

Internet Engineering Task Force (IETF)  
Request for Comments: 9937  
Obsoletes: 6937  
Category: Standards Track  
ISSN: 2070-1721

M. Mathis  
N. Cardwell  
Y. Cheng  
N. Dukkipati  
Google, Inc.  
December 2025

## Proportional Rate Reduction (PRR)

Пропорциональное снижение скорости

### Аннотация

В этом документе описана стандартная (Standards Track) версия алгоритма пропорционального снижения скорости (Proportional Rate Reduction или PRR), которая отменяет экспериментальную версию, опубликованную в RFC 6937. PRR регулирует объем данных, передаваемых по протоколу TCP или иному транспортному протоколу в процессе ускоренного восстановления (fast recovery). PRR точно регулирует размер находящихся в сети данных (flight size) в процессе восстановления так, что в конце процесса быть как можно ближе к порогу замедленного старта (start threshold или ssthresh), заданному алгоритмом контроля перегрузок.

### Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF<sup>1</sup> и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG<sup>2</sup>. Дополнительные сведения о документах Internet Standard приведены в разделе 2 RFC 7841.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке <https://www.rfc-editor.org/info/rfc9937>.

### Авторские права

Авторские права (Copyright (c) 2025) принадлежат IETF Trust и лицам, указанным в качестве авторов документа. Все права защищены.

К документу применимы права и ограничения, указанные в BCP 78 и IETF Trust Legal Provisions и относящиеся к документам IETF (<http://trustee.ietf.org/license-info>), на момент публикации данного документа. Прочтите упомянутые документы внимательно. Фрагменты программного кода, включённые в этот документ, распространяются в соответствии с пересмотренной лицензией BSD (Revised BSD License), как указано в параграфе 4.e документа IETF Trust Legal Provisions, без каких-либо гарантий (как указано в Revised BSD License).

## Оглавление

1. Введение.....	2
2. Уровни требований.....	2
3. Определения.....	2
4. Отличия от RFC 6937.....	3
5. Взаимодействие с другими стандартами.....	4
6. Алгоритм.....	4
6.1. Этапы инициализации.....	4
6.2. Действия при получении АСК.....	5
6.3. Этапы передачи.....	6
6.4. Завершающие действия.....	6
7. Свойства.....	6
8. Примеры.....	7
9. Адаптация PRR к другим транспортным протоколам.....	7
10. Измерения.....	8
11. Эксплуатационные вопросы.....	8
11.1. Поэтапное внедрение.....	8
11.2. Беспристрастность.....	8
11.3. Защита сети от избыточных очередей и потери пакетов.....	8
12. Взаимодействие с IANA.....	8
13. Вопросы безопасности.....	8
14. Литература.....	8
14.1. Нормативные документы.....	8
14.2. Дополнительная литература.....	9
Приложение А. Строгая привязка к сохранению пакетов.....	9
Благодарности.....	10
Адреса авторов.....	10

<sup>1</sup>Internet Engineering Task Force - комиссия по решению инженерных задач Internet.

<sup>2</sup>Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

## 1. Введение

Принцип сохранения пакетов Ван Якобсона (Van Jacobson) [Jacobson88] задаёт процесс самосинхронизации, где  $N$  сегментов данных, доставленных получателю, вызывают генерацию подтверждения, которое отправитель данных использует для синхронизации отправки в сеть других  $N$  пакетов данных.

Алгоритмы контроля перегрузок, такие как Reno [RFC5681] и CUBIC [RFC9438], работают на основе такого процесса самосинхронизации. Они контролируют процесс передачи в транспортном соединении, используя окно перегрузки (congestion window или cwnd) для ограничения объёма находящихся в сети (inflight) данных в текущий момент. Кроме того, эти алгоритмы требуют от транспортного соединения снижения cwnd в ответ на потерю пакетов. Алгоритм быстрого восстановления (fast recovery, см. [RFC5681] и [RFC6675]) управляет снижением cwnd на основе обратной связи в виде подтверждений. Заявленная цель состоит в поддержке самосинхронизации отправителя на основе возвращаемых ему в процессе восстановления пакетов ACK для отправки в сеть большего объёма данных. Без пропорционального снижения скорости (PRR) механизм быстрого восстановления обычно регулирует окно, ожидая, пока пройдёт значительная часть времени кругового обхода (RTT) (половина RTT из ACK для Reno [RFC5681] или 30% RTT для CUBIC [RFC9438]) перед отправкой в сеть каких-либо данных.

[RFC6675] делает быстрое восстановление выполняется по селективным подтверждениям (Selective Acknowledgment или SACK) [RFC2018] более точным за счёт оценки отправителем числа байтов, остающихся в сети. В [RFC6675] быстрое восстановление реализуется отправкой данных по мере необходимости в каждом пакете ACK, что обеспечивает возможность увеличения числа остающихся в сети байтов до размера, соответствующего ssthresh – целевому размеру окна для быстрого восстановления, заданному алгоритмом контроля перегрузок. Это предотвращает тайм-ауты быстрого восстановления во многих случаях высокого уровня потерь. Однако [RFC6675] имеет два существенных недостатка. Во-первых, этот механизм вызывает большое мультипликативное сокращение cwnd в начале быстрого восстановления, что может вызывать тайм-аут при потере всей второй половины окна или пакетов ACK. Во-вторых, один пакет ACK с опцией SACK, подразумевающей потерю большого объёма данных, может приводить к ступенчатому разрыву оценки объёма остающихся в сети данных и активировать механизм Fast Retransmit для передачи большого объёма данных.

PRR регулирует процесс передачи во время быстрого восстановления так, чтобы избежать излишних корректировок окна, благодаря чему изменение происходит плавно, а в конце восстановления фактический размер окна будет максимально близким к ssthresh.

PRR основывается на принципе сохранения пакетов Ван Якобсона. Насколько возможно, PRR полагается на процесс самосинхронизации и лишь незначительно зависит от точности оценок, таких как оценка объёма находящихся в сети данных. Это обеспечивает алгоритму точность при наличии событий, ведущих к неопределённости других оценок.

