Internet Engineering Task Force (IETF)
Request for Comments: 9002
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
I. Swett, Ed.
Google

May 2021

QUIC Loss Detection and Congestion Control

Обнаружение потерь и контроль перегрузок в QUIC

Аннотация

Этот документ описывает механизмы обнаружения потерь и контроля перегрузки для QUIC.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9002.

Авторские права

Авторские права (Copyright (c) 2021) принадлежат IETF Trust и лицам, указанным в качестве авторов документа. Все права защищены.

Этот документ является субъектом прав и ограничений, перечисленных в BCP 78 и IETF Trust Legal Provisions и относящихся к документам IETF (http://trustee.ietf.org/license-info), на момент публикации данного документа. Прочтите упомянутые документы внимательно, поскольку в них описаны права и ограничения, относящиеся к данному документу. Фрагменты программного кода, включённые в этот документ, распространяются в соответствии с упрощённой лицензией BSD, как указано в параграфе 4.e документа Trust Legal Provisions, без каких-либо гарантий (как указано в Simplified BSD License).

Оглавление

[1]Internet Engineering Task Force.
[2]Internet Engineering Steering Group.

1. Введение

QUIC является защищенным транспортным протоколом общего назначения, описанным в [QUIC-TRANSPORT]. Этот документ описывает механизмы обнаружения потерь и контроля перегрузок в QUIC.

2. Соглашения и определения

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (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.

Ниже приведены определения используемых в документе терминов.

Ack-eliciting frames - кадры с запросом подтверждения
: Все кадры, кроме ACK, PADDING, CONNECTION_CLOSE, считаются запрашивающими подтверждение.

Ack-eliciting packets - пакеты с запросом подтверждения
: Пакеты, содержащие кадры с запросом подтверждения, запрашивают ACK у получателя в интервале максимальной задержки подтверждения.

In-flight packets - пакеты в пути
: Пакеты считаются находящимися в пути (на лету - in flight), когда они запрашивают подтверждение или содержат кадр PADDING и были переданы, но ещё не подтверждены, не объявлены потерянными и не отброшены со старыми ключами.

3. Устройство машины передачи QUIC

Во всех передачах QUIC используется заголовок на уровне пакета, указывающий уровень шифрования и включающий порядковый номер пакета. Уровень шифрования указывает пространство порядковых номеров, как описано в параграфе 12.3 [QUIC-TRANSPORT]. Номера пакетов в одном пространстве не повторяются в течение работы соединения. Номера передаваемых пакетов монотонно увеличиваются для предотвращения неоднозначности. Допускается пропуск некоторых номеров.

Такой подход избавляет от неоднозначностей при повторной передаче пакета и устраняет в QUIC сложности обнаружения потери пакетов, присущие механизмам TCP.

Пакеты QUIC могут содержать несколько кадров различных типов. Механизмы восстановления обеспечивают для кадров и данных, требующих гарантированной доставки, подтверждение или обнаружение потери с повтором передачи в новом пакете. Типы кадров в пакете влияют на логику восстановления и контроля перегрузок.

- Все пакеты подтверждаются, хотя пакеты с кадрами, не запрашивающими подтверждения, подтверждаются лишь вместе с запросившими подтверждение пакетами.
- Пакеты с длинным заголовком, содержащие кадры CRYPTO, критически важны для производительности согласования QUIC и для них заданы короткие таймеры подтверждения.
- Пакеты, содержащие кадры помимо ACK или CONNECTION_CLOSE, учитываются в ограничениях контроля перегрузки и считаются находящимися в пути.
- Кадры PADDING заставляют учитывать пакеты в числе находящихся в пути, не вызывая напрямую отправки подтверждений.

4. Важные различия между QUIC и TCP

Читатели, знакомые с алгоритмами обнаружения потерь и контроля перегрузок в TCP, найдут здесь похожие функции. Однако между QUIC и TCP вносят свой вклад и в различия алгоритмов, кратко описанные в этом разделе.

4.1. Раздельные пространства порядковых номеров

QUIC использует своё пространство порядковых номеров для каждого уровня шифрования, за исключением того, что 0-RTT и все варианты ключей 1-RTT находятся в одном пространстве номеров. Раздельные пространства номеров гарантируют, что подтверждения пакетов с одним уровнем шифрования не вызовут ложных повторов пакетов с другим уровнем шифрования. Контроль перегрузок и измерение времени кругового обхода (round-trip time или RTT) являются одинаковыми во всех пространствах номеров.

4.2. Монотонный рост порядковых номеров

TCP объединяет порядок передачи отправителем с порядком приёма получателем, что ведёт к неоднозначности повтора передачи [RETRANSMISSION]. QUIC отделяет порядок передачи от порядка доставки. Номера пакетов указывают порядок передачи а порядок доставки определяется смещением потока в кадрах STREAM.

Номера пакетов QUIC строго возрастают в пространстве номеров и напрямую кодируют порядок отправки. Больший номер означает более позднюю передачу пакета, а меньший - более раннюю. При обнаружении потери пакета с запрашивающими подтверждение кадрами QUIC включает требуемые кадры в новый пакет с новым порядковым номером, устраняя неоднозначность в отношении подтверждаемых пакетов при получении ACK. Это позволяет более точно измерить RTT, легко обнаруживать ложные повторы передачи и возможность повсеместного применения таких механизмов, как Fast Retransmit (ускоренный повтор), на основе лишь номеров пакетов.

Такой подход существенно упрощает механизмы обнаружения потерь в QUIC. Большинство механизмов TCP неявно пытаются определить порядок передачи на основе порядковых номеров TCP, что является нетривиальной задачей особенно при недоступности временных меток TCP.

4.3. Более чёткая эпоха потерь

QUIC начинает эпоху потерь при обнаружении потери и завершает её при подтверждении любого пакета, переданного после начала эпохи. TCP ожидает заполнения пропуска в пространстве порядковых номеров, поэтому при потере нескольких сегментов подряд эпоха потерь может не завершиться в течение нескольких интервалов кругового обхода. Поскольку обоим протоколам следует уменьшать окно перегрузки лишь 1 раз в течение эпохи потерь, QUIC будет делать это один раз в течение кругового обхода, в котором были потери, а TCP может делать это 1 раз за несколько RTT.

4.4. Неотказуемость подтверждений

Кадры QUIC ACK содержат сведения, аналогичные содержащимся в селективных подтверждениях TCP (Selective Acknowledgment или SACK) [RFC2018]. Однако QUIC не позволяет отозвать подтверждение пакета, что существенно упрощает реализацию обеих сторон и расход памяти у отправителя.

4.5. Больше диапазонов ACK

QUIC поддерживает множество диапазонов ACK в отличие от трёх диапазонов SACK в TCP. В среде с высокими потерями это ускоряет восстановление, предотвращает ложные повторы и гарантирует «продвижение вперёд» без опоры на тайм-ауты.

4.6. Явная корректировка для задержанных подтверждений

Конечные точки QUIC измеряют задержку между приёмом пакета и отправкой соответствующего подтверждения, что позволяет партнёру более точно оценить RTT (см. параграф 13.2 в [QUIC-TRANSPORT]).

4.7. Тайм-аут зондов взамен RTO и TLP

QUIC использует тайм-аут проб (probe timeout или PTO, см. параграф 6.2) с таймером на основе расчёта тайм-аута повтора TCP (retransmission timeout или RTO, см. [RFC6298]). QUIC PTO включает максимальную ожидаемую задержку у партнёра вместо фиксированного минимального тайм-аута.

Как алгоритм обнаружения потерь RACK-TLP для TCP [RFC8985], QUIC не сокращает окно перегрузки по истечении PTO, поскольку потеря одного пакета в конце не указывает на постоянную перегрузку. Взамен QUIC сжимает окно перегрузки, когда объявляется сохраняющаяся перегрузка (см. параграф 7.6). Таким способом QUIC предотвращает неоправданное сужение окна перегрузки, избавляясь от потребности в механизмах корректировки, таких как F-RTO (Forward RTO-Recovery) [RFC5682]. Поскольку QUIC не сокращает окно перегрузки по истечении PTO, отправитель

QUIC не ограничен в передаче дополнительных пакетов, остающихся в пути по истечении PTO, если окно перегрузки не препятствует. Это происходит, когда отправитель ограничен приложением и PTO истекает. Это более энергичное поведение по сравнению с TCP RTO, когда приложение ограничено, но идентично ему, если нет ограничения.

QUIC позволяет пакетам зондирования временно выходить за пределы окна перегрузки по завершении таймера.

4.8. Минимальное окно перегрузки в 2 пакета

TCP использует минимальное окно перегрузки в 1 пакет. Однако потеря единственного пакета означает, что для восстановления отправитель должен ждать в течение PTO (параграф 6.2), что может быть много больше RTT. Отправка одного пакета с запросом подтверждения также повышает шансы дополнительной задержки при задержке подтверждения получателем.

Поэтому QUIC рекомендует окно перегрузки в 2 пакета. Хотя это повышает нагрузку на сеть, такой выбор считается безопасным, поскольку отправитель будет экспоненциально снижать скорость передачи при возникновении перегрузки (параграф 6.2).

4.9. Пакеты согласования не являются особыми

TCP считает потерю пакета SYN или SYN-ACK сохраняющейся перегрузкой и снижает размер окна перегрузки до одного пакета [RFC5681]. QUIC считает потерю пакета с данными согласования обычной потерей.

5. Оценка RTT

На высоком уровне конечная точка измеряет время между отправкой пакета и его подтверждением как выборку RTT. Конечная точка использует выборки RTT и полученные от партнёра задержки на хосте (см. параграф 13.2 в [QUIC-TRANSPORT]) для статистического описания RTT на сетевом пути. Конечная точка рассчитывает для каждого пути минимальное значение RTT за период времени (min_rtt), экспоненциально взвешенное среднее значение (smoothed_rtt) и среднее отклонение (variation) наблюдаемых выборок RTT (rttvar).

