Network Transport Layer:
TCP/Reno Analysis, TCP Cubic, TCP/Vegas

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

11/07/2017
Admin.

- Programming assignment 4 Part 1 design review meet
  - 2:30-3:00 pm today
  - 4:00-5:00 pm today (change from 6 to 5 pm)
  - 10:00-11:00 am tomorrow

- Exam 2 date or mini project?
Recap: Transport Design

- Basic structure/reliability: sliding window protocols
- Determine the “right” parameters
  - Timeout
    - mean + variation
  - Sliding window size
    - Related w/ congestion control or more generally resource allocation
      - Bad congestion control can lead to congestion collapse (e.g., zombie packets)
    - Goals: distributed algorithm to achieve fairness and efficiency
congestion

\[ x_i(t+1) = \begin{cases} 
  a_t + b_t x_i(t) & \text{if } d(t) = \text{no cong.} \\
  a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases} \]
no-congestion

\[ x_i(t+1) = \begin{cases} 
  a_i + b_i x_i(t) & \text{if } d(t) = \text{no cong.} \\
  a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} 
\end{cases} \]
Recap: Realizing A(M)IMD: TCP/Reno

Initially:
   cwnd = 1;
   ssthresh = infinite (e.g., 64K);

For each newly ACKed segment:
   if (cwnd < ssthresh)  // slow start: MI
      cwnd = cwnd + 1;
   else
      // congestion avoidance; AI
      cwnd += 1/cwnd;

Triple-duplicate ACKs:
   // MD
   cwnd = ssthresh = cwnd/2;

Timeout:
   ssthresh = cwnd/2;  // reset
   cwnd = 1;

(if already timed out, double timeout value; this is called exponential backoff)
Recap: TCP/Reno: Big Picture

TD: Triple duplicate acknowledgements
TO: Timeout
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
    - design
    - analysis
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
    - design
    - analysis
Objective

- To understand
  - the throughput of TCP/Reno as a function of RTT (RTT), loss rate (p) and packet size
  - the underlying queue dynamics

- We will analyze TCP/Reno under two different setups
TCP/Reno Throughput Analysis

- Given mean packet loss rate $p$, mean round-trip time RTT, packet size $S$
- Consider only the congestion avoidance mode (long flows such as large files)
- Assume no timeout
- Assume mean window size is $W_m$ segments, each with $S$ bytes sent in one RTT:

$$\text{Throughput} = \frac{W_m \times S}{\text{RTT}} \text{ bytes/sec}$$
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
    - design
    - analysis
      - small fish in a big pond
      - loss rate given from the environment
TCP/Reno Throughput Modeling

\[
\Delta W = \begin{cases} 
\frac{1}{W} & \text{if the packet is not lost} \\
-\frac{W}{2} & \text{if packet is lost}
\end{cases}
\]

mean of \( \Delta W = (1 - p) \frac{1}{W} + p(-\frac{W}{2}) = 0 \)

\[\Rightarrow \text{mean of } W = \sqrt{\frac{2(1-p)}{p}} \approx \frac{1.4}{\sqrt{p}}, \text{ when } p \text{ is small}\]

\[\Rightarrow \text{throughput} \approx \frac{1.4S}{RTT\sqrt{p}}, \text{ when } p \text{ is small}\]

This is called the TCP throughput sqrt of loss rate law.
Exercise: Application of Analysis

State of art network link can reach 100 Gbps. Assume packet size 1250 bytes, RTT 100 ms, what is the highest packet loss rate to still reach 100 Gbps?

tcp-reno-tput.xlsx
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
    - design
    - analysis
      - small fish in a big pond
      - big fish in small pond
      - growth causes losses
TCP/Reno Throughput Modeling: Relating $W$ with Loss Rate $p$

Total packets sent per cycle = \((W/2 + W)/2 \times W/2 = 3W^2/8\)
Assume one loss per cycle => $p = 1/(3W^2/8) = 8/(3W^2)$

\[
W = \frac{\sqrt{8/3}}{\sqrt{p}} = \frac{1.6}{\sqrt{p}}
\]

\[
\Rightarrow \text{throughput} = \frac{S}{RTT} \times 3 \times \frac{1.6}{\sqrt{p}} = \frac{1.2S}{RTT \sqrt{p}}
\]
A Puzzle: cwnd and Rate of a TCP Session

Question: although cwnd fluctuates widely (i.e., cut to half), why can the sending rate stay relatively smooth?
TCP/Reno Queueing Dynamics

If the buffer at the bottleneck is large enough, the buffer is never empty (not idle), during the cut-to-half to “grow-back” process.

