Multicost (or QoS) routing

Each link j of the network is assigned a k-dimensional cost vector

$$\mathbf{v_j} = (v_{1j}, v_{2j}, \ldots, v_{kj}),$$

Each path is assigned a cost vector

$$\mathbf{V} = (V_1, V_2, \ldots, V_k)$$

Minimize $f(V)=f(V_1,\ldots,V_k)$ over all paths

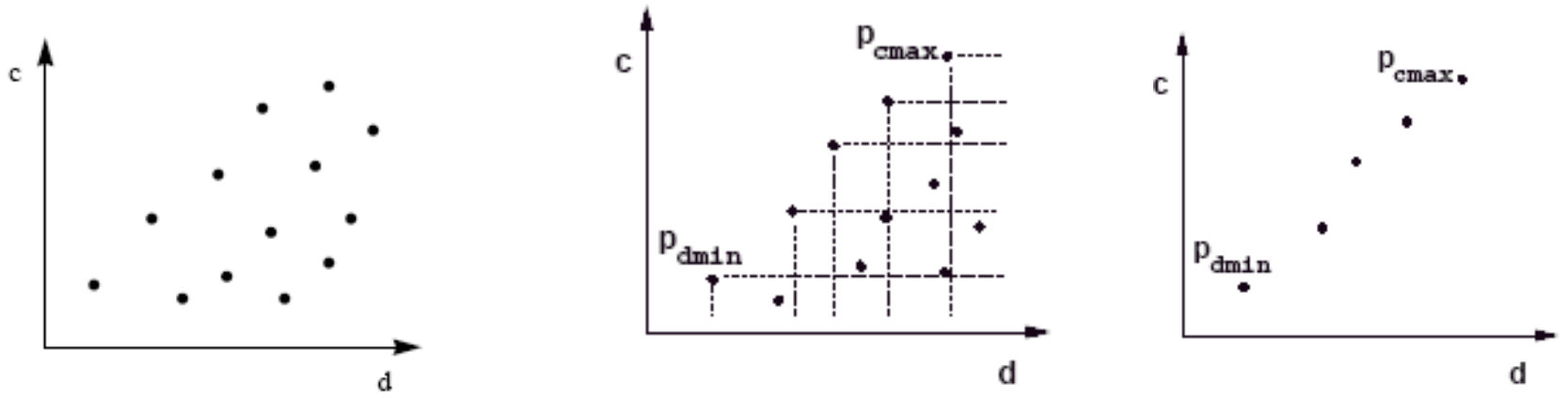
For example:

- $V_i = \sum_{j=1}^{l} v_{ij}$, $v_{ij} \geq 0$, and f is monotonically increasing in V_i (so our objective is to minimize V_i), or
- $V_i = \min_{j=1,\ldots,l}\{v_{ij}\}$, $v_{ij} \geq 0$, and f is monotonically decreasing in V_i (so our objective is to maximize V_i).

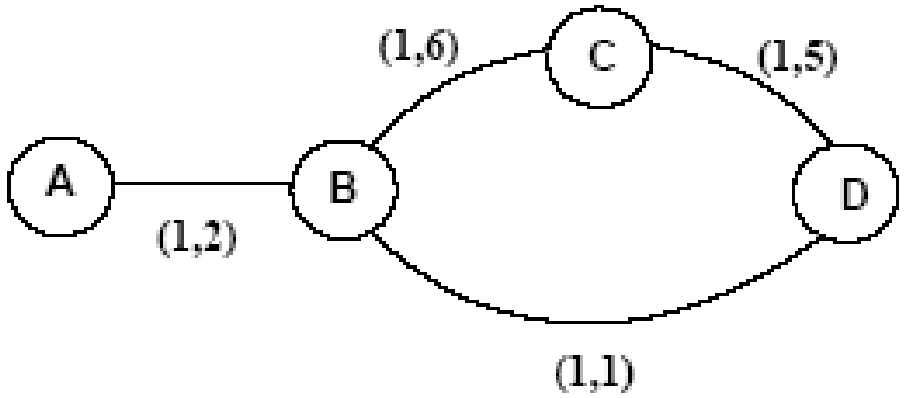
More generally, $V_i = \oplus_{j=1}^{l} v_{ij}$.

Cost of a path: $f(V_1, V_2, \ldots, V_k)$

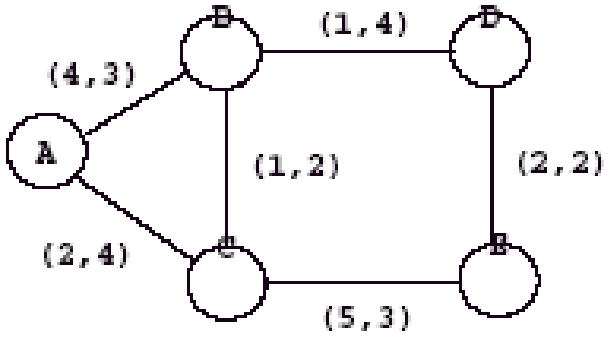
Lemma 1: The set P_{opt} of optimum paths between two nodes is contained in the set Q of non-dominated paths, that is, $P_{opt} \subset Q$.



