Internet Engineering Task Force (IETF)
Request for Comments: 9000
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
M. Thomson, Ed.
Mozilla

May 2021

QUIC: A UDP-Based Multiplexed and Secure Transport

Мультиплексируемый защищённый транспорт QUIC на основе UDP

Аннотация

Этот документ описывает ядро транспортного протокола QUIC, обеспечивающего приложениям управляемые потоки данных для структурированных коммуникаций с малой задержкой при организации соединений и возможностью изменения пути через сеть. QUIC включает средства защиты, обеспечивающие конфиденциальность, целостность и доступность в разных условиях развёртывания. В сопроводительных документах описана интеграция с TLS для согласования ключей, обнаружения потерь и алгоритма контроля перегрузок.

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/rfc9000.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Обзор

QUIC представляет собой защищённый транспортный протокол общего назначения. Этот документ определяет протокол QUIC версии 1 с независимыми от версии свойствами QUIC, определёнными в [QUIC-INVARIANTS]. Протокол QUIC ориентирован на соединения и обеспечивает взаимодействие между клиентом и сервером с учётом состояния.

Согласование (handshake) QUIC объединяет в себе выбор криптографических и транспортных параметров. QUIC включает в себя согласование TLS [TLS13], хотя для защиты пакетов применяется специальное кадрирование. Интеграция TLS и QUIC более подробно описана в [QUIC-TLS]. Согласование структурировано для максимального быстрого начала обмена данными приложения и позволяет клиенту начать передачу данных незамедлительно (0-RTT), для чего нужна та или иная предшествующая связь или настройка конфигурации.

Конечные точки QUIC взаимодействуют путём обмена пакетами QUIC, большинство которых содержат кадры, переносящие между конечными точками данные управления и приложений. QUIC проверяет подлинность каждого пакета и шифрует пакеты, насколько это практично. Пакеты QUIC передаются в дейтаграммах UDP [UDP] для упрощения развёртывания в существующих системах и сетях.

Протоколы приложение обмениваются информацией через соединение QUIC в форме потока, представляющего собой упорядоченную последовательность байтов. Возможно создание двух типов потоков - двухсторонних, где данные могут передавать обе конечных точки и односторонних, где данные может передавать лишь одна конечная точка. Для управления созданием потоков и объёмом передаваемых данных служит схема на основе кредита.

QUIC обеспечивает обратную связь для реализации гарантированной доставки и контроля перегрузок. Алгоритм обнаружения потери данных и восстановления описан в разделе 6 [QUIC-RECOVERY]. Для предотвращения перегрузок в сети QUIC использует специальный механизм, описанный в разделе 7 [QUIC-RECOVERY].

Соединения QUIC не привязаны строго к одному пути через сеть и для их перемещения в сети применяются специальные идентификаторы. В данной версии QUIC соединения могут переноситься лишь клиентами. Протокол позволяет сохранять соединения при изменении топологии сети или отображения адреса, например, при смене привязки NAT.

Для организованного соединения имеется множество вариантов завершения (разрыва). Приложения могут использовать аккуратное завершение (graceful shutdown), конечные точки могут согласовать время ожидания (тайм-аут), ошибки могут приводить к немедленному разрыву соединения, а механизм без учёта состояния обеспечивает завершение соединения после потери одной из конечных точек.

1.1. Структура документа

Этот документ описывает ядро протокола QUIC и состоит из нескольких частей, перечисленных ниже.

- Потоки, являющиеся базовой абстракцией QUIC;
  - базовые концепции, связанные с потоками (раздел 2);
  - эталонная модель состояний потоков (раздел 3);
  - управление потоком данных (раздел 4).
- Соединения, обеспечивающие контекст взаимодействия конечных точек QUIC;
  - базовые концепции, связанные с соединениями (раздел 5);
  - согласование версий (раздел 6);
  - процесс организации соединения (раздел 7);

- проверка действительности адреса и ослабление DoS[1]-атак (раздел 8);
- перенос конечными точками соединения на другой путь (раздел 9);
- варианты прерывания соединений (раздел 10);
- рекомендации по обработке ошибок в потоках и соединениях (раздел 11).

- Пакеты и кадры, являющиеся базовыми блоками коммуникаций QUIC:
  - концепции, связанные с пакетами и кадрами (раздел 12);
  - модели передачи, повтора и подтверждения данных (раздел 13);
  - правила управления размером дейтаграмм для передачи пакетов QUIC (раздел 14).
- Кодирование элементов протокола QUIC:
  - версии (раздел 15);
  - целые числа (раздел 16);
  - заголовки пакетов (раздел 17);
  - транспортные параметры (раздел 18);
  - кадры (раздел 19);
  - ошибки (раздел 20).

В сопровождающих документах описано детектирование потерь и контроль перегрузок в QUIC [QUIC-RECOVERY], а также использование TLS и других криптографических механизмов [QUIC-TLS].

Этот документ определяет протокол QUIC версии 1, соответствующий инвариантам, заданным в [QUIC-INVARIANTS].

Для указания QUIC версии 1 следует ссылаться на этот документ, для указания независимого от версии ограниченного набора свойств QUIC - [QUIC-INVARIANTS].

1.2. Термины 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.

QUIC
: Транспортный протокол, описанный в этом документе. QUIC — это имя, а не сокращение.

Endpoint - конечная точка
: Сущность, которая может участвовать в соединении QUIC путём генерации, получения и обработки пакетов QUIC. В протоколе QUIC имеется лишь два типа конечных точек - клиенты и серверы.

Client - клиент
: Конечная точка, инициирующая соединение QUIC.

Server - сервер
: Конечная точка, воспринимающая соединение QUIC.

QUIC packet - пакет QUIC
: Пригодный для обработки законченный блок QUIC, который может быть инкапсулирован в дейтаграмму UDP. Одна дейтаграмма UDP может содержать один или несколько пакетов QUIC.

Ack-eliciting packet - пакет с запросом подтверждения
: Пакет QUIC, содержащий кадры, отличные от ACK, PADDING, CONNECTION_CLOSE. Такой пакет заставляет получателя передать подтверждение (см. параграф 13.2.1. Передача кадров ACK).

Frame - кадр
: Блок структурированной информации протокола. Имеется множество типов кадров, каждый из которых содержит свою информацию. Кадры содержатся в пакетах QUIC.

Address - адрес
: При использовании без уточнения — кортеж из номера версии IP, адреса IP и номера порта UDP, представляющих один конец пути через сеть.

Connection ID - идентификатор соединения
: Идентификатор, служащий для указание соединения QUIC на конечной точке. Каждая из конечных точек выбирает один или несколько идентификаторов соединения для своего партнёра, включая их в передаваемые в направлении этой конечной точки пакеты. Партнер не анализирует значение идентификатора (opaque).

Stream - поток
: Односторонний или двухсторонний канал (передачи) упорядоченных байтов в соединении QUIC. Соединение QUIC может поддерживать множество потоков одновременно.

Application - приложение
: Сущность, использующая протокол QUIC для передачи и приёма данных.

В документе применяются термины «пакеты QUIC», «дейтаграммы UDP» и «пакеты IP» для обозначения блоков данных соответствующего протокола. Таким образом, один или несколько пакетов QUIC может быть инкапсулировано в дейтаграмму UDP, которая инкапсулирована в пакет IP.

[1]Denial-of-service - отказ в обслуживании.

1.3. Соглашения о нотации

Для представления пакетов и кадров в этом документе применяется свой формат, предназначенный для обобщения, а не для определения элементов протокола. Семантика и детали структур указываются в виде текста. Составные поля указываются именем, а затем следует список полей в фигурных скобках через запятые. Отдельные поля включают сведения о размере, а также указания о фиксированном значении, необязательности или повторах. Для полей применяются указанные ниже обозначения (размеры задаются в битах).

x (A)
Поле x размером A битов.

x (i)
Поле x содержит целое число с представлением размера, описанным в разделе 16.

x (A..B)
Указывает, что x может иметь любой размер от A до B, если A не указано, поле может иметь размер 0, отсутствие B говорит, что размер поля не ограничен. Значения в этом формате всегда завершаются на границе байта.

x (L) = C
Указывает, что x имеет фиксированное значение C, размером L в одной из приведённых выше форм.

x (L) = C..D
Указывает, что x имеет значение из диапазона от C до D, включительно, и имеет размер L, как указано выше.

[x (L)]
Указывает необязательность x размера L.

x (L) …
Указывает повторение x (возможно пустой набор), где каждый экземпляр имеет размер L.

В документе применяется сетевой порядок байтов (big endian), поля указываются со старшего бита в каждом байте.

Отдельные поля составного поля указываются с именем составного поля. На рисунке 1 представлен пример.

```
Example Structure {
  One-bit Field (1),
  7-bit Field with Fixed Value (7) = 61,
  Field with Variable-Length Integer (i),
  Arbitrary-Length Field (..),
  Variable-Length Field (8..24),
  Field With Minimum Length (16..),
  Field With Maximum Length (..128),
  [Optional Field (64)],
  Repeated Field (8) ...,
}
```

Рисунок 1. Пример формата.

При упоминании однобитового поля его позицию можно уточнить используя байт, содержащий поле с установленным значением. Например, значение 0x80 может служить ссылкой на старший бит байта, как One-bit Field на рисунке 1.

2. Потоки

Потоки в QUIC предоставляют приложению облегчённой абстракции упорядоченного потока байтов. Потоки могут быть односторонними и двухсторонними.

Потоки могут создаваться передачей данных. Другие процессы, связанные с управлениям потоками, завершение, отмена, управление потоком данных - организованы так, чтобы вносить минимальные издержки. Например, один кадр STREAM (параграф 19.8. Кадр STREAM) может создать поток, передать в нем данные и закрыть поток. Потоки могут также быть долгоживущими и сохраняться в течение всего срока работы соединения.

Потоки может создавать любая конечная точка и можно чередовать данные разных потоков, а также отменять потоки. QUIC не включает средств упорядочения байтов между разными потоками.

QUIC позволяет одновременную работу произвольного числа потоков и передачу в потоке произвольного объёма данных с учётом ограничений самого потока и управления потоком данных (см. раздел 4. Управление потоком).

2.1. Типы и идентификаторы потоков

Потоки могут быть односторонними и двухсторонними. В одностороннем потоке данные передаются от инициатора его партнёру, в двухстороннем - в обоих направлениях.

Потоки в соединении указываются идентификаторами (stream ID) в форме 62-битовых целочисленных значений от 0 до 2^{62}-1, которые однозначно указывают поток внутри соединения. Идентификаторы потоков представляются целыми числами переменного размера (см. раздел 16). Конечной точке QUIC **недопустимо** повторно применять идентификатор потока в том же соединении.

Младший бит идентификатора (0x01) указывает инициатора потока. Для потоков, созданных клиентом, этот бит сброшен (0), а для инициированных сервером потоков установлен (1). Таким образом, инициированные клиентом потоки имеют чётные идентификаторы, инициированные сервером - нечётные. Второй (с конца) бит идентификатора (0x02) указывает двухсторонний (0) или односторонний (1) поток. Эти два младших бита идентификатора указывают один из 4 типов потоков, показанных в таблице 1.

Таблица 1. Типы идентификаторов потока.

Биты	*Тип потока*
0x00	Двухсторонний по инициативе клиента
0x01	Двухсторонний по инициативе сервера
0x02	Односторонний по инициативе клиента
0x03	Односторонний по инициативе сервера

Пространство номеров потоков каждого типа начинается с минимального значения (от 0x00 до 0x03) и последующие потоки каждого типа создаются с увеличивающимся численным значением идентификатора. Создание потока с пропуском в пространстве идентификаторов ведёт к созданию потоков со всеми пропущенными номерами.

2.2. Передача и приём данных

Кадры STREAM (параграф 19.8) инкапсулируют данные, переданные приложением. Конечная точка использует поля Stream ID и Offset в кадрах STREAM для упорядочения данных. Конечные точки **должны** быть способны доставлять приложению поток данных в форме упорядоченного потока байтов. Доставка упорядоченного потока байтов требует от конечной точки буферизации полученных с нарушением порядка данных вплоть до объявленного предела управления потоком данных. QUIC не задаёт конкретных требований для доставки потоков данных с нарушением порядка. Однако реализация **может** предлагать возможность доставки данных принимающему приложению с нарушением порядка.

Конечная точка может получать данные для потока с одним смещением несколько раз и данные, которые уже были получены, отбрасываются. Данные с одним смещением **недопустимо** менять, если они передаются более одного раза. Конечная точка **может** рассматривать получение разных данных с одним смещением в потоке как ошибку соединения типа PROTOCOL_VIOLATION.

Поток является абстракцией упорядоченного потока байтов без представления протоколу QUIC его структуры. При передаче, повторе в случае потери пакета и доставке данных приложению не предполагается сохранение границ кадров STREAM.

Конечное точке **недопустимо** передавать данные в поток без гарантии выполнения установленных партнёром ограничений управления потоком данных (см. раздел 4. Управление потоком).

2.3. Приоритизация потока

Мультиплексирование потоков может существенно влиять на производительность приложения, если выделяемые потокам ресурсы корректно приоритизированы. QUIC не задаёт механизма обмена данными приоритизации, опираясь на информацию о приоритетах, полученную от приложения.

Реализации QUIC **следует** обеспечивать приложениям способ указания относительного приоритета потоков. Сведения о приоритете от приложения используются при распределении ресурсов, выделяемых потокам.

2.4. Операции над потоками

Этот документ не определяет API для протокола QUIC, а задаёт набор функций для потоков, на которые может полагаться прикладной протокол. Протокол приложения может полагать, что реализация QUIC обеспечивает интерфейс, описанный в этом параграфе. Реализации, созданные для применения с конкретным прикладным протоколом, могут предоставлять лишь используемые этим протоколом операции.

На передающей стороне потока прикладной протокол может:

- записывать данные, понимая, когда протокол управления потоком данных (параграф 4.1) успешно зарезервирован для отправки записанных данных;
- завершать поток (аккуратно), в результате чего в кадре STREAM (параграф 19.8) устанавливается бит FIN;
- сбрасывать поток (аварийно), в результате чего появляется кадр RESET_STREAM (параграф 19.4), если поток ещё не находится в завершающем состоянии.

На приёмной стороне потока прикладной протокол может:

- читать данные;
- прерывать чтение из потока и запрашивать закрытие, возможно с появлением кадра STOP_SENDING (параграф 19.5).

Прикладной протокол может также запросить информирование об изменении статуса потоков, включая открытие или сброс потока партнёром, когда тот прекращает считывание потока, нет доступных данных или управление потоком данных не позволяет запись в поток.

3. Состояния потока

В этом разделе потоки описываются с точки зрения передающей и приёмной стороны. Описаны два конечных автомата, один для потоков передающей конечной точки (параграф 3.1), другой - для приёмной (параграф 3.2).

Односторонние потоки используют один из этих автоматов в зависимости от типа потока и роли конечной точки. В двухсторонних потоках на каждой стороне применяются оба конечных автомата. Использование этих конечных автоматов в значительной степени одинаково для односторонних и двухсторонних потоков. Условия создания потока чуть сложнее для двухстороннего потока, поскольку создание потока приёмной или передающей стороной ведёт к созданию встречного потока.

В этом разделе конечные автоматы представлены в основном для информации. Документ использует состояния потоков для описания правил отправки разных типов кадров и реакции на получение разных типов кадров. Хотя эти автоматы предназначены для использования реализациями QUIC, описание не предназначено для задания

ограничений. Реализация может определить свой конечный автомат, обеспечивающий поведение, соответствующее этому документу.

Примечание. В некоторых случаях одно событие или действие может вызывать переход через несколько состояний. Например, отправка STREAM с установленным флагом FIN может вызывать две смены состояния передающего потока - из Ready в Send и из Send в Data Sent.

3.1. Состояния передающего потока

На рисунке 2 показаны состояния передающей стороны потока. Передающая сторона потока, который инициирует конечная точка (типы 0 и 2 для клиентов, 1 и 3 для серверов), создаётся приложением. Состояние Ready представляет вновь созданный поток, способный воспринимать данные от приложения. В этом состоянии данные потока могут буферизоваться для подготовки к их передаче.

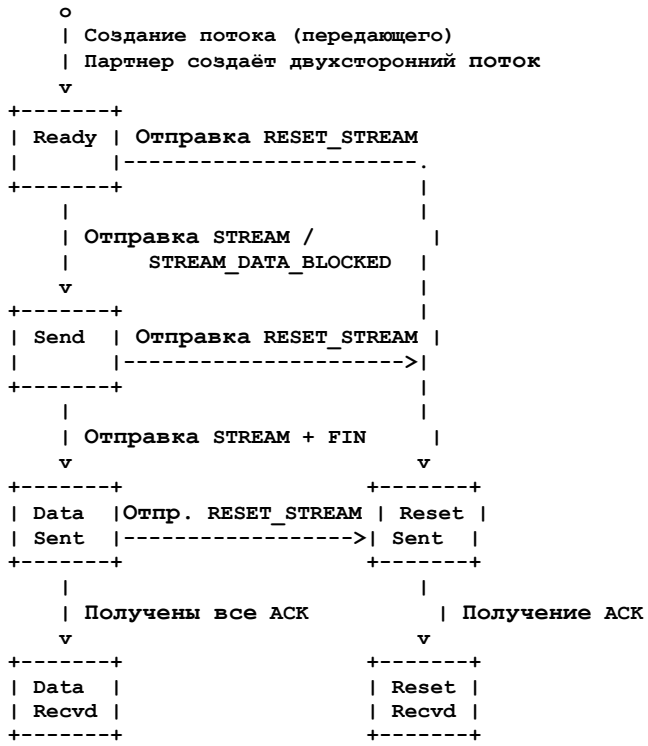

Рисунок 2. Состояния передающих частей потока.

Отправка первого кадра STREAM или STREAM_DATA_BLOCKED вызывает переход передающей стороны в состояние Send. Реализация может отложить выделение идентификатора потоку до отправки первого кадра STREAM и перехода в это состояние, что может позволить лучше приоритизировать потоки.

Передающая часть двухстороннего потока, инициированного партнёром (тип 0 для сервера, 1 для клиента), запускается в состоянии Ready при создании приёмной части.

В состоянии Send конечная точка передаёт (при необходимости повторно) данные потока в кадрах STREAM, соблюдая ограничения управления потоком данных, установленные партнёром, и продолжает воспринимать и обрабатывать кадры MAX_STREAM_DATA. Конечная точка в состоянии Send генерирует кадры STREAM_DATA_BLOCKED, если отправка для неё заблокирована ограничениями управления потоком данных (параграф 4.1).

После того, как приложение указало, что все данные потока отправлены, и передало кадр STREAM с флагом FIN, передающая сторона переходит в состояние Data Sent. Из этого состояния конечная точка лишь повторяет передачу данных потока при необходимости. В этом состоянии конечной точке не нужно проверять ограничения контроля потока данных или передавать кадры STREAM_DATA_BLOCKED. Пока партнёр не получит данные с конечным смещением для потока, могут приходить кадры MAX_STREAM_DATA, которые конечная точка в этом состоянии может игнорировать.

После подтверждения всех данных потока передающая сторона переходит в завершающее состояние Data Recvd.

Из состояний Ready, Send, Data Sent приложение может сигнализировать о своём желании прервать передачу данных потока. Кроме того, конечная точка может получить от партнёра кадр STOP_SENDING. В любом из этих случаев конечная точка передаёт кадр RESET_STREAM, переводящий поток в состояние Reset Sent.

Конечная точка **может** передать RESET_STREAM в качестве первого кадра, где упоминается поток. Это ведёт к созданию передающей стороной потока и его незамедлительный переход в состояние Reset Sent.

После подтверждения пакета, содержащего RESET_STREAM, передающая сторона переходит в завершающее состояние Reset Recvd.

3.2. Состояния принимающего потока

На рисунке 3 показаны состояния принимающей стороны потока. Эти состояния отражают лишь некоторые из состояний потока передающей стороны у партнёра. Приёмная сторона не отслеживает состояний передающей, которые невозможно наблюдать, например Ready, а отслеживает доставку данных приложению и некоторые из этих состояний отправителю не видны.

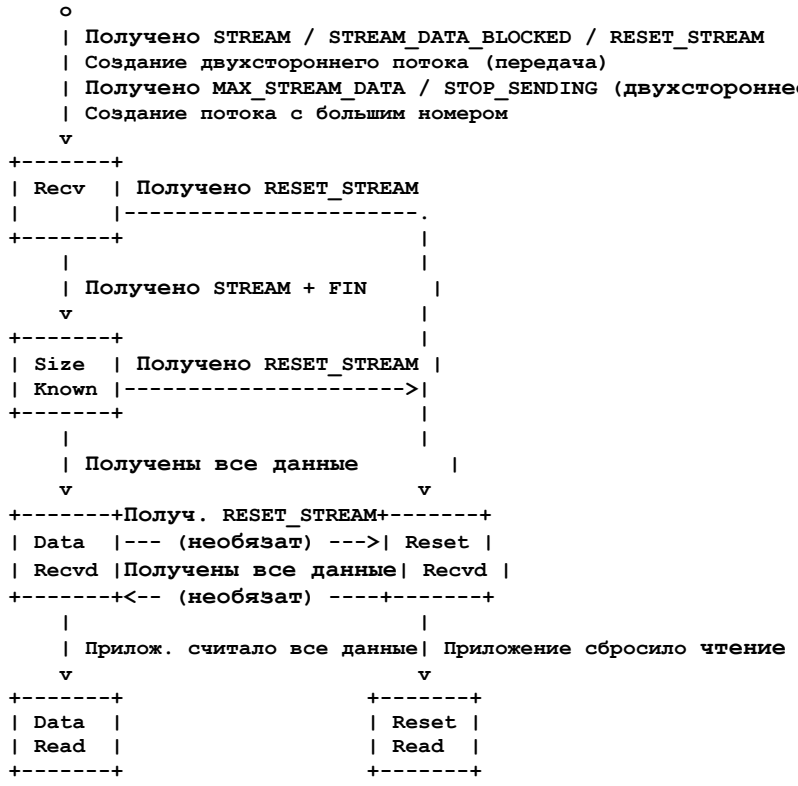

Рисунок 3. Состояния принимающих частей потока.

Принимающая сторона потока, инициированного партнёром (типы 1 и 3 для клиента, 0 и 2 для сервера) создаётся при получении для потока первого кадра STREAM, STREAM_DATA_BLOCKED или RESET_STREAM. Для двухсторонних потоков, инициированных партнёром, получение кадра MAX_STREAM_DATA или STOP_SENDING для передающей стороны также создаёт приёмную часть. Исходным состоянием для приемной стороны потока является Recv. Для двухстороннего потока принимающая часть переходит в состояние Recv, когда инициированная конечной точкой передающая часть (тип 0 для клиента, 1 для сервера) переходит в состояние Ready.

Конечная точка создаёт двухсторонний поток при получении от партнёра кадра MAX_STREAM_DATA или STOP_SENDING. Приём кадра MAX_STREAM_DATA для несозданного потока показывает, что удалённый партнёр создал поток и предоставил кредит контроля потока данных. Получение кадра STOP_SENDING для несозданного потока говорит о нежелании удалённого партнёра продолжать приём данных в этом потоке. Любой из этих кадров может приходить до приёма STREAM или STREAM_DATA_BLOCKED, если пакеты теряются или нарушается порядок.

До создания потока **должны** быть созданы все потоки того же типа с меньшими идентификаторами. Это обеспечивает согласованность порядка потоков на обеих сторонах.

В состоянии Recv конечная точка принимает кадры STREAM и STREAM_DATA_BLOCKED. Входящие данные буферизуются и могут быть собраны в корректном порядке для доставки приложению. По мере чтения данных приложением и освобождения буферного пространства конечная точка передаёт кадры MAX_STREAM_DATA, позволяющие партнёру продолжить передачу данных.

При получении кадра STREAM с флагом FIN становится известным окончательный размер потока (параграф 4.5) и принимающая часть потока переходит в состояние Size Known. В этос состоянии уже не нужно передавать кадры MAX_STREAM_DATA и принимаются лишь повторы данных потока.

После получения всех данных потока приёмная сторона переходит в состояние Data Recvd. Это может быть результатом получения того же кадра STREAM, который вызвал переход в статус Size Known. После приёма всех данных кадры STREAM и STREAM_DATA_BLOCKED для потока могут отбрасываться.

Состояние Data Recvd сохраняется, пока данные потока не будут доставлены приложению, после чего поток переходит в состояние Data Read, которое является завершающим.

Получение кадра RESET_STREAM в состоянии Recv или Size Known переводит поток в статус Reset Recvd, что может приводить к прерыванию доставки данных из потока приложению. При получении RESET_STREAM все данные потока могут быть уже приняты (состояние Data Recvd). Возможно также прибытие остающихся данных потока после получения кадра RESET_STREAM (состояние Reset Recvd). Реализация может обрабатывать это по своему усмотрению.

Отправки RESET_STREAM означает, что конечная точка не может гарантировать доставку данных потока, однако не требуется отказа от доставки данных потока при получении RESET_STREAM. Реализация **может** прервать доставку данных потока, отбрасывая не прочитанные данные и сигнализируя о получении RESET_STREAM. Сигнал RESET_STREAM может быть подавлен или задержан, если данные потока полностью получены и буферизованы для считывания приложением. При подавлении RESET_STREAM приёмная сторона остаётся в состоянии Data Recvd.

После получения приложением сигнала о сбросе потока приёмная сторона переходит в состояние Reset Read, которое является завершающим.

3.3. Разрешённые типы кадров

Отправитель потока передаёт лишь три типа кадров, влияющих на состояние потока у отправителя или получателя: STREAM (параграф 19.8), STREAM_DATA_BLOCKED (параграф 19.13), RESET_STREAM (параграф 19.4).

Отправителю **недопустимо** передавать такие кадры из завершающего состояния (Data Recvd или Reset Recvd), а также **недопустимо** передавать кадры STREAM или STREAM_DATA_BLOCKED для потока в состоянии Reset Sent или завершающем состоянии, т. е. после отправки кадра RESET_STREAM. Получатель может принять любой из этих трёх кадров в любом состоянии, поскольку пакеты с ними могут быть задержаны.

Получатель потока передаёт кадры MAX_STREAM_DATA (параграф 19.10) и STOP_SENDING (параграф 19.5). Кадры MAX_STREAM_DATA получатель передаёт лишь в состоянии Recv. Получатель **может** передать кадр STOP_SENDING из любого состояния, в котором он не получил кадр RESET_STREAM (т. е. состояния, отличные от Reset Recvd и Reset Read). Однако нет большого смысла передавать STOP_SENDING из состояния Data Recvd. Поскольку все данные потока уже получены. Отправитель может получить любой из этих кадров в любом состоянии в результате задержки пакетов.

3.4. Состояния двухстороннего потока

Двухсторонний поток состоит из передающей и принимающей части. Реализация может представлять состояния двухстороннего потока как композицию состояний передающего и принимающего потока. Простейшая модель представляет поток как «открытый», когда любая из сторон находится в незавершающем состоянии, а «закрытым» считается поток, обе стороны которого находятся в завершающем состоянии.

В таблице 2 представлено более сложное сопоставление состояний двухстороннего потока, которые примерно соответствуют состояниям потока в HTTP/2 [HTTP2]. Это показывает, что несколько состояний передающей или приёмной стороны потока отображаются на одно композитное состояние. Отметим, что это лишь один из вариантов такого сопоставления и для него требуется подтверждение доставки до перехода в состояние closed или half-closed.

Таблица 2. Возможное отображение состояний потока на HTTP/2.

Передающая сторона	***Приёмная сторона***	***Композитное состояние***
No Stream / Ready	No Stream / Recv[1]	idle
Ready / Send / Data Sent	Recv / Size Known	open
Ready / Send / Data Sent	Data Recvd / Data Read	half-closed (удалённое)
Ready / Send / Data Sent	Reset Recvd / Reset Read	half-closed (удалённое)
Data Recvd	Recv / Size Known	half-closed (локальное)
Reset Sent / Reset Recvd	Recv / Size Known	half-closed (локальное)
Reset Sent / Reset Recvd	Data Recvd / Data Read	closed
Reset Sent / Reset Recvd	Reset Recvd / Reset Read	closed
Data Recvd	Data Recvd / Data Read	closed
Data Recvd	Reset Recvd / Reset Read	closed

3.5. Запрошенная смена состояния

Если приложение больше не заинтересовано в данных из потока, оно может прервать поток и указать код ошибки.

Если приложение находится в состоянии Recv или Size Known, транспорту **следует** указать это отправкой кадра STOP_SENDING для запроса закрытия потока в обратном направлении. Обычно этого указывает, что принимающее приложение больше не читает полученные в потоке данные, но не гарантирует игнорирование входящих данных.

Кадры STREAM, полученные после отправки STOP_SENDING, по-прежнему учитываются для соединения и управления потоком данных, хотя они могут быть отброшены при получении.

Кадр STOP_SENDING запрашивает у приёмной конечной точки отправку кадра RESET_STREAM. Получившая STOP_SENDING конечная точка **должна** передать кадр RESET_STREAM, если поток находится в состоянии Ready или Send. Если поток находится в состоянии Data Sent, конечная точка **может** отложить передачу кадра RESET_STREAM до подтверждения пакетов с оставшимися данными или признания их потерянными. Если оставшиеся данные сочтены потерянными, конечной точке **следует** передать кадр RESET_STREAM вместо отправки данных.

Конечной точке **следует** копировать код ошибки из кадра STOP_SENDING в передаваемый кадр RESET_STREAM, но можно указать и код ошибки из приложения. Передающая кадр STOP_SENDING конечная точка **может** игнорировать код ошибки в кадрах RESET_STREAM, полученных позднее для этого потока.

STOP_SENDING **следует** передавать лишь для потока, который не был сброшен партнёром и наиболее полезен этот кадр в состоянии Recv или Size Known.

Предполагается, что конечная точка передаст другой кадр STOP_SENDING при потере пакета с предыдущим STOP_SENDING. Однако после приёма для потока всех данных или получения кадра (т. е. для потока с состоянием, отличным от Recv и Size Known) отправка STOP_SENDING не требуется.

Конечная точка, желающая разорвать оба направления двухстороннего потока, может прервать одно направление отправкой кадра RESET_STREAM и ускорить завершение другого направления передачей STOP_SENDING.

4. Управление потоком

Получателям нужно ограничивать объем данных, которые они должны буферизовать для предотвращения перегрузки со стороны быстрого отправителя или потребления излишней памяти в результате действий злоумышленника. Чтобы получатель мог ограничить для соединения расход памяти, в потоках применяется управление потоком данных на уровне отдельного потока и соединения в целом. Получатель QUIC контролирует максимальный объем данных, которые может передать отправитель в потоке или всех потоках соединения, как указано в параграфах 4.1 и4.2.

Для ограничения числа одновременных потоков конечная точка QUIC может задать максимальное число потоков, которые может инициировать партнёр, как описано в параграфе 4.6.

[1]Соединение считается бездействующим (idle), если оно ещё не создано или принимающая сторона находится в состоянии Recv, но ещё не получила кадров.

Для данных, передаваемых в кадрах CRYPTO, не применяется такое же управление потоком данных, как для данных потока. QUIC полагается на реализацию криптографического протокола для предотвращения избыточной буферизации данных (см. [QUIC-TLS]). Для предотвращения избыточной буферизации на разных уровнях реализациям QUIC **следует** поддерживать интерфейс с реализацией криптографического протокола для обмена данными о буферизации.

4.1. Управление потоком данных

QUIC реализует схему управления потоками данных на основе ограничений, где получатель анонсирует предельное число байтов, которые он готов принять в данном потоке или соединении в целом. Это обеспечивает два уровня управления потоками данных в QUIC.

- Управление на уровне потока препятствует занятие одним потоком всего приёмного буфера соединения путём ограничения объёма данных в каждом потоке.
- Управление на уровне соединения не позволяет отправителю выйти за пределы буферной ёмкости получателя путём ограничения числа байтов, которые могут быть переданы в кадрах STREAM всех потоков.

Отправителям **недопустимо** передавать данные сверх заданного предела.

Получатель устанавливает начальные пределы для всех потоков через транспортные параметры в процессе согласования (параграф 7.4). Далее получатель передаёт отправителю кадры MAX_STREAM_DATA (параграф 19.10) или MAX_DATA (параграф 19.9) для анонсирования более высоких ограничений. Получатель может указать более высокий предел для потока, передавая кадр MAX_STREAM_DATA с соответствующим идентификатором потока. Кадр MAX_STREAM_DATA указывает максимальное абсолютное смещение в потоке. Получатель может определить смещение для контроля потока данных на основе текущего смещения воспринятых в потоке данных.

Получатель может установить более высокий предел для соединения, передав кадр MAX_DATA с максимальной суммой абсолютных смещений байтов во всех потоках. Получатель поддерживает совокупную сумму байтов, принятых во всех потоках, которая кружит для контроля ограничений, анонсированных для соединения или потоков. Получатель может определить максимальное значение предела данных на основе суммы данных, воспринятых всеми потоками.

После анонсирования получателем предельного значения для соединения или потока указание меньшего предела не является ошибкой, но не будет оказывать влияния.

Получатель **должен** закрыть соединение с ошибкой FLOW_CONTROL_ERROR, если отправитель нарушает заданные для соединения или потока ограничения (см. раздел 11. Обработка ошибок).

Отправитель **должен** игнорировать кадры MAX_STREAM_DATA или MAX_DATA, которые не увеличивают пределы ограничения потока данных.

Если отправитель передал заданный ограничением объем данных, он не сможет больше отправлять данные и считается заблокированным. Отправителю **следует** передать кадр STREAM_DATA_BLOCKED или DATA_BLOCKED для указания получателю, что имеются дополнительные данные для передачи, но отправитель заблокирован контролем потока данных. Если отправитель заблокирован на время, превышающее тайм-аут бездействия (параграф 10.1), получатель может закрыть соединение даже при наличии у отправителя дополнительных данных для передачи. Чтобы предотвратить закрытие соединения, отправителю, для которого задано управление потоком данных, **следует** периодически отправлять кадры STREAM_DATA_BLOCKED или DATA_BLOCKED, если у него нет находящихся «в полете» пакетов с подтверждениями.

4.2. Расширение пределов управления потоком данных

Реализация решает, когда и какие кредиты на передачу следует анонсировать в кадрах MAX_STREAM_DATA и MAX_DATA, а здесь приведены некоторые соображения на этот счёт. Для предотвращения блокировки отправителя получатель **может** передать кадр MAX_STREAM_DATA или MAX_DATA неоднократно в течение интервала кругового обхода или передать кадр заранее с учётом возможной потери кадра и восстановления.

Кадры управления вносят вклад в издержки соединения, поэтому частая отправка MAX_STREAM_DATA и MAX_DATA с небольшими изменениями нежелательна. С другой стороны, если обновления передавать реже, требуется существенное повышение пределов для предотвращения блокировки отправителя, а это ведёт к большему расходу ресурсов у получателя. Нужен компромисс между выделением ресурсов и издержками.

Получатель может использовать механизм автоматической настройки для выбора частоты анонсов и размера добавочного кредита на основе оценки времени кругового обхода и скорости, с которой принимающее приложение потребляет данные, подобно обычным реализациям TCP. В качестве оптимизации конечная точка может передавать кадры, относящиеся к управлению потоком данных, только при отправке других кадров, что позволит избавиться от передачи лишних пакетов.

Блокированному отправителю не требуется передавать кадры STREAM_DATA_BLOCKED или DATA_BLOCKED. Поэтому получателю **недопустимо** ждать получения STREAM_DATA_BLOCKED или DATA_BLOCKED перед отправкой MAX_STREAM_DATA или MAX_DATA. Такое ожидание может приводить к блокировке отправителя для оставшейся части соединения. Даже при передаче отправителем таких кадров ожидание может привести к блокировке отправителя по меньшей мере на интервал кругового обхода.

При получении отправителем кредита после блокировки он сможет передать в ответ больший объем данных, что может привести к кратковременной перегрузке (см. параграф 7.7 в [QUIC-RECOVERY], где рассматриваются способы предотвращения такой перегрузки).

4.3. Производительность и управление потоком данных

Если конечная точка не может обеспечить партнёру доступный кредит управления потоком данных, превышающий произведение пропускной способности и задержки для этого соединения, производительность будет ограничена контролем потока данных.

Потери пакетов вызывают пропуски в приёмном буфере, препятствующие восприятию данных приложением и освобождению пространства в приёмном буфере.

Своевременная передача обновлений пределов управления потоком данных может повысить производительность. Отправка пакетов лишь для управления потоком данных может увеличить загрузку сети и негативно повлиять на производительность. Передача обновлений контроля потока данных вместе с другими кадрами, такими как ACK, снижает расходы на обновления.

4.4. Обработка отмены потока

Конечные точки должны согласовывать объем кредита управления потоком данных, потребляемого каждым потоком, чтобы учесть все байты для контроля потока данных на уровне соединения.

При получении кадра RESET_STREAM конечная точка прервёт состояние для соответствующего потока и будет игнорировать последующие данные в этом потоке. RESET_STREAM жёстко прерывает одно направление потока и в двухсторонних потоках не влияет на другое направление. Обе конечных точки **должны** поддерживать состояние контроля потока данных для непрерванного направления, пока это направление не перейдёт в завершающий статус.

4.5. Окончательный размер потока

Окончательный размер - это кредит управления потоком данных, потреблённый потоком. В предположении однократной передачи каждого байта этот размер совпадает с числом переданных байтов. В более общем смысле - это значение на 1 больше максимального смещения байта, переданного в поток, или 0, если данные в поток не передавались.

Отправитель всегда гарантированно передаёт окончательный размер потока получателю, независимо от способа завершения потока. Окончательный размер определяет сумма полей Offset и Length в кадре STREAM с флагом FIN, но эти поля могут быть неявными. Кроме того, это значение передаётся в поле Final Size кадра RESET_STREAM. Это гарантирует, что обе конечные точки согласны с размером кредита управления потоком данных, использованного отправителем в этом потоке.

Конечная точка узнает окончательный размер потока, когда принимающая часть потока переходит в состояние Size Known или Reset Recvd (раздел 3). Получатель **должен** использовать окончательный размер потока для учёта всех байтов, переданных в поток, на контроллере управления потоком данных для соединения в целом.

Конечной точке **недопустимо** передавать в поток данные сверх окончательного размера[1].

После того, как окончательный размер стал известен, его уже нельзя изменить. При получении кадра RESET_STREAM или STREAM, указывающего изменение окончательного размера для потока конечной точке **следует** вернуть ошибку типа FINAL_SIZE_ERROR (см. 11. Обработка ошибок). Получателю **следует** трактовать приём данных сверх окончательного размера как ошибку FINAL_SIZE_ERROR даже в случае уже закрытого потока. Возврат этой ошибки не является обязательным, поскольку введение такого требования означало бы необходимость поддержки конечной точкой окончательного размера для закрытых соединений, что может вести к избыточному числу состояний.

4.6. Контроль одновременной работы

Конечная точка ограничивает суммарное число входящих потоков, которые партнёр может создать, разрешая создавать лишь потоки, чей идентификатор меньше max_streams * 4 + first_stream_id_of_type (см. таблицу 1). Начальные ограничения задаются в транспортных параметрах (параграф 18.2. Определения транспортных параметров), а последующие анонсируются в кадрах MAX_STREAMS (параграф 19.11. Кадр MAX_STREAMS). Для односторонних и двухсторонних потоков применяются разные ограничения.

Получение параметра max_streams или кадра MAX_STREAMS со значением больше 2^{60} указывает идентификатор потока, который не может быть выражен целым число переменного размера (16. Представление целочисленных полей переменного размера). Соединение **должно** закрываться немедленно с ошибкой TRANSPORT_PARAMETER_ERROR в случае транспортного параметра и FRAME_ENCODING_ERROR для значения в кадре (10.2. Незамедлительное закрытие).

Конечным точкам **недопустимо** выходить за установленный партнёром предел. Получение кадра с идентификатором потока сверх заданного предела конечная точка **должна** считать ошибкой соединения STREAM_LIMIT_ERROR (11. Обработка ошибок).

После анонсирования получателем своего ограничения в кадре MAX_STREAMS снижение заданного предела не будет оказывать влияния и такие кадры MAX_STREAMS **должны** игнорироваться.

Как и для управления потоком данных, этот документ оставляет за реализацией выбор времени и способа анонсирования ограничений партнёру с помощью MAX_STREAMS. Реализации могут повышать предел по мере закрытия потоков для сохранения числа доступных партнёру потоков более или менее постоянным.

Конечной точке, не способной создать новый поток по причине ограничения партнёра, **следует** передать кадр STREAMS_BLOCKED (параграф 19.14). такая сигнализация может быть полезна для отладки. Конечной точке **недопустимо** ждать получения этого сигнала перед анонсированием дополнительного кредита, поскольку такое ожидание означает блокирование партнёра по меньшей мере на период кругового обхода и может даже приводить к бессрочному блокирования, если партнёр не передаёт кадров STREAMS_BLOCKED.

5. Соединения

Соединение QUIC является общим состоянием клиента и сервера. Каждое соединение начинается с фазы согласования (handshake), когда две конечных точки организуют общий секрет, используя протокол криптографического согласования [QUIC-TLS], и согласуют протокол приложения. Согласование (раздел 7) подтверждает, что точки готовы взаимодействовать (параграф 8.1) и задали параметры соединения (параграф 7.4).

