CSE 222
Graduate Networking

Fall 2001

Lecture 5:
Congestion Control
(Network buffer management)

Stefan Savage
Roadmap: The Class So Far

- Network design and architecture
  - End-to-end argument, Internetworking, Host-oriented networking
- Reliable delivery w/ flow control: Essential transport layer
  - Reliability, flow control at hosts (TCP, sliding window, FEC)
- End-to-end congestion control: Voluntary bandwidth management
  - Need congestion control to prevent collapse, also done at hosts

- End-to-end congestion control: How can network participate?
  - RED: Congestion support in routers improves performance
  - Router incentives: Congestion support in routers to prevent collapse, i.e., now needed for correctness
  - Fair queuing: complete isolation
Review of queuing

- Want queues to absorb transient bursts of traffic
  - Enough for BW*delay
  - Want average queue length low
- Two properties of queues
  - **Scheduling**
    - When to send a packet
    - Which packet to send
  - **Buffer management**
    - When to drop a packet
    - Which packet to drop
- Most routers use FIFO/Drop Tail queues
Problems with FIFO/Drop tail

- Persistent queuing
- Synchronization
- Burst losses
- 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
  - $\text{AvgLen} = (1-\text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}$
  - EWMA: Weight parameter decides importance of new samples
  - Why average? Why no RTT bias?
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

- \( \text{tmpP} = \frac{\text{MaxP} \times (\text{AvgLen} - \text{MinThresh})}{(\text{MaxThresh} - \text{MinThresh})} \)
- \( P = \frac{\text{tmpP}}{1 - \text{count} \times \text{tmpP}} \)
- 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?
Tuning RED

- Guidelines for setting magic variables
  - MinThresh
    - Large enough to allow link to be utilized
  - MaxThresh
    - Enough > MinThresh that average queue length won’t change this much in 1 RTT
    - Enough less than MaxQueue to handle bursts
  - MaxP
    - How aggressively you want to signal congestion
  - Weight
    - Small enough to filter out short term changes in queue length over one RTT
- How do the number of connections impact tuning?
- What about the distribution of flow lengths?
What does RED accomplish?

- Keeps average queue length smaller
- Reduces likelihood of synchronization
- Reduces likelihood of burst losses
  - Why?
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…
Promoting E2E Congest Ctrl

- Problem: Hark! The sky is falling!
  - Non-congestion-controlled best-effort traffic is going to cause another congestion collapse in the Internet
  - Streaming UDP, TCP “accelerators”, etc.

![Page fetch from CNN.com](image-url)
Network Helps Itself

- Implication: The network must now participate in controlling its utilization
  - Does this change the underlying assumptions of the Internet?
  - Does this violate the E2E argument?
- But still need E2E congestion control
  - Network mechanisms provide isolation/protection
  - Hosts must still adapt their own flow behavior
Incentivize Good Behavior

- The approach advocated in this paper is to have the network provide hosts with the incentive to behave
  - If you’re a good citizen, no one bothers you
  - If you misbehave, you get your hand slapped
  - => Remove incentive to misbehave

- Mechanism
  - Identify **non-compliant flows** using RED drop history
  - Rate-limit bandwidth for such flows at router
Regulating Flows

- Three approaches for regulating flows at routers
  - Non TCP-Friendly flows
  - Unresponsive flows
  - Flows consuming disproportionate bandwidth
- Only do this for high bandwidth flows
  - Too much processing to do for all flows
  - High bandwidth flows are the ones causing the problem
  - Estimate bandwidth based on RED drop history
- Only do this when router senses imminent congestion
  - Still have to do more work at the worst time, though
The approaches in this paper examine flow throughput
- Use flow throughput to decide whether a flow is not TCP-friendly, unresponsive, consumes too much b/w

Problem: TCP throughput is a function of RTT
- TCP is clocked by ACKs, e.g., Slow-Start
- TCP needs to wait RTTs to open congestion window
- Larger RTT => lower TCP throughput

Implication: Two TCP flows can have different throughputs even though both are behaving correctly
- Need to make assumptions about when behavior is not RTT dependent, but actually bad behavior
TCP-Friendly Flows

- Approach: Constrain non TCP-Friendly flows
  - Defn: 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

\[ BW \approx \frac{MSS \ 0.7}{RTT \ \sqrt{p}} \]

- Action: Drop flow’s packets to throttle to expected b/w

- Limitations
  - Need to know MSS, RTT, drop rate
    - Use average link drop rate to estimate drop rate
  - B/w depends upon RTT, need to use low RTT estimate
  - Flow length? Fragmentation? Bursty vs smooth traffic?
Unresponsive Flows

- **Approach:** Constrain flows unresponsive to drops
  - Router drops when there is congestion
  - Cooperative flows will decrease throughput in response
    - Drop rate grows $\times x$, throughput should decrease $x^{1/2}$
  - If a flow’s throughput does not drop, it is unresponsive

- **Action:** Drop packets for unresponsive flows
  - Throttle flow back to $x^{1/2}$ level (can be rough approx)

- **Limitations**
  - High initial bandwidth?
  - Flows with variable demands (e.g., short flows)
Bandwidth Hogs

- Approach: Constrain flows using disproportionate amount of bandwidth
  - Disproportionate share: Intuitively, a flow experiencing drops should get no more than \( \frac{1}{n} \) of the bandwidth
  - Relative to level of congestion: Drops due to congestion should limit bandwidth of a cooperating flow
- Action: Drop until flow is only using its portion
- Limitations
  - Gauging level of unsatisfied demand (i.e. is the hog really a big problem?)
Finally... two other approaches

- **Fair queuing**
  - Give each flow its own virtual queue
  - Schedule round-robin bit-by-bit amount these queues
  - Pro: perfect isolation
  - Con: (Perceived?) implementation cost. What’s a flow?
  - You still need congestion control for per-flow adaptation

- **Pricing**
  - Make packets cost more during congestion
  - Create incentive to not send packets that might get dropped
  - Open research... implementation issues, analysis issues, deployment/business model issues
Discussion

- Which approach do you think has the best tradeoffs?
- Do you agree with the high-level argument of the paper?
  - Routers should provide incentive to do congestion control
- Do you think the solutions are too TCP-centric?
  - Based upon assumptions of TCP congestion control
- Tired of congestion control yet? Almost done…
For next time...

- Balakrishnan97, Blakakrishnan99
- Get those project groups together...