(b)



Non dominated paths	Function $f = \frac{h}{c}$	Function $f = \frac{h^2}{c}$
A→D	**A→D**	**A→D**
$(A, B, C, D), (3, 2)$	$f(3, 2) = 1.5$ *	$f(3, 2) = 4.5$
$(A, B, D), (2, 1)$	$f(2, 1) = 2$	$f(2, 1) = 4$*
B→D	**B→D**	**B→D**
$(B, C, D), (2, 5)$	$f(2, 5) = 0.4$*	$f(2, 5) = 0.8$*
$(B, D), (1, 1)$	$f(1, 1) = 1$	$f(1, 1) = 1$

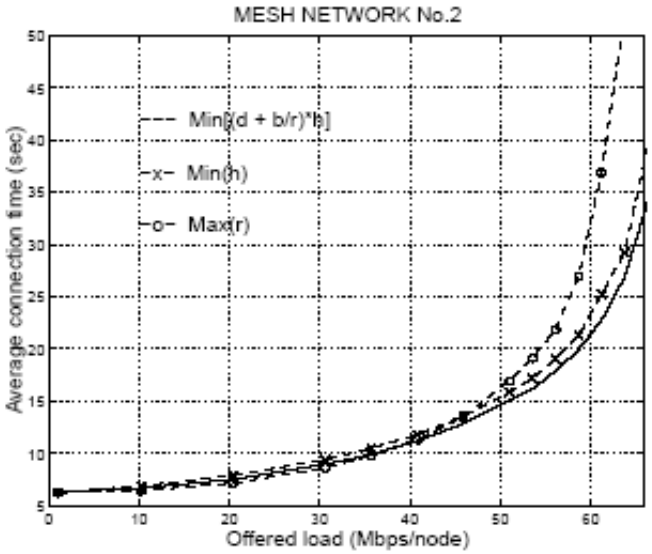


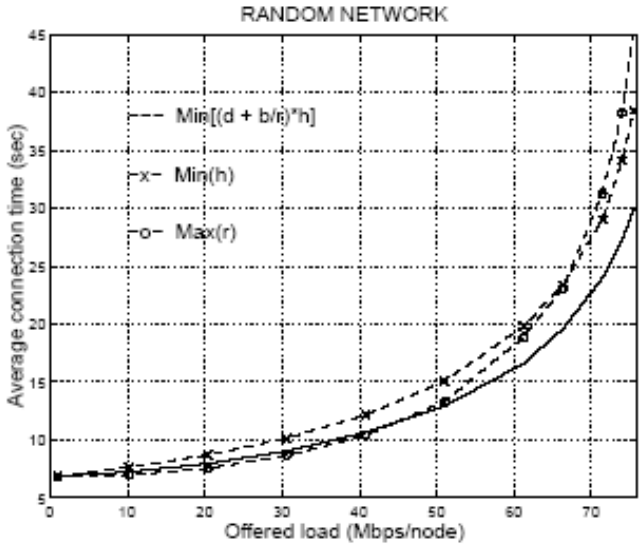
B	(4,3,A)	(1,2)+(2,4) (3,2,C)	(3,2,C) (4,3,A)		(3,2,C) (4,3,A)		(3,2,C) (4,3,A)	
C	(2,4,A)		(2,4,A)		(2,4,A)	(1,2)+(4,3) (5,2,B)	(2,4,A)	
D	(∞,0,-)		(∞,0,-)	(1,4)+(3,2) (4,2,B)	(4,2,B)	(1,4)+(4,3) (5,3,B)	(4,2,B) (5,3,B)	
E	(∞,0,-)	(5,3)+(2,4) (7,3,C)	(7,3,C)		(7,3,C)		(7,3,C)	(2,2)+(4,2) (6,2,D)

B	(3,2,C) (4,3,A)		(3,2,C) (4,3,A)		(3,2,C) (4,3,A)		(3,2,C) (4,3,A)
C	(2,4,A)		(2,4,A)	(5,3)+(6,2) (11,2,E)	(2,4,A)		(2,4,A)
D	(4,2,B) (5,3,B)		(4,2,B) (5,3,B)		(4,2,B) (5,3,B)	(2,2)+(7,3) (9,2,E)	(4,2,B) (5,3,B)
E	(6,2,D) (7,3,C)	(2,2)+(5,3) (7,3,B)	(6,2,D) (7,3,C)		(6,2,D) (7,3,C)		(6,2,D) (7,3,C)

