

TCP Maintenance Working Group
Internet-Draft
Obsoletes: 6937 (if approved)
Intended status: Standards Track
Expires: 24 December 2025

M. Mathis

N. Cardwell
Y. Cheng
N. Dukkupati
Google, Inc.
22 June 2025

Proportional Rate Reduction
draft-ietf-tcpm-prr-rfc6937bis-21

Abstract

This document specifies a standards-track version of the Proportional Rate Reduction (PRR) algorithm that obsoletes the experimental version described in RFC6937. PRR regulates the amount of data sent by TCP or other transport protocols during fast recovery. PRR accurately regulates the actual flight size through recovery such that at the end of recovery it will be as close as possible to the slow start threshold (ssthresh), as determined by the congestion control algorithm.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 December 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document.

Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Conventions	4
3. Document and WG Information	5
4. Definitions	10
5. Changes Relative to RFC 6937	12
6. Relationships to other standards	14
7. Algorithm	15
7.1. Initialization Steps	15
7.2. Per-ACK Steps	16
7.3. Per-Transmit Steps	17
7.4. Completion Steps	18
8. Properties	18
9. Examples	20
10. Adapting PRR to other transport protocols	23
11. Measurement Studies	23
12. Operational Considerations	23
12.1. Incremental Deployment	23
12.2. Fairness	23
12.3. Protecting the Network Against Excessive Queuing and Packet Loss	24
13. Acknowledgements	24
14. IANA Considerations	25
15. Security Considerations	25
16. Normative References	25
17. Informative References	26
Appendix A. Strong Packet Conservation Bound	28
Authors' Addresses	29

1. Introduction

Van Jacobson's packet conservation principle [Jacobson88] defines a self clock process wherein N data segments delivered to the receiver generate acknowledgments that the data sender uses as the clock to trigger sending another N data segments into the network.

Congestion control algorithms like Reno [RFC5681] and CUBIC [RFC9438] are built on the conceptual foundation of this self clock process. They control the sending process of a transport protocol connection by using a congestion window ("cwnd") to limit "inflight", the volume of data that a connection estimates is in-flight in the network at a

given time. Furthermore, these algorithms require that transport protocol connections reduce their cwnd in response to packet losses. Fast recovery (see [RFC5681] and [RFC6675]) is the algorithm for making this cwnd reduction using feedback from acknowledgements. Its stated goal is to maintain a sender's self clock by relying on returning ACKs during recovery to clock more data into the network. Without Proportional Rate Reduction (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.

[RFC6675] makes fast recovery with Selective Acknowledgement (SACK) [RFC2018] more accurate by computing "pipe", a sender-side estimate of the number of bytes still outstanding in the network. With [RFC6675], fast recovery is implemented by sending data as necessary on each ACK to allow pipe to rise to match ssthresh, the target window size for fast recovery, as determined by the congestion control algorithm. This protects fast recovery from timeouts in many cases where there are heavy losses. However, [RFC6675] has two significant drawbacks. First, because it makes a large multiplicative decrease in cwnd at the start of fast recovery, it can cause a timeout if the entire second half of the window of data or ACKs are lost. Second, a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a large burst of data.

PRR regulates the transmission process during fast recovery in a manner that avoids these excess window adjustments, such that transmissions progress smoothly, and at the end of recovery the actual window size will be as close as possible to ssthresh.

PRR's approach is inspired by Van Jacobson's packet conservation principle. As much as possible, PRR relies on the self clock process, and is only slightly affected by the accuracy of estimators such as the estimate of the volume of in-flight data. This is what gives the algorithm its precision in the presence of events that cause uncertainty in other estimators.

When inflight is above ssthresh, PRR reduces inflight smoothly toward ssthresh by clocking out transmissions at a rate that is in proportion to both the delivered data and ssthresh.

When inflight is less than ssthresh, PRR adaptively chooses between one of two Reduction Bounds to limit the total window reduction due to all mechanisms, including transient application stalls and the losses themselves. As a baseline, to be cautious when there may be

considerable congestion, PRR uses its Conservative Reduction Bound (PRR-CRB), which is strictly packet conserving. When recovery seems to be progressing well, PRR uses its Slow Start Reduction Bound (PRR-SSRB), which is more aggressive than PRR-CRB by at most one segment per ACK. PRR-CRB meets the Strong Packet Conservation Bound described in Appendix A; however, when used in real networks as the sole approach, it does not perform as well as the algorithm described in [RFC6675], which prove to be more aggressive in a significant number of cases. PRR-SSRB offers a compromise by allowing a connection to send one additional segment per ACK, relative to PRR-CRB, in some situations. Although PRR-SSRB is less aggressive than [RFC6675] (transmitting fewer segments or taking more time to transmit them), it outperforms due to the lower probability of additional losses during recovery.

The original definition of the packet conservation principle [Jacobson88] treated packets that are presumed to be lost (e.g., marked as candidates for retransmission) as having left the network. This idea is reflected in the inflight estimator used by PRR, but it is distinct from the Strong Packet Conservation Bound as described in Appendix A, which is defined solely on the basis of data arriving at the receiver.

This document specifies several main changes from the earlier version of PRR in [RFC6937]. First, it introduces a new adaptive heuristic that replaces a manual configuration parameter that determined how conservative PRR was when inflight was less than ssthresh (whether to use PRR-CRB or PRR-SSRB). Second, the algorithm specifies behavior for non-SACK connections (connections that have not negotiated [RFC2018] SACK support via the "SACK-permitted" option). Third, the algorithm ensures a smooth sending process even when the sender has experienced high reordering and starts loss recovery after a large amount of sequence space has been SACKed. Finally, this document also includes additional discussion about the integration of PRR with congestion control and loss detection algorithms.