5.1. Генерация выборок RTT

Конечная точка создаёт выбору RTT при получении кадра ACK, соответствующего двум условиям:

- наибольший подтверждённый порядковый номер недавно подтверждён;
- хотя бы один из недавно подтверждённых пакетов запрашивал подтверждение (ack-eliciting).

Выборка RTT (latest_rtt) определяется временем с момента отправки подтверждённого пакета с наибольшим номером

```
latest_rtt = ack_time - send_time_of_largest_acked
```

Выборка RTT создаётся с использованием лишь подтверждённого пакета с наибольшим номером в полученном кадре ACK. Это обусловлено тем, что партнёр сообщает задержку подтверждения в кадре ACK лишь для пакета с наибольшим номером. Хотя сообщённая задержка не применяется при измерении RTT, она используется для корректировки выборки RTT в последующих расчётах smoothed_rtt и rttvar (параграф 5.3).

Для предотвращения множественной выборки RTT на один пакет кадр ACK **не следует** использовать для обновления оценок RTT, если он не является новым подтверждением подтверждённого пакета с наибольшим номером.

Выборку RTT **недопустимо** создавать при получении кадра ACK который не подтверждает хотя бы один пакет ack-eliciting. Партнёр обычно не передаёт кадр ACK, пока приняты лишь пакеты не требующие подтверждения. Поэтому кадр ACK с подтверждениями лишь не запрашивающих подтверждения пакетов может включать произвольно большое значение ACK Delay. Игнорирование таких кадров ACK предотвращает усложнение последующих расчётов smoothed_rtt и rttvar.

Отправитель может создавать несколько выборок RTT в течение RTT, когда получает в течение этого интервала несколько кадров ACK. Как отмечено в [RFC6298], это может приводить к неадекватной истории smoothed_rtt и rttvar. Обеспечение достаточной истории оценки RTT требует дальнейшего исследования.

5.2. Оценка min_rtt

Значение min_rtt указывает оценку отправителем минимального интервала RTT, наблюдавшегося для данного пути через сеть в течение определённого времени. В этом документе min_rtt используется механизмом обнаружения потерь для отклонения неправдоподобно малых выборок RTT.

В качестве min_rtt **должно** устанавливаться значение latest_rtt из первой выборки RTT. Значение min_rtt **должно** быть меньшим из min_rtt и latest_rtt (параграф 5.1) среди всех других измерений.

Конечная точка использует лишь локально измеряемое время при расчёте min_rtt и не учитывает задержку подтверждения, сообщённую партнёром. Это позволяет конечной точке установить нижнюю границу smoothed_rtt исключительно на основе своих наблюдений (см. параграф 5.3) и ограничит возможную недооценку, связанную с ошибочными сведениями о задержках от партнёра.

RTT для пути через сеть может меняться со временем. Если фактическое значение RTT для пути уменьшается, min_rtt корректируется незамедлительно по первой меньшей выборке. Если же RTT растёт, min_rtt не корректируется, позволяя включать будущие выборки RTT, которые меньше нового RTT, в расчёт smoothed_rtt.

Конечным точкам **следует** устанавливать в min_rtt новейшую выборку RTT после возникновения устойчивой перегрузки. Это предотвращает периодическое объявление сохраняющейся перегрузки при увеличении RTT, а также позволяет сбрасывать оценки min_rtt и smoothed_rtt после серьёзных нарушения в сети (см. параграф 5.3).

Конечные точки **могут** заново установить min_rtt в другой момент соединения, например, при малом объёме трафика и получении подтверждений с малой задержкой. Реализациям **не следует** обновлять min_rtt слишком часто, поскольку фактический минимум RTT на пути наблюдается нечасто.

5.3. Оценка smoothed_rtt и rttvar

Значение smoothed_rtt указывает экспоненциально взвешенное скользящее среднее значение выборки RTT конечной точкой, в rttvar — вариации в выборках RTT с использованием среднего отклонения.

В расчёте smoothed_rtt используются выборки RTT после их корректировки на основе задержки подтверждения. Эти задержки извлекаются из поля ACK Delay в кадрах ACK, как описано в параграфе 19.3 [QUIC-TRANSPORT]. Партнёр может сообщать о задержках подтверждения, превышающих max_ack_delay у него при согласовании (параграф 13.2.1 в [QUIC-TRANSPORT]). Для учёта этого конечной точке следует игнорировать max_ack_delay, пока согласование не подтверждено, как указано в параграфе 4.1.2 [QUIC-TLS]. Когда это произойдёт, такие большие задержки подтверждения вероятно не будут повторяться. Следовательно, конечная точка может использовать задержки подтверждения, не ограничивая их значением max_ack_delay и избегая ненужной переоценки RTT.

Отметим, что большая задержка подтверждения может привести к существенному росту smoothed_rtt, если имеются ошибки в измерении партнёром задержки подтверждения или оценке конечной точкой min_rtt. Поэтому до завершения согласования конечная точка **может** игнорировать выборку RTT, если учёт задержки подтверждения даёт значение меньше min_rtt.

После завершения согласования все значения задержки подтверждения от партнёра, превышающие его max_ack_delay, считаются непреднамеренными, но возможно повторяющимися задержками, такими как задержка в планировщике у партнёра или задержка предыдущих подтверждений. Избыточные задержки могут быть также связаны с несоответствием получателя. Поэтому такие задержки считаются частью задержки в пути и учитываются в RTT. При корректировке RTT с учётом полученной от партнёра задержки подтверждения конечная точка:

- **может** игнорировать задержку подтверждения для пакетов Initial, поскольку эти подтверждения партнёр не задерживает (параграф 13.2.1 в [QUIC-TRANSPORT]);
- **следует** игнорировать max_ack_delay у партнёра, пока согласование не подтверждено;
- **должна** использовать меньшее из значений задержки подтверждения и max_ack_delay у партнёра после подтверждения согласования;
- **недопустимо** вычитать задержку подтверждения из RTT , если это даёт значение меньше min_rtt (это ограничивает недооценку smoothed_rtt при ошибках партнёра.).

Кроме того, конечная точка может отложить обработку подтверждений, когда соответствующие ключи расшифровки недоступны. Например, клиент может получить подтверждение для пакета 0-RTT, которое он не может расшифровать, поскольку ключи расшифровки пакетов 1-RTT ещё недоступны. В таких случаях конечной точке **следует** вычитать локальную задержку из выборки RTT, пока согласование не подтверждено.

Подобно [RFC6298], значения smoothed_rtt и rttvar рассчитываются в соответствии с приведённым ниже описанием.

Конечная точка инициализирует измеритель RTT в процессе организации соединения и при его сбросе во время переноса соединения (см. параграф 9.4 в [QUIC-TRANSPORT]). До того как выборки RTT для нового пути станут доступны или при сбросе измерителя этот измеритель инициализируется с использованием начального RTT (см. параграф 6.2.2). Значения smoothed_rtt и rttvar инициализируются, как показано ниже (kInitialRtt содержит исходное значение RTT).

```
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
```

Выборки RTT для сетевого пути записываются в latest_rtt (см. параграф 5.1). При первой выборке RTT после инициализации измеритель сбрасывается с использованием этой выборки. Это обеспечивает исключение истории прошлых измерений. Пакеты, переданные по другому пути, не учитываются в выборке RTT на текущем пути, как указано в параграфе 9.4 [QUIC-TRANSPORT]. При первой выборке RTT после инициализации значения smoothed_rtt и rttvar устанавливаются, как показано ниже.

```
smoothed_rtt = latest_rtt
rttvar = latest_rtt / 2
```

При последующих выборках RTT значения smoothed_rtt и rttvar вычисляются в соответствии с псевдокодом.

```
ack_delay = decoded acknowledgment delay from ACK frame
if (handshake confirmed):
  ack_delay = min(ack_delay, max_ack_delay)
adjusted_rtt = latest_rtt
if (latest_rtt >= min_rtt + ack_delay):
  adjusted_rtt = latest_rtt - ack_delay
smoothed_rtt = 7/8 * smoothed_rtt + 1/8 * adjusted_rtt
rttvar_sample = abs(smoothed_rtt - adjusted_rtt)
rttvar = 3/4 * rttvar + 1/4 * rttvar_sample
```

6. Обнаружение потерь

Отправители QUIC используют подтверждения для обнаружения потери пакетов и PTO для гарантии получения подтверждений (см. параграф 6.2). В этом разделе приведено описание алгоритмов.

Если пакет потерян транспорту QUIC требуется восстановить потерю, например, путём повторной передачи данных, отправки обновлённого кадра или отбрасывания кадра (см. параграф 13.3 в [QUIC-TRANSPORT]). Обнаружение потерь происходит раздельно в каждом пространстве номеров в отличие от измерения RTT и контроля перегрузки, поскольку те являются свойствами пути, а обнаружение потерь связано с доступностью ключей.

6.1. Обнаружение на основе подтверждений

Обнаружение потерь на основе подтверждения реализует дух механизмов TCP Fast Retransmit [RFC5681], Early Retransmit [RFC5827], Forward Acknowledgment [FACK], восстановление потерь SACK [RFC6675] и RACK-TLP [RFC8985]. В этом параграфе представлен обзор реализации этих алгоритмов в QUIC.

Пакет считается потерянным при выполнении всех приведённых ниже условий.

- Пакет не подтверждён, находится в пути и был послан до подтверждённого пакета.
- Пакет был передан на kPacketThreshold пакетов раньше подтверждённого пакета (параграф 6.1.1) или достаточно давно (параграф 6.1.2).

Подтверждение указывает, что отправленный позже пакет был доставлен, а пороговые значения для пакетов и времени обеспечивают некоторую устойчивость к нарушению порядка пакетов.