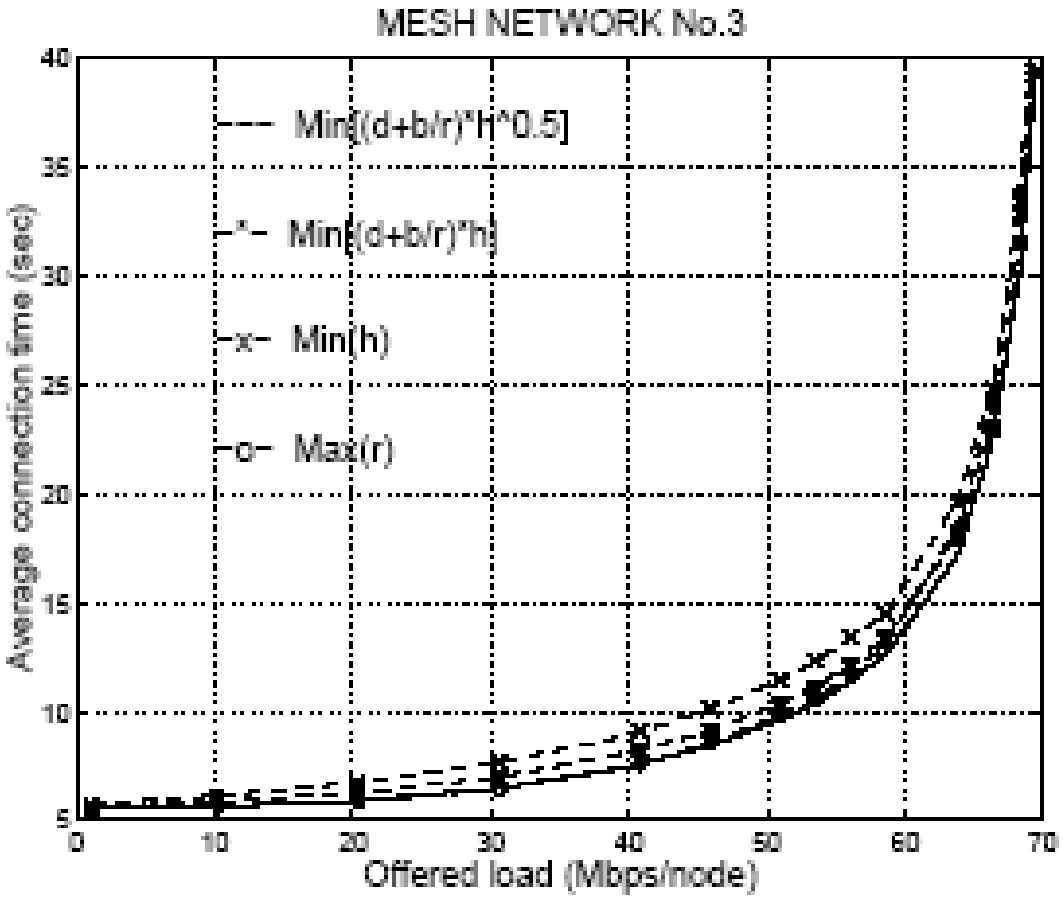
Examples of cost functions

$$f(h,r) = \frac{h}{r} \qquad f(d,b,r) = d + \frac{b}{r} \qquad f(d,b,r,h) = \left(d + \frac{b}{r}\right) * h.$$

