Proportional Rate Reduction for TCP

draft-ietf-tcpm-proportional-rate-reduction-00.txt
IETF 82
16-Nov-2011

Matt Mathis, Nandita Dukkipati, Yuchung Cheng
We want to improve TCP recovery

- Traces frequently show avoidable timeouts
  - TCP misses "obvious" opportunities to transmit
- Current implementation based in part on my prior work
  - Rate-Halving w/ bounding parameters
    - Send data on alternate ACKs during recovery
    - Incomplete ID and web pages from 1998
  - We abandoned it due to unsolved corner cases
  - Philosophy was to aim for cwnd=(FlightSize-losses)/2
    - Too conservative
    - Application stalls are treated like losses
  - Hard wired 50% cwnd reduction, even if CC does not
    - e.g. CUBIC uses only a 30% reduction
Standard TCP fast recovery (RFC3517)

FlightSize: outstanding (original) packets
pipe: estimated packets in the network

Entering recovery:
   cwnd = ssthresh
   sh = FlightSize/2
   retransmit_first_loss()

For every ACK during recovery:
   pipe = update_scoreboard()
   if cwnd > pipe
       transmit(cwnd - pipe)

Issues
- Half-RTT silence under light losses
- May (re)transmit large bursts under heavy losses
- Pipe can be wrong in the presence of reordering
Working from first principles

- Strictly packet conserving:
  - Arriving data triggers equal transmissions
  - Sender computes DeliveredData on each ACK
    - Well defined and robust even with reordering
    - Use DeliveredData as the recovery clock
    - Adjusted +/-1 to track cwnd/ssthresh
- Want recovery rate to be proportional to CC change
- Want final window to be chosen by CC
  - As it is with RFC 3517
  - Losses delay transmissions, but final window is the same
  - If losses exceed CC change, what action?
When losses exceed CC reduction

Three choices:

● Conservative bound (akin to rate halving)
  ○ Follow strict packet conservation during recovery
  ○ Window too small at the end of recovery
  ○ Slowstart after recovery

● Unlimited bound (follows 3517)
  ○ Allow full (ssthresh-pipe) bursts during recovery

● Slowstart bound
  ○ Relax conservative bound by 1 segment per ACK
  ○ Same total number of transmissions as 3517, but not in bursts
PRR with slowstart bound

Start of recovery:
\[ \text{ssthresh} = \text{CongCtrlAlg()} \]  // Target cwnd after recovery.
\[ \text{RecoverFS} = \text{snd.nxt} - \text{snd.una} \]  // FlightSize.
\[ \text{prr\_delivered} = \text{prr\_out} = 0 \]  // Accounting.

On each ACK in recovery, compute:
// DeliveredData: #pkts newly delivered to receiver.
\[ \text{DeliveredData} = \text{delta(snd.una)} + \text{delta(SACKd)} \]
// Total pkts delivered in recovery.
\[ \text{prr\_delivered} += \text{DeliveredData} \]
\[ \text{pipe} = \text{RFC 3517 pipe algorithm} \]

Algorithm:
\[
\begin{align*}
\text{if (pipe} & > \text{ssthresh)} \quad // \text{PRR.} \\
\quad & \quad \quad \text{sndcnt} = \text{CEIL(prr\_delivered} \times \text{ssthresh} \div \text{RecoverFS}) - \text{prr\_out} \\
\text{else} & \quad // \text{Slow start.} \\
\quad & \quad \quad \text{limit} = \text{max(prr\_delivered} - \text{prr\_out, DeliveredData} + 1 \\
\quad & \quad \quad \text{sndcnt} = \text{MIN(ssthresh} - \text{pipe, limit})
\end{align*}
\]
On any data transmission or retransmission:
\[ \text{prr\_out} += \text{(data sent)} \]
PRR properties

- Better (ACK) clocking
  - fewer timeouts
  - more accurate fast recovery in spite of reordering, stretch acks, etc
  - smoother transmissions during recovery
- Cwnd converges to ssthresh
  - Not effected by additional loss or application stalls
PRR results

• Performs better than Rate Halving
  ○ Avoids excess window reductions
  ○ 3-10% better transaction response times
• Performs better than 3517
  ○ Avoids consequences of sending bursts
    (45% loss episodes cause pipe <= ssthresh)
    ■ Fewer lost retransmissions
    ■ Fewer timeouts
• See full results in IMC11 paper (slides attached)
New results for youtube in India

- Similar configuration as the Web experiment
- 3 days in DC$_{youtube-India}$
- Average video response is 2.3MB

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>Standard</th>
<th>PRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retransmission rate</td>
<td>5.0%</td>
<td>6.6%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Retransmission lost</td>
<td>2.4%</td>
<td>16.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>Slow start after recovery</td>
<td>56%</td>
<td>1%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Standard TCP may cause high lost retransmission. PRR strikes the balance.
Onward

- Results are overwhelmingly good
- No substantiated downsides
- Already staged to Linux upstream
Post script: Total TCP retransmissions in two Google data centers

15.2% USA retransmissions are for connections that NEVER recover! WHAT IS GOING ON?
IMC11 presentation

● Below is Yuchung's full presentation to IMC11 (Internet Measurement Conference)
Proportional Rate Reduction for TCP

A fast and smooth loss recovery

Nandita Dukkipati, Matt Mathis, Yuchung Cheng, Monia Ghobadi
Losses hurt Web latency bad

Google HTTP responses. 6.1% experience losses.
How does TCP recover from losses?