Ложное объявление пакетов потерянными ведёт к ненужным повторам передачи и может снижать производительность в результате реакции механизма контроля перегрузок на потерю. Реализации могут обнаруживать ложные повторы и повышать пороги изменения порядка по пакетами или времени для снижения в будущем ложных повторов и фиксации потерь. Реализации с адаптивными временными порогами **могут** запускаться с меньшим начальным порогом нарушения порядка для минимизации задержки восстановления.

6.1.1. Порог по пакетам

Рекомендуемое для начального порога разупорядочения пакетов (kPacketThreshold) значение 3 выбрано на основе опыта обнаружения потерь TCP [RFC5681] [RFC6675]. Для сохранения сходства с TCP реализациям **не следует** использовать порог меньше 3 [RFC5681].

В некоторых сетях разупорядочение пакетов может быть более сильным, что ведёт к ложному обнаружению потерь отправителем. Кроме того, разупорядочение пакетов в QUIC встречается чаще, чем в TCP, поскольку элементы сети, способные наблюдать и менять порядок пакетов TCP, не могут делать этого для QUIC, так как номера пакетов QUIC зашифрованы. Алгоритмы, повышающие порог разупорядочения после ложного обнаружения потерь (такие как RACK [RFC8985]), оказались полезными в TCP и предполагается, что они будут не менее полезны в QUIC.

6.1.2. Порог по времени

Как только будет подтверждён самый поздний пакет в том же пространстве номеров, конечной точке **следует** объявить потерю более раннего пакета, если он был передан в течение заданного порогом интервала в прошлом. Для предотвращения слишком раннего обновления пакетов потерянными этот порог **должен** быть не меньше дискретности локального таймера, заданной параметром kGranularity.

```
max(kTimeThreshold * max(smoothed_rtt, latest_rtt), kGranularity)
```

Если пакеты, переданные до подтверждённого пакета с наибольшим номером, ещё не могут быть объявлены потерянными, таймер **следует** установить на оставшееся время. Использование max(smoothed_rtt, latest_rtt) защищает от двух случаев:

- последняя выборка RTT меньше сглаженного RTT (возможно из-за разупорядочения, когда подтверждение пошло по более короткому пути);
- последняя выборка RTT больше сглаженного RTT (возможно из-за устойчивого роста фактического RTT, когда сглаженное значение RTT ещё отстаёт).

Рекомендуемый порог по времени (kTimeThreshold) составляет 9/8 RTT, **рекомендуемая** дискретность таймера (kGranularity) - 1 мсек.

Примечание. TCP RACK [RFC8985] задаёт несколько больший порог, эквивалентный 5/4. Опыт работы с QUIC показывает, что порог 9/8 подходит лучше.

Реализации **могут** экспериментировать с абсолютными порогами, порогами из прежних соединений, адаптивными порогами или включением вариаций RTT. Меньший порог снижает устойчивость к разупорядочению и увеличивает число ложных повторов передачи, а больший повышает задержку обнаружения потерь.

6.2. Тайм-аут для зондов

Тайм-аут зондирования (Probe Timeout или PTO) вызывает отправку одной или двух пробных дейтаграмм, когда запросившие подтверждение пакеты не подтверждаются в течение ожидаемого интервала или сервер не может проверить адрес клиента. PTO позволяет восстанавливать соединение после потери «хвостовых» пакетов или подтверждений.

Как и в случае обнаружения потерь, PTO определяется по пространствам номеров пакетов, т. е. рассчитывается отдельно в каждом пространстве.

Завершение отсчёта таймера PTO не говорит о потере пакета и по этому событию **недопустимо** считать ранее неподтвержденные пакеты потерянными. При получении повторного подтверждения обнаружение потерь продолжается в соответствии с порогами для числа пакетов и времени, как описано в параграфе 6.1.

Алгоритм PTO в QUIC реализует функции алгоритмов Tail Loss Probe [RFC8985], RTO [RFC5681] и F-RTO для протокола TCP [RFC5682]. Расчёт тайм-аута основан на интервале TCP RTO [RFC6298].

6.2.1. Расчёт PTO

При передаче запрашивающего подтверждение пакета отправитель устанавливает таймер PTO

```
PTO = smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay
```

Интервал PTO задаёт время, в течение которого отправитель должен ждать подтверждения переданного пакета. Это время включает оценку RTT в сети (smoothed_rtt), вариации оценки (4*rttvar) и max_ack_delay для учёта максимального времени, на которое получатель может задержать отправку подтверждения.

При установке PTO для пространств номеров пакетов Initial или Handshake, для max_ack_delay при расчёте PTO принимается значение 0, поскольку ожидается немедленное подтверждение этих пакетов получателем (см. параграф 13.2.1 в [QUIC-TRANSPORT]).

Интервал PTO **должен** быть не меньше kGranularity во избежание немедленного завершения отсчёта.

Когда пакеты с запросом подтверждения их нескольких пространств номеров находятся в сети, таймер **должен** устанавливаться на более раннее значение из пространств номеров Initial и Handshake.

Конечной точке **недопустимо** устанавливать таймер PTO для пространства номеров данных приложения, пока согласование не подтверждено. Это предотвращает повторную передачу конечной точкой данных в пакетах, для которых у партнёра ещё нет ключей для обработки или подтверждения. Например, это может происходить, когда клиент передаёт серверу пакеты 0-RTT, не зная, может ли сервер расшифровать их. Это может происходить также при отправке сервером пакетов 1-RTT до подтверждения проверки клиентом сертификата сервера и возможности чтения пакетов 1-RTT.

Отправителю **следует** перезапускать таймер PTO при каждой отправке или подтверждении запросившего подтверждение пакета, а также при отбрасывании ключей Initial или Handshake (параграф 4.9 в [QUIC-TLS]). Это гарантирует установку PTO на основе последней оценки RTT и для нужного пространства номеров пакетов.

При завершении отсчёта таймера PTO отсрочка PTO **должна** увеличиваться, в результате чего для PTO устанавливается удвоенное текущее значение. Коэффициент отсрочки PTO сбрасывается при получении подтверждения, за исключением одного случая. Серверу может потребоваться больше времени для ответа в процессе согласования, нежели в других случаях. Для защиты такого сервера от повторных зондов клиента отсрочка PTO не сбрасывается у клиента, которые ещё не уверен в том, что сервер завершил проверку его адреса. Т. е. клиент не сбрасывает коэффициент отсрочки PTO в ответ на получение подтверждений пакетов Initial.

Такое экспоненциальное снижение скорости передачи отправителя важно, поскольку последовательные PTO могут быть вызваны потерей пакетов или подтверждений из-за серьёзной перегрузки. Даже при наличии в сети (in flight) пакетов из нескольких пространств номеров экспоненциальное увеличение PTO выполняется во всех пространствах для предотвращения чрезмерной нагрузки на сеть. Например, тайм-аут в пространстве номеров Initial удваивает продолжительность ожидания в пространстве номеров Handshake.

Общая продолжительность времени, в течение которого истекает срок последовательных PTO, ограничен тайм-аутом бездействия.

Установка таймера PTO **недопустима**, если установлен таймер для порога обнаружения потерь по времени (см. параграф 6.1.2). Таймер, установленный для порога обнаружения потерь по времени, будет завершаться в большинстве случаев раньше таймера PTO и вероятность ложного повтора данных будет меньше.

6.2.2. Согласование и новые пути

Возобновляемые в той же сети соединения **могут** использовать финальное сглаженное значение RTT из прежнего соединения в качестве начального RTT. Если прежнего значения RTT нет, в качестве начального RTT RTT следует устанавливать значение 333 мсек. Это ведёт к согласованию, начинающегося с PTO в 1 секунду, как рекомендовано для начального RTO в TCP (см. раздел 2 в [RFC6298]).

Соединение **может** использовать задержку между передачей PATH_CHALLENGE и приёмом PATH_RESPONSE для установки начального RTT (см. kInitialRtt в Приложении A.2) на новом пути, но **не следует** считать её выборкой RTT.

Когда ключи Initial и Handshake отброшены (параграф 6.4), никакие пакеты Initial и Handshake не могут быть подтверждены, поэтому они исключаются из числа байтов, находящихся в пути. При отбрасывании ключей Initial или Handshake таймеры PTO и обнаружения потерь **должны** сбрасываться, поскольку отбрасывание ключей указывает продвижение вперёд и таймер обнаружения потерь может быть установлен для пространства номеров, которые сейчас отброшены.

6.2.2.1. До проверки адреса

Пока сервер не проверил адрес клиента на пути, объем передаваемых им данных ограничен троекратным размером принятых данных, как указано в параграфе 8.1 [QUIC-TRANSPORT]. Если дополнительные данные не могут быть переданы, таймер PTO на сервере **недопустимо** активировать, пока не будут получены дейтаграммы от клиента, поскольку пакеты, передаваемые для PTO, учитываются в пороге антиусиления.

При получении сервером дейтаграммы от клиента порог антиусиления повышается и сервер сбрасывает таймер PTO. Если после этого таймер PTO будет установлен на прошлое время, он считается сработавшим сразу же. Это предотвращает передачу пакетов 1-RTT до пакетов, которые критически важны для согласования. В частности, это может происходить, когда сервер воспринимает 0-RTT, но не может подтвердить адрес клиента.

Поскольку сервер может быть заблокирован до приёма новых дейтаграмм от клиента, ответственность за отправку этих дейтаграмм ложится на клиента, пока у него не будет уверенности в том, что сервер завершил проверку его адреса (см. раздел 8 в [QUIC-TRANSPORT]). Т. е. клиент **должен** установить таймер PTO, если он не получил подтверждения какого-либо из своих пакетов Handshake и согласование не подтверждено (см. параграф 4.1.2 в [QUIC-TLS]), даже при отсутствии находящихся в пути пакетов. При срабатывании PTO клиент **должен** передать пакет Handshake при наличии ключей Handshake, в противном случае он **должен** передать пакет Initial в дейтаграмме UDP, содержимое (payload) которой не меньше 1200 байтов.

6.2.3. Ускоренное завершение согласования

При получении пакета Initial с дубликатом данных CRYPTO сервер может предположить, что клиент не получил все данные CRYPTO, переданные в пакетах Initial, или оценка RTT клиентом слишком мала. При получении клиентом пакетов Handshake или 1-RTT до обретения ключей Handshake он может предположить, что некоторые или все пакеты Initial от сервера были потеряны.

Чтобы ускорить завершение согласования в таких условиях, конечная точка **может** ограниченное число раз за соединение передать пакет с неподтвержденными данными CRYPTO до завершения PTO с учётом ограничений проверки адреса (параграф 8.1 в [QUIC-TRANSPORT]). Выполнения этого не более 1 раза для каждого соединения достаточно для быстрого восстановления при потере одного пакета. Конечная точка, всегда повторяющая пакеты в ответ на получение пакета, который она не может обработать, рискует создать бесконечный обмен пакетами.

Конечные точки могут также объединять пакеты (см. параграф 12.2 в [QUIC-TRANSPORT]), чтобы каждая дейтаграмма запрашивала хотя бы одно подтверждение. Например, клиент может объединить пакет Initial, содержащий кадры PING и PADDING, с пакетом данных 0-RTT, а сервер может объединить пакет Initial, содержащий кадр PING, с одним или несколькими пакетами в своей первой отправке.

6.2.4. Отправка пакетов зондирования

При завершении отсчёта таймера PTO отправитель **должен** передать в качестве зонда по меньшей мере один пакет с запросом подтверждения в пространстве номеров. Конечная точка **может** передать до 2 полноразмерных дейтаграмм с запрашивающими подтверждение пакетами для предотвращения дорогостоящих последовательных PTO в результате потери одной дейтаграммы или для передачи данных из нескольких пространств номеров. Все пробные пакеты, переданные в течение PTO **должны** запрашивать подтверждение.

В дополнение к передаче данных в пространстве номеров, для которого завершился отсчёт таймера, отправителю **следует** передать запрашивающие подтверждение пакеты из других пространств номеров с находящимися в пути данными, объединяя пакеты по возможности. Это особенно ценно при наличии у сервера находящихся в пути данных Initial и Handshake сразу или при наличии у клиента находящихся в пути данных Handshake и Application Data сразу, поскольку партнёр может иметь ключи приёма лишь для одного из двух пространств номеров.

Если отправитель хочет запросить быстрое подтверждение в интервале PTO, он может пропустить номер пакета для предотвращения задержки подтверждения.

Конечной точке **следует** включать новые данные в пакеты, передаваемые по истечении PTO. При отсутствии новых данных **можно** передать отправленные ранее. Реализации **могут** использовать дополнительную стратегию определения содержимого тестовых пакетов, включая отправку новых или повторных данных на основе приоритетов приложения.

У отправителя может не быть новых или уже отправленных данных для передачи. Рассмотрим в качестве примера последовательность событий: новые данные приложения переданы в кадре STREAM, эти данные сочтены потерянными, повторены в новом пакете, а затем исходна передача была подтверждена. Когда нет данных для передачи, отправителю **следует** передать PING или другой кадр с запросом подтверждения в одном пакете, снова запуская таймер PTO.

Вместо отправки запрашивающего подтверждение пакета отправитель **может** пометить любые остающиеся в пути пакеты как потерянные. Это позволяет избежать отправки дополнительного пакета, но повышает риск слишком активного объявления потерь, приводящего к неоправданному снижению скорости контроллером перегрузки.

Последовательные интервалы PTO увеличиваются экспоненциально и в результате экспоненциально растёт задержка восстановления соединения, поскольку отбрасывание пакетов в сети продолжается. Передача двух пакетов по истечении PTO повышает устойчивость к отбрасыванию пакетов, снижая тем самым вероятность последовательных тайм-аутов PTO.

Когда отсчёт таймера PTO завершается несколько раз и новые данные не могут быть переданы, реализация должна выбирать между отправкой каждый раз одного или разного содержимого. Отправка одного содержимого может быть проще и обеспечивает первоочередное прибытие кадров с высшим приоритетом. Передача разного содержимого снижает вероятность ложных повторов отправки.

6.3. Обработка пакетов Retry

Пакет Retry заставляет клиента передать новый пакет Initial, фактически запуская процесс соединения снова. Пакет Retry указывает, что пакет Initial был получен, но не обработан. Пакет Retry не может считаться подтверждением, поскольку он не говорит об обработке пакета и не задаёт порядковый номер.

Клиент, получающий пакет Retry, сбрасывает статус контроля перегрузок и восстановления потерь, включая сброс ожидающих таймеров. Другие состояния соединения, в частности, сообщения криптографического согласования, сохраняются (см. параграф 17.2.5 в [QUIC-TRANSPORT]).

Клиент **может** рассчитать RTT до сервера как интервал между отправкой первого пакета Initial и получением пакета Retry или Version Negotiation. Клиент **может** установить это значение вместо принятого по умолчанию начального RTT.

6.4. Отбрасывание ключей и состояние пакета

Когда ключи защиты пакетов Initial и Handshake отброшены (см. параграф 4.9 в [QUIC-TLS]), переданные с этими ключами пакеты не могут быть подтверждены, поскольку обработать подтверждения невозможно. Отправитель **должен** отбросить все состояния восстановления, связанные с этими пакетами, и **должен** исключить пакеты из числа находящихся в пути байтов.

Конечные точки останавливают передачу и приём пакетов Initial после начала обмена пакетами Handshake (см. параграф 17.2.2.1 в [QUIC-TRANSPORT]). В этот момент отбрасывается состояние восстановления для находящихся в пути пакетов Initial.

Когда данные 0-RTT отвергаются, состояния восстановления для находящихся в пути пакетов 0-RTT отбрасываются. Если сервер воспринимает 0-RTT, но не буферизует пакеты 0-RTT, прибывшие до пакетов Initial, ранние пакеты 0-RTT будут объявлены потерянными, но предполагается, что это будет происходить нечасто.

Предполагается, что ключи будут отброшены в течение некоторого времени после того, как все зашифрованным с ними пакеты будут подтверждены или объявлены потерянными. Однако секреты Initial и Handshake отбрасываются как только станут доступны ключи Handshake и 1-RTT для клиента и сервера (см. параграф 4.9.1 в [QUIC-TLS]).

7. Контроль перегрузки

В этом документе описано контроллер перегрузок на стороне отправителя QUIC, похожий на TCP NewReno [RFC6582]. Сигналы, предоставляемые протоколом QUIC для контроля перегрузок являются базовыми и предназначены для разных механизмов на стороне отправителя, который в одностороннем порядке может выбрать другой механизм,

например CUBIC [RFC8312]. При использовании отправителем иного механизма, выбранный контроллер **должен** соответствовать рекомендациям параграфа 3.1 в [RFC8085].

Подобно TCP, пакеты, содержащие лишь кадры ACK не учитываются в числе находящихся в пути байтов и для них не применяется контроль перегрузки. Однако в отличие от TCP, QUIC может обнаруживать потери таких кадров и **может** использовать эти сведения для настройки контроллера перегрузки или скорости передачи пакетов, содержащих только ACK, но данный документ не описывает механизмов для этого.

Контроллер перегрузки работает на уровне пути, поэтому передаваемые по другому пути пакеты не влияют на контроллер текущего пути, как описано в параграфе 9.4 [QUIC-TRANSPORT]. Описанный в документе алгоритм задаёт и применяет окно перегрузки в контроллере, выраженное в байтах. Конечной точке **недопустимо** передавать пакеты, если это сделает значение bytes_in_flight (см. Приложение B.2) больше окна перегрузки, за исключением случаев отправки пакета по тайм-ауту PTO (см. параграф 6.2) или при входе в режим восстановления (см. параграф 7.3.2).

7.1. Явное указание перегрузки

Если для пути подтверждено явное уведомление о перегрузке (Explicit Congestion Notification или ECN) [RFC3168] [RFC8311], QUIC считает код CE[1] в заголовке IP сигналом перегрузки. Этот документ задаёт реакцию конечных точек на увеличение полученного от партнёра значения ECN-CE (см. параграф 13.4.2 в [QUIC-TRANSPORT]).

7.2. Начальное и минимальное окно перегрузки

QUIC начинает каждое соединение в режиме замедленного старта с установкой начального значения окна перегрузки. Конечным точкам **следует** устанавливать начальное окно перегрузки в 10 максимальных размеров дейтаграмм (max_datagram_size), при этом ограничивая окно значением 14720 байтов или удвоенным размером максимальной дейтаграммы. Это соответствует анализу и рекомендациям [RFC6928], увеличивая предельный размер с учётом 8-байтового заголовка UDP вместо 20-байтового заголовка TCP. Если максимальный размер дейтаграмм меняется в процессе работы соединения, начальное окно перегрузки **следует** пересчитать в соответствии с изменением. Если максимальный размер дейтаграмм уменьшается для завершения согласования, окно перегрузки **следует** установить в соответствии с новым значением начального окна перегрузки.

До проверки адреса клиента сервер может быть дополнительно ограничен пределом антиусиления, как описано в параграфе 8.1 [QUIC-TRANSPORT]. Хотя предел антиусиления может препятствовать полному использованию окна перегрузки и замедлять расширение этого окна, он не влияет на окно перегрузки напрямую.

Минимальным окном перегрузки является меньшее из значений окна перегрузки, которое может быть установлено в ответ на потерю, рост полученного от партнёра значения ECN-CE или сохраняющуюся перегрузку. **Рекомендуется** удвоенное значение max_datagram_size.

7.3. Состояния контроля перегрузки

Контроллер перегрузок NewReno, описанный в этом документе, имеет 3 состояния, показанных на рисунке 1.

