Router Congestion Control: RED, ECN, and XCP

Where we left off

- Signal of congestion: Packet loss
- Fairness: Cooperating end-hosts using AIMD
  - Next lecture: Enforcement for QoS, rate, delay, jitter guarantees
- But note: A packet drop is a very blunt indicator of congestion
- Routers know more than they're telling…

What Would Router Do?

- Congestion Signaling:
  - Drop, mark, send explicit messages
- Buffer management:
  - Which packets to drop?
  - When to signal congestion?
- Scheduling
  - If multiple connections, which one's packets to send at any given time?

Congestion Signaling

- Drops (we've covered)
- In-band marking
  - One bit (congested or not): ECN
  - Multiple bits (how congested / how much available): XCP
- Out-of-band notification
  - IP Source Quench
    - Problem: It sends *more* packets when things are congested…
    - Not widely used.

When to mark packets?

- Drop-tail:
  - When the buffer is full
  - The de-facto mechanism today
  - Very easy to implement
  - Causes packets to be lost in bursts
    - Can lose many packets from a single flow…
    - Can cause synchronization of flows
  - Keeps average queue length high
    - ½ full. → delay
  - Note relation to FIFO (first-in-first out): a scheduling discipline, NOT a drop policy, but they're often bundled

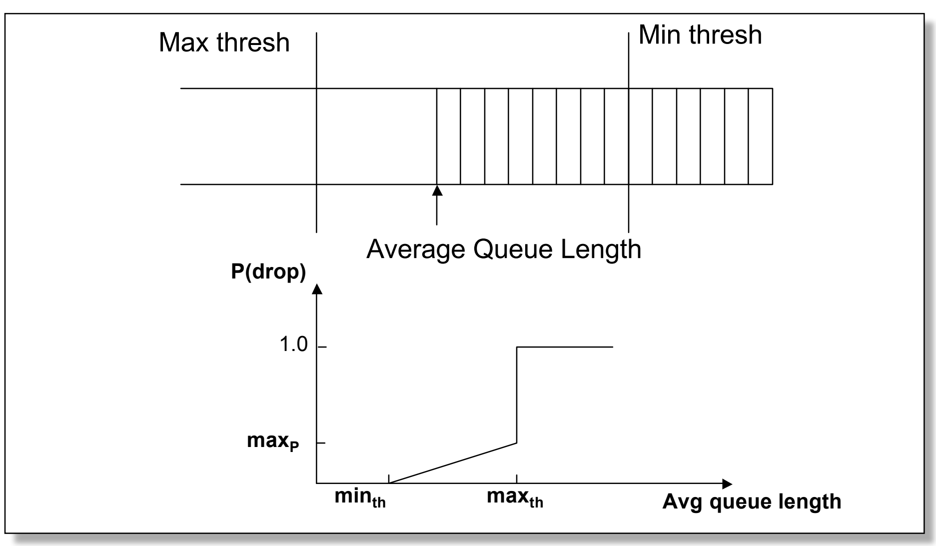
Active Queue Mgmt. w/RED

- Explicitly tries to keep queue small
  - Low delay, but still high throughput under bursts
  - (This is "power": throughput / delay)
- Assumes that hosts respond to lost packets
- Technique:
  - Randomization to avoid synchronization
    - (Recall that if many flows, don't need as much buffer space!)
  - Drop *before* the queue is actually full

RED algorithm

- If qa < min
  - Let all packets through
- If qa > max
  - Drop all packets
- If qa > min && qa < max
  - Mark or drop w/probability p_a
- How to compute qa? How to compute pa?

RED Operation



Computing qa

- What to use as the queue occupancy?
  - Balance fast response to changes
  - With ability to tolerate transient burps
  - Special case for idle periods…
- EWMA to the rescue again…
  - Qa = (1 – wq)*qa + w_q * q
- But what value of wq?
  - Back of the envelope: 0.002
  - RED is sensitive to this value, and it's one of the things that makes it a bit of a pain in practice
  - See http://www.aciri.org/floyd/red.html

Computing pa

- Pb via linear interpolation
  - Pb = max_p * (qa – min / max – min)
- Method 1: pa = pb
  - Geometric random variable for inter-arrivals between drops.
  - Tends to mark in batches (→ Sync)
- Method 2:
  - Uniform r.v. X be uniform in {1, 2, … 1/pb-1}
  - Set pa = pb/(1-count * pb)
    - Count = # unmarked packets since last mark

RED parameter sensitivity

- RED can be very sensitive to parameters
  - Tuning them is a bit of a black art!
- One thing: "gentle" RED
  - max_p <= pb <= 1 as
  - maxthresh <= qa <= 2*maxthresh
  - instead of "cliff" effect. Makes RED more robust to choice of maxthresh, max_p
- But note: Still must choose wq, minthresh…
- RED is not very widely deployed, but testing against both RED and DropTail is very common in research, because it *could* be.

