Re-engineering TCP Vegas

Neal Cardwell

Boris Bak

Building a Better TCP

- In deciding when to send, TCP errs on the side of simplicity
  - Doesn't find the right rate
  - Sends in bursts
- Some more aggressive approaches:
  - TCP Vegas: finding the right rate
  - Pacing: smoothing the bursts
- Old ideas, but not deployed
  - Don't fully understand benefits, costs

Overview

- Congestion control review
- The current approach & its problems
- An alternative: TCP Vegas
- Implementing Vegas in the real world:
  - The devil is in the details
  - Problems and patches

Congestion Control: The Problem

- How fast should TCP send?
- Assumptions:
  - IP's best-effort service model
  - Primary feedback:
    - ACKs from receiver
    - Packet loss
  - No information from routers
  - Perhaps info from past or current flows along the same path

The Sub-Problems

- Finding the right rate at startup
- Staying at the right rate
- Reacting to changes:
  - Path or traffic changes
  - More or less bandwidth available

TCP Reno

- cwnd: bounds un-ACKed packets
- ssthresh: guess as to safe cwnd
- By default, probe for more bandwidth
  - slow start: cwnd *= 1.5 each RTT
  - congestion avoidance: cwnd+=.5 each RTT
- Packet loss signals congestion
  - Fast retransmit, fast recovery: cwnd/=2
  - Retransmission timeout: cwnd=1

Problems with TCP Reno

- During slow start
  - Underutilizes and then swamps path
- No "right rate": cwnd traces a sawtooth
  - Underutilizes path
  - Increases queuing delay
  - Causes loss, reducing throughput
  - Inherently biased against long RTTs

Alternatives: Startup

- Original [Jacobson 88]: always cwnd*=1.5
- Several studied in [Allman, Paxson 99]
  - Tracking slow start flights [BP95]
  - Closely-spaced ACKs [Hoe 96]
  - Tracking closely-spaced ACKs [AD98]
  - Receiver-side estimation [AP99]
- But no studies w/ typical paths

Alternatives: Steady-State

- Original [Jacobson 88]: always cwnd++
- CARD [Jain 89]: RTT
- Tri-S [Wang, Crowcroft 91]: rate
- DUAL [Wang,Crowcroft 92]: RTT
- TCP Vegas [Brakmo,Peterson 94]: rate
- Buffer-Fill Avoidance [Awadallah, Rai 98]: RTT
- New, from UCB (Mo, Walrand 99): RTT, rate

TCP Vegas

- In congestion avoidance
  - cwnd = (actual rate)x(baseRTT) + 2 pkts
  - Each RTT, tweak cwnd by 1 pkt if needed
- During slow start
  - To reduce overshoot, increase cwnd only every other RTT
  - Exit slow start when
    - cwnd > (actual rate)x(baseRTT) + 1 pkt

Vegas in Theory and Practice

- In isolation, 3-70% higher BW than Reno
- Loses to Reno b/c it queues few packets
- In steady state, 2 pkts queued per flow
  - No loss
- Stable
- Basically fair w.r.t. other Vegas flows
  - No RTT bias, unlike Reno
  - Slight bias toward newer connections

Our Experience

- Plan: implement Vegas for Linux
- 2 months fixing Linux TCP bugs
- Found problems with Vegas
  - Some fixed, some are works in progress...
- This congestion control business is tricky stuff...

Time Stamp Granularity

- Arizona Vegas used 1ms
- Linux has 10ms-granularity time in kernel
- Nice for smoothing out noise? Nope!
  - RTT to UCB goes from 2 ticks to 3 - whoa!
- Need resolution much smaller than typical RTT to avoid granularity artifacts
- We now use microsecond resolution

Slow Start Problems

- Vegas slow start is too slow...
  - Increase by 1.5x every other RTT
  - Most flows are short, so...ouch!
- But still eventually overshoots…
  - Suppose cwnd of W = bottleneck BW
  - It will send 1.5*W for two RTTs before slowing down
- Our implementation increases every RTT
  - Need a better heuristic for exiting slow start

Delayed ACKs

- When receiver's delayed ACK timer fires:
  - Causes ~100ms delay that looks just like queuing
  - Artificially small actual rate
  - Can compensate by min filtering
- When receiver ACKs every other packet:
  - Sender sends burst of two packets
  - Only get RTT sample for 2nd packet of bursts
  - But this packet was queued behind the 1st…
  - Can compensate by allowing >1 packets queued

Idleness Bug

- Arizona Vegas assumes flow never idle
  - thus always has recent actual rate info
- On restart from idle, must wait until we get fresh info before making a decision
- Our implementation incorporates this fix

Loss Hurts Vegas More

- TCP uses duplicate ACKs to detect losses
- So big windows help loss recovery
- But Vegas tries to keep a small window
  - For my 512Kbps line, Vegas cwnd=4
  - If Vegas loses 2 packets to my house, RTO
- Could compensate w/ NetReno [Lin, Kung 98]
  - Send new packet out for every duplicate ACK
  - No RTO unless all packets in flight are lost

Route Changes

- Long noted: If new route is longer, this looks like queuing delay
- Vegas slows when it should speed up!
- Simulated proposals: choose baseRTT as min of RTTs over last N ACKs or T secs
- Our experience: tricky; depends on bandwidth, delay

Persistent Queuing

- Long noted: Vegas can't distinguish propagation delay from persistent queuing
- New flows get bigger baseRTT, cwnd
- Two approaches:
  - OK; prioritizes short flows
  - Not OK;causes unacceptable queuing delay
- Proposal: treat as route change
- We have yet to try this

Conclusions

- Hard to fine-tune and be robust to:
  - Random queuing delay
  - Delayed ACKs
  - Route changes
  - Link layer loss, retransmission, compression
- Reno may be dumb, but it's robust
- I still have high hopes for Vegas...