

Packet Scheduling for QoS management Part II

TELCOM2321 – CS2520

Wide Area Networks

Dr. Walter Cerroni

University of Bologna – Italy

Visiting Assistant Professor at SIS, Telecom Program

Slides partly based on Dr. Znati's material

Readings

- Textbook Chapter 17, Section 2

Max-Min weighted fair share

- Max-Min fair share assumes that all flows have equal right to resources
- Different flows might have different rights due to different QoS requirements
 - e.g. voice and video flows require different bandwidth levels
 - customers of video service are willing to pay more to get the required bandwidth
 - they must have the same chances as voice customer to get what they need
- Associate a weight to each demand to take into account different requirements
 - normalize demands with corresponding weight
 - use weights to compute fair share

Max-Min weighted fair share: Example

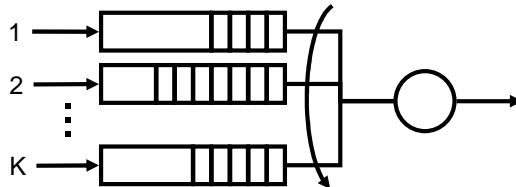
- Demands: $d_1 = d_2 = 10$, $d_3 = d_4 = 25$, $d_5 = 50$
- Bandwidth: $C = 100$
- Max-Min satisfies d_1 , d_2 , d_3 , and d_4 while d_5 gets 30
- If d_5 has the same right as the others, use weighted Max-Min:
 1. Weights: $w_1 = w_2 = w_3 = w_4 = 1$, $w_5 = 5$
 2. Weighted demands: $d'_1 = d'_2 = 10$, $d'_3 = d'_4 = 25$, $d'_5 = 50 / 5 = 10$
 3. $FS = C / (\text{sum of weights}) = 100 / 9 = 11.11$
 4. d'_1 , d'_2 and d'_5 satisfied, d_5 gets 50
 5. Sum of weights of unsatisfied demands: $M = w_3 + w_4 = 2$
 6. Bandwidth left: $R = 100 - (10 + 10 + 50) = 30$
 7. $FS = 30 / 2 = 15$
 8. d'_3 and d'_4 not satisfied \rightarrow they get 15

Fairness in Work Conserving Schedulers

- If a scheduling policy guarantees fair resource sharing (e.g. Max-Min) it is also protective
 - a misbehaving source do not get more than fair share
- Resources should be assigned based on QoS requirements
 - Max-Min weighted fair share
 - weights limited by conservation law
- Fair work conserving scheduling policies
 - Fair Queuing
 - Processor Sharing
 - Generalized Processor Sharing
 - Weighted Fair Queuing

Fair Queuing

- A router maintains multiple queues
 - separate queues for different traffic flows or classes
 - queues are serviced by a round-robin scheduler
 - empty queues are skipped over



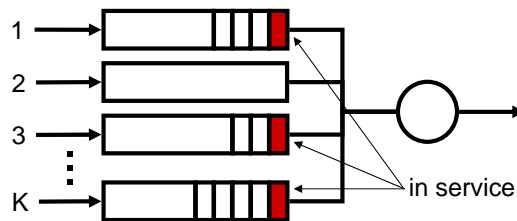
- Greedy connections face long delays, but other connections are not affected by it
- Drawbacks
 - no service differentiation
 - the scheme penalizes flows with short packets

Processor Sharing

- The server is able to serve multiple customers at the same time
- Fluid model
 - traffic is infinitely divisible
 - service can be provided to infinitesimal amounts of data
 - the packet size does not count anymore
 - if N customer are served by server with capacity C , capacity seen by each customer is C / N
 - ideal model to provide an upper bound on the fairness of other schemes

Head-of-Queue Processor Sharing

- Separate queues for different traffic flows or classes
 - head-of-queue packets are served from nonempty queues
 - following packets wait to be serviced
 - a nonempty queue is always served
- Each queue is assigned a fair share of server capacity
 - when some flows become silent, their capacity is equally split among the other flows → Max-Min fair share
 - independently of packet size

