Today: Queue Management

What can individual links do to make good use of shared underlying resources?

Last lecture: Congestion Control

What can end-points do to collectively to make good use of shared underlying resources?

Packet Queues
Queue Management Issues

- **Scheduling discipline**
  - Which packet to send?
  - Some notion of fairness? Priority?

- **Drop policy**
  - When should you discard a packet?
  - Which packet to discard?

- **Goal: balance throughput and delay**
  - Huge buffers minimize drops, but add to queuing delay (thus higher RTT, longer slow start, ...)

FIFO Scheduling and Drop-Tail

- **Access to the bandwidth: first-in first-out queue**
  - Packets only differentiated when they arrive

- **Access to the buffer space: drop-tail queuing**
  - If the queue is full, drop the incoming packet
Early Detection of Congestion

- TCP depends on packet loss
  - Packet loss is indication of congestion
  - TCP additive increase drives network into loss
- Drop-tail leads to *bursty loss*
  - *Congested link*: many packets encounter full queue
  - *Synchronization*: many connections lose packets at once

Slow Feedback from Drop Tail

- Feedback comes when buffer is completely full
  - ... even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - ... making detection even slower
- Better to give early feedback
  - Get 1-2 connections to slow down before it’s too late!

Random Early Detection (RED)

- Router notices that queue is getting full
  - ... and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - Else, set drop probability $f(\text{avg queue length})$

Bursty Loss From Drop-Tail Queuing
Properties of RED

• Drops packets before queue is full
  – In the hope of reducing the rates of some flows
• Tolerant of burstiness in the traffic
  – By basing the decisions on average queue length
• Which of the following are true?
  (Y) Drops packet in proportion to each flow’s rate
  (M) High-rate flows selected more often
  (C) Helps desynchronize the TCP senders
  (A) All of the above

Problems With RED

• Hard to get tunable parameters just right
  – How early to start dropping packets?
  – What slope for increase in drop probability?
  – What time scale for averaging queue length?
• RED has mixed adoption in practice
  – If parameters aren’t set right, RED doesn’t help
• Many other variations in research community
  – Names like “Blue” (self-tuning), “FRED”...
Feedback: From loss to notification

- Early dropping of packets
  - Good: gives early feedback
  - Bad: has to drop the packet to give the feedback

- Explicit Congestion Notification
  - Router marks the packet with an ECN bit
  - Sending host interprets as a sign of congestion

Explicit Congestion Notification

- Needs support by router, sender, AND receiver
  - End-hosts check ECN-capable during TCP handshake
- ECN protocol (repurposes 4 header bits)
  1. Sender marks “ECN-capable” when sending
  2. If router sees “ECN-capable” and congested, marks packet as “ECN congestion experienced”
  3. If receiver sees “congestion experienced”, marks “ECN echo” flag in responses until congestion ACK’d
  4. If sender sees “ECN echo”, reduces cwnd and marks “congestion window reduced” flag in next packet

ECN Questions

Why separate ECN experienced and echo flags?

(Y) Detect reverse path congestion with “experienced”
(M) Congestion could happen in either direction, want sender to react to forward direction
(C) Both of the above

ECN Questions

Why separate ECN experienced and echo flags?

(Y) Detect reverse path congestion with “experienced”
(M) Congestion could happen in either direction, want sender to react to forward direction
(C) Both of the above

Why “echo” resent & “congestion window reduced” ACK?

(Y) Congestion in reverse path can lose ECN-echo, still want to respond to congestion in forward path
(M) Only should apply backoff once per cwnd
(C) Both of the above
ECN Questions

Why separate ECN experienced and echo flags?
- (Y) Detect reverse path congestion with “experienced”
- (M) Congestion could happen in either direction, want sender to react to forward direction
- (C) Both of the above

Why “echo” resent & “congestion window reduced” ACK?
- (Y) Congestion in reverse path can lose ECN-echo, still want to respond to congestion in forward path
- (M) Only should apply backoff once per cwnd
- (C) Both of the above

Link Scheduling

First-In First-Out Scheduling
- First-in first-out scheduling
  - Simple, but restrictive
- Example: two kinds of traffic
  - Voice over IP needs low delay
  - E-mail is not that sensitive about delay
- Voice traffic waits behind e-mail

Strict Priority
- Multiple levels of priority
  - Always transmit high-priority traffic, when present
- Isolation for the high-priority traffic
  - Almost like it has a dedicated link
  - Except for (small) delay for packet transmission
- But, lower priority traffic may starve 😞
Weighted Fair Scheduling

- Weighted fair scheduling
  - Assign each queue a fraction of the link bandwidth
  - Rotate across queues on a small time scale

50% red, 25% blue, 25% green

Max-Min Fairness

- Maximize the minimum rate of each flow
  1. Allocate in order of increasing demand
  2. No flow gets more than demand
  3. The excess, if any, is equally shared

Max-Min Fairness

- Maximize the minimum rate of each flow
  1. Allocate in order of increasing demand
  2. No flow gets more than demand
  3. The excess, if any, is equally shared

\[
\sum \min(r_i, f) = C
\]
**Weighted Fair Queueing**

- Link 1, ingress → Link 2, ingress → Link 3, ingress
- Link 2, egress → Link 3, egress

**Scheduler**

- flow 1
- flow 2
- flow n

**Classifier**

WFQ slides credit: Nick McKeown (Stanford)

---

**Bit-by-Bit Fair Queueing**

**Question:** What is a “flow”?
Flow 5-tuple: protocol, IP source/dest, port src/dest

**Question:** How can we give weights?
Protocol class, bit markings, prefixes, etc.

---

**Bit-by-Bit Weighted Fair Queueing**

- Flows allocated different rates by servicing different number of bits for each flow during each round.

\[ w_1 = 0.1 \]
\[ w_2 = 0.3 \]
\[ w_3 = 0.3 \]
\[ w_4 = 0.3 \]

Order of service for the four queues:
\[ f_1, f_2, f_2, f_3, f_3, f_3, f_4, f_4, f_4, f_1, \ldots \]

---

**Packet vs. “Fluid” System**

- Bit-by-bit FQ is not implementable:
  ...In real packet-based systems:
  - One queue is served at any given time
  - Packet transmission cannot be preempted

- Goal: A packet scheme close to fluid system
  - Bound performance w.r.t. fluid system
First Cut: Simple Round Robin

- Serve a packet from non-empty queues in turn
  - Lets assume all flows have equal weight
- Variable packet length $\Rightarrow$ get more service by sending bigger packets
- Unfair instantaneous service rate (esp. with variable weights)
  - E.g. 500 flows: 250 with weight 1, 250 with weight 10
  - Each round takes 2,750 packet times
  - What if a packet arrives right after its “turn”?

Packet-by-packet Fair Queuing (Weighted Fair Queuing)

Deals better with variable size packets and weights

Key Idea:
1. Determine the *finish time* of packets in bit-by-bit system, assuming no more arrivals
2. Serve packets in order of finish times

Implementing WFQ

**Challenge:** Determining finish time is hard

**Idea:** Don’t need finish time. Need finish *order*.

The finish order is a lot easier to calculate.

Finish order

In what order do the packets finish? **Increasing** $L_i/w_i$

Order of service for the four queues: $f_1, f_2, f_2, f_2, f_3, f_3, f_3, f_4, f_4, f_4, f_1, ...$

Does not change with future packet arrivals!
Bit-by-bit System Round

Round – One complete cycle through all the queues sending $w_i$ bits per queue

**Question:** How long does a round take?

$$\frac{dR}{dt} = \sum_{j\in B(t)} W_j$$

$R(t)$ = # of rounds at time $t$

$C$ = link rate

$B(t)$ = set of backlogged flows

Packet of length $L$ takes $L/w_i$ rounds to serve
Round (aka. “Virtual Time”) Implementation of WFQ

**Question:** What is finish round of kth packet – $F_k$?

**Putting it All Together**

For $k^{th}$ packet of flow $i$ arriving at time $a$:

$$S_i^k = \max(F_i^{k-1}, R(a))$$

$$F_i^k = S_i^k + \frac{L_i^k}{W_i}$$

**Implementation Trade-Offs**

- **FIFO**
  - One queue, trivial scheduler
- **Strict priority**
  - One queue per priority level, simple scheduler
- **Weighted fair scheduling**
  - One queue per class, and more complex scheduler

**Question:** How to compute $R(a)$?

$$\frac{dR}{dt} = \sum_{j \in B_f} C_j W_j$$

**Simple approximation:** Set $R(a)$ to start or finish round of packet currently in service.
Quality of Service Guarantees

Distinguishing Traffic
- Applications compete for bandwidth
  - E-mail traffic can cause congestion/losses for VoIP
- Principle 1: Packet marking
  - So router can distinguish between classes
  - E.g., Type of Service (ToS) bits in IP header

Preventing Misbehavior
- Applications misbehave
  - VoIP sends packets faster than 1 Mbps
- Principle 2: Policing
  - Protect one traffic class from another
  - By enforcing a rate limit on the traffic

Subdividing Link Resources
- Principle 3: Link scheduling
  - Ensure each application gets its share
  - ... while (optionally) using any extra bandwidth
  - E.g., weighted fair queuing
Reserving Resources, and Saying No

- Traffic cannot exceed link capacity
  - Deny access, rather than degrade performance
- Principle 4: Admission control
  - Application declares its needs in advance
  - Application denied if insufficient resources available

Quality of Service (QoS)

- Guaranteed performance
  - Alternative to best-effort delivery model
- QoS protocols and mechanisms
  - Packet classification and marking
  - Traffic shaping
  - Link scheduling
  - Resource reservation and admission control
  - Identifying paths with sufficient resources