[1]Здесь и далее слова «сверх окончательного размера» означают данные со смещением равным или превышающим этот размер. *Прим. перев.*

Использование прикладного протокола в фазе согласования несколько ограничено. 0-RTT позволяет клиенту передать данные приложения до получения отклика от сервера, однако при этом не обеспечивается защиты от атак с повторным использованием (replay), как указано в параграфе 9.2 [QUIC-TLS]. Сервер также может передать клиенту данные приложения до получения финальных сообщения криптографического согласования, которое позволяет ему подтвердить подлинность и живучесть клиента. Эти возможности позволяют прикладному протоколу предложить возможность согласовать некоторые криптографические гарантии для снижения задержки.

Использование идентификаторов соединений (параграф 5.1) позволяет менять путь соединения через сеть в случае прямого перемещения конечной точки или вызванных промежуточными устройствами изменений. В разделе 9 рассматривается смягчение проблем безопасности и приватности при перемещении соединений.

Для соединений, которые стали ненужными или нежелательными, имеется несколько способов разрыва, описанных в разделе 10.

5.1. Идентификаторы соединения

Каждое соединение обладает набором идентификаторов, которые могут указывать соединение. Идентификаторы выбираются конечными точками независимо.

Основным назначением идентификаторов соединений является предотвращение отправки пакетов соединения QUIC не той конечной точке в случае смены адресации протоколов нижележащих уровней (UDP, IP). Каждая конечная точка выбирает идентификаторы соединения в зависимости от реализации (и, возможно, развёртывания) и эти идентификаторы позволяют пакетам соединения попадать в конечную точку и распознаваться при получении. Множество идентификаторов соединения позволяют конечной точке передавать пакеты так, чтобы наблюдатель в пути не мог связать пакеты с конечной точкой без взаимодействия с нею (9.5. Влияние переноса соединения на приватность). В идентификаторы соединений **недопустимо** включать какие-либо сведения, которые позволят стороннему наблюдателю (не взаимодействующему с эмитентом идентификатора) сопоставить их с другими идентификаторами того же соединения. Например, **недопустимо** использовать один идентификатор несколько раз в данном соединении.

Пакеты с длинными заголовками включают поля Source Connection ID и Destination Connection ID, которые служат для установки идентификаторов в новых соединениях (7.2. Согласование идентификаторов соединений). В пакеты с коротким заголовком (параграф 17.3) включается лишь Destination Connection ID без явного указания размера (предполагается, что конечные точки знает его). Конечные точки с балансировщиками нагрузки, распределяющими пакеты по идентификаторам соединений, могут согласовать балансировку по фиксированному размеру идентификаторов или выбрать схему их кодирования. Фиксированная часть может представлять явный размер, что позволяет идентификатору иметь переменный размер с возможностью его использования для балансировки нагрузки.

Пакет согласования версии (Version Negotiation, параграф 17.2.1) возвращает идентификаторы соединений, выбранные клиентом, что обеспечивает корректную маршрутизацию в направлении клиента и указывает, что пакет является откликом на переданный клиентом пакет.

Можно использовать пустой (размер 0) идентификатор соединения, если тот не нужен для маршрутизации корректной конечной точке. Однако мультиплексирование соединений с одним локальным адресом IP и номером порта при пустом идентификаторе соединения столкнётся с отказом при переносе соединения, перестройке NAT или повторном использовании клиентского порта. Конечной точке **недопустимо** применять один адрес IP и порт для нескольких одновременных соединений с пустыми идентификаторами, пока нет уверенности в том, что идентификаторы не нужны.

При использовании конечной точкой пустого идентификатора соединения она должна гарантировать, что у партнёра имеется идентификатор соединения для выбора пакетов, передаваемых этой конечной точке. Эти идентификаторы передаются конечной точкой в кадрах NEW_CONNECTION_ID (параграф 19.15).

5.1.1. Выдача идентификаторов соединения

С каждым идентификатором соединения связан порядковый номер помогающий определить принадлежность кадров NEW_CONNECTION_ID или RETIRE_CONNECTION_ID к одному идентификатору. Начальный идентификатор соединения, выданный конечной точкой, передаётся в поле Source Connection ID пакета с длинным заголовком (параграф 17.2) в процессе согласования. Этот идентификатор имеет порядковый номер 0. Если передаётся параметр транспорта preferred_address порядковым номером представленного идентификатора соединения будет 1.

Дополнительные идентификаторы соединения передаются в кадрах NEW_CONNECTION_ID (параграф 19.15) и для каждого нового идентификатора порядковый номер **должен** увеличиваться на 1. Идентификатор соединения, который клиент выбрал для первого переданного им поля Destination Connection ID, и любой идентификатор соединения из пакета Retry не имеет порядкового номера.

Конечная точка, выдавшая идентификатор соединения, **должна** воспринимать пакеты с этим идентификатором в течение срока действия соединения или до аннулирования идентификатора кадром RETIRE_CONNECTION_ID (параграф 19.16). Выданные и не аннулированные идентификаторы соединения считаются активными и пригодны для использования в любых пакетах соединения в любой момент. Активным считается и идентификатор соединения выданный сервером в транспортном параметре preferred_address.

Конечной точке **следует** обеспечивать для партнёра достаточное число доступных и не использованных идентификаторов соединения. Конечные точки анонсируют число поддерживаемых активных идентификаторов соединения через транспортный параметр active_connection_id_limit. Конечной точке **недопустимо** передавать идентификаторы соединения сверх заданного партнёром предела. Конечная точка **может** передавать идентификаторы, которые временно выходят за установленный партнёром предел, если кадр NEW_CONNECTION_ID также требует изъятия излишка, путём включения достаточно большого значения в поле Retire Prior To. Кадр NEW_CONNECTION_ID может вызвать добавление конечной точкой активных идентификаторов соединения и исключение других на основе значения поля Retire Prior To. После обработки кадра NEW_CONNECTION_ID с добавлением и удалением активных идентификаторов конечная точка **должна** закрыть соединение с ошибкой CONNECTION_ID_LIMIT_ERROR, если число активных идентификаторов превышает значение транспортного параметра active_connection_id_limit.

Конечной точке **следует** предоставлять новый идентификатор соединения, когда партнёр отзывает идентификатор. Если конечная точка предоставляет меньше заданного партнёром числа (active_connection_id_limit) активных идентификаторов, она **может** добавить новый идентификатор при получении пакета с не использованным ранее идентификатором соединения. Конечная точка **может** ограничить общее число выдаваемых идентификаторов соединения для предотвращения риска выхода за пределы (см. 10.3.2. Расчёт маркера Stateless Reset). Конечная точка **может** также ограничивать выпуск идентификаторов соединения для снижения числа поддерживаемых состояний на уровне пути, таких как статус проверки пути, поскольку партнёр может взаимодействовать с ней по числу путей, соответствующему количеству выданных идентификаторов соединения.

Конечной точке, которая инициирует перенос соединения и требует непустых идентификаторов, **следует** гарантировать, что пул доступных партнёру идентификаторов позволяет тому использовать при переходе новый идентификатор, поскольку партнёр не сможет ответить в противном случае.

Конечная точка, выбравшая пустой идентификатор соединения в процессе согласования, не может выбрать новый идентификатор. Поле Destination Connection ID с размером 0 используется во всех пакетах для этой точки по любому пути через сеть.

5.1.2. Потребление и отзыв идентификаторов соединения

Конечная точка может сменить используемый для партнёра идентификатор соединения на доступный в любой момент работы соединения. Конечная точка потребляет идентификатор соединения в ответ на перемещение партнёра (см. 9.5. Влияние переноса соединения на приватность).

Конечная точка поддерживает набор полученных от партнёра идентификаторов соединения, любой из которых она может применять для отправки пакетов. Когда конечная точка желает исключить идентификатор соединения из использования, она передаёт партнёру кадр RETIRE_CONNECTION_ID. Отправка кадра RETIRE_CONNECTION_ID показывает, что идентификатор соединения не будет использоваться снова и запрашивает у партнёра его замену в кадре NEW_CONNECTION_ID.

Как отмечено в параграфе 9.5, конечная точка ограничивает использование идентификатора соединения пакетами, передаваемыми с одного локального адреса на один адрес получателя. Конечным точкам **следует** отзывать идентификаторы соединений, когда они перестают использовать локальный или удалённый адрес, для которого применялся идентификатор соединения.

В некоторый случаях конечное точке может потребоваться прекратить восприятие ранее выданных идентификаторов соединения. Она может заставить своего партнёра отозвать идентификатор соединения путём отправки тому кадра NEW_CONNECTION_ID с увеличенным значением поля Retire Prior To. Конечное точке **следует** продолжать восприятие ранее выданных идентификаторов соединения, пока они не отозваны партнёром. Если конечная точка больше не может обрабатывать выданные идентификаторы соединения, она **может** закрыть соединение.

При получении поля Retire Prior To партнёр **должен** прекратить использование соответствующих идентификаторов соединения и отозвать их в кадрах RETIRE_CONNECTION_ID до добавления новых идентификаторов в число активных идентификаторов соединения. Такой порядок позволяет конечной точке заменить все активные идентификаторы, не оставляя партнёра без доступных идентификаторов и не превышая установленного им предела (параметр транспорта active_connection_id_limit, см. 18.2. Определения транспортных параметров). Неспособность прекратить использование идентификаторов соединения по запросу может приводить к отказу соединения, поскольку выдавшая идентификаторы конечная точка может быть не способна продолжать использование идентификаторов с активным соединением.

Конечной точке **следует** ограничивать число локально отозванных идентификаторов соединения, для которых кадры RETIRE_CONNECTION_ID ещё не подтверждены. Конечное точке **следует** разрешать передачу и отслеживание числа кадров RETIRE_CONNECTION_ID, по меньшей мере вдвое превышающего значение транспортного параметра active_connection_id_limit. Конечной точке **недопустимо** забывать идентификатор соединения без его отзыва, хотя она **может** считать наличие требующих отзыва идентификаторов соединения сверх этого предела ошибкой соединения типа CONNECTION_ID_LIMIT_ERROR.

Конечной точке **не следует** обновлять поле Retire Prior To до приёма кадров RETIRE_CONNECTION_ID, отзывающих идентификаторы сообщений, указанных в предыдущем Retire Prior To.

5.2. Сопоставление пакетов с соединением

Входящие пакеты классифицируются при получении. Пакеты могут быть связаны с одним из имеющихся соединений или (для серверов) создавать новое соединение. Конечные точки пытаются связать пакет с имеющимся соединением. Если в пакете содержится непустое поле Destination Connection ID, соответствующее имеющемуся соединению, QUIC выполняет соответствующую обработку пакета. Как указано в параграфе 5.1, с соединением может быть связано множество идентификаторов. Если поле Destination Connection ID имеет нулевой размер и адресная информация в пакете соответствует адресной информации, используемой конечной точкой для идентификации соединения с пустым идентификатором, QUIC обрабатывает пакет как часть соединения. Конечная точка может использовать для идентификации лишь IP-адрес и порт получателя или адреса отправителя и получателя, хотя это делает соединения более слабыми, как отмечено в параграфе 5.1.

Конечная точка может передать Stateless Reset (параграф 10.3) для любого пакета, который она не может связать с имеющимся соединением. Это позволяет партнёру быстрее обнаруживать непригодность соединения.

Соответствующие имеющемуся соединению пакеты отбрасываются, если они не согласуются с состоянием соединения. Например, отбрасываются пакеты, указывающие иную версию протокола, или при неудачном снятии защиты после того, как ожидаемые ключи стали доступными.

Недействительные пакеты без строгой защиты целостности, такие как Initial, Retry, Version Negotiation, **можно** отбрасывать. Конечная точка **должна** генерировать ошибку соединения, если обработка содержимого таких пакетов была выполнена до обнаружения ошибки, или полностью отменить изменения, внесённые при такой обработке.

5.2.1. Обработка пакетов клиентом

Переданные клиентам действительные пакеты всегда включают поле Destination Connection ID, соответствующее выбранному клиентом значению. Клиенты, выбравшие получение пустого идентификатора соединения, могут применять для идентификации соединения локальный адрес и порт. Пакеты, не соответствующие имеющемуся соединению (Destination Connection ID или локальный адрес IP и порт при пустом идентификаторе), отбрасываются.

В результате потери или нарушения порядка пакетов клиент может получить пакеты, зашифрованные с ключом, которые ещё не рассчитан. Клиент **может** отбросить такие пакеты или буферизовать их в ожидании последующих пакетов, которые позволят рассчитать ключ.

При получении клиентом пакета, использующего отличную от исходно выбранной версию, он **должен** отбросить пакет.

5.2.2. Обработка пакетов сервером

Если сервер получает пакет, указывающий неподдерживаемую версию и размер пакета достаточно велик, чтобы инициировать новое соединение для любой поддерживаемой версии, ему **следует** передать пакет Version Negotiation, как указано в параграфе 6.1. Сервер **может** ограничить число пакетов, на которые он отвечает пакетом Version Negotiation. Серверы **должны** отбрасывать мелкие пакеты, указывающие неподдерживаемую версию.

Первые пакет для неподдерживаемой версии может использовать иную семантику и кодирование для зависимых от версии полей. В частности, разные версии могут применять свои ключи защиты пакетов. Не поддерживающий конкретную версию сервер вряд ли сможет расшифровать данные пакета или корректно интерпретировать результат. Серверам **следует** отвечать пакетом Version Negotiation при условии достаточно большого размера дейтаграммы.

Пакеты с поддерживаемой версией или без поля Version сопоставляются с соединением по идентификатору соединения или локальным адресом и портом (при пустом идентификаторе соединения). Эти пакеты обрабатываются с использованием выбранного соединения, в иных случаях сервер выполняет описанные ниже действия.

Для пакетов Initial, полностью соответствующих данной спецификации, сервер выполняет согласование (раздел 7) с фиксацией сервером выбранной клиентом версии.

Если сервер отвергает новое соединение, ему **следует** передать пакет Initial с кадром CONNECTION_CLOSE, содержащим код ошибки CONNECTION_REFUSED.

Пакеты 0-RTT сервер **может** буферизовать в ограниченном количестве в ожидании последующего получения пакета Initial. Клиенты не могут передавать пакеты Handshake до получения отклика от сервера, поэтому серверу **следует** игнорировать такие пакеты.

В остальных случаях сервер **должен** отбрасывать входящие пакеты.

5.2.3. Простые балансировщики нагрузки

Развёртывание сервера может использовать распределение нагрузки между несколькими серверами, используя лишь IP-адреса и порты отправителей и получателей. Смена в пакетах адреса или порта у клиента может приводить к пересылке пакета не тому серверу. В таких системах может применяться один из указанных ниже способов обеспечения непрерывности соединения при смене адреса клиента.

- Сервер может применять отдельный (out-of-band) механизм для пересылки пакетов корректному серверу на основе идентификатора соединения.
- Если серверы могут использовать выделенные адреса IP или порты, отличающиеся от тех, по которым исходно подключался клиент, они могут использовать транспортный параметр preferred_address для запроса у клиента переноса соединения на выделенный адрес.

Серверу, который не реализует решения по поддержке непрерывности соединения при смене адреса клиента, **следует** указывать, что перенос не поддерживается с помощью транспортного параметра disable_active_migration. Этот параметр не запрещает перемещение клиента после действий клиента в соответствии с параметром preferred_address.

Серверы, использующие эту простую форму распределения нагрузки, **должны** избегать предсказания сброса без учёта состояния (21.11. Предсказание Stateless Reset).

5.3. Операции на соединениях

Этот документ не задаёт API для QUIC, определяя лишь набор функций для соединений QUIC, которые прикладные протоколы могут использовать. Протоколы приложений могут считать, что реализация QUIC обеспечивает интерфейс, включающий описанные в этом параграфе операции. Реализация, рассчитанная на определённый прикладной протокол, может предоставлять лишь нужную данному протоколу часть описанных операций.

При реализации роли клиента протокол приложения может:

- создавать соединения, начинающиеся с обмена, описанного в разделе 7;
- включать Early Data при доступности данных;
- получать информацию о восприятии или отклонении сервером Early Data.

При реализации роли сервера протокол приложения может:

- прослушивать входящие соединения, готовясь к обмену, описанному в разделе 7;
- при поддержке Early Data встраивать контролируемые приложением данные в «квитанцию» (ticket) возобновления TLS, передаваемую клиенту;
- при поддержке Early Data извлекать контролируемые приложением данные из клиентских «квитанций» восстановления и воспринимать или отвергать Early Data на основе этих сведений.

В любой из роле протокол приложения может:

- настраивать минимальные значения для исходного числа разрешённых потоков каждого типа, передаваемые в транспортных параметрах (параграф 7.4);
- управлять выделением ресурсов для приёмных буферов путём установки ограничений контроля потока данных на уровне соединения и потоков;
- определять успешное завершение согласования (handshake) или его продолжение;
- предохранять соединение от закрытия «втихую» путём генерации кадров PING (параграф 19.2) или запроса у транспортного уровня передачи дополнительных кадров до тайм-аута бездействия (параграф 10.1);
- незамедлительно закрывать соединение (параграф 10.2).

6. Согласование версии

Согласование версии позволяет серверу узнать, поддерживает ли он используемую клиентом версию. Отправитель передаёт пакет Version Negotiation в ответ на каждый пакет, который может инициировать новое соединение (см. 5.2. Сопоставление пакетов с соединением).

Размер первого пакета, переданного клиентом, определяет возможность отправки сервером пакета Version Negotiation. Клиентам, поддерживающим несколько версий QUIC, **следует** обеспечить для первой передаваемой дейтаграммы UDP размер, являющийся наибольшим из всех минимальных значений размера дейтаграммы в поддерживаемых клиентом версиях, используя при необходимости кадры PADDING (параграф 19.1). Это гарантирует ответ сервера при наличии поддерживаемой обеими сторонами версии. Сервер может не передавать пакет Version Negotiation, если полученная дейтаграмма меньше минимального размера, указанного в другой версии (см. 14. Размер дейтаграмм).

6.1. Передача пакетов согласования версии

Если для сервера не подходит выбранная клиентом версия, он отвечает пакетом Version Negotiation (параграф 17.2.1), включающий список поддерживаемых сервером версий. Конечной точке **недопустимо** передавать пакет Version Negotiation в ответ на получение пакета Version Negotiation.

Это позволяет серверу обрабатывать пакеты неподдерживаемых версий без сохранения состояния. Поскольку пакет Initial или переданный в ответ пакет Version Negotiation может потеряться, клиент будет передавать новые пакеты, пока не получит ответ или не откажется от попытки соединения.

Сервер **может** ограничить число передаваемых пакетов Version Negotiation. Например, сервер, способный распознавать пакеты как 0-RTT, может отказаться от передачи пакетов Version Negotiation в ответ на пакеты 0-RTT, ожидая получить пакет Initial.

6.2. Обработка пакетов согласования версии

Пакеты Version Negotiation предназначены для определения в будущем функциональности, которая позволит QUIC согласовывать версию протокола для соединения. Будущие спецификации Standards Track могут изменить поведение реализаций с поддержкой нескольких версий QUIC при получении пакетов Version Negotiation в ответ на попытку организации соединения с использованием данной версии.

Клиент, поддерживающий лишь данную версию QUIC, **должен** отказаться от текущей попытки соединения при получении сообщения Version Negotiation, за исключением двух указанных ниже случаев. Клиент **должен** отбрасывать любой пакет Version Negotiation, если он получил и успешно обработал другие пакеты, включая предшествующий пакет Version Negotiation. Клиент **должен** отбрасывать пакет Version Negotiation, указывающий выбранную клиентом версию QUIC.

Согласование версии оставлено для будущих спецификаций Standards Track. В частности, эти будущие документы обеспечат устойчивость к атакам на понижение версии (21.12. Понижение версии).

6.3. Использование зарезервированных версий

Чтобы клиент мог в будущем использовать новую версию, клиентам нужно корректно обрабатывать неподдерживаемые версии. Некоторые номера версий (0x?a?a?a?a, как указано в разделе 15) зарезервированы для включения в поля, содержащие номер версии.

Конечные точки **могут** добавлять резервные версии в любое поле, где неизвестная или неподдерживаемая версия игнорируется, для проверки корректности игнорирования значения партнёром. Например, конечная точка может включить резервную версию в пакет Version Negotiation (см. параграф 17.2.1). Конечные точки **могут** передавать пакеты с резервной версией для проверки корректности отбрасывания пакетов партнёром.

7. Согласование криптографии и транспорта

QUIC использует комбинированное согласование криптографии и транспорта для минимизации задержки при организации соединений. Для криптографического согласования QUIC использует кадр CRYPTO (параграф 19.6). Определённая этим документом версия QUIC указывается идентификатором 0x00000001 и использует TLS, как описано в [QUIC-TLS]. Иная версия QUIC может применять другой протокол криптографического согласования.

QUIC обеспечивает надёжную, упорядоченную доставку данных криптографического согласования с использованием защиты пакетов QUIC для максимально возможного шифрования протокола согласования. Криптографическое согласование **должно** обеспечивать указанные ниже свойства.

- Обмен аутентифицированными ключами, в котором:
  - подлинность сервера проверяется всегда;
  - может проверяться подлинность клиента;
  - каждое соединение создаёт разные и не связанные между собой ключи;

- ключевой материал пригоден для защиты пакетов 0-RTT и 1-RTT.

- Аутентифицированный обмен значениями транспортных параметров обеих конечных точек и защита конфиденциальности для транспортных параметров сервера (параграф 7.4).

- Аутентифицированное согласование прикладного протокола (TLS использует для этого ALPN[1] [ALPN]).

Кадр CRYPTO можно передать в разных пространствах номеров пакетов (параграф 12.3). Смещения в кадрах CRYPTO, используемые для гарантии упорядоченной доставки данных криптографического согласования, начинаются с 0 в каждом пространстве номеров пакетов.

На рисунке 4 упрощенно показано согласование и обмен пакетами и кадрами в процессе согласования. По возможности разрешается обмен данными приложения в процессе согласования (помечено *). По завершении согласования конечные точки могут свободно обмениваться данными.

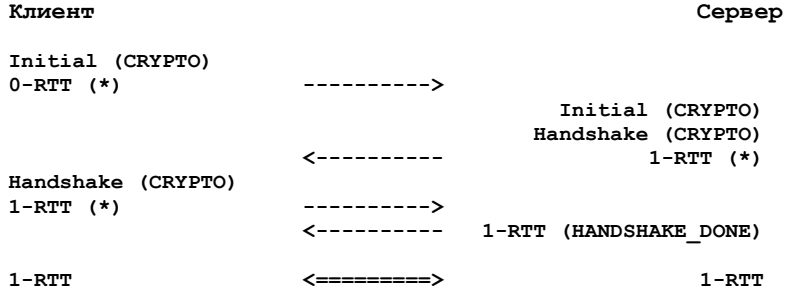

Рисунок 4. Упрощенное согласование QUIC.

Конечные точки могут использовать переданные при согласовании пакеты для тестирования поддержки ECN[2] (см. параграф 13.4). Конечная точка определяет поддержку ECN, контролируя наличие в кадрах ACK, подтверждающих первые пакеты, полей ECN Count, как указано в параграфе 13.4.2.

Конечные точки **должны** явно согласовывать прикладной протокол, что позволяет избежать ситуаций с разногласиями по поводу используемого протокола.

7.1. Примеры потока согласования

Детали интеграции TLS с QUIC представлены в [QUIC-TLS], но здесь даны некоторые рекомендации. Расширение обмена для проверки адреса клиента описано в параграфе 8.1.2.

По завершении обмена для проверки адреса используется криптографическое согласование для ключей. В криптографическом согласовании применяются пакеты Initial (параграф 17.2.2) и Handshake (параграф 17.2.4).

На рисунке 5 приведён обзор согласования 1-RTT. Каждая строка показывает пакет QUIC с указанным типом и номером. Затем следуют кадры, обычно содержащиеся в этих пакетах. Например, первый пакет имеет тип Initial, номер 0 и содержит кадр CRYPTO с ClientHello.

Можно объединить несколько пакетов QUIC (даже разных типов) в одной дейтаграмме UDP (см. параграф 12.2). В результате согласование может состоять всего из 4 дейтаграмм UDP (зависит от присущих протоколу ограничений, таких как контроль перегрузки и предотвращения усиления). Например, первая отправка сервера содержит пакеты Initial, Handshake и данные 0.5-RTT в пакетах 1-RTT.

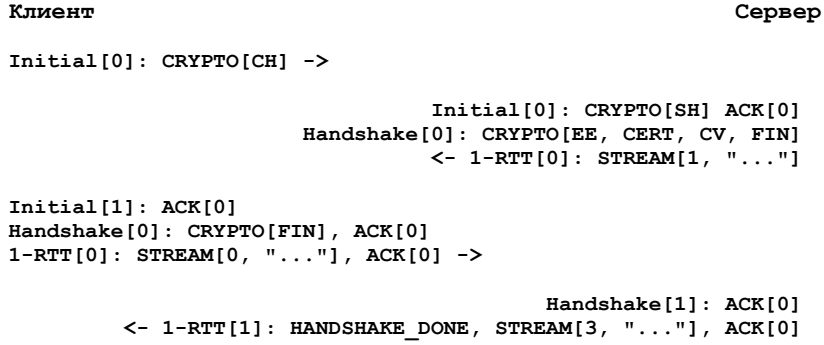

Рисунок 5. Пример согласования 1-RTT.

На рисунке 6 приведён пример соединения с согласованием 0-RTT и одним пакетом данных 0-RTT. Как указано в параграфе 12.3, сервер подтверждает данные 0-RTT в пакетах 1-RTT, а клиент передаёт пакеты 1-RTT в том же пространстве номеров.

7.2. Согласование идентификаторов соединений

Идентификатор соединения служит для обеспечения согласованной маршрутизации пакетов, как описано в параграфе 5.1. Длинный заголовок содержит два идентификатора соединений - Destination Connection ID выбирается получателем пакета и служит для обеспечения согласованной маршрутизации, Source Connection ID служит для установки Destination Connection ID, используемого партнёром.

В процессе согласования применяются пакеты с длинным заголовком (параграф 17.2) для организации идентификаторов соединений, используемых обеими конечными точками. Каждая конечная точка использует поле Source Connection ID для указания идентификатора соединения, указываемого в поле Destination Connection ID

[1]Application-Layer Protocol Negotiation - согласование протокола прикладного уровня.
[2]Explicit Congestion Notification - явное уведомление о перегрузке.

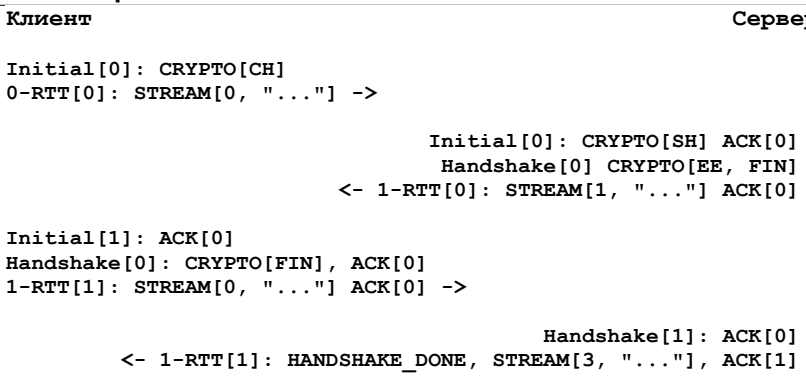

Рисунок 6. Пример согласования 0-RTT.

передаваемых ей пакетов. После обработки первого пакета Initial каждая конечная точка устанавливает поле Destination Connection ID в последующих пакетах в соответствии с полученным значением поля Source Connection ID.

Когда пакет Initial передаётся клиентом, не получавшим ранее от сервера пакета Initial или Retry, клиент указывает в поле Destination Connection ID непредсказуемое значение. Размер поля Destination Connection ID **должен** быть не менее 8 байтов. Пока не будет получен пакет от сервера, клиент **должен** использовать одно значение Destination Connection ID во всех пакетах соединения.

Поле Destination Connection ID из первого пакета от клиента служит для определения ключей защиты пакетов Initial. Эти ключи меняются после приёма пакета Retry (см. параграф 5.2 в [QUIC-TLS]).

Клиент указывает в поле Source Connection ID выбранное им значение и указывает его размер в поле Source Connection ID Length.

Пакеты 0-RTT в первой отправке используют те же значения Destination Connection ID и Source Connection ID, что и в первом пакете Initial от клиента.

При получении от сервера пакета Initial или Retry клиент использует представленное сервером поле Source Connection ID в качестве значения Destination Connection ID для последующих пакетов, включая любые пакеты 0-RTT. Это значит, что клиенту может потребоваться изменить идентификатор соединения в поле Destination Connection ID дважды в процессе организации соединения - одни раз в ответ на пакет Retry и другой в ответ на пакет Initial от сервера. После получения клиентом действительного пакета Initial от сервера он **должен** отбрасывать в этом соединении все последующие пакеты с другим Source Connection ID.

Клиент **должен** менять Destination Connection ID в передаваемых пакетах лишь в ответ на первый принятый пакет Initial или Retry. Сервер **должен** установить Destination Connection ID для отправляемых пакетов на основе первого полученного пакета Initial. Последующие изменения Destination Connection ID разрешены лишь на значения из кадров NEW_CONNECTION_ID. Если последующие пакеты Initial включают другое значение Source Connection ID, они **должны** отбрасываться. Это позволяет избежать непредсказуемых результатов при обработке нескольких пакетов Initial с разными Source Connection ID без сохранения состояния.

Поле Destination Connection ID, передаваемое конечной точкой, может меняться в течение срока действия соединения, особенно при переносе соединения (раздел 9 и параграф 5.1.1).

7.3. Аутентификация идентификаторов соединений

Выбор каждой конечной точкой идентификатора соединения в процессе согласования аутентифицируется путём включения всех значений в транспортные параметры (см. параграф 7.4). Это гарантирует для всех идентификаторов соединений, используемых при согласовании, проверку подлинности в процессе криптографического согласования.

Каждая конечная точка включает значение Source Connection ID из первого переданного пакета Initial в транспортный параметр initial_source_connection_id (см. параграф 18.2). Сервер включает поле Destination Connection ID из первого полученного от клиента пакета Initial в транспортный параметр original_destination_connection_id. Если сервер передаёт пакет Retry это относится к первому пакету Initial, полученному до отправки Retry. Если сервер передаёт пакет Retry, он также включает поле Source Connection ID из пакета Retry в транспортный параметр retry_source_connection_id.

Предоставленные партнёром значения этих транспортных параметров **должны** совпадать со значениями, используемыми конечной точкой в полях Destination Connection ID и Source Connection ID передаваемых (и принимаемых для сервера) пакетов Initial. Конечные точки **должны** проверить соответствие полученных транспортных параметров полученным идентификаторам соединений. Включение идентификаторов соединений в транспортные параметры и их проверка предотвращают влияние атакующего на выбор идентификатора соединения для успешного подключения путём вставки пакетов с выбранным атакующим идентификатором соединения в процессе согласования.

Конечная точка **должна** считать отсутствие транспортного параметра initial_source_connection_id или транспортного параметра original_destination_connection_id от сервера как ошибку соединения TRANSPORT_PARAMETER_ERROR.

Конечная точка **должна** считать ошибкой TRANSPORT_PARAMETER_ERROR или PROTOCOL_VIOLATION:

- отсутствие транспортного параметра retry_source_connection_id от сервера после получения пакета Retry;
- присутствие транспортного параметра retry_source_connection_id, когда пакет Retry не был получен;
- несоответствие значений. полученных от партнёра в этих транспортных параметрах, и значений, переданных в полях Destination Connection ID или Source Connection ID пакетов Initial.

При выборе пустого идентификатора соединения соответствующие транспортные параметры имеют размер 0.

На рисунке 7 показаны идентификаторы соединения (DCID=Destination Connection ID, SCID=Source Connection ID), используемые для полного согласования (handshake). Указан обмен пакетами Initial, а также последующий обмен пакетами 1-RTT, включающий идентификаторы соединения, организованные при согласовании.

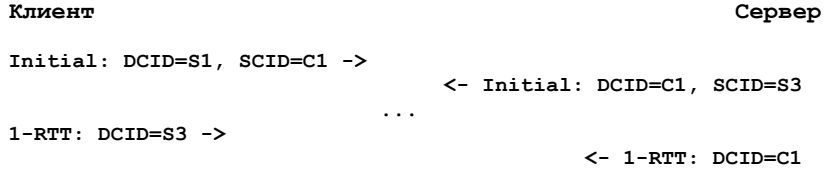

Рисунок 7. Использование идентификаторов соединения при согласовании.

На рисунке 8 показано похожее согласование с включением пакета Retry.

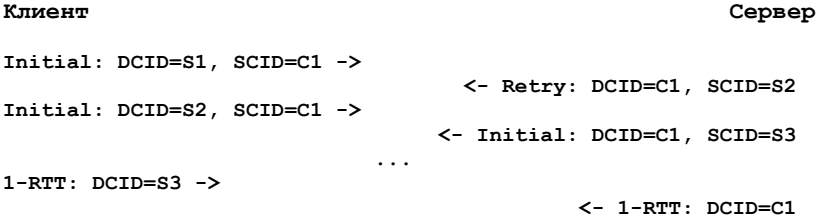

Рисунок 8. Использование идентификаторов соединения при согласовании с Retry.

В обоих случаях (рисунки 7 и 8) клиент устанавливает для транспортного параметра initial_source_connection_id значение C1. При согласовании без пакета Retry (рисунок 7) сервер устанавливает original_destination_connection_id = S1 (это значение выбрано клиентом) и initial_source_connection_id = S3 и не включает транспортный параметр retry_source_connection_id. В согласовании с Retry (рисунок 8) сервер устанавливает original_destination_connection_id = S1, retry_source_connection_id = S2 и initial_source_connection_id = S3.

7.4. Транспортные параметры

В процессе организации соединения обе конечные точки заявляют свои транспортные параметры с использованием аутентификации. Конечные точки должны выполнять ограничения, заданные каждым параметром. Описания параметров включают правила обработки.

Транспортные параметры каждая конечная точка объявляет в одностороннем порядке и может выбрать значения независимо от значений своего партнёра. Представление транспортных параметров описано в разделе 18.

QUIC включает кодированные транспортные параметры в криптографическое согласование, по завершении которого становятся доступными транспортные параметры, заявленные партнёром. Каждая конечная точка проверяет действительность полученных от партнёра значений. Определения транспортных параметров даны в параграфе 18.2.

Получение транспортного параметра с недействительным значением конечная точка **должна** считать ошибкой соединения типа TRANSPORT_PARAMETER_ERROR. Конечным точкам **недопустимо** передавать параметр более одного раза в данном расширении транспортных параметров. Получение дубликата транспортного параметра конечной точке **следует** считать ошибкой соединения типа TRANSPORT_PARAMETER_ERROR.

Конечные точки применяют транспортные параметры для аутентификации идентификаторов соединения при согласовании (см. параграф 7.3).

ALPN [ALPN] позволяет клиентам предложить несколько прикладных протоколов на этапе организации соединения. Транспортные параметры, включённые клиентом в процессе согласования, применяются ко всем прикладным протоколам, предлагаемым клиентом. Прикладные протоколы могут рекомендовать значения для транспортных параметров, такие как начальные ограничения контроля потока данных. Однако прикладные протоколы, устанавливающие ограничения для транспортных параметров, могут не позволить клиенту предложить множество прикладных протоколов, если возникают конфликты.

7.4.1. Значения транспортных параметров для 0-RTT

Использование 0-RTT зависит как от клиента, так и от сервера, применяющих параметры, согласованные в предшествующем соединении. Для включения 0-RTT конечные точки сохраняют значения транспортных параметров сервера со всеми сеансовыми квитанциями, полученными в соединении. Конечные точки также сохраняют сведения, требуемые прикладному протоколу или криптографическому согласованию (см. параграф 4.6 в [QUIC-TLS]). Значения сохранённых транспортных параметров используются при попытке применить 0-RTT с сеансовыми квитанциями.

Сохранённые параметры транспорта применяются к новому соединению, пока не будет завершено согласование и клиент не начнёт передачу пакетов 1-RTT. По завершении согласования клиент применяет заданные в нем параметры транспорта. Запоминаются не все транспортные параметры, поскольку некоторые из них не применимы к будущим соединениям или не влияют на использование 0-RTT.

Определение нового транспортного параметра (параграф 7.4.2) **должно** указывать режим его сохранения для 0-RTT (обязательно, опционально, запрещено). Клиенту не нужно сохранять параметры, которые он не может обработать.

Клиенту недопустимо применять сохранённые значения параметров ack_delay_exponent, max_ack_delay, initial_source_connection_id, original_destination_connection_id, preferred_address, retry_source_connection_id, stateless_reset_token и он **должен** использовать новые значения, полученные от сервера при согласовании. Если сервер не представил эти параметры, используются принятые по умолчанию значения.

Пытающийся передать данные 0-RTT клиент **должен** запомнить все другие транспортные параметры сервера, которые он может обрабатывать. Сервер может запоминать эти параметры или хранить их копии с защитой целостности в квитанции, восстанавливая информацию при восприятии данных 0-RTT. Сервер использует транспортные параметры при решении вопроса о восприятии данных 0-RTT.

Если сервер воспринимает данные 0-RTT, ему **недопустимо** снижать значения пределов или менять другие значения, которые могут быть нарушены клиентом через его данные 0-RTT. В частности, воспринимающему данные 0-RTT серверу **недопустимо** устанавливать значения меньше сохранённых для параметров (параграф 18.2):

- active_connection_id_limit;
- initial_max_data;
- initial_max_stream_data_bidi_local;
- initial_max_stream_data_bidi_remote;
- initial_max_stream_data_uni;
- initial_max_streams_bidi;
- initial_max_streams_uni.

Пропуск некоторых параметров транспорта или установка для них значения 0 может приводить к тому, что данные 0-RTT будут разрешены, но не пригодны для использования. Для подмножества применимых транспортных параметров, разрешающих передачу данных приложения, **следует** задавать отличные от 0 значения для 0-RTT. Это включает initial_max_data и (1) initial_max_streams_bidi и initial_max_stream_data_bidi_remote или (2) initial_max_streams_uni и initial_max_stream_data_uni.

Сервер может задать более высокие начальные ограничения для контроля потока данных, нежели сохранённые значения, которые клиент применяет при отправке 0-RTT. По завершении согласования клиент обновляет ограничения контроля потока данных на всех передающих потоках, используя новые значения initial_max_stream_data_bidi_remote и initial_max_stream_data_uni.

Сервер **может** сохранять и восстанавливать переданные ранее значения max_idle_timeout, max_udp_payload_size, disable_active_migration и отвергать 0-RTT при выборе меньших значений. Снижение значений этих параметров при одновременном восприятии данных 0-RTT может снижать производительность соединения. В частности, снижение max_udp_payload_size может вызывать отбрасывание пакетов, ведущее к снижению производительности по сравнению с отклонением данных 0-RTT.

Сервер **должен** отвергать данные 0-RTT, если восстановленные значения параметров транспорта не могут поддерживаться.

При отправке кадров в пакетах 0-RTT клиент **должен** применять лишь запомненные параметры транспорта и **недопустимо** использовать обновлённые значения, полученные от сервера в транспортных параметрах или кадрах из пакетов 1-RTT. Например, ограничения контроля потока данных из запомненных параметров транспорта применяются ко всем пакетам 0-RTT даже при увеличении значений параметров в согласовании или кадрах из пакетов 1-RTT. сервер **может** считать применения обновлённых параметров транспорта в 0-RTT ошибкой соединения типа PROTOCOL_VIOLATION.

7.4.2. Новые транспортные параметры

Для согласования нового поведения протокола могут применяться новые параметры транспорта. Конечная точка **должна** игнорировать неподдерживаемые транспортные параметры. Отсутствие параметра отключает необязательную функцию протокола, согласуемую этим параметром. Как указано в параграфе 18.1, некоторые идентификаторы зарезервированы для исполнения этого требования.

Клиент, не понимающий транспортный параметр, может отбросить его и попытаться примерить 0-RTT в последующих соединениях. Однако при добавлении на клиенте поддержки отброшенного транспортного параметра возникает риск того, что задаваемые параметром ограничения будут нарушены при попытке использовать 0-RTT. Новые параметры транспорта позволяют избежать таких проблем за счёт установки по умолчанию наиболее консервативного значения. Клиенты могут избежать проблемы запоминая все параметры, включая неподдерживаемые.

Новые параметры транспорта могут регистрироваться в соответствии с правилами параграфа 22.3.

7.5. Буферизация криптографических сообщений

Реализациям нужно поддерживать буфер данных CRYPTO, принятых с нарушением порядка. Поскольку для кадров CRYPTO нет управления потоком данных, конечная точка может вынудить партнёра буферизовать неограниченный объем данных.

Реализация **должна** обеспечивать буфер не менее 4096 байтов для полученных с нарушением порядка кадров CRYPTO. Конечные точки **могут** согласовать выделение больших буферов. Увеличение предела в процессе согласования позволяет обмениваться более длинными ключами и свидетельствами. В течение срока действия соединения конечная точка не обязана сохранять размер буфера.

Неспособность буферизовать кадры CRYPTO в процессе согласования может приводить к отказу соединения. Если буфер конечной точки заполняется в процессе согласования, его можно временно расширить для завершения согласования. Если конечная точка не расширяет буфер, она **должна** закрыть соединение с кодом ошибки CRYPTO_BUFFER_EXCEEDED.

Если по завершении согласования конечная точка не способна буферизовать все данные кадра CRYPTO, она **может** отбросить кадр CRYPTO и все будущие кадры CRYPTO или **может** закрыть соединение с кодом ошибки CRYPTO_BUFFER_EXCEEDED. Пакеты, содержащие отброшенные кадры CRYPTO, **должны** подтверждаться, поскольку они были получены и обработаны транспортом, несмотря на отбрасывание кадров CRYPTO.

