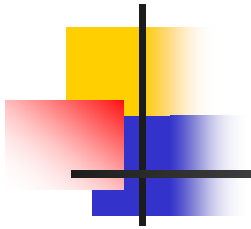
CS 5229

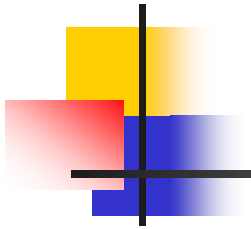
Scheduling

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



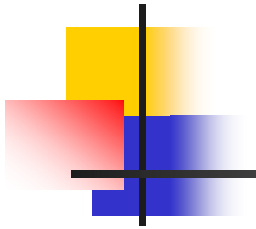
Acknowledgement/Reference

- Some slides are taken from the following source:
  - S. Keshav, “An Engineering Approach to Computer Networking”, Chapter 9: Scheduling



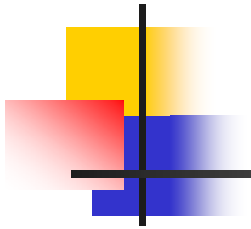
Outline

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



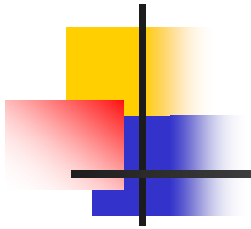
Scheduling

- 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



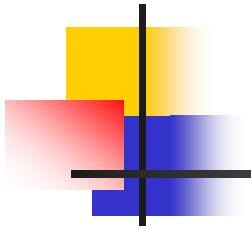
Components

- 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



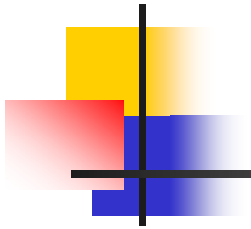
Where?

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



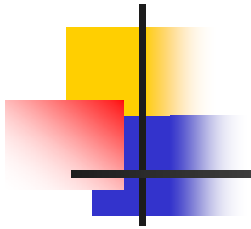
Why do we need one?

- *Because applications need it*
  - *Whenever we need to decide how resources are to be allocated*
- 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



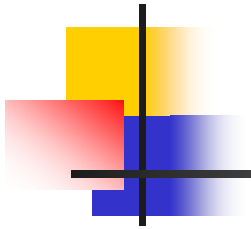
What can scheduling disciplines do?

- 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



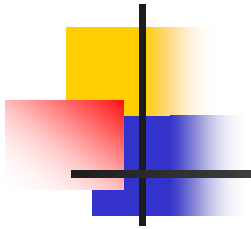
Cont'd

- Applications have different demands on the networks
  - Long flow vs. short flow
  - TCP vs. UDP
  - Rate control vs. continuous stream



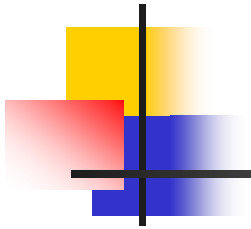
Outline

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



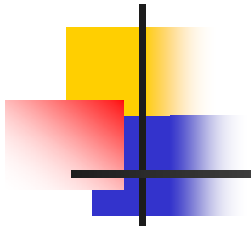
Requirements

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



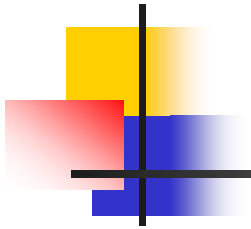
Ease of implementation

- 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



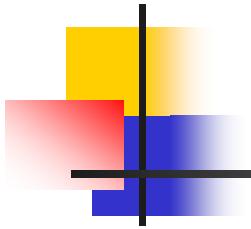
Fairness

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



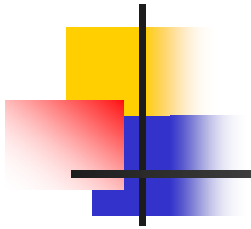
Fairness (contd.)

- 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



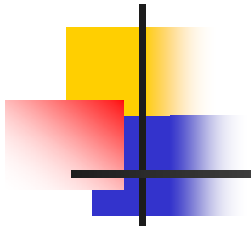
Notion of Fairness

- 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?



