Fair Queueing

Design space

- Buffer management:
  - RED, Drop-Tail, etc.
- Scheduling: which flow to service at a given time
  - FIFO
  - Fair Queueing

Scheduling

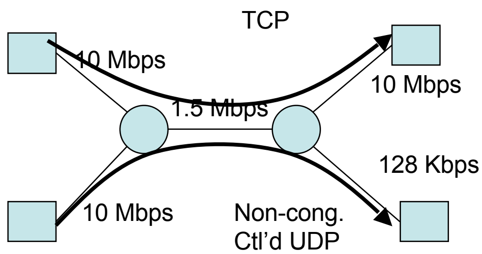
- Work-conserving:
  - Link is never idle if there are packets to send
  - Examples: FIFO, Fair Queueing
- Non-work conserving
  - …
  - Examples: TDMA

Fairness Goals

- Allocate resources fairly
- Isolate ill-behaved users
  - Router does not send explicit feedback to source
  - Still needs e2e congestion control
- Still achieve statistical muxing
  - One flow can fill entire pipe if no contenders
  - Work conserving → scheduler never idles link if it has a packet

A Caveat: Still need e2e

- Congestion collapse can still happen if you have fair queueing (router-assisted sharing)



Example from Floyd and Fall, 1999

What does "fairness" divide between?

- At what granularity?
  - Flows, connections, domains?
- What if users have different RTTs/links/etc.
  - Should it share a link fairly or be TCP fair?
- Basically a tough question to answer – typically design mechanisms instead of policy
  - User = arbitrary granularity
  - Paper has a nice argument for (src, dst) pairs

Max-min Fairness (reminder)

- Allocate user with "small" demand what it wants, evenly divide unused resources to "big" users
- Formally:
  - Resources allocated in terms of increasing demand
  - No source gets resource share larger than its demand
  - Sources with unsatisfied demands get equal share of resource

Max-min Fairness Example (reminder)

- Assume sources 1..n, with resource demands X1..Xn in ascending order
- Assume channel capacity C.
  - Give C/n to X1; if this is more than X1 wants, divide excess (C/n - X1) to other sources: each gets C/n + (C/n - X1)/(n-1)
  - If this is larger than what X2 wants, repeat process

Implementing Max-min Fairness

- Important point:
  - Converge to some α, s.t.
    - Flows with offered load ri < α get ri
    - Flows with load > α get α
  - Σ i=1 to n min(ri, α) = C (capacity)
- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues
- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - What if arrive just before/after packet departs?

Bit-by-bit RR

- Multiple flows: clock ticks when a bit from *all* active flows is transmitted → a "round"
  - μ = #bits/sec router can send, N = # active flows
  - dR/dt (the rate at which the round #increases) is *variable* = μ / N
  - Why count this way? # of rounds to send a packet is *independent* of number of active flows. Useful way of viewing things…

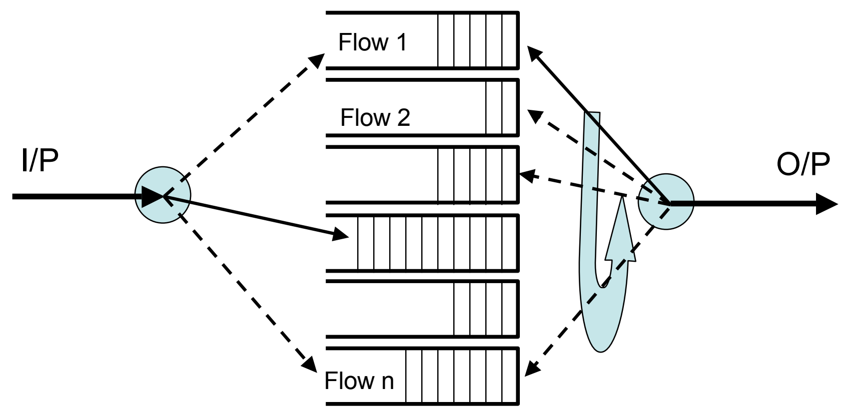
Bit-by-bit round robin

- Packet arrives in queue Q:
  - It's the *i*th packet in the queue
  - It's p_i^q bits long
  - When does it start being transmitted?
    - If q empty, immediately: R(t)
    - Else, just after prior pkt finishes: F_{i-1}^q
    - $S_i^q = \max(R(t), F_{i-1}^q)$
  - When does it complete?
    - $S_i^q + p_i^q$ (p_i^q rounds later…)
  - Can compute the finish *round* of every packet in the queue. (Even at the point when the packet is enqueued). Note that we don't know the actual finish *time*, just the round #.

Packet-based Fair Queueing

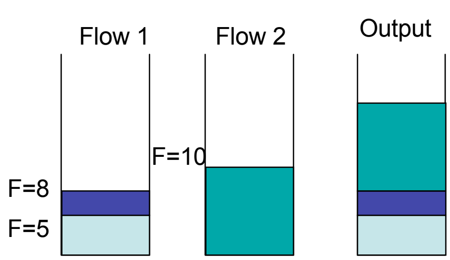
- Simple: Send the packet with the smallest finishing round #.
- Approximates bit-by-bit RR
  - Why isn't it exact? Preemption!

