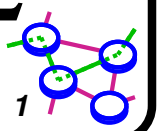
CS551
Random Early Detection (RED)
[Floyd93a]

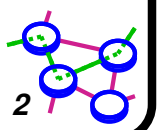
Bill Cheng

http://merlot.usc.edu/cs551-f12



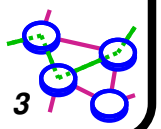
Key Ideas

- Want better fairness without having per-flow state
- Avoid synchronization
  - drop packets randomly
- Allow bursty traffic (internet traffic is very bursty)
  - by keep queues small, can handle bursts in your buffer
- Do early congestion feedback
  - drop packets early (before queue is full)
  - get people to slow down quickly (don't wait)



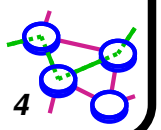
Random Early Detection (RED)

- **Motivation:**
  - high bandwidth-delay flows have large queues to accommodate transient congestion
  - TCP detects congestion from loss - after queues have built up and increase delay
- **Aim:**
  - keep throughput high and delay low
  - accommodate bursts (leave some room)



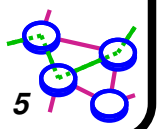
Why Active Queue Management? (RFC2309)

- Lock-out problem
  - drop-tail allows a few flows to monopolize the queue space, locking out other flows (due to synchronization)
- Full queues problem:
  - drop tail maintains full or nearly-full queues during congestion; but queue limits should reflect the size of bursts we want to absorb, not steady-state queueing



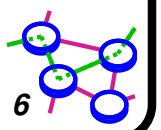
Other Options

- **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 lockout problem but not the full-queues problem**



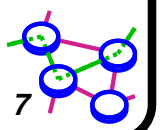
Solving the Full Queues Problem

- 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



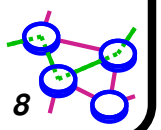
Differences With DEC-bit

- Random marking/dropping of packets
- Exponentially weighted queue lengths
- Senders react to single packet
- Rationale:
  - exponential weighting better for high bandwidth connections
  - no bias when weighting interval different from roundtrip time, since packets are marked randomly
  - random marking avoids bias against bursty traffic