8. Проверка адреса

Проверка адреса предотвращает возможность использования конечной точки для атак с усилением трафика, где пакеты передаются серверу с фиктивным адресом отправителя, указывающим жертву. Если сервер возвращает

большее число пакетов или они более крупные, атакующий может воспользоваться сервером для увеличения объёма данных, передаваемых жертве.

Основной защитой от атак с усилением трафика является проверка способности партнёра принимать пакеты по заявленному им транспортному адресу. Поэтому после приёма пакета с непроверенного адреса конечная точка должна ограничить объем передаваемых по непроверенному адресу данных до троекратного размера полученных с этого адреса данных. Этот предел размера откликов называют «антиусилительным» (anti-amplification).

Проверка адреса выполняется при организации (параграф 8.1) и переносе (параграф 8.2) соединения.

8.1. Проверка адреса при организации соединения

При организации соединения неявно выполняется проверка адресов обеих конечных точек. В частности, получение пакета, защищённого ключами Handshake, подтверждает, что партнёр успешно обработал пакет Initial. После успешной обработки пакета Handshake от партнёра конечная точка может считать его адрес проверенным. Кроме того, конечная точка **может** считать адрес партнёра проверенным, если тот использует идентификатор соединения, выбранный конечной точкой и содержащий не менее 64 битов энтропии.

Значение поля Destination Connection ID в его первом пакете Initial позволяет клиенту проверить адрес сервера в процессе обработки пакета. Пакеты Initial от сервера защищены с использованием ключей, выведенных из этого значения (см. параграф 5.2 в [QUIC-TLS]). Кроме того, значение возвращается сервером в пакетах Version Negotiation (раздел 6) или включается в тег Integrity пакетов Retry (параграф 5.8 в [QUIC-TLS]).

До проверки адреса клиента серверу **недопустимо** передавать тому данные, объем которых более чем втрое превышает объем полученных данных. Это ограничивает возможности атак с усилением, которые могут быть организованы с использованием фиктивных адресов отправителя. Для предотвращения атак с усилением трафика до проверки адреса сервер **должен** учитывать все байты данных в полученных дейтаграммах, которые однозначно связаны с одним соединением. Учитываются дейтаграммы с успешно обработанными и отброшенными пакетами.

Клиент **должен** обеспечить в дейтаграмме UDP с пакетом Initial не менее 1200 байтов данных UDP (payload), добавляя при необходимости кадры PADDING. Отправка клиентом дополненных дейтаграмм позволяет серверу передать больше данных до завершения проверки адреса.

Потеря пакета Initial или Handshake от сервера может вызвать блокировку, если клиент не передаст дополнительных пакетов Initial или Handshake. Блокировка может возникнуть при достижении на сервере порода «антиусиления» и получении клиентом подтверждения для всех отправленных данных. Если в таком случае у клиента нет причин продолжать отправку дополнительных пакетов, сервер не сможет передать больше данных, поскольку адрес клиента ещё не проверен. Для предотвращения блокировки клиент **должен** передать пакет по тайм-ауту PTO (Probe Timeout, параграф 6.2 в [QUIC-RECOVERY]). В частности, клиент **должен** передать пакет Initial в дейтаграмме UDP, содержащей не менее 1200 байтов, если у него нет ключей Handshake, или передать пакет Handshake.

Сервер может захотеть проверить адрес клиента до начала криптографического согласования. QUIC использует маркер в пакете Initial для проверки адреса до завершения согласования. Этот маркер доставляется клиенту при организации согласования с пакетом Retry (параграф 8.1.2) или в предыдущем соединении с использованием кадра NEW_TOKEN (параграф 8.1.3).

Помимо ограничений на отправку до проверки адреса серверы также ограничены в возможности управлять ограничениями, заданными контроллером перегрузки. Клиенты ограничены лишь контроллером перегрузки.

8.1.1. Создание маркера

Маркер в кадре NEW_TOKEN или пакете Retry **должен** создаваться так, чтобы сервер мог определить способ его предоставления клиенту. Маркер в обоих случаях передаётся в одном поле, но обработка на сервере различается.

8.1.2. Проверка адреса с помощью пакетов Retry

После приёма пакета Initial от клиента сервер запрашивает проверку адреса, передавая пакет Retry (параграф 17.2.5) с маркером. Маркер **должен** повторяться клиентом во всех пакетах Initial, передаваемых для соединения после приёма пакета Retry. По результату обработки пакета Initial с маркером из пакета Retry сервер не может передать другой пакет Retry, а может лишь отвергнуть соединение или продолжить обработку. Пока атакующий не способен создать действительный маркер для своего адреса (см. параграф 8.1.4), а клиент может вернуть маркер, это подтверждает серверу получение маркера клиентом.

Сервер может также использовать пакет Retry, чтобы отложить на время состояние и расходы на обработку организации соединения. Требование к серверу предоставить другой идентификатор соединения вместе с исходным параметром транспорта original_destination_connection_id transport, определенным в параграфе 18.2, заставляет сервер продемонстрировать, что он или взаимодействующий с ним объект получил от клиента исходный пакет Initial. Представление другого идентификатора соединения также даёт серверу некоторый контроль над маршрутизацией последующих пакетов, что можно использовать для отправки соединений другому экземпляру сервера.

При получении сервером от клиента корректного пакета Initial с недействительным маркером Retry, он знает, что клиент не воспримет другой маркер Retry. Сервер может отбросить такой пакет и разрешить клиенту тайм-аут для обнаружения отказа при согласовании, но это может внести для клиента существенную задержку. Вместо этого серверу **следует** незамедлительно закрыть соединение (параграф 10.2) с ошибкой INVALID_TOKEN. Сервер на этом этапе не установил состояния для соединения и не задал время закрытия. Поток с использованием пакета Retry показан на рисунке 9.

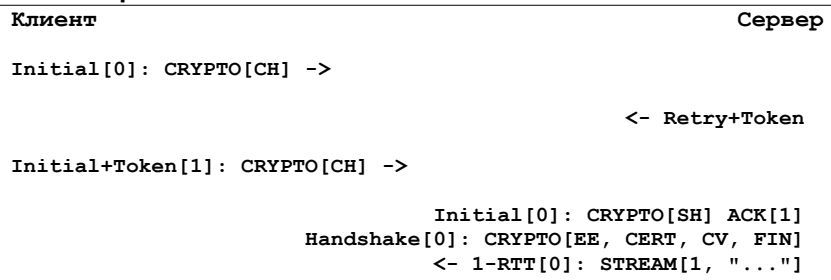

Рисунок 9. Пример согласования с пакетом Retry.

8.1.3. Проверка адреса для будущих соединений

Сервер **может** предоставить клиенту маркер проверки адреса для использования в последующем соединении. Проверка адреса особо важна для 0-RTT, поскольку сервер может передавать клиенту значительный объем данных в ответ на данные 0-RTT. Сервер применяет кадр NEW_TOKEN (параграф 19.7) для предоставления клиенту маркера, служащего для проверки адреса в будущих соединениях. В таком соединении клиент включает маркер в передаваемые пакеты Initial, пока пакет Retry не заменит маркер новым. Клиенту **недопустимо** применять маркер из пакета Retry для будущих соединений. Серверы **могут** отбрасывать пакеты Initial, не включающие ожидаемый маркер.

В отличие от сразу применяемого маркера из пакета Retry, маркер в кадре NEW_TOKEN можно использовать в течение некоторого времени после его передачи. Поэтому маркеру **следует** иметь срок действия, который может быть явно указан временем окончания или временной меткой, служащей для динамического расчёта времени окончания срока действия. Сервер может сохранять время завершения срока действия или включать его в зашифрованный маркер. В маркер кадра NEW_TOKEN **недопустимо** включать сведения, которые позволят стороннему наблюдателю связать значение с соединением, где оно было передано. Например, нельзя включать идентификатор предыдущего соединения или адресные данные, если эти значения не зашифрованы. Сервер **должен** обеспечить для каждого нового кадра NEW_TOKEN уникальность среди всех клиентов, за исключением случаев передачи с целью восстановления при потере предшествующих кадров NEW_TOKEN. Информация, позволяющая серверу различать маркеры из Retry и NEW_TOKEN. **может** быть доступна не только серверу.

Вероятность совпадения номера клиентского порта для двух разных соединений мала, поэтому проверка порта вряд ли будет эффективна.

Маркер из кадра NEW_TOKEN применим к любому серверу, для которого соединение считается полномочным (например, для серверов, имена которых указаны в сертификате). При соединении с сервером, для которого у клиента имеется применимый и не использованный маркер, **следует** включать этот маркер в поле Token пакета Initial. Включение маркера может позволить серверу проверить адрес клиента без дополнительного кругового обхода. Клиенту **недопустимо** включать маркер, не применимый для сервера, с которым соединяется клиент, пока у него нет уверенности, что передавший маркер сервер и сервер, к которому клиент подключается, управляют маркерами совместно. Клиент **может** использовать маркер из любого прежнего соединения с сервером.

Маркер позволяет серверу сопоставить соединение, где маркер был выпущен, с соединением, где тот используется. Клиенты, желающие прервать своё отождествление на сервере, могут отбросить маркеры из кадров NEW_TOKEN. Для сравнения, маркер из пакета Retry **должен** применяться сразу же в соединении, где он получен, и не может использоваться в последующих попытках соединения.

Клиенту **не следует** повторно использовать маркер из кадра NEW_TOKEN для другой попытки соединения. Повторное использование позволяет сопоставить соединения находящимся на пути через сеть элементам (см. параграф 9.5). Клиенты могут получать несколько маркеров за одно соединение. Помимо предотвращения возможности сопоставления, любой маркер можно использовать в любой попытке соединения. Серверы могут передавать дополнительные маркеры, чтобы разрешить проверку адреса для нескольких попыток соединения или заменить устаревшие маркеры, которые могли стать недействительными. Для клиента эта неоднозначность показывает, что отправка наиболее свежего не использованного маркера будет скорей всего результативна. Хотя сохранение и использование старых маркеров не ведёт к негативным последствиям, клиенты могут считать старые маркеры менее подходящими серверу для проверки адресов.

При получении сервером пакета Initial с маркером проверки адреса он **должен** попытаться проверить маркер, если проверка адреса ещё не завершена. Если маркер недействителен, серверу **следует** действовать как при отсутствии для клиента проверенного адреса, включая возможную отправку пакета Retry. Маркеры из кадра NEW_TOKEN и пакета Retry серверы могут различать (параграф 8.1.1) и последние могут проверяться более строго. При успешной проверке серверу **следует** разрешать выполнение согласования.

> Примечание. Основанием считать клиента неподтвержденным вместо отбрасывания его пакетов служит то, что клиент мог получить маркер в кадре NEW_TOKEN предыдущего соединения и при утрате сервером состояния он не может проверить маркер, что приведёт к отказу в соединении, если пакет будет отброшен.

При работе без сохранения состояний сервер может использовать зашифрованные и аутентифицированные маркеры для передачи клиентам информации, которую он позднее может восстановить и применить для проверки адреса клиента. Маркеры не встроены в криптографическое согласование, поэтому они не аутентифицируются. Например, клиент может использовать маркер неоднократно. Для предотвращения атак, использующих эту особенность. Сервер может разрешать использование в маркерах лишь, сведений, применяемых для проверки адреса клиента.

Клиенты **могут** использовать полученные в соединении маркеры при другой попытке соединения с той же версией. При выборе маркера для применения клиент не обязан рассматривать другие свойства соединения, включая выбор возможных прикладных протоколов, сеансовые квитанции и пр.

8.1.4. Целостность маркера проверки адреса

Маркер проверки адреса **должен** быть трудно предсказуемым. Включения в маркер случайного значения, содержащего не менее 128 битов энтропии, будет достаточно, но это зависит от того, помнит ли сервер значения, переданные клиентам.

Схема с маркером позволяет серверу выгрузить (offload) состояние, связанное с проверкой клиента. Чтобы это работало, маркер **должен** учитываться системой защиты целостности от изменения или поддержки клиентами. Без защиты целостности вредоносные клиенты могут создать или угадать маркеры, которые будут восприняты сервером. Доступ к ключу защиты целостности маркеров нужен лишь серверу.

Не требуется использовать для маркеров чётко заданный формат, поскольку маркеры воспринимает создавший их сервер. В маркеры, передаваемые в пакетах Retry, **следует** включать сведения, которые позволят серверу проверить неизменность IP-адреса и порта в пакетах клиента. Маркеры в кадрах NEW_TOKEN **должны** включать сведения, позволяющие серверу убедиться, что IP-адрес клиента не изменился с момента выдачи маркера. Сервер может использовать маркеры из кадров NEW_TOKEN для решения вопроса об отправке пакета Retry даже при смене клиентом адреса. Если IP-адрес клиента поменялся, сервер **должен** соблюдать предел «антиусиления», как указано в разделе 8. Отметим, что в присутствии NAT этого может быть недостаточно для защиты других хостов за NAT от атак с усилением.

Атакующий может воспроизвести маркеры для использования сервера в качестве усилителя DDoS-атак. Для защиты от таких атак серверы **должны** гарантировать предотвращение или ограничение повторного использования маркеров. Серверам **следует** воспринимать маркеры из пакетов Retry лишь в течение короткого времени, поскольку клиенты возвращают их незамедлительно. Маркеры из кадров NEW_TOKEN (параграф 19.7) должны действовать дольше, но **не следует** воспринимать их несколько раз. Серверам рекомендуется разрешать лишь однократное применение маркера, если это возможно. Маркеры **могут** включать другие сведения, дополнительно ограничивающие применимость и возможность повторного использования.

8.2. Проверка пути

Проверка пути используется обоими партнёрами при переносе соединения (см. раздел 9) для проверки доступности после смены адреса. Партнёры проверяют доступность пути между конкретным локальным адресом и портом и заданным адресом партнёра и его портом. При проверке пути тестируется получение партнёром. отправленных ему пакетов. Проверка служит для обеспечения гарантии того, что пакет, полученный от сменившего адрес партнёра, не содержит подставной адрес.

Проверка пути не подтверждает возможность передачи партнёром. пакетов в обратном направлении. Подтверждения не могут служить проверкой обратного пути, поскольку они не содержат достаточной энтропии и могут быть подделаны. Конечные точки независимо проверяют доступность пути в каждом направлении и доступность обратного пути может быть указана лишь партнёром. Проверку пути может использовать любая конечная точка в любой момент. Например, конечная точка может проверить, что партнёр по-прежнему владеет адресом после периода бездействия.

Проверка пути не предназначена для работы через NAT. Хотя описанный здесь механизм может быть эффективным для создания привязок NAT, поддерживающих прохождение через NAT, предполагается, что одна конечная точка способна принимать пакеты без предварительной отправки пакета по этому пути. Для эффективной работы через NAT нужны дополнительные механизмы синхронизации, не представленные здесь.

Конечная точка может дополнять другими кадрами используемые для проверки пути кадры PATH_CHALLENGE и PATH_RESPONSE. В частности, конечная точка может включать кадры PADDING вместе с кадром PATH_CHALLENGE для PMTUD[1] (см. параграф 14.2.1), а также отправлять свой кадр PATH_CHALLENGE при передаче PATH_RESPONSE.

Конечная точка использует новый идентификатор соединения для отправки пробных пакетов с нового локального адреса (см. параграф 9.5). Про зондировании нового пути конечная точка может гарантировать своему партнёру наличие неиспользованного идентификатора соединения для отправки откликов. Передача NEW_CONNECTION_ID и PATH_CHALLENGE в одном пакете (если позволяет параметр партнёра active_connection_id_limit) обеспечивает доступность неиспользованного идентификатора соединения для передачи отклика.

Конечная точка может проверять одновременно несколько путей, число которых ограничено числом дополнительных идентификаторов соединения, поскольку для проверки каждого нового локального адреса нужен не использовавшийся ранее идентификатор соединения.

8.2.1. Инициирование проверки пути

Для начала проверки пути конечная точка передаёт кадр PATH_CHALLENGE с непредсказуемыми данными по проверяемому пути. Конечная точка **может** передать множество кадров PATH_CHALLENGE для защиты от потери пакетов, однако **не следует** передавать несколько кадров PATH_CHALLENGE в одном пакете. Конечной точке **не следует** проверять новый путь с пакетами, содержащими PATH_CHALLENGE, более часто, чем передаются пакеты Initial. Это гарантирует, что при переносе соединения нагрузка на новом пути не будет выше нагрузки при организации соединения. Конечная точка **должна** использовать непредсказуемые данные в каждом кадре PATH_CHALLENGE, чтобы связать отклик партнёра с соответствующим PATH_CHALLENGE.

Конечная точка **должна** обеспечивать дейтаграммам с кадром PATH_CHALLENGE размер не меньше минимально разрешённого максимума в 1200 байтов, если предел антиусиления не запрещает это. Отправка дейтаграмм UDP такого размера гарантирует, что путь от конечной точки до партнёра подходит для QUIC (см. раздел 14). Если конечная точка не может расширить дейтаграмму до 1200 байтов из-за ограничений антиусиления, значение MTU на пути не будет проверено. Чтобы гарантировать достаточный размер MTU на пути, конечная точка **должна** выполнить втору проверку пути, отправляя кадр PATH_CHALLENGE в дейтаграмме размером не менее 1200 байтов. Эта дополнительная проверка может быть выполнена после приёма кадра PATH_RESPONSE или получения по данному пути достаточного числа байтов, чтобы можно было передать большую дейтаграмму без ограничений антиусиления.

[1]Path Maximum Transmission Unit Discovery - определение MTU для пути.

В отличие от других случаев расширения размера дейтаграмм, конечным точкам **недопустимо** отбрасывать дейтаграммы, представляющиеся слишком мелкими, если они содержат PATH_CHALLENGE или PATH_RESPONSE.

8.2.2. Отклики при проверке пути

При получении кадра PATH_CHALLENGE конечная точка **должна** ответить кадром PATH_CHALLENGE с копией данных из PATH_RESPONSE. Конечной точке **недопустимо** задерживать отправку кадров PATH_RESPONSE, если ей не препятствует механизм контроля перегрузок. Кадр PATH_RESPONSE **должен** передаваться в путь, по которому был получен кадр PATH_CHALLENGE. Это обеспечивает успешную проверку пути партнёром. лишь при работе пути в обоих направлениях. Инициатору проверки пути **недопустимо** применять это требование, поскольку это открывает возможность атак на перенос соединений (см. параграф 9.3.3).

Конечная точка **должна** увеличивать размер дейтаграмм с кадром PATH_RESPONSE по меньшей мере до наименьшего разрешённого максимума в 1200 байтов для подтверждения поддержки дейтаграмм такого размера в обоих направлениях. Однако конечной точке **недопустимо** расширять дейтаграммы с PATH_RESPONSE сверх предела антиусиления. Предполагается, что это может происходить лишь при получении PATH_CHALLENGE без расширения дейтаграммы.

Конечной точке **недопустимо** передавать более одного кадра PATH_RESPONSE в ответ на кадр PATH_CHALLENGE (см. параграф 13.3). Предполагается, что при необходимости партнёр передаст дополнительные кадры PATH_CHALLENGE, чтобы получить добавочные отклики PATH_RESPONSE.

8.2.3. Успешная проверка пути

Проверка пути успешна при получении PATH_RESPONSE с данными из соответствующего кадра PATH_CHALLENGE. Кадр PATH_RESPONSE, принятый по любому пути, подтверждает путь, по которому отправлен PATH_CHALLENGE.

Если конечная точка передаёт кадр PATH_CHALLENGE в дейтаграмме размером не более 1200 байтов и отклик на неё подтверждает адрес партнёра, путь считается проверенным, но без подтверждения MTU для него. В результате конечная точка может передать данные размером более троекратного размера принятых данных. Однако конечная точка **должна** инициировать другую проверку пути с расширенными дейтаграммами для проверки значения MTU.

Получение подтверждения для пакета с кадром PATH_CHALLENGE не обеспечивает проверки пути, поскольку подтверждение можно подделать.

8.2.4. Отказ при проверке пути

Проверка пути считается неудачной лишь в том случае, когда проверяющая путь конечная точка отказывается от дальнейших попыток проверить путь. Конечным точкам **следует** прекращать проверку пути по таймеру. При установке таймера реализации следует понимать, что новый путь может иметь больший период кругового обхода. **Рекомендуется** устанавливать значение в 3 раза больше текущего PTO или PTO для нового пути (с использованием kInitialRtt, как определено в [QUIC-RECOVERY]). Такой тайм-аут позволяет проверять путь в течение нескольких PTO, чтобы потеря одного кадра PATH_CHALLENGE или PATH_RESPONSE не приводила к отказу.

Отметим, что проверяющая система может получать по новому пути другие кадры, но для успешной проверки нужен кадр PATH_RESPONSE с соответствующими данными.

Когда конечная точка отказывается от проверки пути, она считает его непригодным для использования. Это не обязательно вызывает отказ в соединении и конечные точки могут продолжать передачу пакетов по другим путям. Если доступных путей нет, конечная точка может подождать появления нового пути или закрыть соединение. Конечная точка, не имеющая доступного пути к партнёру, **может** сообщить об этом, используя ошибку соединения NO_VIABLE_PATH и указав при этом, что это возможно лишь при наличии пути без поддержки требуемого MTU (параграф 14).

Проверка пути может быть прекращена по иным причинам, прежде всего, в результате перехода на новый путь в процессе выполняемой проверки пути.

9. Перенос соединения

Использование идентификаторов соединения обеспечивает устойчивость соединений к смене адресов конечных точек (IP-адрес и порт), например при переходе конечной точки в другую сеть. В этом разделе описан процесс перехода конечной точки на новый адрес. Работа QUIC основана на сохранении конечными точками стабильных адресов в процессе согласования (handshake). Конечной точке **недопустимо** инициировать перенос до завершения согласования, как указано в параграфе 4.1.2 [QUIC-TLS].

Если партнёр передал транспортный параметр disable_active_migration, конечной точке **недопустимо** передавать пакеты (включая пакеты зондирования, см. параграф 9.1) с другого локального адреса на использованный при согласовании адрес партнёра, коль скоро конечная точка не действует в соответствии с транспортным параметром preferred_address от партнёра. Если партнёр нарушает это требование, конечная точка **должна** отбрасывать входящие пакеты на этом пути без генерации Stateless Reset или выполнить проверку пути и позволить партнёру переместиться. Генерация Stateless Reset или закрытие соединения позволили бы третьей стороне вызвать закрытие соединения путём использования обманных адресов или иных манипуляций с наблюдаемым трафиком.

Не каждая смена адреса партнёром. связана с преднамеренным (активным) перемещением. Это может быть связано с перестройкой NAT - сменой адреса промежуточным устройством (обычно NAT), выделением нового выходного порта или даже адреса IP для потока. Конечная точка **должна** проверить путь (параграф 8.2) при обнаружении смены партнёром. адреса, если этот адрес не был проверен ранее.

Когда у конечной точки нет проверенного пути для отправки пакетов, она **может** сбросить соединение. Конечная точка, способная переносить соединение, **может** выждать появления доступного пути без сброса соединения.

Этот документ ограничивает перенос соединений сменой адреса клиента, а также случаем, описанным в параграфе 9.6. Предпочтительный адрес сервера. За инициирование переноса отвечают клиенты. Серверы не передают пакетов,

не связанных с зондированием (см. параграф 9.1) по адресу клиента, пока они не получат с этого адреса пакетов, не являющихся зондами. Пакеты, полученные клиентом с неизвестного адреса сервера **должны** отбрасываться.

9.1. Зондирование нового пути

Конечная точка **может** проверить доступность партнёра с нового локального адреса, используя проверку пути (параграф 8.2) до переноса соединения на этот адрес. Отказ при проверке пути просто означает непригодность пути для этого соединения и не ведёт к разрыву соединения, если имеются другие пути.

Кадры PATH_CHALLENGE, PATH_RESPONSE, NEW_CONNECTION_ID и PADDING считаются «кадрами зондирования», а пакет, содержащий лишь такие кадры - «пакетом зондирования».

9.2. Инициирование переноса соединения

Конечная точка может перенести соединение на новый локальный адрес, передавая с этого адреса пакеты, не относящиеся к зондированию.

Каждая конечная точка проверяет адрес партнёра в процессе организации соединения. Поэтому перемещающаяся конечная точка может передать пакеты своему партнёру, зная, что он готов получать данные по своему текущему адресу. В результате конечная точка может перейти на новый локальный адрес без предшествующей проверки пути к адресу партнёра.

Чтобы убедиться в доступности нового пути, конечная точка организует его проверку (параграф 8.2). Проверку **можно** отложить до момента, когда партнёр отправит на новый адрес кадр, не относящийся к зондированию.

При переносе новый путь может не поддерживать текущую скорость передачи конечной точки, поэтому конечная точка сбрасывает свой контроллер перегрузок и оценку RTT, как описано в параграфе 9.4. На новом пути возможности ECN могут отличаться, поэтому конечная точка проверяет их, как описано в параграфе 13.4.

9.3. Реакция на перенос соединения

Получение с нового адреса партнёра не относящегося к зондированию пакета указывает переход партнёра на новый адрес. Если получатель разрешает переход, он **должен** передавать последующие пакеты по новому адресу, а также **должен** начать проверку пути (параграф 8.2), чтобы убедиться во владении партнёра адресом, если проверка ещё не проведена. Если у получателя нет неиспользуемого идентификатора соединения от партнёра, он не сможет ничего передать по новому пути (см. параграф 9.5).

Конечная точка меняет адрес, по которому она отправляет пакеты, лишь в ответ на пакет с большим номером, не относящийся к зондированию. Это предотвращает отправку конечной точкой пакетов по старому адресу партнёра в случае приёма разупорядоченных пакетов.

Конечная точка **может** передавать данные по непроверенному адресу партнёра, но она **должна** защищаться от атак , описанных в параграфах 9.3.1 и 9.3.2. Конечная точка **может** пропустить проверку адреса партнёра, если она видела этот адрес раньше. В частности, при возврате конечной точки на проверенный ранее путь после обнаружения того или иного ложного переноса пропуск проверки адреса и восстановление статуса обнаружения потерь могут снижать влияние атаки на производительность.

После смены адреса, по которому передаются не являющиеся зондами пакеты, конечная точка может отказаться от проверки пути для других адресов.

Получение пакета с новым адресом партнёра может быть результатом перестройки NAT на стороне партнёра.

После проверки нового адреса клиента серверу **следует** передать новые маркеры проверки адреса (параграф 8).

9.3.1. Подмена адреса партнёра.

Возможна подделка партнёром. своего адреса отправителя, чтобы заставить конечную точку передать чрезмерный объем данных не желающему этого хосту. Если конечная точка передаёт существенно больше данных, нежели обманный партнёр, это может служить для увеличения объёма данных, которые атакующий может отправить жертве.

Как описано в параграфе 9.3, конечной точке требуется проверять новый адрес партнёра для подтверждения владения этим адресом. Пока адрес партнёра не признан действительным, конечная точка ограничивает объем передаваемых по этому адресу данных (см. раздел 8). При отсутствии такого ограничения возникает риск использования конечной точки для атаки на отказ в обслуживании ничего не подозревающей жертвы.

Если конечная точка не проверяет адрес партнёра, как описано выше, ей не нужно ограничивать скорость отправки.

9.3.2. Подмена адреса на пути

Атакующий может вызвать ложный перенос соединения, копируя и пересылая пакеты с фиктивным адресом так, чтобы они приходили раньше исходных пакетов. Пакеты с фиктивными адресами будут казаться пакетами с нового адреса, а исходные пакеты - дубликатами (отбрасываются). После ложного переноса проверка адреса отправителя может привести к отказу, поскольку у объекта с подставным адресом нет криптографических ключей для чтения кадра PATH_CHALLENGE и отклика на него.

Для защиты соединения от таких ложных переносов конечная точка **должна** вернуться к использованию проверенного адреса партнёра. в случае отказа при проверке нового адреса. Кроме того, получение пакетов с большими номерами с легитимного адреса партнёра вызовет ещё один перенос соединения. В результате проверка адреса для ложного переноса будет отменена, ограничивая ложный перенос вставкой одного пакета.

Если у конечной точки нет статуса последнего проверенного адреса партнёра, она **должна** закрыть соединение без уведомления партнёра, просто сбросив его состояние. Это ведёт к обычной обработке пакетов в новом соединении. Например, конечная точка **может** передавать a Stateless Reset в ответ на последующие входящие пакеты.

9.3.3. Подмена адреса вне пути

Находящийся вне пути злоумышленник, способный наблюдать пакеты, может пересылать конечным точкам копии настоящих пакетов. Если копия приходит раньше оригинала это будет выглядеть результатом перестройки NAT и исходные пакеты будут отброшены как дубликаты. Если атакующий способен продолжать пересылку пакетов, он может вынудить к переходу на новый путь через себя. Это позволит ему видеть или отбрасывать все последующие пакеты.

Этот тип атак основан на наличии у атакующего пути с характеристиками, похожими на характеристики прямого пути между конечными точками. Атака становится более эффективной, если передано небольшое число пакетов или атака совпадает с потерей пакетов.

Не связанные с зондированием пакеты, полученные по исходному пути с ростом порядковых номеров в принятых пакетах заставят конечную точку вернуться на этот путь. Выявление таких пакетов повышает вероятность обнаружения атаки и её неудачи. Поэтому смягчение таких атак зависит от инициирования обмена пакетами.

В ответ на очевидный перенос соединения конечные точки **должны** проверить ранее активный путь с помощью кадра PATH_CHALLENGE. Это включает отправку по данному пути новых пакетов. Если путь больше не действует, попытка проверки приведёт к тайм-ауту и отказу. Если же путь действует, но перестал быть желательным, проверка завершится успехом, но приведёт лишь к отправке пакетов зондирования.

Конечной точке, принявшей кадр PATH_CHALLENGE по активному пути, **следует** передать в ответ не являющийся зондом пакет. Последующий перенос пути запустит процесс снова.

Такая защита несовершенна, но не создаёт серьёзных проблем. Если путь через атакующего действительно быстрее исходного пути, атаку невозможно отличить от изменения маршрутизации, несмотря на попытки использовать исходный путь.

Конечная точка может использовать эвристические методы для более надёжного обнаружения этого типа атак. Например, смену привязки NAT невозможно проверить, если пакеты были недавно приняты по старому пути. Кроме того, смена привязки редко возникает на путях IPv6. Конечные точки могут также искать дубликаты пакетов. Изменение идентификатора соединения более чётко указывает преднамеренный перенос соединения, а не атаку.

9.4. Обнаружение потерь и контроль перегрузок

Пропускная способность нового пути может отличаться от прежней. Отправленные по прежнему пути пакеты **недопустимо** учитывать при контроле перегрузки или расчёте RTT для нового пути.

После подтверждения владения партнёром. новым адресом конечная точка **должна** немедленно сбросить контроллер перегрузки и оценку времени кругового обхода для нового пути (см. Приложения A.3 и B.3 в [QUIC-RECOVERY]), если единственным изменением не является номер порта партнёра. Поскольку смена лишь порта обычно является результатом перестройки NAT или действий промежуточного устройства, конечная точка **может** сохранить статус контроля перегрузки и оценку RTT для таких случаев. Если состояние контроля перегрузки старого пути применяется на новом пути с существенно иными характеристиками, отправитель может передать много данных, пока контроль перегрузок и оценка RTT будут скорректированы. Обычно разработчикам рекомендуется с осторожностью применять прежние значения для нового пути.

В процессе переноса соединения, когда конечная точка может отправлять данные и зонды по разным адресам или с разных адресов, у получателя может нарушаться порядок доставки пакетов из-за разного времени кругового обхода на путях. Получатель пакетов по нескольким путям будет по-прежнему отправлять кадры ACK, подтверждающие все принятые пакеты. Хотя в процессе переноса может использоваться несколько путей, может быть достаточно одного контекста контроля перегрузок и одного контекста восстановления при потерях, как указано в [QUIC-RECOVERY]. Например, конечная точка может задержать переключение на новый контекст контроля перегрузок, пока не убедиться в ненужности прежнего пути (как в случае, описанном в параграфе 9.3.3).

Отправитель может делать исключения для пакетов зондирования, чтобы обнаружение их потери было независимым и не вызывало чрезмерного снижения скорости контроллером перегрузки. Конечная точка может установить при отправке PATH_CHALLENGE отдельный таймер, который останавливается при получении соответствующего кадра PATH_RESPONSE. Если отсчёт таймера завершается до получения PATH_RESPONSE, конечная точка может передать новый кадр PATH_CHALLENGE и запустить таймер на большее время. Этот таймер **следует** устанавливать в соответствии с параграфом 6.2.1 в [QUIC-RECOVERY] и **недопустимо** делать более настойчивым.

9.5. Влияние переноса соединения на приватность

Использование стабильного идентификатора соединения на нескольких путях позволит пассивному наблюдателю сопоставить активность на этих путях. Перемещающаяся между путями конечная точка может не желать сопоставления своей активности сторонними объектами, поэтому следует использовать разные идентификаторы соединений при передаче с разных локальных адресов, как описано в параграфе 5.1. Для того, чтобы это было эффективным, конечным точкам нужно предотвратить возможность других связать идентификаторы между собой.

Конечные точки **могут** в любой момент сменить передаваемое значение Destination Connection ID на не применявшийся ранее на другом пути идентификатор.

Конечной точке **недопустимо** использовать один идентификатор соединения для отправки с нескольких локальных адресов, например при инициировании переноса соединения (параграф 9.2) или зондировании пути (параграф 9.1). Точно так же конечной точке **недопустимо** использовать один идентификатор соединения для отправки по нескольким адресам. В результате изменений, не контролируемых партнёром., конечная точка может получить пакеты с нового адреса отправителя, но с тем же Destination Connection ID и в этом случае она **может** продолжать использование идентификатора соединения с новым адресом, если отправка происходит с того же локального адреса.

Требования в части повторного использования идентификаторов соединения относятся лишь к отправке пакетов, поскольку возможны непреднамеренные изменения пути без смены идентификатора соединения. Например, после бездействия перестройка NAT может приводить к передаче пакетов по новому пути после восстановления передачи клиентом. Реакция конечной точки на такие пакеты описана в параграфе 9.3.

Использование разных идентификаторов соединения для пакетов обоих направлений на каждом новом пути предотвращает возможность связать по идентификаторам пакеты одного соединения, передаваемые разными путями. Защита заголовков предотвращает сопоставление по номерам пакетов. Однако это не мешает применять для сопоставления другие свойства пакетов, такие как размер и время.

Конечной точке **не следует** инициировать переход для партнёра, запросившего пустой идентификатор соединения, поскольку трафик по новому пути тривиально сопоставляется с трафиком по прежнему пути. Способность сервера привязать пакеты с пустым идентификатором соединения к нужному соединению означает использование сервером других сведений для демультиплексирования пакетов. Например, сервер может предоставить каждому клиенту уникальный адрес, используя дополнительные службы HTTP [ALTSVC]. Сведения,, позволяющие корректно маршрутизировать пакеты по нескольким путям, позволяют также стороннему наблюдателю сопоставить активность на этих путях.

Клиент может снизить вероятность сопоставления путём смены идентификатора соединения, порта UDP у отправителя, адреса IP (см. [RFC8981]) при передача трафика после временного бездействия. Смена адреса, с которого передаются пакеты, может приводить к тому, что сервер сделает вывод о переносе соединения. Это ведёт к применению механизмов поддержки переноса соединения даже для клиентов, с которыми не связана перестройка NAT или фактический перенос. Смена адреса может заставить партнёра сбросить состояние контроля перегрузки (параграф 9.4), поэтому адреса **следует** менять нечасто.

Конечная точка, исчерпавшая запас идентификаторов соединений, не может проверить новые пути или инициировать перенос, а также ответить на зонды или попытку партнёра перенести соединение. Для обеспечения возможности переноса и предотвращения сопоставления переданных по разным путям пакетов конечным точкам **следует** предоставлять новые идентификаторы соединения до переноса партнёра (см. параграф 5.1.1). Если партнёр мог исчерпать доступные идентификаторы соединения, переносимой конечной точке следует включать кадр NEW_CONNECTION_ID во все пакеты, передаваемые по новому пути.

9.6. Предпочтительный адрес сервера

QUIC позволяет серверу воспринять соединение по одному адресу IP и попытаться перенести его после согласования на более предпочтительный адрес. Это особенно полезно в тех случаях, когда клиенты изначально подключаются к адресу, совместно используемому несколькими серверами, но предпочитают для обеспечения стабильности использовать свой индивидуальный адрес. В этом параграфе описан перенос соединения на предпочтительный адрес.

Перенос соединения на новый адрес сервера в действующем соединении не поддерживается описанной в документе версией QUIC. Если клиент получает пакеты с нового адреса сервера, когда он не запускал переход на этот адрес, такие пакеты **следует** отбрасывать.

9.6.1. Передача предпочтительного адреса

Сервер передаёт предпочтительный адрес в параметре preferred_address во время согласования TLS. Сервер **может** сообщать предпочтительный адрес в каждом семействе (IPv4 и IPv6), позволяя клиенту выбрать наиболее подходящий.

По завершении согласования клиенту **следует** выбрать один или два представленных клиентом адреса для инициирования проверки пути (параграф 8.2). Клиент создаёт пакеты, используя не применявшиеся ранее идентификаторы соединения, взятые из транспортного параметра preferred_address или кадра NEW_CONNECTION_ID.

После успешной проверки пути клиенту **следует** передавать все будущие пакеты по новому адресу сервера, используя новый идентификатор соединения, и прекратить отправку по прежнему адресу сервера. Если проверка пути завершилась отказом, клиент **должен** продолжать отправку пакетов по прежнему IP-адресу сервера.

9.6.2. Переход на предпочтительный адрес

Переходящий на предпочтительный адрес клиент **должен** проверить выбранный адрес (см. параграф 21.5.3).

Сервер может получить пакет, направленный на предпочитаемый им адрес IP в любой момент после восприятия соединения. Если этот пакет содержит кадр PATH_CHALLENGE, сервер передаёт пакет с кадром PATH_RESPONSE, как указано в параграфе 8.2. Сервер **должен** передавать не относящиеся к зондированию пакеты со своего исходного адреса, пока он не получит от клиента не связанный с зондированием пакет на своём предпочтительном адресе и не проверит новый путь. Север **должен** проверять путь к клиенту со своего предпочтительного адреса. Это помогает защититься от ложных переносов, инспирированных злоумышленником. По завершении проверки пути сервером и получении не связанного с зондированием пакета с новым наибольшим номером на предпочтительном адресе сервер начинает отправку клиенту не связанных с зондированием пакетов исключительно с предпочтительного адреса IP. Серверу **следует** отбрасывать для этого соединения новые пакеты, направленные по прежнему адресу IP. Сервер **может** продолжать обработку задержанных пакетов, принятых на старом адресе IP.

Адрес, предложенный сервером в параметре preferred_address, действителен лишь для соединения, в котором он был представлен. Клиенту **недопустимо** применять этот адрес для других соединений, восстановленных из текущего.

9.6.3. Перенос клиента и переход на предпочтительный адрес сервера

Клиенту может потребоваться перенести соединение до перехода на предпочтительный адрес сервера. В таком случае клиенту **следует** одновременно выполнить проверку пути для исходного и предпочтительного адреса сервера со своего нового адреса. Если проверка пути к предпочтительному адресу сервера успешна, клиент **должен** отказаться от проверки исходного адреса сервера и использовать предпочтительный. Если же проверка пути к предпочтительному адресу завершилась отказом, а для исходного - успехом, клиент **может** перейти на свой новый адрес и продолжать отправку пакетов по исходному адресу сервера.

Если адрес отправителя в пакетах, принятых на предпочтительном адресе сервера, отличается от представленного клиентом при согласовании, сервер **должен** использовать защиту от возможных атак, как указано в параграфах 9.3.1 и 9.3.2. Кроме преднамеренного переноса клиента это может быть связано с использованием клиентской сетью доступа другой привязки NAT для предпочтительного адреса сервера.

Серверу **следует** начать проверку пути к новому адресу клиента при получении пакета зондирования с другого адреса (см. раздел 8).

Клиенту, переходящему на новый адрес, **следует** использовать предпочтительный адрес сервера из того же семейства.

Идентификатор соединения в транспортном параметре preferred_address не зависит от представленных адресов. Этот идентификатор служит для предоставления клиенту идентификаторов, используемых при переносе, но клиент **может** использовать идентификаторы на любом пути.

9.7. Использование метки потока IPv6 и перенос

Конечным точкам, передающим данные по протоколу IPv6, **следует** применять метки потоков IPv6 в соответствии с [RFC6437], если локальный API не запрещает их установку. Генерация меток потоков **должна** обеспечивать минимизацию возможности выделить ранее использованную метку потока, поскольку это позволит сопоставить активность на разных путях (см. параграф 9.5).

В [RFC6437] предлагается выделять метки потоков с использованием псевдослучайной функции. Включение поля Destination Connection ID в дополнение к адресам отправителя и получателя при генерации меток гарантирует синхронизацию изменений с изменениями других наблюдаемых идентификаторов. Криптографическая функция, комбинирующая эти входные данные с локальным секретом, является одним из вариантов реализации.

10. Завершение соединений

Организованное соединение QUIC может быть прервано одним из трёх способов:

- тайм-аут бездействия (параграф 10.1);
- незамедлительное закрытие(параграф 10.2);
- сброс без учёта состояния (параграф 10.3).

Конечная точка **может** отбросить статус соединения, если у неё нет проверенного пути для передачи пакетов (параграф 8.2. Проверка пути).

10.1. Тайм-аут бездействия

Если конечная точка задала транспортный параметр max_idle_timeout (параграф 18.2), соединение просто закрывается с отбрасыванием состояния при бездействии в течение времени, превышающего значение max_idle_timeout, заданное обеими конечными точками. Значение max_idle_timeout анонсирует каждая конечная точка и действовать будет меньшее из заявленных значений или единственное, если одна из точек не указала параметр. Анонсируя max_idle_timeout, конечная точка принимает на себя обязательство незамедлительно инициировать закрытие соединения (параграф 10.2), если она завершает его раньше действующего тайм-аута.