"Marking", "Detection"

- RED is "Random Early *Detection"*
  - Could mean marking, not dropping
- Marking?
  - DECbit: "congestion indication" binary feedback scheme.
  - If avg queue len >thresh, set the bit
  - If > half of packets marked, exponential decrease, otherwise linear increase

Marking 2: ECN

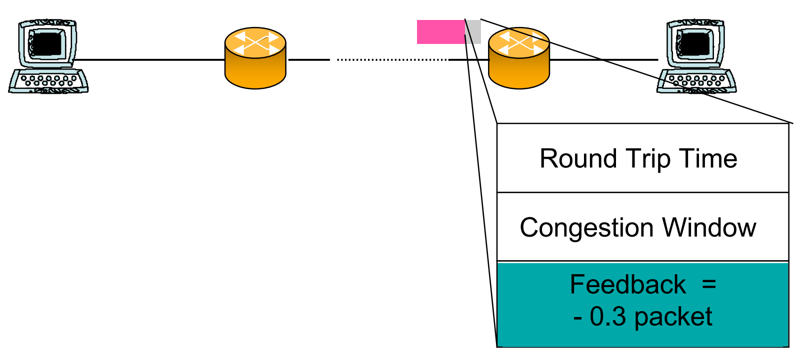
- In IP-land
  - Instead of *dropping* a packet, set a bit
  - If bit set, react the same way as if it had been dropped (but you don't have to retransmit or risk losing ACK clocking)
- Where does it help?
  - Delay-sensitive apps, particularly low-bw ones
  - Small window scenarios
- Some complexity:
  - How to send in legacy IP packets (IP ToS field)
  - Determining ECN support: two bits (one "ECN works", one "congestion or not"
  - How to echo bits to sender (TCP header bit)
- More complexity: Cheating!
  - We'll come back to this later. :)

Beyond congestion indication

- Why do we want to do more?
- TCP doesn't do so well in a few scenarios:
  - High bandwidth-delay product environments
    - Additive increase w/1000 packet window
    - Could take many RTTs to fill up after congestion
    - "not a problem" with a single flow with massive buffers (in theory)
    - a real problem with real routers and bursty cross-traffic
  - Short connections
    - TCP never has a chance to open its window
  - One caveat: A practical work-around to many of these problems is opening multiple TCP connections. The effects of this are still somewhat unexplored with regard to stability, global fairnes and efficiency, etc.

XCP

How does XCP Work?



How Does an XCP Router Compute the Feedback?

Congestion Controller

Goal: Matches input traffic to link capacity & drains the queue

Looks at aggregate traffic & queue

Algorithm: **MIMD**

Aggregate traffic changes by Δ

Δ ~ Spare Bandwidth

Δ ~ - Queue Size

So, $\Delta = \alpha\, d_{avg}\, Spare - \beta\, Queue$

Fairness Controller

Goal: Divides Δ between flows to converge to fairness

Looks at a flow's state in Congestion Header

Algorithm: **AIMD**

If Δ > 0 ⇒ Divide Δ equally between flows

If Δ < 0 ⇒ Divide Δ between flows proportionally to their current rates

Getting the devil out of the details …

Congestion Controller

$\Delta = \alpha\, d_{avg}\, Spare - \beta\, Queue$

Theorem: System converges to optimal utilization (i.e., stable) for any link bandwidth, delay, number of sources if:

$$0 < \alpha < \frac{\pi}{4\sqrt{2}} \quad and \quad \beta = \alpha^2\sqrt{2}$$

No Parameter Tuning

Fairness Controller

Algorithm:

If Δ > 0 ⇒ Divide Δ equally between flows

If Δ < 0 ⇒ Divide Δ between flows proportionally to their current rates

Need to estimate number of flows N

$$N = \sum_{pkts\ in\ T} \frac{1}{T \times (Cwnd_{pkt} / RTT_{pkt})}$$

No Per-Flow State

Apportioning feedback

- Tricky bit: Router sees queue sizes and throughputs; hosts deal in cwnd. Must convert.
- Next tricky bit: Router sees packets; host's response is the *sum* of feedback received across its packets. Must apportion feedback onto packets.
- Requirement: No per-flow state at router

XCP: Positive Feedback

- spare b/w to allocate
- N flows
- per-flow: Δ propto rtt
  - Larger RTT needs more cwnd increase to add same amount of b/w
- per-packet:
  - # packets observed in time d ~ cwnd/rtt
  - combining them: pi ~ spare/N * rtt^2 / cwnd

But must allocate to a flow

- How many packets does flow I send in time T?
  - T * cwnd_I / RTT/I
- So to count # of flows
  - counter += 1 / (T * cwnd_pkt / RTT_pkt)
  - every time you receive a packet
- So: per-flow increase ~ spare / counter
- This is a cute trick for statelessly counting the # of flows.
- Similar to tricks used in CSFQ (Core Stateless Fair Queueing), which we'll be hitting next time

XCP decrease

- Multiplicative Decrease
  - cwnd = beta * cwnd_old (same beta for all flows)
  - This is like the reverse of the slow-start mechanism
    - Slow start: Each ACK, increase cwnd by 1
      - Results in exponential _increase_
    - XCP decrease: Each packet, decrease cwnd
    - BUT: Must account for rtt_I != avg RTT, so normalize
      - ni = total decrease * (rtt_I / avg_rtt)

XCP benefits & issues

- Requires “policers” at edge if you don’t trust hosts to report cwnd/rtt correctly
  - Much like CSFQ…
- Doesn’t provide much benefit in *today’s* common case
  - But may be very significant for *tomorrow’s*.
  - High bw*rtt environments (10GigE coming to a desktop near you…)
  - Short flows, highly dynamic workloads
- Cool insight: Decoupled fairness and congestion control
- Pretty big architectural change

Beyond RED

- What if you want to use RED to try to enforce fairness?

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?

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

- 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

Acknowledgements

- Several of the XCP slides are from Dina Katabi’ SIGCOMM presentation slides.
- http://www.ana.lcs.mit.edu/dina/XCP/