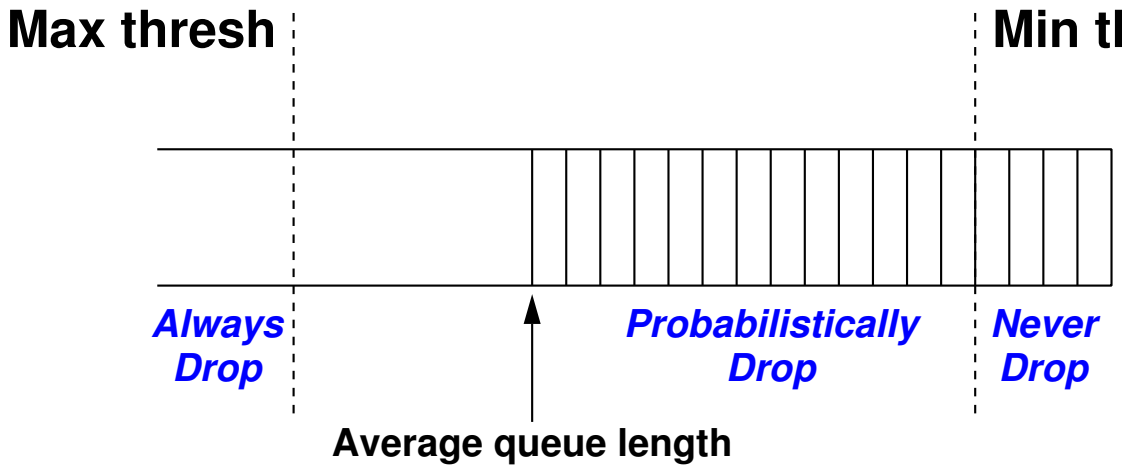


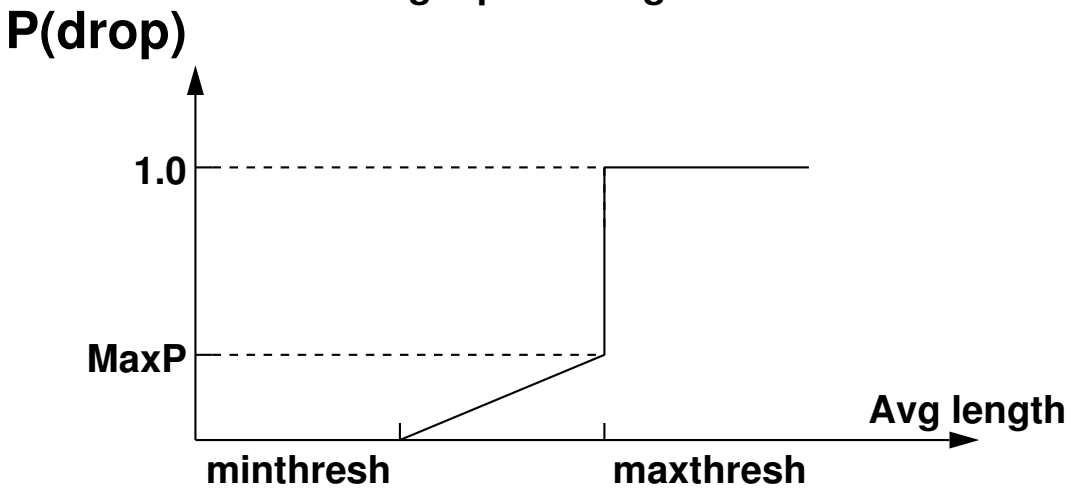
RED Goals

- Detect incipient (soon to happen) 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



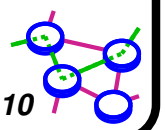
RED Operation





Red Algorithm

```
Initialization:
    avg ← 0
    count ← −1
for each packet arrival
    calculate the new average queue size avg:
        if the queue is nonempty
            avg ← (1 − w_q)avg + w_q q
        else
            m ← f(time − q_time)
            avg ← (1 − w_q)^m avg
    if min_th ≤ avg < max_th
        increment count
        calculate probability p_a:
            p_b ← max_p(avg − min_th)/(max_th − min_th)
            p_a ← p_b/(1 − count · p_b)
        with probability p_a:
            mark the arriving packet
            count ← 0
    else if max_th ≤ avg
        mark the arriving packet
        count ← 0
    else count ← −1
when queue becomes empty
    q_time ← time
```

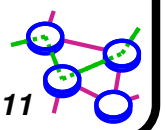


Queue Estimation

- Standard EWMA: avg = (1-w_q) avg + w_q qlen
- Upper bound on w_q depends on min_{th}
  - want to set w_q to allow a certain burst size L

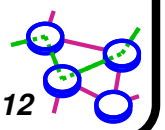
$$L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} < min_{th}$$

  - Ex: min_{th}=5 and L=50, w_q < 0.0042
- Lower bound on w_q to detect congestion relatively quickly
  - Ex: w_q = 0.002



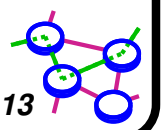
Thresholds

- min_{th} determined by the utilization requirement
  - needs to be high for fairly bursty traffic
- max_{th} set to twice min_{th}
  - rule of thumb
  - difference must be larger than queue size increase in one RTT
    - bandwidth dependence



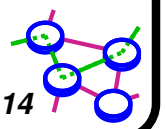
Packet Marking

- Marking probability based on queue length
  - $Pb = max_p(avg - min_{th}) / (max_{th} - min_{th})$
- Just marking based on Pb can lead to clustered marking
  $\Rightarrow$ global synchronization
- Better to bias Pb by history of unmarked packets
  - $Pb = Pb/(1 - count \times Pb)$
    - where count is the number of unmarked packets that have arrived since the last marked packet



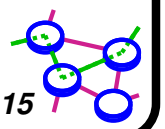
RED Variants

- FRED: Fair Random Early Drop (Sigcomm, 1997)
  - maintain per flow state only for active flows (ones having packets in the buffer)
- CHOKe (choose and keep/kill) (Infocom 2000)
  - compare new packet with random pkt in queue
  - if from same flow, drop both
  - if not, use RED to decide fate of new packet



Extending RED for Flow Isolation

- Problem: what to do with non-cooperative flows?
- Fair queuing achieves isolation using perflow state
  - expensive at backbone routers
- Pricing can have a similar effect
  - but needs much infrastructure to be developed
- How can we isolate unresponsive flows without per-flow state?



Red Penalty Box

- With RED, monitor history for packet drops, identify flows that use disproportionate bandwidth
- Isolate and punish those flows