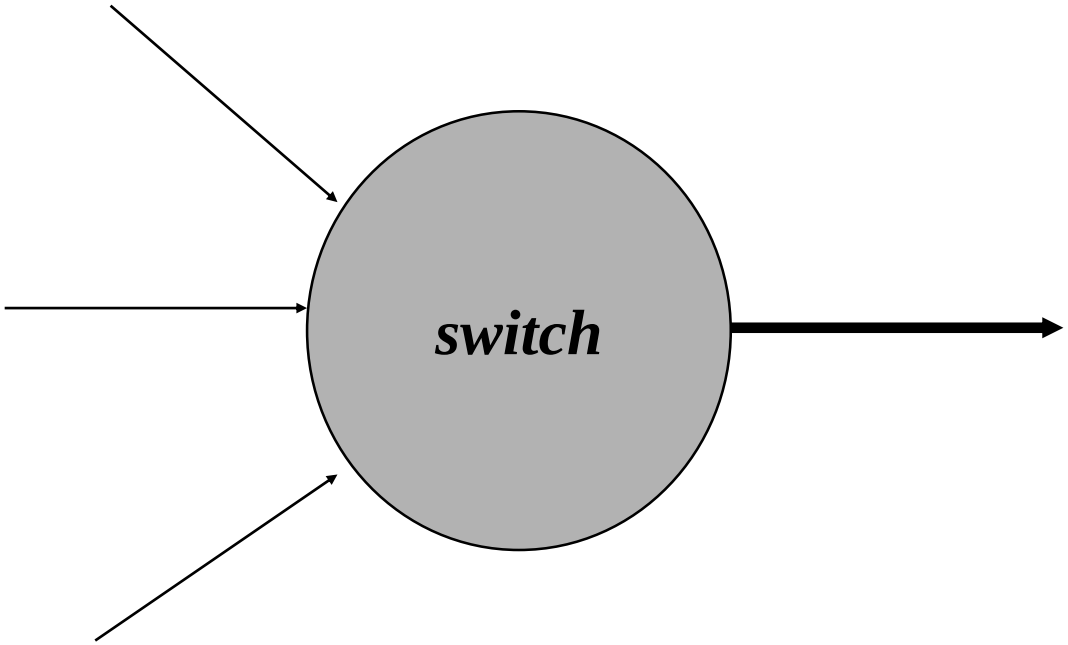




$$f(d, b, r, h, n) = \left(d + \frac{b}{r}\right) * h^n.$$



Scheduling



- ***Sharing always results in contention***
- ***A scheduling discipline resolves contention and decides order: who's next?***
- ***The objective is to share resources fairly and provide performance guarantees***

What can scheduling disciplines do?

- **Give different users different QoS (example: passengers waiting to board a plane). Best effort vs guaranteed service**
- **Scheduling disciplines can allocate bandwidth, delay, loss**
- **They also determine how *fair* the network is**

Requirements

- **An ideal scheduling discipline is easy to implement, is fair, provides performance bounds, etc**

Notion of fairness

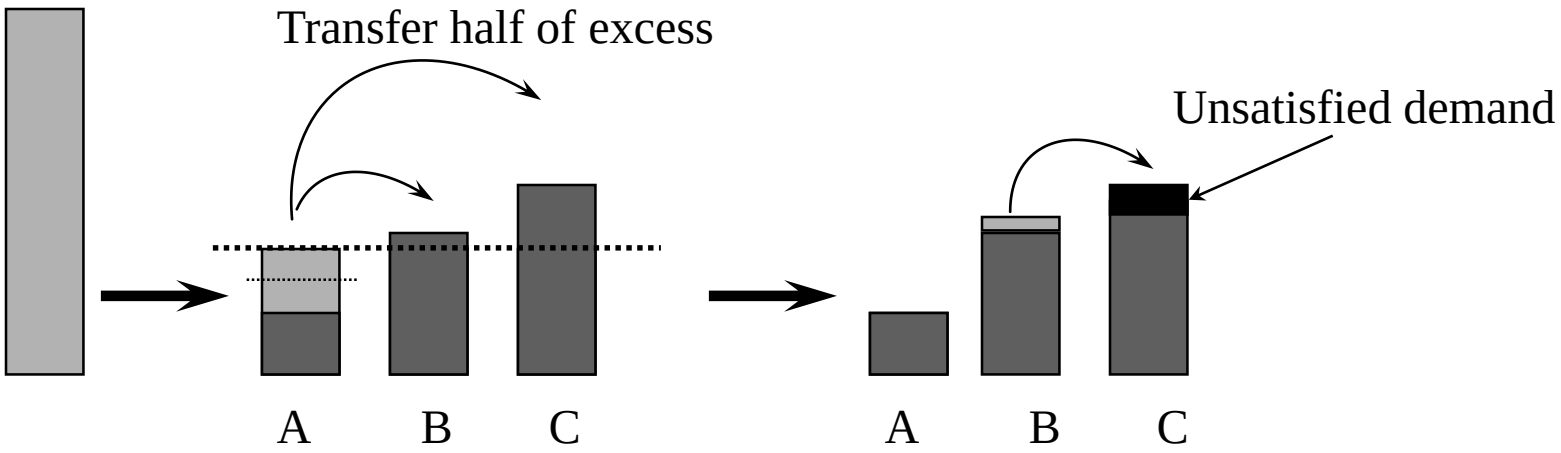
Τι είναι δίκαιο?