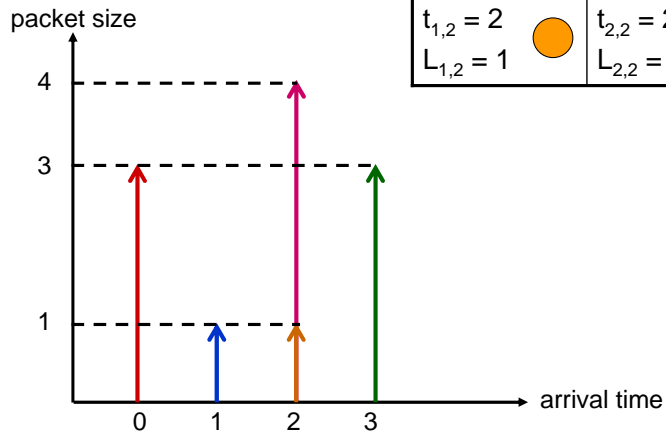


Processor Sharing: Virtual time

- $0 \leq N(t) \leq K$, number of nonempty queues at time t
- $r(t) = C / N(t)$, capacity available to each nonempty queue when $N(t) > 0$
- $t_{k,i}$, actual arrival time of packet i to queue k
- $L_{k,i}$, length of packet i in queue k
- Actual service time depends on $r(t)$
 - actual service time stretches or shrinks based on arrival and departure events
- It is useful to define a virtual time scale $v(t)$
 - continuous piecewise linear function of t with slope $1/N(t)$
- Now it is possible to define
 - $s_{k,i}$, virtual starting transmission time of packet i in queue k
 - $f_{k,i}$, virtual ending transmission time of packet i in queue k

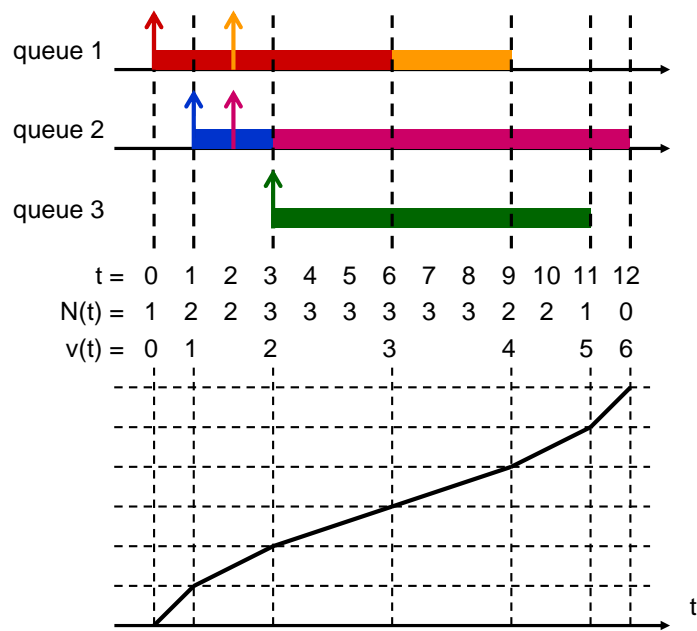
Processor Sharing: Example

$C = 1$



Queue 1	Queue 2	Queue 3
$t_{1,1} = 0$ $L_{1,1} = 3$ ●	$t_{2,1} = 1$ $L_{2,1} = 1$ ●	$t_{3,1} = 3$ $L_{3,1} = 3$ ●
$t_{1,2} = 2$ $L_{1,2} = 1$ ●	$t_{2,2} = 2$ $L_{2,2} = 4$ ●	

Processor Sharing: Example



Virtual starting and ending times

Recursive definition

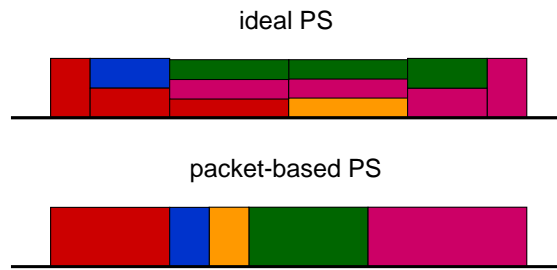
- $f_{k,i} = s_{k,i} + L_{k,i}/C$
- $s_{k,i} = \max [f_{k,i-1}, v(t_{k,i})]$