Конечная точка перезапускает таймер бездействия при получении и успешной обработке пакета от партнёра, а также при отправке пакета запроса подтверждения, если такие пакеты не передавались с момента обработки полученного последним пакета. Перезапуск таймера при отправке пакета пакета обеспечивает сохранение соединения после его активизации.

Для предотвращения слишком короткого времени ожидания конечные точки **должны** устанавливать тайм-аут не менее 3 текущих значений тайм-аута зондирования (Probe Timeout или PTO). Это позволяет отправить несколько зондов (которые могут теряться) до истечения тайм-аута ожидания.

10.1.1. Проверка живучести

Конечная точка, передающая пакеты, незадолго до завершения тайм-аута бездействия рискует, что эти пакеты будут отброшены партнёром., поскольку время ожидания может завершиться до прибытия пакета.

Конечная точка может передать PING или иной кадр запроса подтверждения для проверки жизнеспособности соединения, если партнёр может вскоре достичь тайм-аута, такого как PTO (см. параграф 6.2 в [QUIC-RECOVERY]). Это особенно полезно, если нет возможности безопасно повторить отправку данных приложения. Отметим, что безопасность повтора передачи данных определяет приложение.

10.1.2. Отсрочка тайм-аута бездействия

Конечной точке может потребоваться отправить пакеты с запросом подтверждения, чтобы избежать тайм-аута бездействия, если ей нужны данные отклика, но нет возможности передать данные приложения.

Реализация QUIC может предоставлять приложениям возможность отложить тайм-аут бездействия. Это можно использовать в случаях, когда приложение не хочет потерять статус соединения, но не ожидает обмена данными в течение некоторого времени. Конечная точка может периодически передавать кадр PING (параграф 19.2), заставляющий партнёра сбрасывать таймер ожидания. Отправка пакета с кадром PING также перезапускает таймер ожидания для этой конечной точки, если это первый пакет с запросом подтверждения с момента приёма пакета. Отправка кадра PING заставляет партнёра ответить подтверждением и это также перезапускает таймер ожидания для конечной точки.

Прикладным протоколам, использующим QUIC, **следует** давать указания о возможности отсрочить тайм-аут. Отправка избыточных кадров PING может негативно влиять на производительность.

Тайм-аут возникает при отсутствии приёма или передачи пакетов в течение времени, больше заданного транспортным параметром max_idle_timeout (раздел 10). Однако состояния в промежуточных устройствах могут истекать раньше этого времени. Хотя REQ-5 в [RFC4787] рекомендует время ожидания 2 минуты, опыт показывает, что нужно отправлять пакеты каждые 30 секунд, чтобы большинство промежуточных устройств могло сохранить статус потоков UDP [GATEWAY].

10.2. Незамедлительное закрытие

Конечная точка передаёт кадр CONNECTION_CLOSE (параграф 19.19) для незамедлительного разрыва соединения. Этот кадр немедленно закрывает все потоки и открытые потоки можно считать закрываемыми неявно. После отправки кадра CONNECTION_CLOSE конечная точка незамедлительно переходит в состояние закрытия (параграф 10.2.1), а получившая такой кадр точка - в состояние сброса (draining, см. параграф 10.2.2).

Нарушения протокола ведут к незамедлительному закрытию.

Немедленное закрытие может использоваться после того, как протокол приложения организовал закрытие соединения. Это может быть после согласования прикладным протоколом аккуратного отключения (graceful shutdown). Прикладной протокол может организовать обмен сообщениями, которые нужны обеим конечным точкам для решения вопроса о закрытии соединения, после чего приложение запрашивает закрытие соединения QUIC. При закрытии соединения протоколом QUIC используется кадр CONNECTION_CLOSE с кодом ошибки для информирования партнёра.

Состояния закрытия и сброса соединения предназначены для аккуратного завершений соединений с отбрасыванием задержанных и разупорядоченных пакетов. Эти состояния **следует** сохранять в течение по меньшей мере 3 интервалов PTO, как указано в [QUIC-RECOVERY].

Удаление статуса соединения до выхода из состояния закрытия или сброса может приводить к генерации конечной точкой ненужного сообщения Stateless Reset при получении пакета с опозданием. Конечные точки, имеющие дополнительные средства предотвращения откликов на запоздалые пакеты, такие как закрытие сокета UDP, **могут** завершить эти состояния раньше для более быстрого освобождения ресурсов. Серверам, которые сохраняют открытый сокет для восприятия новых соединений, **не следует** раньше завершать состояние закрытия или сброса.

По завершении состояния закрытия или сброса конечной точке **следует** отбросить все состояния соединения и **можно** передавать Stateless Reset в ответ на запоздалые пакеты закрытого соединения.

10.2.1. Состояние Closing

Конечная точка переходит в состояние закрытия после инициирования немедленного закрытия. В этом состоянии конечная точка сохраняет лишь сведения, достаточные для генерации кадра CONNECTION_CLOSE и идентификации пакетов, относящихся к соединению. Конечная точка в таком состоянии передаёт пакет с кадром CONNECTION_CLOSE в ответ на любой входящий пакет, относящийся к соединению. Конечной точке **следует** ограничивать частоту генерации пакетов в состоянии закрытия. Например, можно ожидать получения прогрессивно увеличивающегося числа пакетов или просто увеличивать время отклика на принятые пакеты.

Выбранного конечной точкой идентификатора соединения и версии QUIC достаточно для идентификации пакетов закрываемого соединения. От остальных состояний конечная точка **может** отказаться и обработка полученных кадров от неё не требуется. Конечная точка **может** сохранять ключи защиты для входящих пакетов, чтобы прочесть и обработать кадр CONNECTION_CLOSE. Конечная точка **может** отбросить ключи защиты пакетов при переходе в состояние закрытия и передавать пакет с кадром CONNECTION_CLOSE в ответ на любую принятую дейтаграмму UDP. Однако, отбросив ключи защиты пакетов, конечная точка не сможет идентифицировать и отбросить недействительные пакеты. Для предотвращения атак с усилением такая конечная точка **должна** ограничивать общий размер передаваемых пакетов троекратным размером полученных пакетов, которые связаны с соединением. Для минимизации числа поддерживаемых состояний при закрытии соединения конечная точка **может** передавать один и тот же пакет в ответ на все принятые пакеты.

> Примечание. Возможность повторной передачи закрывающего пакета является исключением из правила использования нового номера для каждого пакета (см. параграф 12.3). Отправка новых номеров предназначена в первую очередь для восстановления при потерях, которые считаются несущественными для закрываемых соединений. Повторная передача финального пакета требует меньше состояний.

Находясь в состоянии закрытия, конечная точка может получать пакеты с новым адресом отправителя, возможно говорящие о переносе соединения (см. раздел 9). Она **должна** отбрасывать пакеты с непроверенных адресов или ограничивать общий размер передаваемых откликов троекратным размером принятого пакета.

В состоянии закрытия от конечной точки не ожидается обработка обновления ключей (раздел 6 в [QUIC-TLS]). Обновление ключа может помешать конечной точке перейти из состояния закрытия в состояние сброса, поскольку она не сможет обработать полученные позднее пакеты, но иного влияния это не оказывает.

10.2.2. Состояние Draining

Конечная точка переходит в состояние сброса при получении кадра CONNECTION_CLOSE, который указывает закрытие или сброс соединения партнёром. Хотя в остальном оно аналогично состоянию закрытия, конечной точке **недопустимо** передавать в состоянии сброса какие-либо пакеты. Сохранение ключей защиты пакетов не требуется.

Получив кадр CONNECTION_CLOSE, конечная точка **может** передать 1 пакет с кадром CONNECTION_CLOSE до перехода в состояние сброса, используя при необходимости код NO_ERROR. Последующая передача пакетов конечной точкой **недопустима**, поскольку может приводить к безостановочному обмену кадрами CONNECTION_CLOSE, пока одна из конечных точек не выйдет из состояния закрытия.

Конечная точка **может** перейти в состояние сброса из состояния закрытия при получении кадра CONNECTION_CLOSE, указывающего, что партнёр находится в состоянии закрытия или сброса. В этом случае состояние сброса заканчивается в тот момент, когда завершилось бы состояние закрытия. Иными словами, конечная точка в состоянии сброса использует то же время окончания, но не передаёт каких-либо пакетов.

10.2.3. Немедленное закрытие в процессе согласования

Целью отправки кадра CONNECTION_CLOSE является его обработка партнёром. В общем случае это означает передачу кадра в пакете с высшим уровнем защиты для предотвращения отбрасывания пакета. После того, как согласование подтверждено (см. параграф 4.1.2 в [QUIC-TLS]), конечная точка **должна** передавать любые кадры CONNECTION_CLOSE в пакетах 1-RTT. Однако до завершения согласования ключи расширенной защиты пакетов могут быть недоступны, поэтому другой кадр CONNECTION_CLOSE **может** быть передан в пакете с меньшей защитой.

- Клиент всегда знает о наличии у сервера ключей Handshake (см. параграф 17.2.2.1), но сервер может не знать о наличии таких ключей у клиента. При таких обстоятельствах серверу **следует** передавать кадр CONNECTION_CLOSE в обоих пакетах Handshake и Initial, чтобы клиент мог обработать хотя бы один из них.
- Клиент, передающий CONNECTION_CLOSE в пакете 0-RTT, не может быть уверен в восприятии сервером пакетов 0-RTT. Отправка CONNECTION_CLOSE в пакете Initial повышает вероятность получения сервером сигнала закрытия даже при отсутствии кода ошибки от приложения.
- До подтверждения согласования партнёр может быть не способен обрабатывать пакеты 1-RTT, поэтому конечной точке **следует** передавать CONNECTION_CLOSE в обоих пакетах Handshake и 1-RTT. Серверу также **следует** передавать кадр CONNECTION_CLOSE в пакете Initial.

Отправка CONNECTION_CLOSE типа 0x1d в пакете Initial или Handshake может раскрывать или изменять состояние приложения. Кадры CONNECTION_CLOSE типа 0x1d **должны** заменяться кадрами CONNECTION_CLOSE типа 0x1c при их отправке в пакете Initial или Handshake для предотвращения раскрытия статуса приложения. Конечные точки **должны** очищать поле Reason Phrase, а также им **следует** использовать код APPLICATION_ERROR при преобразовании CONNECTION_CLOSE типа 0x1c.

Кадры CONNECTION_CLOSE в пакетах разных типов могут быть объединены в одну дейтаграмму UDP (см. параграф 12.2).

Конечная точка может передать кадр CONNECTION_CLOSE в пакете Initial как отклик на неаутентифицированные данные в пакете Initial или Handshake. Такое незамедлительное закрытие может приводить к отказу в обслуживании легитимных соединений. Протокол QUIC не включает мер защиты от атак в пути в процессе согласования (см. параграф 21.2), однако за счёт снижения числа отзывов об ошибках некоторые формы атак на службы можно осложнить, если конечные точки отбрасывают нелегитимные пакеты вместо прерывания соединения с помощью CONNECTION_CLOSE. По этой причине конечные точки **могут** отбрасывать пакеты без аутентификации при обнаружении в них ошибок.

Конечная точка, не установившая состояние (например, сервер, обнаруживший ошибку в пакете Initial), не может перейти в состояние закрытия. Конечная точка, не имеющая состояния для соединения, не может задать время для состояния закрытия или сброса при отправке кадра CONNECTION_CLOSE.

10.3. Сброс без учёта состояния

Сброс без учёта состояния предоставляется в качестве крайней меры для конечной точки, не имеющей доступа к состоянию соединения. Отказ или остановка работы могут приводить к продолжению отправки данных партнёром. конечной точке, не способной продолжать соединение. Конечная точка **может** передать Stateless Reset в ответ на получение пакета, который она не может связать с активным соединением. Сброс без учёта состояния не подходит для индикации ошибок в активных соединениях. Конечная точка, желающая сообщить о критической ошибке, **должна** использовать кадр CONNECTION_CLOSE. Если это возможно.

Для поддержки процесса конечная точка выпускает маркер сброса без учёта состояния, который является трудно предсказуемым 16-байтовым значением. Если партнёр после этого получает Stateless Reset в форме дейтаграммы UDP, заканчивающейся маркером сброса, он будет незамедлительно прерывать соединение. Маркер сброса связан с идентификатором соединения и конечная точка вводит маркер, включая значение в поле Stateless Reset Token кадра NEW_CONNECTION_ID. Серверы также вводят при согласовании параметр транспорта stateless_reset_token, который применяется к идентификатору соединения, выбранному сервером при согласовании. Эти обмены защищены шифрованием, поэтому значение известно лишь клиенту и серверу. Клиент не может использовать транспортный параметр stateless_reset_token, поскольку для его транспортных параметров не обеспечивается конфиденциальность. Маркер аннулируется при удалении связанного с ним идентификатора соединения кадром RETIRE_CONNECTION_ID (параграф 19.16).

Конечная точка, получившая пакет, который она не способна обработать, передаёт пакет с показанной ниже схемой (см. параграф 1.3).

```
Stateless Reset {
  Fixed Bits (2) = 1,
  Unpredictable Bits (38..),
  Stateless Reset Token (128),
}
```

Рисунок 10. Сброс без учета состояния.

Это гарантирует, что пакет с Stateless Reset (насколько это возможно) не отличается от обычного пакета с коротким заголовком. Stateless Reset использует всю дейтаграмму UDP, начиная с первых двух битов заголовка пакета. Остаток первого байта и произвольное число байтов после него содержат значение, которому **следует** быть неотличимым от случайного. Последние 16 байтов дейтаграммы содержат маркер сброса без учёта состояния. Для всех объектов, кроме предусмотренного получателя, Stateless Reset будет выглядеть обычным пакетом с коротким заголовком. Чтобы пакет с Stateless Reset выглядел обычным пакетом QUIC, в поле Unpredictable Bits нужно включать не менее 38 битов данных (5 байтов без двух битов). Получаемый при этом минимальный размер в 21 байт не гарантирует, что Stateless Reset будет трудно отличить от других пакетов, если получатель требует использовать идентификатор соединения. Чтобы сделать пакет неотличимым, конечной точке **следует** обеспечить для всех передаваемых пакетов размер по меньшей мере на 22 байта больше минимального размера идентификатора соединения, запрошенного у партнёра для включения в пакеты, путём добавления кадров PADDING. Конечной точке, передающей Stateless Reset в ответ на пакет размером не более 43 байтов **следует** передавать Stateless Reset на 1 байт короче пакета, который вызвал сброс. Эти значения предполагают, что размер маркер сброса совпадает с размером минимального расширения пакетов при защите AEAD. Дополнительные непредсказуемые байты нужны в тех случаях, когда конечная точка может иметь согласованную схему защиты с большим увеличением размера пакета.

Конечное точке **недопустимо** передавать Stateless Reset с размером в 3 и более раз превышающим размер вызвавшего сброс пакета для предотвращения возможности использования этого для усиления атак. Дополнительные ограничения размера Stateless Reset приведены в параграфе 10.3.3.

Конечные точки **должны** отбрасывать пакеты, которые слишком малы для QUIC. Например, с набором функций AEAD, определенным в [QUIC-TLS], пакеты с коротким заголовком, размер которых меньше 21 байта, недействительны.

Конечные точки **должны** передавать Stateless Reset как пакеты с коротким заголовком. Однако конечные точки **должны** принимать любой пакет с корректным маркером сброса в конце как Stateless Reset, поскольку другие версии QUIC могут разрешать использование длинного заголовка.

Конечная точка **может** передать Stateless Reset в ответ на пакет с длинным заголовком. Отправка Stateless Reset не работает, пока у партнёра нет маркера сброса без учёта состояния. В данной версии QUIC пакеты с длинным заголовком применяются лишь при организации соединения. Поскольку маркер сброса не доступен, пока организация соединения не завершена, игнорирование неизвестных пакетов с длинным заголовком может быть так же эффективно, как отправка Stateless Reset.

Конечная точка не может определить Source Connection ID из короткого заголовка, поэтому она не может указать Destination Connection ID в Stateless Reset. Следовательно, Destination Connection ID будет отличаться от значения в предшествующих пакетах. Случайное значение Destination Connection ID представляет идентификатор соединения как результат переноса на другой идентификатор, представленный в кадре NEW_CONNECTION_ID (см. параграф 19.15). Применение случайного идентификатора соединения создаёт две проблемы, указанных ниже.

- Пакет может не попасть к партнёру. Если поле Destination Connection ID важно для маршрутизации к партнёру, пакет может быть отправлен не туда. Это также может вызвать в ответ другой пакет Stateless Reset (см. параграф 10.3.3). При некорректной маршрутизации Stateless Reset не будет эффективным механизмом обнаружения ошибок и восстановления. В этом случае конечным точкам нужны иные методы (такие как таймеры) для обнаружения отказов в соединениях.
- Случайный идентификатор соединения может использоваться сторонними объектами как индикатор возможного Stateless Reset. Конечная точка, использующая другие идентификаторы соединения, может внести ту или иную неопределённость.

Такой сброс без учёта состояния принят в QUIC версии 1. Конечной точке, которая может поддерживать несколько версий QUIC, потребуется генерировать Stateless Reset, которые будут восприняты партнёрами, поддерживающими любую версию, которую может (или могла до потери состояния) использовать конечная точка. Разработчики новых версий QUIC должны понимать это и (1) сохранять данную схему или (2) использовать для передачи данных часть пакета, отличную от последних 16 байтов.

10.3.1. Обнаружение Stateless Reset

Конечная точка обнаруживает Stateless Reset по последним 16 байтам дейтаграммы UDP, помня все маркеры сброса, связанные с идентификаторами соединения и удалёнными адресами для недавно отправленных дейтаграмм. Это включает значения поля Stateless Reset Token из кадров NEW_CONNECTION_ID и транспортных параметров сервера, но без маркеров сброса, связанных с идентификаторами соединения, которые не использовались или отозваны. Конечная точка сравнивает последние 16 байтов принятых дейтаграмм со всеми маркерами сброса для удалённого адреса, с которого получена дейтаграмма. Сравнение может выполняться для каждой принятой дейтаграммы. Конечная точка **может** пропустить эту проверку, если какой-либо пакет из дейтаграммы успешно обработан. Однако сравнение **должно** выполняться, если первый пакет во входящей дейтаграмме не удаётся связать с соединением или расшифровать. Конечной точке **недопустимо** проверять маркеры сброса, связанные с не использованным или отозванным идентификатором соединения.

При сравнении дейтаграммы с маркером сброса конечная точка **должна** выполнять операцию без утечки значения маркера. Например, сравнение в постоянный момент защищает значение отдельного маркера сброса от утечки информации через побочные каналы синхронизации. Другим подходом может служить сохранение и сравнение преобразованных значений маркеров вместо необработанных, где преобразование определяется как криптографическая псевдослучайная функция с использованием секретного ключа (например, блочный шифр или HMAC[1] [RFC2104]). От конечной точки не ожидается защиты сведений о расшифровке пакета или числе действительных маркеров сброса.

Если 16 последних байтов дейтаграммы совпадают с маркером сброса, конечная точка **должна** ввести период сброса и не передавать пакетов через это соединение.

10.3.2. Расчёт маркера Stateless Reset

Маркер сброса без учёта состояния **должен** быть трудно предсказуемым. Конечная точка может генерировать случайный секрет [RANDOM] для каждого организуемого соединения, который применяется при создании маркеров сброса. Однако здесь возникает проблема координации при наличии кластера с несколькими экземплярами или проблема хранилища для конечной точки, которая может потерять состояние. Сброс без учёта состояния предназначен именно для случаев потери состояния, поэтому такой подход не будет оптимальным.

Можно использовать один статический ключ для всех соединений с одной конечной точкой, генерируя свидетельство (proof) с использованием псевдослучайной функции, принимающей на входе статический ключ и идентификатор соединения, выбранный конечной точкой (см. параграф 5.1). Конечная точка может использовать HMAC [RFC2104] (например, HMAC(static_key, connection_id)) или функцию вывода ключей на основе HMAC (HKDF[2]) [RFC5869] (например, используя статический ключ в качестве входного ключевого материала и идентификатор соединения как затравку - salt). Результат функции отсекается до 16 байтов для создания ключа сброса в соединении. Конечная точка, потерявшая состояние, может применять тот же метод для создания маркера сброса, беря идентификатор соединения из принятого пакета.

Это решение основано на том, что конечная точка всегда передаёт идентификатор соединения в пакетах, поэтому его можно применять для сброса соединения. Использующая такое решение конечная точка **должна** применять во всех соединениях идентификаторы одного размера или кодировать их размер так, чтобы была возможность восстановить идентификатор при потере состояния. Кроме того, конечная точка не может использовать пустой идентификатор.

[1]Hashed Message Authentication Code - код аутентификации хэшированного сообщения.
[2]HMAC-based Key Derivation Function.

Обнаружение маркера сброса позволяет любому объекту разорвать соединение, поэтому значение маркера может применяться лишь один раз. Этот метод выбора маркера сброса без учёта состояния означает, что комбинацию статического ключа и маркера соединения **недопустимо** применять в другом соединении. Если использовать один идентификатор экземплярами, применяющими общий статический ключ, или атакующий может направить пакет экземпляру без состояния, знающему статический ключ, возможна организация DoS-атаки (см. параграф 21.11). Идентификатор соединения, сбрасываемого путём обнаружения маркера сброса, **недопустимо** снова применять в соединениях узлов, знающих общий статический ключ.

Один маркер сброса **недопустимо** использовать с несколькими идентификаторами соединения. Конечные точки не обязаны сравнивать новое значение со всеми прежними, но дубликаты **могут** трактоваться как ошибка соединения типа PROTOCOL_VIOLATION.

Отметим, что для Stateless Reset не обеспечивается криптографическая защита.

10.3.3. Зацикливание

Схема Stateless Reset такова, что без знания маркера пакеты сброса неотличимы от обычных пакетов. Например, если сервер передаст Stateless Reset другому серверу, он может получить в ответ Stateless Reset, что может приводить к бесконечному обмену. Конечная точка **должна** гарантировать, что передаваемый ею пакет Stateless Reset меньше вызвавшего отправку пакета, если у неё нет состояния, предотвращающего зацикливание. Такой подход ведёт к тому, что при возникновении петли пакеты в конечном итоге станут слишком мелкими, чтобы вызвать ответ.

Конечная точка может запоминать число переданных Stateless Reset и прекращать генерацию по достижении заданного предела. Использование независимых ограничений для каждого удалённого адреса обеспечивает возможность использовать Stateless Reset для закрытия соединения, когда партнёр или соединение уже исчерпали свои попытки сброса.

Stateless Reset размером меньше 41 байта может быть идентифицирован сторонним наблюдателем как пакет сброса, в зависимости от размера партнерского идентификатора соединения. И наоборот, отказ от передачи Stateless Reset в ответ на мелкий пакет может приводить к тому, что Stateless Reset станет бесполезным для обнаружения оборванных соединений, где передаются лишь мелкие пакеты. Такие отказы можно обнаруживать лишь иными средствами, такими как таймеры.

11. Обработка ошибок

Обнаружившей ошибку конечной точке **следует** сообщить об этом партнёру. На соединение в целом могут влиять ошибки как транспортного, так и прикладного уровня (см. параграф 11.1). В одном потоке могут изолироваться лишь ошибки на уровне приложения (см. параграф 11.2).

В указывающий ошибку кадр **следует** включать наиболее подходящий код ошибки (раздел 20). Если эта спецификация указывает условия ошибки, задаётся и код, который следует передавать. Хотя это и задано в форме требования, разные стратегии реализации могут приводить к разным сообщениям об ошибке. В частности, конечная точка **может** использовать любой применимый код ошибки, когда она видит условия ошибки. Вместо конкретного кода всегда можно использовать базовый код ошибки, например, PROTOCOL_VIOLATION или INTERNAL_ERROR.

Сброс без учёта состояния (параграф 10.3) не подходит для ошибок, которые можно указать в кадре CONNECTION_CLOSE или RESET_STREAM. Сброс без учёта состояния **недопустимо** применять конечной точке, имеющей состояние, которое требует отправки кадра в соединение.

11.1. Ошибки соединения

Ошибки, делающие невозможным использование соединения (например, очевидное нарушение семантики протокола или повреждение состояния, влияющее на все соединение), **должны** указываться кадром CONNECTION_CLOSE (параграф 19.19).

Относящиеся к приложению ошибки протокола указываются в кадре CONNECTION_CLOSE типа 0x1d, ошибки связанные с транспортом, включая описанные в этом документе, - в кадре CONNECTION_CLOSE типа 0x1c.

Пакет с кадром CONNECTION_CLOSE может быть потерян, поэтому конечной точке **следует** быть готовой к повтору пакета с кадром CONNECTION_CLOSE при получении дополнительных пакетов в разрываемом соединении. Ограничение числа повторов и времени, в течение которого они выполняются, упрощает разрыв соединения. При отказе конечной точки от повтора пакетов с кадром CONNECTION_CLOSE возникает риск того, что партнёр не получит пакет. В таком случае единственным механизмом прерывания соединения остаётся сброс без учёта состояния (параграф 10.3).

Поскольку AEAD для пакетов Initial не обеспечивает строгой аутентификации, конечная точка **может** отбросить недействительный пакет Initial. Такое отбрасывание разрешено даже при иной трактовке спецификацией ошибок соединения. Конечная точка может отбросить пакет лишь в том случае, когда она не обрабатывает кадры из него или не отменяет результат обработки. Отбрасывание недействительных пакетов Initial может служить для снижения риска DoS-атак (см. параграф 21.2).

11.2. Ошибки потока

Если ошибка на уровне приложения затрагивает один поток, а состояние соединения может быть восстановлено, конечная точка может передать кадр RESET_STREAM (параграф 19.4) с подходящим кодом ошибки для сброса потока. Сброс потока без участия протокола приложения может приводить к невосстанавливаемому состоянию прикладного протокола. Отправка RESET_STREAM **должна** вызываться лишь протоколом приложения, использующего QUIC. Семантика кода ошибки приложения в RESET_STREAM определяется прикладным протоколом и только он может вызвать прерывание потока. Локальный экземпляр прикладного протокола применяет прямой вызов API, а удалённый использует кадр STOP_SENDING, который автоматически ведёт к отправке RESET_STREAM.

Прикладному протоколу **следует** задавать правила обработки потоков преждевременно прерываемых конечными точками.

12. Пакеты и кадры

Конечные точки QUIC обмениваются пакетами, для который обеспечивается защита целостности и конфиденциальности (см. параграф 12.1). Пакеты передаются в дейтаграммах UDP (см. параграф 12.2).

Эта версия QUIC использует в пакетах длинный заголовок в процессе организации соединения (см. параграф 17.2). К таким пакетам относятся Initial (параграф 17.2.2), 0-RTT (параграф 17.2.3), Handshake (параграф 17.2.4), Retry (параграф 17.2.5). Для согласования версии используются независимые от версии пакеты с длинным заголовком (см. параграф 17.2.1).

Для минимизации издержек используются пакеты с коротким заголовком, которые передаются после организации соединения, когда доступны ключи 1-RTT (см. параграф 17.3).

12.1. Защищённые пакеты

Уровень криптографической защиты пакетов QUIC зависит от типа пакета. Детали защиты пакетов описаны в [QUIC-TLS], а в этом параграфе представлен обзор предоставляемой защиты.

Для пакетов согласования версии криптографическая защита не применяется (см. [QUIC-INVARIANTS]), пакеты Retry используют функцию AEAD [AEAD] для защиты от случайного изменения. Для пакетов Initial применяется защита AEAD с ключами, выводимыми из значения, доступного в линии, поэтому защита этих пакетов не эффективна и фактически у них нет защиты конфиденциальности. Защита пакетов Initial служит для гарантии того, что отправитель пакета присутствует в сетевом пути. Любой объект, получивший пакет Initial от клиента, может восстановить ключи, которые позволят ему прочитать содержимое пакета и создать пакеты Initial, которые будут аутентифицированы другой конечной точкой. AEAD также защищает пакеты Initial от непреднамеренных изменений.

Все остальные пакеты защищаются с использованием ключей, выведенных из криптографического согласования, которое обеспечивает получение соответствующих ключей для пакетов Handshake, 0-RTT и 1-RTT лишь взаимодействующими конечными точками.

Поле Packet Number в некоторых типах пакетов обеспечивает дополнительную защиту конфиденциальности, применяемую как часть защиты заголовка (см. параграф 5.4 в [QUIC-TLS]). Номер увеличивается для каждого пакета, переданного в данном пространстве номеров (см. параграф 12.3).

12.2. Объединение пакетов

Пакеты Initial (параграф 17.2.2), 0-RTT (параграф 17.2.3), Handshake (параграф 17.2.4) содержат поле Length, указывающее конец пакета. Размер учитывает поля Packet Number и Payload, для которых используется защита конфиденциальности и начальный размер не известен. Размер поля Payload определяется после снятия защиты заголовка.

Используя поле Length, отправитель может объединить несколько пакетов QUIC в одну дейтаграмму UDP. Это позволяет снизить число дейтаграмм UDP, требуемых для криптографического согласования и начала передачи данных, а также может применяться для создания зондов PMTU[1] (см. параграф 14.4.1). Получатели **должны** быть способны обрабатывать сборные пакеты.

Объединение пакетов в порядке роста уровня шифрования (Initial, 0-RTT, Handshake, 1-RTT, см. параграф 4.1.4 в [QUIC-TLS]) повышает вероятность обработки получателем всех пакетов за один проход. Пакет с коротким заголовком не включает поля размера, поэтому он может быть лишь последним в дейтаграмме UDP. Конечной точке **следует** включать несколько кадров в один пакет, если их нужно отправить с одним уровнем шифрования, вместо объединения нескольких пакетов с одним уровнем шифрования.

Получатели **могут** маршрутизировать информацию на основе первого пакета в дейтаграмме UDP. Отправителям **недопустимо** объединять пакеты QUIC с разными идентификаторами соединения в одну дейтаграмму UDP. Получателям **следует** игнорировать любые последующие пакеты с полем Destination Connection ID отличным от указанного в первом пакете дейтаграммы.

Каждый пакет QUIC, объединяемый в дейтаграмму UDP, является отдельным и полным. Получатель сборных пакетов QUIC **должен** отдельно обрабатывать каждый пакет QUIC и независимо подтверждать пакеты как при получении каждого в отдельной дейтаграмме UDP. Например, при отказе расшифровки (в результате недоступности ключей или по иной причине) получатель **может** отбросить пакет или буферизовать его для последующей обработки и **должен** попытаться обработать остальные пакеты.

Пакеты Retry (параграф 17.2.5), Version Negotiation (параграф 17.2.1) и пакеты с коротким заголовком (параграф 17.3) не включают поля Length, поэтому в дейтаграмме UDP за ними не могут следовать другие пакеты. Отметим также, что пакеты Retry или Version Negotiation никогда не объединяются с другими пакетами.

12.3. Номера пакетов

Номер пакета является целым числом от 0 до 2^{62}-1. Номера используются при определении криптографических значений nonce для защиты пакетов. Каждая конечная точка поддерживает раздельную нумерацию для принимаемых и передаваемых пакетов. Диапазон номеров пакетов ограничен для того, чтобы их можно было представит целиком в поле Largest Acknowledged кадра ACK (параграф 19.3). Однако при указании в длинном или коротком заголовке номера пакетов занимают от 1 до 4 байтов (см. параграф 17.1).

Пакеты Version Negotiation (параграф 17.2.1) и Retry (параграф 17.2.5) не включают порядкового номера.

Номера пакетов берутся из трёх пространств QUIC:

Пространство Initial
: Номера для всех пакетов Initial (параграф 17.2.2).

Пространство Handshake
: Номера для всех пакетов Handshake (параграф 17.2.4).

[1]Path Maximum Transmission Unit - максимальный размер передаваемого блока для пути.

Пространство данных приложения

Номера для всех пакетов 0-RTT (параграф 17.2.3) и 1-RTT (параграф 17.3.1).

Как указано в [QUIC-TLS], для каждого типа пакетов применяются свои ключи защиты. Концептуально пространство номеров пакетов является контекстом, где пакеты обрабатываются и подтверждаются. Пакеты Initial могут передаваться лишь с ключами защиты пакетов Initial и подтверждаются также в пакетах Initial. Аналогично пакеты Handshake передаются с уровнем шифрования Handshake и подтверждаются лишь в пакетах Handshake. Это обеспечивает криптографическое разделение между пакетами из разных пространств номеров. Номера пакетов в каждом пространстве начинаются с 0 и в каждом следующем пакете из того же пространства номер **должен** увеличиваться по меньшей мере на 1.

Данные 0-RTT и 1-RTT передаются в одном пространстве номеров для упрощения реализации механизмов восстановления при потере пакетов для этих двух типов.

Конечной точке QUIC **недопустимо** в одном соединении повторно использовать номера из того же пространства. Если номер передаваемого пакета достигает значения 2^{62}-1, отправитель **должен** закрыть соединение без отправки кадра CONNECTION_CLOSE и каких-либо дальнейших пакетов. Конечная точка **может** передать Stateless Reset (параграф 10.3) при получении дополнительных пакетов.

Получатель **должен** отбрасывать новый незащищённый пакет, если он не уверен, что не обработал другой пакет с тем же номером из этого же пространства. Подавление дубликатов **должно** выполняться после снятия защиты пакетов по причинам, указанным в параграфе 9.5 [QUIC-TLS].

Конечные точки, отслеживающие каждый отдельный пакет в поиске дубликатов, подвержены риску накопления избыточных состояний. Данные, требуемые для обнаружения дубликатов, можно ограничить, поддерживая минимальный номер для незамедлительного отбрасывания всех пакетов с меньшим номером. Минимум должен учитывать значительные вариации времени кругового обхода, которые включают возможность зондирования партнёром. путей через сеть с большими значениями времени кругового обхода (см. раздел 9).

Кодирование порядковых номеров отправителем и декодирование их получателем описаны в параграфе 17.1.

12.4. Кадры и их типы

Данные (payload) в пакетах QUIC после снятия защиты состоят из последовательности полных кадров, как показано на рисунке 11. Пакеты Version Negotiation, Stateless Reset, Retry не содержат в себе кадров.

```
Packet Payload {
  Frame (8..) ...,
}
```

Рисунок 11. Данные QUIC.

Данные пакетов, содержащих кадры, **должны** включать по меньшей мере 1 кадр и **могут** содержать несколько кадров разных типов. Конечная точка **должна** считать приём пакета без кадров ошибкой соединения PROTOCOL_VIOLATION. Кадр всегда помещается в один пакет QUIC и не может размещаться в нескольких пакетах. Каждый кадр начинается с поля Frame Type, указывающего тип кадра, за которым могут следовать другие поля, определяемые типом.

```
Frame {
  Frame Type (i),
  Type-Dependent Fields (..),
}
```

Рисунок 12. Базовая схема кадра.

В таблице 3 приведён список кадров, определённых в этой спецификации, и краткие сведения о них. Более подробные описания приведены ниже.

Таблица 3. Типы кадров.

Идентификатор типа	***Имя типа***	***Определение***	***Пакеты***	***Spec***
0x00	PADDING	Параграф 19.1	IH01	NP
0x01	PING	Параграф 19.2	IH01	
0x02-0x03	ACK	Параграф 19.3	IH_1	NC
0x04	RESET_STREAM	Параграф 19.4	__01	
0x05	STOP_SENDING	Параграф 19.5	__01	
0x06	CRYPTO	Параграф 19.6	IH_1	
0x07	NEW_TOKEN	Параграф 19.7	___1	
0x08-0x0f	STREAM	Параграф 19.8	__01	FF
0x10	MAX_DATA	Параграф 19.9	__01	
0x11	MAX_STREAM_DATA	Параграф 19.10	__01	
0x12-0x13	MAX_STREAMS	Параграф 19.11	__01	
0x14	DATA_BLOCKED	Параграф 19.12	__01	
0x15	STREAM_DATA_BLOCKED	Параграф 19.13	__01	
0x16-0x17	STREAMS_BLOCKED	Параграф 19.14	__01	
0x18	NEW_CONNECTION_ID	Параграф 19.15	__01	PP
0x19	RETIRE_CONNECTION_ID	Параграф 19.16	__01	
0x1a	PATH_CHALLENGE	Параграф 19.17	__01	P
0x1b	PATH_RESPONSE	Параграф 19.18	___1	P
0x1c-0x1d	CONNECTION_CLOSE	Параграф 19.19	ih01	NN
0x1e	HANDSHAKE_DONE	Параграф 20.20	___1	

Формат и семантика кадров описаны в разделе 19, а в оставшейся части параграфа рассмотрены важные и базовые свойства кадров.

Поле Frame Type в кадрах ACK, STREAM, MAX_STREAMS, STREAMS_BLOCKED, CONNECTION_CLOSE служит для передачи зависимых от кадра флагов. Для других кадров поле Frame Type просто указывает тип.

Столбец «Пакеты» в таблице 3 указывает типы пакетов, в которых может появляться кадр каждого типа.

I

Initial (параграф 17.2.2).

H

Handshake (параграф 17.2.4).

0

0-RTT (параграф 17.2.3).

1

1-RTT (параграф 17.3.1).

ih

Только кадр CONNECTION_CLOSE типа 0x1c может присутствовать в пакетах Initial и Handshake.

Дополнительные сведения об этих ограничениях приведены в параграфе 12.5. Отметим, что в пакетах 1-RTT могут присутствовать все типы кадров. Конечная точка **должна** считать получение кадра в пакете, для которого кадр не разрешён, ошибкой соединения типа PROTOCOL_VIOLATION.

В столбце Spec таблицы 3 указаны специальные правила генерации и обработки кадра.

N

Пакеты, содержащие лишь кадры с этой маркировкой, не требуют подтверждения (см. параграф 13.2).

C

Пакеты, содержащие лишь кадры с этой маркировкой, не учитываются как находящиеся в пути байты для контроля перегрузок (см. [QUIC-RECOVERY]).

P

Пакеты, содержащие лишь кадры с этой маркировкой, могут использоваться для зондирования новых путей при переносе соединения (см. параграф 9.1).

F

Содержимое кадров с этой маркировкой контролируется управлением потоком данных (см. раздел 4).

Столбцы «Пакеты» и Spec в таблице 3 не являются частью реестра IANA (см. параграф 22.4).

Конечная точка **должна** считать получение кадра неизвестного типа ошибкой соединения типа FRAME_ENCODING_ERROR.

В этой версии QUIC все кадры идемпотентны, т. е. действительный кадр не вызывает нежелательных побочных эффектов или ошибок при его неоднократном получении.

Поле Frame Type использует кодирование целых чисел переменного размера (раздел 16) с одним исключением. Для обеспечения простой и эффективной реализации синтаксического анализа кадров тип кадра **должен** кодироваться максимально кратко. Для определённых в этом документе типов кадров это означает однобайтовое представление, хотя их можно представить 2-, 4- или 8-байтовыми целыми числами переменного размера. Например, хотя 0x4001 является легитимным 2-байтовым представлением целого числа переменного размера со значением 1, кадры PING всегда представляются одним байтом со значением 0x01. Это правило применяется ко всем имеющимся и будущим типам кадров QUIC. Конечная точка **может** считать получение кадра с более длинным кодированием типа, чем требуется, ошибкой соединения типа PROTOCOL_VIOLATION.

12.5. Кадры и пространства номеров

Некоторые кадры запрещены в отдельных пространствах номеров. Приведённые здесь правила обобщают правила TLS в том смысле, что кадры, связанные с организацией соединения, обычно могут появляться в пакетах любого пространства номеров, тогда как связанные с передачей данных кадры могут присутствовать лишь в пакетах пространства номеров приложения.

- Кадры PADDING, PING, CRYPTO **могут** включаться в кадры любого пространства номеров.
- Кадры CONNECTION_CLOSE, сообщающие об ошибках уровня QUIC (тип 0x1c) **могут** присутствовать в пакетах любого пространства номеров. Кадры CONNECTION_CLOSE, сообщающие об ошибках приложения (тип 0x1d) **должны** включаться лишь в пакеты прикладного пространства номеров.
- Кадры ACK **могут** появляться в пакетах любого пространства номеров, но могут подтверждать лишь пакеты из того же пространства. Однако (см. ниже) в пакеты 0-RTT нельзя включать кадры ACK.
- Все остальные типы кадров **должны** передаваться лишь в пакетах прикладного пространства номеров.

Отметим, что по ряду причин в пакетах 0-RTT невозможно передавать кадры ACK, CRYPTO, HANDSHAKE_DONE, NEW_TOKEN, PATH_RESPONSE, RETIRE_CONNECTION_ID. Сервер **может** считать получение таких кадров в пакете 0-RTT ошибкой соединения типа PROTOCOL_VIOLATION.

13. Пакетизация и надёжность доставки

Отправитель передаёт один или несколько кадров в пакете QUIC (см. параграф 12.4). Отправитель может минимизировать расход пропускной способности и вычислительные затраты на уровне пакета путём включения максимально возможного числа кадров в каждый пакет QUIC. Отправитель **может** ждать в течение короткого времени сбора нескольких кадров перед отправкой пакета, который не заполнен до максимума, чтобы избежать отправки большого числа мелких пакетов. Реализация **может** использовать сведения о поведении или эвристику для определения времени ожидания. Этот период определяется реализацией и следует быть осторожным при выборе задержки, поскольку она будет влиять на задержку, с которой сталкивается приложение.

Мультиплексирование потоков обеспечивается чередованием кадров STREAM из разных пакетов в одном или нескольких пакетах QUIC. Пакет QUIC может включать множество кадров STREAM из одного или разных потоков.