PRR has extensive deployment experience in multiple TCP implementations since the first widely deployed TCP PRR implementation in 2011 [First_TCP_PRR].

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Document and WG Information

RFC Editor: please advise on how we can specify the "Janey C. Hoe" name in the "Acknowledgements" section as XML that would be correctly translated by xm2rfc into plain ASCII txt output with a single space after the "C." (interpreting the "C." as an initial) rather than a double space after the "C." (interpreting the "C." as the end of the sentence).

RFC Editor: please remove this section before publication

Formatted: 2025-06-22 19:27:52-07:00

Please send all comments, questions and feedback to tcpm@ietf.org

About revision 00:

The introduction above was drawn from draft-mathis-tcpm-rfc6937bis-00. All of the text below was copied verbatim from RFC 6937, to facilitate comparison between RFC 6937 and this document as it evolves.

About revision 01:

- * Recast the RFC 6937 introduction as background
- * Made "Changes From RFC 6937" an explicit section
- * Made Relationships to other standards more explicit
- * Added a generalized SafeACK heuristic
- * Provided hints for non TCP implementations
- * Added language about detecting ACK splitting, but have no advice on actions (yet)

About revision 02:

- * Companion RACK loss detection RECOMMENDED
- * Non-SACK accounting in the pseudo code
- * cwnd computation in the pseudo code
- * Force fast retransmit at the beginning of fast recovery
- * Remove deprecated Rate-Halving text

- * Fixed bugs in the example traces

About revision 03 and 04:

- * Clarify when and how SndCnt becomes 0
- * Improve algorithm to smooth the sending rate under higher reordering cases

About revision 05:

- * Revert the RecoverFS text and pseudocode to match the behavior in draft revision 03 and more closely match Linux TCP PRR

About revision 06:

- * Update RecoverFS to be initialized as: RecoverFS = pipe.

About revision 07:

- * Restored the revision 04 prose description for the rationale for initializing RecoverFS as: RecoverFS = pipe.
- * Added reference to [Hoe96Startup] in acknowledgements

About revision 08:

- * Inserted missing reference to [RFC9293]
- * Recategorized "voluntary window reductions" as a phrase introduced by PRR

About revision 09:

- * Document the setting of cwnd = ssthresh when the sender completes a PRR episode, based on Linux TCP PRR experience and the mailing list discussion in the TCPM mailing list thread: "draft-ietf-tcpm-prr-rfc6937bis-03: set cwnd to ssthresh exiting fast recovery?". Mention the potential for bursts as a result of setting cwnd = ssthresh. Say that pacing is RECOMMENDED to deal with this.
- * Revised RecoverFS initialization to handle fast recoveries with mixes of real and spurious loss detection events (due to reordering), and incorporate consideration for a potentially large volume of data that is SACKed before fast recovery starts.
- * Fixed bugs in the definition of DeliveredData (reverted to definition from RFC 6937).

- * Clarified PRR triggers initialization based on start of congestion control reduction, not loss recovery, since congestion control may reduce ssthresh for each round trip with new losses in recovery.

- * Fixed bugs in PRR examples.

About revision 10:

- * Minor typo fixes and wordsmithing.

About revision 11:

- * Based on comments at the TCPM session at IETF 120, clarified the scope of congestion control algorithms for which PRR can be used, and clarified that it can be used for Reno or CUBIC.

About revision 12:

- * Added "About revision 11" and "About revision 12" sections.
- * Added a clarification about the applicability to CUBIC in the algorithm section.

About revision 13:

- * Switch from using the RFC 6675 "pipe" concept to an "inflight" concept that is independent of loss detection algorithm, and thus is usable with RACK-TLP loss detection [RFC8985]

About revision 14:

- * Numerous editorial changes based on 2025-04-15 review from WIT area director Gorrry Fairhurst.
- * Added a note to the RFC Editor to remove this "Document and WG Information" section before publication.
- * Rephrased all sentences with "we" or "our" to remove those words.
- * Updated the RFC2119 MUST/SHOULD/MAY/... text to use the latest boilerplate text from RFC8174, and moved this text into a separate section.
- * Ensured that each term in the "Definitions" section is listed with (a) the term, (b) an actual in-line definition, and (c) the citation of the original source reference, where appropriate.

- * Added missing definitions for terms used in the document: cwnd, rwnd, ssthresh, SND.NXT, RMSS
- * In the "Relationships to other standards", after the paragraph about the congestion control algorithms with which PRR can be used, added a paragraph about PRR's independence from loss detection algorithm details and an explicit list of loss detection algorithms with which PRR can be used.
- * Where appropriate, changed "TCP" to a more generic phrase, like: "transport protocol", "connection", or "sender", depending on the context. Left "TCP" in place where that was the precise term that was appropriate in the context, given the protocol or packet header details. There are now no references to "TCP" in between the definition of SMSS and the "Adapting PRR to other transport protocols" section. The "Algorithm", "Examples", and "Properties" sections no longer mention "TCP".
- * Corrected the two occurrences of "MSS" in the pseudocode to use "SMSS", since "SMSS" has a definition and is consistent with the Reno (RFC5681) and CUBIC (RFC9438) documents.
- * Clarified the recommendation to use pacing to avoid bursts, and moved this into its own paragraph to make it easier for the reader to see.

About revision 15:

- * Fixed the description of the initialization of RecoverFS to match the latest RecoverFS pseudocode
- * Add a note that in the first example both algorithms (RFC6675 and PRR) complete the fast recovery episode with a cwnd matching the ssthresh of 20.
- * Revised order of 2nd and 4th co-author
- * Numerous editorial changes based on 2025-05-27 last call Genart review from Russ Housley, including the following changes.
- * Fixed abstract and intro sections that said that this document "updates" the experimental PRR algorithm to clarify that this document obsoletes the experimental PRR RFC
- * To address the feedback 'The 7th paragraph of Section 5 begins with "A final change"; yet the 8th paragraph talks about another adaptation to PRR', reworded the "A final change" phrase.