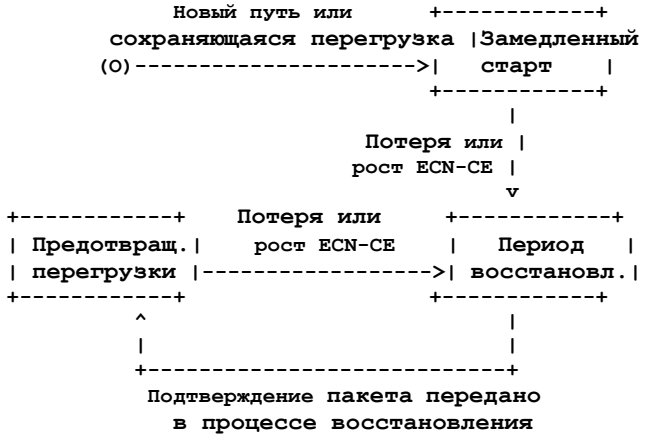

Рисунок 1. Смена состояний контроля перегрузок.

Эти состояния и переходы между ними описаны в последующих параграфах.

7.3.1. Замедленный старт

Отправитель NewReno находится в режиме замедленного старта каждый раз, когда окно перегрузки ниже порога замедленного старта. Отправитель начинает в этом режиме, поскольку пород замедленного старта инициализируется бесконечным значением. Когда отправитель находится в режиме замедленного старта, окно перегрузки увеличивается на число подтверждённых байтов при обработке каждого подтверждения. Это ведёт к экспоненциальному росту окна перегрузки.

Отправитель **должен** выходить из режима замедленного старта при потере пакета или росте полученного от партнёра значения ECN-CE. Отправитель возвращается в режим замедленного старта всякий раз, когда окно перегрузки становится меньше порога замедленного старта, что происходит лишь при возникновении сохраняющейся перегрузки.

7.3.2. Восстановление

Отправитель NewReno входит в период восстановления при обнаружении потери пакета или росте полученного от партнёра значения ECN-CE. Отправитель, уже находящийся в периоде восстановления, остаётся в нем без входа заново. При входе в период восстановления отправитель **должен** установить для порога замедленного старта

[1]Congestion Experienced - наступила перегрузка.

половину размера окна перегрузки при обнаружении потери. Для окна перегрузки **должно** быть установлено сниженное значение порога замедленного старта при выходе из периода восстановления.

Реализация **может** сократить окно перегрузки сразу после входа в период восстановления или с помощью других механизмов, таких как пропорциональное снижение скорости (Proportional Rate Reduction [PRR]), для более постепенного сокращения. Если окно перегрузки сокращается сразу же, перед сокращением может быть передан 1 пакет. Это ускоряет восстановление, если потерянный пакет передаётся повторно, и похоже на TCP, как описано в разделе 5 [RFC6675].

Период восстановления направлен на ограничение сокращения окна перегрузки один раз за период кругового обхода. Поэтому в период восстановления окно перегрузки не сокращается в ответ на новые потери и рост ECN-CE. Период восстановления заканчивается и отправитель переходит в режим предотвращения перегрузки, когда подтверждаются пакеты, переданные в период восстановления. Это несколько отличается от определения восстановления в TCP, которое завершается при подтверждении сегмента, с потери которого восстановления началось [RFC5681].

7.3.3. Предотвращение перегрузки

Отправитель NewReno находится в режиме предотвращения перегрузки все время, когда окно перегрузки не меньше порога замедленного старта и нет периода восстановления. Отправитель в этом режиме использует аддитивное увеличение и мультипликативное сокращение (Additive Increase Multiplicative Decrease или AIMD), которое **должно** ограничить расширение окна перегрузки величиной не более размера одной максимальной дейтаграммы для каждого подтверждённого окна перегрузки. Отправитель переходит из режима предотвращения перегрузки в период восстановления при потере пакета или росте полученного от партнёра значения ECN-CE.

7.4. Игнорирование потери нерасшифровываемых пакетов

В процессе согласования некоторые ключи защиты пакетов могут быть недоступны в момент прибытия пакета и получатель может принять решение об отбрасывании пакета. В частности, пакеты Handshake и 0-RTT невозможно обработать до получения пакетов Initial, а 1-RTT - до завершения согласования. Конечные точки **могут** игнорировать потерю пакетов Handshake, 0-RTT, 1-RTT, которые приходят к партнёру до обретения ключей для их обработки. Конечным точкам **недопустимо** игнорировать потерю пакетов, переданных после наиболее раннего подтверждённого пакета из того же пространства номеров.

7.5. Тайм-аут зондирования

Контроллеру перегрузки **недопустимо** блокировать пакеты зондирования. Однако отправитель **должен** учитывать эти пакеты как находящиеся в пути, поскольку они увеличивают нагрузку в сети, но их потеря не учитывается. Отметим, что отправка пробных пакетов может приводить к выходу числа находящихся в пути байтов за пределы окна перегрузки, пока не получено подтверждение доставки или потери пакетов.

7.6. Сохраняющаяся перегрузка

Когда отправитель констатирует потерю всех пакетов в течение достаточно долгого интервала, считается, что в сети наблюдается сохраняющаяся (постоянная) перегрузка.

7.6.1. Продолжительность

Продолжительность сохраняющейся перегрузки рассчитывается как

```
(smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay) *
    kPersistentCongestionThreshold
```

В отличие от расчёта PTO в параграфе 6.2, эта продолжительность включает max_ack_delay безотносительно к пространству номеров, где были отмечены потери. Эта продолжительность позволяет отправителю передать до фиксации сохраняющейся перегрузки (включая отклик на завершение PTO) столько пакетов, сколько разрешено для TCP с Tail Loss Probes [RFC8985] и RTO [RFC5681].

Большее значение kPersistentCongestionThreshold делает отправителя менее восприимчивым к сохраняющейся перегрузке в сети, что может привести к избыточной передаче в перегруженную сеть. Слишком малое значение может приводить к неоправданному решению о сохраняющейся перегрузке, снижающему скорость передачи у отправителя. Для kPersistentCongestionThreshold **рекомендуется** значение 3, обеспечивающее поведение, близкое к поведению отправителя TCP, объявляющего RTO после двух TLP.

В этой схеме не применяются последовательные события PTO для фиксации сохраняющейся перегрузки, поскольку на PTO влияет картина трафика приложения. Например, отправитель, передающий небольшой объем данных, перезапускает таймер PTO при каждой передаче, что может приводить к тому, что таймер PTO будет срабатывать редко даже при отсутствии подтверждений. Использование продолжительности позволяет отправителю фиксировать сохраняющуюся перегрузку без учёта таймера PTO.

7.6.2. Фиксация сохраняющейся перегрузки

Отправитель констатирует сохраняющуюся перегрузку после приёма подтверждения, если объявлены потерянными 2 пакета с запросом подтверждения и выполняются указанные ниже условия:

- во всех пространствах номеров не подтверждён ни один из пакетов, переданных между этими 2 пакетами;
- время между отправкой этих 2 пакетов больше продолжительности постоянной перегрузки (параграф 7.6.1);
- при отправке этих двух пакетов имелась предыдущая выборка RTT.

Эти 2 пакета **должны** быть запрашивающими подтверждение, поскольку получатель обязан подтверждать лишь такие пакеты в интервале максимальной задержки подтверждения (см. параграф 13.2 в [QUIC-TRANSPORT]).

Период постоянной перегрузки **недопустимо** начинать, пока не сделана хотя бы одна выборка RTT. До первой выборки RTT отправитель активирует таймер PTO на основе начального значения RTT (параграф 6.2.2), которое может

быть существенно больше фактического RTT. Требование предшествующей выборки RTT предотвращает фиксацию сохраняющейся перегрузки отправителем на основании слишком малого числа пакетов.

Поскольку с перегрузкой сети пространства номеров пакетов не связаны, при такой перегрузке **следует** учитывать пакеты из всех пространств номеров. Отправитель, у которого нет состояния для всех пространств номеров или реализация не может сравнивать время отправки из разных пространств, **может** использовать состояние для пространства, в котором был подтверждён пакет. Это может приводить к ложному объявлению сохраняющейся перегрузки, но не ведёт к пропуску обнаружения такой перегрузки.

Когда объявлена сохраняющаяся перегрузка, окно перегрузки у отправителя **должно** сокращаться до минимального размера (kMinimumWindow), как в реакции отправителя TCP на RTO [RFC5681].

7.6.3. Пример

Приведённый пример иллюстрирует фиксацию сохраняющейся перегрузки отправителем. Предположим, что

```
smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay = 2
kPersistentCongestionThreshold = 3
```

Рассмотрим приведённую в таблице 1 последовательность событий

Таблица 1.

Время	***Действие***
t=0	Передан пакет 1 (данные приложения)
t=1	Передан пакет 2 (данные приложения)
t=1,2	Получено подтверждение пакета 1
t=2	Передан пакет 3 (данные приложения)
t=3	Передан пакет 4 (данные приложения)
t=4	Передан пакет 5 (данные приложения)
t=5	Передан пакет 6 (данные приложения)
t=6	Передан пакет 7 (данные приложения)
t=8	Передан пакет 8 (PTO 1)
t=12	Передан пакет 9 (PTO 2)
t=12,2	Получено подтверждение пакета 9

Пакеты 2 - 8 объявляются потерянными, когда приходит подтверждение пакета 9 в момент t = 12,2. Период перегрузки рассчитывается как время между самым старым и самым новым потерянными пакетами 8 - 1 = 7. Продолжительность сохраняющейся перегрузки составляет 2 * 3 = 6. Поскольку порог был достигнут и ни один пакет между самым старым и самым новым потерянными пакетами не был подтверждён, считается, что в сети сохраняется перегрузка.

Хотя в примере показан тайм-аут PTO, он не требуется для обнаружения сохраняющейся перегрузки.

