Avoiding Congestion: Strategies

- TCP’s strategy: congestion control
  - Control congestion *once* it occurs
    - Repeatedly increase load imposed on network
    - Back-off when congestion occurs
  - TCP *needs* to create losses to find available bandwidth for transfer

- Alternative strategy: congestion avoidance
  - Predict when congestion is about to happen
  - Reduce host sending rate *before* packets start getting dropped

- We’ll study four different congestion avoidance strategies:
  - Router-assisted strategies:
    - ECN
    - RED
  - End-point strategy:
    - TCP Vegas
    - TCP Rapid
ECN: Explicit Congestion Notification

- **Basic Approach:**
  - Equally split responsibility of congestion control between routers and hosts

- **Router:**
  - Monitors the load it is experiencing
    - Average queue length, average utilization, etc
  - Explicitly notifies end hosts when congestion is about to occur
    - By setting a binary congestion bit in packets that it forwards
    - Destination hosts echo the bit in ACKs sent to the source

- **Source:**
  - Adjusts sending rate on receiving congestion notification

RED: Random Early Detection

- **Two main characteristics:**
  - Implicit notification
    - Just drop packet (end-host detects loss and infers congestion)
  - Early random drop
    - Don’t wait for queues to be full
    - Drop packets with some “drop probability” whenever queue exceeds some “drop level”

- **Is an example of an Active Queue Management (AQM) scheme**
  - Queues are monitored and managed before heavy congestion sets in
  - Other examples: PI, REM, Blue, …
Computes an average queue length:
\[ \text{AvgLen} = (1-a)\times \text{AvgLen} + a \times \text{SampleLen}; \quad \text{where } a \approx 0.002 \]
- Captures the notion of “long-lived” congestion
  - Bursty traffic could cause queue to fill up and empty again quickly

Uses two queue length thresholds:
- if \( \text{AvgLen} \leq \text{MinThresh} \)
  - enqueue the packet
- else if \( \text{AvgLen} < \text{MaxThresh} \)
  - drop arriving packet with probability \( p \)
- else drop arriving packet

**RED: Discussion**

- **Fairness of resource allocation:**
  - The probability that RED drops a packet from a given flow is proportional to the flow’s current share of bandwidth
    - Flows that are sending more traffic are more likely to be penalized if queues grow

- **How to set the \text{MinThresh} and \text{MaxThresh}?**
  - If traffic is bursty?
    - \text{MinThresh} should be set high to allow good link utilization
  - Given that sources take RTT delay to respond to first indication of congestion?
    - \((\text{MaxThresh} - \text{MinThresh})\) should be larger than typical increase in \text{AvgLen} in RTT
      - \text{MaxThresh} = 2^a \times \text{MinThresh}
    - Value of \( a \) should help filter out changes in queue length over timescales much smaller than 100 ms
**End-point Congestion Avoidance: Indicators**

- How can you detect incipient stages of congestion at end-hosts?
  - See if there’s a measurable increase in RTTs
  - See if it is correlated with increase in cwin
    
    \[
    \text{if } \frac{(curr\text{Win} - old\text{Win})}{(curr\text{RTT} - old\text{RTT})} > 0, \text{ decrease cwin; else increase cwin}
    \]

**TCP Vegas: Approach**

- Vegas uses a mixture of above ideas
  - Controls the amount of extra data that a transfer has in transit
    - Extra \(\Rightarrow\) data that would not have been transmitted if sending at exactly the network’s available bandwidth
  - Computes and maintains for the transfer:
    - \(\text{BaseRTT}\): min RTT seen for this transfer
    - \(\text{ExpectedRate} = \frac{\text{cwin}}{\text{BaseRTT}}\)
    - \(\text{ActualRate}\): # of packets sent before an ACK for the first one arrives
  - Window update (where: \(\text{diff} = \text{ExpectedRate} - \text{ActualRate}\)):
    - If \(\text{diff} < 0\), update \(\text{BaseRTT}\)
    - Else if \(\text{diff} < a\), increase \(\text{cwin}\) linearly
    - Else if \(\text{diff} > b\), decrease \(\text{cwin}\) linearly
    - Else leave \(\text{cwin}\) unchanged
    - If packet loss, decrease \(\text{cwin}\) multiplicatively

*Goal: Keep between \(a\times\text{RTT} - b\times\text{RTT}\) bytes in bottleneck queue*
TCP Vegas: Approach

- Objective: Keep between $a*RTT - b*RTT$ bytes in bottleneck queue

Rapid: The Packet-scale Paradigm

- High speed protocols
  - Still too sluggish – speed-time overhead
  - Still too un-friendly – overwhelm shared queues

- Manage speed-time dilemma
  - By shrinking congestion-control timescale
    - Higher rates “tried out” for only very small durations

- Make them friendlier
  - By reacting to high delays as well
    - “Back off” before TCP does

- Packet-scale paradigm – instantiation of above ideas
  - Faster, smaller queues, friendlier, fair(er)
  - Issues – sensitive to noise and cross-traffic burstiness
Rapid: Explicit Bandwidth Estimation

- Abc
- Abc

Rapid Feedback Loop:
1. Sender continuously sends multi-rate probe-streams:
   - Controls packet-gaps to probe for an exponentially-wide range
2. Receiver estimates available bandwidth for each p-stream:
   - By observing for increasing trends in inter-packet gaps
   - Sends AB estimate back to sender
3. Sender “acquires” estimated AB:
   - By setting the average rate of next p-stream equal to AB estimate

RAPID Congestion Control

RAPID Feedback Loop:
- No Persistent Queuing
- Quick AB-search
- Simultaneous Probing for AB Increase/Decrease
Friendliness to Regular TCP Traffic: HighSpeed

HighSpeed highly intrusive to TCP traffic!

Available Bandwidth (only Tmix)

Queues with only Tmix

HighSpeed Throughput

Queues with HighSpeed+Tmix

Friendliness to Regular TCP Traffic: RAPID

RAPID uses available bandwidth efficiently!

While being friendly to low-speed TCP traffic

Available Bandwidth (only Tmix)

Queues with only Tmix

RAPID Throughput

Queues with RAPID+Tmix
Delay-based Congestion Control: Concerns

- Is it efficient?
  - Will it react to transient queues (that loss-based TCP will simply let the buffers absorb)?
  - Can the RTT signal be tainted by OS issues such as interrupt-coalescence, burst-switching, etc?
  - Will Vegas react to queuing on the reverse path?

- Is it fair?
  - How will Vegas survive in a TCP-dominated world?
    - Would it get a fair share of bandwidth against competing TCP transfers?

- How will it survive in wireless environments?
  - Where several sources of random delays exist
    - Medium access times
    - Collision-induced exponential retransmissions
    - Environment-based rate-adaptation

Can Noise Be Addressed?

- What causes noise?
  - Short-scale “traffic bursts” at bottleneck queue
  - Temporary queuing at non-bottleneck queues
    - You want to smooth one and ignore other

- What is “short-scale”, anyways?
  - What type of bursts do you not want to ignore?

- How to eliminate noise?
  - Smoothing?
    - Over what timescale?
    - Is a fixed timescale going to work?
  - Buffering-aware smoothing strategy (BASS):
    - Identify start and end of short-scale “buffering event”
    - Smooth within each event