In our example

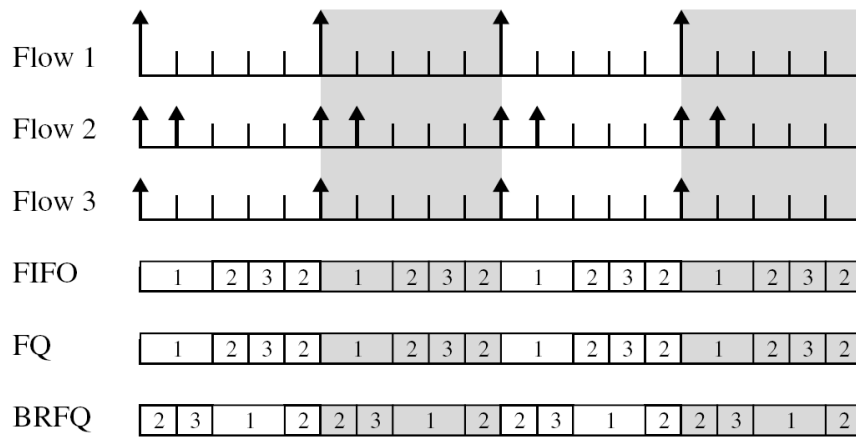
- $s_{1,1} = 0, f_{1,1} = 3$
- $s_{2,1} = 1, f_{2,1} = 2$
- $s_{2,2} = 2, f_{2,2} = 6$
- $s_{3,1} = 2, f_{3,1} = 5$
- $s_{1,2} = 3, f_{1,2} = 4$

Packet-based Processor Sharing

- Real-life schedulers are not able to serve infinitesimal amounts of data
- A packet-based scheduling policy that approximates the PS scheme can be implemented as follows
 - compute virtual starting and ending times as in the ideal PS
 - when a packet finishes its service, next packet to be serviced is the one with smallest virtual finishing time
 - In our example:
 - $s_{1,1} = 0, f_{1,1} = 3$
 - $s_{2,1} = 1, f_{2,1} = 2$
 - $s_{1,2} = 3, f_{1,2} = 4$
 - $s_{3,1} = 2, f_{3,1} = 5$
 - $s_{2,2} = 2, f_{2,2} = 6$

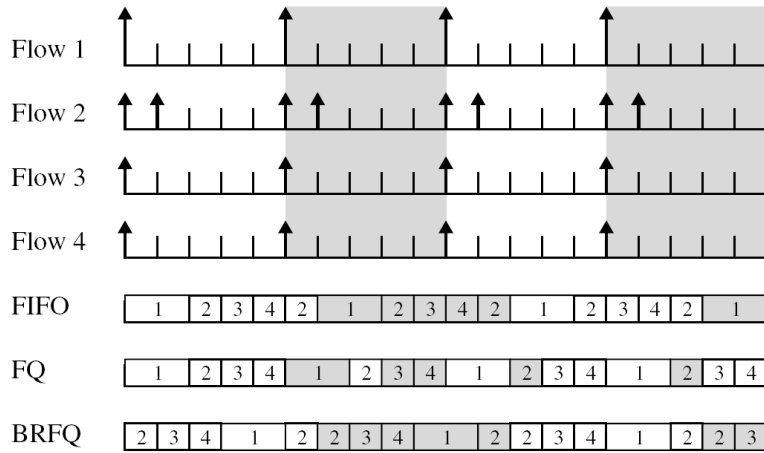


Comparison with FIFO and FQ



- Load = capacity
- PS gives preference to short packets → average delay of flows 2 and 3 smaller than FIFO

Comparison with FIFO and FQ



- Load > capacity → queue builds up
- FQ: flow 2 is penalized, although has same load as flow 1
- PS gives preference to short packets and it's fair

Generalized Processor Sharing

- PS and its packet-level equivalent provide Max-Min fairness but do not differentiate resource allocation
- Generalized Processor Sharing (GPS) is capable of differentiation by associating a weight to each flow
- Each nonempty queue is serviced for an infinitesimal amount of data in proportion to the associated weight
 - server capacity is split among nonempty queues according to weights
- GPS provides an exact Max-Min weighted fair share allocation

GPS: Formal Definition

- GPS is a work conserving scheduler operating at fixed capacity C over K flows

- Each flow is characterized by a positive real weight

$$\phi_1, \phi_2, \dots, \phi_K$$

- $S_i(\tau, t)$ is the amount of traffic from flow i serviced in the interval $[\tau, t]$

