Congestion Control – Routers

CSE 561 Lecture 8, Spring 2002.
David Wetherall

Problems with FIFO/Drop tail

- **Persistent queuing**
- **Synchronization**
- **Burst losses**
- **Lack of flow isolation**

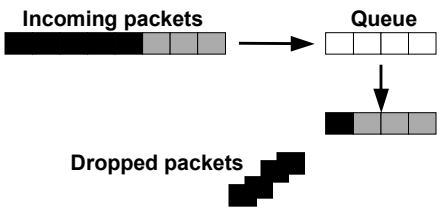
Persistent queuing

- Queues exist to absorb transient bursts
  - Need big enough queue to deal with BW*delay of link
- We want average queue length to be small
  - Non-transient queuing unnecessarily increases latency

- TCP will fill queue until a loss occurs
- Naturally keeps average queue length high

Burst losses

- Real life traffic is bursty and structured
  - Lots of packets from same flow back-to-back
- If the queue is full, many packets from the same flow may be lost
- TCP will likely timeout
  - May not have enough duplicate acks for fast retransmit
  - TCP Reno doesn't handle multiple losses in a window well



Synchronization

- Multiple TCP flows through a router
- Queue fills up
- Arriving packets from all flows will be dropped
- Drops cause all TCP flows to slow down
- Queue empties
- TCPs ramp up together causing congestion

- Repeat…

Flow Isolation

- FIFO / Drop Tail doesn't differentiate among flows

- Scenario:
  - 1 UDP flow sending at 100Mbps
  - 99 TCP flows
  - What happens?

Active Queue Management

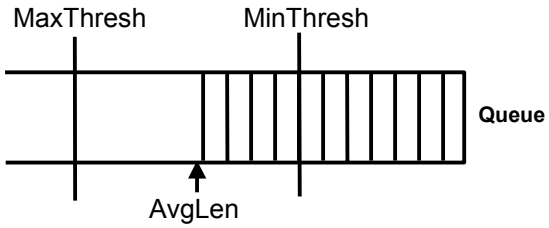
- Idea: Use buffer management to improve congestion signaling and hence queuing behavior
- Precursors
  - IP Source Quench
    - Router sends ICMP packet to host, "hey, partner, slow down"
  - Early Random Drop
    - When buffer beyond drop level, drop incoming packets according to a drop probability
    - Biases bursty traffic
  - DECbit
    - Set congestion-indication bit in packets when average queue length is greater than a threshold
    - Source reduces window when it sees half of packets with bit set

Random Early Detection (RED)

- Key ideas
  - Use *congestion avoidance* to keep average queue size low
  - Detect congestion by monitoring queue size
  - Signal congestion **probabilistically**
    - Drop packets (compatible with existing TCPs)
    - Also supports packet marking (ala DECbit)
- Nice side effects
  - Less synchronization (random)
  - Fewer burst losses

Basic RED algorithm I

- Set two static trigger parameters
  - MinThresh, MaxThresh: define where queue length *should* be
- Calculate average queue length
  - AvgLen = (1-Weigth)*AvgLen + Weight * SampleLen
  - EWMA: Weight parameter decides importance of new samples

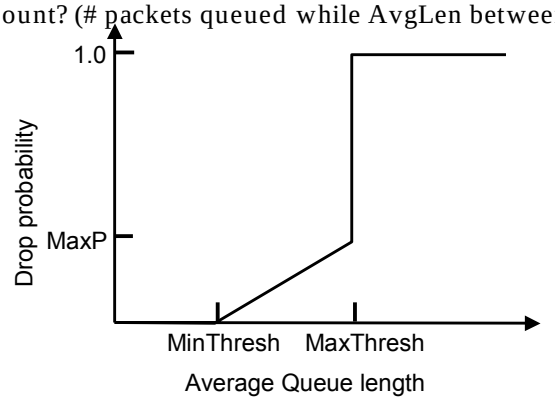


Basic RED algorithm II

- When a packet arrives:
  - If AvgLen < MinThresh **do nothing**
  - If AvgLen > MaxThresh **drop packet**
  - If MinThresh < AvgLen < MaxThresh,
    Drop / mark packet with probability P
- Where does P come from?

Basic RED algorithm III

- tmpP = MaxP*(AvgLen – MinThresh) / (MaxThresh-MinThresh)
- P = tmpP / (1 – count * tmpP)
- Count? (# packets queued while AvgLen between thresholds)



RED Marking

- With RED, you can either mark or drop packets
  - When would it be better to mark instead of drop? Vice versa?
  - Think about assumptions you make about the hosts
- Marking represents another congestion signal to TCP
  - Bit in header: Explicit Congestion Notification (ECN)
    - Proposed, RFCed, not deployed
  - Forced a drop before queue fills up (c.f., FIFO)

Implementation/Deployment issues

- RED was first introduced in the early 1990's w/lots of academic/IETF political support
- Still not widely deployed…

- Three key issues
  - "You're going to drop my packets randomly?"
  - Naïve implementation is slow
    - Recalculate P for each forwarded packet
    - Random number generation is slow
  - How to configure it?

What does RED accomplish?

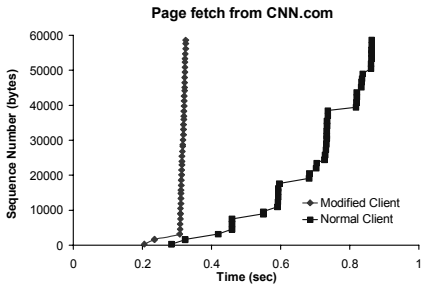
- Keeps average queue length smaller
- Reduces likelihood of synchronization
- Reduces likelihood of burst losses

What doesn't RED do?

- Flows still aren't isolated
- RED doesn't differentiate between flows
- Depends on hosts to be well behaved and back off when packets are dropped / marked
- Return to the UDP scenario…

The Role of the Network vs. End Host

- Problem: Hark! The sky is falling!
  - Non-congestion-controlled traffic is going to cause another congestion collapse (streaming media, "accelerators", etc.)



- Implication is that the network must help itself.

Network Protection/Isolation

- Assume the network must now participate in controlling its utilization, but that we still need E2E congestion control
  - Network mechanisms provide isolation / protection
  - Hosts must still adapt their own flow behavior
- Possibilities:
  - RED "penalty box": punish aggressive flows to incent good behavior
  - Fair queuing: better isolate competing flows from one another
  - Pricing: charge user for packets during congestion!

Issue: What is "good behavior"?

- One answer is "TCP-Friendly" flows
  - A TCP-Friendly flow has an arrival rate limited by VJ congestion avoidance (mult. decrease, add. increase)
  - In steady state this has a known analytic upper bound

$$BW \approx \frac{MSS}{RTT}\frac{0.7}{\sqrt{p}}$$

- Then drop flow's packets to throttle to expected bandwidth. There are some limitations however:
  - Need to know MSS, RTT, drop rate
  - B/w depends upon RTT, need to use low RTT estimate
  - Flow length? Fragmentation? Bursty vs smooth traffic?
  - Unresponsive flows?

Fair Queuing