Lecture 12: Queuing, Caching, and Content Distribution

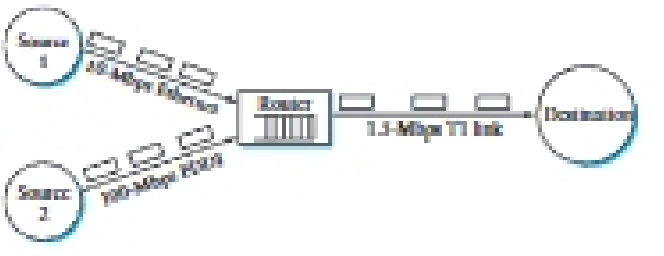
Overview

- How routers queue affects how TCP and other protocols behave
- Two router questions: drop policy, scheduling policy
- Reducing congestion through content distribution
  - Clients can cache
  - Services can use a CDN

Congestion Control Revisited

- Congestion is when the input rate ≫ output rate
  - In TCP, flow control window ensures sender does not exceed rate at which receiver consumes data
  - What if senders exceed a router's maximum output rate?
- What should routers do? Make sender slow down
- TCP sending rate = window-size/RTT, so 2 options:
  1. Increase RTT – buffer more packets ⟶ more queuing delay
  2. Reduce window size – happens if router drops packets
- Recall TCP reacts to packet loss by shrinking congestion window
  - Triple duplicate ack: halve window, enter CA state
  - Timeout: set window to 1, enter SS state

Congestion at Router

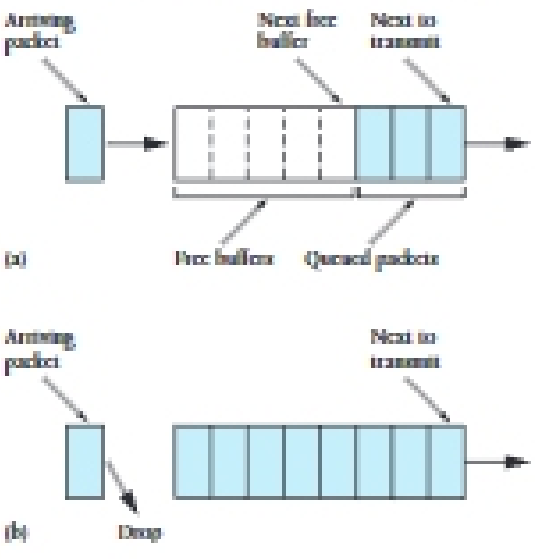


- Router goals
  - Prioritize who gets limited resources
  - Somehow interact well with TCP

Router design issues

- *Scheduling discipline*
  - Which of multiple packets should you send next?
  - May want to achieve some notion of fairness
  - May want some packets to have priority
- *Drop policy*
  - When should you discard a packet?
  - Which packet to discard?
  - Some packets more important (perhaps BGP)
  - Some packets useless w/o others (IP fragments)
- Need to balance throughput & delay
  - Could minimize/eliminate drops with enormous buffers
  - But queuing delay highly frowned upon (interactive apps)

Example: FIFO tail drop



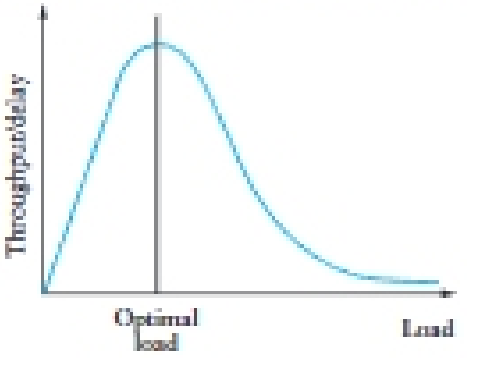
- Differentiates packets only by when they arrive
  - Packet dropped if queue full when it arrives

Tail drop issues

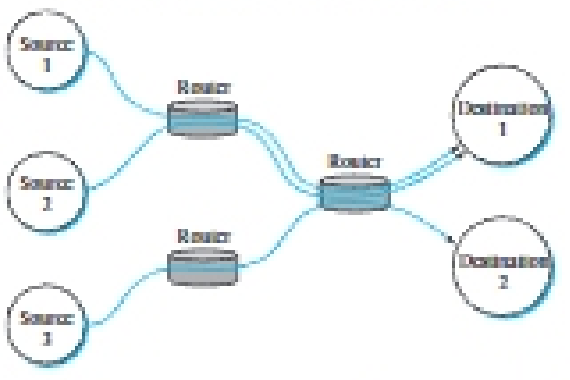
- When stable, queue will always be nearly full
  - Guarantees high latency for all traffic
- Possibly unfair for flows with small windows
  - E.g., small flow (< 4 segments) may be stuck in backoff, while larger flows can use fast retransmit to recover
- Window synchronization
  - Consider many flows in a stable configuration
  - New flow comes in, causes a bunch of packet losses
  - Existing flows all cut their windows together (underutilizing link)
  - Flows all grow their windows together until link again overloaded and many packets lost. Repeat…

What to optimize for?

- *Fairness* (in two slides)
- *High throughput* – queue should never be empty
- *Low delay* – so want short queues
- Crude combination: *power* = Throughput/Delay
  - Want to convince hosts to offer optimal load

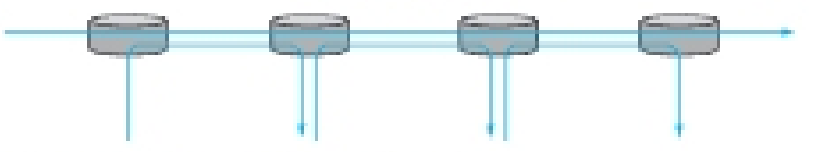


Connectionless flows



- Even in Internet, routers can have a notion of flows
  - E.g., base on IP addresses & TCP ports (or hash of those)
  - *Soft state*—doesn't have to be correct
  - But if often correct, can use to form router policies

Fairness



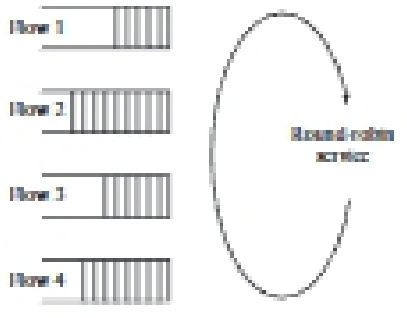
- What is fair in this situation?
  - Each flow gets 1/2 link b/w? Long flow gets less?
- Usually fair means equal
  - For flow bandwidths $(x_1, \ldots, x_n)$, *fairness index*:

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

  - If all x_is are equal, fairness is one
  - Weighted fairness is a simple extension
- So what policy should routers follow?

Scheduling Policy: Fair Queuing (FQ)

- Explicitly segregates traffic based on flows
- Ensures no flow consumes more than its share
  - Variation: weighted fair queuing (WFQ)
- Note: if all packets were same length, would be easy



Fair Queuing Basics

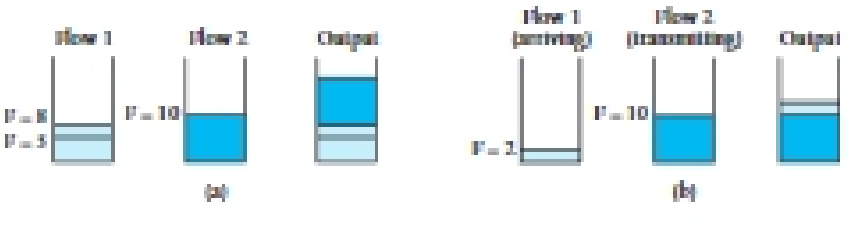
- Keep track of how much time each flow has used link
- Compute how long a flow will have used link if it transmits next packet
- Send packet from flow which will have lowest use if it transmits
  - Why not flow with smallest use so far?
  - Because next packet may be huge (examples coming)

FQ Algorithm

- Suppose clock ticks each time a bit is transmitted
- P_i: length of packet i
- S_i: time when packet i started transmission
- F_i: time when packet i finished transmission
- $F_i = S_i + P_i$
- When does router start transmitting packet i?
  - If arrived before router finished packet $i-1$ from this flow, then immediately after last bit of $i-1$ (F_{i-1})
  - If no current packets for this flow, then start transmitting when arrives (call this A_i)
- Thus: $F_i = \max(F_{i-1}, A_i) + P_i$

FQ Algorithm (cont)

- For multiple flows
  - Calculate F_i for each packet that arrives on each flow
  - Treat all F_is as timestamps
  - Next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example:



FQ Algorithm (cont)

- One complication: inactive flows are penalized ($A_i > F_{i-1}$)
- Over what interval do you consider fairness?
  - Standard algorithm considers no history
  - Each flow gets fair share while packets queued
- Solution: $B_i = P_i + \max(F_{i-1}, A_i - \delta)$
- δ = interval of history to consider

Fair Queueing Importance

- "Our packet-by-packet transmission algorithm is simply defined by the rule that, whenever a packet finishes transmission, the next packet is the one with the smallest F_i."
- But, fair queueing not used in core routers: finding min F in hundreds of thousands of flows is expensive. Can be used on edge routers and low speed links.

Drop Policy: Random Early Detection (RED)

- Notification of congestion is implicit in Internet
  - Just drop the packet (TCP will timeout)
  - Could make explicit by marking the packet (ECN extension to IP allows routers to mark packets)
- Early random drop
  - Don't wait for full queue to drop packet
  - Instead, drop packets with some *drop probability* whenever the queue length exceeds some *drop level*
  - Prevents global window synchronization: many TCP flows speed up, all have packets dropped, all slow down, etc.

RED Details

- Compute average queue length
  - AvgLen = (1 − Weight) · AvgLen + Weight · SampleLen
  - 0 < Weight < 1 (usually 0.002)
- SampleLen is queue length each time a packet arrives