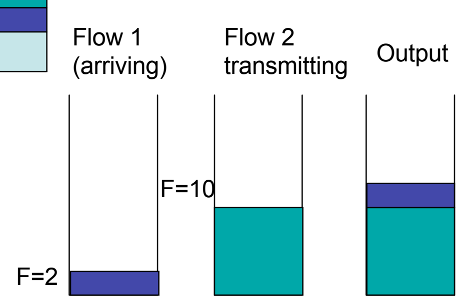
FQ Illustration



Variation: Weighted Fair Queuing (WFQ)

Bit-by-bit RR Example





Cannot preempt packet currently being transmitted

Delay Allocation

- Reduce delay for flows using less than fair share
  - Advance finish times for sources whose queues drain temporarily
- Schedule based on B_i instead of F_i
  - $F_i = P_i + \max(F_{i-1}, A_i) \rightarrow B_i = P_i + \max(F_{i-1}, A_i - \delta)$
  - If $A_i < F_{i-1}$, conversation is active and δ has no effect
  - If $A_i > F_{i-1}$, conversation is inactive and δ determines how much history to take into account
    - Infrequent senders do better when history is used

Fair Queuing Tradeoffs

- FQ can control congestion by monitoring flows
  - Non-adaptive flows can still be a problem – why?
- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. do fairness per domain)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times
  - dR/dt changes whenever the flow count changes

Core-Stateless Fair Queuing

- Key problem with FQ is core routers
  - Must maintain state for many (50-100k!) flows
  - Must update state at Gbps line speeds
- CSFQ (Core-Stateless FQ) objectives
  - Edge routers should do complex tasks since they have fewer flows (1000s)
  - Core routers can do simple tasks
    - No per-flow state/processing → this means that core routers can only decide on dropping packets not on order of processing
    - Can only provide max-min bandwidth fairness not delay allocation

Core-Stateless Fair Queuing

- Edge routers keep state about flows and do computation when packet arrives
- DPS (Dynamic Packet State)
  - Edge routers label packets with the result of state lookup and computation
  - Note: Generalizes beyond CSFQ!
- Core routers use DPS and local measurements to control processing of packets

Key ideas

- DPS: Edges estimate arrival rate for each flow (per-flow state)
- Core routers use
  - Estimated arrival rates from edge
  - Internal measure of fair-share
  - To generate a drop probability. Labels changed on outbound flow with new post-drop arrival rate.
- Estimation for fair-share value converges rapidly

Edge Router Behavior

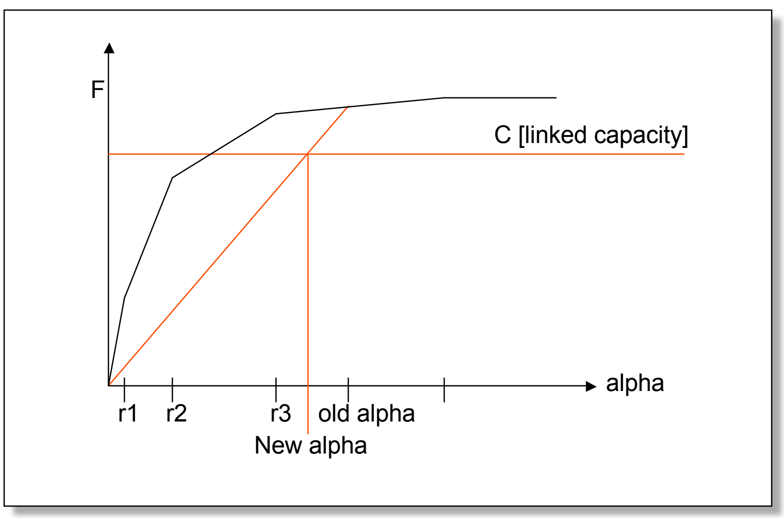
- Monitor each flow i to measure its arrival rate (r_i)
  - EWMA of rate
    - t_i^k, l_i^k = arrival time, length of kth packet in flow i
  - Non-constant EWMA constant
    - T_i^k = interarrival (time since last pkt) (t_i^k – t_{i-1}^k)
    - Constant: $e^{-T/K}$ where T , K = constant
    - Ri new = (1 – const)* length/interarrival + const*(ri old)
    - Helps adapt to different packet sizes and arrival patterns
      - Intuition: Trusts the “old” values less as the time interval increases (*negative* T)
- Rate is attached to each packet

Core Router Behavior

- Drop probability for packet = max(1- α/r, 0)

- Track aggregate input A
- Track accepted rate F(α)
- Estimate fair share rate α
  – Solve F(α) = C; but this is hard:
  – Note: Increasing α does not increase load (F) by N * $\Delta\alpha$ (why?)
  – $F(\alpha) = \Sigma_i \min(r_i, \alpha)$ → what does this look like?

F vs. Alpha



Estimating Fair Share

- Need $F(\alpha)$ = capacity = C
  - Can't keep map of $F(\alpha)$ values → would require per flow state
  - If we're overutilized:
  - Since $F(\alpha)$ is concave, piecewise-linear
    - $F(0) = 0$ and $F(\alpha)$ = current accepted rate = F_c
    - $F(\alpha) = F_c / \alpha$
    - $F(\alpha_{new}) = C \rightarrow \alpha_{new} = \alpha_{old} * C/F_c$
  - If underutilized:
    - α = max_i (ri) (No drops at all)
- What if a mistake was made?
  - Forced into dropping packets due to buffer capacity
  - When queue overflows α is decreased slightly
  - Note that this is an increase/decrease rule in disguise. ☺

Other Issues

- Punishing fire-hoses – why?
  - Easy to keep track of in a FQ scheme
- What are the real edges in such a scheme?
  - Must trust edges to mark traffic accurately
  - Could do some statistical sampling to see if edge was marking accurately