CSE 222
Graduate Networking

Winter 2001

Lecture 5: Congestion Control
(TCP Hack Fest)

Geoffrey M. Voelker

Overview

- Today
  - The wonderful world of TCP congestion control
  - Some lecturing, more discussion
- Just like TCP, from class to class I'm going to swing back and forth between lecturing and discussion to find the optimal class bandwidth
  - Note: TCP humor
- If you're final project is successful, who knows, you, too, can have a TCP hack named after you
  - Note: More TCP humor

The Problem

- Here's the situation:
  - You send out as many packets as your window (congestion/threshold/receive) allows
  - You get some ACKs back
    - » If you're lucky, they look like VJ88
    - » If you're on the Internet, bad stuff happens
  - Given a sequence of ACKs coming back, what do you do?
    - » Want to retransmit drops to keep the pipe full
    - » Don't want to retransmit too much (overfill pipe, duplicate data)

TCP Mods

- Nagle (84)
- Karn/Partridge (87)
- Congestion Avoidance (VJ88)
- Slow-Start (88)
- Fast Retransmit (88)
- Fast Recovery (90)
- Congestion Avoidance (Vegas95)
- Partial ACKs (96)
- SACK (96)

TCP Taxonomy

- TCP Tahoe
  - Slow-Start, Fast Retransmit, Congestion Avoidance
- TCP Reno
  - Tahoe + Fast Recovery
- TCP New-Reno
  - Reno + Hoe's partial ACK change that keeps TCP in Fast Recovery
- TCP Vegas
  - Real congestion avoidance
- SACK TCP
  - Selective acknowledgements

Early TCP Mods

- Nagle (84)
  - Address telnet tinygrams: 40 bytes of header, 1 byte of data
  - Only one small segment unacknowledged
  - Small segment < MSS (maximum segment size)
  - When might you not want to use Nagle?
- Karn/Partridge (87)
  - Don't sample RTT on retransmissions, double timeout
  - Exponential backoff in face of potential congestion
  - Why might retransmitting faster/same rate be a bad idea?

TCP Mods (2)

- VJ88: Seminal congestion control and avoidance
  - Additive increase/multiplicative decrease
    - » Oscillate around bottleneck capacity
    - » Congestion avoidance
  - Slow-Start
    - » Probe to quickly identify bottleneck capacity
    - » Will get back to this with Vegas
  - Fast Retransmit
  - Fast Recovery

Fast Retransmit

- How to detect packet loss without waiting for timeout?
- Duplicate ACKs imply packet drop or reordering
- Retransmit before waiting for timeout
  - Tahoe: 3 dups
  - Why 3? Hint: Dup ACKs don't necessarily mean a drop
- Doesn't work if packets lost in burst
  - Losses compressed into one ACK (feature of go-back-N)

Fast Recovery

- Use duplicate ACKs to maintain ACK pacing
  - Keep pipe full while recovering from loss
  - Don't enter Slow-Start upon a loss
  - Continue to transmit when receive dup ACKs
  - Go into congestion avoidance when drops acknowledged
- Also doesn't work if packets lost in burst
  - Losses compressed into one ACK

Delayed ACKs

- In request/response programs, want to combine an ACK to a request with a response in same packet
- Wait 200ms before ACKing
- Must ACK every other packet (or packet burst)
- Must not delay duplicate ACKs
  - Why? What is the interaction with the congestion control algorithms?

Partial ACKs

- A partial ACK acknowledges only some of the packets outstanding at the start of Fast Recovery
- Should you exit Fast Recovery and go into congestion avoidance, or continue in FR?
  - Hoe: Stay in Fast Recovery, retransmit packet after ACK
  - When all packets in window when FR was initiated have been ACKed, go into congestion avoidance

TCP Cross Talk

- Given all of the different versions of TCP, chances are that you're talking to a different TCP implementation
- Does this matter?
  - For correctness?
  - For performance?

SACK TCP

- When are selective acknowledgements useful?
  - Problem: When you receive a duplicate acknowledgement signaling a potential loss, which packets do you retransmit?
- TCP is go-back-N
  - Could retransmit just one (presumed dropped)
  - Could retransmit multiple in window
- SACK
  - Instead of guessing, have receiver tell the sender what to retransmit
  - Rate at which packets retransmitted determined by Reno, but SACK knows to retransmit drops
    - » Reno will transmit new data instead of additional dropped packets

Tahoe vs. Reno

- Tahoe
  - Fast Retransmit and Slow-Start
  - Congestion window goes to 1
- Reno
  - Use Fast Recovery to retransmit packets in lost window
  - ½ congest window, dup ACKs clock transmits, no Slow-Start
- When is it better to do Tahoe vs. Reno?
  - Depends on number of drops in window

Reno and Multiple Drops

- Multiple drops in Reno can cause TCP to wait for a timeout
  - Much worse than Tahoe
- Successive waves of dup ACKs
  - Dups cause TCP to enter Fast Retransmit/Fast Recovery
  - Partial ACKs drop TCP out of Fast Recovery
  - Additional dup ACKs make it re-enter Fast Recovery
  - Each time with congestion window halved until TCP is throttled
    - » Window is too small to initiate new transmissions
    - » No new transmissions -> no ACK clocking -> wait for timeout

SACK Practicalities

- New option to TCP packet format
- Requires changes to both sender and receiver
- Implemented in Win98/NT4something (?)
- What about Dump Receiver design philosophy?

Experimental Methodology

- SACK methodology is great
  - Simulation
  - Replay using real Internet traces
- What is one performance question that hasn't been answered directly?
  - Hint: Think end-to-end
- Can it be answered from graphs in the paper?

Vegas

- You know your graphs are too complicated when you need an appendix to describe how to read them
- Vegas has three components
  - New Retransmission
  - New Congestion Avoidance
  - New Slow-Start
- Vegas theme
  - Instead of adding complexity to deal with drops (Fast Retransmit, Fast Recover, Partial ACKs, SACK…)
  - Add complexity to *prevent* drops

Vegas Retransmission

- Don't wait for multiple duplicate ACKs to initiate retransmission
- Instead, use duplicate ACK arrival to check to see if segment has "timed out"
- How does this differ from Reno?

Vegas Congestion Avoidance

- Reno
  - Increase congestion window until a loss is triggered
  - Go into Fast Retransmit/Recovery, repeat
- Problem
  - Wasteful: Bursty, extra packets cause congestion for other flows, losses during slow-start, harder to keep pipe full
- Solution
  - Estimate available b/w, measure available b/w, adjust
  - Tweak: available b/w is link b/w + router buffers
  - Why include router buffer?

Initial Congestion Estimate

- Slow-Start
  - Start at 1, exponential increase until you get a drop
  - Oversubscribes bottleneck and router buffers
- Vegas
  - Exponential growth only every other RTT (stabilize b/w est.)
- Vegas*
  - Increase congest win only if b/w estimate says its OK
- What would be another approach for estimating available bandwidth at connection startup?
  - Hint: Think out of the box

Short Connections

- How do short connections and Slow-Start interact?
  - What happens when there is a drop in Slow-Start?
  - What happens when the SYN is dropped?
- Bottom line: Which packet gets dropped matters a lot
  - Sync
  - Slow-Start
  - Congestion avoidance
- Do you think most flows are short or long?
  - What's the current most popular application?
  - What were the most popular applications when Slow-Start was developed?

Router Buffers

- What affect does adding more buffers to routers have on TCP performance?
  - Buffering absorbs bursts – how does this affect drop rate?
  - Absorbtion makes slow-start over-estimate (Vegas) – how does this affect drop rate?
  - How does buffering affect RTT?

Further Discussion

- Additional questions, comments, clarifications about the SACK and Vegas papers?
- Why not use Vegas?
  - Very controversial

For Next Time...

- Read Chapters 6.2 and 6.4
- Read Floyd93, Floyd99