Exercise: How big should the buffer be to achieve full utilization?
Extreme: Zero Buffer

- Assume
  - BW: 10 G
  - RTT: 100 ms
  - Packet: 1250 bytes
  - BDP (full window size): 100,000 packets

- A loss can cut window size from 100,000 to 50,000 packets

- To fully grow back
  - Need 50,000 RTTs => 5000 seconds, 1.4 hours

- Q: What is the link utilization in one cycle?
Assume a generic AIMD alg:
- reduce to $\beta W$ after each loss event

Q: What value $\beta$ gives higher utilization (assume small/zero buffer)?

Q: Assume picking a high value $\beta$, how to make the alg TCP friendly?
Generic AIMD and TCP Friendliness

Total packets sent per cycle = \( \frac{\beta W + W}{2} \cdot \frac{(1-\beta)W}{\alpha} = \frac{(1-\beta)(1+\beta)}{2\alpha}W^2 \)

Assume one loss per cycle \( p = \frac{2\alpha}{(1-\beta)(1+\beta)w^2} \)

\( w = \sqrt{\frac{2\alpha}{(1-\beta)(1+\beta)p}} \)

\( \text{tput} = \frac{W_mS}{RTT} = \frac{S(1+\beta)W}{RTT} = \frac{S}{RTT} \sqrt{\frac{\alpha(1+\beta)}{2(1-\beta)p}} \)

TCP friendly => \( \alpha = 3 \frac{1-\beta}{1+\beta} \)
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
  - TCP Cubic
TCP Cubic

- Designed in 2008
- Default for Linux
- Most sockets in MAC appear to use cubic as well
  - `sw_vers`
  - `sysctl -a`
# TCP Cubic Goals

- **Improve TCP efficiency over fast, long-distance links**
  - Smaller reduction, longer stay at BDP, faster than linear increase---cubic function

- **TCP friendliness**
  - Follows TCP if TCP gives higher rate

- **Fairness of flows w/ different RTTs**
  - Window growth depends on real-time (from congestion-epoch through synchronized losses)
If (received ACK && state == cong avoid)

- Compute $W_{\text{cubic}}(t+\text{RTT})$.
- If cwnd < $W_{\text{TCP}}$
  - Cubic in TCP mode
- If cwnd < $W_{\text{max}}$
  - Cubic in concave region
- If cwnd > $W_{\text{max}}$
  - Cubic in convex region
The Cubic function

\[ W_{tcp(t)} = W_{max} \beta' + 3 \frac{1-\beta'}{1+\beta'} \frac{t}{RTT} \]

\[ \beta' = 1 - \beta \]

where \( C \) is a scaling factor, \( t \) is the elapsed time from the last window reduction, and \( \beta \) is a constant multiplication decrease factor.
(a) CUBIC window curves.

(b) Throughput of two CUBIC flows.
Outline

- Admin and recap
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/Reno congestion control
  - TCP Cubic
  - TCP/Vegas
TCP/Vegas (Brakmo & Peterson 1994)

- Idea: try to detect congestion by *delay before loss*
- Objective: not to overflow the buffer; instead, try to maintain a *constant* number of packets in the bottleneck queue
TCP/Vegas: Key Question

How to estimate the number of packets queued in the bottleneck queue?
Recall: Little’s Law

- For any system with no or (low) loss.
- Assume
  - mean arrival rate $X$, mean service time $T$, and mean number of requests in the system $W$
- Then relationship between $W$, $X$, and $T$:

$$W = XT$$
Estimating Number of Packets in the Queue
TCP/Vegas CA algorithm

\[ T = T_{\text{prop}} + T_{\text{queueing}} \]

Applying Little’s Law:

\[ x_{\text{vegas}} \cdot T = x_{\text{vegas}} \cdot T_{\text{prop}} + x_{\text{vegas}} \cdot T_{\text{queueing}}, \]

where \( x_{\text{vegas}} = \frac{W}{T} \) is the sending rate.

Then number of packets in the queue is

\[ x_{\text{vegas}} \cdot T_{\text{queueing}} = x_{\text{vegas}} \cdot T - x_{\text{vegas}} \cdot T_{\text{prop}} \]

\[ = W - \frac{W}{T} \cdot T_{\text{prop}} \]
TCP/Vegas CA algorithm

Maintain a constant number of packets in the bottleneck buffer.

For every RTT:

\{  
  \text{if } W - \frac{W}{RTT} \cdot RTT_{\text{min}} < \alpha \text{ then } W++ \\
  \text{if } W - \frac{W}{RTT} \cdot RTT_{\text{min}} > \alpha \text{ then } W-- 
\}\n
For every loss:

\[ w := \frac{w}{2} \]
Discussions

- If two flows, one TCP Vegas and one TCP reno run together, how may bandwidth be partitioned among them?

- Issues that limit Vegas deployment?