eecs589

ADVANCED COMPUTER NETWORKS

[DKS89] Demers, Keshav, and Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. of ACM SIGCOMM '89*, 19(4):1-12, Sep. 1989

Queueing and Scheduling

When would you see a queue?

- when resource is limited
- and there is a contention for the resource
- in networking context: when packet incoming rate is faster than outgoing (service) rate

Want to be fair in service and to protect against misbehaving connections

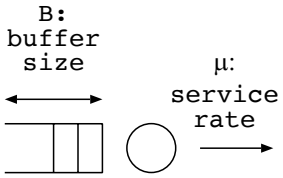
Scheduling discipline: which packet to serve next?

Queue Management

Queue management design issues:

- Fairness
  - at the minimum want protection against malicious sources
- Delay bound
- Drop policy
- Cost of operation

FIFO/FCFS Scheduler:



A resource centric view: max-min fair share, assuming users have equal rights to resource

Max-Min Fair Share

Let:

μ_{total}: total resource (e.g., bandwidth) available

μ_i: total resource given to (flow) i

μ_{fair}: fair share of resource

ρ_i: request for resource by (flow) i

Max-Min fair share is $\mu_i = \text{MIN}(\rho_i, \mu_{fair})$

$$\mu_{total} = \sum_{i=1 \text{ to } n} \mu_i$$

Max-Min Fair Share

In words: max-min fair share maximizes minimum shares of flows whose demands have not been fully satisfied

1. no flow gets more than its request
2. no other allocation satisfying condition 1 has a higher minimum allocation
3. condition 2 remains recursively true as we remove the minimal request and reduce total resource accordingly

Max-Min Fair Share Example

Let: $\mu_{total} = 30$

i	ρ_i	μ_i
A	12	11
B	11	11
C	8	8

Initialy $\mu_{fair} = 10$

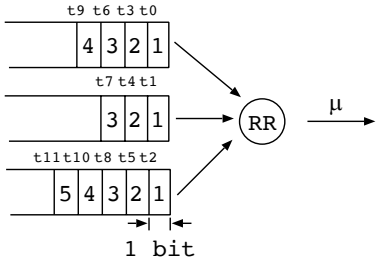
$\rho_C = 8$, so unused resource $(10 - 8 = 2)$ is divided evenly between the remaining i's whose demands are not yet fully met

Thus, $\mu_{fair} = 10 + 2/2 = 11$

Bit-by-Bit Round Robin

1 round, $R(\ )$, is defined as all non-empty queues have been served 1 quantum

- $R(t_5) = 2$
- time at Round 3? Round 4?



BbB-RR achieves max-min fair share

Max-min fair-share isolates flows

BbB-RR protects against misbehaving flows

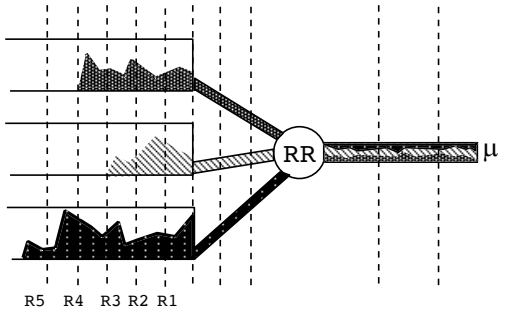
A.k.a. Generalized Processor Sharing (GPS)

Fluid-Flow Approximation

A continuous service model

- instead of thinking of each quantum as serving discrete bits in a given order
- think of each connection as a stream of fluid, described by the speed and volume of flow

At each quantum the same amount of fluid from each (non-empty) stream flows out concurrently

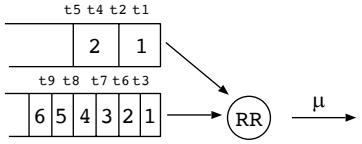


Packetized Scheduling

Packet-by-packet round robin:

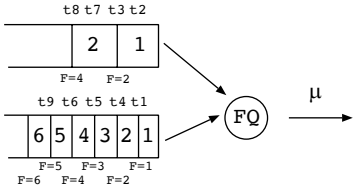
Problem:

Gives bigger share to flows with big packets

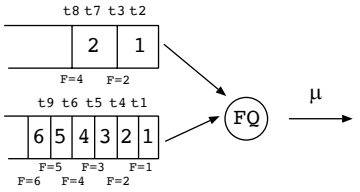


Packet-by-packet fair-queueing:

- F: finish round, the round a packet finishes service
- simulates RR in the computation of F's
- serve packets with the smallest F first



Start and Finish Rounds



At what round does packet i of flow α start seeing service?

