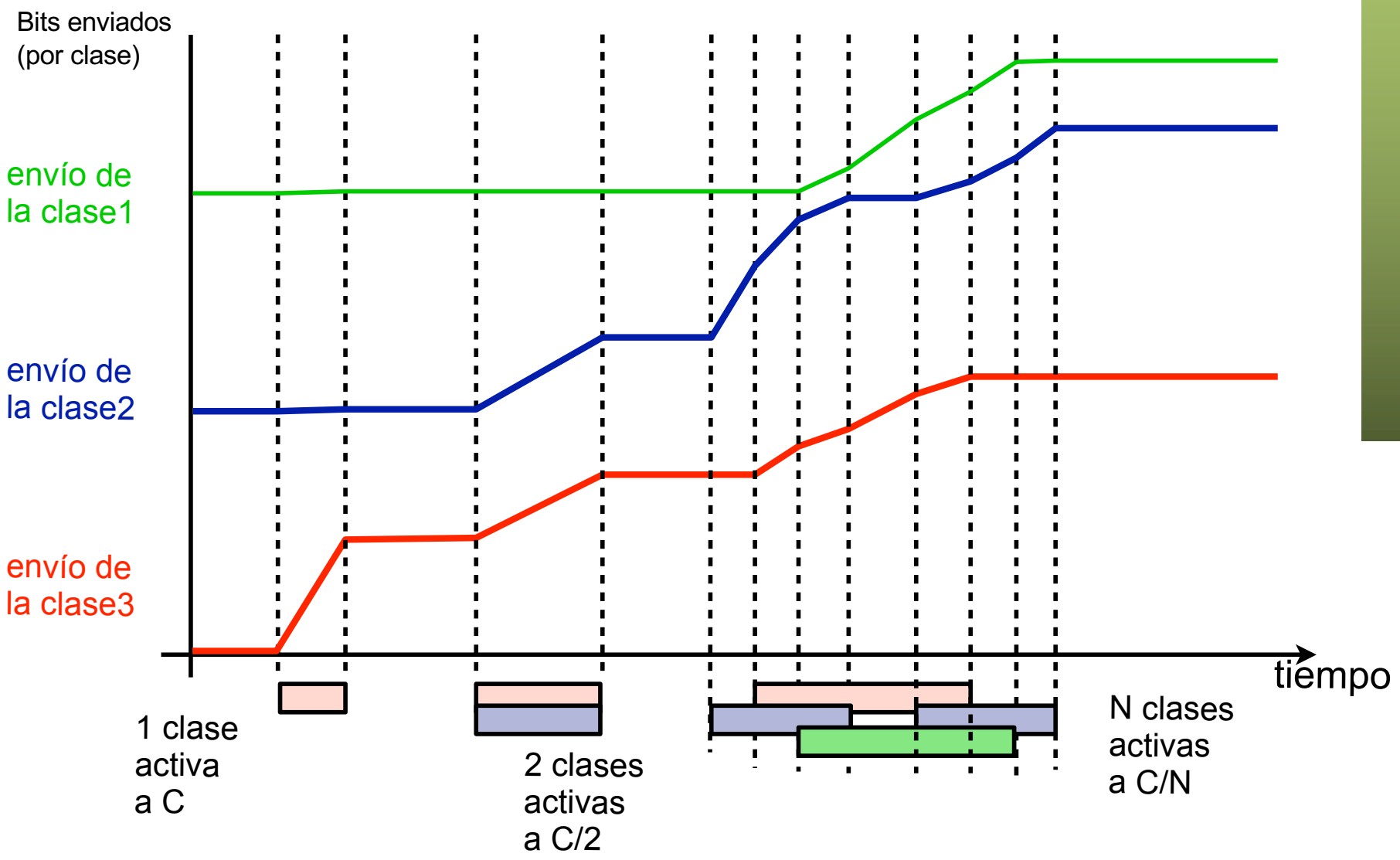
PS

- Si una cola está vacía pasa a la siguiente, de forma que su tiempo se está repartiendo entre el resto (y de ahí el max-min)
- Aproximación de tráfico como un fluido
- Es un planificador ideal e imposible de implementar, aunque se puede aproximar



Processor Sharing

Ejemplo



Round Robin

Round Robin (RR)

- Opera en “turnos” (*rounds*), conservativo en trabajo
- En cada turno visita cada cola (en *round-robin*)
- En cada cola FCFS
- Se sirve un número de paquetes o paquetes durante un cierto tiempo fijo (la diferencia es cómo afectan sus tamaños)