CS 5224

Scheduling and Buffer Management

Dr. Chan Mun Choon
School of Computing, National University of Singapore

Acknowledgement/Reference

- Slides are taken from the following source:
  - S. Keshav, "An Engineering Approach to Computer Networking", Chapter 9: Scheduling
- Reading materials on course web-site
- References
  - L. Kleinrock, "Queuing Systems," Volume II, Chapter 3 and 4, 1975.

Outline

- What is scheduling, why we need it?
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling disciplines
- Buffer management and packet drop strategies

Scheduling

- Sharing always results in contention
- A *scheduling discipline* resolves contention:
  - who's next?
- Key is to share resources fairly and provide some form of performance guarantees

Components

- A scheduling discipline does two things:
  - decides service order (scheduling)
  - manages queue of service requests (buffer management)
- Example:
  - consider queries awaiting web server
  - scheduling discipline decides service order
  - and also if some query should be ignored

Where?

- Anywhere where contention may occur
- At every layer of protocol stack
- Usually studied at network layer, at output queues of switches

Why do we need one?

- *Because applications need it*
- We expect at least two types of future applications
  - best-effort (adaptive, non-real time)
    - e.g. email, some types of file transfer
  - guaranteed service (non-adaptive, real time)
    - e.g. packet voice, interactive video, stock quotes

What can scheduling disciplines do?

- Give different users different qualities of service
- Example of passengers waiting to board a plane
  - early boarders spend less time waiting
  - bumped off passengers are 'lost'!
- Scheduling disciplines can allocate
  - bandwidth
  - delay
  - loss
- They also determine how *fair* the network is

Outline

- What is scheduling, why we need it?
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling disciplines
- Buffer management and packet drop strategies

Requirements

- An ideal scheduling discipline
  - is easy to implement
  - is fair (what is fair?)
  - provides performance bounds
  - allows easy *admission control* decisions
    - to decide whether a new flow can be allowed

Ease of implementation

- Scheduling discipline has to make a decision once every few microseconds!
- Should be implementable in a few instructions or hardware
  - for hardware: critical constraint is VLSI *space*
- Work per packet should scale less than linearly with number of active connections

Fairness

- Scheduling discipline *allocates* a *resource*
- An allocation is fair if it satisfies *some notion of fairness*
- Intuitively
  - each connection gets what it "deserves"

Fairness (contd.)

- Fairness is *intuitively* a good idea
- But it also provides *protection*
  - traffic hogs cannot overrun others
  - automatically builds *firewalls* around heavy users
- Fairness is a *global* objective, but scheduling is local
- Each endpoint must restrict its flow to the smallest fair allocation

Notion of Fairness

- What is "fair" in resource sharing?
  - Everybody gets what they need?
  - How about excess resources?
- Example:
  - A "flat" tax system whereby everybody pays the same tax rate.
  - A "progressive" tax system whereby people who has larger income pay at a higher tax rate.
- Factors to consider
  - How does fairness relate to ability to use resource?
  - How does fairness affects overall resource utilization?

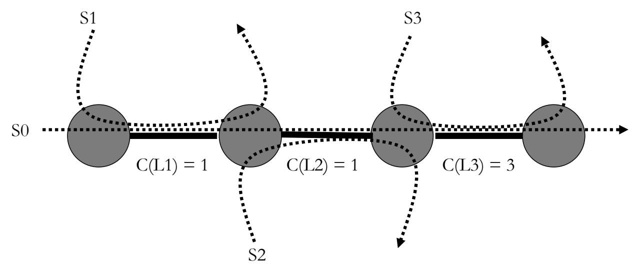
Fairness

- Equal Share
  - Resources are shared among all users independent of user requirements and resource utilization
  - Is it a good model for resource sharing?