$S^{\alpha}_i = \mathrm{MAX}(F^{\alpha}_{i-1}, A^{\alpha}_i)$

- $A^{\alpha}_i = R(t^{\alpha}_i)$: round at the time packet i arrives
- $S^{\alpha}_i = F^{\alpha}_{i-1}$ if there is a queue, A^{α}_i otherwise

When does packet i finish service?

$F^{\alpha}_i = S^{\alpha}_i + P^{\alpha}_i$,

where P^{α}_i is the size (service time) of packet i

Round Computation

Recall: 1 round is defined as all active flows have been served 1 quantum

Round's rate of change: $\partial R/\partial t = \mu/N_{ac}(t)$, where

- μ: link bandwidth
- $N_{ac}(t)$: number of active flows at time t

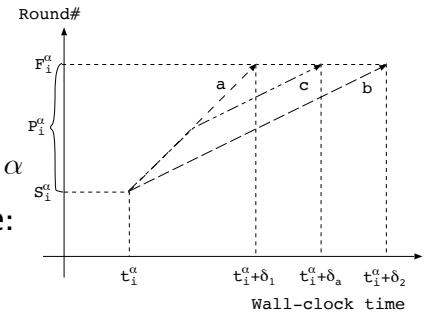
Flow α is active at time t if $R(t) \le F^{\alpha}_i$, where i is the last bit of flow α in queue

The speed of 1 round through all active flows depends on the number of active flows: faster if there are less active flows

Round# vs. Wall-Clock Time

Let:

- time: wall-clock time
- round: virtual-clock time
- $\mu = 1$ unit
- t^{α}_i: arrival time of packet i of flow α



Computing the rate of change:

a: $N_{ac} = 1$, $\partial R/\partial t = 1$,

b: $N_{ac} = 2$, $\partial R/\partial t = ½$, $\delta_2 = 2*\delta_1$

c: at the beginning, $N_{ac} = 1$, $\partial R/\partial t = 1$, halfway serving packet i, a packet belonging to another flow arrives, $N_{ac} = 2$, $\partial R/\partial t = ½$

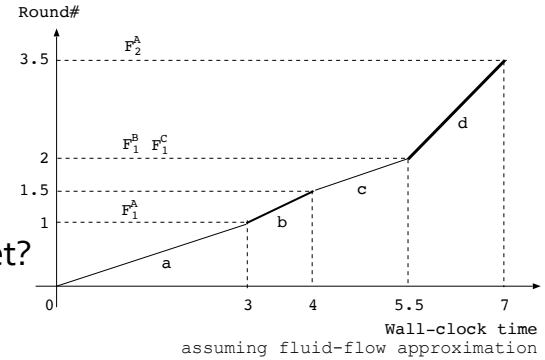
As $N_{ac}(t)$ changes, finish round stays the same, actual time stretches

Round Computation Example

Scenario:
- flows A has 1 packet of size 1 arriving at time 0
- flows B and C each has 1 packet of size 2 arriving at time 0
- flow A has another packet of size 2 arriving at time 4

Slope ($\partial R/\partial t$):
$a = ⅓, b = ½,$
$c = ⅓, d = 1$

What is the arrival round of A's 2nd packet?
$R(t^{A}_{2}) = 1.5$



Arrival Round Computation

When packet i of an active flow arrives, its finish time is computed as $F^{\alpha}_{i} = F^{\alpha}_{i-1} + P^{\alpha}_{i}$, where F^{α}_{i-1} is the finish time of the last packet in α's queue

If flow α is inactive, there's no packet in its queue, $F^{\alpha}_{i} = A^{\alpha}_{i} + P^{\alpha}_{i}$, how do we compute A^{α}_{i}?