- * Moved the paragraph about measurement studies to a new "Measurement Studies" section, to address the feedback: 'The last paragraph of Section 5 is not really about changes since the publication of RFC 6937'
- * Fixed various minor editorial issues identified in the review

About revision 16:

- * Revised the description and caption for the figures to try to improve clarity.

About revision 17:

- * Moved the explanation of "Van Jacobson's packet conservation principle" to be before the first use of the concept in the phrase "strictly packet conserving".
- * Numerous editorial changes based on the 29 suggestions in the 2025-06-03 perfmtrdir review from Paul Aitken ("perfmtrdir review of draft-ietf-tcpm-prr-rfc6937bis-16"), including the following larger-scale changes.
- * Ensured that all references to RFCs (mainly RFC6675 and RFC6937) used proper xref tags.
- * Moved the "Definitions" section to be immediately before the "Background" section, so that more terms are defined before being used.

About revision 18:

- * Several editorial changes based on the 2025-06-04 Opsdir review from Daniele Ceccarelli ("draft-ietf-tcpm-prr-rfc6937bis-16 ietf last call Opsdir review"), including the following larger-scale changes.
- * Moved the content in the "Background" section into the "Introduction" section and revised the content to ensure that each passage only uses terms and concepts already described by the earlier text.
- * Made things simpler and more consistent by replacing a few "Reduction Bound algorithms" with "Reduction Bounds". In revision 16 we already had the simpler "Reduction Bounds" phrasing in four spots, so this makes the text more self-consistent.

About revision 19:

- * Fix a nit in the abstract caught by "idnits" online tool: 'The abstract seems to contain references ([RFC6937]), which it shouldn't. Please replace those with straight textual mentions of the documents in question.'
- * Several editorial changes based on the suggestions in the 2025-06-12 perfmtrdir review from Paul Aitken (tcpm thread: "perfmtrdir review of draft-ietf-tcpm-prr-rfc6937bis-16").

About revision 20:

- * Several editorial changes based on the suggestions in the 2025-06-13 review from Mohamed Boucadair (tcpm thread: "Mohamed Boucadair's Yes on draft-ietf-tcpm-prr-rfc6937bis-19: (with COMMENT)"), including the following larger changes.
- * Changed the "Proportional Rate Reduction for TCP" title to "Proportional Rate Reduction"
- * Added an "Operational Considerations" section.
- * Moved the prose description of the computation of DeliveredData, inflight, RecoverFS, etc, from the "Definitions" section to the "Algorithm" section.
- * Moved the example section so that it is immediately after the discussion about properties, rather than immediately before.

About revision 21:

- * Fix a typo from revision 20 where an extra/old sentence about multiple implementations was accidentally left in the document.

4. Definitions

The following terms, parameters, and state variables are used as they are defined in earlier documents:

SND.UNA: The oldest unacknowledged sequence number. This is defined in Section 3.4 of [RFC9293].

SND.NXT: The next sequence number to be sent. This is defined in Section 3.4 of [RFC9293].

duplicate ACK: An acknowledgment is considered a "duplicate ACK" or "duplicate acknowledgment" when (a) the receiver of the ACK has outstanding data, (b) the incoming acknowledgment carries no data, (c) the SYN and FIN bits are both off, (d) the acknowledgment number

is equal to SND.UNA, and (e) the advertised window in the incoming acknowledgment equals the advertised window in the last incoming acknowledgment. This is defined in Section 2 of [RFC5681].

FlightSize: The amount of data that has been sent but not yet cumulatively acknowledged. This is defined in Section 2 [RFC5681].

Receiver Maximum Segment Size (RMSS): The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup (see Section 3.7.1 of [RFC9293]). Or, if the MSS option is not used, it is the default of 536 bytes for IPv4 or 1220 bytes for IPv6 (see Section 3.7.1 of [RFC9293]). The size does not include the TCP/IP headers and options. The RMSS is defined in Section 2 of [RFC5681] and section 3.8.6.3 of [RFC9293].

Sender Maximum Segment Size (SMSS): The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191] [RFC8201] [RFC4821] algorithm, RMSS, or other factors. The size does not include the TCP/IP headers and options. This is defined in Section 2 of [RFC5681].

Receiver Window (rwnd): The most recently received advertised receiver window, in bytes. At any given time, a connection MUST NOT send data with a sequence number higher than the sum of SND.UNA and rwnd. This is defined in section 2 [RFC5681].

Congestion Window (cwnd): A state variable that limits the amount of data a connection can send. At any given time, a connection MUST NOT send data if inflight (see below) matches or exceeds cwnd. This is defined in Section 2 of [RFC5681].

Slow Start Threshold (ssthresh): The slow start threshold (ssthresh) state variable is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission. During fast recovery, ssthresh is the target window size for a fast recovery episode, as determined by the congestion control algorithm. This is defined in Section 3.1 of [RFC5681].

PRR defines additional variables and terms:

Delivered Data (DeliveredData): The data sender's best estimate of the total number of bytes that the current ACK indicates have been delivered to the receiver since the previously received ACK.

In-Flight Data (inflight): The data sender's best estimate of the number of unacknowledged bytes in flight in the network; i.e., bytes that were sent and neither lost nor received by the data receiver.

Recovery Flight Size (RecoverFS): The number of bytes the sender estimates might possibly be delivered over the course of the current PRR episode.

