Queue Management

- RED
- Other AQMs
- Assigned reading
  - [FJ93] Random Early Detection Gateways for Congestion Avoidance

Overview

- Queuing Disciplines
- RED
- RED Alternatives

Queuing Disciplines

- Each router must implement some queuing discipline
- 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
Packet Drop Dimensions

- Per-connection state
- Single class
- Class-based queuing
- Drop position
  - Head
  - Tail
  - Random location
- Early drop
- Overflow drop

Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
  - FIFO (first-in-first-out)
    - Implies single class of traffic
  - Drop-tail
    - Arriving packets get dropped when queue is full regardless of flow or importance
  - Important distinction:
    - FIFO: scheduling discipline
    - Drop-tail: drop policy

FIFO + Drop-tail Problems

- Leaves responsibility of congestion control to edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets \(\rightarrow\) get more service
- Synchronization: end hosts react to same events

Active Queue Management

- Design active router queue management to aid congestion control
- Why?
  - Routers can distinguish between propagation and persistent queuing delays
  - Routers can decide on transient congestion, based on workload
**Active Queue Designs**

- Modify both router and hosts
  - DECbit – congestion bit in packet header
- Modify router, hosts use TCP
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion is starting

**Overview**

- DECbit

---

**The DECbit Scheme**

- Basic ideas:
  - On congestion, router sets congestion indication (CI) bit on packet
  - Receiver relays bit to sender
  - Sender adjusts sending rate
- Key design questions:
  - When to set CI bit?
  - How does sender respond to CI?

**Setting CI Bit**

![Diagram showing the calculation of AVG queue length and the timing of CI bits]

Queue length

Current time

Previous cycle

Current cycle

Averaging interval

AVG queue length = (previous busy+idle + current interval)/(averaging interval)
DECbit Routers
- Router tracks average queue length
  - Regeneration cycle: queue goes from empty to non-empty to empty
  - Average from start of previous cycle
  - If average > 1 → router sets bit for flows sending more than their share
  - If average > 2 → router sets bit in every packet
  - Threshold is a trade-off between queuing and delay
  - Compromise between sensitivity and stability
- Acks carry bit back to source

DECbit Source
- Source averages across acks in window
  - Congestion if > 50% of bits set
  - Will detect congestion earlier than TCP
- Additive increase, multiplicative decrease
  - Decrease factor = 0.875
    - Lower than TCP (1/2) – why?
  - Increase factor = 1 packet
- After change, ignore DECbit for packets in flight (vs. TCP ignore other drops in window)
- No slow start

DECbit Evaluation
- Relatively easy to implement
- No per-connection state
- Stable
- Assumes cooperative sources
- Conservative window increase policy

Overview
- Queuing Disciplines
  - RED
- RED Alternatives
Internet Problems

- Full queues
  - Routers are forced to have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state
- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily → allows a few flows to monopolize the queue space

Design Objectives

- Keep throughput high and delay low
- Accommodate bursts
- Queue size should reflect ability to accept bursts rather than steady-state queuing
- Improve TCP performance with minimal hardware changes

Lock-out Problem

- 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

Full Queues Problem

- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If qlen > drop level, drop each new packet with fixed probability $p$
    - Does not control misbehaving users
Random Early Detection (RED)

- Detect incipient congestion, allow bursts
- Keep power (throughput/delay) high
  - Keep average queue size low
  - Assume hosts respond to lost packets
- Avoid window synchronization
- Randomly mark packets
- Avoid bias against bursty traffic
- Some protection against ill-behaved users

RED Algorithm

- Maintain running average of queue length
  - If avgq < min\_th, do nothing
    - Low queuing, send packets through
  - If avgq > max\_th, drop packet
    - Protection from misbehaving sources
  - Else mark packet in a manner proportional to queue length
    - Notify sources of incipient congestion

RED Operation

<table>
<thead>
<tr>
<th>Min thresh</th>
<th>Max thresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg queue length</td>
<td></td>
</tr>
</tbody>
</table>

P(drop)

- 1.0
- max_p

min_q, max_q

Avg queue length

RED Algorithm

- Maintain running average of queue length
- Byte mode vs. packet mode – why?
- For each packet arrival
  - Calculate average queue size (avg)
  - If min\_th = avgq < max\_th
    - Calculate probability P_a
    - With probability P_a
      - Mark the arriving packet
  - Else if max\_th = avg
    - Mark the arriving packet
Queue Estimation

- Standard EWMA: $\text{avgq} = (1 - w_q) \text{avgq} + w_qqlen$
  - Special fix for idle periods – why?
- Upper bound on $w_q$ depends on $\min_{th}$
  - Want to ignore transient congestion
  - Can calculate the queue average if a burst arrives
    - Set $w_q$ such that certain burst size does not exceed $\min_{th}$
- Lower bound on $w_q$ to detect congestion relatively quickly
  - Typical $w_q = 0.002$

Thresholds

- $\min_{th}$ determined by the utilization requirement
- Tradeoff between queuing delay and utilization
- Relationship between $\max_{th}$ and $\min_{th}$
  - Want to ensure that feedback has enough time to make difference in load
  - Depends on average queue increase in one RTT
  - Paper suggest ratio of 2
    - Current rule of thumb is factor of 3