- *Ο καθένας παίρνει το ίδιο?*
- *Ο καθένας παίρνει αυτό που θέλει? (Τι γίνεται με το congestion?)*
- *Πως σχετίζεται η δικαιοσύνη με την ικανότητα κάποιου να χρησιμοποιήσει ένα resource?*
- *Τι γίνεται με τα excess resources?*
- *Πως παίρνουμε υπ' όψιν μας την δυνατότητα κάποιου να πληρώσει παραπάνω κόστος για ένα resource?*

Π.χ. φορολογικό σύστημα (flat tax rate ή progressive tax rate ή social based)

max-min fairness

- Intuitively: each connection gets no more than what it wants, and the excess, if any, is equally shared



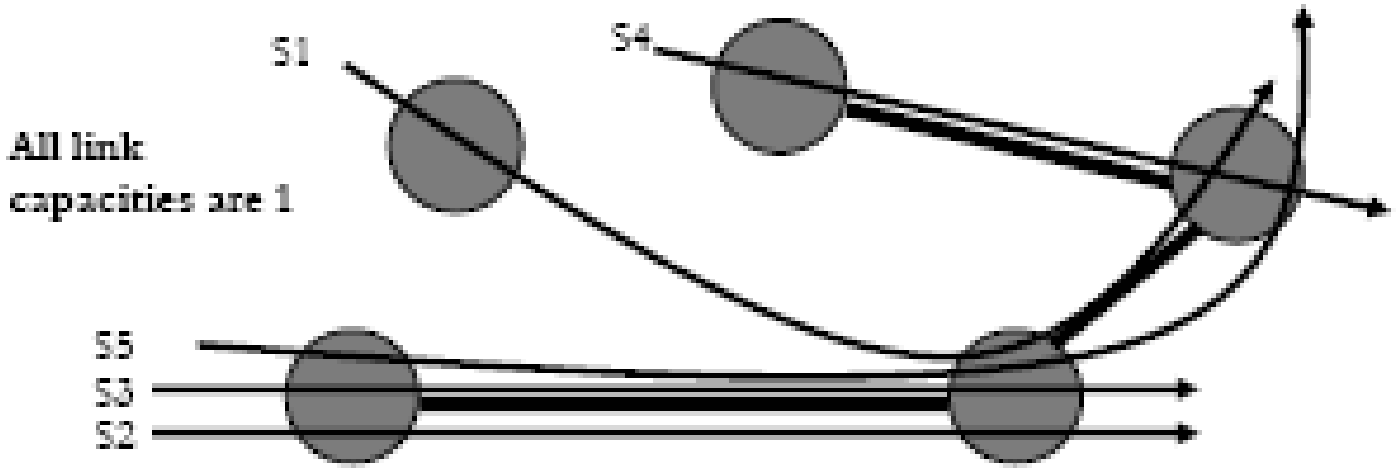
Demanded Rates	Max-Min Fair Sharing (First iteration)	Max-Min Fair Sharing (Second iteration)	Fair Rates
10	7.5	10	10
3	3	3	3
5	5	5	5
15	7.5	11	12
Residue	30-23=7	1	0

Table 1: **An example of the non-weighted Max-Min fair Sharing algorithm if the overall processor capacity is 30.**

*There is also **weighted** max-min fairness*

Max-Min Flow Control

- A rate allocation is max-min fair if no rate can be increased without decreasing another rate with a smaller or equal value



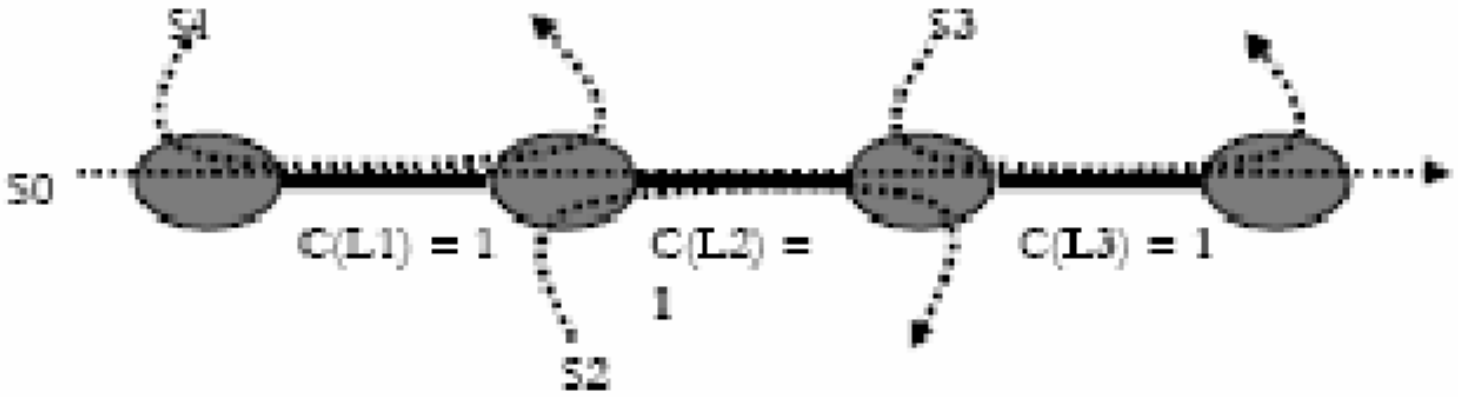
1. {1/3, 1/3, 1/3, 1/3, 1/3}
2. {2/3, 1/3, 1/3, 2/3, 1/3}
3. {2/3, 1/3, 1/3, 1, 1/3}