SafeACK: A local boolean variable indicating that the current ACK indicates the recovery is making good progress and the sender can send more aggressively, increasing inflight, if appropriate.

SndCnt: A local variable indicating exactly how many bytes should be sent in response to each ACK.

Voluntary window reductions: choosing not to send data in response to some ACKs, for the purpose of reducing the sending window size and data rate.

5. Changes Relative to RFC 6937

The largest change since [RFC6937] is the introduction of a new heuristic that uses good recovery progress (for TCP, when the latest ACK advances SND.UNA and does not indicate that a prior fast retransmit has been lost) to select the Reduction Bound (PRR-CRB or PRR-SSRB). [RFC6937] left the choice of Reduction Bound to the discretion of the implementer but recommended to use PRR-SSRB by default. For all of the environments explored in earlier PRR research, the new heuristic is consistent with the old recommendation.

The paper "An Internet-Wide Analysis of Traffic Policing" [Flach2016policing] uncovered a crucial situation not previously explored, where both Reduction Bounds perform very poorly, but for different reasons. Under many configurations, token bucket traffic policers can suddenly start discarding a large fraction of the traffic when tokens are depleted, without any warning to the end systems. The transport congestion control has no opportunity to measure the token rate, and sets ssthresh based on the previously observed path performance. This value for ssthresh may cause a data rate that is substantially larger than the token replenishment rate, causing high loss. Under these conditions, both Reduction Bounds perform very poorly. PRR-CRB is too timid, sometimes causing very long recovery times at smaller than necessary windows, and PRR-SSRB is too aggressive, often causing many retransmissions to be lost for multiple rounds. Both cases lead to prolonged recovery, decimating application latency and/or goodput.

Investigating these environments led to the development of a "SafeACK" heuristic to dynamically switch between Reduction Bounds: by default conservatively use PRR-CRB and only switch to PRR-SSRB when ACKs indicate the recovery is making good progress (SND.UNA is advancing without detecting any new losses). The SafeACK heuristic was experimented with in Google's CDN [Flach2016policing] and implemented in Linux TCP since 2015.

This SafeACK heuristic is only invoked where losses, application-limited behavior, or other events cause the current estimate of in-flight data to fall below ssthresh. The high loss rates that make the heuristic essential are only common in the presence of heavy losses such as traffic policers [Flach2016policing]. In these environments the heuristic performs better than either bound by itself.

Another PRR algorithm change improves the sending process when the sender enters recovery after a large portion of sequence space has been SACKed. This scenario could happen when the sender has previously detected reordering, for example, by using [RFC8985]. In the previous version of PRR, RecoverFS did not properly account for sequence ranges SACKed before entering fast recovery, which caused PRR to initially send too slowly. With the change, PRR properly accounts for sequence ranges SACKed before entering fast recovery.

Yet another change is to force a fast retransmit upon the first ACK that triggers the recovery. Previously, PRR may not allow a fast retransmit (i.e., SndCnt is 0) on the first ACK in fast recovery, depending on the loss situation. Forcing a fast retransmit is important to maintain the ACK clock and avoid potential retransmission timeout (RTO) events. The forced fast retransmit only happens once during the entire recovery and still follows the packet conservation principles in PRR. This heuristic has been implemented since the first widely deployed TCP PRR implementation in 2011 [First_TCP_PRR].

In another change, upon exiting recovery a data sender sets cwnd to ssthresh. This is important for robust performance. Without setting cwnd to ssthresh at the end of recovery, with application-limited sender behavior and some loss patterns cwnd could end fast recovery well below ssthresh, leading to bad performance. The performance could, in some cases, be worse than [RFC6675] recovery, which simply sets cwnd to ssthresh at the start of recovery. This behavior of setting cwnd to ssthresh at the end of recovery has been implemented since the first widely deployed TCP PRR implementation in 2011 [First_TCP_PRR], and is similar to [RFC6675], which specifies setting cwnd to ssthresh at the start of recovery.

Since [RFC6937] was written, PRR has also been adapted to perform multiplicative window reduction for non-loss based congestion control algorithms, such as for [RFC3168] style Explicit Congestion Notification (ECN). This can be done by using some parts of the loss recovery state machine (in particular the RecoveryPoint from [RFC6675]) to invoke the PRR ACK processing for exactly one round trip worth of ACKs. However, note that using PRR for cwnd reductions for [RFC3168] ECN has been observed, with some approaches to Active Queue Management (AQM), to cause an excess cwnd reduction during ECN-triggered congestion episodes, as noted in [VCC].

6. 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. In particular, PRR is applicable to both Reno [RFC5681] and CUBIC [RFC9438] congestion control. PRR is described as a modification to "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP" [RFC6675]. It is most accurate with SACK [RFC2018] but does not require SACK.

PRR can be used in conjunction with a wide array of loss detection algorithms. This is because PRR does not have any dependencies on the details of how a loss detection algorithm estimates which packets have been delivered and which packets have been lost. Upon the reception of each ACK, PRR simply needs the loss detection algorithm to communicate how many packets have been marked as lost and how many packets have been marked as delivered. Thus PRR MAY be used in conjunction with the loss detection algorithms specified or described in the following documents: Reno [RFC5681], NewReno [RFC6582], SACK [RFC6675], FACK [FACK], and RACK-TLP [RFC8985]. Because of the performance properties of RACK-TLP, including resilience to tail loss, reordering, and lost retransmissions, it is RECOMMENDED that PRR is implemented together with RACK-TLP loss recovery [RFC8985].

The SafeACK heuristic came about as a result of robust Lost Retransmission Detection under development in an early precursor to [RFC8985]. Without Lost Retransmission Detection, policers that cause very high loss rates are at very high risk of causing retransmission timeouts because Reno [RFC5681], CUBIC [RFC9438], and [RFC6675] can send retransmissions significantly above the policed rate.