Когда  $\text{inflight} > \text{ssthresh}$ , PRR плавно снижает  $\text{inflight}$  в направлении ssthresh за счёт управления скоростью передачи по объёмам доставленных данных и ssthresh.

Когда  $\text{inflight} < \text{ssthresh}$ , PRR адаптивно выбирает один из режимов сокращения (Reduction Bound) для ограничения общего сокращения окна всеми механизмами, включая временные остановки приложений и потери. В качестве базы при сильных перегрузках PRR из осторожности использует консервативное снижение (Conservative Reduction Bound или CRB), которое строго сохраняет пакеты. Когда восстановление представляется проходящим успешно, PRR использует режим снижения замедленного старта (Slow Start Reduction Bound или SSRB), что является более агрессивным по сравнению с PRR-CRB (максимум на 1 сегмент на ACK). PRR-CRB соответствует строгому сохранению пакетов (Strong Packet Conservation Bound), как описано в Приложении А. Однако при использовании в реальных сетях как единственного механизма он работает не так хорошо, как алгоритм, описанный в [RFC6675], который во многих случаях оказывается более агрессивным. PRR-SSRB предлагает компромисс, позволяя в некоторых случаях передавать в соединение 1 дополнительный сегмент на ACK по сравнению с PRR-CRB. Хотя механизм PRR-SSRB менее агрессивен чем [RFC6675] (передаёт меньше сегментов или более медленно), он обеспечивает большую производительность за счёт снижения вероятности дополнительных потерь в процессе восстановления.

В исходном определении принципа сохранения [Jacobson88] пакетов, которые предполагались потерянными (например, помеченные как кандидаты на повтор передачи), считались остающимися в сети. Эта идея отражена в используемой PRR системе оценки остающихся в сети пакетов, но она отличается от Strong Packet Conservation Bound (Приложение А), где оценка выполняется исключительно на основе прибывших к получателю данных.

В этом документе указано несколько основных изменений в исходной версии PRR [RFC6937]. Во-первых, вводится новая адаптивная эвристика взамен настройки вручную параметров, определяющих уровень консервативности PRR при  $\text{inflight} < \text{ssthresh}$  (как для PRR-CRB, так и для PRR-SSRB). Во-вторых, алгоритм задаёт поведение соединений без SACK (не согласовавших поддержку SACK [RFC2018] через опцию SACK-permitted). В-третьих, алгоритм обеспечивает сглаженный процесс доставки даже если отправитель часто переупорядочивает данные и начинает восстановление после того, как для значительной части пространства номеров были получены SACK. Кроме того, в документе дополнительно обсуждается интеграция PRR с алгоритмами контроля перегрузок и обнаружения потерь.

Для PRR имеется значительный опыт внедрения в разных реализациях TCP с начала широкого внедрения реализации TCP PRR в 2011 г. [First\_TCP\_PRR].

## 2. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

## 3. Определения

Приведённые ниже термины, параметры и переменные состояния используются в соответствии с определениями предыдущих документов

**SND.UNA**

Самый старый из неподтвержденных номеров (параграф 3.4 в [RFC9293]).

**SND.NXT**

Следующий номер для передачи (параграф 3.4 в [RFC9293]).

**duplicate ACK**

Подтверждение считается «дубликатом» в описанных здесь алгоритмах, если (a) получатель ACK имеет остающиеся для передачи данные, (b) входящее подтверждение не содержит каких-либо данных, (c) оба флага SYN и FIN сброшены, (d) номер подтверждения равен или превышает значение наибольшего номера подтверждения, полученного в данном соединении (SND.UNA) и (e) анонсируемое во входящем подтверждении окно равно окну, анонсированному в последнем входящем подтверждении (раздел 2 в [RFC5681]).

**FlightSize**

Количество данных, которые уже переданы, но ещё не подтверждены кумулятивно (раздел 2 в [RFC5681]).

**Receiver Maximum Segment Size (RMSS)**

RMSS - размер максимального сегмента, который желает принимать получатель. Это значение задаётся в опции MSS, передаваемой получателем при организации соединения (см. параграф 3.7.1 в [RFC9293]). Если опция MSS при организации соединения не была задана, используется значение 536 байтов для IPv4 и 1220 байтов для IPv6 (см. параграф 3.7.1 в [RFC9293]). Размер не включает заголовков и опций TCP/IP. Определение RMSS дано в разделе 2 [RFC5681] и параграфе 3.8.6.3 [RFC9293].

**Sender Maximum Segment Size (SMSS)**

SMSS представляет собой размер самого большого сегмента, который может быть передан отправителем. Это значение может определяться на основе максимального блока данных, передаваемого через сеть (Maximum Transmission Unit или MTU), алгоритма определения Path MTU [RFC1191] [RFC8201] [RFC4821], RMSS и других факторов. Размер не включает заголовков и опций TCP/IP (раздел 2 в [RFC5681]).

**Receiver Window (rwnd)**

Последнее анонсированное значение размера окна принимающей стороны в байтах. В любой момент через соединение **недопустимо** передавать данные с порядковым номером больше SND.UNA + rwnd (раздел 2 в [RFC5681]).

**Congestion Window (cwnd)**

Переменная состояния TCP, которая ограничивает количество данных, разрешённых протоколу для передачи. В любой момент **недопустимо** передавать данные, если объем находящихся в сети данных (см. ниже) не меньше cwnd (раздел 2 в [RFC5681]).

**Slow Start Threshold (ssthresh)**

Порог замедленного старта (переменная состояния ssthresh) используется для определения момента, когда следует использовать алгоритм замедленного старта или предотвращения перегрузки для управления передачей данных. В процесс быстрого восстановления ssthresh - это максимальный размер окна для эпизода быстрого восстановления, как задано алгоритмом контроля перегрузки (параграф 3.1 в [RFC5681]).

В PRR определяются дополнительные переменные и термины, указанные ниже.

**Delivered Data (DeliveredData)**

Максимальная оценка отправителем общего числа байтов, доставка которых получателю подтверждена текущим ACK с момента получения предыдущего ACK.