Outline

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

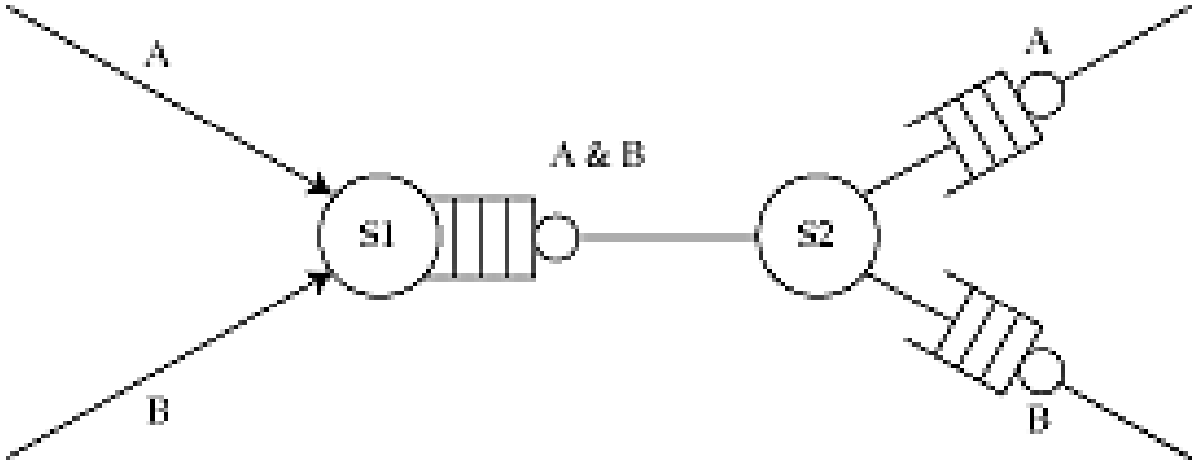


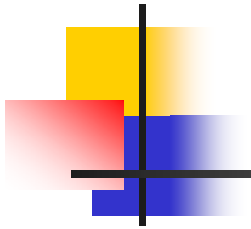
Fundamental choices

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

Work conserving or not?

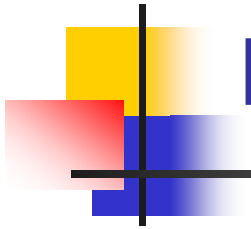
- Work conserving: server is never idle when there is packets awaiting service
  - Maximizes utilization of server resource
- Why bother with non-work conserving?





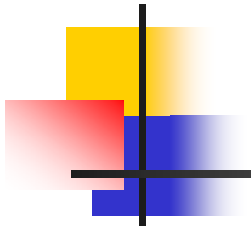
Non-work-conserving disciplines

- 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



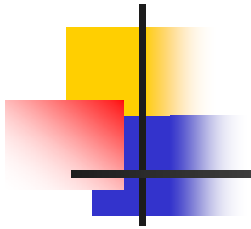
Do we need non-work-conservation?

- 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 (if available)



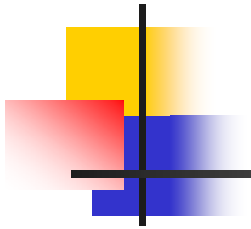
Degree of aggregation

- 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

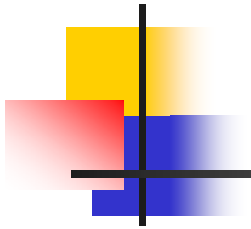
Service-Time Dependent Scheduling

- 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

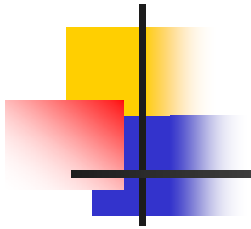
- References: L. Kleinrock, “Queuing Systems,” Volume II, Chapter 3 and 4, 1975.

- AK Parekh, RG Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the single-node case,” IEEE/ACM Transactions on Networking 1993.



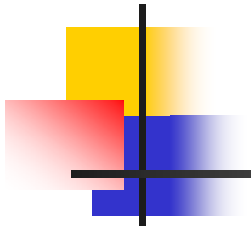
General Process Sharing (GPS)

- 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

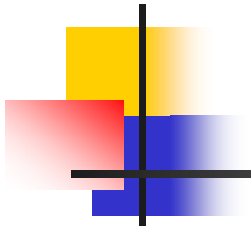


GPS

- 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 $\phi(1),..., \phi(N)$
  - 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

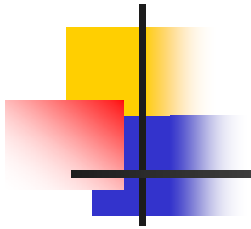
    $S(i,\tau,t) / S(j,\tau,t) >= \phi(i)/\phi(j)$

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



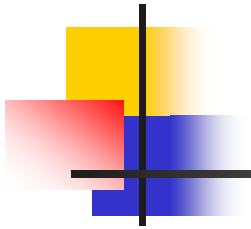
What next?

- 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



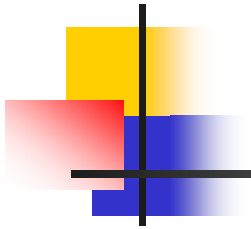
(Weighted) round robin

- 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