7. Algorithm

7.1. Initialization Steps

At the beginning of a congestion control response episode initiated by the congestion control algorithm, a 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.

The PRR initialization allows a congestion control algorithm, `CongCtrlAlg()`, that might set `ssthresh` to something other than `FlightSize/2` (including, e.g., CUBIC [RFC9438]).

A key step of PRR initialization is computing Recovery Flight Size (`RecoverFS`), the number of bytes the data sender estimates might possibly be delivered over the course of the PRR episode. This can be thought of as the sum of the following values at the start of the episode: inflight, the bytes cumulatively acknowledged in the ACK triggering recovery, the bytes SACKed in the ACK triggering recovery, and the bytes between `SND.UNA` and `SND.NXT` that have been marked lost. The `RecoverFS` includes losses because losses are marked using heuristics, so some packets previously marked as lost may ultimately be delivered (without being retransmitted) during recovery. PRR uses `RecoverFS` to compute a smooth sending rate. Upon entering fast recovery, PRR initializes `RecoverFS`, and `RecoverFS` remains constant during a given fast recovery episode.

The full sequence of PRR algorithm initialization steps is as follows:

```
ssthresh = CongCtrlAlg()      // Target flight size in recovery
prrr_delivered = 0            // Total bytes delivered in recovery
prrr_out = 0                  // Total bytes sent in recovery
RecoverFS = SND.NXT - SND.UNA
// Bytes SACKed before entering recovery will not be
// marked as delivered during recovery:
RecoverFS -= (bytes SACKed in scoreboard)
// Include the (common) case of selectively ACKed bytes:
RecoverFS += (bytes newly SACKed)
// Include the (rare) case of cumulatively ACKed bytes:
RecoverFS += (bytes newly cumulatively acknowledged)
```

7.2. Per-ACK Steps

On every ACK starting or during fast recovery, excluding the ACK that concludes a PRR episode, PRR executes the following steps.

First, the sender computes `DeliveredData`, the data sender's best estimate of the total number of bytes that the current ACK indicates have been delivered to the receiver since the previously received ACK. With SACK, `DeliveredData` can be computed precisely as the change in `SND.UNA`, plus the (signed) change in `SACKed`. Thus, in the special case when there are no `SACKed` sequence ranges in the scoreboard before or after the ACK, `DeliveredData` is the change in `SND.UNA`. In recovery without SACK, `DeliveredData` is estimated to be 1 SMSS on receiving a duplicate ACK, and on a subsequent partial or full ACK `DeliveredData` is the change in `SND.UNA`, minus 1 SMSS for each preceding duplicate ACK. Note that without SACK, a poorly-behaved receiver that returns extraneous duplicate ACKs (as described in [Savage99]) could attempt to artificially inflate `DeliveredData`. As a mitigation, if not using SACK then PRR disallows incrementing `DeliveredData` when the total bytes delivered in a PRR episode would exceed the estimated data outstanding upon entering recovery (`RecoverFS`).

Next, the sender computes `inflight`, the data sender's best estimate of the number of bytes that are in flight 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].

Next, the sender computes `SafeACK`, a local boolean variable indicating that the current ACK reported good progress. `SafeACK` is true only when the ACK has cumulatively acknowledged new data and the ACK does not indicate further losses. For example, an ACK triggering [RFC6675] "rescue" retransmission (Section 4, `NextSeg()` condition 4) may indicate further losses. Both conditions indicate the recovery is making good progress and the sender can send more aggressively, increasing `inflight`, if appropriate.

Finally, the sender uses `DeliveredData`, `inflight`, `SafeACK`, and other PRR state to compute `SndCnt`, a local variable indicating exactly how many bytes should be sent in response to each ACK, and then uses `SndCnt` to update `cwnd`.

The full sequence of per-ACK PRR algorithm steps is as follows:

```
if (DeliveredData is 0)
    Return

prp_delivered += DeliveredData
inflight = (estimated volume of in-flight data)
SafeACK = (SND.UNA advances and no further loss indicated)
if (inflight > ssthresh) {
    // Proportional Rate Reduction
    // This uses integer division, rounding up:
    #define DIV_ROUND_UP(n, d) (((n) + (d) - 1) / (d))
    out = DIV_ROUND_UP(prp_delivered * ssthresh, RecoverFS)
    SndCnt = out - prp_out
} else {
    // PRR-CRB by default
    SndCnt = MAX(prp_delivered - prp_out, DeliveredData)
    if (SafeACK) {
        // PRR-SSRB when recovery is making good progress
        SndCnt += SMSS
    }
    // Attempt to catch up, as permitted
    SndCnt = MIN(ssthresh - inflight, SndCnt)
}

if (prp_out is 0 AND SndCnt is 0) {
    // Force a fast retransmit upon entering recovery
    SndCnt = SMSS
}
cwnd = inflight + SndCnt
```

After the sender computes `SndCnt` and uses it to update `cwnd`, the sender transmits more data. Note that the decision of which data to send (e.g., retransmit missing data or send more new data) is out of scope for this document.

7.3. Per-Transmit Steps

On any data transmission or retransmission, PRR executes the following:

```
pr_r_out += (data sent)
```

7.4. Completion Steps

A PRR episode ends upon either completing fast recovery, or before initiating a new PRR episode due to a new congestion control response episode.

On the completion of a PRR episode, PRR executes the following:

```
cwnd = ssthresh
```

Note that this step that sets cwnd to ssthresh can potentially, in some scenarios, allow a burst of back-to-back segments into the network.

It is RECOMMENDED that implementations use pacing to reduce the burstiness of data traffic. This recommendation is consistent with current practice to mitigate bursts (e.g., [I-D.welzl-iccrp-pacing]), including pacing transmission bursts after restarting from idle.

8. Properties

The following properties are common to both PRR-CRB and PRR-SSRB, except as noted:

PRR attempts to maintain the sender's ACK clocking across recovery events, including burst losses. By contrast, [RFC6675] can send large, unclocked bursts following burst losses.

Normally, PRR will spread voluntary window reductions out evenly across a full RTT. This has the potential to generally reduce the burstiness of Internet traffic, and could be considered to be a type of soft pacing. Hypothetically, any pacing increases the probability that different flows are interleaved, reducing the opportunity for ACK compression and other phenomena that increase traffic burstiness. However, these effects have not been quantified.

If there are minimal losses, PRR will converge to exactly the target window chosen by the congestion control algorithm. Note that as the sender approaches the end of recovery, pr_r_delivered will approach RecoverFS and SndCnt will be computed such that pr_r_out approaches ssthresh.

Implicit window reductions, due to multiple isolated losses during recovery, cause later voluntary reductions to be skipped. For small numbers of losses, the window size ends at exactly the window chosen by the congestion control algorithm.

For burst losses, earlier voluntary window reductions can be undone by sending extra segments in response to ACKs arriving later during recovery. Note that as long as some voluntary window reductions are not undone, and there is no application stall, the final value for inflight will be the same as ssthresh.

PRR using either Reduction Bound improves the situation when there are application stalls, e.g., when the sending application does not queue data for transmission quickly enough or the receiver stops advancing its receive window. When there is an application stall early during recovery, `pr_r_out` will fall behind the sum of transmissions allowed by `SndCnt`. The missed opportunities to send due to stalls are treated like banked voluntary window reductions; specifically, they cause `pr_r_delivered - pr_r_out` to be significantly positive. If the application catches up while the sender is still in recovery, the sender will send a partial window burst to grow inflight to catch up to exactly where it would have been had the application never stalled. Although such a burst could negatively impact the given flow or other sharing flows, this is exactly what happens every time there is a partial-RTT application stall while not in recovery. PRR makes partial-RTT stall behavior uniform in all states. Changing this behavior is out of scope for this document.

PRR with Reduction Bound is less sensitive to errors in the inflight estimator. While in recovery, inflight is intrinsically an estimator, using incomplete information to estimate if un-SACKed segments are actually lost or merely out of order in the network. Under some conditions, inflight can have significant errors; for example, inflight is underestimated when a burst of reordered data is prematurely assumed to be lost and marked for retransmission. If the transmissions are regulated directly by inflight as they are with [RFC6675], a step discontinuity in the inflight estimator causes a burst of data, which cannot be retracted once the inflight estimator is corrected a few ACKs later. For PRR dynamics, inflight merely determines which algorithm, PRR or the Reduction Bound, is used to compute `SndCnt` from `DeliveredData`. While inflight is underestimated, the algorithms are different by at most 1 segment per ACK. Once inflight is updated, they converge to the same final window at the end of recovery.

Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. This Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck with no cross traffic, the queue will maintain exactly constant length for the duration of the recovery, except for ± 1 fluctuation due to differences in packet arrival and exit times. See Appendix A for a detailed discussion of this property.

Although the Strong Packet Conservation Bound is very appealing for a number of reasons, earlier measurements (in section 6 of [RFC6675]) demonstrate that it is less aggressive and does not perform as well as [RFC6675], which permits bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits a sender to send one extra segment per ACK as compared to the Packet Conserving Bound when the ACK indicates the recovery is in good progress without further losses. From the perspective of a strict Packet Conserving Bound, PRR-SSRB does indeed open the window during recovery; however, it is significantly less aggressive than [RFC6675] in the presence of burst losses. The [RFC6675] "half window of silence" may temporarily reduce queue pressure when congestion control does not reduce the congestion window entering recovery to avoid further losses. The goal of PRR is to minimize the opportunities to lose the self clock by smoothly controlling inflight toward the target set by the congestion control. It is the congestion control's responsibility to avoid a full queue, not PRR.

9. Examples

This section illustrates the PRR and [RFC6675] algorithms by showing their different behaviors for two example scenarios: a connection experiencing either a single loss or a burst of 15 consecutive losses. All cases use bulk data transfers (no application pauses), Reno congestion control [RFC5681], and `cwnd = FlightSize = inflight = 20` segments, so `ssthresh` will be set to 10 at the beginning of recovery. The scenarios use standard Fast Retransmit [RFC5681] and Limited Transmit [RFC3042], so the sender will send two new segments followed by one retransmit in response to the first three duplicate ACKs following the losses.

Each of the diagrams below shows the per ACK response to the first round trip for the two recovery algorithms when the zeroth segment is lost. The top line ("ack#") indicates the transmitted segment number triggering the ACKs, with an X for the lost segment. The "cwnd" and "inflight" lines indicate the values of cwnd and inflight, respectively, for these algorithms after processing each returning

ACK but before further (re)transmission. The "sent" line indicates how much 'N'ew or 'R'etransmitted data would be sent. Note that the algorithms for deciding which data to send are out of scope of this document.

RFC 6675

a	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
c		20	20	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
i		19	19	18	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9	9	9	9
s		N	N	R										N	N	N	N	N	N	N	N	N	N

PRR

a	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
c		20	20	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11	10	10	10
i		19	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11	10	10	9	9
s		N	N	R		N		N		N		N		N		N		N		N		N	N

a: ack#; c: cwnd; i: inflight; s: sent

Figure 1

In this first example, ACK#1 through ACK#19 contain SACKs for the original flight of data, ACK#20 and ACK#21 carry SACKs for the limited transmits triggered by the first and second SACKed segments, and ACK#22 carries the full cumulative ACK covering all data up through the limited transmits. ACK#22 completes the fast recovery episode, and thus completes the PRR episode.

Note that both algorithms send the same total amount of data, and both algorithms complete the fast recovery episode with a cwnd matching the ssthresh of 20. [RFC6675] experiences a "half window of silence" while PRR spreads the voluntary window reduction across an entire RTT.

Next, consider an example scenario with the same initial conditions, except that the first 15 packets (0-14) are lost. During the remainder of the lossy round trip, only 5 ACKs are returned to the sender. The following examines each of these algorithms in succession.

RFC 6675

a	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
c																20	20	10	10	10
i																19	19	4	9	9
s																N	N	6R	R	R

PRR

a	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
c																20	20	5	5	5
i																19	19	4	4	4
s																N	N	R	R	R

a: ack#; c: cwnd; i: inflight; s: sent

Figure 2

In this specific situation, [RFC6675] is more aggressive because once Fast Retransmit is triggered (on the ACK for segment 17), the sender immediately retransmits sufficient data to bring inflight up to cwnd. Earlier measurements (in section 6 of [RFC6675]) indicate that [RFC6675] significantly outperforms [RFC6937] PRR using only PRR-CRB, and some other similarly conservative algorithms that were tested, showing that it is significantly common for the actual losses to exceed the cwnd reduction determined by the congestion control algorithm.

Under such heavy losses, during the first round trip of fast recovery PRR uses the PRR-CRB to follow the packet conservation principle. Since the total losses bring inflight below ssthresh, data is sent such that the total data transmitted, prr_out, follows the total data delivered to the receiver as reported by returning ACKs. Transmission is controlled by the sending limit, which is set to prr_delivered - prr_out.

While not shown in the figure above, once the fast retransmits sent starting at ACK#17 are delivered and elicit ACKs that increment the SND.UNA, PRR enters PRR-SSRB and increases the window by exactly 1 segment per ACK until inflight rises to ssthresh during recovery. On heavy losses when cwnd is large, PRR-SSRB recovers the losses exponentially faster than PRR-CRB. Although increasing the window during recovery seems to be ill advised, it is important to remember that this is actually less aggressive than permitted by [RFC6675], which sends the same quantity of additional data as a single burst in response to the ACK that triggered Fast Retransmit.

For less severe loss events, where the total losses are smaller than the difference between `FlightSize` and `ssthresh`, PRR-CRB and PRR-SSRB are not invoked since PRR stays in the proportional rate reduction mode.

10. Adapting PRR to other transport protocols

The main PRR algorithm and reductions bounds can be adapted to any transport that can support [RFC6675]. In one major implementation (Linux TCP) PRR has been the fast recovery algorithm for its default and supported congestion control modules since its introduction in 2011 [First_TCP_PRR].

The SafeACK heuristic can be generalized as any ACK of a retransmission that does not cause some other segment to be marked for retransmission.

11. Measurement Studies

For [RFC6937] a companion paper [IMC11] evaluated [RFC3517] and various experimental PRR versions in a large-scale measurement study. At the time of publication, the legacy algorithms used in that study are no longer present in the code base used in that study, making such comparisons difficult without recreating historical algorithms. Readers interested in the measurement study should review section 5 of [RFC6937] and the IMC paper [IMC11].

12. Operational Considerations

12.1. Incremental Deployment

PRR is incrementally deployable, because it utilizes only existing transport protocol mechanisms for data delivery acknowledgment and the detection of lost data. PRR only requires only changes to the transport protocol implementation at the data sender; it does not require any changes at data receivers or in networks. This allows data senders using PRR to work correctly with any existing data receivers or networks. PRR does not require any changes to or assistance from routers, switches, or other devices in the network.

12.2. Fairness

PRR is designed to maintain the fairness properties of the congestion control algorithm with which it is deployed. PRR only operates during a congestion control response episode, such as fast recovery or response to [RFC3168] ECN, and only makes short-term, per-acknowledgment decisions to smoothly regulate the volume of in-flight data during an episode such that at the end of the episode it will be

as close as possible to the slow start threshold (ssthresh), as determined by the congestion control algorithm. PRR does not modify the congestion control cwnd increase or decrease mechanisms outside of congestion control response episodes.

12.3. Protecting the Network Against Excessive Queuing and Packet Loss

Over long time scales, PRR is designed to maintain the queuing and packet loss properties of the congestion control algorithm with which it is deployed. As noted above, PRR only operates during a congestion control response episode, such as fast recovery or response to ECN, and only makes short-term, per-acknowledgment decisions to smoothly regulate the volume of in-flight data during an episode such that at the end of the episode it will be as close as possible to the slow start threshold (ssthresh), as determined by the congestion control algorithm.

Over short time scales, PRR is designed to cause lower packet loss rates than preceding approaches like [RFC6675]. At a high level, PRR is inspired by the packet conservation principle, and, as much as possible, PRR relies on the self clock process. By contrast, with [RFC6675] a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a large burst of data that is much larger than the volume of delivered data. PRR avoids such bursts by basing transmission decisions on the volume of delivered data rather than the volume of lost data. Furthermore, as noted above, PRR-SSRB is less aggressive than [RFC6675] (transmitting fewer segments or taking more time to transmit them), and it outperforms due to the lower probability of additional losses during recovery.

13. Acknowledgements

This document is based in part on previous work by Janey C. Hoe (see section 3.2, "Recovery from Multiple Packet Losses", of [Hoe96Startup]) and Matt Mathis, Jeff Semke, and Jamshid Mahdavi [RHID], and influenced by several discussions with John Heffner.