**In-Flight Data (inflight)**

Максимальная оценка отправителем общего числа байтов, ещё находящихся в сети (не потерянных, но ещё не принятых адресатом).

**Recovery Flight Size (RecoverFS)**

Число байтов, которые по оценке отправителя могут быть доставлены в течение текущего эпизода PRR.

**SafeACK**

Локальная логическая переменная, указывающая, что текущее подтверждение ACK говорит об успешном процессе восстановления и возможности отправителя передавать более агрессивно, с увеличением inflight, если это нужно.

**SndCnt**

Локальная переменная, указывающая число байтов, которые следует передать в ответ на каждое подтверждение ACK.

**Voluntary window reductions - произвольное сокращение окна**

Решение не передавать данные в ответ на некоторые ACK с целью сокращения окна передачи и скорости отправки.

## 4. Отличия от RFC 6937

Самым большим изменением с момента выпуска [RFC6937] является введение новой эвристики которая использует течение процесса восстановления (для TCP, когда последний пакет ACK достигает SND.UNA и не указывает потерю предшествующего ускоренного повтора передачи) для выбора режима сокращения (PRR-CRB или PRR-SSRB). В [RFC6937] выбор режима сокращения оставлен за разработчиками, но рекомендовано по умолчанию применять PRR-SSRB. Для всех сред, рассмотренных в предшествующих исследованиях PRR новая эвристика соответствует этой рекомендации.

В статье «An Internet-Wide Analysis of Traffic Policing» [Flach2016policing] отмечена критическая ситуация, которая не исследовалась ранее, когда оба варианта сокращения работают очень плохо, но по разным причинам. Во многих конфигурациях системы организации трафика на основе маркеров (token bucket) могут внезапно начать отбрасывание значительной части трафика при исчерпаниии маркеров без выдачи предупреждений конечным системам. Система транспортного контроля перегрузок не имеет возможности измерять скорость расхода маркеров и устанавливает ssthresh на основе ранее наблюдавшейся производительности пути. Такое значение ssthresh может привести к значительному превышению скорости данных над скоростью пополнения маркеров, что приведёт к высоким потерям. В таких условиях оба режима сокращения работают очень плохо. Метод PRR-CRB слишком мягок, что иногда ведёт к очень долгому восстановлению при меньших, чем требуется окнах, а PRR-SSRB слишком агрессивен и часто ведёт к потере множества повторно переданных пакетов в течение нескольких интервалов кругового обхода. Оба случая приводят к длительному восстановлению, ухудшающему задержки в приложениях и/или производительность.

Изучение этих сред привело к разработке эвристики SafeACK для динамического переключения режима сокращения. По умолчанию применяется консервативный метод PRR-CRB, а переключение на PRR-SSRB происходит лишь при

указании пакетами ACK успешного хода восстановления (SND.UNA продвигается без новых потерь). Эвристика SafeACK была опробована в сети доставки содержимого Google (Content Delivery Network или CDN) [Flach2016policing] и реализована в Linux TCP с 2015 г.

Эвристика SafeACK применяется лишь в случаях, когда потери, ограничения приложений или иные события вызывают занижение оценки находящихся в сети пакетов до значения ниже `ssthresh`. Высокая частота потерь, делающая эвристику важной, характерна лишь при значительных потерях, таких как политика организации трафика (traffic policer) [Flach2016policing]. В таких средах эвристика работает лучше любого из двух ограничений, самого по себе.

Другое изменение PRR улучшает процесс передачи, когда отправитель переходит к восстановлению после того, как большая часть пространства номеров была подтверждена SACK. Такая ситуация может возникать при предшествующем обнаружении отправителем нарушения порядка доставки, например, с помощью [RFC8985]. В прежней версии PRR переменная `RecoverFS` некорректно учитывала диапазоны номеров, подтверждённые SACK до начала быстрого восстановления, что вызывало слишком медленную отправку в PRR. Изменение PRR корректно учитывает диапазоны порядковых номеров, подтверждённые SACK до начала быстрого восстановления.

Ещё одно изменение заключается в принудительном повторе передачи по первому пакету ACK, запустившему восстановление. Ранее механизм PRR мог, в зависимости от ситуации с потерями, не разрешать быстрый повтор передачи (т. е. `SndCnt = 0`) по первому пакету ACK при быстром восстановлении. Форсирование ускоренного повтора передачи важно для поддержки синхронизации ACK и предотвращения возможных тайм-аутов при повторе (retransmission timeout или RTO). Форсированный ускоренный повтор передачи выполняется лишь один раз в процессе восстановления и не нарушает принципов сохранения пакетов PRR. Эта эвристика внедрена в первой широко распространённой реализации TCP PRR в 2011 г. [First\_TCP\_PRR].

В соответствии с другим изменением, отправитель устанавливает `cwnd = ssthresh` при выходе из процедуры восстановления. Это важно для отказоустойчивости. Без установки `cwnd = ssthresh` в конце восстановления, а также с учётом отправителей с ограничениями и некоторых картин потерь, `cwnd` в конце восстановления может быть ниже `ssthresh`, что ведёт к снижению производительности. В некоторых случаях производительность может быть ниже, чем при восстановлении [RFC6675], где просто устанавливается `cwnd = ssthresh` в начале восстановления. Установка `cwnd = ssthresh` в конце восстановления была внедрена в первой широко распространённой реализации TCP PRR в 2011 г. [First\_TCP\_PRR] и это похоже на [RFC6675], где устанавливается `cwnd = ssthresh` в начале восстановления.

С момента публикации [RFC6937] механизм PRR был адаптирован для мультипликативного сокращения окна в алгоритмах контроля перегрузки, не основанных на потерях, таких как явное уведомление о перегрузке (Explicit Congestion Notification или ECN) [RFC3168]. Это можно сделать, используя некоторые части конечного автомата восстановления потерь (в частности, `RecoveryPoint` [RFC6675]) для вызова обработки PRR ACK ровно на один интервал кругового обхода из ACK. Однако могут существовать взаимосвязи между применением PRR и подходами с активным управлением очередями (Active Queue Management или AQM) и ECN. Рекомендации по разработке и внедрению механизмов контроля перегрузки приведено в [RFC9743].

## 5. Взаимодействие с другими стандартами

PRR **можно** применять в сочетании с любым алгоритмом контроля перегрузок, предназначенным для мультипликативного снижения скорости передачи данных в течение примерно одного интервала кругового обхода при условии, что текущий объём находящихся в сети данных ограничен размером окна перегрузки (`cwnd`) и целевой объём остающихся в сети данных в процессе сокращения является фиксированным значением, заданным `ssthresh`. В частности, PRR может работать с контролем перегрузок Reno [RFC5681] и CUBIC [RFC9438]. PRR описывается как модификация «A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP» [RFC6675]. Механизм наиболее точен в сочетании с SACK [RFC2018], но не требует наличия SACK.

PRR можно применять в сочетании с широким спектром алгоритмов обнаружения потерь. Это обусловлено независимостью PRR от деталей определения доставленных и потерянных пакетов механизмом обнаружения потерь. При получении каждого пакета ACK механизму PRR просто нужен алгоритм обнаружения потерь для информирования о числе пакетов, помеченных как доставленные и потерянные. Таким образом, PRR **можно** применять в сочетании с алгоритмами обнаружения потерь, заданными или описанными в Reno [RFC5681], NewReno [RFC6582], SACK [RFC6675], Forward Acknowledgment (FACK) [FACK], Recent Acknowledgment Tail Loss Probe (RACK-TLP) [RFC8985]. Благодаря свойствам RACK-TLP, включая устойчивость к потерям данных, нарушению порядку и потерям при повторной передаче **рекомендуется** реализовать PRR в сочетании с восстановлением потерь RACK-TLP [RFC8985].

Эвристика SafeACK является результатом отказоустойчивого обнаружения потерь при повторе передачи (Lost Retransmission Detection) при разработке раннего предшественника [RFC8985]. Без такого обнаружения системы управления трафиком (policer), ведущие к значительному уровню потерь, имеют очень высокий риск возникновения тайм-аутов при повторе передачи, поскольку в Reno [RFC5681], CUBIC [RFC9438] и [RFC6675] повторы возможны со значительным превышением скорости, заданной системой управления трафиком.

## 6. Алгоритм

### 6.1. Этапы инициализации

В начале эпизода реагирования на перегрузку, вызванного алгоритмом контроля отправитель, применяющий PRR, **должен** инициализировать состояние PRR. Момент начала реагирования на перегрузку полностью определяется алгоритмом контроля перегрузки и может, например, совпадать с запуском быстрого восстановления или однократным (за интервал кругового обхода) сокращением при обнаружении потери исходно или повторно переданных пакетов уже после начала быстрого восстановления. Инициализация PRR позволяет использовать алгоритм контроля перегрузок `CongCtrlAlight()`, который может установить для `ssthresh` значение, отличное от `FlightSize/2` (включая , например, CUBIC [RFC9438]).

Ключевым этапом инициализации PRR является расчёт `RecoverFS` (число байтов, которые по оценке отправителя могут быть доставлены в течение эпизода PRR). Это можно представить как сумму установленных в начале эпизода значений объёма данных, остающихся в сети, числа байтов, кумулятивно подтверждённых запустившим восстановление пакетом ACK, числа байтов, подтверждённых SACK в пакете ACK, запустившим восстановление, и

числа байтов между SND.UNA и SND.NXT, помеченных как потерянные. Значение RecoverFS включает потери, поскольку они маркируются с использованием эвристики, и некоторые пакеты, отмеченные как потерянные, могут быть всё-таки доставлены (без повтора передачи) в процессе восстановления. PRR использует RecoverFS для расчёта сглаженной скорости отправки. В начале быстрого восстановления PRR использует RecoverFS и это значение остаётся неизменным в течение данного эпизода быстрого восстановления.

Вся последовательность инициализации PRR приведены в псевдокоде ниже.

```
ssthresh = CongCtrlAlg() // Параметры восстановления
// Целевое значение остающихся в сети данных
prrr_delivered = 0
// Общее число байтов, доставленных при восстановлении
prrr_out = 0
// Общее число байтов, переданных при восстановлении
RecoverFS = SND.NXT - SND.UNA
// Байты, подтверждённые SACK до начала восстановления,
// не будут маркироваться в процессе восстановления
RecoverFS -= (bytes SACKed in scoreboard)
// Включает (распространённый) случай селективно
// подтверждённых ACK байтов
RecoverFS += (bytes newly SACKed)
// Включает (редкий) случай кумулятивно подтверждённых байтов
RecoverFS += (bytes newly cumulatively acknowledged)
```

## 6.2. Действия при получении ACK

При получении каждого пакета ACK в начале и в процессе восстановления (за исключением завершения завершения эпизода PRR) выполняет описанные ниже действия.

Сначала отправитель рассчитывает DeliveredData (оценка отправителем общего числа байтов, доставку которых подтверждает текущий пакет ACK, после предшествующего ACK). При наличии SACK значение DeliveredData можно рассчитать точно как изменение SND.UNA плюс изменение (со знаком) объёма данных, помеченных SACK в таблице результатов. Таким образом, в частном случае отсутствия в таблице результатов подтверждённых SACK диапазонов или после ACK, значение DeliveredData будет указано изменением SND.UNA. При восстановлении без SACK значение DeliveredData оценивается как 1 SMSS на каждый полученный дубликат ACK (т. е. SND.UNA не меняется). При увеличении SND.UNA (т. е. полное или частичное подтверждение ACK) значением DeliveredData будет изменение SND.UNA за вычетом SMSS на каждый принятый дубликат ACK. При отсутствии SACK некорректно работающий получатель, возвращающий лишние дубликаты ACK (см. [Savage99]), может попытаться искусственно завысить значение DeliveredData. В качестве меры предосторожности при отсутствии SACK механизм PRR запрещает увеличение DeliveredData, если общее число байтов, доставленных в эпизоде PRR, превышает оценку объёма данных, оставшихся в сети на момент начала восстановления (RecoverFS).

Затем отправитель вычисляет inflight. Для расчёта в соединениях с поддержкой SACK и обнаружением потерь [RFC6675] **можно** использовать алгоритм pipe, как указано в [RFC6675]. В соединениях с SACK и RACK-TLP [RFC8985] или иным алгоритмом обнаружения потерь расчёт **должен** начинаться с SND.NXT — SND.UNA, из которого (по таблице результатов) вычитаются байты, подтверждённые SACK и потерянные байты, затем добавляются байты, переданные повторно с момента их последней маркировки как потерянных. Для соединений без SACK вместо вычитания подтверждённых SACK байтов из таблицы результатов отправитель **должен** вычитать меньшее из значений RecoverFS и (1 SMSS для каждого предшествующего дубликата ACK в последнем эпизоде восстановления). Функция min() с RecoverFS служит для защиты от некорректно работающих получателей [Savage99].

Далее отправитель определяет SafeACK - локальную логическую переменную, указывающую, что текущий пакет ACK сообщил об успешном ходе восстановления. SafeACK имеет значение true лишь в том случае, когда ACK кумулятивно подтверждает новые данные и не указывает дополнительных потерь. Например, пакет ACK, вызывающий «аварийный» (rescue) повтор передачи (раздел 4 в [RFC6675], NextSeg(), условие 4) может указывать дополнительные потери. Оба условия указывают на успешный ход восстановления и отправитель может передавать более агрессивно, увеличивая при необходимости inflight.

В заключение отправитель использует DeliveredData, inflight, SafeACK и другие переменные состояния PRR для расчёта SndCnt (локальная переменная, точно показывающая сколько байтов следует передать в ответ на каждый пакет ACK), а затем использует SndCnt для обновления cwnd.

Полный ход действия PRR при получении ACK показан ниже

```
if (DeliveredData is 0)
    Return

prrr_delivered += DeliveredData
inflight = (оценка остающихся в сети данных)
SafeACK = (SND.UNA растёт и новых потерь не указано)
if (inflight > ssthresh) {
    // Пропорциональное снижение скорости
    // Используется целочисленное деление с округлением вверх:
    #define DIV_ROUND_UP(n, d) (((n) + (d) - 1) / (d))
    out = DIV_ROUND_UP(prrr_delivered * ssthresh, RecoverFS)
    SndCnt = out - prrr_out
} else {
    // PRR-CRB применяется по умолчанию
    SndCnt = MAX(prrr_delivered - prrr_out, DeliveredData)
    if (SafeACK) {
        // PRR-SSRB при успешном ходе
        SndCnt += SMSS
    }
    // Попытка наверстать упущенное
    SndCnt = MIN(ssthresh - inflight, SndCnt)
```

```
}  
  
if (prrr_out is 0 AND SndCnt is 0) {  
    // Форсирование быстрого повтора в начале восстановления  
    SndCnt = SMSS  
}  
cwnd = inflight + SndCnt
```

После расчёта отправителем SndCnt и использования его для обновления cwnd отправитель передаёт больше данных. Отметим, что выбор данных для отправки (например, повтор или новые данные) выходит за рамки документа.

### 6.3. Этапы передачи

При любой передаче или повторе данных PRR выполняет указанное ниже действие.

```
prrr_out += (переданные данные)
```

### 6.4. Завершающие действия

Эпизод PRR заканчивается завершением процедуры быстрого восстановления или перед началом нового эпизода PRR в результате нового отклика на возникновение перегрузки. При завершении эпизода PRR устанавливается

```
cwnd = ssthresh
```

Отметим, что установка cwnd = ssthresh в некоторых случаях может приводить к всплеску отправляемых в сеть сегментов. Реализациям **рекомендуется** регулировать такие всплески трафика данных. Эта рекомендация соответствует современной практике сглаживания пиков (например, [PACING]), включая регулирование отправки после перезапуска из состояния бездействия.

## 7. Свойства

Перечисленные ниже свойства являются общими для PRR-CRB и PRR-SSRB, за исключением отмеченных случаев.

PRR пытается поддерживать синхронизацию отправки с ACK в процессе восстановления, включая пики потерь. В отличие от этого [RFC6675] может вызывать значительные всплески после пиков потерь. Обычно PRR равномерно распределяет сокращение окна в всего течение интервала RTT. Это может сглаживать пики трафика Internet и рассматриваться как своего рода мягкое регулирование. Гипотетически, любое регулирование может повышать вероятность чередования различных потоков, сокращая вероятность сжатия ACK и других явлений, способных вызывать пики. Однако количественная оценка таких эффектов не проводилась.

При низких потерях PRR будет сходиться точно к целевому окну, выбранному алгоритмом контроля перегрузок. Отметим, что по мере приближения отправителя к завершению восстановления, значение prr\_delivered будет стремиться к RecoverFS, а значение SndCnt будет рассчитываться так, что prr\_out будет стремиться к ssthresh.

Неявное сокращение окна в результате множественных изолированных потерь в процессе восстановления будет приводить к пропуску последующих сокращений. При небольшом числе потерь размер окна будет установлен в точности равным выбранному алгоритмом контроля перегрузок.

При пиках потерь раннее сокращение окна может быть отменено путём отправки дополнительных сегментов данных в ответ на пакеты ACK, приходящие позже в процессе восстановления. Отметим, что пока не будут отменены некоторые сокращения окна и приложения не будет заторможено, финальное значение inflight будет совпадать с ssthresh.

Используя любую из методов сокращения, PRR улучшает ситуацию, когда приложение тормозится, например, передающее приложение недостаточно быстро помещает данные в очередь передачи или получатель перестаёт продвигать окно приёма. Если приложение тормозится в начале восстановления, prr\_out будет отставать от суммарного объёма передачи, разрешённого SndCnt. Упущенные из-за задержки возможности отправки данных рассматриваются как отложенные сокращения, в частности, они делают разность prr\_delivered - prr\_out существенно положительной. Если приложение синхронизируется, пока отправитель ещё находится в состоянии восстановления, отправитель передаст всплеск (burst) для увеличения inflight, что бы достичь того состояния, которое было бы при отсутствии торможения. Хотя такой всплеск может негативно влиять на данный поток или другие потоки в «канале», именно это происходит всякий раз, когда приложение тормозится на часть RTT не в процессе восстановления. PRR делает такое торможение более однородным во всех состояниях. Изменение такого поведения выходит за рамки этого документа.

Метод PRR с привязкой сокращения менее чувствителен к ошибкам при оценке inflight. При восстановлении inflight по сути является оценкой на основе неполных сведений о потере или нарушении порядка данных, не указанных в SACK. В некоторых случаях inflight может включать существенные ошибки, например, значение недооценивается, если всплеск переупорядоченных данных будет сочтён потерей и промаркирован для повтора передачи. Если передача регулируется напрямую значением inflight, как в [RFC6675], разрывы в оценке inflight вызывают всплеск данных, который невозможно отменить после корректировки inflight новыми пакетами ACK. В части динамики PRR, inflight просто определяет, какой алгоритм, PRR или вариант сокращения применяется для расчёта SndCnt из DeliveredData. Несмотря на недооценку Winflight, алгоритмы различаются не более чем на 1 сегмент на ACK. После обновления оценки inflight они сходятся к одному размеру окна в конце восстановления.

При любых условиях и последовательности событий в процессе восстановления PRR-CRB строго ограничивает передачу данных, чтобы она не превысила объем данных, доставленных адресату. Strong Packet Conservation Bound (строгая привязка к сохранению пакетов) - это наиболее агрессивный алгоритм, не вызывающий дополнительных потерь в некоторых средах. Это обусловлено тем, что при наличии сохраняющейся очереди передачи в узком месте без кросс-трафика размер очереди в процессе восстановления будет меняться не более чем на +1/-1 из-за различий времени прибытия и выхода. Подробно этот вопрос рассматривается в Приложении A.

Хотя строгая привязка сохранения пакетов по многим причинам очень привлекательно, измерения (см. раздел 6 в [RFC6675]) показывают, что этот метод менее агрессивен и работает не так хорошо, как [RFC6675], где разрешены пики данных во время всплесков потерь. PRR-SSRB является компромиссом, позволяющим отправителю передать 1 дополнительный сегмент на ACK сверх Packet Conservation Bound (привязка сохранения пакетов), когда ACK показывает хороший ход восстановления без добавочных потерь. С точки зрения строгой привязки, PRR-SSRB открывает окно возможностей в процессе восстановления, однако при всплеске потерь этот метод менее агрессивен, чем [RFC6675].

## 8. Примеры

Этот раздел иллюстрирует поведение алгоритмов PRR и [RFC6675] демонстрируя различия в поведении для двух случаев: соединения с 1 потерей и всплеском из 15 последовательных потерь. В обоих случаях передаётся большой объем данных (без пауз в приложениях), применяется контроль перегрузок Reno [RFC5681], и установлено  $cwnd = FlightSize = inflight = 20$  сегментов, поэтому в начале восстановления будет значение  $ssthresh = 10$ . В обоих случаях применяется ускоренный повтор (Fast Retransmit) [RFC5681] и ограниченная передача (Limited Transmit) [RFC3042], поэтому отправитель передаёт 2 сегмента, после чего следует 1 повтор в ответ на первые 3 дубликата ACK после потерь.

На рисунках ниже показаны отклики на пакет ACK за первый круговой обход для двух алгоритмов восстановления при потере нулевого сегмента. Верхняя строка (ack#) показывает номер переданного сегмента, вызвавшего пакеты ACK, где X указывает потерянный сегмент. Строки cwnd и inflight указывают значения соответствующих переменных для алгоритмов после обработки каждого возвращённого пакета ACK, но до последующей передачи (повтора). Строка sent указывает число новых (N) или повторённых (R) данных, которые были переданы. Алгоритмы выбора данных для передачи выходят за рамки этого документа.

```

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

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

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

```

Рисунок 1.

В первом примере ACK#1 - ACK#19 содержат SACK для исходной отправки данных, в ACK#20 и ACK#21 содержатся SACK для ограниченной передачи, вызванной первым и вторым сегментами, подтверждёнными SACK, а ACK#22 содержит полное кумулятивное подтверждение ACK, охватывающее все данные вплоть до ограниченной передачи. ACK#22 завершает эпизод восстановления и, следовательно, - эпизод PRR.

Отметим, что оба алгоритма суммарно передают одинаковый объем данных и завершают эпизод восстановления с cwnd, соответствующим  $ssthresh = 20$ . В [RFC6675] возникает «половина окна тишины», а PRR распределяет сокращение окна по всему интервалу RTT.

Далее рассматривается случай с такими же начальными условиями, где теряется 15 первых пакетов (0-14). В результате прохождения оставшейся части пути отправителю возвращается лишь 5 пакетов 5 ACK.

```

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

```

Рисунок 2

В этой ситуации алгоритм [RFC6675] более агрессивен, поскольку при запуске Fast Retransmit (по ACK для сегмента 17) отправитель сразу передаёт повторно данные, достаточные для увеличения inflight до cwnd. Измерения (см. раздел 6 в [RFC6675]) показали, что [RFC6675] значительно превосходит version PRR [RFC6937] с использованием лишь PRR-CRB и некоторые похожие консервативные алгоритмы, которые тестировались. Это показывает, что фактические потери обычно превышают сокращение cwnd, определяемое алгоритмом контроля перегрузок.

При столь значительных потерях в первом интервале кругового обхода при восстановлении PRR использует PRR-CRB в соответствии с принципом сохранения пакетов. Поскольку общие потери ведут к тому, что inflight становится ниже  $ssthresh$ , данные передаются так, что общий объем отправки (prr\_out) соответствует общему объёму доставленных получателю данных, сообщённого пакетами ACK. Передача контролируется пределом отправки (prr\_delivered - prr\_out).

Хотя на рисунке это не показано, после того, как ускоренные повторы по ACK#17 доставлены и вызвали пакеты ACK, увеличивающие SND.UNA, PRR переходит в режим PRR-SSRB и увеличивает окно ровно на 1 сегмент по каждому ACK, пока inflight не достигнет значения  $ssthresh$  в процессе восстановления. При больших потерях, когда значение cwnd велико, PRR-SSRB восстанавливает потери в разы быстрее, чем PRR-CRB. Хотя увеличение окна в процессе восстановления представляется разумным, важно помнить, что на деле это менее агрессивно, нежели разрешает [RFC6675], где передаётся тот же объем дополнительных данных в одном всплеске как отклик на пакет ACK, вызвавший Fast Retransmit.

При меньших потерях (суммарно меньше FlightSize -  $ssthresh$ ) PRR-CRB и PRR-SSRB не вызываются, поскольку PRR остаётся в режиме пропорционального снижения скорости.

## 9. Адаптация PRR к другим транспортным протоколам

Основной алгоритм PRR и привязки сокращения можно адаптировать для любого транспорта, поддерживающего [RFC6675]. В одной из основных реализаций (Linux TCP) PRR является алгоритмом быстрого восстановления для принятых по умолчанию и поддерживаемых модулей контроля перегрузок с 2011 г. [First\_TCP\_PRR].

Эвристику SafeACK можно обобщить как любое подтверждение (ACK) повторной передачи, которое не ведёт к маркировке какого-либо другого сегмента для повторной передачи.

## 10. Измерения

Для [RFC6937] в документе [IMC11] проведены оценки [RFC3517] и экспериментальных версий PRR в рамках крупномасштабных измерений. На момент публикации этого документа использованные в исследовании устаревшие алгоритмы были уже исключены из реестров, что затрудняет сравнение без использования их. Интересующимся результатами измерений читателям следует обратиться к разделу 5 в [RFC6937] и статье IMC [IMC11].

## 11. Эксплуатационные вопросы

### 11.1. Поэтапное внедрение

PRR можно разворачивать постепенно, поскольку метод использует для доставки подтверждений и обнаружения потерь только существующие механизмы транспортных протоколов. PRR требует изменить лишь реализацию транспортного протокола на стороне отправителя, не требуя что-либо менять у получателей данных или на узлах сети. Это позволяет применяющим PRR отправителям данных корректно работать со всеми имеющимися получателями и сетями. PRR не требует участия маршрутизаторов, коммутаторов и иных устройств или внесения изменений в них.

### 11.2. Беспристрастность

Механизм PRR предназначен для обеспечения непристрастности алгоритма контроля перегрузок, с которым он применяется. PRR работает лишь во время отклика контроля перегрузок, такого как быстрое восстановление, или при постепенном снижении `swnd` в результате реакции TCP ECN [RFC3168], и принимает только краткосрочные решения на основе приёма подтверждений для плавной регулировки объёма остающихся в сети данных в течение эпизода так, чтобы в конце эпизода значение `swnd` было как можно ближе к порогу замедленного старта, заданному алгоритмом контроля перегрузок. PRR не меняет механизмов изменения `swnd` алгоритмом контроля перегрузок вне эпизодов отклика на перегрузку.

### 11.3. Защита сети от избыточных очередей и потери пакетов

В долгосрочной перспективе механизм PRR рассчитан на поддержку свойств постановки в очереди и потери пакетов, характерных для алгоритмов контроля перегрузок, с которыми он применяется. Как отмечено выше, PRR работает лишь в эпизодах откликов на перегрузку, таких как быстрое восстановление, или откликов на ECN, и принимает только краткосрочные решения для плавной регулировки объёма остающихся в сети данных, чтобы в конце эпизода размер окна `swnd` был как можно ближе к порогу замедленного старта (`ssthresh`), заданному алгоритмом контроля перегрузок.

В краткосрочном масштабе механизм PRR предназначен для снижения скорости потери пакетов по сравнению с прежними подходами, например, [RFC6675]. На высоком уровне механизм PRR основан на принципе сохранения пакетов и опирается на процесс самосинхронизации, насколько это возможно. В отличие от этого, в [RFC6675] один пакет ACK с опцией SACK, подразумевающий отсутствие большого объёма данных, может вызвать разрыв в оценке остающихся в сети данных, который может активировать механизм ускоренного повтора (Fast Retransmit) для отправки пикового объёма данных, значительно превышающего объём доставленных данных. PRR позволяет избежать таких всплесков, принимая решение о передаче на основе объёма доставленных, а не потерянных данных. Кроме того, как отмечено выше, PRR-SSRB менее агрессивен, чем [RFC6675] (передаёт меньше сегментов или делает это медленней), и обладает большей производительностью из-за снижения вероятности дополнительных потерь при восстановлении.

## 12. Взаимодействие с IANA

Этот документ не требует действий IANA.

## 13. Вопросы безопасности

PRR не изменяет профилей рисков транспортных протоколов

Разработчики, переводящие PRR с подсчёта байтов на подсчёт сегментов, должны быть осторожны в отношении атак с расщеплением ACK [Savage99], когда получатель подтверждает частичные сегменты с целью запутать отправителя при обработке перегрузки.

## 14. Литература

### 14.1. Нормативные документы

- [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>>.

