Proportional Rate Reduction for TCP

draft-ietf-tcpm-prr-rfc6937bis-13

IETF 122 tcpm Mar 2025

Matt Mathis <ietf@mattmathis.net>

Nandita Dukkipati <nanditad@google.com>

Yuchung Cheng <ycheng@google.com>

Neal Cardwell <ncardwell@google.com> [presenting]

RFC 6937: Proportional Rate Reduction for TCP

- Published in 2013 as an experimental RFC
  - At the time, only implemented by Linux; implemented without RFC 8985 (RACK-TLP)

- A "**mini** congestion control" during fast recovery
  - Send packets at the ratio of CC's cwnd reduction
    - Reno: 0.5, Cubic: 0.7
  - If in-flight data drops below ssthresh
    - Mode SSRB: slow-start
    - Mode CRB: packet conservation
  - Implementation has to pick SSRB or CRB option

- Interesting fact: for a flow that mostly operates in fast recovery, its cwnd is mostly controlled by RFC 6937 instead of Cubic/CC
  - e.g. policed video streaming flows using Cubic

RFC 6937-bis

10 years after RFC 6937, in 2021

- PRR is default-enabled in Linux, FreeBSD, NetFlix-BSD/RACK, MS Windows TCP
- tcpm voted to revise and publish as a standard RFC

Current draft is: [draft-ietf-tcpm-prr-rfc6937bis]

Revisions since RFC 6937 that were previously described at IETF meetings:

- Algorithm refinements
  - Merge SSRB and CRB modes: automatically choose the mode based on if the last ACK indicates further losses // bis-01
  - Force a fast retransmit on entering recovery to maintain ACK clock // bis-02
  - Non-SACK support // bis-03
  - Streamline the sending process if experienced higher network reordering previously // bis-04

- Editorial clarifications
  - Do not slow start on ACKs that trigger RFC 6675 “last resort” retransmission as it may indicate further losses
  - Relationships with RFC 6675 (pipe), RFC 8985 (RACK-TLP)
  - Removed experiments section

RFC 6937-bis: changes between 12 and 13

Most changes were in response to comments from Markku Kojo on TCPM list (many thanks for the comments!)

Here are the specific changes since the last PRR presentation, IETF 121 in Nov 2024, rev 12 [rev 12 slides]:

  text diff between version 12 and version 13

The following slides summarize:

  The main substantive differences between revisions 12 and 13:

    1: "pipe" => "inflight"

Previous use of "pipe" in PRRbis rev 12 and earlier

- Raised in tcpm email from Markku Kojo, Oct 29, 2024
- Markku:
    - (3) ["pipe" and current loss detection algorithm]: 'Shouldn't the "pipe" estimate in the algo be based on the loss detection algorithm in use?'
    - (4) ["pipe" and current loss detection algorithm for non-SACK connections]: 'The same holds for non-SACK fast recovery, that is NewReno. I think the document/algo should clarify that, without SACK, the SACKed segments for pipe calculation are estimated in the similar way as they are estimated or DeliveredData (i.e., one SACKd segment = one duplicate ACK).'

Rationale for "pipe" => "inflight"

- Rather than "pipe", we changed the PRRbis draft to use "inflight" since:
  - (a) "inflight" is the term used for this concept in existing CC specs/code:
    - Linux TCP
    - BBR (CACM article and draft-ietf-ccwg-bbr)
    - RACK-TLP [RFC8985] (in passing)
    - draft-briscoe-iccrg-prague-congestion-control: "`inflight:` `The amount of data that the sender has sent but not yet received ACKs for`"
  - (b) there should not be confusion because "inflight" is not used anywhere currently in any of:
    - PRR (RFC6937 or PRRbis)
    - RFC6675
    - Reno/RFC5681
    - CUBIC/RFC9438

Changes for "pipe" => "inflight"

- 1: In Sec 5, "Definitions", we added a definition for "inflight":

  ```
  inflight: The data sender's best estimate of the number of bytes outstanding in the network. To calculate
  inflight, connections with SACK enabled and using [RFC6675] loss detection MAY use the "pipe" algorithm as
  specified in [RFC6675]. SACK-enabled connections using RACK-TLP loss detection [RFC8985] or other loss
  detection algorithms MUST calculate inflight by starting with SND.NXT - SND.UNA, subtracting out bytes
  SACKed in the scoreboard, subtracting out bytes marked lost in the scoreboard, and adding bytes in the
  scoreboard that have been retransmitted since they were last marked lost. For non-SACK-enabled
  connections, instead of subtracting out bytes SACKed in the SACK scoreboard, senders MUST subtract out:
  min(RecoverFS, 1 SMSS for each preceding duplicate ACK in the fast recovery episode); the min() with
  RecoverFS is to protect against misbehaving receivers [Savage99].
  ```

  - Note: this matches the computation of "inflight" implemented in Linux TCP

- 2: Then we replaced phrases w/ "pipe" in rev 12 with analogous phrases w/ "inflight" in rev 13
  - In the prose, pseudocode, and figures

[Full diff between rev 12 and 13]()

Conclusion

- Are we done yet? :-)