- GPS server is defined as a server for which

$$\frac{S_i(\tau, t)}{S_j(\tau, t)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, 2, \dots, K$$

for any flow i that is backlogged (i.e. has nonempty queue) throughout $[\tau, t]$

GPS: Flow Rate

- Summing over all flows

$$S_i(\tau, t) \sum_{j=1}^K \phi_j \geq \phi_i \sum_{j=1}^K S_j(\tau, t) = \phi_i C (t - \tau)$$

- Each backlogged flow receives a service rate of

$$r_i(t) = \lim_{\tau \rightarrow t} \frac{S_i(\tau, t)}{t - \tau} \geq \frac{\phi_i}{\sum_{j=1}^K \phi_j} C = g_i$$

- If a source rate is $\rho_i \leq g_i$ then such rate is always guaranteed independently of the other flows
- $0 \leq N(t) \leq K$ number of backlogged queues at time t
- The service rate seen by flow i when $N(t) > 0$ is

$$r_i(t) = \frac{\phi_i}{\sum_j \phi_j} C$$

with the sum taken over all backlogged queues

GPS: Virtual Time

- Now virtual time definition must consider weights
 - only backlogged queues contribute to the virtual time computation
 - the sum of the weights of backlogged queues may change at arrival and departure times only

- $v(t)$ continuous piecewise linear function of t with slope

$$\frac{1}{\sum_j \phi_j}$$

- Actual arrival time of packet n of flow i is $t_{i,n}$

- Its length is $L_{i,n}$

- Virtual starting and ending transmission times are

$$f_{i,n} = s_{i,n} + \frac{L_{i,n}/C}{\phi_i}$$

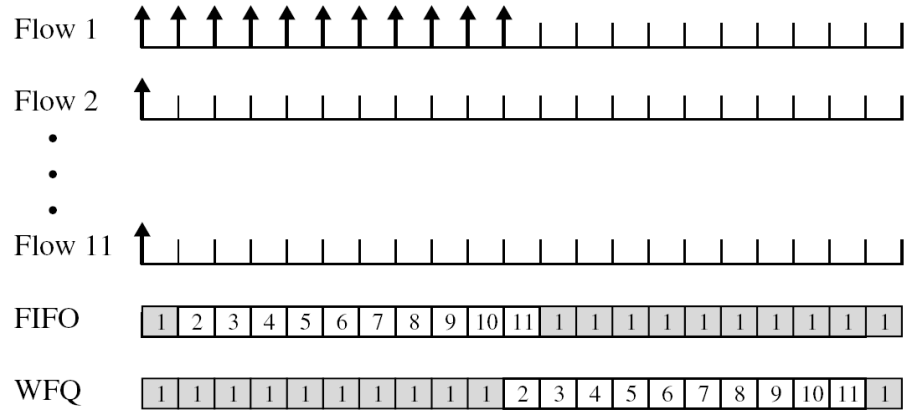
$$s_{i,n} = \max [f_{i,n-1}, v(t_{i,n})]$$

GPS Emulation: Weighted Fair Queuing

- It is a packet-based GPS
 - compute virtual starting and ending times as in the ideal GPS
 - when a packet finishes its service, next packet to be serviced is the one with smallest virtual finishing time
- Increased computational complexity since scheduler
 - must keep track of the backlogged flows and their weights
 - must compute virtual finishing times and sort them out
 - must update virtual time at arrival and departure events