If flow α has been inactive for Δt time and there has been N_{ac} flows during the whole time, we can compute $A^{\alpha}_{i} = F^{\alpha}_{i-1} + \Delta t(1/N_{ac}) \Rightarrow$ inactive flow must store F^{α}_{i-1}

But what if N_{ac} has changed, several times, over Δt?

Iterated Deletion

Variables:
- t_l: the time when the round was last computed
- $R(t_l)$: the round at that time
- global array of finish rounds of inactive flows, sorted

Want to compute the current Round by taking into account each of the finish rounds in the array

Finish round rate at time t: $\partial F^{\alpha}_{i}/\partial t = \mu/N_{ac}(t)$

We know $N_{ac}(t)$ hasn't changed since t_l:
- Finish round, given finish time: $F^{\alpha}_{i} = R(t_l) + \mu\,(t(F^{\alpha}_{i}) - t_l)$
- Finish time, given finish round: $t(F^{\alpha}_{i}) = t_l + [F^{\alpha}_{i} - R(t_l)]N_{ac}/\mu$

Now walk the global array and compute "round catchup" to the present time

Iterated Deletion: the Algorithm

Let t^{C}_{i} be the arrival time of packet i of flow C
For each flow α with packets enqueued or
has last finish round in the global array {
  compute $t(F^{\alpha}_{l})$ // finish time of flow α
  if $(t(F^{\alpha}_{l}) \le t^{C}_{i})$ { // if flow is inactive by time t^{C}_{i}
    $R(t(F^{\alpha}_{l})) = F^{\alpha}_{l}$ // finish round when flow went inactive
    $t_l = t(F^{\alpha}_{l})$ // the last round computation time
    N_{ac}--
  } else { break }
}

Time complexity: $O(N_{ac})$

$R(t^{C}_{i}) = R(t_l) + \mu/N_{ac}(t^{C}_{i} - t_l)$, // A^{α}_{i}
assuming $N_{ac} \neq 0$, else may reset R
$t_l = t^{C}_{i}$

Variation: Credit Accumulation

Allow a flow to have a bigger share if it has been idle

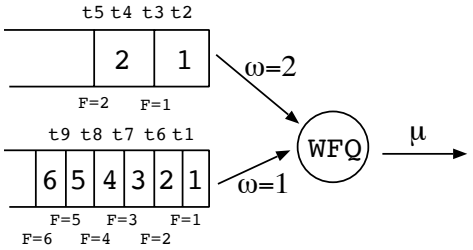
Compute bid per packet:

$B^{\alpha}_{i} = \mathrm{MAX}(B^{\alpha}_{i-1}, A^{\alpha}_{i} - \delta) + P^{\alpha}_{i}$

- if $\delta = 0$, no credit accumulated, $B^{\alpha}_{i} = F^{\alpha}_{i}$
- if $\delta = \infty$, $B^{\alpha}_{i} = F^{\alpha}_{i-1} + P^{\alpha}_{i}$, regardless of packet i's arrival time

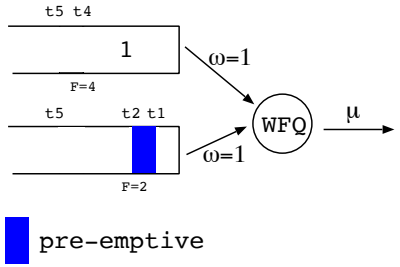
Credit accumulation is discouraged because it can be abused: accumulate credits for a long time, then send a big burst of data

Weighted Fair Queueing



$F^{\alpha}_{i-1} = S^{\alpha}_{i} + P^{\alpha}_{i}/\omega^{\alpha}$,
where ω^{α} is the weight (reserved rate) of flow α

Pre-emptive WFQ



Non-preemptive worst-case: packet arrives as another packet of larger F just started service

Packet has to wait P_{max} round before seeing service

Limitations of WFQ

Round computation expensive: must re-compute R every time $N_{ac}(t)$ changes

Packetization causes two kinds of unfairness:

- shift from fluid-flow round-robin: fairness "quantized" by minimum packet size (Absolute Fairness Bound)
- unfairness to some flows: if flows have different packet sizes (Relative Fairness Bound)