7.7. Темп передачи

Отправителю **следует** управлять передачей пакетов на основе входных данных контроллера перегрузки. Отправка нескольких пакетов в сеть без задержки между ними создаёт пик пакетов, который может вызвать кратковременную перегрузку и потери. Отправитель **должен** контролировать такие пики и ему **следует** ограничивать их начальным окном перегрузки (см. параграф 7.2). Отправитель, знающий, что путь к получателю способен принимать большие пики, **может** установить более высокое значение.

Реализации следует обеспечивать контроллеру перегрузки поведение, похожее на задание темпа (pacer). Например, можно дополнять контроллер перегрузок и управлять доступностью окна перегрузки или ускоренно передавать пакеты от контроллера перегрузки.

Для эффективного восстановления потерь важна своевременная доставка кадров ACK. Для предотвращения задержки их доставки партнёру **следует** передавать пакеты, содержащие только кадры ACK без управления темпом. Конечные точки могут реализовать управление темпом передачи по своему усмотрению. Идеальный отправитель передаёт пакеты строго равномерно по времени. Для контроллера перегрузки на основе окна, подобного рассмотренному здесь, эту скорость можно рассчитать, усредняя окно перегрузки в интервале RTT. Скорость в байтах за единицу времени при congestion_window, заданном в байтах, будет определяться выражением

```
rate = N * congestion_window / smoothed_rtt
```

Можно также задать скорость межпакетным интервалом в единицах времени

```
interval = ( smoothed_rtt * packet_size / congestion_window ) / N
```

Использование небольшого (но не меньше 1) значения N (например, 1,25) гарантирует, что изменения RTT не будут приводить к неполному использованию окна перегрузки.

Практические соображения, такие как пакетизация, задержки планирования и вычислительная эффективность, могут вынудить сервер отклониться от этой скорости в течение интервалов много меньше RTT.

Одной из возможных стратегий управления темпом является применение алгоритма «текущего ведра» (leaky bucket), объем которого ограничен максимальным размером пиков, а скорость вытекания задаётся приведённой функцией.

7.8. Неполное использование окна перегрузки

Когда число находящихся в пути байтов меньше окна перегрузки и скорость отправки не задаётся стимулятором, возможно неполное использование окна перегрузки. Это может быть результатом неверно выбранных ограничений данных приложением или управления потоком данных. В таких случаях **не следует** увеличивать окно перегрузки в режиме замедленного старта или предотвращения перегрузки.

Отправитель, управляющий темпом передачи (параграф 7.7) может задерживать отправку пакетов и в результате не использовать полностью окно перегрузки. Отправителю **не следует** считать себя ограниченным приложением, если он может полностью использовать окно перегрузки без вносимой управлением темпом задержки. Отправитель **может**

реализовать дополнительные механизмы для обновления окна перегрузки после периодов неполного использования, такие как предложены для TCP в [RFC7661].

8. Вопросы безопасности

8.1. Сигналы потерь и перегрузки

Обнаружение потерь и контроль перегрузки основаны на сигналах, таких как задержка, потери, маркировка ECN, от неаутентифицированных объектов. Злоумышленник может заставить конечные точки снизить скорость передачи, манипулируя этими сигналами путём отбрасывания пакетов, изменения задержки на пути или подмены кодов ECN.

8.2. Анализ трафика

Пакеты, содержащие лишь кадры ACK, можно определить эвристически по их размеру. Картины подтверждений могут раскрывать информацию о характеристиках канала и поведении приложения. Для снижения утечки сведений конечная точка может объединять подтверждения с другими кадрами или добавлять кадры PADDING ценой снижения производительности.

8.3. Неверная маркировка ECN

Получатель может передавать некорректную маркировку ECN для изменения реакции отправителя на перегрузку. Подавление сведений о маркировке ECN-CE может вынудить отправителя к увеличению скорости передачи, что может приводить к перегрузке и потерям.

Отправитель может обнаружить подавление сведений, случайно помечая передаваемые пакеты маркером ECN-CE. Если для пакета, переданного с маркировкой ECN-CE, не сообщается о наличии маркера CE при его подтверждении, отправитель может отключить ECN для пути, не устанавливая коды ECT в последующих пакетах для этого пути [RFC3168].

Сообщение о дополнительных маркерах ECN-CE вынудит отправителя снизить скорость передачи, что похоже на эффект анонсирования сниженного порога управления потоком данных, поэтому не даёт каких-либо преимуществ.

Конечные точки сами выбирают контроллер перегрузки. Контроллеры реагируют на сведения о ECN-CE снижением скорости, но отклики могут различаться. Маркировку можно считать эквивалентом потери [RFC3168], но возможны и другие отклики, например, [RFC8511] или [RFC8311].

9. Литература

9.1. Нормативные документы

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[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>.

[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>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[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>.

9.2. Дополнительная литература

[FACK] Mathis, M. and J. Mahdavi, "Forward acknowledgement: Refining TCP Congestion Control", ACM SIGCOMM Computer Communication Review, DOI 10.1145/248157.248181, August 1996, <https://doi.org/10.1145/248157.248181>.

[PRR] 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>.

[RETRANSMISSION] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", ACM Transactions on Computer Systems, DOI 10.1145/118544.118549, November 1991, <https://doi.org/10.1145/118544.118549>.

[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>.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/info/rfc3465>.

[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>.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <https://www.rfc-editor.org/info/rfc5682>.

[RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, DOI 10.17487/RFC5827, May 2010, <https://www.rfc-editor.org/info/rfc5827>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[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>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/info/rfc7661>.

[RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/info/rfc8511>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, 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>.

Приложение A. Псевдокод обнаружения потерь

Здесь рассматривается пример реализации механизмов обнаружения потерь, описанных в разделе 6. Приведённый псевдокод лицензируется как компоненты кода (см. «Авторские права»).

A.1. Отслеживание переданных пакетов

Для корректной реализации контроля перегрузок отправитель QUIC отслеживает все пакеты с запросом подтверждения, пока пакет не будет потерян или подтверждён. Предполагается, что реализация будет иметь доступ к этим сведениям по номеру пакета или криптографическому контексту и будет сохранять для каждого пакета поля (Приложение A.1.1), служащие для восстановления потерь и контроля перегрузки.

Когда пакет объявлен потерянным, конечная точка может некоторое время сохранять его состояние, чтобы учесть нарушение порядка пакетов (см. параграф 13.3 в [QUIC-TRANSPORT]). Это позволит отправителю обнаружить ложные повторы передачи. Передача отслеживается для каждого пространства номеров, а обработка ACK выполняется лишь в одном пространстве.

A.1.1. Поля переданных пакетов

packet_number
: Номер передаваемого пакета.

ack_eliciting
: Логический параметр, указывающий для пакета запрос подтверждения. Значение true указывает, что отправитель ждёт подтверждения, хотя партнёр может задержать отправку кадра ACK с этим подтверждением на время до max_ack_delay.

in_flight
: Логический параметр, указывающий учёт пакета в числе находящихся в пути байтов.

sent_bytes
: Число переданных в пакете байтов без учёта заголовков UDP и IP, но с учётом кадрирования QUIC.

time_sent
: Время передачи пакета.

A.2. Константы

Используемые для восстановления потерь константы основаны на комбинации RFC, статей и опыта.

kPacketThreshold
: Максимальное разупорядочение пакетов для фиксации потери пакета по достижению этого порога. В параграфе 6.1.1 рекомендовано значение 3.

kTimeThreshold
: Максимальное время разупорядочения пакетов по достижению которого пакет считается потерянным. Задаётся в интервалах RTT. В параграфе 6.1.2 рекомендовано значение 9/8.

kGranularity
: Дискретность таймера, зависящая от системы. В параграфе 6.1.2 рекомендовано значение 1 мсек.

kInitialRtt
: Значение RTT, использованное перед выборкой RTT. В параграфе 6.2.2 рекомендовано значение 333 мсек.

kPacketNumberSpace
: Перечисляемый тип для трёх пространств номеров пакетов.

```
enum kPacketNumberSpace {
  Initial,
```

```
    Handshake,
    ApplicationData,
  }
```

A.3. Переменные

Ниже приведены переменные, используемые для восстановления потерь.

latest_rtt
Наиболее свежее измерение RTT выполненное при получении подтверждения ранее не подтверждённого пакета.

smoothed_rtt
Сглаженное значение RTT для соединения, рассчитанное в соответствии с параграфом 5.3.

rttvar
Вариации RTT, рассчитанные в соответствии с параграфом 5.3.

min_rtt
Минимальное значение RTT за период времени без учёта задержки подтверждения, как указано в параграфе 5.2.

first_rtt_sample
Время первой выборки RTT.

max_ack_delay
Максимальное время, на которое получатель намерен задерживать подтверждения для пакетов из пространства номеров Application Data, определяемое одноимённым транспортным параметром (параграф 18.2 в [QUIC-TRANSPORT]). Отметим, что фактическое значение ack_delay в полученном кадре ACK может быть больше из-за запаздывания таймеров, нарушения порядка и потери пакетов.

loss_detection_timer
Многорежимный таймер, используемый для обнаружения потерь.

pto_count
Число фактов передачи PTO без получения подтверждения.

time_of_last_ack_eliciting_packet[kPacketNumberSpace]
Время передачи последнего пакета с запросом подтверждения.

largest_acked_packet[kPacketNumberSpace]
Наибольший номер, подтверждённый на данный момент в пространстве номеров пакетов.

loss_time[kPacketNumberSpace]
Время, когда следующий пакет в пространстве номеров может быть сочтён потерянным по выходу за пределы окна разупорядочения во времени.

sent_packets[kPacketNumberSpace]
Привязка номеров пакетов в определённом пространстве к информации о них (см. Приложение A.1).