Одним из преимуществ QUIC является предотвращение блокировки head-of-line между разными потоками. При наличии потерь пакетов блокируются лишь потоки с данными из потерянного пакета в ожидании повторной передачи, а другие потоки продолжают работать нормально. Отметим, что при включении данных из нескольких потоков в один пакет QUIC потеря такого пакета блокирует обработку всех этих потоков. Разработчикам рекомендуется включать в исходящие пакеты минимальное число потоков без потери эффективности в результате передачи мелких пакетов.

13.1. Обработка пакета

Пакет **недопустимо** подтверждать до снятия защиты и обработки всех содержащихся в нем кадров. Для кадров STREAM это означает, что данные помещены в очередь для отправки протоколу приложения, но не требуется их фактическая доставка и потребление.

После полной обработки пакета получатель подтверждает приём отправкой одного или нескольких кадров ACK с номером полученного пакета.

Конечной точке **следует** считать подтверждение пакета, который она не передавала, ошибкой соединения типа PROTOCOL_VIOLATION, если она может это обнаружить. Дополнительное обсуждение приведено в параграфе 21.4.

13.2. Генерация подтверждений

Конечные точки подтверждают все пакеты, которые они получили и обработали. Однако лишь пакеты с запросом подтверждения (ack-eliciting) вызывают отправку кадра ACK в интервале максимально разрешённой задержки подтверждения. Остальные пакеты подтверждаются лишь при отправке кадра ACK по иным причинам.

При любой отправке пакета конечной точке **следует** попытаться включить в него кадр ACK, если он ещё не передан. Это помогает партнёру своевременно обнаруживать потери пакетов.

В общем случае частые отклики от получателя улучшают реакцию на потери и перегрузку, но следует принимать во внимание нагрузку на получателя, связанную с генерацией кадра ACK в ответ на каждый принятый пакет с запросом подтверждения. Ниже приведены рекомендации, позволяющие обеспечить баланс.

13.2.1. Передача кадров ACK

Каждый пакет **следует** подтверждать по меньшей мере один раз, а пакеты с запросом подтверждения **должны** подтверждаться не менее 1 раза в интервале максимальной задержки, указанного конечной точкой в транспортном параметре max_ack_delay (см. раздел 18.2). Параметр max_ack_delay задаёт явное требование - конечная точка обещает никогда не задерживать преднамеренно подтверждения пакетов ack-eliciting более указанного времени. Если задержка происходит, любое превышение учитывается при оценке RTT и может приводить к ложным или задержанным повторам передачи партнёром. Отправитель использует значение max_ack_delay, заданное получателем, при определении тайм-аутов для повтора по времени, как описано в параграфе 6.2 [QUIC-RECOVERY].

Конечная точка **должна** подтверждать все пакеты Initial и Handshake с запросом подтверждения незамедлительно, а все пакеты 0-RTT и 1-RTT с запросом подтверждения в анонсированном интервале max_ack_delay, с одним исключением. До подтверждения согласования (handshake) у конечной точки может не быть ключей для расшифровки принимаемых пакетов Handshake, 0-RTT, 1-RTT, поэтому она может буферизовать их и подтверждать после обретения ключей.

Поскольку для пакетов, содержащих лишь кадры ACK, не применяется контроль перегрузки, конечной точке **недопустимо** передавать более одного такого пакета в ответ на приём пакета с запросом подтверждения.

Конечной точке **недопустимо** передавать пакет без запроса подтверждения в ответ на пакет без запроса подтверждения, даже при наличии пропуска перед полученным пакетом. Это позволяет избежать зацикливания откликов с подтверждениями, который может препятствовать переходу соединения в режим бездействия (idle). Пакеты без запроса подтверждения в конечном итоге подтверждаются отправкой кадра ACK в ответ на другие события.

Конечная точка, передающая лишь кадры ACK, не будет получать подтверждений от партнёра, пока эти подтверждения не будут включены в пакеты с кадрами, запрашивающими подтверждение. Конечной точке **следует** передавать кадр ACK с другими кадрами, когда имеются новые пакеты с запросом подтверждения. Когда нужно подтверждать лишь пакеты, не запрашивающие подтверждения, конечная точка **может** не передавать кадр ACK, пока не будет получен пакет с запросом подтверждения.

Конечная точка, передающая лишь пакеты без запроса подтверждения, может иногда добавлять в пакеты кадр с запросом подтверждения, чтобы получить подтверждение своих пакетов (см. параграф 13.2.4). В таких случаях конечной точке **недопустимо** передавать кадр с запросом подтверждения во всех пакетах, которые без этого не запрашивали подтверждения, чтобы избежать бесконечного зацикливания подтверждений.

Для оказания помощи в обнаружении потерь отправителем конечной точке **следует** создавать и отправлять кадр ACK без задержки при получении пакета с запросом подтверждения, а также:

- при получении пакета с номером меньше чем в другом полученном пакете с запросом подтверждения;
- при получении пакета с номером больше максимального среди полученных пакетов с запросом подтверждения, когда имеется пропуск номером между этими пакетами.

Точно так же пакеты с кодом ECN CE[1] в заголовке IP **следует** подтверждать незамедлительно для снижения времени отклика партнёра на перегрузку.

Предполагается, что алгоритмы [QUIC-RECOVERY] будут устойчивы к получателям, которые не следуют приведённым выше рекомендациям. Однако реализациям следует отходить от этих требований лишь при внимательном рассмотрении влияния на производительность для соединений данной конечной точки и других пользователей сети.

13.2.2. Частота подтверждений

Получатель определяет частоту отправки подтверждений в ответ на пакеты с запросом подтверждения с учётом рассмотренных ниже компромиссов.

Конечные точки полагаются на своевременные подтверждения при обнаружении потерь (см. раздел 6 в [QUIC-RECOVERY]). Контроллеры перегрузки на основе окна, такие как описано в разделе 7 [QUIC-RECOVERY], используют подтверждения для управления окном перегрузки. В обоих случаях задержка подтверждений может существенно влиять на производительность. С другой стороны, снижение частоты передачи пакетов, содержащих лишь

[1]Congestion Experienced - наличие перегрузки.

подтверждения, уменьшает издержки на передачу и обработку в обеих конечных точках. Это может значительно повысить пропускную способность на ассиметричных каналах и снизить объем трафика подтверждений на пути возврата (см. раздел 3 в [RFC3449]).

Получателю **следует** передавать кадр ACK после приёма по меньшей мере двух пакетов с запросом подтверждения. Эта рекомендация имеет общий характер и согласуется с рекомендациями для конечных точек TCP [RFC5681]. Знание условий в сети и сведения о контроллере перегрузок у партнёра, а также дополнительные исследования и эксперименты могут предложить иную стратегию подтверждения с лучшими показателями производительности.

Получатель **может** обработать несколько доступных пакетов до решения вопроса об отправке в ответ кадра ACK.

13.2.3. Поддержка диапазонов ACK

При отправке кадра ACK включается один или несколько диапазонов подтверждаемых пакетов. Подтверждение более старых пакетов снижает вероятность ненужных повторов, вызванных потерей кадров ACK за счёт роста размера ACK.

Кадрам ACK всегда **следует** подтверждать полученные недавно пакеты и чем больше разупорядоченных пакетов, тем более важна быстрая отправка обновлённого кадра ACK, чтобы партнёр не счёл пакеты потерянными и не повторил их. Предполагается, что кадр ACK поместится в один пакет QUIC. Если это не так, более старые пакеты (с меньшими номерами) исключаются из подтверждения.

Получатель ограничивает число диапазонов ACK (параграф 19.3.1), которые он запоминает и передаёт в кадрах ACK, как для ограничения размера кадров, так и для экономии ресурсов. После приёма подтверждений для кадра ACK получателю **следует** остановить отслеживание подтверждённых диапазонов ACK. Отправители могут ожидать подтверждения для большинства пакетов, но QUIC не гарантирует получение подтверждения для каждого пакета, обработанного получателем.

Сохранение множества диапазонов ACK может привести к чрезмерному росту кадра ACK. Получатель может отбросить неподтвержденные ACK Range для снижения размера кадра ACK за счёт роста числа повтором передачи отправителем. Это необходимо, если кадр ACK слишком велик для включения в пакет. Получатель **может** также дополнительно ограничить размер кадра ACK, чтобы сохранить место для других кадров и снизить расходуемую на подтверждения пропускную способность.

Получатель **должен** сохранять ACK Range, пока не может гарантировать, что позднее не воспримет пакеты из этого диапазона. Поддержка минимального номера пакета, возрастающего по мере отбрасывания диапазонов является одним из способов решения задачи с минимальным состоянием. Получатель может отбросить все ACK Range, но он **должен** хранить максимальный номер успешно обработанного пакета, поскольку он служит для восстановления номеров из последующих пакетов (см. параграф 17.1). Получателю **следует** включать ACK Range с наибольшим полученным номером пакета в каждый кадр ACK. Поле Largest Acknowledged применяется при проверке ECN у отправителя и включение значения меньше указанного в предыдущем кадре ACK может приводить к ненужному отключению ECN (см. параграф 13.4.2).

В параграфе 13.2.4 описан примерный подход к определению пакетов, подтверждаемых в каждом кадре ACK. Хотя цель алгоритма состоит в генерации подтверждения для каждого обработанного пакета, это не препятствует потере подтверждений.

13.2.4. Ограничение диапазонов путём отслеживания кадров ACK

При отправке пакета с кадром ACK поле Largest Acknowledged из этого кадра может быть сохранено. При подтверждении пакета с кадром ACK получатель может остановить подтверждение пакетов, в которых номера не больше Largest Acknowledged в переданном кадре ACK.

Получатель, который передаёт лишь пакеты без запроса подтверждения, такие как кадры ACK, может долго не получать подтверждений. Это может вынудить его поддерживать состояние для большого числа кадров ACK в течение длительного времени, а передаваемые кадра ACK могут стать чересчур большими. В таком случае получатель может передать PING или другой мелкий кадр с запросом подтверждения, например, один раз за интервал кругового обхода, для запроса у партнёра подтверждения ACK.

При отсутствии потерь ACK этот алгоритм допускает как минимум один интервал RTT с разупорядочением. При потере и нарушении порядка ACK этот подход не гарантирует, что каждое подтверждение отправитель увидит до того, как оно перестанет включаться в кадр ACK. Пакеты могут приниматься с нарушением порядка, а все последующие кадры ACK, содержащие подтверждение, могут быть потеряны. В этом случае алгоритм восстановления потерь может вызывать ложные повторы, но отправитель продолжит «двигаться вперёд».

13.2.5. Измерение и передача задержки на хосте

Конечная точка измеряет задержки, намеренно внесённые между приёмом пакета с наибольшим порядковым номером и отправкой подтверждения. Конечная точка представляет эту задержку подтверждения в поле ACK Delay кадра ACK (см. параграф 19.3), что позволяет получателю кадра ACK уточнить преднамеренные задержки, важные для более точной оценки RTT на пути при задержке подтверждений.

Пакет может перед обработкой храниться в ядре OS или ином месте. Конечной точке **недопустимо** включать не контролируемую ею задержку в поле ACK Delay кадра ACK. Однако конечным точкам **следует** включать задержки на буферизацию, вызванные недоступностью ключей расшифровки, поскольку такие задержки могут быть большими и обычно не повторяются.

Когда измеренная задержка подтверждения превышает max_ack_delay, конечной точке **следует** сообщить об измеренной задержке. Эта информация особенно полезна при согласовании, когда задержки могут быть велики (см. параграф 13.2.1).

13.2.6. Кадры ACK и защита пакета

Кадры ACK **должны** передаваться лишь в пакетах из одного пространства номеров с подтверждаемым пакетом (см. параграф 12.1). Например, пакеты, защищённые ключами 1-RTT, **должны** подтверждаться в пакетах, также защищённых ключами 1-RTT.

Пакеты, которые клиент передаёт с защитой 0-RTT, **должны** подтверждаться сервером в пакетах, защищённых ключами 1-RTT. Это означает, что клиент не может использовать эти подтверждения, если пакеты криптографического согласования от сервера задержаны или потеряны. Отметим, что такое же ограничение применимо и к другим данным, которые сервер защищает с помощью ключей 1-RTT.

13.2.7. Кадры PADDING, занимающие окно перегрузки

Пакеты с кадрами PADDING считаются находящимися в пути для контроля перегрузок [QUIC-RECOVERY]. Поэтому пакеты, содержащие лишь кадры PADDING, занимают окно перегрузки, не не создают подтверждений, открывающих окно перегрузки. Для предотвращения блокировки отправителю **следует** обеспечивать периодическую передачу других кадров в дополнение к кадрам PADDING, чтобы запросить подтверждения у получателя.

13.3. Повторная передача информации

Пакеты QUIC, для которых определена потеря, не передаются целиком. То же относится к кадрам, содержащимся в потерянных пакетах. Вместо этого информация, которая могла содержаться в потерянных кадрах, при необходимости передаётся в новых кадрах. Новые кадры и пакеты используются для передачи информации, которая сочтена потерянной. В общем случае информация передаётся снова, когда ясно, что пакет с этой информацией потерян и отправка прекращается, если пакет с этой информацией подтверждён.

- Данные из кадров CRYPTO передаются повторно в соответствии с правилами [QUIC-RECOVERY], пока все данные не будут подтверждены. Данные в кадрах CRYPTO пакетов Initial и Handshake отбрасываются при отбрасывании ключей для соответствующего пространства номеров.
- Прикладные данные из кадров STREAM передаются снова в других кадрах STREAM, пока конечная точка не передала RESET_STREAM для этого потока. После передачи передачи конечной точкой кадра RESET_STREAM, кадры STREAM передавать не требуется.
- Кадры ACK содержат набор наиболее свежих подтверждений и задержку подтверждения из самого большого подтверждённого пакета, как описано в параграфе 13.2.1. Задержка передачи пакетов с кадрами ACK или повторная передача ACK может вынудить партнёра генерировать завышенную выборку RTT или неоправданный запрет ECN.
- Отмена передачи потока в кадре RESET_STREAM передаётся, пока не будет подтверждена или пока партнёр не подтвердит все данные потока (т. е. будет достигнуто состояние Reset Recvd или Data Recvd на передающей стороне потока). Содержимое RESET_STREAM **недопустимо** менять при повторе передачи.
- Аналогично, запрос на отмену передачи потока в кадре STOP_SENDING, отправляется пока принимающая сторона потока не перейдёт в состояние Data Recvd или Reset Recvd (см. параграф 3.5).
- Сигналы закрытия соединения, включая пакеты с кадрами CONNECTION_CLOSE, не повторяются при обнаружении потери пакета. Повторная отправка этих сигналов описана в разделе 10.
- Максимальный объем данных текущего соединения передаётся в кадрах MAX_DATA. Обновлённое значение передаётся в кадре MAX_DATA, если пакет с наиболее свежим переданным кадром MAX_DATA был объявлен потерянным или конечная точка решила обновить предел. Нужна осторожность, чтобы избежать слишком частой отправки этого кадра, поскольку предел может увеличиваться часто и это вызовет передачу слишком большого числа кадров MAX_DATA (см. параграф 4.2).
- Текущее максимальное смещение данных потока передаётся в кадрах MAX_STREAM_DATA. Подобно MAX_DATA, обновлённое значение передаётся, если наиболее свежий кадр MAX_STREAM_DATA для потока потерян или предел обновлён, с мерами предосторожности для предотвращения отправки кадров слишком часто. Конечной точке **следует** останавливать передачу кадров MAX_STREAM_DATA, когда принимающая сторона потока переходит в состояние Size Known или Reset Recvd.
- Ограничение числа потоков данного типа передаётся в кадрах MAX_STREAMS. Подобно MAX_DATA, обновлённое значение передаётся, когда пакет с наиболее свежим MAX_STREAMS для этого типа потока объявлен потерянным или при обновлении предела, с мерами предосторожности для предотвращения слишком частой отправки.
- Сигналы блокировки передаются в кадрах DATA_BLOCKED, STREAM_DATA_BLOCKED и STREAMS_BLOCKED. Областью действия DATA_BLOCKED является соединение, STREAM_DATA_BLOCKED - поток, STREAMS_BLOCKED - конкретный тип потока. Новый кадр передаётся, если пакет с наиболее свежим кадром для области действия считается потерянным при условии блокировки конечной точки по соответствующему пределу. Эти кадры всегда включают предел, вызвавший блокировку на момент передачи.
- Проверка живучести или пути использует кадры PATH_CHALLENGE, передаваемые периодически, пока не будет получен соответствующий кадр PATH_RESPONSE или не отпадёт необходимость в проверке. Кадры PATH_CHALLENGE включают при каждой отправке разные данные.
- Отклики при проверке пути с использованием кадров PATH_RESPONSE передаются однократно. Предполагается, что партнёр передаст при необходимости дополнительные кадры PATH_CHALLENGE для отправки новых кадров PATH_RESPONSE.
- Новые идентификаторы соединения передаются в кадрах NEW_CONNECTION_ID и повторяются при потере пакета с сохранением порядкового номера. Отозванные идентификаторы соединения передаются в кадрах RETIRE_CONNECTION_ID и повторяются при потере пакета.

- Кадры NEW_TOKEN передаются повторно при потере пакета с кадром. Для этих кадров не выполняется обнаружения разупорядочения или дублирования сверх прямого сравнения содержимого кадров.
- Кадры PING и PADDING не содержат информации, поэтому их потеря не требует восстановления.
- Кадр HANDSHAKE_DONE **должен** передаваться повторно, пока не будет получено подтверждение.

Конечным точкам **следует** отдавать приоритет повторной передаче перед отправкой новых данных, пока заданные приложением приоритеты не требуют иного (см. параграф 2.3).

Несмотря на рекомендации отправителю собирать кадры с актуальной информацией при каждой отправке пакета, ему не запрещено повторно отправлять копии кадров из потерянных пакетов. Отправителю, повторно передающему копии кадров необходимо учитывать снижение доступного объёма данных в результате изменения размеров номера пакета и идентификатора соединения, а также MTU для пути. Получатель **должен** воспринимать пакеты с устаревшими кадрами, такими как MAX_DATA с меньшим максимальным объёмом данных нежели в полученных пакетах.

Отправителю **следует** избегать повторной передачи информации из пакетов после их подтверждения. Это включает пакеты, подтверждённые после того, как были сочтены потерянными, что может происходить при нарушении порядка пакетов в сети. Это требует от отправителя сохранять сведения о пакетах, которые объявлены потерянными. Отправитель может отбросить эти сведения по истечении времени, достаточного для переупорядочения, например, PTO (параграф 6.2 в [QUIC-RECOVERY]), или по другим событиям, таким как ограничение памяти.

При обнаружении потерь отправитель **должен** принять меры по контролю перегрузки. Детали обнаружения потерь и контроля перегрузки описаны в [QUIC-RECOVERY].

13.4. Явное уведомление о перегрузке

Конечные точки QUIC могут использовать ECN [RFC3168] для обнаружения перегрузки в сети и реагирования на неё. ECN позволяет конечной точке установить код ECT[1] в поле ECN пакета IP. Узел сети может тогда указать перегрузку, устанавливая код ECN-CE в поле ECN вместо отбрасывания пакета [RFC8087]. Конечные точки реагируют на сведения о перегрузке путём снижения скорости передачи, как описано в [QUIC-RECOVERY].

Для включения ECN передающая конечная точка QUIC сначала проверяет поддержку маркировки ECN на пути и возможность партнёра сообщать значения ECN в принятых пакетах IP (см. параграф 13.4.2).

13.4.1. Информирование о значениях ECN

Использование ECN требует от приёмной конечной точки считывать значение поля ECN из пакета IP, что возможно не на всех платформах. Если конечная точка не поддерживает ECN или не имеет доступа к полученным полям ECN, она не будет сообщать значения ECN из полученных пакетов.

Даже если конечная точка не устанавливает поле ECT в передаваемых пакетах, она **должна** предоставлять отклики о полученных маркерах ECN, если они доступны. Отказ от передачи этих значений вынудит отправителя отключить использование ECN в соединении.

При получении пакета IP с кодом ECT(0), ECT(1) или ECN-CE конечная точка с поддержкой ECN обращается к полю ECN и увеличивает соответствующее значение ECT(0), ECT(1) или ECN-CE. Значения ECN включаются в последующие кадры ACK (см. параграфы 13.2 и 19.3).

В каждом пространстве номеров пакетов поддерживаются отдельные состояния подтверждений и значения ECN. Объединённые пакеты QUIC (см. параграф 12.2) используют общий заголовок IP, поэтому счётчики ECN инкрементируются 1 раз для каждого объединённого пакета QUIC. Например, если пакеты Initial, Handshake и 1-RTT объединяются в одну дейтаграмму UDP, счётчики ECN для всех трёх пространств номеров будут инкрементироваться на 1.

Счётчики ECN инкрементируются лишь в том случае, когда пакеты QUIC из полученного пакета IP обработаны. Таким образом, дубликаты пакетов QUIC не увеличивают счётчики ECN (см. параграф 21.10 в части безопасности).

13.4.2. Проверка ECN

Неисправные устройства в сети могут повреждать или ошибочно отбрасывать пакеты с ненулевым кодом ECN. Для обеспечения связности при наличии таких устройств конечная точка проверяет значения ECN для каждого пути через сеть и отключает ECN на путях с ошибками. Для проверки ECN на новом пути:

- конечная точка устанавливает код ECT(0) в заголовке IP ранних исходящих пакетов на новом пути к партнёру [RFC8311];
- конечная точка отслеживает, все ли пакеты, переданные с кодом ECT, в конечном итоге будут потеряны (параграф 6 в [QUIC-RECOVERY]), что говорит об отказе при проверке ECN.

Если у конечной точки есть основания ожидать отбрасывания пакетов IP с кодом ECT неисправными элементами сети, она может устанавливать код ECT лишь для первого десятка исходящих пакетов на этом пути или в течении трёх PTO (см. параграф 6.2 в [QUIC-RECOVERY]). Если все пакеты с ненулевым кодом ECN будут потеряны, она может отключить маркировку в предположении, что та вызывает потерю пакетов. Таким образом, конечная точка пытается использовать ECN и проверяет это для каждого нового соединения при переключении на предпочтительный адрес сервера или активном переходе соединения на новый путь. Возможный алгоритм представлен в Приложении A.4. Возможны и другие методы проверки ECN на пути, а также иная стратегия маркировки. Реализации **могут** применять другие методы, определённые в RFC, см. [RFC8311].Реализациям, использующим код ECT(1) нужно выполнять проверку ECN с использованием сообщаемых значений ECT(1).

13.4.2.1. Прием кадров ACK со значениями ECN

[1]ECN-Capable Transport - транспорт с поддержкой ECN.

Ошибочная маркировка ECN-CE в сети может снижать производительность соединения. Конечная точка, получившая кадр ACK со значением ECN, проверяет полученное значение до его использования. Эта проверка выполняется путём сравнения недавно полученного значения со значениями из последнего успешно обработанного кадра ACK. Любое увеличение счётчика ECN проверяется на основе маркировки ECN, которая применялась для пакетов, которые недавно подтверждены в кадре ACK.

Если кадр ACK снова подтверждает пакет, переданный конечной точкой с кодом ECT(0) или ECT(1), это говорит о неудачной проверке ECN при отсутствии соответствующих счётчиков ECN в кадре ACK. Такая проверка обнаруживает элементы сети, которые обнуляют поле ECN, или партнёра, не сообщающего маркировку ECN. Проверка ECN также будет неудачной, если сумма роста счётчиков ECT(0) и ECN-CE меньше числа недавно подтверждённых пакетов, которые были переданы с маркировкой ECT(0). Точно так же проверка ECN будет неудачной, если сумма увеличения счётчиков ECT(1) и ECN-CE будет меньше числа недавно подтверждённых пакетов, которые были переданы с маркировкой ECT(1). Это позволяет обнаружить перемаркировку ECN-CE в сети.

Конечная точка может пропустить подтверждения для пакета при потере кадров ACK, поэтому общее увеличение счётчиков ECT(0), ECT(1) и ECN-CE может быть больше числа пакетов, недавно подтверждённых в кадре ACK. По этой причине счётчикам ECN разрешено превышать общее число подтверждённых пакетов.

Проверка счётчиков ECN из разупорядоченных кадров ACK может завершаться отказом. Конечной точке **недопустимо** учитывать отказ при проверке ECN в результате обработки кадра ACK, не увеличивающего максимальный подтверждённый порядковый номер.

Проверка ECN может быть неудачно если полученное общее значение счётчика для ECT(0) или ECT(1) превышает общее число пакетов, переданных с соответствующим кодом ECT. В частности, проверка будет неудачной, когда конечная точка получит отличный от 0 счётчик ECN, соответствующий не применявшемуся коду ECT. Такая проверка обнаруживает перемаркировку в ECT(0) или ECT(1) внутри сети.

13.4.2.2. Результаты проверки ECN

При неудачной проверке конечная точка **должна** отключить ECN и прекратить установку кода ECT в передаваемых пакетах IP, предполагая, что путь через сеть или партнёр не поддерживают ECN. Даже после неудачной проверки конечная точка **может** снова проверить ECN для того же пути позднее в процессе работы соединения. Конечная точка может периодически выполнять такие проверки.

После успешной проверки конечная точка **может** продолжать установку кода ECT в передаваемых пакетах, предполагая, что путь поддерживает ECN. Маршрутизация и элементы пути могут меняться при работе соединения и конечная точка **должна** отключить ECN, если проверка завершается отказом.

14. Размер дейтаграмм

Дейтаграмма UDP может содержать один или несколько пакетов QUIC. Размер дейтаграммы определяется общим размером данных UDP в одной дейтаграмме, переносящей пакеты QUIC. В размере дейтаграммы учитываются заголовки одного или нескольких пакетов QUIC и защищённые данные, но не учитываются заголовки UDP и IP.

Максимальный размер дейтаграммы определяется как наибольший размер данных UDP (payload), которые могут быть переданы через сетевой путь в одной дейтаграмме UDP. **Недопустимо** использование QUIC, если путь через сеть не поддерживает дейтаграммы размером по меньшей мере 1200 байтов.

В QUIC предполагается минимальный размер пакета IP не менее 1280 байтов. Эти минимальный размер для IPv6 [IPv6] и он поддерживается в большинстве современных сетей IPv4. В предположении заголовка IP размером 40 байтов для IPv6 и 20 для IPv4, а также размера заголовка UDP 8 байтов максимальный размер дейтаграммы будет 1232 байта для IPv6 и 1252 для IPv4. Таким образом, пути через современные сети IPv4 и все сети IPv6 предполагаются пригодными для QUIC.

> Примечание. Требование поддержки данных UDP размером 1200 байтов ограничивает пространство для заголовков расширения IPv6 значением 32 байта и опций IPv4 - значением 52 байта, если путь поддерживает лишь минимальное для IPv6 значение MTU в 1280 байтов. Это влияет на пакеты Initial и проверку пути.

Превышение значения 1200 байтов для максимального размера в можно обнаружить с помощью механизма PMTUD[1] (см. параграф 14.2.1) или DPLPMTUD[2] (см. параграф 14.3).

Применение транспортного параметра max_udp_payload_size (параграф 18.2) может служить дополнительным ограничением максимального размера дейтаграмм. Отправитель может избежать превышения этого предела, если значение известно. Однако до получения транспортного параметра конечные точки рискуют терять слишком большие дейтаграммы при передаче с превышением наименьшего разрешённого максимума в 1200 байтов.

Дейтаграммы UDP **недопустимо** фрагментировать на уровне IP. В IPv4 [IPv4] **должен** устанавливаться флаг запрета фрагментирования (Don't Fragment или DF), если это возможно.

QUIC иногда требует, чтобы дейтаграммы были не меньше определённого размера (см., например, параграф 8.1). Однако размер дейтаграмм не аутентифицируется, т. е. при получении конечной точкой дейтаграммы она не может знать, что отправитель передал дейтаграмму того же размера. Поэтому конечным точкам **недопустимо** закрывать соединение при получении дейтаграммы, не соответствующей ограничениям размера, но **можно** отбрасывать такие дейтаграммы.

14.1. Размер дейтаграмм Initial

Клиент **должен** расширять данные в дейтаграммах UDP с пакетом Initial по меньшей мере до наименьшего максимального размера дейтаграмм в 1200 байтов, добавляя кадры PADDING к пакету Initial или объединяя пакет Initial с другими (см. параграф 12.2). Пакет Initial можно объединить даже с недействительным пакетом, который получатель

[1] Path Maximum Transmission Unit Discovery - определение MTU на пути.
[2] Datagram Packetization Layer PMTU Discovery - определение MTU на пути для пакетизации дейтаграмм.

отбросит. Сервер также **должен** расширять данные всех дейтаграмм UDP с пакетом Initial, запрашивающим подтверждение, до наименьшего максимального размера дейтаграмм в 1200 байтов.

Передача дейтаграмм UDP такого размера гарантирует поддержку на пути через сеть приемлемого значения PMTU в обоих направлениях. Кроме того, расширение клиентом пакетов Initial помогает снизить амплитуду атак с усилением, вызываемых ответами сервера по непроверенным адресам клиентов (см. раздел 8).

Дейтаграммы с пакетом Initial **могут** иметь размер больше 1200 байтов, если отправитель считает, что путь в сети и партнёр поддерживают такой размер.

Сервер **должен** отбрасывать пакет Initial, переданный в дейтаграмме UDP с объёмом данных меньше наименьшего максимума в 1200 байтов. Сервер **может** также незамедлительно закрыть соединение, передав кадр CONNECTION_CLOSE с кодом ошибки PROTOCOL_VIOLATION (см. параграф 10.2.3).

Сервер также **должен** ограничивать число байтов, передаваемых до проверки адреса клиента (см. раздел 8).

14.2. MTU на пути

PMTU определяет максимальный размер пакета IP с учётом заголовков IP и UDP, а также данных UDP, включая один или несколько заголовков пакетов QUIC и защищённые данные. PMTU может зависеть от характеристик пути и меняться с течением времени. Максимальный размер данных UDP (payload), которые конечная точка может передать в данный момент называется максимальным размером дейтаграммы для конечной точки.

Конечной точке следует применять механизм DPLPMTUD (параграф 14.3) или PMTUD (параграф 14.2.1) для определения максимального размера (без фрагментации) дейтаграмм на пути к получателю. При отсутствии этих механизмов конечным точкам QUIC **не следует** передавать дейтаграммы размером больше наименьшего максимума.

DPLPMTUD и PMTUD передают дейтаграммы размером больше текущего максимума, называемые зондами PMTU. Всем пакетам QUIC, передаваемым не в зондах PMTU, **следует** ограничивать размер для размещения в дейтаграммах максимального размера, чтобы избежать фрагментации или отбрасывания [RFC8085].

Если конечная точка QUIC определяет, что PMTU между локальным и удаленным адресом IP не позволяет передать дейтаграммы с наименьшим максимальным размером в 1200 байтов, она **должна** незамедлительно прекратить передачу пакетов QUIC, за исключением пакетов в зондах PMTU, а также пакетов с кадрами CONNECTION_CLOSE по соответствующему пути. Конечная точка **может** прервать соединение при отсутствии других путей.

Каждая пара из локального и удалённого адреса может иметь своё значение PMTU. Реализациям QUIC, использующим какой-либо механизм определения PMTU, **следует** поддерживать максимальный размер дейтаграмм для каждой пары адресов IP (локальный и удалённый).

Реализация QUIC **может** быть более консервативной при расчёте максимального размера дейтаграмм, чтобы учесть неизвестные издержки туннелирования и/или опций (расширений) в заголовках IP.

14.2.1. Обработка сообщений ICMP в PMTUD

PMTUD [RFC1191] [RFC8201] полагается на приём сообщений ICMP (т. е. IPv6 Packet Too Big), указывающий, что пакет IP отброшен из-за превышения MTU на маршрутизаторе. DPLPMTUD также может использовать такие сообщения. Этот подход имеет уязвимость для атак со стороны объектов, не способных наблюдать за пакетами, но могущим угадать используемые на пути адреса. Такие атаки позволяют снизить значение PMTU для уменьшения пропускной способности. Конечная точка **должна** игнорировать сообщения ICMP, заявляющие о снижении PMTU ниже наименьшего для QUIC максимума размера дейтаграмм.

В требованиях к генерации ICMP [RFC1812] [RFC4443] сказано, что в пакет следует включать максимально возможную часть исходного пакета без превышения минимального MTU для версии IP. Размер включаемой в сообщение части пакета может быть меньше или информация может быть непонятной, как указано в параграфе 1.1 [DPLPMTUD].

Конечным точкам QUIC, использующим PMTUD, **следует** проверять сообщения ICMP для защиты от вставки пакетов, как указано в [RFC8201] и параграфе 5.2 [RFC8085]. Для этой проверки **следует** использовать включённую в сообщение часть пакета для сопоставления сообщения ICMP с соответствующим транспортным соединением (см. параграф 4.6.1 в [DPLPMTUD]). Проверка сообщения ICMP **должна** включать сопоставление адреса IP и порта UDP [RFC8085], а по возможности идентификатор соединения для активной сессии QUIC. Конечное точке **следует** игнорировать сообщения ICMP, не прошедшие проверку.

Конечной точке **недопустимо** увеличивать PMTU на основе сообщений ICMP (см. п. 6 в разделе 3 [DPLPMTUD]). Снижение максимального размера дейтаграмм QUIC в ответ на сообщения ICMP **может** быть предварительным, пока алгоритм обнаружения потерь QUIC не укажет, что пакет был действительно потерян.

14.3. Определение PMTU для уровня пакетизации дейтаграмм

DPLPMTUD [DPLPMTUD] полагается на отслеживание потерь или подтверждений пакетов QUIC, переданных в зондах PMTU. Зонды PMTU для DPLPMTUD, использующие кадр PADDING, реализуют «Зондирование с использованием заполнения», описанное в параграфе 4.1 [DPLPMTUD].

Конечным точкам **следует** устанавливать исходное значение BASE_PLPMTU (параграф 5.1 в [DPLPMTUD]) в соответствии с наименьшим максимальным размером дейтаграмм QUIC. MIN_PLPMTU совпадает с BASE_PLPMTU.

Конечные точки QUIC, реализующие DPLPMTUD, поддерживают максимальный размер пакета DPLPMTUD (Maximum Packet Size или MPS) (параграф 4.4 в [DPLPMTUD]) для каждой пары из локального и удалённого адреса IP. Это соответствует максимальному размеру дейтаграмм.

14.3.1. DPLPMTUD и связность для Initial

С точки зрения DPLPMTUD протокол QUIC является уровнем пакетизации (Packetization Layer или PL) с подтверждениями. Поэтому отправитель QUIC может перейти в состояние DPLPMTUD BASE (параграф 5.2 в [DPLPMTUD]) по завершении согласования QUIC.

14.3.2. Проверка сетевого пути с DPLPMTUD

QUIC является PL с подтверждениями, поэтому отправитель QUIC не реализует DPLPMTUD CONFIRMATION_TIMER в состоянии SEARCH_COMPLETE (см. параграф 5.2 в [DPLPMTUD]).

14.3.3. Обработка сообщений ICMP в DPLPMTUD

Конечная точка, применяющая DPLPMTUD, требует проверки любого полученного сообщения ICMP PTB до использования сведений PTB, как указано в параграфе 4.6 [DPLPMTUD]. В дополнение к проверке порта UDP протокол QUIC проверяет сообщение ICMP путём использования других сведений PL (например, проверку идентификатора соединения во включённой в сообщение ICMP части исходного пакета).

Рассмотренные в параграфе 14.2.1 вопросы обработки сообщений ICMP применимы и к сообщениям, используемым DPLPMTUD.

14.4. Отправка зондов QUIC PMTU

Зонды PMTU являются пакетами с запросом подтверждения.

Конечные точки могут ограничивать содержимое зондов PMTU кадрами PING и PADDING, поскольку пакеты с размером, превышающий текущий максимум для дейтаграмм, с большой вероятностью будут отброшены сетью. Поэтому потеря пакета QUIC в зонде PMTU не является надёжной индикацией перегрузки и ей **не следует** вызывать реакцию механизма контроля перегрузок (см. п. 7 в разделе 3 [DPLPMTUD]). Однако зонды PMTU занимают окно перегрузки, что может задерживать последующую передачу данных приложения.

14.4.1. Зонды PMTU с Source Connection ID

Конечные точки, полагающиеся на поле Destination Connection ID при маршрутизации входящих пакетов QUIC, вероятно потребуют включения идентификатора соединения в зонды PMTU для корректной маршрутизации возвращаемых сообщений ICMP (параграф 14.2.1). Однако поле Source Connection ID содержат лишь длинные заголовки (параграф 17.2), а пакеты с таким заголовком не шифруются и не подтверждаются партнёром после завершения согласования.

Одним из способов создания зонда PMTU является объединение (см. параграф 12.2) пакета с длинным заголовком, такого как Handshake или 0-RTT (параграф 17.2) и пакета с коротким заголовком в одну дейтаграмму UDP. Если полученный зонд PMTU достигает конечной точки, пакет с длинным заголовком она проигнорирует, но подтвердит пакет с коротким заголовком. Если зонд PMTU вызывает отправку сообщения ICMP, в это сообщение будет включена первая часть пробного пакета. Если поле Source Connection ID попадает в эту часть, оно может служить для маршрутизации или проверки сообщения ICMP.

> Примечание. Цель использования пакета с длинным заголовком заключается лишь в попытке включить поле Source Connection ID в часть пакета, возвращаемую в сообщении ICMP. Этот пакет не обязан быть действительным и его можно передавать даже в том случае, когда пакеты данного типа не используются.

15. Версии

Версия QUIC указывается 32-битовым целым числом без знака. Значение 0x00000000 зарезервировано для согласования версий, а текущая версия протокола имеет номер 0x00000001.

Другие версии QUIC могут отличаться свойствами. Набор свойств QUIC, гарантированных в любой версии, описан в [QUIC-INVARIANTS].

Версия 0x00000001 протокола QUIC применяет TLS в качестве протокола криптографического согласования, как указано в [QUIC-TLS].

Версии, где 16 старших битов имеют значение 0, зарезервированы для использования в согласованных IETF документах.

Версии, соответствующие шаблону 0x?a?a?a?a, зарезервированы для использования в принудительном согласовании версии, когда принимается любой номер версии, где 4 младших бита каждого байта имеют двоичное значение 1010. Клиент или сервер **может** анонсировать поддержку любой из этих зарезервированных версий.

Резервные номера версий не представляют реальный протокол. Клиент **может** использовать такую версию в ожидании того, что сервер инициирует согласование версии. Сервер **может** анонсировать поддержку одной из таких версий, ожидая, что клиент проигнорирует это значение.

16. Представление целочисленных полей переменного размера

В пакетах и кадрах QUIC широко применяется представление неотрицательных целых чисел с переменным размером. Такое представление обеспечивает меньший размер для небольших целых чисел.

QUIC резервирует 2 старших бита для представления размера числа двоичным логарифмом. Остальные биты служат для значения числа с использованием сетевого порядка байтов. Это означает представление целых чисел 1, 2, 4 или 8 байтами и значения этих чисел выражаются 6, 14, 30 или 62 байтами, соответственно (таблица 4).

Таблица 4. Сводка представления целых чисел.

2 старших бита	Число байтов	Число битов значения	Диапазон
00	1	6	0-63
01	2	14	0-16383
10	4	30	0-1073741823
11	8	62	0-4611686018427387903

Пример алгоритма декодирования и образцы кодирования приведены в Приложении A.1.

Значения не требуется кодировать с минимально возможным размером за исключением поля Frame Type (параграф 12.4).

Версии (раздел 15), номера пакетов в заголовках (раздел 17.1) и размер идентификатора соединения в длинном заголовке (параграф 17.2) задаются целыми числами, но не используют описанного здесь кодирования.

17. Формат пакетов

Все численные значения используют сетевой порядок байтов (big endian), размеры полей указываются в битах, значения полей указаны в шестнадцатеричном формате.

17.1. Кодирование и декодирование номера пакетов

Номера пакетов указываются числами от 0 до 2^{62}-1 (параграф 12.3). При указании в заголовке они имеют размер от 1 до 4 байтов. Число используемых битов уменьшается за счёт включения лишь младших битов порядкового номера. Номер пакета защищается, как указано в параграфе 5.4 [QUIC-TLS].

До получения подтверждения пространства порядковых номеров **должен** указываться полный номер пакета без отсечки, описанной ниже. После подтверждения пространства номеров отправитель **должен** использовать для номера размер, позволяющий указать диапазон в 2 раза больше разности между наибольшим подтверждённым номером и номером передаваемого пакета. Получающий пакет партнёр сможет корректно декодировать номер пакета, если тот не был задержан при передаче так, что до него было получено много пакетов с большими номерами. Конечной точке **следует** использовать достаточно длинное представление номера, чтобы его можно было восстановить даже по прибытии пакета позже отправленных после него других пакетов.

В результате при кодировании номера пакета применяется по меньшей мере на 1 бит больше двоичного логарифма числа непрерывных неподтвержденных пакетов, включая новый пакет. Псевдокод и примеры кодирования номеров пакетов представлены в Приложении A.2.

На стороне получателя защита порядкового номера снимается до восстановления полного номера. Затем восстанавливается полный номер на основе числа присутствующих значимых битов, их значений и наибольшего номера, полученного в успешно аутентифицированном пакете. Восстановление полного номера требуется для полного снятия защиты пакета. После снятия защиты заголовка номер пакета декодируется путём нахождения номера, ближайшего к следующему ожидаемому пакету, которым служит пакет с номером на 1 больше максимального номера принятого пакета. Псевдокод и примеры декодирования порядкового номера представлены в Приложении A.3.