Max-Min Fairness

- Maximizes the minimum share of a resource whose demand is not fully satisfied
- Intuitively:
  - each connection gets no more than what it wants
  - the excess, if any, is equally shared
- Start with max-min fairness for flow control
  - [BG, chapter 6: Flow Control]

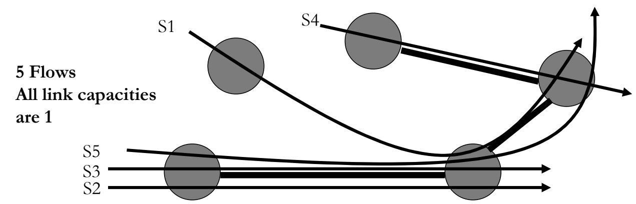
Max-Min Flow Control



How much rate should be allocated to S0, S1, S2 and S3?
Two possibilities:
{0.5,0.5,0.5,0.5} but L3 is under-utilized
{0.5,0.5,0.5,2.5} S3 gets more bw with no impact on others

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}

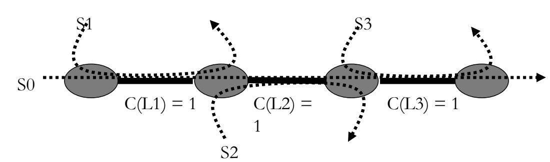
Max-Min Allocation

- Apply max-min allocation to a single resource
- Interesting case is when demand is greater than capacity
- Given users with demands {2,2.6,4,5} and capacity 10. Total demand = 13.5.

1. {2.5, 2.5, 2.5, 2.5} {0.5,-0.1,-1.5,-2.5} excess=0.5
2. {2,2.66,2.66,2.66} {0,0.06,1.34,2.34} excess=0.06
3. {2,2.6,2.7,2.7} {0,0,1.3,2.3}

Proportional Fair (PF)

- Maximize sum of utility (a function of the allocated rate), a reasonable utility function is log()
- A PF allocation x_i satisfies $\Sigma(y_i - x_i)/x_i <= 0$ for any feasible allocation y
- The allocation below would be
  - Max-Total: {0,1,1,1}. Total = 3. Utility = ?
  - Max-Min Fair: {0.5,0.5,0.5,0.5}. Total = 2. Utility = ?
  - Proportional Fair: {0.25,0.75,0.75,0.75}. Total = 2.5. Utility = ?



Outline

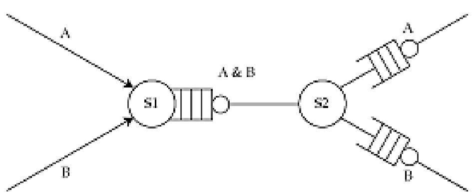
- What is scheduling, why we need it?
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling disciplines
- Buffer management and packet drop strategies

Fundamental choices

1. Work-conserving vs. non-work-conserving
2. Degree of aggregation

Work conserving or not?

- Work conserving: server is never idle when there is packets awaiting service
  - Maximizes utilization of server resource
- Why bother with non-work conserving?



Non-work-conserving disciplines

- Key conceptual idea: delay packet till *eligible*
- Reduces delay-jitter => fewer buffers in network
- How to choose eligibility time?
  - rate-jitter regulator
    - bounds maximum outgoing rate
  - delay-jitter regulator
    - compensates for variable delay at previous hop

Do we need non-work-conservation?

- Can remove delay-jitter at an endpoint instead
  - but also reduces size of switch buffers…
- Increases mean delay
  - not a problem for *playback* applications
- Wastes bandwidth
  - can serve best-effort packets instead
- Always punishes a misbehaving source
  - can't have it both ways
- Bottom line: not too bad, implementation cost may be the biggest problem

Degree of aggregation

- More aggregation
  - less state: less memory and computation
  - cheaper: smaller VLSI, less to advertise
  - cost: 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?

Outline

- What is scheduling, why we need it?
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling disciplines
- Buffer management and packet drop strategies

First In First Out (FIFO)

- Most common scheduling
  - Schedule packets according to the time of arrival
- Disadvantages
  - Cannot differentiate between packets
- Advantages
  - Easy to implement
- Question: How does a complex scheduler improves the performance?

The Conservation Law

- If the scheduler is work conserving, and the scheduling is **independent of the packet service time**
  - $\Sigma \rho_i q_i$ = constant
  - where ρ_i = mean utilization of connection i and q_i = mean waiting time of connection I
- Therefore, if by using a different scheduling discipline, a particular connection receives a lower delay than with FCFS, at least one other connection must have a higher delay.
- The average delay with FCFS is a tight lower bound for work conserving and service time independent scheduling disciplines

Service-Time Dependent Scheduling

- D(.) be the average waiting time
- FCFS: First Come First Serve
- SPT: shortest processing time first
- SRPT: shortest remaining processing time first
- **D(FCFS) >= D(SPT) >= D(SRPT)***
- **However, s**ervice-time dependent scheduling are not common in packet switching because the packet ordering will be modified and delay for large packets increases
- *Reference: [KLE76]