Fairness is a *global* objective, but scheduling is local

Proportional Fair (PF)

The allocation below would be

- Max-Min Fair: {0.5,0.5,0.5,0.5}
- Proportional Fair: {0.25,0.75,0.75,0.75}



Performance guarantees

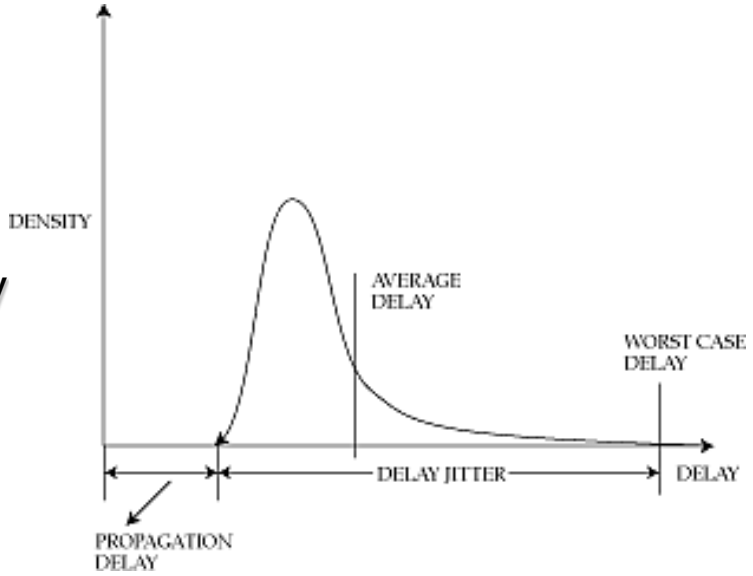
- This is a way to obtain a desired level of service
- Can be *deterministic* or *statistical*
- Common parameters are bandwidth, delay, delay-jitter, loss

Bandwidth

- Specified as minimum bandwidth measured over a prespecified interval (e.g. > 5Mbps over intervals of > 1 sec)
- Meaningless without an interval!
- Can be a bound on average (sustained) rate or peak rate

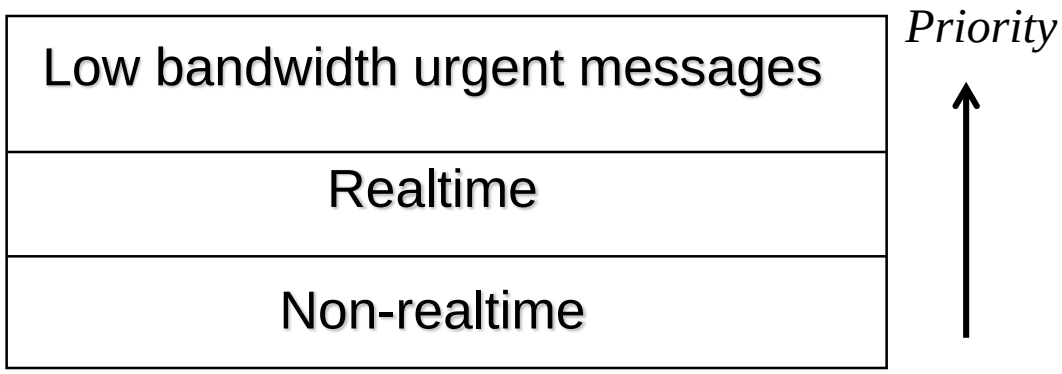
Delay and delay-jitter

- Bound on some parameter of the delay distribution curve



Priority

- Packet is served from a given priority level only if no packets exist at higher levels
- Highest level gets lowest delay
- Watch out for starvation!



Fundamental choices

1. Number of priority levels
2. Degree of aggregation
3. Service order within a level

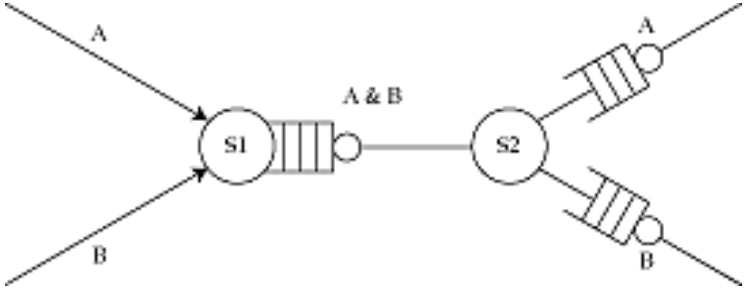
Degree of aggregation

- **More aggregation**
  - **less state**
  - **cheaper (e.g. smaller VLSI)**
  - **BUT: less individualization/differentiation**
- **Solution**
  - **aggregate to a *class*, members of class have same performance requirement**
  - **no protection within class**

  Issue: what is the appropriate class definition?

Work conserving vs non-work-conserving

- **Work conserving discipline is never idle when packets await service**
- **Why bother with non-work conserving?**

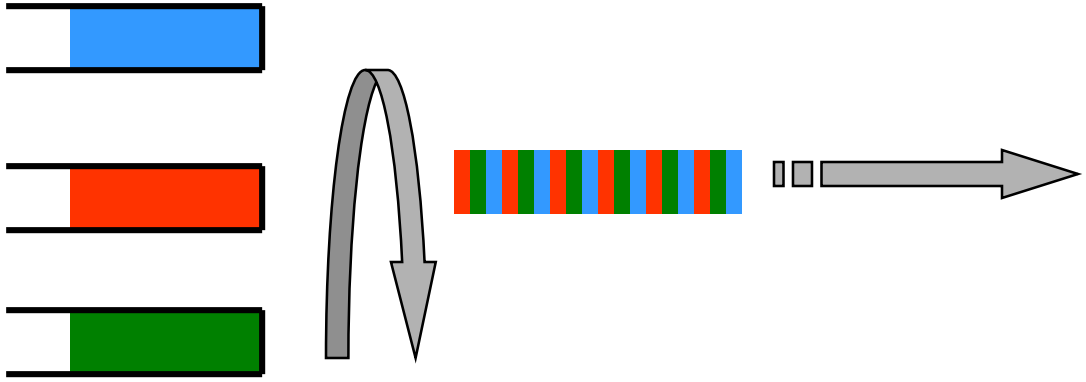


Non-work-conserving disciplines

- **Key conceptual idea: delay packet till *eligible***
- **Reduces delay-jitter => fewer buffers in network**
- **Increases mean delay (not a problem for *playback* applications)**
- **Wastes bandwidth (can serve best-effort packets instead)**
- **Always punishes a misbehaving source**
- **How to choose eligibility time?**
  - **rate-jitter regulator (bounds maximum outgoing rate)**
  - **delay-jitter regulator (compensates for variable delay at previous hop)**

Scheduling best-effort connections

- Main requirement is *fairness (in the max-min sense)*
- Achievable using *Generalized processor sharing (GPS)*
  - Visit each non-empty queue in turn
  - Serve infinitesimal from each
  - This is exactly the definition of max-min fairness (but impractical)
  - How can we give weights to connections?



- We can't implement GPS, so, lets see how to emulate it
- We want to be as fair (i.e. as close to GPS) as possible
- But also have an efficient implementation

Weighted round robin

- **Serve a packet from each non-empty queue in turn**
- **Unfair if packets are of different length or weights are not equal**
- **Solution: Normalize weights by mean packet size**
  - **e.g. weights {0.5, 0.75, 1.0}, mean packet sizes {50, 500, 1500}**
  - **normalize weights: {0.5/50, 0.75/500, 1.0/1500} = { 0.01, 0.0015, 0.000666}, normalize again {60, 9, 4}**

Problems with Weighted Round Robin

- **With variable size packets and different weights, need to know mean packet size in advance**
- **Can be unfair for long periods of time. E.g.:**
  - **T3 trunk with 500 connections, each connection has mean packet length 500 bytes, 250 with weight 1, 250 with weight 10**
  - **Each packet takes 500 * 8/45 Mbps = 88.8 µs**
  - **Round time =2750 * 88.8 = 244.2 ms**

Weighted Fair Queueing (WFQ)

- **Deals better with variable size packets and weights**
- **GPS is fairest discipline**
- **Find the *finish time* of a packet, *had we been doing GPS***
- **Then serve packets in order of their finish times**

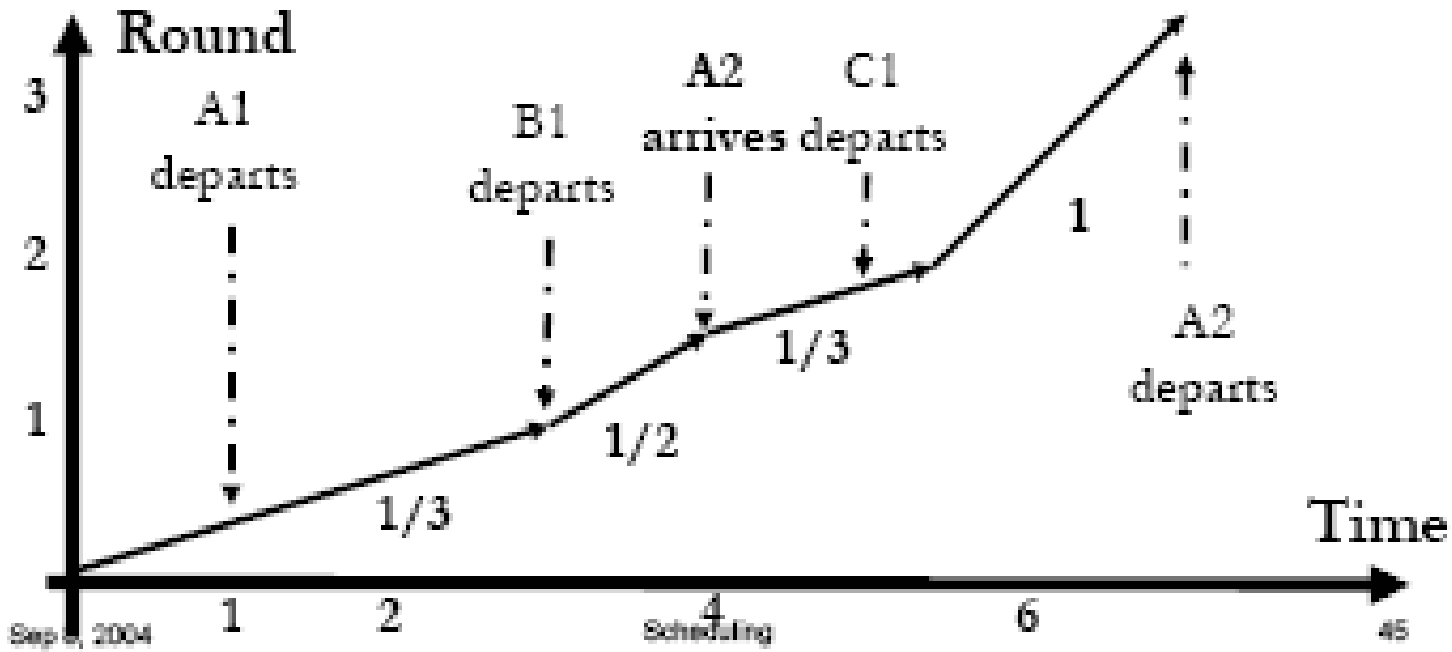
WFQ: first cut

- **Suppose, in each *round,* the server served one bit from each active connection**
- ***Round number* is the number of rounds already completed (can be fractional)**
- **If a packet of length *p* arrives to an empty queue when the round number is *R*, it will complete service when the round number is *R* + *p* => *finish number* is *R* + *p* (independent of the number of other connections)**
- **If a packet arrives to a non-empty queue, and the previous packet has a finish number of *f*, then the packet's finish number is *f+p***
- **Serve packets in order of finish numbers**

WFQ: computing the round number

Example

- Three connections: A,B,C. At t=0, packet of size 1,2 and 2 arrives. (A1,B1,C1). Finish time: A1 = 1, B1 = C1 = 2.
- With GPS, at t=3, round 1 is completed, A1 departs, only 2 connections active
- At t=4, round is 1.5, A2 of size 2 arrives, finish time is (1.5+2) 3.5



WFQ implementation

- On packet arrival:
  - use source + destination address (or VCI) to classify it and look up finish number of last packet waiting to be served
  - recompute round number
  - compute finish number
  - insert in priority queue sorted by finish numbers
  - if no space, drop the packet with largest finish number
- On service completion
  - select the packet with the lowest finish number

Scheduling guaranteed-service connections

- With best-effort connections, goal is fairness
- With guaranteed-service connections what performance guarantees are achievable?

WFQ

- Turns out that WFQ also provides performance guarantees
- Bandwidth bound (=ratio of weights * link capacity)
  - e.g. connections with weights 1, 2, 7; link capacity 10
  - connections get at least 1, 2, 7 units of b/w each
- End-to-end delay bound
  - assumes that the connection is *leaky-bucket* regulated
  - # bits sent in time $[t_1, t_2] <= r\ (t_2 - t_1) + \sigma$

Parekh-Gallager theorem

- *g*= least bandwidth a connection is allocated at a WFQ scheduler
- The connection pass through *K* schedulers, where the *k*th has rate *r(k)*
- *P* = the largest packet allowed in the network be *P*

$$end\_to\_end\_delay \leq \sigma / g + \sum_{k=1}^{K-1} P / g + \sum_{k=1}^{K} P / r(k)$$

Significance

- Theorem shows that WFQ can provide end-to-end delay bounds
- So WFQ provides both fairness and performance guarantees
- Bound holds regardless of cross traffic behavior