A.4. Инициализация

В начале соединения инициализируются переменные обнаружения потерь, как показано ниже.

```
loss_detection_timer.reset()
pto_count = 0
latest_rtt = 0
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
min_rtt = 0
first_rtt_sample = 0
for pn_space in [ Initial, Handshake, ApplicationData ]:
  largest_acked_packet[pn_space] = infinite
  time_of_last_ack_eliciting_packet[pn_space] = 0
  loss_time[pn_space] = 0
```

A.5. Отправка пакета

Информация о пакете сохраняется после его передачи. Параметры OnPacketSent подробно описаны в Приложении A.1.1, а псевдокод приведён ниже.

```
OnPacketSent(packet_number, pn_space, ack_eliciting,
             in_flight, sent_bytes):
  sent_packets[pn_space][packet_number].packet_number =
                                        packet_number
  sent_packets[pn_space][packet_number].time_sent = now()
  sent_packets[pn_space][packet_number].ack_eliciting =
                                        ack_eliciting
  sent_packets[pn_space][packet_number].in_flight = in_flight
  sent_packets[pn_space][packet_number].sent_bytes = sent_bytes
  if (in_flight):
    if (ack_eliciting):
      time_of_last_ack_eliciting_packet[pn_space] = now()
    OnPacketSentCC(sent_bytes)
    SetLossDetectionTimer()
```

A.6. Приём дейтаграммы

Когда сервер блокируется пределом антиусиления, приём дейтаграммы снимает блокировку, даже если ни один из пакетов в этой дейтаграмме не был успешно обработан. В таком случае не требуется повторно активировать таймер PTO. Псевдокод OnDatagramReceived представлен ниже.

```
OnDatagramReceived(datagram):
  // Если эта дейтаграмма снимает блокировку сервер,
  // активируется таймер PTO для предотвращения блокировки.
  if (server was at anti-amplification limit):
    SetLossDetectionTimer()
    if loss_detection_timer.timeout < now():
      // Выполняется PTO, если отсчёт таймера завершился,
```

```
      // пока применялся порог антиусиления.
      OnLossDetectionTimeout()
```

A.7. Приём подтверждения

При получении кадра ACK он может заново подтверждать любое число пакетов. Псевдокод OnAckReceived и UpdateRtt приведён ниже.

```
IncludesAckEliciting(packets):
  for packet in packets:
    if (packet.ack_eliciting):
      return true
  return false

OnAckReceived(ack, pn_space):
  if (largest_acked_packet[pn_space] == infinite):
    largest_acked_packet[pn_space] = ack.largest_acked
  else:
    largest_acked_packet[pn_space] =
        max(largest_acked_packet[pn_space], ack.largest_acked)

  // DetectAndRemoveAckedPackets находит пакеты, которые недавно
  // подтверждены и удаляет их из sent_packets.
  newly_acked_packets =
      DetectAndRemoveAckedPackets(ack, pn_space)
  // Нечего делать при отсутствии недавно подтверждённых пакетов.
  if (newly_acked_packets.empty()):
    return

  // Обновление RTT при недавнем подтверждении наибольшего номера,
  // если недавно получено хотя бы одно запрошенное подтверждение.
  if (newly_acked_packets.largest().packet_number ==
        ack.largest_acked &&
      IncludesAckEliciting(newly_acked_packets)):
    latest_rtt =
      now() - newly_acked_packets.largest().time_sent
    UpdateRtt(ack.ack_delay)

  // Обработка сведений ECN (при наличии).
  if (ACK frame contains ECN information):
      ProcessECN(ack, pn_space)

  lost_packets = DetectAndRemoveLostPackets(pn_space)
  if (!lost_packets.empty()):
    OnPacketsLost(lost_packets)
  OnPacketsAcked(newly_acked_packets)

  // Сброс pto_count, пока клиент не уверен, что сервер
  // проверил его адрес.
  if (PeerCompletedAddressValidation()):
    pto_count = 0
  SetLossDetectionTimer()

UpdateRtt(ack_delay):
  if (first_rtt_sample == 0):
    min_rtt = latest_rtt
    smoothed_rtt = latest_rtt
    rttvar = latest_rtt / 2
    first_rtt_sample = now()
    return

  // min_rtt игнорирует задержку подтверждения.
  min_rtt = min(min_rtt, latest_rtt)
  // Ограничивает ack_delay значением max_ack_delay после
  // подтверждения согласования.
  if (handshake confirmed):
    ack_delay = min(ack_delay, max_ack_delay)

  // Настройка задержки подтверждения при возможности.
  adjusted_rtt = latest_rtt
  if (latest_rtt >= min_rtt + ack_delay):
    adjusted_rtt = latest_rtt - ack_delay

  rttvar = 3/4 * rttvar + 1/4 * abs(smoothed_rtt - adjusted_rtt)
  smoothed_rtt = 7/8 * smoothed_rtt + 1/8 * adjusted_rtt
```

A.8. Установка таймера обнаружения потерь

Для обнаружения потерь в QUIC применяется один таймер. Продолжительность отсчёта зависит от режима таймера, который установлен в пакете, и описанных ниже событий. Определённая ниже функция SetLossDetectionTimer показывает, как устанавливается один таймер. Этот алгоритм может устанавливать для таймера время в прошлом, особенно при позднем пробуждении таймера. В таком случае таймер срабатывает сразу же. Псевдокод SetLossDetectionTimer приведён ниже (^ обозначает возведение в степень).

```
GetLossTimeAndSpace():
  time = loss_time[Initial]
  space = Initial
```

```
    for pn_space in [ Handshake, ApplicationData ]:
      if (time == 0 || loss_time[pn_space] < time):
        time = loss_time[pn_space];
        space = pn_space
    return time, space

  GetPtoTimeAndSpace():
    duration = (smoothed_rtt + max(4 * rttvar, kGranularity))
        * (2 ^ pto_count)
    // Антиблокировочный тайм-аут PTO запускается с текущего момента
    if (no ack-eliciting packets in flight):
      assert(!PeerCompletedAddressValidation())
      if (has handshake keys):
        return (now() + duration), Handshake
      else:
        return (now() + duration), Initial
    pto_timeout = infinite
    pto_space = Initial
    for space in [ Initial, Handshake, ApplicationData ]:
      if (no ack-eliciting packets in flight in space):
          continue;
      if (space == ApplicationData):
        // Пропуск Application Data, пока согласование не подтверждено.
        if (handshake is not confirmed):
          return pto_timeout, pto_space
        // Включение max_ack_delay и отсрочки для Application Data.
        duration += max_ack_delay * (2 ^ pto_count)

      t = time_of_last_ack_eliciting_packet[space] + duration
      if (t < pto_timeout):
        pto_timeout = t
        pto_space = space
    return pto_timeout, pto_space

  PeerCompletedAddressValidation():
    // Предположим, что клиент проверяет адрес сервера неявно.
    if (endpoint is server):
      return true
    // Сервер завершает проверку адреса при получении
    // защищённого пакета.
    return has received Handshake ACK ||
         handshake confirmed

  SetLossDetectionTimer():
    earliest_loss_time, _ = GetLossTimeAndSpace()
    if (earliest_loss_time != 0):
      // Временной порог обнаружения потерь.
      loss_detection_timer.update(earliest_loss_time)
      return

    if (server is at anti-amplification limit):
      // Таймер сервера не устанавливается, если нечего передать.
      loss_detection_timer.cancel()
      return

    if (no ack-eliciting packets in flight &&
        PeerCompletedAddressValidation()):
      // Нет ничего для обнаружения потерь и таймер не устанавливается.
      // Однако клиент должен активировать таймер, если сервер может
      // быть заблокирован пределом антиусиления.
      loss_detection_timer.cancel()
      return

    timeout, _ = GetPtoTimeAndSpace()
    loss_detection_timer.update(timeout)
```

A.9. Тайм-аут

При завершении отсчёта таймера обнаружения потерь выполняемые действия определяются режимом таймера. Псевдокод OnLossDetectionTimeout приведён ниже

```
  OnLossDetectionTimeout():
    earliest_loss_time, pn_space = GetLossTimeAndSpace()
    if (earliest_loss_time != 0):
      // Временной порог обнаружения потерь
      lost_packets = DetectAndRemoveLostPackets(pn_space)
      assert(!lost_packets.empty())
      OnPacketsLost(lost_packets)
      SetLossDetectionTimer()
      return

    if (no ack-eliciting packets in flight):
      assert(!PeerCompletedAddressValidation())
      // Клиент передал противоблокироваочный пакет: пакет Initial
      // дополнен для увеличения предела антиусиления,
      // пакет Handshake подтверждает владение адресом.
```

```
      if (has Handshake keys):
        SendOneAckElicitingHandshakePacket()
      else:
        SendOneAckElicitingPaddedInitialPacket()
    else:
      // PTO. При наличии передаются новые данные или повторяются прежние.
      // Если ничего не доступно, передаётся один кадр PING.
      _, pn_space = GetPtoTimeAndSpace()
      SendOneOrTwoAckElicitingPackets(pn_space)

    pto_count++
    SetLossDetectionTimer()
```

A.10. Обнаружение потери пакетов

DetectAndRemoveLostPackets вызывается каждый раз при получении ACK или завершении отсчёта таймера порога обнаружения потерь. Эта функция работает с sent_packets для пространства номеров пакета и возвращает список пакетов, недавно сочтённых потерянными. Псевдокод DetectAndRemoveLostPackets приведён ниже.

```
DetectAndRemoveLostPackets(pn_space):
  assert(largest_acked_packet[pn_space] != infinite)
  loss_time[pn_space] = 0
  lost_packets = []
  loss_delay = kTimeThreshold * max(latest_rtt, smoothed_rtt)

  // Минимальное время kGranularity перед тем как пакет
  // будет сочтён потерянным.
  loss_delay = max(loss_delay, kGranularity)

  // Пакеты, переданные до этого времени, считаются потерянными.
  lost_send_time = now() - loss_delay

  foreach unacked in sent_packets[pn_space]:
    if (unacked.packet_number > largest_acked_packet[pn_space]):
      continue

    // Маркировка пакета как потерянного или установка времени, когда
    // его следует пометить. Отметим, что kPacketThreshold здесь
    // предполагает отсутствие заданных отправителем пропусков номеров.
    if (unacked.time_sent <= lost_send_time ||
        largest_acked_packet[pn_space] >=
          unacked.packet_number + kPacketThreshold):
      sent_packets[pn_space].remove(unacked.packet_number)
      lost_packets.insert(unacked)
    else:
      if (loss_time[pn_space] == 0):
        loss_time[pn_space] = unacked.time_sent + loss_delay
      else:
        loss_time[pn_space] = min(loss_time[pn_space],
                                  unacked.time_sent + loss_delay)
  return lost_packets
```

A.11. Отбрасывание ключей Initial или Handshake

При отбрасывании ключей Initial или Handshake пакеты из этого пространства отбрасываются и обновляется статус обнаружения потерь. Псевдокод OnPacketNumberSpaceDiscarded приведён ниже.

```
OnPacketNumberSpaceDiscarded(pn_space):
  assert(pn_space != ApplicationData)
  RemoveFromBytesInFlight(sent_packets[pn_space])
  sent_packets[pn_space].clear()
  // Сброс обнаружения потерь и таймера PTO.
  time_of_last_ack_eliciting_packet[pn_space] = 0
  loss_time[pn_space] = 0
  pto_count = 0
  SetLossDetectionTimer()
```

Приложение B. Псевдокод контроля перегрузки

Здесь рассматривается пример реализации контроллера перегрузок, описанного в разделе 7. Приведённый псевдокод лицензируется как компоненты кода (см. «Авторские права»).

B.1. Константы

Используемые для контроля перегрузок константы основаны на комбинации RFC, статей и опыта.

kInitialWindow
: Используемый по умолчанию предел находящихся в пути (in flight) байтов, как описано в параграфе 7.2.

kMinimumWindow
: Минимальное окно перегрузки (в байтах), как описано в параграфе 7.2.

kLossReductionFactor
: Фактор масштабирования, применяемый для сужения окна перегрузки при обнаружении новой потери. Раздел 7 рекомендует значение 0,5.

kPersistentCongestionThreshold
: Интервал времени фиксации сохраняющейся перегрузки, задаваемый коэффициентом для PTO. Параграф 7.6 рекомендует значение 3.

B.2. Переменные

Ниже приведены переменные, требуемые для реализации механизмов контроля перегрузки, описанных здесь.

max_datagram_size
: Текущий максимальный размер содержимого (payload) для отправителя без учёта заголовков UDP и IP. Значение max_datagram_size применяется при расчёте окна перегрузки. Конечная точка устанавливает значение этой переменной на основе максимального блока данных для пути (Path Maximum Transmission Unit или PMTU, см. параграф 14.2 в [QUIC-TRANSPORT]), но не менее 1200 байтов.

ecn_ce_counters[kPacketNumberSpace]
: Наибольшее значение счётчика ECN-CE в пространстве номеров пакета, сообщённое партнёром в кадре ACK. Это значение применяется для обнаружения роста сообщённых счётчиков ECN-CE.

bytes_in_flight
: Суммарное число байтов во всех переданных пакетах, содержащих кадр с запросом подтверждения или PADDING, которые ещё не были подтверждены или объявлены потерянными. Заголовки IP и UDP не учитываются, но включается заголовок QUIC и данные аутентифицированного шифрования со связанными данными (Authenticated Encryption with Associated Data или AEAD). Пакеты, содержащие лишь кадры ACK, не учитываются в bytes_in_flight, чтобы контроль перегрузок не препятствовал обратной связи в случае насыщения.

congestion_window
: Максимальное число байтов, которые могут находиться в пути.

congestion_recovery_start_time
: Время начала текущего периода восстановления в ответ на потерю или ECN. Когда подтверждается отправленный после этого момента пакет, QUIC выходит из режима восстановления при перегрузке.

ssthresh
: Порог замедленного старта (в байтах). Когда окно перегрузок меньше ssthresh, используется режим замедленного старта и окно увеличивается на число подтверждённых байтов.

Псевдокод контроля перегрузок использует также некоторые переменные псевдокода восстановления потерь.

B.3. Инициализация

В начале соединения инициализируются переменные контроля перегрузок, как показано ниже.

```
congestion_window = kInitialWindow
bytes_in_flight = 0
congestion_recovery_start_time = 0
ssthresh = infinite
for pn_space in [ Initial, Handshake, ApplicationData ]:
  ecn_ce_counters[pn_space] = 0
```

B.4. Передача пакета

При каждой отправке пакета, содержащего кадры, отличные от ACK, увеличивается значение bytes_in_flight.

```
OnPacketSentCC(sent_bytes):
  bytes_in_flight += sent_bytes
```

B.5. Подтверждение пакета

Приведённый ниже псевдокод вызывается из OnAckReceived для обнаружения потерь с недавними acked_packets из числа sent_packets. В механизмах предотвращения перегрузки разработчикам, применяющим целочисленное представление congestion_window, следует быть осторожными с делением и можно применять другой подход, предложенный в параграфе 2.1 [RFC3465].

```
InCongestionRecovery(sent_time):
  return sent_time <= congestion_recovery_start_time

OnPacketsAcked(acked_packets):
  for acked_packet in acked_packets:
    OnPacketAcked(acked_packet)

OnPacketAcked(acked_packet):
  if (!acked_packet.in_flight):
    return;
  // Исключение из bytes_in_flight.
  bytes_in_flight -= acked_packet.sent_bytes
  // Не увеличивать congestion_window, если ограничено
  // приложением или контролем потока данных.
  if (IsAppOrFlowControlLimited())
    return
  // Не увеличивать congestion_window в период восстановления.
  if (InCongestionRecovery(acked_packet.time_sent)):
    return
  if (congestion_window < ssthresh):
    // Замедленный старт.
    congestion_window += acked_packet.sent_bytes
  else:
    // Предотвращение перегрузки.
    congestion_window +=
      max_datagram_size * acked_packet.sent_bytes
      / congestion_window
```

B.6. Событие перегрузки

Приведённый ниже псевдокод вызывается из ProcessECN и OnPacketsLost при обнаружении нового факта перегрузки. Если восстановления ещё не происходит, это запускает период восстановления и сразу же уменьшает порог замедленного старта и окно перегрузки.

```
OnCongestionEvent(sent_time):
  // Нет реакции, если период восстановления уже начат.
  if (InCongestionRecovery(sent_time)):
    return

  // Вход в период восстановления.
  congestion_recovery_start_time = now()
  ssthresh = congestion_window * kLossReductionFactor
  congestion_window = max(ssthresh, kMinimumWindow)
  // Можно передать пакет для ускорения восстановления.
  MaybeSendOnePacket()
```

B.7. Обработка сведений ECN

Приведённый ниже псевдокод вызывается при получении от партнёра кадра ACK с разделом ECN.

```
ProcessECN(ack, pn_space):
  // Если сообщённое партнёром значение ECN-CE увеличивается,
  // это может быть новым фактом перегрузки.
  if (ack.ce_counter > ecn_ce_counters[pn_space]):
    ecn_ce_counters[pn_space] = ack.ce_counter
    sent_time = sent_packets[ack.largest_acked].time_sent
    OnCongestionEvent(sent_time)
```

B.8. Потеря пакета

Приведённый ниже псевдокод вызывается, когда DetectAndRemoveLostPackets считает пакеты потерянными.

```
OnPacketsLost(lost_packets):
  sent_time_of_last_loss = 0
  // Исключение потерянных пакетов из bytes_in_flight.
  for lost_packet in lost_packets:
    if lost_packet.in_flight:
      bytes_in_flight -= lost_packet.sent_bytes
      sent_time_of_last_loss =
        max(sent_time_of_last_loss, lost_packet.time_sent)
  // Перегрузка, если находящиеся в пути пакеты потеряны
  if (sent_time_of_last_loss != 0):
    OnCongestionEvent(sent_time_of_last_loss)

  // Сброс окна перегрузки, если потеря этих пакетов указывает
  // продолжающуюся перегрузку. Учитываются лишь пакеты,
  // переданные после выборки RTT.
  if (first_rtt_sample == 0):
    return
  pc_lost = []
  for lost in lost_packets:
    if lost.time_sent > first_rtt_sample:
      pc_lost.insert(lost)
  if (InPersistentCongestion(pc_lost)):
    congestion_window = kMinimumWindow
    congestion_recovery_start_time = 0
```

B.9. Исключение отброшенных пакетов из числа байтов в пути

Когда ключи Initial или Handshake отбрасываются, переданные пакеты из этого пространства номеров больше не учитываются в числе байтов, находящихся в пути. Псевдокод функции RemoveFromBytesInFlight приведён ниже.

```
RemoveFromBytesInFlight(discarded_packets):
  // Исключение неподтвержденных пакетов
  // из числа находящихся в пути.
  foreach packet in discarded_packets:
    if packet.in_flight
      bytes_in_flight -= size
```

Участники работы

Рабочая группа IETF QUIC получила огромную поддержку от многих людей. Ниже перечислены те, кто внёс существенный вклад в этот документ.

Alessandro Ghedini
Benjamin Saunders
Gorry Fairhurst
山本和彦 (Kazu Yamamoto)
奥 一穂 (Kazuho Oku)
Lars Eggert
Magnus Westerlund
Marten Seemann
Martin Duke
Martin Thomson
Mirja Kühlewind
Nick Banks
Praveen Balasubramanian

Адреса авторов

Jana Iyengar (editor)

Fastly

Email: jri.ietf@gmail.com

Ian Swett (editor)

Google

Email: ianswett@google.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru