III. SCHEDULING

(a) General Concepts

- servicing of queues in network requires scheduling discipline to determine order that packets serviced

  eg. queues on network links, queues for servers

  ⇒ scheduler responsible for managing delays of packets and discarding of packets when queue full

- most obvious approach: first-come, first-serve (FCFS)

- in general, packets associated with different connections or classes of service can have different priorities, delays, and loss requirements

- conceptually:

- 2 general service categories:

  (1) guaranteed service

  → network resources reserved to achieve performance bounds

  ⇒ "quality of service" (QoS)

  (2) best effort

  → no reservation of network resources required

  ⇒ fairness in resource allocation critical

Typical QoS Parameters

Bandwidth:	- minimum specified
Delay:	- could specify worst case upper bound or average case upper bound
Loss:	- upper bound on fraction of packets lost
Delay Jitter:	- upper bound on difference between largest and smallest delays

- delay jitter critical in audio and video playback applications where constant information required

- jitter can be removed at receiver by using "elasticity" buffer but larger jitter $\Rightarrow$ large buffers

  $\therefore$ desirable to minimize jitter in network

(b) Priority Scheduling

- obvious way to give connections levels of service

- approach: logically put packets into a queue based on priority level

- discipline: serve packet from queue *n* unless empty, then serve packet from queue *n*-1 unless empty, then serve packet from queue *n*-2, etc.

- problem: low priority queues can be starved if higher priorities take all server's time

- very easy to implement in both hardware and software but cannot guarantee quality of service

(c) Work-Conserving vs. Non-Work-Conserving Scheduling

- scheduler is "work-conserving" if only idle when queues empty

  ⇒ decreasing delay for queue *i* by giving more service to queue *i* will increase delay for other queues

  ⇒ scheduling discipline can only trade-off delay between queues

- scheduler is "non-work-conserving" if server may be idle when queues not empty

  ⇒ NWC scheduling can be utilized to minimize delay jitter and to reduce buffers required for small packet loss by smoothing out traffic

(d) Max-Min Fair Sharing

- one technique to share resource that satisfies users with small demands and distributes remaining resource evenly to large demand users

- assume resource with capacity C and n users with demands $d_1, d_2, \ldots, d_n$ (in ascending order)

- procedure:

  → resources allocated in order of increasing demand so user gets a share of no more than $1/k$ of remaining capacity of resource if k users left

  → no user gets a share greater than its demand

  → users with unsatisfied demand get an equal share

eg. - user 1 gets up to C/n of resource

- if $d_1 > C/n$, then user 1 gets C/n of resource and user 2 gets up to C/n of resource

- if $d_1 < C/n$, then user 1 gets d_1 of resource and user 2 gets up to $(C-d_1)/(n-1)$ of resource

etc.

- no user gets more than demanded or, if demand not met, no less than any source with higher demand

- "max-min fair" $\rightarrow$ maximizes minimum share of a user whose demand not satisfied

- can give users weights $w_1, w_2, \ldots, w_n$ to reflect relative share (i.e., reflects priority of resource use)

- now limit on allocated share is in proportion to weight

Example:

(e) Scheduling Best-Effort Connections

- fairness critical, i.e., max-min fair sharing desirable

Generalized Processor Sharing (GPS)

- idealized mechanism that achieves max-min fair sharing

- packets go into a logical queue associated with their connection

- each queue visited in round robin fashion and served for a very small increment of time Δt if queue not empty

- as $\Delta t \rightarrow 0$, achieves max-min fair share

- can weight queue i with w_i and serve for time $w_i \times \Delta t$ in each rotation

- GPS unimplementable!
  $\rightarrow$ How can we serve a portion of a packet in a time $\Delta t \rightarrow 0$?

Example:

Weighted Round Robin

- similar to GPS except $\Delta t \neq 0$ but serve an entire packet in a rotation

- connections (i.e., queues) can have weights

- emulates GPS well for small, fixed size packets (eg. ATM)
  $\Rightarrow$ over long enough periods of time very fair

What if packet size not constant?
  $\rightarrow$ normalize connection weights by dividing by mean packet size
  $\rightarrow$ difficult to know mean packet size!

Weighted Fair Queuing

- basic concept:
  $\rightarrow$ compute "times" to finish serving packets with GPS server and then serve packets in order of finishing "times" (actually "finishing numbers")

- let finish number = number used to reflect time at which packet finished (but not really finish time)

- consider a GPS server where Δt = 1 bit time

- one round = serving one bit from all "active" connections (i.e., one round not constant time)

- an active connection has largest finish number of packet waiting in queue or currently being served > current round number

$\rightarrow$ time for one round = # active connections $\times$ Δt

FN calculation:

- packet arriving at inactive connection (i.e., empty queue):

FN = current round number + # bits in packet

- packet arriving at active connection (i.e., non-empty queue):

FN = largest FN of packet in queue + # bits in packets

- FN does not depend on future arrivals, i.e., once computed does not change

- round number does not increase at a constant rate with respect to time but at a rate inversely proportional to # active connections

RN = $t \times$ link rate / # active connections + constant
(bits)

$\rightarrow$ RN vs. time = piece-wise linear graph

- when RN = FN indicates packet "done" according to simulated GPS servicing

- in real scheduling, packets are served in order of FN

- do not need to view RN as # rounds for bit-by-bit round robin server

  $\rightarrow$ can view as real-valued variable proportional to # active connections, i.e., RN just an abstraction

- complexity not in determining FN given RN but in determining RN

See WFQ example.

- buffer drop policy: when packet comes into queue, if necessary, packets with largest finishing numbers can be dropped to make room for packets

- for "weighted" fair queuing:
  → assume w_i = weight of i-th connection

- WFQ implemented in routers and ATM switches

(f) Scheduling Guaranteed-Service Connections

- can use WFQ to provide bandwidth and worst-case delay bounds

eg. bandwidth allocated to connection i on a link can be determined by

$$\frac{w_i}{\sum_j w_j} \times link\,rate$$

- delay jitter: WFQ not directly useful but one non-work-conserving approach:

- can hold packets in regulator queue for connection *i* so that packets do not arrive at scheduler any faster than a specified rate

- evens out delays during packet bursts so that total end-to-end delay more consistent