EECS 122, Lecture 20

Today's Topics:

Packet Scheduling

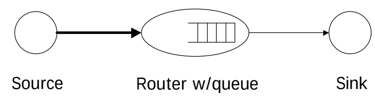
Buffer Management

Congestion Control

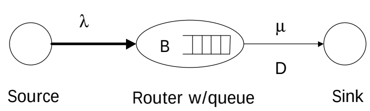
Kevin Fall, kfall@cs.berkeley.edu

Congestion Control Model

– Reduced model includes:
  - data source(s)
  - data sink
  - the router in front of the slowest link (bottleneck router), its queue and queuing discipline

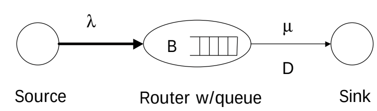


Congestion Control Model



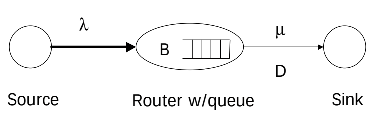
- Reduced Model Parameters:
  - λ: arrival rate
  - μ: service rate
  - D: total round-trip delay (RTT)
  - B: buffer at bottleneck router

Congestion Control Model



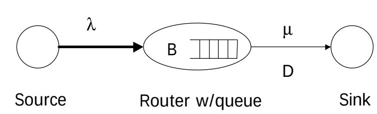
- Observations:
  - $\lambda > \mu$: buffer overrun if persistent
  - $\lambda < \mu$: empty buffer
  - λ and μ are not constant
  - only know μ after a delay (near the RTT)

Relationship to Window Sizes



- In the case of a window-based protocol:
  - recall, throughput is ~ w/RTT
  - so, need (λ = w/RTT) <= μ
  - or: w <= (D μ) [bandwidth-delay product]
  - equality achieves maximum utilization

Goal of Congestion Control



- So, the goal of congestion control is to:
  - keep B at least minimally occupied (with stat mux, will keep link fully utilized)
  - not allow $\lambda > \mu$ to persist

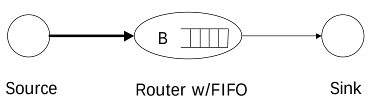
What Happens at a Router

- Router’s job is to *classify* a packet (determine where it is going, and possibly other information)
- Packets often must wait at an output queue before being sent
- Questions: How are these queues maintained? How many of them exist? Does any of this really matter?

What Happens at a Router

- So, really two key questions:
  - what sort of <u>packet scheduling</u> is used:
    - multiple queues?
    - special resources/priorities?
  - what sort of <u>buffer management</u> is used:
    - on overload, what packets are discarded?
    - possible to discard prior to overload?

FIFO Queues



- most simple scheduling and buffer management discipline
- classifier is NULL (no special marking)
- always service head-of-line (FCFS)
- new arrivals to full buffer are dropped (also called “drop-tail”)

Observations on FIFO

- pushes responsibility of congestion control to edges of network
- no sensitivity to type/class of traffic
- A theoretical result [Kleinrock75]:
  - a scheduling discipline can reduce a particular connection’s mean delay, compared with FCFS, only at the expense of another connection

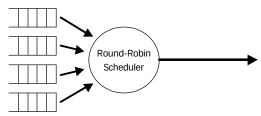
Variants on FIFO

- multiple FIFOs w/priority
- FIFO scheduling with alternative buffer management/discard policies (e.g. drop from head, random drop)

Traffic-Sensitive Queuing

- Problem with simple FIFO is no sensitivity to traffic class/type
- Two issues:
  - not clear that congestion control can be completely effective if implemented only at endpoints
  - lack of per-flow separation allows ill-behaved flows to harm the performance of reactive flows

Fair Queuing (and R/R)



- To provide flow *isolation*, give each flow its own queue and perform *round-robin* scheduling between them
- Provides *local fairness* among flows using end-to-end congestion control algorithms and same packet size

FQ Details

- Packet-by-packet RR fails to give equal bw partitioning when different packet sizes are used
- So, really want *bit-by-bit* round-robin (not practical, instead try to simulate)
- Compute when a packet would have finished (using bit-rr), then use this to order the list of outgoing packets

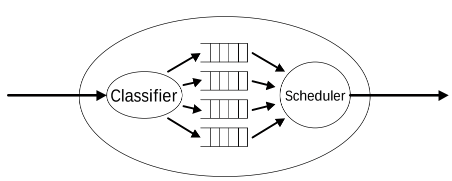
FQ Details

- Proceed as follows:
  - S[i]: start xmit time for pkt i, F[i]: finish xmit time for pkt i, A[i]: arrival time pkt i
  - P[i]: time to xmit pkt i (in bit ticks)
  - F[i] = S[i] + P[i]
  - F[i] = MAX(F[i-1], A[i]) + P[i]
- Use F[i] for each packet of each flow as a deadline, and emit packets earliest deadline first (work-conserving)

Observations on FQ

- work-conserving
- for n flows, each gets <= 1/n bw of link
- can extend FQ to weighted FQ (WFQ) to provide different service between queues (but router must known weight vector)

General Packet Handling Model



- Model for packet handling at router:
  - packet classification (queue selection)
  - scheduling
  - buffer management for each queue

Active Buffer Management

- We have seen both active and passive scheduling (FQ and FCFS)
- Similar issues with buffer management
- Drop-tail is simple, passive buffer management technique
- Active techniques allow for reaction prior to buffer exhaustion
- One example: RED gateways

Random Early Detection (RED)

- Active buffer management technique
- Key components:
  - underlying FIFO packet queue
  - measure of time-averaged queue occupancy
  - randomization
- Idea is that when congestion is imminent, notify sources they should reduce their sending rates

RED Operation

- Time-averaged queue occupancy measure is based on an exponentially-weighted moving average (EWMA):
  - avg = (1-w) * avg + w * (new sample)
  - w is "weight" (gain constant), ~0.002
- Two thresholds:
  - minth: min threshold to initiate random drop/mark
  - maxth: max threshold to use random drop/mark

RED Operation

- On packet arrival, do the following:
  - avg < minth: queue packet normally
  - avg > maxth: drop/mark packet
  - minth < avg < maxth: mark/drop w/prob p
- Probability p given by:
  - t = maxp*(avg-minth)/(maxth-minth)
  - p = t/(1-cnt*t)
  - gives initial p on [0...maxp]
  - cnt is pkt cnt since last random mark/drop

RED Characteristics

- Uses early mark/drop to notify sources prior to buffer overrun; randomization tends to distribute notifications across sources
- Drop/mark probability is roughly proportional to a flow's bandwidth utilization at router
- Underlying buffer size usually considerably bigger than maxth to accommodate short-term bursts

Congestion Avoidance & Control

- We have now seen actions taken at routers/switches to affect traffic flow
- We may also use techniques at sources to limit their load on the network, or combine approaches
- Several ways of doing this...

Congestion Control Taxonomies

- Several ways of characterizing approaches...
- *open loop* or *closed loop*
- *network enforced* or *host enforced*

Open Loop Congestion Control

- source establishes traffic descriptor with network describing its needs
- net typically reserves resources and performs enforcement:
  - admission control for new connections
  - policing at edges for data
- challenges: choosing the traffic descriptor, choosing scheduling discipline at routers, performing admission control

Closed Loop Congestion Control

- network does not reserve resources (no such capability, or want stat. muxing)
- source adjusts its traffic volume based on feedback from network or sink:
  - explicit or implicit state measurement
  - rate-based or window-based
  - hop-by-hop or end-to-end

Perspective on Approaches

- Most common approach today is feedback-based closed-loop congestion control with enforcement at the edges
- Functionality beyond best-effort service (class of service, quality of service) may involve support similar to that in open loop congestion control systems
- For now, we will proceed with studying the predominant closed-loop approach...

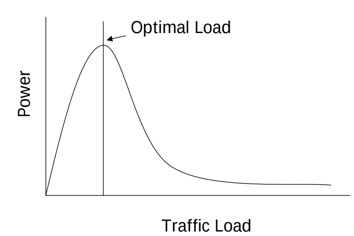
Evaluation Criteria

- Effectiveness
  - want to fully utilize links in network, but filling all queues increases end-to-end delay
  - how to measure throughput/delay tradeoff?
- Fairness
  - how do multiple flows share a common network?
  - if we assume fair means equal, how to measure if a set of flows are receiving equal treatment?

Effectiveness

- Throughput/delay tradeoff
  - with stat muxing (and a *work-conserving* service discipline), outgoing link is always fully utilized if any packet present
  - want to avoid empty queues, but larger queues mean larger delays
- Network power:
  - Power = $(\text{Throughput})^{\alpha} / (\text{Delay})$
  - $0 < \alpha < 1$

Network Power



Jain's Fairness Index

$$f(x_1, x_2 x_3, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

- A definition for fairness:
  - 0 <= f() <= 1, given flow throughputs x_i
  - locally equal partitioning of bandwidth achieves index of 1. If only k of n flows receive equal bw (and others get none), index is k/n
  - what about different-length flows? (p.401)

Congestion Control with TCP

- Congestion control added to TCP in late 80s as a result of congestion collapse problem
- Idea:
  - host figures out how many packets it can safely inject into network
  - each received indicates 1 (or possibly more) packets have been removed from network, allowing host to inject another
  - self-clocking property ensures stability

Challenges for TCP

- How to determine how many packets to inject into network?
  - Too many: overrun buffers
  - too few: underutilization of link
- Additional problems:
  - available bandwidth changes over time as new connections start and terminate
- More next time...