Proportional Rate Reduction for TCP

draft-ietf-tcpm-prr-rfc6937bis-12

IETF 121 tcpm Nov 2024

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 10 and 12

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 120 in Jul 2024, version 10:

text diff between version 10 and version 12

The following slides summarize:

The main substantive differences between revisions 10 and 12

The open issues based on recent TCPM mailing list discussions from Oct 29 - Nov 1

Clarified that PRR can be used with any CC MD

- Previously draft only mentioned PRR is applicable to "Standard congestion control [RFC5681]"
- Clarified that PRR **MAY** be used with any CC making a multiplicative cwnd decrease to a fixed ssthresh
- Thanks to Mirja Kühlewind for the suggestion

New text in Sec 4, "Relationships to other standards":

> PRR MAY be used in conjunction with any congestion control algorithm that intends to make a multiplicative decrease in its sending rate over approximately the time scale of one round trip time, as long as the current volume of in-flight data is limited by a congestion window (cwnd) and the target volume of in-flight data during that reduction is a fixed value given by ssthresh.

Clarified that PRR can be used with Reno or CUBIC

- Previously draft only mentioned PRR is applicable to "Standard congestion control [RFC5681]"
- Removed references to "standard congestion control", which is ambiguous/unclear in 2024
- Clarified that PRR can be used with either Reno [RFC5681] or CUBIC [RFC9438]

New text in Sec 2, "Background":

```
Congestion control algorithms like Reno [RFC5681] and CUBIC [RFC9438] require that TCP (and other
protocols) reduce their congestion window (cwnd) in response to losses. ... Without PRR, fast recovery
typically adjusts the window by waiting for a large fraction of a round-trip time (one half round-trip
time of ACKs for Reno [RFC5681], or 30% of a round-trip time for CUBIC [RFC9438]) to pass before sending
any data.¶
```

New text in Sec 4, "Relationships to other standards":

```
...In particular, PRR is applicable to both Reno [RFC5681] and CUBIC [RFC9438] congestion control.  ...
```

New reference to CUBIC [RFC9438] in Sec 13, "Normative References":

```
[RFC9438] Xu, L., Ha, S., Rhee, I., Goel, V., and L. Eggert, Ed.,
          "CUBIC for Fast and Long-Distance Networks", RFC 9438,
          DOI 10.17487/RFC9438, August 2023,
          <https://www.rfc-editor.org/info/rfc9438>.
```

Open issue 1: when/how to reduce ssthresh

- Raised in tcpm email from Markku Kojo, Oct 29, 2024
- Markku:
  - (1) [when to reduce ssthresh on lost retransmits]: "I thin[k] this document should also explicitly repeat that on detecting loss of a retransmission, the TCP sender MUST lower ssthresh once per RTT (and give enough details on how to do it correctly with PRR to avoid pitfalls with flightsize and cwnd, but not saying how much to lower)."
  - (2) [how to set ssthresh on lost retransmits]: "ssthresh MUST NOT be reinitialized using flightsize or cwnd. ...Maybe on detecting loss of a rexmit, ssthresh could be reinitialized... ssthresh = multiplicative_decrease_factor * ssthresh?"

Open issue 1: proposal to address this issue

- When/how to reduce ssthresh is very tricky, and opens up a whole can of worms
- See the great IETF 120 "CC Response While Application-Limited" deck from Matt Mathis
- The issue is so thorny that all the major TCP implementations (Linux, FreeBSD, Windows, MacOS/iOS) diverge from the Reno [RFC5681] and CUBIC [RFC9438] standards, and do not set ssthresh based on FlightSize, but rather based on cwnd
- This is a deeper and wider problem best left out of the PRR draft
- The model we are advocating with PRR is a separation of concerns:
  - The congestion control algorithm decides:
    - (a) when to slow down (reduce ssthresh and/or cwnd)
    - (b) what the exact ssthresh value is as a result of the slow-down decision
  - PRR decides: given (a) and (b) decisions made by the congestion control algorithm, how to update the cwnd using each ACK

Open issue 1: proposed text

- In section 6, "Algorithm", we propose we change the existing text:

```
At the beginning of recovery, initialize the PRR state.
```

to:

```
At the beginning of a congestion control response episode initiated
by the congestion control algorithm, a TCP data sender using PRR
MUST initialize the PRR state. The timing of the start of a
congestion control response episode is entirely up to the
congestion control algorithm, and (for example) could correspond to
the start of a fast recovery episode, or a once-per-round-trip
reduction when lost retransmits or lost original transmissions are
detected after fast recovery is already in progress.
```

Open issue 2: problems with "pipe" in PRRbis

- 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)."

Open issue 2: proposal to address this issue

- Rather than "pipe", we propose that the PRRbis draft use "inflight" since:
  - (a) "inflight" is the term used for this concept in existing CC specs/code:
    - Linux TCP
    - BBR (CACM article and Internet draft)
    - 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: PRR (RFC6937 or PRRbis), RFC6675, Reno/RFC5681, or CUBIC/RFC9438.

Open issue 2: proposed text

- In Sec 5, "Definitions", we propose to add:

```
inflight: The transport connection's sender-side estimate of the number of bytes still
outstanding in the network. Connections with SACK and using RFC 6675 loss detection MAY use
"pipe" as specified in RFC 6675. For SACK-enabled connections using RACK-TLP loss detection
[RFC8985] or other loss detection algorithms, they MUST calculate inflight by starting with
SND.NXT -  SND.UNA, subtracting out bytes SACKed in the SACK scoreboard, subtracting out bytes
marked lost in the SACK scoreboard, and adding bytes in the scoreboard that have been
retransmitted since they were last marked lost. For non-SACK 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].
```

- Then we propose to replace occurrences of "pipe" in the current draft-ietf-tcpm-prr-rfc6937bis-12 with "inflight".

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

Conclusion

- Are we done yet? :-)