Proportional Rate Reduction for TCP

draft-mathis-tcpm-proportional-rate-reduction-00

Matt Mathis, Nandita Dukkipati, Yuchung Cheng
{mattmathis, nanditad, ycheng}@google.com

TCPM, IETF-80 Prague
Motivation

- Two widely deployed algorithms in loss recovery: RFC 3517 fast recovery and Rate halving.
- There are cases when both are prone to timeouts.
  - RFC 3517 fast recovery waits for half of ACKs to pass before sending data.
  - Rate halving does not compensate for implicit cwnd reduction under large losses.
- Goals of Proportional Rate Reduction.
  - Reduce timeouts by avoiding excessive window reductions.
  - Converge to cwnd chosen by congestion control by the end of recovery.
Proportional Rate Reduction with Reduction Bound (PRR-RB)

- PRR-RB has two algorithms.
- Proportional rate reduction (PRR):
  - Patterned after rate halving but with a factor that depends on congestion control reduction.
  - Main idea: sending rate = r * queue drain rate in recovery; r is CC reduction factor.
  - For precision, queue drain rate is computed as amount of newly delivered data to receiver.
- Reduction bound (RB):
  - Main purpose: inhibit further cwnd reductions under large losses (pipe < ssthresh).
Algorithm:

if (pipe > ssthresh) // Proportional Rate Reduction.
    sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
else // Reduction Bound.
    sndcnt = MIN(ssthresh - pipe, prr_delivered - prr_out)

On any data transmission or retransmission:
prr_out += (data sent) // Smaller than or equal to sndcnt.
Properties of PRR-RB

- Spreads out window reduction evenly across the recovery period.
- Converges to target cwnd chosen by CC for minimal losses.
- Maintains ACK clocking even for large burst losses.
- Precision of PRR-RB is derived from computing exact segments delivered to the receiver during recovery.
- On every ACK, cumulative data sent during recovery is bound by the cumulative data delivered to the receiver during recovery.
- Banks the missed opportunities to send if application stalls during recovery.
- Less sensitive to errors of the pipe estimator.
Proportional Rate Reduction without Reduction Bound (PRR w/o RB)

- Observation: there are cases where RFC 3517 sends more than the reduction bound of PRR-RB.
  - Example: pipe == cwnd == 100 packets, lost packets 1 - 90, packet 93 can generate a burst up to 40 packets.
  - RFC 3517 is not conservative in this scenario.
  - PRR-RB bounds #pkts sent by (prr_delivered - prr_out).

- PRR w/o RB: cwnd is bound below only by ssthresh.

```plaintext
if (pipe > ssthresh) // Proportional Rate Reduction
   sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
else
   sndcnt = ssthresh - pipe
```
Example flow in PRR-RB, PRR w/o RB

Ratio of pkts sent to delivered is 0.7.

Different from PRR-RB only for large burst losses (not shown here).
Example flow in Linux and RFC 3517

Linux 2.6.34 recovery

Rate-halving: send one new pkt on alternate ACKs.

\[ \text{cwnd} = \text{pipe} + 1 \]

RFC 3517 fast recovery

Silent period at start of recovery.

Other recovery events.

Linux flows often end up in slow start after recovery when short responses have no new data to send and pipe reduces to 0.
Performance

Two kinds of experiments: Google Web server, datacenter traffic.

- PRR w/o RB reduces timeouts in recovery.
- Reduces tail latency of 1MB RPCs by 35% (w.r.t. RFC) and 16% (w.r.t Linux).

<table>
<thead>
<tr>
<th>Event Type</th>
<th>PRR w/o RB</th>
<th>RFC 3517</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastRecovery events</td>
<td>1,759</td>
<td>2,755</td>
<td>1,738</td>
</tr>
<tr>
<td>TotRetrans</td>
<td>8,471</td>
<td>15,236</td>
<td>8,282</td>
</tr>
<tr>
<td>FastRetrans</td>
<td>8,371</td>
<td>14,881</td>
<td>8,158</td>
</tr>
<tr>
<td>RegularFastRetrans</td>
<td>7,916</td>
<td>13,202</td>
<td>7,971</td>
</tr>
<tr>
<td>ForwardFastRetrans</td>
<td>455</td>
<td>1,679</td>
<td>187</td>
</tr>
<tr>
<td>TimeoutRetrans</td>
<td>41</td>
<td>177</td>
<td>50</td>
</tr>
<tr>
<td>TimeoutOnOpen</td>
<td>7</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>TimeoutOnRecovery</td>
<td>18</td>
<td>148</td>
<td>20</td>
</tr>
<tr>
<td>TimeoutOnDisorder</td>
<td>16</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>99% latency of 1MB RPCs (us)</td>
<td>15,064</td>
<td>23,105</td>
<td>17,956</td>
</tr>
</tbody>
</table>

Data in this spreadsheet: [http://goo.gl/xl6cF](http://goo.gl/xl6cF)
Benefits and costs

• Benefits:
  o Smaller number of timeouts.
  o In some cases, smaller number of recovery events.
  o Lower tail latencies of request-response traffic.

• Cost:
  o Need to investigate if there are indirect costs to sending packets early on in recovery as opposed to letting more ACKs pass first.
Going forward

- Should the PRR draft be a WG item?
  - Design of PRR improves upon the current standard in reducing timeouts.
  - Initial experiments demonstrate promising results.
  - Goal: adopt as experimental so people can gain experience.