## 14.2. Дополнительная литература

- [FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", SIGCOMM '96: Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 281-291, DOI 10.1145/248156.248181, August 1996, <<https://dl.acm.org/doi/10.1145/248156.248181>>.
- [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., Karim, T., Katz-Bassett, E., and R. Govindan, "An Internet-Wide Analysis of Traffic Policing", SIGCOMM '16: Proceedings of the 2016 ACM SIGCOMM Conference, pp. 468-482, DOI 10.1145/2934872.2934873, August 2016, <<https://doi.org/10.1145/2934872.2934873>>.
- [Hoe96Startup] Hoe, J., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", SIGCOMM '96: Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 270-280, DOI 10.1145/248156.248180, August 1996, <<https://doi.org/10.1145/248156.248180>>.
- [IMC11] Dukkkipati, N., Mathis, M., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", IMC '11: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, pp. 155-170, DOI 10.1145/2068816.2068832, November 2011, <<https://doi.org/10.1145/2068816.2068832>>.
- [Jacobson88] Jacobson, V., "Congestion Avoidance and Control", Symposium proceedings on Communications architectures and protocols (SIGCOMM '88), pp. 314-329, DOI 10.1145/52325.52356, August 1988, <<https://doi.org/10.1145/52325.52356>>.
- [PACING] Welzl, M., Eddy, W., Goel, V., and M. Tüxen, "Pacing in Transport Protocols", Work in Progress, Internet-Draft, draft-welzl-iccrp-pacing-03, 7 July 2025, <<https://datatracker.ietf.org/doc/html/draft-welzl-iccrp-pacing-03>>.
- [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., Dukkkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <<https://www.rfc-editor.org/info/rfc6937>>.
- [RFC9743] Duke, M., Ed. and G. Fairhurst, Ed., "Specifying New Congestion Control Algorithms", BCP 133, RFC 9743, DOI 10.17487/RFC9743, March 2025, <<https://www.rfc-editor.org/info/rfc9743>>.
- [Savage99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM SIGCOMM Computer Communication Review, vol. 29, no. 5, pp. 71-78, DOI 10.1145/505696.505704, October 1999, <<https://doi.org/10.1145/505696.505704>>.
- [TCP-RH] Mathis, M., Mahdavi, J., and J. Semke, "The Rate-Halving Algorithm for TCP Congestion Control", Work in Progress, Internet-Draft, draft-mathis-tcp-ratehalving-00, 30 August 1999, <<https://datatracker.ietf.org/doc/html/draft-mathis-tcp-ratehalving-00>>.

## Приложение А. Строгая привязка к сохранению пакетов

Привязка PRR-CRB основана на консервативной, философски чистой и эстетически привлекательной строгой привязке к сохранению пакетов (Strong Packet Conservation Bound), описанной здесь. Хотя механизм основан на принципе сохранения пакетов [Jacobson88], он отличается трактовкой пакетов, которые отсутствуют или считаются потерянными. При любых условиях и вариантах событий в процессе восстановления PRR-CRB строго ограничивает отправляемые данные - не больше объёма данных, доставленных получателю. Отметим, что последствия предполагаемых потерь учитываются при расчёте inflight, но не влияют на результат PRR-CRB, если inflight падает ниже ssthresh.

Этот алгоритм является наиболее агрессивным из числа не приводящих к дополнительным потерям в некоторых средах. Это свойство обусловлено тем, что при наличии в узком месте сохраняющейся очереди, через которую не передаётся другой трафик, очередь будет сохранять размер в течение всего периода восстановления с флуктуациями +1/-1 из-за разницы времени прибытия и выхода пакетов. Менее агрессивные алгоритмы приведут к сокращению очереди в узком месте, а более агрессивные вызовут рост очереди или дополнительные потери, если очередь переполнится.

Это свойство демонстрируется ниже с помощью мысленного эксперимента.

Представим сетевой путь с незначительными задержками в обоих направлениях, за исключением времени обработки и очереди в одном узком месте на пути пересылки. В частности, при «обслуживании» пакета из головы очереди в узком месте, указанные далее события происходят существенно быстрее, чем обработка пакета в узком месте: прибытие пакета к получателю, отправка получателем пакета ACK, который приходит к отправителю, обработка ACK потребителем с передачей некоторых данных и постановка данных в очередь узкого места пути.

Если `SndCnt = DeliveredData` и ничто не мешает передаче данных, очевидно, что данные, приходящие в очередь узкого места, будут в точности заменять данные, отправленные из головы этой очереди, поэтому размер очереди не будет меняться. При заполнении очереди с отбрасыванием хвоста она будет оставаться заполненной. Потери или нарушение порядка на пути ACK будет лишь вызывать более значительные колебания размера очереди без превышения максимального размера, независимо от соблюдения порядка данных (включая восстановление потерь из более раннего RTT). Более агрессивный алгоритм, отправляющий больше данных, переполнит очередь и вызовет потери, а при менее агрессивной передаче очередь будет не заполнена. Поэтому установка `SndCnt = DeliveredData` является наиболее агрессивным вариантом, не вызывающим дополнительных потерь в такой простой сети. Ослабление допущений (например, повышение достоверности задержек и добавление потоков, задержки ACK и т. п.) может увеличить флуктуации очереди, но не изменят базовое поведение.

Отметим, что алгоритм контроля перегрузок более широко трактует оптимальность, включая надлежащее распределение пропускной способности сети. Типичные протоколы контроля перегрузок, будут, вероятно, сокращать объем передаваемых данных по сравнению с ограничением сохранения пакетов, применяемым в PRR, что приведёт к сокращению фактического окна TCP до величины `ssthresh`.

## Благодарности

Этот документ частично основан на работе Janey C. Hoe (см. «Recovery from Multiple Packet Losses» в параграфе 3.2 [Hoe96Startup]), Matt Mathis, Jeff Semke и Jamshid Mahdavi [TCP-RH], а также обсуждениях с John Heffner.

Monia Ghobadi и Sivasankar Radhakrishnan помогли с анализом экспериментов. Ilpo Jarvinen проанализировал начальную реализацию. Mark Allman, Richard Scheffenegger, Markku Kojo, Mirja Kuehlewind, Gorry Fairhurst, Russ Housley, Paul Aitken, Daniele Ceccarelli, Mohamed Boucadair улучшили документ своими рецензиями и предложениями.

## Адреса авторов

**Matt Mathis**

Email: [matt.mathis@gmail.com](mailto:matt.mathis@gmail.com)

**Neal Cardwell**

Google, Inc.

Email: [ncardwell@google.com](mailto:ncardwell@google.com)

**Yuchung Cheng**

Google, Inc.

Email: [ycheng@google.com](mailto:ycheng@google.com)

**Nandita Dukkupati**

Google, Inc.

Email: [nanditad@google.com](mailto:nanditad@google.com)

## Перевод на русский язык

Николай Малых

[nmalykh@protokols.ru](mailto:nmalykh@protokols.ru)