Monia Ghobadi and Sivasankar Radhakrishnan helped analyze the experiments. Ilpo Jarvinen reviewed the initial implementation. Mark Allman, Richard Scheffenegger, Markku Kojo, Mirja Kuehlewind, Gorrry Fairhurst, Russ Housley, Paul Aitken, Daniele Ceccarelli, and Mohamed Boucadair improved the document through their insightful reviews and suggestions.

14. IANA Considerations

This memo includes no request to IANA.

15. Security Considerations

PRR does not change the risk profile for transport protocols.

Implementers that change PRR from counting bytes to segments have to be cautious about the effects of ACK splitting attacks [Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

16. Normative References

- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<https://www.rfc-editor.org/info/rfc2018>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <<https://www.rfc-editor.org/info/rfc6582>>.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <<https://www.rfc-editor.org/info/rfc6675>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.
- [RFC8985] Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <<https://www.rfc-editor.org/info/rfc8985>>.
- [RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <<https://www.rfc-editor.org/info/rfc9293>>.
- [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>>.

17. Informative References

- [FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM SIGCOMM1996, August 1996, <<https://dl.acm.org/doi/pdf/10.1145/248157.248181>>.
- [First_TCP_PRR] "Proportional Rate Reduction for TCP.", commit a262f0cdf1f2916ea918dc329492abb5323d9a6c, August 2011, <<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=a262f0cdf1f2916ea918dc329492abb5323d9a6c>>.
- [Flach2016policing] Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Al Karim, T., Katz-Bassett, E., and R. Govindan, "An Internet-Wide Analysis of Traffic Policing", ACM SIGCOMM SIGCOMM2016, August 2016.
- [Hoe96Startup] Hoe, J., "Improving the start-up behavior of a congestion control scheme for TCP", ACM SIGCOMM SIGCOMM1996, August 1996.

- [I-D.welzl-iccrp-pacing]
Welzl, M., Eddy, W., Goel, V., and M. Txen, "Pacing in Transport Protocols", Work in Progress, Internet-Draft, draft-welzl-iccrp-pacing, 3 March 2025, <<https://datatracker.ietf.org/doc/html/draft-welzl-iccrp-pacing>>.
- [IMC11] Dukkipati, N., Mathis, M., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany, November 2011.
- [Jacobson88]
Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Comput. Commun. Rev. 18(4), August 1988.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <<https://www.rfc-editor.org/info/rfc3042>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, DOI 10.17487/RFC3517, April 2003, <<https://www.rfc-editor.org/info/rfc3517>>.
- [RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <<https://www.rfc-editor.org/info/rfc6937>>.
- [RHID] Mathis, M., Semke, J., and J. Mahdavi, "The Rate-Halving Algorithm for TCP Congestion Control", Work in Progress, August 1999, <<https://datatracker.ietf.org/doc/html/draft-mathis-tcp-ratehalving>>.
- [Savage99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

[VCC] Cronkite-Ratcliff, B., Bergman, A., Vargaftik, S., Ravi, M., McKeown, N., Abraham, I., and I. Keslassy, "Virtualized Congestion Control (Extended Version)", August 2016, <http://www.ee.technion.ac.il/~isaac/p/sigcomm16_vcc_extended.pdf>.

Appendix A. Strong Packet Conservation Bound

PRR-CRB is based on a conservative, philosophically pure, and aesthetically appealing Strong Packet Conservation Bound, described here. Although inspired by the packet conservation principle [Jacobson88], it differs in how it treats segments that are missing and presumed lost. Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. Note that the effects of presumed losses are included in the inflight calculation, but do not affect the outcome of PRR-CRB, once inflight has fallen below ssthresh.

This Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck that is carrying no other traffic, the queue will maintain exactly constant length for the entire duration of the recovery, except for ± 1 fluctuation due to differences in packet arrival and exit times. Any less aggressive algorithm will result in a declining queue at the bottleneck. Any more aggressive algorithm will result in an increasing queue or additional losses if it is a full drop tail queue.

This property is demonstrated with a thought experiment:

Imagine a network path that has insignificant delays in both directions, except for the processing time and queue at a single bottleneck in the forward path. In particular, when a packet is "served" at the head of the bottleneck queue, the following events happen in much less than one bottleneck packet time: the packet arrives at the receiver; the receiver sends an ACK that arrives at the sender; the sender processes the ACK and sends some data; the data is queued at the bottleneck.

If SndCnt is set to DeliveredData and nothing else is inhibiting sending data, then clearly the data arriving at the bottleneck queue will exactly replace the data that was served at the head of the queue, so the queue will have a constant length. If queue is drop tail and full, then the queue will stay exactly full. Losses or reordering on the ACK path only cause wider fluctuations in the queue size, but do not raise its peak size, independent of whether the data

is in order or out of order (including loss recovery from an earlier RTT). Any more aggressive algorithm that sends additional data will overflow the drop tail queue and cause loss. Any less aggressive algorithm will under-fill the queue. Therefore, setting SndCnt to DeliveredData is the most aggressive algorithm that does not cause forced losses in this simple network. Relaxing the assumptions (e.g., making delays more authentic and adding more flows, delayed ACKs, etc.) is likely to increase the fine grained fluctuations in queue size but does not change its basic behavior.

Note that the congestion control algorithm implements a broader notion of optimal that includes appropriately sharing the network. Typical congestion control algorithms are likely to reduce the data sent relative to the Packet Conserving Bound implemented by PRR, bringing TCP's actual window down to ssthresh.

Authors' Addresses

Matt Mathis
Email: ietf@mattmathis.net

Neal Cardwell
Google, Inc.
Email: ncardwell@google.com

Yuchung Cheng
Google, Inc.
Email: ycheng@google.com

Nandita Dukkkipati
Google, Inc.
Email: nanditad@google.com