Packet Marking

- Marking probability based on queue length
  - $P_b = \max_p (\text{avgq} - \min_{th}) / (\max_{th} - \min_{th})$
  - Just marking based on $P_b$ can lead to clustered marking
    - Could result in synchronization
    - Better to bias $P_b$ by history of unmarked packets
  - $P_a = P_b / (1 - \text{count} \cdot P_b)$

Packet Marking

- $\max_p$ is reflective of typical loss rates
  - Paper uses 0.02
    - 0.1 is more realistic value
  - If network needs marking of 20-30% then need to buy a better link!
    - Gentle variant of RED (recommended)
      - Vary drop rate from $\max_p$ to 1 as the avgq varies from $\max_{th}$ to $2 \cdot \max_{th}$
      - More robust to setting of $\max_{th}$ and $\max_p$
Extending RED for Flow Isolation

• Problem: what to do with non-cooperative flows?
• Fair queuing achieves isolation using per-flow state – expensive at backbone routers
  • How can we isolate unresponsive flows without per-flow state?
• RED penalty box
  • Monitor history for packet drops, identify flows that use disproportionate bandwidth
  • Isolate and punish those flows

Overview

• Queuing Disciplines
• RED
• RED Alternatives

FRED

• Fair Random Early Drop (Sigcomm, 1997)
• Maintain per flow state only for active flows (ones having packets in the buffer)
  • min_q and max_q → min and max number of buffers a flow is allowed occupy
  • avgcq = average buffers per flow
  • Strike count of number of times flow has exceeded max_q

FRED – Fragile Flows

• Flows that send little data and want to avoid loss
  • min_q is meant to protect these
  • What should min_q be?
    • When large number of flows → 2-4 packets
      • Needed for TCP behavior
    • When small number of flows → increase to avgcq
**FRED**

- Non-adaptive flows
  - Flows with high strike count are not allowed more than avgcq buffers
  - Allows adaptive flows to occasionally burst to max_q, but repeated attempts incur penalty
- Fixes to queue averaging
  - RED only modifies average on packet arrival
  - What if queue is 500 and slowly empties out?
    - Add averaging on exit as well

**CHOKe**

- CHOse and Keep/Kill (Infocom 2000)
  - Existing schemes to penalize unresponsive flows (FRED/penalty box) introduce additional complexity
  - Simple, stateless scheme
- During congested periods
  - Compare new packet with random pkt in queue
  - If from same flow, drop both
  - If not, use RED to decide fate of new packet

**CHOKe**

- Can improve behavior by selecting more than one comparison packet
  - Needed when more than one misbehaving flow
- Does not completely solve problem
  - Aggressive flows are punished but not limited to fair share
  - Not good for low degree of multiplexing → why?

**Blue**

- Uses packet loss and link idle events instead of average queue length – why?
  - Hard to decide what is transient and what is severe with queue length
  - Based on observation that RED is often forced into drop-tail mode
  - Adapt to how bursty and persistent congestion is by looking at loss/idle events
Blue

- Basic algorithm
  - Upon packet loss, if no update in freeze_time then increase $p_m$ by $d_1$
  - Upon link idle, if no update in freeze_time then decrease $p_m$ by $d_2$
  - $d_1 >> d_2$ → why?
    - More critical to react quickly to increase in load

Comparison: Blue vs. RED

- $\max_p$ set to 1
  - Normally only 0.1
  - Based on type of tests & measurement objectives
    - Want to avoid loss → marking is not penalized
    - Enough connections to ensure utilization is good
    - Is this realistic though?
  - Blue advantages
    - More stable marking rate & queue length
    - Avoids dropping packets
    - Much better behavior with small buffers

Stochastic Fair Blue

- Same objective as RED Penalty Box
  - Identify and penalize misbehaving flows
- Create $L$ hashes with $N$ bins each
  - Each bin keeps track of separate marking rate ($p_m$)
  - Rate is updated using standard technique and a bin size
  - Flow uses minimum $p_m$ of all $L$ bins it belongs to
  - Non-misbehaving flows hopefully belong to at least one bin without a bad flow
    - Large numbers of bad flows may cause false positives

Stochastic Fair Blue

- Is able to differentiate between approx. $N^L$ flows
- Bins do not actually map to buffers
  - Each bin only keeps drop rate
  - Can statistically multiplex buffers to bins
  - Works well since Blue handles small queues
  - Has difficulties when large number of misbehaving flows
Stochastic Fair Blue

- False positives can continuously penalize same flow
- Solution: moving hash function over time
  - Bad flow no longer shares bin with same flows
  - Is history reset → does bad flow get to make trouble until detected again?
    - No, can perform hash warmup in background

Next Lecture: Fair Queuing

- Fair Queuing
- Core-stateless Fair queuing
- Assigned reading
  - [DKS90] Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience
  - [SSZ98] Core-Stateless Fair Queueing: Achieving Approximately Fair Allocations in High Speed Networks