17.2. Пакеты с длинным заголовком

```
Long Header Packet {
Header Form (1) = 1,
Fixed Bit (1) = 1,
Long Packet Type (2),
Type-Specific Bits (4),
Version (32),
Destination Connection ID Length (8),
Destination Connection ID (0..160),
Source Connection ID Length (8),
Source Connection ID (0..160),
Type-Specific Payload (..),
}
```

Рисунок 13. Формат пакета с длинным заголовком.

Длинные заголовки применяются в пакетах, передаваемых до организации ключей 1-RTT. Когда ключи 1-RTT доступны, отправитель переходит к передаче пакетов с коротким заголовком (параграф 17.3). Длинная форма позволяет представить особые пакеты, такие как Version Negotiation, в едином формате с фиксированным размером. Поля пакетов с длинным заголовком перечислены ниже.

Header Form
: Старший бит (0x80) байта 0 (первый байт) устанавливается в длинном заголовке.

Fixed Bit
: Следующий бит (0x40) байта 0 устанавливается (1), если пакет не относится к Version Negotiation. Пакеты со сброшенным битом считаются в этой версии недействительными и **должны** отбрасываться. Установка этого бита позволяет QUIC сосуществовать с другими протоколами, см. [RFC7983].

Long Packet Type
: Два следующих бита (маска 0x30) байта 0 указывают тип пакета (см. таблицу 5).

Type-Specific Bits
: Семантика 4 младших битов (маска 0x0f) байта 0 определяется типом пакета.

Version
: QUIC Version - 32-битовое поле, следующее за первым байтом, указывающее используемую версию QUIC и определяющее интерпретацию остальных полей протокола.

Destination Connection ID Length
: Байт после номера версии указывает размер (в байтах) следующего за ним поля Destination Connection ID, представляемый 8-битовым целым числом без знака. В QUIC версии 1 этому значению **недопустимо** превосходить 20. Конечная точка версии 1, получившая заголовок со значением больше 20 в этом поле, **должна** отбросить пакет. Для корректного формирования пакета Version Negotiation серверам **следует** поддерживать возможность чтения более длинного идентификатора соединения из других версий QUIC.

Destination Connection ID
: Поле Destination Connection ID следует за полем Destination Connection ID Length, указывающим его размер. Использование этого поля описано в параграфе 7.2.

Source Connection ID Length
: Байт после поля Destination Connection ID указывает размер (в байтах) следующего за ним поля Source Connection ID, представленный 8-битовым целым числом без знака. В QUIC версии 1 этому значению **недопустимо**

превосходить 20. Конечная точка версии 1, получившая заголовок со значением больше 20 в этом поле, **должна** отбросить пакет. Для корректного формирования пакета Version Negotiation серверам **следует** поддерживать возможность чтения более длинного идентификатора соединения из других версий QUIC.

Source Connection ID

Поле Source Connection ID следует за полем Source Connection ID Length, указывающим его размер. Использование этого поля описано в параграфе 7.2.

Type-Specific Payload

Оставшаяся часть пакета (при её наличии) зависит от типа пакета.

В этой версии QUIC определены показанные в таблице 5 типы пакетов с длинным заголовком.

Таблица 5. Типы пакетов с длинным заголовком.

Тип	***Имя***	***Параграф***
0x00	Initial	17.2.2. Начальный пакет
0x01	0-RTT	17.2.3. Пакет 0-RTT
0x02	Handshake	17.2.4. Пакет согласования
0x03	Retry	17.2.5. Пакет Retry

Бит формы заголовка, поля размера идентификаторов соединения и самих идентификаторов, а также поле Version в длинном заголовке не зависят от версии. Остальные поля первого байта зависят от версии протокола. В [QUIC-INVARIANTS] описана интерпретация пакетов разных версий QUIC. Интерпретация этих полей и данных (payload) определяется версией и типом пакета. Задаваемая типом семантика для данной версии описана в следующих параграфах, однако некоторые типы пакетов с длинным заголовком в этой версии QUIC содержат дополнительные поля.

Reserved Bits

Два бита (маска 0x0c) байта 0 зарезервированы для нескольких типов пакетов. Эти биты охватывается защитой заголовка (см. параграф 5.4 в [QUIC-TLS]). До установки защиты биты **должны** иметь значение 0. Конечная точка **должна** считать получение пакета с отличными от 0 значениями этих битов после снятия защиты пакета и заголовка ошибкой соединения типа PROTOCOL_VIOLATION. Отбрасывание такого пакета после снятия лишь защиты заголовка может раскрывать конечную точку для атаки (см. параграф 9.5 в [QUIC-TLS]).

Packet Number Length

В типах пакетов с полем Packet Number два младших бита (маска 0x03) байта 0 указывают размер поля Packet Number, представленный 2-битовым целым число без знака на 1 меньше фактического размера поля Packet Number в байтах. Т. е. размер поля Packet Number определяется добавлением 1 к значению этого поля. Эти биты охватывается защитой заголовка (см. параграф 5.4 в [QUIC-TLS]).

Length

Размер оставшейся части пакета (поля Packet Number и Payload) в байтах, представленный целым числом без знака с переменным размером (раздел 16).

Packet Number

Поле размером от 1 до 4 Байтов, охватываемое защитой заголовка (см. параграф 5.4 в [QUIC-TLS]). Размер поля Packet Number представлен в поле Packet Number Length байта 0, как описано выше.

Packet Payload

Данные (содержимое) пакета, представляющие собой последовательность кадров, охватываемую защитой пакета.

17.2.1. Пакет согласования версии

Пакет Version Negotiation по своей природе не зависит от версии. При получении клиентом пакет будет идентифицирован как Version Negotiation по полю Version = 0. Пакет согласования версии служит откликом на пакет клиент, запросивший у сервера не поддерживаемую тем версию. Пакет передаётся только серверами.

```
Version Negotiation Packet {
  Header Form (1) = 1,
  Unused (7),
  Version (32) = 0,
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Supported Version (32) ...,
}
```

Рисунок 14. Пакет согласования версии.

В поле Unused сервер устанавливает произвольное значение, а клиент **должен** игнорировать это поле. Если QUIC может мультиплексироваться с другими протоколами (см. [RFC7983]), серверу **следует** установить старший бит поля (0x40), чтобы в пакетах Version Negotiation было установлено поле Fixed Bit. Отметим, что в других версиях QUIC эта рекомендация может отсутствовать.

Поле Version в пакете Version Negotiation **должно** иметь значение 0x00000000.

Сервер **должен** включить значение Source Connection ID из полученного от клиента пакета в поле Destination Connection ID. Значение поля Source Connection ID **должно** копироваться из поля Destination Connection ID в полученном пакете, которое клиент выбрал случайно. Возврат обоих идентификаторов соединения даёт клиенту уверенность в том, что сервер получил запрос от клиента и пакет Version Negotiation не был создан объектом, который не видел пакета Initial.

В будущих версиях QUIC могут смениться требования к размеру идентификаторов соединения. В частности, для них может выбран меньший минимальный размер или больший максимальный. Зависящие от версии правилам для идентификаторов соединения **недопустимо** учитывать при принятии решения об отправке пакета Version Negotiation.

Оставшаяся часть пакета Version Negotiation содержит список 32-битовых номеров версий, поддерживаемых сервером.

Пакеты Version Negotiation не подтверждаются и передаются лишь в ответ на пакет, указывающий неподдерживаемую версию (см. параграф 5.)2.2.

Пакет Version Negotiation не включает полей Packet Number и Length, имеющихся в других пакетах с длинным заголовком, поэтому такой пакет занимает дейтаграмму UDP целиком. Серверу **недопустимо** передавать более 1 пакета Version Negotiation в ответ на одну дейтаграмму UDP. Процесс согласования версии описан в разделе 6.

17.2.2. Начальный пакет

Пакет Initial использует длинный заголовок со значением типа 0x00 и передаёт первые кадры CRYPTO от клиента и сервера для обмена ключами, а также кадры ACK в любом из направлений.

```
Initial Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 0,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Token Length (i),
  Token (..),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Рисунок 15. Пакет Initial.

Пакет Initial содержит длинный заголовок с полями Length и Packet Number (см. параграф 17.2). Первый байт содержит биты Reserved и Packet Number Length (см. параграф 17.2). Между полями Source Connection ID и Length размещаются два дополнительных поля, специфических для пакетов Initial.

Token Length

Целое число переменного размера, задающее размер поля Token в байтах. Значение 0 указывает отсутствие маркера. Пакеты Initial от сервера **должны** содержать Token Length = 0 и клиент, получающий пакет Initial с ненулевым значением Token Length **должен** отбросить пакет или установить ошибку соединения типа PROTOCOL_VIOLATION.

Token

Значение маркера, представленное ранее в пакете Retry или кадре NEW_TOKEN (см. параграф 8.1).

Чтобы предотвратить вмешательство промежуточных устройств, не знающих о версии, пакеты Initial защищаются с ключами, зависящими от соединения и версии (ключи Initial), как описано в [QUIC-TLS]. Эта защита не обеспечивает конфиденциальности и целостности при атаках с возможностью наблюдения пакетов, но препятствует злоумышленникам, не способным наблюдать пакеты, организовать атаку с фиктивными пакетами Initial.

Клиент и сервер могут применять пакеты типа Initial для любых пакетов, включающих сообщение начального криптографического согласования. Сюда входят все случаи, когда нужно создавать новые пакеты начального криптографического согласования, такие как пакеты, передаваемые после пакета Retry (см. параграф 17.2.5).

Сервер передаёт свой первый пакет Initial в ответ на клиентский пакет Initial и **может** передать несколько пакетов Initial. Для обмена криптографическими ключами может потребоваться несколько круговых обходов или повтор передачи.

Данные в пакете Initial включают кадр(ы) CRYPTO с сообщением криптографического согласования, кадры ACK или оба типа. Разрешены также кадры PING, PADDING, CONNECTION_CLOSE типа 0x1c. Конечная точка, получившая пакет Initial с кадрами иных типов, может отбросить пакет как ложный или считать его ошибкой соединения.

Первый пакет от клиента всегда включает кадр CRYPTO, содержащий начало или все сообщение криптографического согласования. Первый кадр CRYPTO всегда начинается со смещения 0 (см. раздел 7).

Отметим, что при передаче сервером TLS HelloRetryRequest (параграф 4.7 в [QUIC-TLS]) клиент будет передавать другую серию пакетов Initial, которые будут продолжать криптографическое согласования и включать кадры CRYPTO, начинающиеся со смещения, соответствующего размеру кадров CRYPTO в первой отправке пакетов Initial.

17.2.2.1. Прекращение использования пакетов Initial

Клиент прекращает отправку и обработку пакетов Initial при отправке своего первого пакета Handshake, сервер - при получении своего первого пакета Handshake. Хотя пакеты ещё могут оставаться в сети или ждать подтверждения, дополнительные пакеты Initial больше не нужны с этого момента. Ключи защиты пакетов Initial отбрасываются (см. параграф 4.9.1 в [QUIC-TLS]) вместе с состояниями восстановления потерь и контроля перегрузки (см. параграф 6.4 в [QUIC-RECOVERY]). Данные из кадров CRYPTO отбрасываются (и не передаются повторно) после отбрасывания ключей Initial.

17.2.3. Пакет 0-RTT

Пакет 0-RTT использует длинный заголовок типа 0x01, за которым следуют поля Length и Packet Number (см. параграф 17.2). Первый байт содержит поля Reserved и Packet Number Length (см. параграф 17.2). Пакеты 0-RTT используются для передачи «ранних» данных от клиента к серверу как части первой отправки до завершения согласования. В процессе согласования TLS сервер может воспринять или отвергнуть эти данные. Обсуждение данных 0-RTT и связанных с ними ограничений приведено в параграфе 2.3 [TLS13].

Номера пакетов с защитой 0-RTT используют то же пространство, что и защищённые пакеты 1-RTT.

```
0-RTT Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 1,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Рисунок 16. Пакет 0-RTT.

Получение клиентом пакета Retry говорит, что пакеты 0-RTT вероятно были потеряны или отброшены сервером. Клиенту **следует** попытаться повторно передать пакеты 0-RTT после отправки нового пакета Initial. Для новых пакетов **должны** использоваться новые номера, как указано в параграфе 17.2.5.3, поскольку повторное использование номеров может ставить защиту пакетов под угрозу.

Клиент получает подтверждения для его пакетов 0-RTT лишь по завершении согласования, как указано в параграфе 4.1.1 [QUIC-TLS].

Клиенту **недопустимо** передавать пакеты 0-RTT после начала обработки пакетов 1-RTT от сервера. Это значит, что пакеты 0-RTT не могут содержать отклики на кадры из пакетов 1-RTT. Например, клиент не может передать кадр ACK в пакете 0-RTT, поскольку этот кадр может подтверждать лишь пакет 1-RTT. Подтверждение для пакета 1-RTT **должно** передаваться в пакете 1-RTT.

Серверу **следует** считать нарушение запомненных ограничений (параграф 7.4.1) ошибкой соединения подходящего типа (например, FLOW_CONTROL_ERROR при нарушении ограничения для данных потока).

17.2.4. Пакет согласования

Пакеты Handshake используют длинный заголовок типа 0x02, за которым следуют поля Length и Packet Number (см. параграф 17.2). Первый байт содержит поля Reserved и Packet Number Length (см. параграф 17.2). Пакеты служат для передачи сообщений криптографического согласования и подтверждений от сервера и клиента.

```
Handshake Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 2,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Рисунок 17. Защищенный пакет Handshake.

После получения клиентом сообщения Handshake от сервера ин использует пакеты Handshake для отправки серверу последующих сообщений криптографического согласования и подтверждений.

Поле Destination Connection ID в пакете Handshake содержит идентификатор соединения выбранный получателем пакета, Source Connection ID - идентификатор соединения, который желает использовать отправитель (параграф 7.2).

Пакеты Handshake имеют своё пространство номеров, поэтому сервер передаёт первый пакет Handshake с номером 0.

Данные (payload) этого пакета включают кадры CRYPTO и могут включать PING, PADDING или ACK. Пакеты Handshake **могут** содержать кадры CONNECTION_CLOSE типа 0x1c. Конечная точка **должна** считать получения пакета Handshake с кадрами других типов ошибкой соединения PROTOCOL_VIOLATION.

Подобно пакетам Initial (см. параграф 17.2.2.1), данные в кадрах CRYPTO пакетов Handshake отбрасываются (и не передаются повторно), когда ключи защиты Handshake отбрасываются.

17.2.5. Пакет Retry

Как показано на рисунке 18, пакет Retry использует длинный заголовок с типом 0x03 и служит для передачи маркера проверки адреса, созданного отправителем. Пакет применяется сервером, желающим выполнить повтор (см. параграф 8.1).

Пакет Retry не содержит защищённых полей. В поле Unused помещается произвольное значение, выбранное сервером, клиент **должен** игнорировать эти биты. В дополнение к полям длинного заголовка пакет содержит указанные ниже поля.

```
Retry Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 3,
  Unused (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Retry Token (..),
  Retry Integrity Tag (128),
}
```

Рисунок 18. Пакет Retry.

Retry Token

Неанализируемый маркер, который сервер может использовать для проверки адреса клиента.

Retry Integrity Tag

Тег, определённый в параграфе 5.8 (Целостность пакета Retry) [QUIC-TLS].

17.2.5.1. Отправка пакета Retry

Сервер указывает в поле Destination Connection ID идентификатор соединения, включенный клиентом в поле Source Connection ID пакета Initial. В поле Source Connection ID сервер помещает выбранный им идентификатор соединения. В этом поле **недопустимо** указывать значение Destination Connection ID из переданного клиентом пакета. Клиент **должен** отбрасывать пакет Retry, в котором поле Source Connection ID совпадает с полем Destination Connection ID в пакете Initial от клиента. Клиент **должен** использовать значение Source Connection ID из пакета Retry в полях Destination Connection ID последующих пакетов, которые он передаёт.

Сервер **может** передавать пакеты Retry в ответ на Initial и 0-RTT. Сервер может отбрасывать или буферизовать полученные пакеты 0-RTT и может отправлять несколько пакетов Retry при получении Initial или 0-RTT. Серверу **недопустимо** передавать более одного пакета Retry в ответ на одну дейтаграмму UDP.

17.2.5.2. Обработка пакета Retry

Клиент **должен** воспринимать и обрабатывать не более одного пакета Retry для каждой попытки соединения. После приёма и обработки клиентом сообщения Initial или Retry от сервера клиент **должен** отбрасывать последующие пакеты Retry.

Клиенты **должны** отбрасывать пакеты Retry с полем Retry Integrity Tag, которое они не могут проверить (параграф 5.8 в [QUIC-TLS]). Это препятствует внедрению пакетов Retry злоумышленниками и защищает от случайно повреждённых пакетов Retry. Клиент **должен** отбрасывать пакеты с пустым (размер 0) полем Retry Token.

Клиент отвечает на Retry пакетом Initial с маркером из пакета Retry для продолжения организации соединения. Клиент помещает в поле Destination Connection ID пакета Initial значение поля Source Connection ID из пакета Retry. Смена поля Destination Connection ID ведёт к изменению ключей, применяемых для защиты пакета Initial. Клиенту **недопустимо** менять Source Connection ID, поскольку сервер может включать идентификатор соединения в логику проверки маркера (см. параграф 8.1.4).

Пакет Retry не включает номера и не может быть явно подтверждён клиентом.

17.2.5.3. Продолжение согласования после пакета Retry

Последующие пакеты Initial от клиента включают идентификатор соединения и маркер из пакета Retry. Клиент копирует поле Source Connection ID из Retry в поле Destination Connection ID и использует это значение, пока не будет получен пакет Initial с обновлённым значением (см. параграф 7.2). Значение поля Token копируется во все последующие пакеты Initial (см. параграф 8.1.2).

Помимо обновления полей Destination Connection ID и Token, на передаваемый клиентом пакет Initial распространяются те же ограничения, которые применяются к первому пакету Initial. Клиент **должен** использовать сообщение криптографического согласования, включённое в этот пакет. Сервер **может** считать пакет с другим сообщением криптографического согласования ошибкой соединения или отбрасывать его. Отметим, что включение поля Token сокращает пространство, доступное для сообщения криптографического согласования, что может потребовать от клиента передачи нескольких пакетов Initial.

Клиент может попытаться использовать 0-RTT после приёма сообщения Retry, передавая пакеты 0-RTT с полученным от сервера идентификатором соединения.

Клиенту **недопустимо** сбрасывать номер пакета в каком-либо из пространство номеров после обработки пакета Retry. В частности, пакеты 0-RTT содержат конфиденциальные данные, которые, скорей всего, будут передаваться повторно после получения пакета Retry. Ключи, применяемые для этих новых пакетов 0-RTT, не будут меняться в результате отклика на пакет Retry. Однако данные в этих пакетах могут отличаться от переданных ранее. Отправка новых пакетов с теми же порядковыми номерами может ставить под угрозу защиту пакетов, поскольку для защиты другого содержимого могут использоваться те же ключ и nonce. Сервер **может** прервать соединение при обнаружении сброса порядкового номера клиентом.

Идентификаторы соединения в пакетах Initial и Retry, которыми обмениваются клиент и сервер, копируются в транспортные параметры и проверяются, как описано в параграфе 7.3.

17.3. Пакеты с коротким заголовком

В этой версии QUIC определён единственный пакет, использующий короткий заголовок.

17.3.1. Пакет 1-RTT

Пакеты 1-RTT использует короткий и передаются заголовок после согласования версии и ключей 1-RTT.

```
1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),
  Key Phase (1),
  Packet Number Length (2),
  Destination Connection ID (0..160),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Рисунок 19. Пакет 1-RTT.

Header Form

Старший бит (0x80) байта 0 сброшен (0) для короткого заголовка.

Fixed Bit

Следующий бит (0x40) байта 0 имеет значение 1. Пакеты, в которых этот бит сброшен, с данной версии протокола являются недействительными и **должны** отбрасываться. Установка этого бита (1) позволяет использовать QUIC вместе с другими протоколами [RFC7983].

Spin Bit

Третий бит (0x20) байта 0 является битом задержки и устанавливается в соответствии с параграфом 17.4.

Reserved Bits

Два следующих бита (маска 0x18) байта 0 являются резервными. Эти биты охватываются защитой заголовка (см. параграф 5.4 в [QUIC-TLS]). До применения защиты в поле **должно** быть установлено значение 0. Конечная точка **должна** считать отличное от 0 значение этого поля после снятия защиты ошибкой соединения типа PROTOCOL_VIOLATION. Отбрасывание таких пакетов после снятия защиты заголовка может делать конечную точку уязвимой для атак (см. параграф 9.5 в [QUIC-TLS]).

Key Phase

Следующий бит (0x04) байта 0 указывает фазу ключа, которая позволяет получателю пакета идентифицировать ключи, использованные для защиты пакета [QUIC-TLS]. Этот бит охватывается защитой заголовка (см. параграф 5.4 в [QUIC-TLS]).

Packet Number Length

Два младших бита (маска 0x03) байта 0 указывают размер поля Packet Number, представленный 2-битовым целым число без знака на 1 меньше фактического размера поля Packet Number в байтах. Т. е. размер поля Packet Number определяется добавлением 1 к значению этого поля. Эти биты охватывается защитой заголовка (см. параграф 5.4 в [QUIC-TLS]).

Destination Connection ID

Идентификатор соединения, выбранный предполагаемым получателем пакета (см. параграф 5.1).

Packet Number

Поле размером от 1 до 4 Байтов, охватываемое защитой заголовка (см. параграф 5.4 в [QUIC-TLS]). Размер поля Packet Number представлен в поле Packet Number Length байта 0 (см. параграф 17.1).

Packet Payload

Пакеты 1-RTT всегда включают данные, защищённые с использованием режима 1-RTT.

Бит формы заголовка и поле Destination Connection ID в коротком заголовке не зависят от версии. Остальные поля первого байта зависят от версии протокола. В [QUIC-INVARIANTS] описана интерпретация пакетов разных версий QUIC.

17.4. Spin Bit для задержки

Поле Spin Bit, определённое для пакетов 1-RTT (параграф 17.3.1), позволяет вести пассивных мониторинг задержки из точек наблюдения на пути через сеть в течение соединения. Сервер возвращает полученное значение бита, а клиент «поворачивает» (spin) его после одного RTT. Находящиеся на пути наблюдатели могут измерить время между двумя переключениями бита для оценки сквозного значения RTT в соединении.

Spin Bit применяется лишь в пакетах 1-RTT, поскольку можно измерить начальное значение RTT в соединении, наблюдая за согласованием (handshake). Поэтому бит становится доступным по завершении согласования версии и организации соединения. Измерение в пути и использование бита дополнительно рассматриваются в [QUIC-MANAGEABILITY].

Spin Bit является **необязательным в этой версии** QUIC. Конечная точка, не поддерживающая это свойство, **должна** отключить его, как описано ниже. Каждая из конечных точек в одностороннем порядке управляет включением этого бита для соединения. Реализации **должны** позволять администраторам на стороне клиента и сервера включать или отключать Spin Bit глобально или на уровне соединения. Деже в случаях, когда Spin Bit не отключён администратором, конечная точка **должна** отключать его использование для по меньшей мере для одного случайного выбранного из каждых 16 путей через сеть или 16 идентификаторов соединения, чтобы обеспечить присутствие в сети соединения QUIC с отключённым Spin Bit. Поскольку каждая конечная точка отключает свойство независимо, это обеспечивает отключение сигнала Spin Bit примерно на каждом восьмом пути через сеть.

При отключённом Spin Bit конечная точка **может** установить для бита любое значение и **должна** игнорировать входящее значение. **Рекомендуется** устанавливать для Spin Bit случайное значение независимо для каждого пакета или идентификатора соединения.

Если Spin Bit включён для соединения, конечная точка поддерживает значение для каждого пути через сеть и устанавливает бит в заголовках пакетов в соответствии с текущим сохраненным значением, когда по этому пути отправляется пакет 1-RTT. Spin Bit инициализируется значением 0 в конечной точке для каждого пути через сеть. Каждая конечная точка также запоминает наибольший номер пакета, увиденный её партнёром на каждом пути.

Когда сервер получает пакет 1-RTT с большим порядковым номером, чем видел сервер до этого на данном пути через сеть, он устанавливает для пути значение Spin Bit из полученного пакета. Когда клиент получает пакет 1-RTT, увеличивающий порядковый номер, видимый клиентом в пакетах от сервера для данного пути, он устанавливает для этого пути инвертированное значение Spin Bit из полученного пакета.

Конечная точка сбрасывает в 0 значение Spin Bit для пути при смене идентификатора соединения на этом пути.

18. Кодирование транспортных параметров

Поле extension_data расширения quic_transport_parameters, определённое в [QUIC-TLS], содержит транспортные параметры QUIC, кодируемые в виде последовательности, как показано на рисунке 20.

```
Transport Parameters {
  Transport Parameter (..) ...,
}
```

Рисунок 20. Последовательность транспортных параметров.

Каждый параметр представляется триплетом (идентификатор, размер, значение), как показано на рисунке 21

```
Transport Parameter {
  Transport Parameter ID (i),
  Transport Parameter Length (i),
  Transport Parameter Value (..),
}
```

Рисунок 21. Кодирование транспортных параметров.

Поле Transport Parameter Length указывает размер поля Transport Parameter Value в октетах.

QUIC кодирует параметры транспорта в последовательности байтов, включаемые в криптографическое согласование.

18.1. Резервные параметры транспорта

Транспортные параметры с идентификаторами вида 31*N+27 при целочисленных N зарезервированы для выполнения требования по игнорированию неизвестных параметров. Эти параметры не имеют семантики и могут включать произвольные значения.

18.2. Определения транспортных параметров

В этом параграфе детализируются транспортные параметры, определённые в документе. Многие из перечисленных параметров транспорта имеют целочисленные значения и для них используется кодирование с переменным размером, описанное в разделе 6. Для неуказанных параметров по умолчанию принимается значение 0, если не указано иное.

original_destination_connection_id (0x00)

Значение поля Destination Connection ID из первого пакета Initial от клиента (см. параграф 7.3). Параметр передают только серверы.

max_idle_timeout (0x01)

Число миллисекунд максимального тайм-аута бездействия, представленное целочисленным значением. (параграф 10.1). Тайм-аут бездействия отключается, если обе точки укажут значение 0 или опустят этот параметр.

stateless_reset_token (0x02)

Маркер, используемый для проверки сброса без учёта состояния (см. параграф 10.3) и задаваемый последовательностью из 16 байтов. Этот параметр **недопустимо** передавать клиенту, но сервер **может** его отправлять. Не передав этот параметр, сервер не сможет использовать сброс без учёта состояния (параграф 10.3) для идентификатора соединения, заданного при согласовании.

max_udp_payload_size (0x03)

Максимальный размер данных UDP (payload), выраженный целым числом, для ограничения размера дейтаграмм UDP, которые эта конечная точка хочет получать. Дейтаграммы UDP с большим размером данных могут не обрабатываться их получателем. По умолчанию для этого параметра используется максимальный размер данных в дейтаграмме UDP (65527 байтов). Значения меньше 1200 являются недействительными. Этот параметр служит дополнительным ограничением размера дейтаграмм к значению MTU на пути, но является свойством конечной точки, а не пути (см. раздел 14). Предполагается, что это значение определяется пространством, выделенным конечной точкой для входящих пакетов.

initial_max_data (0x04)

Целое число, указывающее заданный в начале максимальный объем данных, которые могут быть переданы в соединении. Это эквивалентно передаче MAX_DATA (параграф 19.9) для соединения сразу после согласования.

initial_max_stream_data_bidi_local (0x05)

Целое число, задающее начальный предел управления потоком данных, для локально инициированных двухсторонних потоков. Этот предел относится к недавно созданным двухсторонним потокам, открытым конечной точкой, которая передала транспортный параметр. В параметрах клиента этот относится к потокам с идентификатором, где два младших бита имеют значение 0x00, в параметрах сервера - 0x01.

initial_max_stream_data_bidi_remote (0x06)

Целое число, задающее начальный предел управления потоком данных, для инициированных партнёром двухсторонних потоков. Этот предел относится к недавно созданным двухсторонним потокам, открытым конечной точкой, которая получила транспортный параметр. В параметрах клиента этот относится к потокам с идентификатором, где два младших бита имеют значение 0x01, в параметрах сервера - 0x00.

initial_max_stream_data_uni (0x07)

Целое число, задающее начальный предел управления потоком данных, для односторонних потоков. Этот предел относится к недавно созданным односторонним потокам, открытым конечной точкой, которая получила транспортный параметр. В параметрах клиента этот относится к потокам с идентификатором, где два младших бита имеют значение 0x03, в параметрах сервера - 0x02.

initial_max_streams_bidi (0x08)

Целое число, задающее начальное ограничение числа двухсторонних потоков, которые разрешено инициировать принявшей параметр конечной станции. Если параметр отсутствует или имеет значение 0, партнёр не может создавать двухсторонних поток до передачи кадра MAX_STREAMS. Установка параметра эквивалентна отправке MAX_STREAMS (параграф 19.11) соответствующего типа с тем же значением.

initial_max_streams_uni (0x09)

Целое число, задающее начальное ограничение числа односторонних потоков, которые разрешено инициировать принявшей параметр конечной станции. Если параметр отсутствует или имеет значение 0, партнёр не может

создавать двухсторонних поток до передачи кадра MAX_STREAMS. Установка параметра эквивалентна отправке MAX_STREAMS (параграф 19.11) соответствующего типа с тем же значением.

ack_delay_exponent (0x0a)

Целое число, указывающее показатель степени, используемый для декодирования поля ACK Delay в кадре ACK (параграф 19.3). По умолчанию используется значение 3 (коэффициент 8). Значения больше 20 недействительны.

max_ack_delay (0x0b)

Целое число, указывающее максимальную задержку (в миллисекундах) отправки подтверждения конечной точкой. В значении **следует** учитывать задержку, после которой получатель устанавливает сигнал тревоги. Например, если получатель установил таймер на 5 мсек а сигнал тревоги обычно задерживается на 1 мсек, следует задать max_ack_delay = 6 мсек. По умолчанию предполагается 25 мсек. Значения 2^{14} и больше недействительны.

disable_active_migration (0x0c)

Параметр запрета активного переноса включается, если конечная точка не поддерживает активный перенос соединений (раздел 9) для адреса, использованного при согласовании. Получившей этот параметр конечной точке **недопустимо** использовать новый локальный адрес при отправке по адресу, который партер использовал для согласования. Этот параметр не запрещает перенос соединения, после того, как клиент воздействовал на транспортный параметр preferred_address transport. Параметр имеет нулевой размер.

preferred_address (0x0d)

Предпочтительный адрес сервера служит для изменения адреса сервера в конце согласования, как описано в параграфе 9.6. Этот параметр передают только серверы. Сервер **может** выбрать предпочтительный адрес лишь для одного семейства, указав для другого нулевой адрес и порт (0.0.0.0:0 или [::]:0). Адрес IP указывается с сетевым порядком байтов.

Параметр preferred_address содержит адрес и порт для IPv4 и IPv6. За 4-байтовым полем IPv4 следует 2 байта поля IPv4, затем 16 байтов IPv6 Address и 2 байта IPv6 Port. После адресов и портов указывается поле Connection ID Length, задающее размер следующего за ним поля Connection ID. Последним является 16-байтовое поле Stateless Reset Token с маркером сброса без учёта состояния, связанным с идентификатором соединения. Формат транспортного параметра показан на рисунке 22.

Поля Connection ID и Stateless Reset Token относятся к дополнительному идентификатору соединения, имеющему номер 1 (см. параграф 5.1.1). Отправки этих полей с предпочтительным адресом обеспечивает наличие хотя бы одного не использованного активного идентификатора соединения при переходе клиента на предпочтительный адрес сервера. Синтаксис и семантика полей Connection ID и Stateless Reset Token в предпочтительном адресе такие же как для соответствующих полей в кадре NEW_CONNECTION_ID (параграф 19.15). Серверу, выбравшему пустой идентификатор соединения, **недопустимо** указывать предпочтительный адрес. Клиент **должен** считать нарушение этого требования ошибкой соединения типа TRANSPORT_PARAMETER_ERROR.

```
Preferred Address {
  IPv4 Address (32),
  IPv4 Port (16),
  IPv6 Address (128),
  IPv6 Port (16),
  Connection ID Length (8),
  Connection ID (..),
  Stateless Reset Token (128),
}
```

Рисунок 22. Формат предпочтительного адреса.

active_connection_id_limit (0x0e)

Целое число, задающее максимальное количество идентификаторов соединений от партнёра, которые конечная точка желает сохранить. Это значение включает идентификатор соединения, полученный при согласовании соединения, а также полученные в транспортном параметре preferred_address и кадрах NEW_CONNECTION_ID. Значение параметра active_connection_id_limit **должно** быть не меньше 2. Конечная точка, получившая значение меньше 2, **должна** закрыть соединение с ошибкой типа TRANSPORT_PARAMETER_ERROR. При отсутствии этого транспортного параметра предполагается значение 2. Если конечная точка указывает пустой идентификатор соединения, она не будет передавать кадров NEW_CONNECTION_ID, поэтому игнорирует полученное от партнёра значение active_connection_id_limit.

initial_source_connection_id (0x0f)

Значение, указанное конечной точкой в поле Source Connection ID первого пакета Initial в соединении (параграф 7.3).

retry_source_connection_id (0x10)

Значение, указанное сервером в поле Source Connection ID пакета Retry (параграф 7.3). Этот параметр передаёт только сервер.

При наличии транспортных параметров, устанавливающих начальные пределы для управления потоком данных на уровне потока (initial_max_stream_data_bidi_local, initial_max_stream_data_bidi_remote, initial_max_stream_data_uni), они эквивалентны отправке кадра MAX_STREAM_DATA (параграф 19.10) в каждом потоке соответствующего типа сразу после создания. При отсутствии транспортного параметра потоки данного типа запускаются с пределом управления потоком данных 0.

Клиенту **недопустимо** включать какие-либо транспортные параметры, предназначенные лишь для серверов (original_destination_connection_id, preferred_address, retry_source_connection_id, stateless_reset_token). Сервер **должен** считать получение такого транспортного параметра ошибкой соединения типа TRANSPORT_PARAMETER_ERROR.

19. Типы и форматы кадров

Как указано в параграфе 12.4, пакет содержит 1 или несколько кадров. В этом разделе описан формат и семантика основных типов кадров QUIC.

19.1. Кадр PADDING

Кадр PADDING (тип 0x00) не имеет семантического значения и служит для увеличения размера пакета. Заполнение может применяться для увеличения пакетов Initial до минимально требуемого размера или для защиты от анализа трафика защищённых пакетов.

Формат кадра PADDING показан на рисунке 23, из которого видно отсутствие в кадре содержимого. Т. е. кадр PADDING содержит лишь байт идентификатора типа.

```
PADDING Frame {
  Type (i) = 0x00,
}
```

Рисунок 23. Формат кадра PADDING.

19.2. Кадр PING

Конечные точки могут применять кадры PING (тип 0x01) для проверки живучести партнёра или его достижимости.

Формат кадра PING показан на рисунке 24, из которого видно отсутствие в кадре содержимого.

```
PING Frame {
  Type (i) = 0x01,
}
```

Рисунок 24. Формат кадра PING.

Получателю кадра PING нужно просто подтвердить пакет, содержащий кадр. Кадры PING можно использовать для сохранения активности в соединении, когда приложение или прикладной протокол желают предотвратить тайм-аут (см. параграф 10.1.2).

19.3. Кадр ACK

Получатели передают кадры ACK (тип 0x02 и 0x03) для информирования отправителя о приёме и обработке его пакетов. Кадр ACK включает один или несколько диапазонов ACK Range, указывающих подтверждаемые пакеты. Кадры ACK типа 0x03 содержат также кумулятивный счётчик пакетов QUIC со связанной маркировкой ECN, полученных через соединение к моменту отправки кадра. Реализации QUIC **должны** обрабатывать оба типа кадров, а при поддержке ECN для передаваемых пакетов им также **следует** использовать сведения из раздела ECN для поддержки статуса перегрузки.

Подтверждения QUIC являются безотзывными, т. е. после подтверждения пакет считается доставленным, даже если он не указан в будущем кадре ACK. Это отличается от отказа для селективных подтверждений TCP (Selective Acknowledgment или SACK) [RFC2018].

Пакеты из разных пространств номеров могут идентифицироваться одним числовым значением. В подтверждении пакета требуется указывать порядковый номер и пространство номеров. Это достигается указанием в каждом кадре ACK номеров пакетов лишь из одного пространства с пространством номеров пакета, содержащего кадр ACK.

Пакеты Version Negotiation и Retry не могут подтверждаться, поскольку у них нет порядкового номера. Вместо кадров ACK для них используются неявные подтверждения в следующем пакете Initial, переданном клиентом.

Формат кадра ACK показан на рисунке 25.

```
ACK Frame {
  Type (i) = 0x02..0x03,
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) ...,
  [ECN Counts (..)],
}
```

Рисунок 25. Формат кадра ACK.

Largest Acknowledged
: Целое число с переменным размером, указывающее наибольший порядковый номер, подтверждённый партнёром. Обычно это наибольший номер пакета, который партнёр принял перед генерацией кадра ACK. В отличие от номера пакета в длинном или коротком заголовке QUIC значение в кадре ACK не отсекается.

ACK Delay
: Целое число с переменным размером, указывающее задержку подтверждения в миллисекундах (параграф 13.2.5). Значение декодируется путём умножения поля на 2 в степени, заданной транспортным параметром ack_delay_exponent, переданным отправителем кадра ACK (параграф 18.2). По сравнению с простым заданием задержки целым числом такое кодирование позволяет указать больший диапазон тем же числом битов за счёт снижения точности.

ACK Range Count
: Целое число с переменным размером, указывающее количество полей ACK Range в кадре.

First ACK Range
: Целое число с переменным размером, указывающее количество непрерывных пакетов, предшествующих подтверждаемому значению Largest Acknowledged. Таким образом, меньший номер, подтверждаемый диапазоном, определяется вычитанием First ACK Range из Largest Acknowledged.

ACK Ranges
: Дополнительные диапазоны пропущенных (Gap) и подтверждённых (ACK Range) пакетов (параграф 19.3.1).

ECN Counts
: Три значения счётчиков ECN (см. параграф 19.3.2).

19.3.1. Диапазоны ACK

Каждое поле ACK Range состоит из чередующихся значений Gap и ACK Range Length в порядке уменьшения номеров пакетов. Поля ACK Range могут повторяться. Число значений Gap и ACK Range Length определяется полем ACK Range Count. Структура ACK Range показана на рисунке 26.

```
ACK Range {
  Gap (i),
  ACK Range Length (i),
}
```

Рисунок 26. Диапазоны ACK.

Gap

Целое число переменного размера, указывающее количество последовательных неподтвержденных пакетов перед пакетом с меньшим на 1 номером по сравнению с меньшим из предшествующего ACK Range.

ACK Range Length

Целое число переменного размера, указывающее количество последовательных подтверждённых пакетов перед пакетом с наибольшим номером из предшествующего Gap.

В полях Gap и ACK Range Length используется относительное кодирование целых чисел для эффективности. Хотя каждое значение является положительным, значения вычитаются так, что в каждом ACK Range описывается диапазон с меньшими номерами. Каждое поле ACK Range подтверждает непрерывный диапазон пакетов, указывая число подтверждённых пакетов, предшествующих большему значению диапазона. Значение 0 указывает, что подтверждается лишь пакет с наибольшим номером. Большие значения ACK Range указывают больший диапазон с соответствующим меньшим значением для наименьшего номера пакета в диапазоне. Таким образом, по большему номеру пакета в диапазоне (largest) меньший (smallest) определяется как smallest = largest - ack_range.

ACK Range подтверждает все пакеты от меньшего до большего номера, включительно. Наибольшее значение для ACK Range определяется путём последовательного вычитания размера всех предшествующих полей ACK Range Length и Gap. Каждое значение Gap указывает диапазон пакетов, которые не подтверждаются. Число пропущенных пакетов на 1 больше значения поля Gap. Поле Gap задаёт наибольший номер пакета для последующего ACK Range выражением largest = previous_smallest - gap - 2. Если какой-либо из рассчитанных номеров имеет отрицательное значение, конечная точка должна генерировать ошибку типа FRAME_ENCODING_ERROR.

19.3.2. ECN Counts

Кадры ACK используют младший бит типа (0x03) для индикации обратной связи ECN и информирования о получении пакетов QUIC с кодами ECN ECT(0), ECT(1), ECN-CE в заголовке IP. Поле ECN присутствует лишь в кадрах ACK типа 0x03. Формат поля показан на рисунке 27.

```
ECN Counts {
  ECT0 Count (i),
  ECT1 Count (i),
  ECN-CE Count (i),
}
```

Рисунок 27. Формат ECN Counts.

ECT0 Count

Целое число переменного размера - количество пакетов с кодом ECT(0) в пространстве номеров кадра ACK.

ECT1 Count

Целое число переменного размера - количество пакетов с кодом ECT(1) в пространстве номеров кадра ACK.

ECN-CE Count

Целое число переменного размера - количество пакетов с кодом ECN-CE в пространстве номеров кадра ACK.

Счётчики ECN поддерживаются независимо в каждом пространстве порядковых номеров пакетов.

19.4. Кадр RESET_STREAM

Конечная точка применяет кадр RESET_STREAM (тип 0x04) для внезапной остановки передающей части потока. После отправки RESET_STREAM конечная точка прекращает передачу кадров STREAM в указанном потоке. Получатель RESET_STREAM может отбросить любые данные, уже принятые в этом потоке. Принявшая RESET_STREAM конечная точка для потока, который лишь передаёт (send-only), **должна** прервать соединение с ошибкой STREAM_STATE_ERROR. Формат кадра RESET_STREAM показан на рисунке 28.

```
RESET_STREAM Frame {
  Type (i) = 0x04,
  Stream ID (i),
  Application Protocol Error Code (i),
  Final Size (i),
}
```

Рисунок 28. Формат кадра RESET_STREAM.

Stream ID

Целое число переменного размера, представляющее идентификатор прерываемого потока.

Application Protocol Error Code

Целое число переменного размера, указывающее код ошибки прикладного протокола (см. параграф 20.2), задающий причину закрытия потока.

Final Size

Целое число переменного размера, указывающее конечный размер потока у отправителя RESET_STREAM в байтах (см. параграф 4.5).

19.5. Кадр STOP_SENDING

Конечная точка использует кадр STOP_SENDING (тип 0x05) для информирования о том, что входящие данные отбрасываются при получении по запросу приложения. STOP_SENDING запрашивает у партнёра прекращение передачи в поток. Кадр STOP_SENDING можно передать для потоков в состоянии Recv или Size Known 9см. параграф 3.2). Получение STOP_SENDING для локально инициированного потока, который ещё не создан, **должно** считаться ошибкой соединения типа STREAM_STATE_ERROR. Конечная точка, получившая STOP_SENDING для потока, который только принимает (receive-only), **должна** прервать соединение с ошибкой STREAM_STATE_ERROR. Формат кадра STOP_SENDING показан на рисунке 29.

```
STOP_SENDING Frame {
  Type (i) = 0x05,
  Stream ID (i),
  Application Protocol Error Code (i),
}
```

Рисунок 29. Формат кадра STOP_SENDING.

Stream ID

Целое число переменного размера, указывающее идентификатор игнорируемого потока.

Application Protocol Error Code

Целое число переменного размера с заданным приложением кодом причины игнорирования потока отправителем (см. параграф 20.2).

19.6. Кадр CRYPTO

Кадр CRYPTO (тип 0x06) служит для передачи сообщений криптографического согласования и может передаваться во всех типах пакетов кроме 0-RTT. Кадр CRYPTO предлагает криптографическому протоколу упорядоченный поток байтов. Функционально кадры CRYPTO идентичны STREAM, но они не включают идентификатор потока, для них не применяется управления потоком данных и в кадрах нет маркеров смещения, размера и конца потока. Формат CRYPTO показан на рисунке 30.

```
CRYPTO Frame {
  Type (i) = 0x06,
  Offset (i),
  Length (i),
  Crypto Data (..),
}
```

Рисунок 30. Формат кадра CRYPTO.

Offset

Целое число переменного размера - байтовое смещение в потоке для данных в этом кадре CRYPTO.

Length

Целое число переменного размера, указывающее размер поля Crypto Data в этом кадре CRYPTO.

Crypto Data

данные криптографического сообщения.

На каждом уровне шифрования имеется отдельный поток данных криптографического согласования, начинающийся со смещения 0. Это означает обработку каждого уровня шифрования как отдельного потока данных CRYPTO.

Наибольшее смещение данных, доставляемых в потоке (сумма смещения и размера данных) не может превышать 2^{62}-1. Получение кадра сверх этого предела **должно** считаться ошибкой соединения типа FRAME_ENCODING_ERROR или CRYPTO_BUFFER_EXCEEDED.

В отличие от кадров STREAM, включающих идентификатор потока, кадры CRYPTO содержат данные для одного потока на уровень шифрования. Поток не имеет явного завершения, поэтому в кадрах CRYPTO не передаётся бит FIN.

19.7. Кадр NEW_TOKEN

Сервер передаёт кадр NEW_TOKEN (тип 0x07) для предоставления клиенту маркера, который тот передаёт в заголовке пакета Initial для будущего соединения. Формат кадра NEW_TOKEN показан на рисунке 31.

```
NEW_TOKEN Frame {
  Type (i) = 0x07,
  Token Length (i),
  Token (..),
}
```

Рисунок 31. Формат кадра NEW_TOKEN.

Token Length

Целое число переменного размера, указывающее размер маркера в байтах.

Token

Необрабатываемый блок данных (blob), который клиент может указать в будущем пакете Initial. Пустой маркер не допустим и клиент должен считать получение кадра NEW_TOKEN с пустым полем Token ошибкой соединения типа FRAME_ENCODING_ERROR.

Клиент может получить несколько кадров NEW_TOKEN с одним значением маркера, если для таких пакетов была некорректно определена потеря. Клиенты отвечают за отбрасывание дубликатов, которым можно воспользоваться для сопоставления попыток соединения (см. параграф 8.1.3). Клиентам **недопустимо** передавать кадры NEW_TOKEN и сервер **должен** считать получение кадра NEW_TOKEN ошибкой соединения типа PROTOCOL_VIOLATION.

19.8. Кадр STREAM

Кадры STREAM неявно создают поток и переносят его данные. Поле Type в кадре STREAM принимает форму 0b00001XXX (или набор значений от 0x08 до 0x0f). Три младших бита типа определяют присутствующие в кадре поля.

- Бит OFF (0x04) в типе кадра указывает наличие поля Offset. При установленном (1) биту поле Offset присутствует, а при сброшенном его нет и данные потока (Stream Data) начинаются со смещения 0 (т. е. кадр содержит первые байты потока или конец пустого потока).
- Бит LEN (0x02) в типе кадра указывает наличие поля Length. Сброшенный (0) бит говорит, что поля Length нет и поле Stream Data продолжается до конца потока. Установленный (1) бит указывает наличе поля Length.
- Бит FIN (0x01) указывает, что кадр содержит конец потока. Окончательный размер потока определяет сумма смещения и размера этого кадра.

Конечная точка **должна** прерывать соединение с ошибкой STREAM_STATE_ERROR при получении кадра STREAM для локально инициированного потока, который ещё не создан, или только передающего потока (send-only). Формат кадра STREAM показан на рисунке 32.

```
STREAM Frame {
  Type (i) = 0x08..0x0f,
  Stream ID (i),
  [Offset (i)],
  [Length (i)],
  Stream Data (..),
}
```

Рисунок 32. Формат кадра STREAM.

Stream ID

Целое число переменного размера - идентификатор потока (см. параграф 2.1).

Offset

Целое число переменного размера - смещение в потоке данных этого кадра STREAM. Поле присутствует при установленном бите OFF. При отсутствии поля Offset смещение принимается равным 0.

Length

Целое число переменного размера, указывающее размер поля Stream Data в этом кадре STREAM. Поле присутствует при установленном бите LEN. Если LEN = 0, поле Stream Data занимает оставшиеся байты пакета.

Stream Data

Байты указанного потока для доставки.

Когда поле Stream Data имеет размер 0, смещение в кадре STREAM указывает следующий байт, который будет передан. Первый байт потока имеет смещение 0. Наибольшее смещение в потоке (сумма смещения и размера данных) не может быть больше 2^{62}-1, поскольку невозможно предоставить кредит управления потоком данных для таких данных. Получение кадра с превышением этого предела **должно** считаться ошибкой соединения типа FRAME_ENCODING_ERROR или FLOW_CONTROL_ERROR.

19.9. Кадр MAX_DATA

Кадр MAX_DATA (тип 0x10) применяется в управлении потоком данных для указания партнёру максимального объёма данных, которые он может передать в соединении. Формат кадра MAX_DATA показан на рисунке 33.

```
MAX_DATA Frame {
  Type (i) = 0x10,
  Maximum Data (i),
}
```

Рисунок 33. Формат кадра MAX_DATA.

Maximum Data

Целое число переменного размера - максимальный объем данных (в байтах) для передачи через соединение. Учитываются все данные, переданные в кадрах STREAM. Сумме окончательных размеров всех потоков, включая потоки в завершающих состояниях, **недопустимо** превышать значение, анонсированное получателем. Конечная точка **должна** прерывать соединение с ошибкой FLOW_CONTROL_ERROR при получении данных сверх заданного предела с учётом нарушения запомненных пределов в Early Data (см. параграф 7.4.1).

19.10. Кадр MAX_STREAM_DATA

Кадр MAX_STREAM_DATA (тип 0x11) применяется в управлении потоком данных для указания партнёру максимального объёма данных, которые он может передать в потоке. Кадр MAX_STREAM_DATA можно передавать для потоков в состоянии Recv (см. параграф 3.2). Получение MAX_STREAM_DATA для локально инициированного потока, который ещё не создан, **должно** считаться ошибкой соединения типа STREAM_STATE_ERROR. Конечная точка, получившая MAX_STREAM_DATA для только принимающего (receive-only) потока, **должна** прервать соединения с ошибкой STREAM_STATE_ERROR. Формат MAX_STREAM_DATA показан на рисунке 34.

```
MAX_STREAM_DATA Frame {
  Type (i) = 0x11,
  Stream ID (i),
  Maximum Stream Data (i),
}
```

Рисунок 34. Формат кадра MAX_STREAM_DATA.

Stream ID

Целое число переменного размера - идентификатор потока, на который воздействует кадр.

Maximum Stream Data

Целое число переменного размера - максимальный объем данных (в байтах) для передачи в указанном потоке. Конечная точка учитывает максимальное смещение полученных данных, которые были переданы или приняты в потоке. Потери или нарушение порядка могут приводить к тому, что наибольшее смещение превысит размер данных, полученных в потоке. Приме кадров STREAM может не увеличивать наибольшее смещение полученных данных. Данным, переданным в потоке, **недопустимо** превышать максимальное значение, анонсированное партнёром. Конечная точка **должна** прервать соединение с ошибкой FLOW_CONTROL_ERROR при получении данных сверх заданного предела с учётом нарушения запомненных пределов в Early Data (см. параграф 7.4.1).

19.11. Кадр MAX_STREAMS

Кадр MAX_STREAMS (тип 0x12 или 0x13) указывает партнёру общее число потоков, которые можно создать. Кадры типа 0x12 относятся к двухсторонним потокам, типа 0x13 - к односторонним. Формат кадров показан на рисунке 35.

```
MAX_STREAMS Frame {
  Type (i) = 0x12..0x13,
  Maximum Streams (i),
}
```

Рисунок 35. Формат кадра MAX_STREAMS.

Maximum Streams

Число потоков соответствующего типа, которые могут быть созданы в течение срока работы соединения. Это значение не может превышать 2^{60}, поскольку невозможно указать идентификатор потока больше 2^{62}-1. Приём кадра, разрешающего создать поток сверх заданного предела, **должен** считаться ошибкой соединения типа FRAME_ENCODING_ERROR.

Потери и нарушение порядка могут приводить к получению конечной точкой кадра MAX_STREAMS со значением меньше полученного ранее. Кадры MAX_STREAMS, не увеличивающие предел для потока, **должны** игнорироваться. Конечной точке **недопустимо** создавать больше потоков, чем указал партнёр. Например, сервер, получивший для односторонних потоков предел 3, может создать потоки 3, 7 и 11, но не 15. Конечная точка **должна** прерывать соединение с ошибкой STREAM_LIMIT_ERROR если партнёр превышает заданный предел с учётом нарушения запомненных пределов в Early Data (см. параграф 7.4.1). Отметим, что эти кадры (и соответствующие параметры транспорта) не задают число одновременных потоков. Принимаются во внимание открытые и закрытые потоки.

19.12. Кадр DATA_BLOCKED

Отправителю **следует** передать кадр DATA_BLOCKED (тип 0x14), когда он хочет передать данные, но управление потоком данных на уровне соединения не позволяет это (см. раздел 4). Кадры DATA_BLOCKED служат входными данными для настройки алгоритмов управления потоком данных (параграф 4.2). Формат кадра показан на рисунке 36.

```
DATA_BLOCKED Frame {
  Type (i) = 0x14,
  Maximum Data (i),
}
```

Рисунок 36. Формат кадра DATA_BLOCKED.

Maximum Data

Целое число переменного размера - ограничение на уровне соединения, при котором произошла блокировка.

19.13. Кадр STREAM_DATA_BLOCKED

Отправителю **следует** передать кадр STREAM_DATA_BLOCKED (тип 0x15), когда он хочет передать данные, но управление потоком данных на уровне потока не позволяет это. Этот тип кадров похож на DATA_BLOCKED (параграф 19.12). Конечная точка, получившая STREAM_DATA_BLOCKED для только передающего (send-only) потока, **должна** прервать соединение с ошибкой STREAM_STATE_ERROR. Формат кадра показан на рисунке 37.

```
STREAM_DATA_BLOCKED Frame {
  Type (i) = 0x15,
  Stream ID (i),
  Maximum Stream Data (i),
}
```

Рисунок 37. Формат кадра STREAM_DATA_BLOCKED.

Stream ID

Целое число переменного размера - идентификатор потока, заблокированного управлением потоком данных.

Maximum Stream Data

Целое число переменного размера - смещение потока, при котором произошла блокировка.

19.14. Кадр STREAMS_BLOCKED

Отправителю **следует** передать кадр STREAMS_BLOCKED (тип 0x16 или 0x17), когда он хочет создать поток, но заданное партнёром ограничение не позволяет это (см. параграф 19.11). Кадры типа 0x16 относятся к двухсторонним потокам, типа 0x17 - к односторонним. Кадр STREAMS_BLOCKED не создаёт поток, но информирует партнёра о потребности в новых потоках и достижении заданного им предела. Формат кадра показан на рисунке 38.

```
STREAMS_BLOCKED Frame {
  Type (i) = 0x16..0x17,
  Maximum Streams (i),
}
```

Рисунок 38. Формат кадра STREAMS_BLOCKED.

Maximum Streams

Целое число переменного размера, указывающее максимальное число потоков соответствующего типа на момент передачи кадра. Это значение не может превышать 2^{60}, поскольку невозможно указать идентификатор потока больше 2^{62}-1. Приём кадра, указывающего поток с большим номером, **должен** считаться ошибкой соединения типа STREAM_LIMIT_ERROR или FRAME_ENCODING_ERROR.

19.15. Кадр NEW_CONNECTION_ID

Конечная точка передаёт кадр NEW_CONNECTION_ID (тип 0x18) для предоставления партнёру дополнительных идентификаторов соединения, которые могут служить для предотвращения возможности сопоставлений при переносе соединения (см. параграф 9.5). Формат кадра NEW_CONNECTION_ID показан на рисунке 39.

Sequence Number

Целое число переменного размера - порядковый номер выделенный идентификатору соединения отправителем (параграф 5.1.1).

```
NEW_CONNECTION_ID Frame {
  Type (i) = 0x18,
  Sequence Number (i),
  Retire Prior To (i),
  Length (8),
  Connection ID (8..160),
  Stateless Reset Token (128),
}
```

Рисунок 39. Формат кадра NEW_CONNECTION_ID.

Retire Prior To

Целое число переменного размера - идентификаторы соединения, которые следует удалить (параграф 5.1.2).

Length

8-битовое целое число без знака, указывающее размер идентификатора соединения. Значения меньше 1 и больше 20 недействительны и **должны** считаться ошибкой соединения типа FRAME_ENCODING_ERROR.

Connection ID

Идентификатор соединения заданного размера.

Stateless Reset Token

128-битовое значение используемое для сброса без учёта состояния при использовании соответствующего идентификатора соединения (см. параграф 10.3).

Конечной точке **недопустимо** передавать этот кадр, если она в настоящее время требует от партнёра передачи пакетов с пустым Destination Connection ID. Замена пустого идентификатора соединения непустым и обратно осложняет определение момента смены идентификатора соединения. Конечная точка, передающая пакеты с пустым Destination Connection ID, **должна** считать получение кадра NEW_CONNECTION_ID ошибкой соединения типа PROTOCOL_VIOLATION.

Ошибки при передаче, тайм-ауты и повторы передачи могут приводить к неоднократному получению одного и того же кадра NEW_CONNECTION_ID и это **недопустимо** считать ошибкой соединения. Получатель может использовать порядковый номер из NEW_CONNECTION_ID для обработки таких ситуаций.

При получении кадра NEW_CONNECTION_ID с ранее заданным идентификатором соединения, но с иным полем Stateless Reset Token, иным значением Sequence Number или порядковым номером, использованным для другого идентификатора соединения конечная точка **может** считать это ошибкой соединения типа PROTOCOL_VIOLATION.

Поле Retire Prior To применяется к идентификаторам, созданным в процессе организации соединения, и параметру транспорта preferred_address (см. параграф 5.1.2). Значение Retire Prior To **должно** быть не больше значения поля Sequence Number. Получение Retire Prior To больше Sequence Number **должно** считаться ошибкой соединения типа FRAME_ENCODING_ERROR. После того, как отправитель указал Retire Prior To, меньшие значения в последующих кадрах NEW_CONNECTION_ID не оказывают влияния. Получатель **должен** игнорировать поля Retire Prior To, не увеличивающие полученное ранее значение.

Конечная точка, получившая кадр NEW_CONNECTION_ID с порядковым номером меньше значения поля Retire Prior To в ранее полученном NEW_CONNECTION_ID **должна** передать соответствующий кадр RETIRE_CONNECTION_ID, удаляющий полученный недавно идентификатор соединения, если это ещё не сделано для порядкового номера.

19.16. Кадр RETIRE_CONNECTION_ID

Конечная точка передаёт кадр RETIRE_CONNECTION_ID (тип 0x19) для указания того, что она больше не будет применять идентификатор соединения, выданный партнёром. Это включает идентификаторы, представленные во время согласования. передача RETIRE_CONNECTION_ID служит также запросом к партнёру на передачу дополнительных идентификаторов соединения на будущее (см. параграф 5.1). Новые идентификаторы могут быть доставлены в кадрах NEW_CONNECTION_ID (параграф 19.15). Удаление идентификатора соединения делает недействительным связанный с ним маркер сброса без учёта состояния. Формат кадра показан на рисунке 40.

```
RETIRE_CONNECTION_ID Frame {
  Type (i) = 0x19,
  Sequence Number (i),
}
```

Рисунок 40. Формат кадра RETIRE_CONNECTION_ID.

Sequence Number

Порядковый номер удаляемого идентификатора соединения (см. параграф 5.1.2).

Получение кадра RETIRE_CONNECTION_ID с порядковым номером больше переданных ранее партнёру **должно** считаться ошибкой соединения типа PROTOCOL_VIOLATION.

Порядковому номеру в кадре RETIRE_CONNECTION_ID **недопустимо** указывать поле Destination Connection ID в пакете, где содержится кадр. Партнер **может** считать это ошибкой соединения типа PROTOCOL_VIOLATION.

Конечная точка не может передавать этот кадр, если партнёр предоставил пустой идентификатор соединения. Конечная точка, предоставившая пустой идентификатор соединения, **должна** считать получение кадра RETIRE_CONNECTION_ID ошибкой соединения типа PROTOCOL_VIOLATION.

19.17. Кадр PATH_CHALLENGE

Конечная точка может использовать кадры PATH_CHALLENGE (тип 0x1a) для проверки доступности партнёра и проверки пути при переносе соединения. Формат кадра PATH_CHALLENGE показан на рисунке 41.

```
PATH_CHALLENGE Frame {
  Type (i) = 0x1a,
  Data (64),
}
```

Рисунок 41. Формат кадра PATH_CHALLENGE.

Data

8-битовое поле с произвольными данными.

Включение 64 битов энтропии в кадр PATH_CHALLENGE гарантирует, что легче принять кадр, чем угадать значение. Получатель кадра **должен** генерировать кадр PATH_RESPONSE (параграф 19.18) с тем же значением в поле Data.

19.18. Кадр PATH_RESPONSE

Кадр PATH_RESPONSE (тип 0x1b) передаётся в ответ на PATH_CHALLENGE. Формат кадра показан на рисунке 42.

```
PATH_RESPONSE Frame {
  Type (i) = 0x1b,
  Data (64),
}
```

Рисунок 42. Формат кадра PATH_RESPONSE.

Если содержимое кадра PATH_RESPONSE не соответствует ранее переданному PATH_CHALLENGE конечная точка **может** считать это ошибкой соединения типа PROTOCOL_VIOLATION.

19.19. Кадр CONNECTION_CLOSE

Конечная точка передаёт кадр CONNECTION_CLOSE (тип 0x1c или 0x1d) для уведомления партнёра о закрытии соединения. Кадр типа 0x1c используется для сигнализации об ошибках лишь на уровне QUIC или отсутствии ошибок (код NO_ERROR), а типа 0x1d - для сигнализации об ошибках приложения, использующего QUIC. При наличии потоков, не закрытых явно, они будут неявно закрыты при завершении соединения. Формат кадра показан на рисунке 43.

```
CONNECTION_CLOSE Frame {
  Type (i) = 0x1c..0x1d,
  Error Code (i),
  [Frame Type (i)],
  Reason Phrase Length (i),
  Reason Phrase (..),
}
```

Рисунок 43. Формат кадра CONNECTION_CLOSE.

Error Code

Целое число переменного размера - код причины закрытия соединения. CONNECTION_CLOSE типа 0x1c использует коды из пространства, определённого в параграфе 20.1, CONNECTION_CLOSE типа 0x1d - определённые прикладным протоколом (см. параграф 20.2).

Frame Type

Целое число переменного размера - тип вызвавшего ошибку кадра. Значение 0 (эквивалентное указанию кадра PADDING) служит для неизвестных типов кадров. Определяемый приложением кадр CONNECTION_CLOSE (тип 0x1d) не использует это поле.

Reason Phrase Length

Целое число переменного размера, указывающее размер объяснения причины в байтах. Поскольку кадр CONNECTION_CLOSE нельзя разделить между пакетами, ограничения на размер пакетов влияют на это поле.

Reason Phrase

Диагностические сведения о причине закрытия. Поле может быть пустым, если отправитель счёл нужным предоставить лишь значение Error Code. В поле **следует** указывать строку в кодировке UTF-8, хотя в кадре нет информации (такой, как тег языка).

Определяемый приложением вариант CONNECTION_CLOSE (тип 0x1d) может передаваться лишь в кадрах 0-RTT или 1-RTT (см. параграф 12.5). При желании приложения прервать соединение в процессе согласования конечная точка может передать CONNECTION_CLOSE типа 0x1c с кодом APPLICATION_ERROR в пакете Initial или Handshake.

19.20. Кадр HANDSHAKE_DONE

Сервер использует кадр HANDSHAKE_DONE (тип 0x1e) для подтверждения согласования клиенту. Кадр HANDSHAKE_DONE не имеет содержимого, как показано на рисунке 44.

```
HANDSHAKE_DONE Frame {
  Type (i) = 0x1e,
}
```

Рисунок 44. Формат кадра HANDSHAKE_DONE.

Кадр HANDSHAKE_DONE может передавать только сервер. Серверам **недопустимо** передавать HANDSHAKE_DONE до завершения согласования. Приём кадра HANDSHAKE_DONE сервером **должен** считаться ошибкой соединения типа PROTOCOL_VIOLATION.

19.21. Кадры расширения

Кодирование кадров QUIC не описывает себя, поэтому конечной точке нужно понимать синтаксис всех кадров для успешной обработки пакетов. Это позволит эффективно декодировать кадры, но конечная точка не сможет передать кадры, которые неизвестны партнёру. Расширение QUIC, желающее применять кадры нового типа, **должно** сначала обеспечить понимание этого типа партнёрами. Конечная точка может использовать транспортный параметр для индикации поддержки расширенных типов.

Расширения, меняющие или замещающие базовую функциональность протокола (включая типы кадров), сложно объединить с другими расширениями, которые меняют или замещают те же функции, пока поведение такой комбинации не задано явно. Таким расширениям **следует** определять их взаимодействие с другими расширениями, меняющими те же компоненты протокола. Для кадров расширения **должен** обеспечиваться контроль перегрузки и они **должны** вызывать передачу кадров ACK. Исключением являются расширения кадров, дополняющие или заменяющие кадры ACK. Кадры расширения не включаются в управление потоком данных, пока они не указаны в расширении.

Для управления назначением новых типов кадров служит реестр IANA, описанный в параграфе 22.4.

20. Коды ошибок

Коды транспортных ошибок QUIC и ошибок приложение представляются 62-битовыми целыми числами без знака.

20.1. Коды транспортных ошибок

В этом разделе приведён список ошибок транспорта QUIC, которые могут указываться в кадрах CONNECTION_CLOSE типа 0x1c. Эти ошибки относятся к соединению в целом.

NO_ERROR (0x00)

Конечная точка использует этот код с CONNECTION_CLOSE для информирования о внезапном закрытии соединения без ошибок.

INTERNAL_ERROR (0x01)

Внутренняя ошибка конечной точки, препятствующая работе соединения.

CONNECTION_REFUSED (0x02)
Сервер отверг новое соединение.

FLOW_CONTROL_ERROR (0x03)
Конечная точка получила больше данных, чем задано анонсированным пределом (см. раздел 4).

STREAM_LIMIT_ERROR (0x04)
Конечная точка получила кадр для идентификатора потока, выходящего за пределы ограничения для соответствующего типа потоков.

STREAM_STATE_ERROR (0x05)
Конечная точка получила кадр для потока, находящегося в состоянии, для которого такой кадр не разрешён (см. раздел 3).

FINAL_SIZE_ERROR (0x06)
(1) Конечная точка получила кадр STREAM с данными, выходящими за пределы ранее согласованного общего размера. (2) Конечная точка получила кадр STREAM или RESET_STREAM, содержащий общий размер, который меньше объёма уде принятых в потоке данных. (3) Конечная точка получила кадр STREAM или RESET_STREAM, содержащий общий размер данных, отличающийся от уже согласованного.

FRAME_ENCODING_ERROR (0x07)
Конечная точка получила кадр с ошибкой формата, например, кадр неизвестного типа или кадр ACK, в котором диапазон подтверждения больше, нежели может перенести остальная часть пакета.

TRANSPORT_PARAMETER_ERROR (0x08)
Конечная точка получила транспортные параметры с ошибочным форматом, включая недействительное значение, пропуск обязательного параметра, запрещённый транспортный параметр, или вызвавшие иную ошибку.

CONNECTION_ID_LIMIT_ERROR (0x09)
Число представленных партнёром идентификаторов соединения превышает анонсированное значение active_connection_id_limit.

PROTOCOL_VIOLATION (0x0a)
Конечная точка обнаружила протокольную ошибку, для которой не задано кода.

INVALID_TOKEN (0x0b)
Сервер получил от клиента пакет Initial с недействительным полем Token.

APPLICATION_ERROR (0x0c)
Закрытие соединения, вызванное приложением или прикладным протоколом.

CRYPTO_BUFFER_EXCEEDED (0x0d)
Конечная точка получила в кадрах CRYPTO объем данных, который она не может буферизовать.

KEY_UPDATE_ERROR (0x0e)
Конечная точка обнаружила ошибку при обновлении ключей (см. раздел 6 в [QUIC-TLS]).

AEAD_LIMIT_REACHED (0x0f)
Конечная точка достигла предела защиты конфиденциальности или целостности используемого в соединении алгоритма AEAD.

NO_VIABLE_PATH (0x10)
Конечная точка определила, что путь через сеть не может поддерживать QUIC. Получение конечной точкой кадра CONNECTION_CLOSE с таким кодом маловероятно за исключением случая, когда путь не имеет достаточного MTU.

CRYPTO_ERROR (0x0100-0x01ff)
Отказ при криптографическом согласовании. Диапазон из 256 значений зарезервирован для кодов ошибок используемого криптографического согласования. Коды ошибок при криптографическом согласовании TLS приведены а параграфе [QUIC-TLS].

Регистрация новых кодов ошибок рассмотрена в параграфе 22.5.

При определении кодов применялось несколько принципов. Для ошибочных состояний, которые могут требовать конкретного действия на стороне получателя, выделены уникальные коды. Для ошибок, представляющих общие условия, заданы конкретные коды. При отсутствии какого-либо из этих условий коды ошибок служат для идентификации базовой функции стека, такой как управление потоком данных или обработка параметров транспорта. Ошибки общего типа указывают случаи, когда реализация не может или не хочет указать конкретную ошибку.

20.2. Коды ошибок прикладного протокола

Поддержка кодов ошибок прикладного протокола оставлена этим протоколам. Коды ошибок прикладного протокола используются в кадрах RESET_STREAM (параграф 19.4), STOP_SENDING (параграф 19.5) и CONNECTION_CLOSE типа 0x1d (параграф 19.19).

21. Вопросы безопасности

Целью QUIC является обеспечение защищённых транспортных соединений. В параграфе 21.1 представлен обзор свойств защиты, а далее обсуждаются ограничения и предостережения, связанные с этими свойствами, включая описания известных атак и мер противодействия.

21.1. Обзор защитных свойств

Полный анализ безопасности QUIC выходит за рамки этого документа. В этом параграфе приведено неформальное описание свойств защиты в помощь разработчикам при анализе протокола.

QUIC предполагает модель угроз, описанную в [SEC-CONS], и обеспечивает защиту от соответствующих модели атак. Атаки делятся на активные и пассивные. В пассивных атаках у злоумышленников имеется возможность считывать пакеты из сети, а в активных им доступна также запись. Однако в пассивной атаке может участвовать злоумышленник, способный менять маршрутизацию или иначе влиять на пути прохождения пакетов, составляющих соединение.

Атакующие делятся на размещённых в пути доставки пакетов и вне этого пути. Расположенные на пути злоумышленники могут читать, изменять или удалять любой наблюдаемый пакет так, что он не достигнет адресата. Атакующие извне пути могут наблюдать пакеты, но не способны воспрепятствовать их доставке. Оба типа атакующих

могут также передавать произвольные пакеты. Эта классификация отличается от принятой в параграфе 3.5 [SEC-CONS] тем, что находящийся вне пути злоумышленник может наблюдать пакеты.

Свойства согласования, защищённых пакетов и переноса соединений рассматриваются отдельно.

21.1.1. Согласование

Согласование QUIC (handshake) включает согласование TLS 1.3 и наследует криптографические свойства, описанные в Приложении E.1 к [TLS13]. Многие из свойств защиты QUIC зависят от согласования TLS, обеспечивающего эти свойства. Любая атака на TLS может влиять на QUIC.

Любая атака на согласование TLS, ставящая под угрозу секретность или уникальность сеансовых ключей или аутентификацию участвующих партнёров, влияет на предоставляемые QUIC другие гарантии безопасности, зависящие от этих ключей. Например, перенос соединения (раздел 9) зависит от эффективности защиты конфиденциальности как для согласования ключей с использованием согласования TLS, так и для защиты пакетов QUIC, чтобы избежать возможности сопоставления по путям через сеть.

Атака на целостность согласования TLS может позволить злоумышленнику повлиять на выбор прикладного протокола или версии QUIC.

В дополнение к предоставляемым TLS свойствам согласование QUIC обеспечивает некоторую защиту от DoS-атак на согласование.

21.1.1.1. Антиусиление

Для проверки того, что заявивший адрес объект способен принимать пакеты по этому адресу выполняется проверка адресов (раздел 8). Эта проверка ограничивает возможности организации атак с усилением, когда злоумышленник может наблюдать за пакетами. До проверки адреса конечные точки ограничены в возможности передавать по нему и не могут отправить по этому адресу данные, объем которых превышает троекратный размер принятых оттуда данных.

> Примечание. Предел антиусиления применяется лишь при ответе конечной точки на пакеты с непроверенного адреса. Это ограничение не используется для клиентов, организующих новое соединение или пытающихся перенести соединение.

21.1.1.2. DoS на стороне сервера

Расчёт первой отправки с сервера для полного согласования может быть затратным, включая расчёт подписи и обмен ключами. Для предотвращения DoS-атак, потребляющих ресурсы сервера, пакеты Retry обеспечивают недорогой механизм обмена маркерами, позволяющий серверу проверить IP-адрес клиента до выполнения затратных расчётов за счёт одного дополнительного кругового обхода. После успешного согласования серев может выпустить для клиента новые маркеры, позволяющие тому организовать новое соединение без дополнительных издержек.

21.1.1.3. Прерывание согласования в пути

Злоумышленник на пути или вне его может вызвать отказ согласования, подменив или разогнав пакеты Initial. После обмена действительными пакетами Initial последующие пакеты Handshake защищаются ключами Handshake и находящийся на пути злоумышленник не может вызвать сбой согласования иной, нежели отбрасывание пакетов, вынуждающее прекратить попытку. Такой атакующий может также заменить адреса в пакетах любой из сторон, вынуждая клиента или сервер видеть некорректный адрес удалённой точки. Такая атака неотличима от преобразований NAT.

21.1.1.4. Согласование параметров

Согласование целиком защищено криптографически, причём пакеты Initial шифруются зависящими от версии ключами, а Handshake и последующие пакеты - ключами, выведенными из обмена TLS. Кроме того, согласование параметров охватывается TLS и для него обеспечивается защита целостности как для обычного согласования TLS. Атакующий может видеть транспортные параметры клиента (если известна зависимая от версии затравка), но не может видеть транспортных параметров сервера и влиять на согласование параметров.

Идентификаторы соединений не шифруются, но охватываются защитой целостности во всех пакетах.

Эта версия QUIC не включает механизма согласования версий и реализации несовместимых версий просто не смогут организовать соединение.

21.1.2. Защищённые пакеты

Для защиты пакетов (параграф 12.1) применяется аутентифицированное шифрование всех пакетов, кроме Version Negotiation, хотя пакеты Initial и Retry имеют ограниченную защиту по причине использования зависимого от версии ключевого материала (см. [QUIC-TLS]). В этом параграфе рассмотрены пассивные и активные атаки на защищённые пакеты.

Атакующие на пути и вне его могут организовать пассивную атаку с сохранением наблюдаемых пакетов для последующего взлома защиты. Это возможно для любого наблюдателя любого пакета в сети.

Злоумышленник, внедряющий пакеты, не имея возможности наблюдать действительные пакеты в соединении, скорей всего не добьётся успеха, поскольку защита гарантирует возможность создания действительных пакетов лишь конечными точками, владеющими ключевым материалом, установленным в процессе согласования (см. раздел 7 и параграф 21.1.1). Точно так же активный атакующий, который наблюдает пакеты и пытается внедрить новые данные или изменить имеющиеся, не сможет создать пакеты, которые принимающая конечная точка сочтёт действительными (кроме пакетов Initial).

Атаки с подменой, где активный злоумышленник переписывает незащищённые части пакета, который он пересылает или внедряет (такие как адрес отправителя или получателя), будут оказывать влияние лишь в том случае, когда злоумышленник может пересылать пакеты исходной точке. Защита пакетов гарантирует, что возможность обработка лишь в конечной точке, выполнившей согласование, а недействительные пакеты игнорируются такими точками.

Атакующий может также изменить границы между пакетами и дейтаграммами UDP, вызывая объединение множества пакетов в одну дейтаграмму или разделение объединённых пакетов на несколько дейтаграмм. Помимо дейтаграмм с пакетами Initial, которые требуют заполнения, смена размещения пакетов в дейтаграммах не оказывает функционального влияния на соединение, хотя может изменить характеристики производительности.

21.1.3. Перенос соединения

Перенос соединения (раздел 9) позволяет конечным точка менять IP-адреса и порты на нескольких путях, используя в каждый момент один путь для передачи и приёма кадров, не являющихся зондами. Проверка пути (параграф 8.2) устанавливает, что партнёр хочет и может принимать пакеты, переданные по конкретному пути. Это помогает снизить влияние фиктивных адресов путём ограничения числа пакетов, передаваемых по такому адресу. В этом параграфе рассмотрены свойства защиты от разных типов DoS-атак при переносе соединения.

21.1.3.1. Активные атаки в пути

Атакующий, способный воспрепятствовать доставке наблюдаемого им пакета предусмотренному получателю, считается находящимся на пути. При наличии злоумышленника между клиентом и сервером, конечным точкам приходится передавать через него пакеты для соединения по данному пути. Находящийся на пути атакующим может:

- просматривать пакеты;
- изменять заголовки IP и UDP;
- внедрять свои пакеты;
- задерживать пакеты;
- менять порядок пакетов;
- отбрасывать пакеты;
- расщеплять и объединять дейтаграммы по границам пакетов.

Однако он не может менять аутентифицируемые части пакетов так, чтобы получатель воспринял пакет.

Размещённый на пути атакующий имеет возможность менять наблюдаемые пакеты, однако изменение аутентифицируемой части пакета приведёт к его отбрасыванию получателем как недействительного, поскольку данные (payload) в пакете аутентифицируются и шифруются.

QUIC пытается ограничить возможности расположенного на пути злоумышленника несколькими способами.

1. Атакующий может препятствовать использованию пути для соединения, вызывая отказ, если нет возможности организовать другой путь мимо злоумышленника. Действия атакующего могут включать отбрасывание всех пакетов, их изменение, препятствующее расшифровке, а также иные воздействия.
2. Атакующий может препятствовать переходу на новый путь, который также проходит через него, вызывая отказ при проверке нового пути.
3. Атакующий не может препятствовать переходу клиента на новый путь, не проходящий через него.
4. Злоумышленник может снизить пропускную способность, задерживая или отбрасывая пакеты.
5. Атакующие не может заставить конечную точку воспринимать пакеты, в которых он изменил аутентифицируемую часть.

21.1.3.2. Активные атаки извне пути

Злоумышленник вне пути не находится непосредственно на пути между клиентом и сервером, но может получать копии всех или части пакетов, передаваемых между ними, а также передавать копии этих пакетов любой из конечных точек. Атакующий может:

- просматривать пакеты;
- внедрять новые пакеты;
- менять порядок внедряемых пакетов.

Атакующий не способен:

- менять переданные конечными точками пакеты;
- задерживать пакеты;
- отбрасывать пакеты;
- менять порядок исходных пакетов.

Расположенный вне пути злоумышленник может изменять полученные копии пакетов и снова внедрять их в сеть, возможно с фиктивными адресами отправителя и получателя. В этом обсуждении предполагается, что атакующий может внедрять в сеть изменённые копии пакетов, которые придут к получателю раньше оригинала, наблюдаемого злоумышленником. Иными словами, атакующий может «выиграть» соперничество с легитимными пакетами между конечными точками, что может привести к игнорированию исходного пакета получателем. Предполагается также, что у злоумышленника достаточно ресурсов для воздействия на состояние NAT. В частности, он может вынудить конечную точку потерять свою привязку NAT и захватить порт для своего трафика. QUIC пытается ограничить возможности расположенного вне пути злоумышленника.

1. Атакующий может «разгонять» пакеты и пытаться стать «ограниченным» злоумышленником на пути.

2. Атакующий может имитировать успешную проверку пути для пересылаемых пакетов с адресом отправителя, указанным как злоумышленник вне пути, если он способен обеспечить лучшее соединение между клиентом и сервером.
3. Злоумышленник не может закрыть соединение после завершения согласования.
4. Злоумышленник не может вызвать перенос соединения на новый путь, если он не может наблюдать этот путь.
5. Атакующий может стать ограниченным злоумышленником на пути в процессе переноса на новый путь, который также не проходит через него.
6. Атакующий может стать ограниченным злоумышленником на пути, воздействуя на общее состояние NAT так, чтобы отправлять пакеты на сервер с того же адреса и порта, которые изначально использовал клиент.

21.1.3.3. Ограниченные активные атаки на пути

Ограниченным атакующим на пути является расположенный вне пути злоумышленник, который предлагает улучшенную маршрутизацию пакетов, дублируя и пересылая исходные пакеты между клиентом и сервером так, что копии приходят раньше оригиналов и последние отбрасываются получателем.

Ограниченный атакующий на пути отличается от обычного злоумышленника, находящегося на пути, тем, что он не находится на исходном пути между конечными точками и пакеты от отправителя по-прежнему приходят к адресату. Это означает, что будущая неспособность маршрутизировать копии быстрее прохождения оригиналов по исходному пути не будет препятствовать доставке пакетов получателю.

Ограниченный атакующий на пути может:

- просматривать пакеты;
- внедрять свои пакеты;
- менять нешифрованные заголовки пакетов;
- изменять порядок пакетов.

Однако такой злоумышленник не способен:

- задерживать пакеты так, чтобы они приходили позже пакетов по исходному пути;
- отбрасывать пакеты;
- менять аутентифицированные и шифрованные части пакета, заставляя получателя воспринимать их.

Ограниченный атакующий на пути может задерживать пакеты лишь до момента, когда исходные пакеты прибудут раньше дубликатов, а это значит, что он не может предлагать маршрутизацию с задержкой больше чем на исходном пути. Если такой злоумышленник отбрасывает копии пакетов, исходные пакеты все равно попадают к получателю. QUIC пытается ограничить возможности ограниченного атакующего на пути.

1. Атакующий не может вынудить соединение закрыться после завершения согласования.
2. Атакующий не может вынудить бездействующее соединение закрыться, если клиент первым возобновит работу.
3. Атакующий не может вынудить бездействующее соединение казаться потерянным, если сервер первым возобновит работу.

Отметим, что такие же гарантии обеспечиваются для любого NAT по тем же причинам.

21.2. DoS-атаки на согласование

Как шифрованный транспорт с аутентификацией, QUIC обеспечивает ряд средств защиты от атак типа «отказ в обслуживании» (DoS). После завершения криптографического согласования конечные точки QUIC отбрасывают большинство пакетов, которые не аутентифицированы, существенно ограничивая возможности атакующих нарушить работу имеющихся соединения.

После организации соединения конечные точки QUIC могут воспринимать некоторые неаутентифицированные пакеты ICMP (см. параграф 14.2.1), но использование таких пакетов очень ограничено. Единственный другой тип пакета, воспринимаемый конечной точкой, - это пакет сброса без учёта состояния (параграф 10.3), в котором используется маркер, сохраняемый в секрете до его применения.

В процессе создания соединения QUIC обеспечивает лишь защиту от атак извне пути через сеть. Все пакеты QUIC содержат подтверждение того, что получателю видел предыдущий пакет от партнёра.

Адрес нельзя сменить в процессе согласования, поэтому конечная точка может отбрасывать пакеты, полученные по другому пути через сеть.

Поля Source Connection ID и Destination Connection ID служат основным средством защиты от атак извне пути в процессе согласования (см. параграф 8.1). Они должны соответствовать установленным партнёром значениям. За исключением Initial и Stateless Resets, конечная точка воспринимает лишь пакеты с полем Destination Connection ID, соответствующим значению, которое конечная точка выбрала раньше. Это единственная защита для пакетов Version Negotiation.

Поле Destination Connection ID в пакете Initial клиент выбирает как непредсказуемое, что служит дополнительной цели. Пакеты криптографического согласования защищены с использованием ключа, выведенного из этого идентификатора соединения и затравки, определяемой версией QUIC. Это позволяет конечной точке использовать один процесс для аутентификации получаемых пакетов в процессе и по завершении криптографического согласования. Не прошедшие аутентификацию пакеты отбрасываются. Такая защита пакетов обеспечивает уверенность в том, что отправитель пакета видел пакет Initial и понял его.

Эта защита не эффективна против злоумышленников, способных получать пакеты QUIC до организации соединения. Такой атакующий потенциально может отправить пакеты, которые будут восприняты конечными точками QUIC. Эта версия QUIC пытается обнаруживать такие атаки, но предполагает, что конечные точки не смогут организовать соединение в таких случаях. По большей части протокол криптографического согласования [QUIC-TLS] отвечает за обнаружение подделок в процессе согласования.

Конечным точкам разрешено применять другие методы обнаружения и восстановления при вмешательстве в согласование. Недействительные пакеты можно обнаруживать и отбрасывать другими методами и данный документ не задаёт конкретный метод.

21.3. Атаки с усилением

Атакующий может получить маркер проверки адреса (раздел 8) от сервера, а затем освободить использованный для этого адрес IP. Позднее он может инициировать соединение 0-RTT с сервером, подставив тот же адрес, который фактически может принадлежать другой конечной точке (жертве). После этого атакующий потенциально может вынудить сервер передать этой жертве данные в размере начального окна перегрузки. Серверам **следует** обеспечивать смягчение таких атак, ограничивая срок действия маркеров проверки адреса (см. параграф 8.1.3).

21.4. Атаки с избыточными подтверждениями

Конечная точка, подтверждающая непринятые пакеты, может заставит контроллер перегрузок разрешить передачу со скоростью, превосходящей возможности сети. Конечная точка **может** пропускать номера для передаваемых пакетов с целью обнаружить такое поведение. Затем конечная точка может немедленно закрыть соединение с ошибкой типа PROTOCOL_VIOLATION (см. параграф 10.2).

21.5. Атаки с подменой запросов

Атака с подделкой запроса происходит, когда конечная точка вынуждает партнёра ввести запрос к жертве. Контролируемый данной точкой. Такие атаки направлены на получение злоумышленником доступа к возможностям своего партнёра, которые без этого могут быть недоступны атакующему. Для сетевых протоколов атаки с подменой запроса часто применяются для использования неявного предоставления жертвой полномочий партнёру злоумышленника на основе его местоположения в сети.

Для эффективной подделки запросов атакующему нужна возможность влиять на то, какие пакеты и куда отправляет партнёр. Если злоумышленник может нацелить уязвимую службу с контролируемым содержимым (payload), эта служба может выполнить действия, которые невольный атакующий выполнит для злоумышленника. Например, подделка межсайтовых запросов [CSRF] в Web заставляет клиента выдавать запросы, включающие cookie проверки полномочий [COOKIE], что открывает одному сайту доступ к информации и действиям, разрешённым для другого сайта.

Поскольку QUIC работает по протоколу UDP, основная проблема атак заключается в том, что злоумышленник может выбрать адрес, на который его партнёр будет передавать дейтаграммы UDP, и контролировать незащищённое содержимое пакетов. Поскольку большая часть данных, отправляемых конечными точками QUIC, защищена, это включает контроль над шифрованными данными. Атака будет успешной, если злоумышленник может вынудить партнёра передать дейтаграмму UDP хосту, который выполнит некое действие в зависимости от содержимого дейтаграммы.

В этом параграфе описаны возможные способы использования QUIC для атак с поддельными запросами. Описаны также ограниченные меры противодействия, которые могут быть реализованы конечными точками QUIC. Эти меры ослабления могут применяться в одностороннем порядке реализацией или развёртыванием QUIC без принятия каких-либо мер у потенциальных жертв. Однако этих мер может быть недостаточно, если службы на основе UD не проверяют полномочия должным образом.

Поскольку атака с переносом, описанная в параграфе 21.5.4, является достаточно мощное и для неё нет достаточных мер противодействия, реализациям серверов QUIC следует предполагать, что злоумышленники могут заставить их генерировать произвольное содержимое UDP для произвольных получателей. Серверы QUIC **не следует** развёртывать в сети без фильтрации на входе [BCP38] и с конечными точками UDP без адекватной защиты.

Хотя в общем случае невозможно гарантировать, что клиенты не будут совмещены с уязвимыми конечными точками, эта версия QUIC не позволяет серверам мигрировать для предотвращения атак с ложным переносом на клиентов. Любое будущее расширение, которое разрешит миграцию серверов, **должно** обеспечивать меры защиты от таких атак.

21.5.1. Возможности управления для конечных точек

QUIC открывает для злоумышленников некоторые возможности влияния на отправку партнёром дейтаграмм UDP:

- при организации соединения (раздел 7) сервер может указать, куда клиенту следует передавать дейтаграммы (например, путём заполнения записей DNS);
- с помощью предпочтительных адресов (параграф 9.6) сервер может выбирать, куда клиенту следует передавать дейтаграммы;
- с помощью фиктивного переноса соединения (параграф 9.3.1) клиент может использовать обмену адреса отправителя для выбора адреса, по которому сервер будет передавать последующие дейтаграммы;
- обманные пакеты могут вынудить сервер передать пакет Version Negotiation (параграф 21.5.5).

Во всех перечисленных случаях злоумышленник может вынудить партнёра передавать дейтаграммы жертве, которая может не понимать QUIC, т. е. пакеты будут передаваться до проверки адреса (см. раздел 8).

За пределами защищённой части пакета QUIC предоставляет конечной точке несколько возможностей управления содержимым дейтаграмм UDP, передаваемых партнёром. Поле Destination Connection ID предоставляет прямой контроль над байтами в ранних пакетах, передаваемых партнёром (см. параграф 5.1). Поле Token в пакетах Initial открывает серверу контроль над другими байтами в пакетах Initial (см. параграф 17.2.2).

В данной версии QUIC нет мер предотвращения косвенного контроля над защищёнными частями пакетов. Необходимо учитывать возможность конечной точки контролировать содержимое кадров, передаваемых партнёром, особенно для кадров с данными приложения, таких как STREAM. Хотя это в некоторой степени зависит от прикладного протокола, во многих вариантах использования протоколов некоторый контроль возможен. Когда у злоумышленника есть доступ к ключам защиты пакетов, он вероятно сможет угадать, как партнёр будет шифровать последующие пакеты. Для успешного контроля над содержимым дейтаграмм атакующему нужна лишь возможность предсказать номер пакета и размещение кадров в пакете с той или иной достоверностью.

В этом параграфе предполагается, что ограничение контроля над содержимым дейтаграмм неосуществимо. В последующих параграфах основное внимание уделяется ограничению возможностей подделки запросов с помощью дейтаграмм, передаваемых до проверки адреса.

21.5.2. Обманный запрос в клиентских пакетах Initial

Злоумышленник, действующий как сервер, может выбрать IP-адрес и порт, для которых он анонсирует свою доступность, поэтому пакеты Initial от клиента считаются доступными для использования в этом типе атак. Неявная проверка адреса в согласовании (handshake) гарантирует для нового соединения, что клиент не будет передавать другие типы пакетов адресату, которые не понимает QUIC или не желает воспринимать соединения QUIC.

Защита пакетов Initial (параграф 5.2 в [QUIC-TLS]) осложняет для серверов контроль над содержимым пакетов Initial, передаваемых клиентами. Выбор клиентом непредсказуемого значения Destination Connection ID не позволяет серверам контролировать какую-либо шифрованную часть пакетов Initial от клиента. Однако поле Token открыто для сервера и позволяет ему использовать клиентов для организации атак с обманными запросами. Использование маркеров, передаваемых в кадрах NEW_TOKEN (параграф 8.1.3) открывает лишь возможность подделки запроса в процессе организации соединения. Однако клиенты не обязаны использовать кадр NEW_TOKEN. Атаки с поддельными запросами на основе поля Token можно предотвратить, если клиенты передают пустое поле Token при изменении адреса сервера с момента получения кадра NEW_TOKEN. Клиенты могут избегать применения NEW_TOKEN, если адрес сервера изменился. Однако исключение поля Token может негативно влиять на производительность. Серверы могут на основе поля NEW_TOKEN разрешать передачу данных в объёме, превышающем утроенный объем принятых данных (см. параграф 8.1). В частности, это влияет на запросы клиентом дополнительных данных от сервера с использованием 0-RTT.

Передача пакета Retry (параграф 17.2.5) позволяет серверу изменить поле Token. После отправки Retry сервер может также контролировать поле Destination Connection ID в последующих пакетах Initial от клиента. Это также может открыть опосредованный контроль над шифрованным содержимым пакетов Initial. Однако передача Retry подтверждает адрес сервера, предотвращая тем самым использование последующих пакетов Initial для обманных запросов.

21.5.3. Обманный запрос с предпочтительным адресов

Серверы могут указывать предпочтительный адрес, на который клиенты переходят после подтверждения согласования (см. параграф 9.6). Поле Destination Connection ID в пакетах от клиента по предпочтительному адресу можно использовать для обманных запросов. Клиенту **недопустимо** передавать не являющиеся зондами кадры по предпочтительному адресу до его проверки (см. раздел 8). Это существенно снижает возможности контроля сервера над шифрованными частями дейтаграмм. Этот документ не предлагает дополнительных мер противодействия, связанных с использованием предпочтительных адресов, которые могут быть реализованы конечными точками. Базовые меры из параграфа 21.5.6 могут служить для смягчения атак.

21.5.4. Обманный запрос с фиктивным переносом

Клиент может представить фиктивный адрес отправителя как часть переноса соединения, чтобы вынудить сервер отправлять дейтаграммы по этому адресу. Поле Destination Connection ID во всех пакетах, которые сервер после этого будет передавать по фиктивному адресу, можно использовать для обманных запросов. Клиент может также влиять на шифрованные данные.

Сервер, который до проверки адреса отправляет по нему лишь пакеты зондирования (параграф 9.1), даст злоумышленнику лишь ограниченный контроль над шифрованной частью дейтаграмм. Однако (в частности, при перестройке NAT) это может негативно влиять на производительность. Если сервер передаёт кадры с данными приложения, атакующий сможет контролировать большую часть их содержимого.

Этот документ не предлагает дополнительных мер противодействия, которые могут быть реализованы конечными точками, сверх описанных в параграфе 21.5.6. Однако меры предотвращения подмены адресов на сетевом уровне (в частности, фильтрация на входе [BCP38]) обеспечивают защиту от атак с подменой адресов из внешней сети.

21.5.5. Обманный запрос с Version Negotiation

Клиенты, способные предоставить фиктивный адрес отправителя, могут вынудить сервер передать по этому адресу пакет Version Negotiation (параграф 17.2.1). Отсутствие ограничения размера идентификатора соединения в пакетах неизвестной версии увеличивает объем данных, контролируемых клиентом в дейтаграммах. Первый байт пакета не находится под контролем клиента, а следующие 4 байта содержат нули, но клиент может контролировать 512 байтов, начиная с пятого. Для противодействия таким атакам нет мер, сверх указанных в параграфе параграф 21.5.6. Фильтрация на входе [BCP38] будет работать.

21.5.6. Базовые меры противодействия обманным запросам

Наиболее эффективной защитой от атак с подменой запросов является добавление строгой аутентификации в уязвимые службы. Одно это не всегда попадает в зону контроля развёртывания QUIC. В этом параграфе описаны некоторые шаги, которые конечные точки могут предпринять в одностороннем порядке. Эти шаги являются дискреционными, поскольку в зависимости от обстоятельств могут препятствовать легитимному использованию.

Услуги, предоставляемые через loopback-интерфейс, часто не имеют должной аутентификации. Конечные точки **могут** предотвращать попытки соединения или переход на loopback-адрес. Конечным точкам **не следует** разрешать переход

на loopback-адрес, если те же услуги были ранее доступны на другом интерфейсе или адрес был предоставлен службой на адресе, не относящемся к loopback. Конечные точки, зависящие от этих возможностей, могут предлагать вариант отключения защиты.

Конечные точки могут считать замену адреса на link-local [RFC4291] или адрес из частного диапазона [RFC1918] вместо глобального, уникального локально [RFC4193] или публичного (non-private) адреса как возможную попытку обманного запроса. Конечные точки могут полностью отказаться от использования таких адресов, но это связано с риском нарушения легитимных вариантов работы. Конечным точкам **не следует** использовать адрес, пока нет конкретных сведений о сети, указывающих, что передача дейтаграмм по непроверенным адресам из данного диапазона безопасна.

Конечные точки могут снизить риск обманных запросов, не включая значений из кадров NEW_TOKEN в пакеты Initial или передавая лишь пробные пакеты до завершения проверки адреса. Отметим, что это не препятствует использованию для атак поля Destination Connection ID.

Предполагается, что у конечных точек нет конкретной информации о местоположении серверов, которые могут быть уязвимы для атак с подменой запросов. Однако со временем может появиться возможность идентифицировать конкретные порты UDP, являющиеся целями атак, или базовые шаблоны дейтаграмм, используемые в атаках. Конечные точки **могут** избегать отправки дейтаграмм в такие порты или не передавать дейтаграмм с такими шаблонами до проверки адреса получателя. Конечные точки **могут** отзывать идентификаторы соединения с шаблонами, использование которых заведомо связано с проблемами.

> Примечание. Изменение конечных точек для применения этих средств защиты более эффективно, чем развёртывание сетевых средств защиты, поскольку от конечных точек не требуется выполнение дополнительной обработки при передаче по непроверенным адресам.

21.6. Slowloris-атаки

Атаки, обычно называемые Slowloris [SLOWLORIS], пытаются сохранить множество открытых соединений с целевой конечной точкой как можно дольше. Такие атаки могут быть нацелены на конечную точку QUIC с минимальной активностью, требуемой для сохранения соединения. Это может быть передача небольших объёмов данных, постепенное открытие окон управления потоком данных для управления скоростью передачи или создание кадров ACK, имитирующих высокую скорость потерь. При развёртывании QUIC следует обеспечивать смягчение Slowloris-атак такими способами, как увеличение максимального числа клиентов, поддерживаемого сервером, ограничение числа соединений с одного адреса IP, ограничение на минимальную скорость обмена данные в соединении, ограничение продолжительности соединений по времени.

21.7. Атаки с фрагментацией и сборкой пакетов

Злонамеренный отправитель может осознанно не передать часть данных потока, заставляя получателя выделять ресурсы для неотправленных данных. Это может вызвать у получателя выделение непропорционального приёмного буфера в памяти и/или создание большой и неэффективной структуры данных. Атакующий также может намеренно не подтверждать пакеты в попытке заставить отправителя сохранять неподтвержденные данные потоков для повторной передачи.

Атаки на получателей ослабляются, если окно управления потоком данных соответствует доступной памяти. Однако некоторые получатели будут чрезмерно использовать память и анонсировать смещения управления потоком данных, суммарно превосходящие размер доступной памяти. Стратегия выделения избыточной памяти может повышать производительность при корректном поведении конечной точки, но делает такую точку уязвимой для атак с фрагментацией потока. При развёртывании QUIC **следует** обеспечивать смягчение атак с фрагментацией потоков. Это может быть предотвращение выделения избыточной памяти, задержка сборки кадров STREAM, реализация эвристики на основе продолжительности пропусков в сборке и комбинации этих методов.

21.8. Атака с представлением потоков

Злонамеренная конечная точка может создать большое число потоков, исчерпав состояния партнёра, а также повторять процесс для организации большого числа соединения в стиле атак SYN flood на протокол TCP. Обычно клиенты создают потоки последовательно, как описано в параграфе 2.1. Однако при запуске нескольких потоков с короткими интервалами потеря или нарушение порядка могут вызвать нарушение порядка доставки кадров STREAM. На приёмной стороне при получении потока с большим номером получатель должен создать все промежуточные потоки того же типа (см. параграф 3.2). таким образом, в новом соединении создание потока 4000000 приведёт к созданию миллиона и одного инициированного клиентом двухстороннего потока. Число активных потоков ограничено транспортными параметрами initial_max_streams_bidi и initial_max_streams_uni, которые обновляются любыми принятыми кадрами MAX_STREAMS, как описано в параграфе 4.6. При разумном выборе эти ограничения смягчают атаки с представлением потока, однако установка слишком низкого предела может повлиять на производительность приложений, ожидающих большого числа потоков.

21.9. DoS-атаки на партнёров

QUIC и TLS содержат кадры или сообщения, легитимные в определённом контексте, но ими можно злоупотреблять для принуждения партнёра расходовать ресурсы для обработки без заметного влияния на состояние соединения.

Сообщения можно также использовать для изменения или восстановления состояния незаметным способом, таким как передача с небольшим превышением ограничений контроля перегрузок. Если затраты на обработку непропорционально превышают расход пропускной способности этот может позволить вредоносному партнёру выйти за пределы возможностей обработки.

Хотя для всех сообщений имеются легитимные способы применения, реализации **следует** отслеживать расходы на обработку и считать избыток непродуктивных пакетов индикацией атаки. Конечные точки **могут** отвечать на такие события ошибкой соединения или отбрасыванием пакетов.

21.10. Атаки с явной индикацией перегрузки

Атакующий на пути может менять значения ECN в заголовке IP для воздействия на скорость передачи. В [RFC3168] рассмотрены такие манипуляции и их влияние. Ограниченный атакующий на пути может дублировать и передавать пакеты с изменёнными полями ECN для влияния на скорость передачи. Если дубликаты пакетов получатель отбрасывает, атакующему придётся состязаться с исходными пакетами для успешной атаки. Поэтому конечные точки QUIC игнорируют поле ECN в пакетах IP, если не будет успешно обработан хотя бы один пакет QUIC в этом пакете IP (см. параграф 13.4).

21.11. Предсказание Stateless Reset

Сброс без учёта состояния открывает возможность для атак, похожих на внедрение TCP reset . Такая атака возможна, если злоумышленник способен вызвать генерацию маркера сброса для соединения с выбранным идентификатором.

Если пакет может быть направлен в разные экземпляры с общим статическим ключом, например, путём смены IP-адреса или порта, злоумышленник может вынудить сервер передать сброс без учёта состояния. Для защиты от такой атаки конечные точки, имеющие общий ключ для сброса без учёта состояния (см. параграф 10.3.2), **должны** быть организованы так, чтобы пакеты с данным идентификатором соединения всегда поступали экземпляру, имеющему статус соединения, если только это соединение не утратило активность.

В более общем смысле серверам **недопустимо** генерировать сброс без учёта состояния, если соединение с соответствующим идентификатором может быть активным на любой из конечных точек, применяющих общий статический ключ.

В случае кластера с динамическим распределением нагрузки может произойти изменение конфигурации балансировщика при сохранении активным экземпляром статуса соединения. Даже если экземпляр сохраняет состояние соединения, смена маршрутизации и результирующий сброс без учёта состояния будут приводить к разрыву соединения. Если нет никаких шансов направить пакет нужному экземпляру, лучше передать сброс без учёта состояния, чем ждать тайм-аута. Однако это приемлемо лишь в тех случаях, когда атакующий не может влиять на маршрутизацию.

21.12. Понижение версии

В этом документе определены пакеты QUIC Version Negotiation (раздел 6), служащие для согласования версии QUIC между парой конечных точек. Однако документ не задаёт способ согласования данной версии с будущими. В частности, пакеты Version Negotiation не включают механизма предотвращения атак на понижение версии. Будущие версии QUIC при использовании пакетов Version Negotiation **должны** задать механизм, устойчивый к таким атакам.

21.13. Нацеливание атак через маршрутизацию

При развёртывании следует ограничивать возможность злоумышленников нацелить атаку на конкретных экземпляр сервиса. В идеале решения о маршрутизации следует принимать независимо от выбранных клиентом значений, включая адреса. После выбора экземпляра можно выбрать идентификатор соединения, чтобы последующие пакеты попадали в тот же экземпляр.

21.14. Анализ трафика

Размер пакетов QUIC может раскрывать сведения о размере их содержимого. Кадры PADDING позволяют конечным точкам возможность скрыть размер содержимого пакетов (см. параграф 19.1).

Предотвращение анализа трафика является сложной задачей и предметом активных исследований. Размер не является единственным путём утечки. Конечные точки могут раскрывать деликатные сведения и по другим побочным каналам, включая синхронизацию пакетов.

22. Взаимодействие с IANA

Этот документ создаёт несколько реестров для поддержки кодов, применяемых в QUIC и управляемых одним набором правил, описанных в параграфе 22.1.

22.1. Правила регистрации для реестров QUIC

Во всех реестрах QUIC возможна предварительная и постоянная регистрация кодов в соответствии с приведёнными ниже правилами.

22.1.1. Предварительная регистрация

Предварительная регистрация кодов предназначена для приватного использования и экспериментов с расширениями QUIC. Для предварительной регистрации нужно лишь указать значение кода и контактные данные. Однако выделенные предварительно коды могут быть заявлены повторно и назначены для иных целей.

Предварительная регистрация выполняется по процедуре Expert Review, описанной в параграфе 4.5 [RFC8126]. Назначенным экспертам рекомендуется отвергать лишь регистрации, запрашивающие слишком большую часть оставшегося пространства или первое невыделенное значение (см. параграф 22.1.2). Предварительная регистрация включат поле Date, указывающее дату последнего обновления регистрации. Запрос на обновление предварительной регистрации можно сделать без рецензии назначенных экспертов.

Все реестры QUIC включают указанные ниже поля для поддержки предварительной регистрации.

Value
: Назначенный код.

Status
: Постоянная (permanent) или предварительная (provisional) регистрация.

Specification
Ссылка на доступную публично спецификацию значения.

Date
дата последнего обновления регистрации.

Change Controller
Субъект, ответственный за определение регистрации.

Contact
Контактные данные регистрирующего лица.

Notes
Дополнительные примечания для регистрации.

В предварительных регистрациях поля Specification и Notes **могут** быть опущены, равно как дополнительные поля, требуемые для постоянной регистрации. Поле Date не требуется в запросе на регистрацию и устанавливается по факту регистрации или обновления.

22.1.2. Выбор кодов

Для новых запросов выделения кодов из реестров QUIC значения **следует** выбирать случайно, исключая выделенные значения и первый невыделенный код в соответствующем пространстве. По запросам на выделение нескольких кодов **могут** предоставляться непрерывные блоки, это минимизирует риск привязки разной семантики одному коду в разных реализациях.

Первый невыделенный код резервируется для назначения по процедуре Standards Action (см. параграф 4.9 в [RFC8126]). Для таких значений можно использовать процесс раннего назначения [EARLY-ASSIGN]. Для кодов, представляемых целыми числами переменного размера (раздел 16), таких как типы кадров, **следует** использовать коды, представляемые 4 или 8 байтами (значение 2^{14} и выше), если применение не зависит существенно от размера кодирования.

Приложения для регистрации кодов в реестрах QUIC **могут** включать запрошенные коды как часть регистрации. Агентство IANA **должно** выделить выбранный код, если он ещё не занят и соблюдается процедура регистрации.

22.1.3. Повторное заявление предварительных кодов

Может быть подан запрос на удаление неиспользованных кодов с предварительной регистрацией для освобождения пространства в реестре или части реестра (например, из диапазона 64-16383 с кодами переменного размера). Это **следует** применять лишь для кодой с наиболее ранней датой регистрации, а записи, обновлённые менее чем за год до события, **не следует** заявлять повторно.

Запрос на удаление кода **должен** рассматриваться назначенными экспертами. Эксперты **должны** попытаться определить, используется ли удаляемый код. Экспертам рекомендуется связаться с указанными при регистрации контактными лицами, а также с как можно более широким кругом разработчиков протокола для получения сведений о применении кода. Экспертам рекомендуется ждать откликов не менее 4 недель.

Если при поиске обнаружено какое-либо применение кода или сделан запрос на обновление регистрации, повторное заявление кода **недопустимо** и вместо этого обновляется дата регистрации. В регистрационную запись может быть добавлено примечание с указанием полученных сведений. Если код не используется и заявки на повторное выделение нет, код **можно** удалить из реестра.

Процесс рецензирования и консультаций применяется также для запросов на перевод предварительной регистрации в постоянную, но в этом случае цель состоит в проверке представления регистрацией какого-либо используемого развёртывания.

22.1.4. Постоянная регистрация

Для постоянной регистрации в реестрах QUIC применяется процедура Specification Required (параграф 4.6 в [RFC8126]), если не указано иное. Назначенные эксперты проверяют наличие и доступность спецификаций. Экспертам рекомендуется быть беспристрастными, одобряя регистрации, если те не являются оскорбительными, легкомысленными или вредоносными (не просто неприятными эстетически или сомнительными архитектурно). При создании реестра **могут** задаваться дополнительные ограничения для постоянных регистраций.

При создании реестра **могут** указываться диапазоны кодов, для которых регистрация выполняется по особым правилам. Например, в реестре QUIC Frame Types (параграф 22.4) применяется более строгий подход для кодов из диапазона 0 - 63. Любые более строгие требования к постоянной регистрации не препятствуют предварительной регистрации соответствующих кодов. Например, можно запросить предварительную регистрацию кода типа кадра 61.

Все регистрации в документах Standards Track **должны** быть постоянными. Всем регистрациям этого документа выделен статус постоянных и список меняет контролёр IETF в контакте с рабочей группой QUIC (quic@ietf.org).

22.2. Реестр версий QUIC

Агентство IANA создало реестр QUIC Versions в разделе QUIC. Реестр QUIC Versions управляет 32-битовым пространством значений (см. раздел 15). Правила регистрации указаны в параграфе 22.1. Постоянная регистрация в этом реестре выполняется по процедуре Specification Required (параграф 4.6 в [RFC8126]).

Код 0x00000001 выделен с постоянным статусом для протокола, описанного в этом документе. Код 0x00000000 задан как постоянный резерв и указывает, что номер версии зарезервирован для согласования версий. Коды, соответствующие шаблону 0x?a?a?a?a являются резервными и IANA **недопустимо** выделять их, а также **недопустимо** указывать в списке выделенных значений.

22.3. Реестр транспортных параметров QUIC

Агентство IANA создало реестр QUIC Transport Parameters в разделе QUIC. Реестр QUIC Transport Parameters управляет 62-битовым пространством значений. Правила регистрации указаны в параграфе 22.1. Постоянная регистрация в этом реестре выполняется по процедуре Specification Required (параграф 4.6 в [RFC8126]), за

исключением диапазона 0x00 - 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфом 4.9 или 4.10 в [RFC8126].

В дополнение к полям, указанным в параграфе 22.1.1, для постоянной регистрации в этом реестре **должно** указываться поле Parameter Name с кратким мнемоническим обозначением параметра. Исходное содержимое реестра приведено в таблице 6.

Таблица 6. Исходное содержимое реестра параметров транспорта.

Значение	***Имя параметра***	***Документ***
0x00	original_destination_connection_id	Параграф 18.2
0x01	max_idle_timeout	Параграф 18.2
0x02	stateless_reset_token	Параграф 18.2
0x03	max_udp_payload_size	Параграф 18.2
0x04	initial_max_data	Параграф 18.2
0x05	initial_max_stream_data_bidi_local	Параграф 18.2
0x06	initial_max_stream_data_bidi_remote	Параграф 18.2
0x07	initial_max_stream_data_uni	Параграф 18.2
0x08	initial_max_streams_bidi	Параграф 18.2
0x09	initial_max_streams_uni	Параграф 18.2
0x0a	ack_delay_exponent	Параграф 18.2
0x0b	max_ack_delay	Параграф 18.2
0x0c	disable_active_migration	Параграф 18.2
0x0d	preferred_address	Параграф 18.2
0x0e	active_connection_id_limit	Параграф 18.2
0x0f	initial_source_connection_id	Параграф 18.2
0x10	retry_source_connection_id	Параграф 18.2

Значения вида 31*N+27 для целых N (т. е. 27, 58, 89 ...) являются резервным и их **недопустимо** выделять IANA и **недопустимо** включать в список выделенных значений.

22.4. Реестр типов кадров QUIC

Агентство IANA создало реестр QUIC Frame Types в разделе QUIC. Реестр QUIC Frame Types управляет 62-битовым пространством значений. Правила регистрации указаны в параграфе 22.1. Постоянная регистрация в этом реестре выполняется по процедуре Specification Required (параграф 4.6 в [RFC8126]), за исключением диапазона 0x00 - 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфом 4.9 или 4.10 в [RFC8126].

В дополнение к полям, указанным в параграфе 22.1.1, для постоянной регистрации в этом реестре **должно** указываться поле Frame Type Name с кратким мнемоническим обозначением типа кадра.

В дополнение к рекомендациям параграфа 22.1 в спецификации новых постоянных регистраций **следует** включать описание способов, какими конечная точка может определить возможность передачи данного типа кадров. В большинстве случаев предполагается регистрация сопровождающих транспортных параметров (см. параграф 22.3). В спецификациях для постоянной регистрации требуется также описать формат и семантику полей кадра. Исходное содержимое реестра представлено в таблице 3. Реестр не включает колонки «Пакеты» и «Spec» из таблицы 3.

22.5. Реестр кодов транспортных ошибок QUIC

Агентство IANA создало реестр QUIC Transport Error Codes в разделе QUIC. Реестр QUIC Transport Error Codes управляет 62-битовым пространством значений, разделенным на 3 части. Постоянные регистрации выполняются по процедуре Specification Required (параграф 4.6 в [RFC8126]), за исключением диапазона 0x00 - 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфом 4.9 или 4.10 в [RFC8126].

В дополнение к полям, указанным в параграфе 22.1.1, для постоянной регистрации в этом реестре **должны** указываться перечисленные ниже поля.

Code
: Краткое мнемоническое обозначение параметра.

Description
: Краткое описание семантики кода ошибки, в качестве которого может служить ссылка на спецификацию.

Исходное содержимое реестра приведено в таблице 7.

Таблица 7. Исходные записи реестра транспортных ошибок QUIC.

Значение	***Код***	***Описание***	***Документ***
0x00	NO_ERROR	Нет ошибок	Раздел 20
0x01	INTERNAL_ERROR	Ошибка реализации	Раздел 20
0x02	CONNECTION_REFUSED	Сервер отклонил соединение	Раздел 20
0x03	FLOW_CONTROL_ERROR	Ошибка управления потоком данных	Раздел 20
0x04	STREAM_LIMIT_ERROR	Открыто слишком много потоков	Раздел 20
0x05	STREAM_STATE_ERROR	Кадр получен в недействительном состоянии потока	Раздел 20
0x06	FINAL_SIZE_ERROR	Изменение окончательного размера	Раздел 20
0x07	FRAME_ENCODING_ERROR	Ошибка кодирования кадра	Раздел 20
0x08	TRANSPORT_PARAMETER_ERROR	Ошибка в транспортных параметрах	Раздел 20
0x09	CONNECTION_ID_LIMIT_ERROR	Получено слишком много идентификаторов соединения	Раздел 20
0x0a	PROTOCOL_VIOLATION	Нарушение базового протокола	Раздел 20
0x0b	INVALID_TOKEN	Получен недействительный маркер	Раздел 20
0x0c	APPLICATION_ERROR	Ошибка приложения	Раздел 20
0x0d	CRYPTO_BUFFER_EXCEEDED	Переполнение буфера данных CRYPTO	Раздел 20
0x0e	KEY_UPDATE_ERROR	Недействительное обновление защиты пакета	Раздел 20

0x0f	AEAD_LIMIT_REACHED	Избыточное применение ключей защиты пакетов	Раздел 20
0x10	NO_VIABLE_PATH	Нет подходящего пути через сеть	Раздел 20
0x0100-0x01ff	CRYPTO_ERROR	Код сигнала TLS	Раздел 20

23. Литература

23.1. Нормативные документы

[BCP38] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, May 2000. <https://www.rfc-editor.org/info/bcp38>

[DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[EARLY-ASSIGN] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/info/rfc8999>.

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

[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>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

23.2. Дополнительная литература

[AEAD] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[ALTSVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[COOKIE] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust defenses for cross-site request forgery", Proceedings of the 15th ACM conference on Computer and communications security - CCS '08, DOI 10.1145/1455770.1455782, 2008, <https://doi.org/10.1145/1455770.1455782>.

[EARLY-DESIGN] Roskind, J., "QUIC: Multiplexed Stream Transport Over UDP", 2 December 2013, <https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit?usp=sharing>.

[GATEWAY] Hätönen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and M. Kojo, "An experimental study of home gateway characteristics", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement - IMC '10, DOI 10.1145/1879141.1879174, November 2010, <https://doi.org/10.1145/1879141.1879174>.

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[QUIC-MANAGEABILITY] Kuehlewind, M. and B. Trammell, "Manageability of the QUIC Transport Protocol", Work in Progress, Internet-Draft, draft-ietf-quic-manageability-11, 21 April 2021, <https://tools.ietf.org/html/draft-ietf-quic-manageability-11>.

[RANDOM] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <https://www.rfc-editor.org/info/rfc1812>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[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>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7983] Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", RFC 7983, DOI 10.17487/RFC7983, September 2016, <https://www.rfc-editor.org/info/rfc7983>.

[RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[RFC8981] Gont, F., Krishnan, S., Narten, T., and R. Draves, "Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6", RFC 8981, DOI 10.17487/RFC8981, February 2021, <https://www.rfc-editor.org/info/rfc8981>.

[SEC-CONS] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[SLOWLORIS] "RSnake" Hansen, R., "Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client!", June 2009, <https://web.archive.org/web/20150315054838/http://ha.ckers.org/slowloris/>.

Приложение A. Псевдокод

В этом приложении описан псевдокод примеров алгоритмов с упором на корректность и ясность, а не производительность. Сегменты псевдокода лицензируются как компоненты кода (Code Component, см. «Авторские права»).

A.1. Пример декодирования целого числа с переменным размером

Псевдокод на рисунке 45 иллюстрирует считывание целого числа переменного размера из потока байтов. Функция ReadVarint принимает один аргумент - последовательность байтов, которая может быть прочитана в сетевом порядке.

Например, 8-байтовая последовательность 0xc2197c5eff14e88c декодируется в десятичное число 151288809941952652, 4-байтовая последовательность 0x9d7f3e7d - в 494878333, 2-байтовая последовательность 0x7bbd - в 15293, а однобайтовая последовательность 0x25 - в 37 (как и двухбайтовая 0x4025).

```
ReadVarint(data):
  // Размер целых чисел кодируется 2 первыми битами первого байта.
  v = data.next_byte()
  prefix = v >> 6
  length = 1 << prefix

  // Когда размер известен, эти биты удаляются и считываются
  // остальные байты.
  v = v & 0x3f
  повтор length-1 раз:
    v = (v << 8) + data.next_byte()
  return v
```

Рисунок 45. Пример декодирования целого числа с переменным размером.

A.2. Пример алгоритма кодирования номеров пакетов

Псевдокод на рисунке 46 показывает, как реализация может выбрать подходящий размер для кодирования номера пакета. Функция EncodePacketNumber принимает 2 аргумента - full_pn задаёт полный номер пакета, который будет передан, largest_acked - наибольший номер пакета, подтверждённый партнёром в текущем пространстве номеров.

```
EncodePacketNumber(full_pn, largest_acked):
   // Число битов должно быть по меньшей мере на 1 больше
   // двоичного логарифма числа непрерывных неподтвержденных
   // номеров пакетов, включая новый пакет.
   if largest_acked is None:
     num_unacked = full_pn + 1
   else:
     num_unacked = full_pn - largest_acked

   min_bits = log(num_unacked, 2) + 1
   num_bytes = ceil(min_bits / 8)

   // Кодирование целого числа и отсечка до num_bytes
   // младших байтов.
   return encode(full_pn, num_bytes)
```

Рисунок 46. Простой алгоритм декодирования номера пакета.

Например, если конечная точка получила подтверждение для пакета 0xabe8b3 и передаёт пакет с номером 0xac5c02, имеется 29519 (0x734f) остающихся пакетов. Для представления по меньшей мере удвоенного диапазона (59038 пакетов или 0xe69e) требуется 16 битов. В том же состоянии при отправке пакета с номером 0xace8fe применяется 24-битовое кодирование, поскольку нужно не менее 18 битов для представления удвоенного диапазона (131222 пакетов или 0x020096).

A.3. Пример алгоритма декодирования номеров пакетов

```
DecodePacketNumber(largest_pn, truncated_pn, pn_nbits):
   expected_pn  = largest_pn + 1
   pn_win       = 1 << pn_nbits
   pn_hwin      = pn_win / 2
   pn_mask      = pn_win - 1
   // Входящему пакету следует быть больше expected_pn — pn_hwin,
   // но не больше expected_pn + pn_hwin. Это означает, что
   // нельзя просто вырезать биты после expected_pn и добавить
   // truncated_pn, поскольку значение может выйти из окна.
   //
   // Следующий код рассчитывает значение-кандидат и проверяет
   // его попадание в окно номеров. Дополнительная проверка
   // предотвращает переполнение и опустошение.
   candidate_pn = (expected_pn & ~pn_mask) | truncated_pn
   if candidate_pn <= expected_pn - pn_hwin and
      candidate_pn < (1 << 62) - pn_win:
      return candidate_pn + pn_win
   if candidate_pn > expected_pn + pn_hwin and
      candidate_pn >= pn_win:
      return candidate_pn - pn_win
   return candidate_pn
```

Рисунок 47. Пример алгоритма декодирования номера пакета.

Псевдокод на рисунке 47 показывает пример алгоритма декодирования номера пакета после снятия защиты заголовка. Функция DecodePacketNumber принимает 3 аргумента - largest_pn указывает наибольший номер успешно обработанного пакета из текущего пространства номеров, truncated_pn содержит значение поля Packet Number, pn_nbits указывает число битов в поле Packet Number (8, 16, 24 или 32). Например, если максимальный номер успешно аутентифицированного пакета составляет 0xa82f30ea, 16-битовое значение 0x9b32 будет декодировано как 0xa82f9b32.

A.4. Пример алгоритма проверки ECN

Каждый раз, когда конечная точка начинает передачу пакетов по новому пути, она проверяет поддержку ECN на этом пути (см. параграф 13.4). Если путь поддерживает ECN, ставится цель использовать ECN. Конечные точки могут также периодически снова проверять путь, который указал отсутствие поддержки ECN. В этом параграфе описан один из методов проверки пути, но конечные точки могут применять и другие методы.

Пути назначается одно из состояний - testing (проверка), unknown (неизвестно), failed (отказ), capable (поддержка). На путях с состоянием testing или capable конечная точка по умолчанию передаёт пакеты с маркировкой ECT(0), в

остальных случаях пакеты не маркируются. Для начала проверки пути устанавливается статус ECN testing и имеющиеся значения счётчиков ECN запоминаются как базовые. Период тестирования ограничивается числом пакетов или временем, задаваемым конечной точкой. Цель состоит не в ограничении продолжительности тестирования, а в обеспечении передачи достаточного числа маркированных пакетов для получения значений счётчиков ECN, обеспечивающих чёткую индикацию трактовки маркированных пакетов на пути. В параграфе 13.4.2 предлагается устанавливать ограничение в 10 пакетов или 3 интервала PTO.

По завершении тестирования статус ECN для пути становится unknown. Из этого состояния успешная проверка счётчиков ECN в кадре ACK (см. параграф 13.4.2.1) меняет состояние ECN для пути на capable, если не был подтверждён ни один маркированный пакет. При отказе проверки счётчиков ECN статус ECN для пути становится failed. Конечная точка может установить этот статус также в случае обнаружения потери маркированных пакетов или установки для всех таких пакетов маркера ECN-CE.

Применение этого алгоритма гарантирует редкое отключение ECN на путях с корректной поддержкой ECN. Любой путь с некорректным изменением маркировки будет приводить к отключению ECN. В редких случаях, когда маркированные пакеты отбрасываются на пути незначительная продолжительность периода тестирования позволит потерять лишь небольшое число пакетов.

Участники работы

Первоначальное устройство и обоснование этого протокола в значительной степени основаны на работе Jim Roskind [EARLY-DESIGN]. Рабочая группа IETF QUIC получила огромную поддержку от многих людей. Ниже перечислены люди, внёсшие существенный вклад в этот документ.

Alessandro Ghedini
Alyssa Wilk
Antoine Delignat-Lavaud
Brian Trammell
Christian Huitema
Colin Perkins
David Schinazi
Dmitri Tikhonov
Eric Kinnear
Eric Rescorla
Gorry Fairhurst
Ian Swett
Igor Lubashev
奥 一穂 (Kazuho Oku)
Lars Eggert
Lucas Pardue
Magnus Westerlund
Marten Seemann
Martin Duke
Mike Bishop
Mikkel Fahnøe Jørgensen
Mirja Kühlewind
Nick Banks
Nick Harper
Patrick McManus
Roberto Peon
Ryan Hamilton
Subodh Iyengar
Tatsuhiro Tsujikawa
Ted Hardie
Tom Jones
Victor Vasiliev

Адреса авторов

Jana Iyengar (editor)
Fastly
Email: jri.ietf@gmail.com

Martin Thomson (editor)
Mozilla
Email: mt@lowentropy.net

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru