CS640: Introduction to Computer Networks

Aditya Akella

Lecture 20 -
Queuing and Basics of QoS

Queuing Disciplines

- Each router must implement some queuing discipline
  - Scheduling discipline
  - Drop policy
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Queuing also affects latency

2

Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
  - FIFO: scheduling discipline
  - Drop-tail: drop policy
- FIFO (first-in-first-out)
  - Implies single class of traffic, no priority
- Drop-tail
  - Arriving packets get dropped when queue is full regardless of flow or importance

3

FIFO + Drop-tail Problems

- Lock-out problem
  - Allows a few flows to monopolize the queue space
  - Send more, get more → No implicit policing
- Full queues
  - TCP detects congestion from loss
  - Forces network to have long standing queues in steady-state
  - Queueing delays – bad for time sensitive traffic
  - Synchronization: end hosts react to same events
    - Full queue → empty → Full → empty…
- Poor support for bursty traffic

4

Lock-out Problem

- Priority queueing can solve some problems
  - Starvation
  - Determining priorities is hard
- Simpler techniques: Random drop
  - Packet arriving when queue is full causes some random packet to be dropped
- Drop front
  - On full queue, drop packet at head of queue
- Random drop and drop front solve the lock-out problem but not the full-queues problem

5

Random Early Detection (RED)

- Drop packets before queue becomes full (early drop)
- Detect incipient congestion
- Avoid window synchronization
  - Randomly mark packets
- Random drop helps avoid bias against bursty traffic

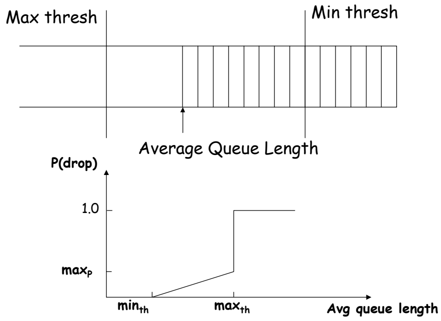
6

RED Algorithm

- Maintain running average of queue length
- If avg < min_{th} do nothing
  - Low queuing, send packets through
- If avg > max_{th}, drop packet
  - Protection from misbehaving sources
- Else mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion

7

RED Operation



8

Fair Queuing: Goals

- How do you protect the most important packets?
  - How do you provide some isolation in general?
  - Simple priority queuing does not help
- Two approaches:
  - Fair Queuing (in itself is sufficient)
  - Leaky bucket (with other techniques which we will cover next class)
- FQ key goal: Allocate resources “fairly”
  - Keep separate queue for each flow
- Isolate ill-behaved users
- Still achieve statistical muxing
  - One flow can fill entire pipe if no contenders
  - *Work conserving* → scheduler never idles link if it has a packet

9

What is "Fairness"?

- At what granularity?
  - Flows, connections, domains?
- What if users have different RTTs/links/etc.
  - TCP is "RTT-Fair"
    - BW inversely proportional to RTT of flow
  - Should they share a link equally or be TCP-fair?
- Maximize fairness index?
  - Fairness = $(\Sigma x_i)^2/n(\Sigma x_i^2)$ $0<\text{fairness}<1$

10

Max-min Fairness

- Allocate user with "small" demand what it wants, evenly divide unused resources to "big" users
- Formally:
  - Resources allocated in terms of increasing demand
  - No source gets resource share larger than its demand
  - Sources with unsatisfied demands get equal share of resource

11

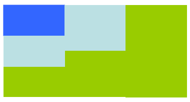
Implementing Max-min Fairness

- Use separate queues per flow
  - Round-robin scheduling across queues
- Why not simple round robin at packet level?
  - Variable packet length → can get more service by sending bigger packets
- Ideally: Bitwise round robin among all queues

12

Bit-by-bit RR Illustration

- Not feasible to interleave bits on real networks
  - FQ simulates bit-by-bit RR



13

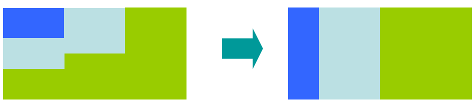
Bit-by-bit RR Simulation

- Single flow: clock ticks when a bit is transmitted. For packet i:
  - P_i = length, A_i = arrival time, S_i = begin transmit time, F_i = finish transmit time
  - $F_i = S_i + P_i = \max(F_{i-1}, A_i) + P_i$
- Multiple flows: clock ticks when a bit from all active flows is transmitted → round number
  - Can calculate F_i for each packet if number of flows is know at all times
    - Why do we need to know flow count? → need to know A → This can be complicated

14

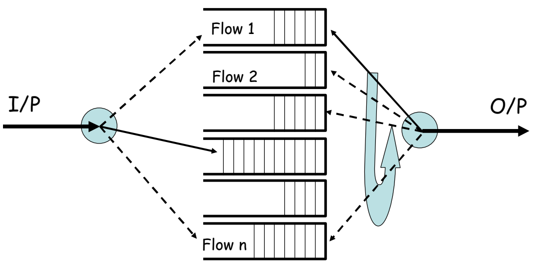
Fair Queuing

- Mapping bit-by-bit schedule onto packet transmission schedule
- Transmit packet with the lowest F_i at any given time



15

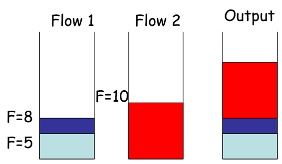
FQ Illustration



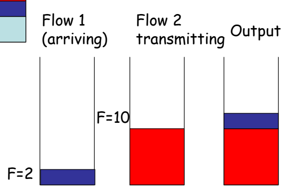
Variation: Weighted Fair Queuing (WFQ) → Flows can have weight
Key to QoS, as we will see in next class.

16

No Pre-emption



Cannot preempt packet
currently being transmitted



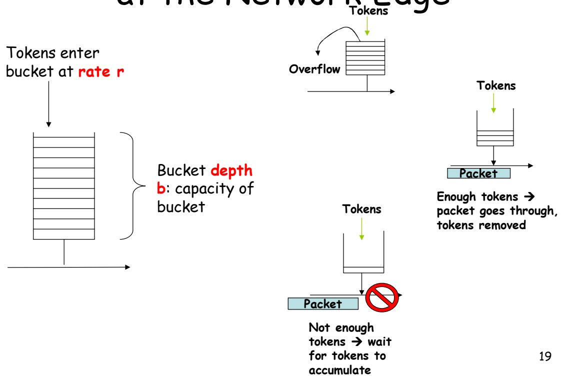
17

Fair Queuing Tradeoffs

- FQ can control congestion by monitoring flows
  - Need flows to be adaptive to avoid congestion collapse
- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. do fairness per domain)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times
  - Must track number of flows at fine time scales

18

Token Bucket for Traffic Policing at the Network Edge



19

Token Bucket Characteristics

- On the long run, rate is limited to r
- On the short run, a burst of size b can be sent
- Amount of traffic entering at interval T is bounded by:
  - Traffic = b + r*T
  - Can provide a lose sense of isolation among flows.
    - Especially because the send rate of each flow is throttled at the source
    - Still need some mechanism within the network to ensure performance guarantees

20