General Process Sharing (GPS)

- A scheduler should be easy to implement, fair, provides performance bounds, and allows easy admission control decisions
- GPS achieves a max-min allocation
  - provides performance (throughput/delay/jitter) bound and allows admission control (when used with additional mechanisms)

General Process Sharing (GPS)

- Conceptually, GPS serves packets as if they are in separate logical queues, visiting each non-empty queues in turn
  - In each turn, an infinitesimally small amount of data is served so that in any finite time interval, it can visit all logical queues
  - Obviously, GPS is unimplementable since one cannot serve infinitesimals, only bits or packets
  - However, GPS provides a baseline for the most (max-min) fair packet scheduling

GPS

- A more formal definition of GPS
  - A connection is backlogged whenever it has data in its queue
  - There are N connections with real positive weights $\phi(1),\ldots,\phi(N)$
  - Let $S(i,\tau,t)$ be the amount of data from connection i served in the interval $[\tau,t]$
  - For any backlogged connection i, in any interval $[\tau,t]$ and for j

    $S(i,\tau,t)/S(j,\tau,t) >= \phi(i)/\phi(j)$

    - A non-backlog connection is getting all the resource it needs
    - Backlog connections share all excess resources evenly

What next?

- We can't implement GPS
- So, lets see how to emulate it
- We want to be as fair as possible (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
- Different weights, fixed packet size
  - serve more than one packet per visit, after normalizing to obtain integer weights
  - Example: weight = {1,1.5}, in each round, serves 2 packets from queue 1 and 3 packets from queue 2

(Weighted) round robin

- Different weights, variable size packets
  - 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 microseconds
  - Round time = (250*10 + 250*1) * 88.8 = 2750 * 88.8 = 244.2 ms

Weighted Fair Queueing (WFQ)

- Deals better with variable size packets and weights
- The idea is that assume 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**
- The scheduler tries to emulate the order in which packets are processed by GPS

WFQ: first cut

- Suppose, in each *round,* the server served one bit from each active connection
  - begins with emulating bit-by-bit Round-Robin
- *Round number* is the number of rounds already completed
  - can be fractional
- Each round of service takes a variable amount of time
  - The more connections served, the longer the round takes

WFQ (cont'd)

- 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

- Naively: round number = number of rounds of service completed so far
  - what if a server has not served all connections in a round?
  - what if new conversations join in halfway through a round?
- *Redefine* round number as a real-valued variable that **increases at a rate inversely proportional to the number of currently active connections**
- With this change, WFQ emulates GPS instead of bit-by-bit RR

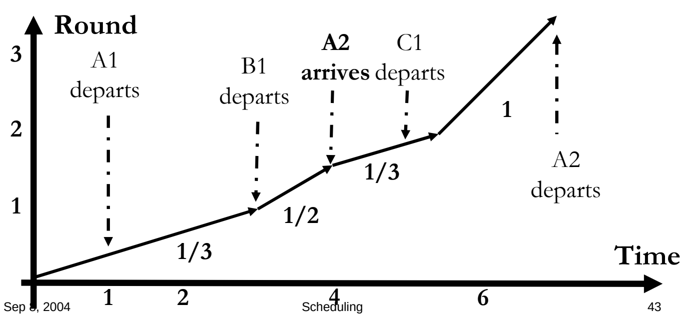
A. Demers and S. Kevhav, “Analysis and Simulation of a Fair Queueing Algorithm,” ACM SIGCOMM’89

WFQ implementation

- On packet arrival:
  - classify packet and look up finish number of last packet served (or waiting to be served)
    - O(1) to O(N)
  - re-compute round number
    - worst case O(N)
  - compute finish number
  - insert in priority queue sorted by finish numbers
    - O(logN)
  - if no space, drop the packet with largest finish number
- On service completion
  - select the packet with the lowest finish number

Example: FQ

- 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



Example: GPS

- Three connections: A,B,C. At t=0, packet of size 1,2 and 2 arrives. (A1,B1,C1). At t=4, A2 of size 2 arrives
- Using GPS:
  - At t=3, all packets get 1 bit of service
    - A1 departs
  - At t=4, B1 and C1 get 1.5 bits of service
    - A2 arrives
  - At t=5 1/2, B1 and C1 get 2 bits of service
    - A2 gets ½ bits of service
    - B1 and C1 depart
  - At t=7, A2 departs
  - Sequence of service = A1, {B1,C1}, A2
  - Departure time = 3, 5.5, 5.5, 7

Example

- FQ
  - Finish #:
    - A1 = 1, B1 = C1 = 2
    - A2 = 3.5
  - Sequence of service: A1, {B1,C1}, A2
  - Departure Time: 1, 2, 5, 7
- GPS
  - Sequence of service: A1, {B1,C1}, A2
  - Departure time: 3, 5.5, 5.5, 7

Analysis

- Unweighted case:
  - if GPS has served *x* bits from connection A by time t
  - WFQ would have served at least *x* - *P* bits, where *P* is the largest possible packet in the network
  - However, WFQ could send *much more* than GPS would => absolute fairness bound > *P*

Evaluation

- Pros
  - like GPS, it provides protection
  - can obtain worst-case end-to-end delay bound
  - gives users incentive to use intelligent flow control (and also provides rate information implicitly)
- Cons
  - needs per-connection state
  - iterated deletion is complicated (occurs during round number computation)
  - requires a priority queue

WFQ Variants

- There are many WFQ variants that are easier to implement and provides different levels of performance bounds
  - SCFQ – self clock fair queueing (1994)
  - DRR – Deficit Round-Robin (1995)
  - W^2FQ – worst-case fair WFQ (1996)
  - and many, many more ….
- In practice, when WFQ variants are available on routers, the number of classes/flows supported tend to be small

Outline

- What is scheduling, why we need it?
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling disciplines
- Buffer management and packet drop strategies

Buffer Management

- Packets that cannot be served immediately are buffered
- How should the buffer be shared among flows/connections?
  - **When buffers is full, a *packet drop strategy is needed***
- Packet losses happen almost always from best-effort connections (why?)
- Shouldn't drop packets unless imperative
  - packet drop wastes resources (why?)

Why is buffer management important?

- Consider the case where there are 2 flows, flow 1 has strict priority over flow 2.
  - Let both flows share the same buffer, of size N, with no differentiation.
  - Let the buffer be empty initially
  - Assume >N packets from flow 2 arrives, occupying all the buffer space
  - Packets from flow 1 arrives later and are dropped (since buffer is full)
  - With sufficiently large difference in arrival rates between flow 2 and flow 1, packets from flow 1 may never be (buffered and) scheduled even though it has higher scheduling priority!!!

Classification of drop strategies

1. Degree of aggregation
2. Drop priorities
3. Drop position
4. Early or late

1. Degree of aggregation

- Degree of discrimination in selecting a packet to drop
  - E.g. in vanilla FIFO, all packets are in the same class
- Instead, can classify packets and drop packets selectively
- Issues:
  - Who decides the aggregation: router or another element?
  - If another element decides, how's the aggregation indicated to the router?
  - How many aggregations are needed?
  - The finer the classification the better the protection but more work/overhead for the network elements

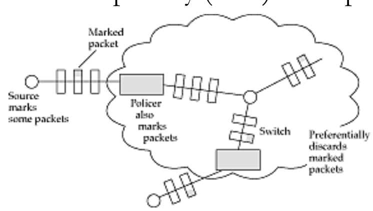
How much buffer space per flow?

- One way is to define a maximum queue length threshold
  - How should the maximum queue length be set ?
  - Static threshold is inflexible
    - If the thresholds are too small, does not support statistical multiplexing efficiently
    - If the thresholds are too large, does not provide isolation
    - Number of flows/connections can change
  - One approach: dynamic thresholding
    - the maximum permissible length at any instant is proportional to the amount of unused buffer
    - **$T(t) = \alpha (B - Q(t)), \alpha = 2,4,\ldots$**
    - Thresholds are sensitive to load and number of flows
    - Some spare capacity is left to handle transit load

- A. K. Choudhury and E. L. Hahne, "Dynamic Queue Length Thresholds for Shared-Memory Packet Switches," IEEE/ACM Trans. Commun., vol. 6, no. 2, Apr. 1998, pp. 130--40.

2. Drop priorities

- Drop lower-priority packets first
- How to choose?
  - endpoint marks packets
  - regulator marks packets
  - congestion loss priority (CLP) bit in packet header



CLP bit: pros and cons

- Pros
  - if network has spare capacity, all traffic is carried
  - during congestion, load is automatically shed
- Cons
  - separating priorities within a single connection is hard
  - what prevents all packets being marked as high priority?

2. Drop priority (contd.)

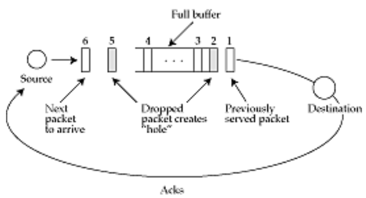
- Special case of AAL5
  - want to drop an entire frame, not individual cells
  - cells belonging to the selected frame are preferentially dropped
- Drop packets from 'nearby' hosts first
  - because they have used the least network resources
  - can't do it on Internet because hop count (TTL) decreases

2. Drop priority (contd.)

- Given a set of aggregates of the same "weight", which aggregate to drop from?
- Drop packet from class with the longest queue
  - Why?
  - Max-min fair allocation of buffers to aggregates

3. Drop position

- Can drop a packet from head, tail, or random position in the queue
- Tail
  - easy
  - default approach
- Head
  - harder
  - lets source detect loss earlier (useful for TCP)



3. Drop position (contd.)

- Random
  - harder to implement
  - if no aggregation, hurts hogs most
- Drop entire longest queue
  - easy
  - almost as effective as drop tail from longest queue

4. Early vs. late drop

- Early drop => drop even if space is available
  - signals endpoints to reduce rate in early stages of congestion
  - cooperative sources get lower overall delays, uncooperative sources get severe packet loss
- Early random drop
  - drop arriving packet with fixed drop probability if queue length exceeds threshold
  - intuition: misbehaving sources more likely to send packets and see packet losses
  - Does it work?

RED

- Random early detection (RED) makes three improvements
- Metric is moving average of queue lengths
  - small bursts pass through unharmed
  - only affects sustained overloads
- Packet drop probability is a function of mean queue length
  - prevents severe reaction to mild overload
- Can mark packets instead of dropping them
  - allows sources to detect network state without losses
- RED improves performance of a network of cooperating TCP sources
- No bias against bursty sources
- Controls queue length regardless of endpoint cooperation

RED Algorithm

- **For each packet arrival**
  - **Calculate the average queue size ave Q**
- **If min <= Q < <= max**
  - **Calculate probability P_a**
  - **With probability P_a, mark the packet**
- **Else if Q > max**
  - **Mark the packet**

- Q is the smoothed version of the queue defined by

$$Q_{k+1}=(1-w) \times Q_k + w \times q$$

where q is the current queue size.
- The smaller the w, the slower Q reflects changes in the queue size.

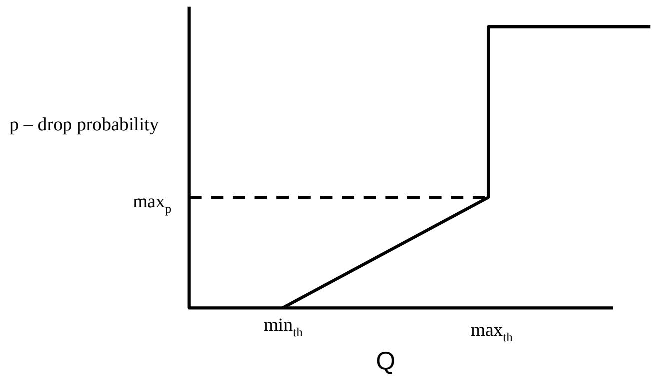
Packet Drop Probability (P_a)

- $$P_b \leftarrow \frac{max_p\,(avg - min_{th})}{(max_{th} - min_{th})}$$

The final packet marking probability

- $$P_a \leftarrow \frac{P_b}{(1 - count \,.\, P_b)}$$
- *count*: # of unmarked packets that have arrived since the last marked packet
- Ensures that the gateway does not wait too long before marking a packet

RED (cont'd)



Issues with RED

- RED is extremely sensitive to # sources and parameter settings
  - Static values of min, max and max_p are not good when network conditions change
- Many variants of RED are proposed:
  - ARED - Adaptive RED
  - FRED - Flow Random Early Drop
  - SRED - Stabilized RED
- Other Active Queue Management
  - BLUE
  - REM