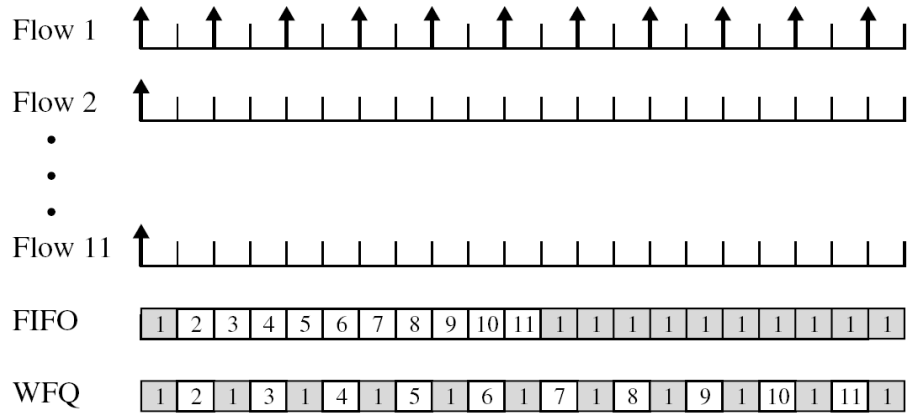
Comparison with FIFO

$$\phi_1 = 10 \phi_j, \quad j = 2, 3, \dots, 11$$



Comparison with FIFO

Flow 1 now sends packets at the guaranteed rate = 0.5



Weighted Fair Queuing: Delay Bound

- WFQ approximates GPS at packet level
 - Max-Min weighted fair share achieved
 - weights allow service differentiation
- WFQ also keeps queuing delay bounded for well-behaved flows
- Assume each flow shaped with a token bucket scheme, (ρ_i, σ_i) token bucket parameters for flow i
- Choose weights proportional to token rates
$$\phi_i = \rho_i / C$$
- Assume all buckets full of tokens when transmission starts from all flows (worst case)
 - each flow sends a burst σ_i and then keeps transmitting at ρ_i

Weighted Fair Queuing: Delay Bound

- In case of GPS, each queue i is filled up to σ_i and is both serviced and fed at rate ρ_i
- Since queue size cannot exceed σ_i , the maximum delay experienced by packets of flow i is

$$d_i \leq \frac{\sigma_i}{\rho_i}$$

- In case of WFQ, the effect of servicing packets increases the delay, but this is still bounded
- End-to-end queuing delay of flow i through H_i WFQ hops is actually bounded

$$d_i \leq \frac{\sigma_i}{\rho_i} + \frac{(H_i - 1)L_i}{\rho_i} + \sum_{m=1}^{H_i} \frac{L_{\max}}{C_m}$$

Weighted Fair Queuing: Delay Bound

- L_i maximum packet size of flow i
- $L_{\max} = \max L_i$
- C_m server capacity at node $m = 1, 2, \dots, H_i$

$\frac{\sigma_i}{\rho_i}$ maximum delay at the first node due to burst transmission (same as GPS)

$\frac{(H_i - 1)L_i}{\rho_i}$ maximum delay when a packet arrives at each node just after its supposed starting time and must wait for preceding packet

$\sum_{m=1}^{H_i} \frac{L_{\max}}{C_m}$ maximum delay due to packet-level service: each arriving packet must wait for the one currently in service

GPS Emulation: Weighted Round Robin

- Each flows is characterized by its
 - weight
 - average packet size
- The server divides each flow weight by its mean packet size to a obtain a normalized set of weights
- Normalized weights are used to determine the number of packets serviced from each connection in each round

- Simpler scheduler than WFQ
- Mean packet size may not be known a-priori

GPS Emulation: Deficit Round Robin

- DRR modifies WRR to handle variable packet sizes without knowing the mean packet size of each flow in advance
 - DRR associates each flow with a “deficit counter”
 - in each turn, the server attempts to serve one “quantum” worth of bits from each visited flow
- A packet is serviced if the sum of the deficit counter and the quantum is larger than the packet size
 - if the packet receives service, its deficit counter is reduced by the packet size
 - if the packet does not receive service, a quantum is added to its deficit counter
- During a visit, if connection is idle, its deficit counter is reset to 0
 - prevents a connection from cumulating service while idle

Scheduling and admission control

- Given a set of currently admitted connections and a descriptor for a new connection request, the network must verify whether it is able to meet performance requirements of new connection
- It is also desirable that the admission policy does not lead to network underutilization
- A scheduler define its “schedulable region” as the set of all possible combinations of performance bounds that it can simultaneously meet
 - resources are finite, so a server can only provide performance bounds to a finite number of connections
- New connection requests are matched with the schedulable region of relevant nodes

Scheduling and admission control

- Schedulable region is a technique for efficient admission control and a way to measure the efficiency of a scheduling discipline
 - given a schedulable region, admission control consists of checking whether the resulting combination of connection parameters lies within the scheduling region or not
 - the larger the scheduling region, the more efficient the scheduling policy