TCP retransmission breakdown in two Google DCs. Over 96% connections support SACK.
Standard TCP fast recovery (RFC3517)

FlightSize: outstanding (original) packets  
pipe: estimated packets in the network

**Entering recovery:**
- cwnd = ssthreş
- sh = FlightSize/2
- retransmit_first_loss()

**For every ACK during recovery:**
- pipe = update_scoreboard()
- if cwnd > pipe
  - transmit(cwnd - pipe)

**Issues**
- Half-RTT silence under light losses
- May (re)transmit large burst under heavy losses
Linux TCP fast recovery

- Rate-halving: send one packet every other ACK
  - Too conservative under heavy losses

- cwnd moderation: $cwnd = \text{pipe} + 1$ exiting recovery
  - Often slow start w/ $cwnd == 2$

Courtesy of "Application Flow Control in YouTube Video Streams", CCR, Apr., 2011
Proportional rate reduction (PRR)

Design principles

● VJ's packet conservation principle

● Decouples loss detection and window adjustment
  ○ Loss detection
    ■ *dupack_thresh*, *FACK*, *lost-retrans*, etc.
  ○ Window adjustment
    ■ Gradually reduces *cwnd* across acks
    ■ *pipe converges to ssthresh*
    ■ Works with different congestion controls
Proportional Rate Reduction (PRR)

Entering recovery: \[ P = \frac{ssthresh}{cwnd} \]

For every ACK received:
- \( \text{pipe} > \text{ssthresh} \)
  - Reduce \( cwnd \) every \( P \) packets delivered
  - Transmit rate = \( P \times \text{delivery\_rate} \)
- \( \text{pipe} \leq \text{ssthresh} \)
  - Slow start to bring pipe to ssthresh
PRR properties

● Maintain ACK clocking

● Adjust cwnd by data delivered
  ○ More robust against reordering, stretched acks, loss detection errors, esp. with SACK

● cwnd converges to ssthresh after recovery

● Bank sending opportunities during application stalls
Google Web server experiment in US

● Experiment
  ○ Linux 2.6 with FACK, Cubic
  ○ Split servers in 3 groups: Standard, Linux, PRR
  ○ 5 days in DC\textsubscript{web-usa}

● PRR
  ○ 45% fast recoveries start with pipe $\leq$ ssthresh
  ○ Reduce average TCP latency by 3-10% vs. Linux
Youtube experiment in India

- Similar configuration as the Web experiment
- 3 days in DC_{youtube-India}
- Average video response is 2.3MB

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>Standard</th>
<th>PRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retransmission rate</td>
<td>5.0%</td>
<td>6.6%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Retransmission lost</td>
<td>2.4%</td>
<td>16.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>Slow start after recovery</td>
<td>56%</td>
<td>1%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Standard TCP may cause high lost retransmission. PRR strikes the balance.
Early retransmit (RFC 5827)

- $\text{dupack\_thresh} = 1$ or $2$ if $\text{FlightSize} = 2$ or $3$
  - Increase fast retransmit by $13\%$
  - $24\%$ are spurious due to (small) network reordering
- Mitigation
  - Stop if reordering $> 3$
  - Delay RTT/4 before early retransmit
  - Reduce spurious retransmission rate to $6\%$

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Linux</th>
<th>ER w/ mitigation</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>10th</td>
<td>319 ms</td>
<td>301 ms</td>
<td>-5.6%</td>
</tr>
<tr>
<td>50th</td>
<td>1084 ms</td>
<td>997 ms</td>
<td>-8.0%</td>
</tr>
<tr>
<td>90th</td>
<td>4223 ms</td>
<td>4084 ms</td>
<td>-3.3%</td>
</tr>
</tbody>
</table>

TCP latency of all responses except ones that has < 2 packets or do not experience losses
Conclusion

- Packet losses significantly increase Web latency

- PRR is a new TCP fast recovery algorithm
  - Recovers quickly and smoothly
  - Adopted by Linux upstream :-)
  - IETF RFC in progress

- Early retransmit (ER)
  - Useful but needs to mitigate reordering
  - Both PRR and ER are being deployed on all Google servers

- Ongoing efforts
  - Timeout recovery, mobile TCP, TCP Fast Open, TCP/video
PRR full algorithm

Start of recovery:
\[
\text{ssthresh} = \text{CongCtrlAlg()} \quad // \text{Target cwnd after recovery.}
\text{RecoverFS} = \text{snd.nxt} - \text{snd.una} \quad // \text{FlightSize.}
\text{prr\_delivered} = \text{prr\_out} = 0 \quad // \text{Accounting.}
\]

On each ACK in recovery, compute:
// DeliveredData: #pkts newly delivered to receiver.
\[
\text{DeliveredData} = \text{\text{delta}(snd.una)} + \text{\text{delta}(SACKd)}
\]
// Total pkts delivered in recovery.
\[
\text{prr\_delivered} += \text{\text{DeliveredData}}
\]
\[
\text{pipe} = \text{RFC 3517 pipe algorithm}
\]

Algorithm:
if (pipe > ssthresh) // PRR.
\[
\text{sndcnt} = \text{CEIL(prr\_delivered * ssthresh / RecoverFS)} - \text{prr\_out}
\]
else // Slow start.
\[
\text{ss\_limit} = \text{max(prr\_delivered - prr\_out, DeliveredData)} + 1
\]
\[
\text{sndcnt} = \text{MIN(ssthresh - pipe, ss\_limit)}
\]

On any data transmission or retransmission:
\[
\text{prr\_out} += \text{(data sent)}
\]