Internet Engineering Task Force (IETF) M. Bishop, Ed.
Request for Comments: 9114 Akamai
Category: Standards Track June 2022
ISSN: 2070-1721

HTTP/3

Аннотация

Транспортный протокол QUIC имеет ряд особенностей, таких как мультиплексирования потоков, управление потоком данных на уровне потока, быстрая организация соединений, которые желательно использовать в транспорте для HTTP. В этом документе рассматривается отображение семантики HTTP на QUIC. Документ также указывает свойства HTTP/2, включённые в QUIC, и описывает перенос расширений HTTP/2 в HTTP/3.

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/rfc9114.

Оглавление

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

1. Введение

Семантика HTTP ([HTTP]) используется для широкого спектра услуг Internet и чаще всего применяется с HTTP/1.1 и HTTP/2. Протокол HTTP/1.1 использовался с разными транспортными и сеансовыми уровнями, HTTP/2 - в основном с TLS по протоколу TCP. HTTP/3 поддерживает ту же семантику с новым транспортным протоколом QUIC.

1.1. Прежние версии HTTP

В HTTP/1.1 [HTTP/1.1] при передаче сообщений HTTP используются поля, ограниченные пробельными символами. Хотя это удобно для человека, использование пробельных символов при форматировании сообщений усложняет разбор и слишком терпимо к вариативному поведению.

Поскольку в HTTP/1.1 нет уровня мультиплексирования, часто используется множество соединений TCP для обслуживания запросов в параллель. Однако это оказывает негативное влияние на контроль перегрузок и эффективность сети, поскольку в TCP нет общего контроля перегрузок для нескольких соединений.

В HTTP/2 ([HTTP/2]) введено двоичное кадрирование и уровень мультиплексирования для сокращения задержек без изменения транспортного уровня. Однако параллельная работа мультиплексирования HTTP/2 не видна механизмам восстановления потерь в TCP, поэтому потеря или нарушение порядка пакетов ведёт к остановке всех активных транзакций, независимо от их связи с потерянными пакетами.

1.2. Передача полномочий в QUIC

Транспортный протокол QUIC включает мультиплексирование потоков и управление потоком данных ка уровне потока (stream), аналогичные предоставляемым уровнем кадрирования HTTP/2. Обеспечивая надёжность на уровне потока и контроль перегрузок во всем соединении, QUIC способен повысить производительность HTTP по сравнению с работой по протоколу TCP. QUIC также включает TLS 1.3 ([TLS]) на транспортном уровне, обеспечивая конфиденциальность и целостность, сравнимые с работой TLS по протоколу TCP, а также сокращение задержки при организации соединений TCP (Fast Open) [TFO].

Этот документ определяет HTTP/3 - отображение семантики HTTP на транспортный протокол QUIC в значительной степени основанное на HTTP/2. HTTP/3 полагается на QUIC для защиты конфиденциальности и целостности данных, проверки подлинности партнёров, а также надёжной, упорядоченной доставки по потокам. Полномочия управления сроком действия потока и управления потоком данных передаются в QUIC, а двоичное кадрирование в каждом потоке похоже на кадрирование HTTP/2. Некоторые свойства HTTP/2 включены в QUIC, а иные реализованы над QUIC.

Протокол QUIC описан [QUIC-TRANSPORT], полное описание HTTP/2, приведено в [HTTP/2].

2. Обзор протокола HTTP/3

HTTP/3 предоставляет транспорт для семантики HTTP, используя транспортный протокол QUIC и внутренний уровень кадрирования, похожий на HTTP/2.

Когда клиент знает о наличии сервера HTTP/3 в некой конечной точке, он организует соединение QUIC. QUIC обеспечивает согласование протокола, мультиплексирование по потокам и управление потоком данных. Обнаружение конечных точек HTTP/3 описано в параграфе 3.1.

В каждом потоке базовым коммуникационным блоком HTTP/3 служит кадр (параграф 7.2). Каждый тип кадра применяется для своих целей, например, кадры HEADERS и DATA являются основой запросов и откликов HTTP (параграф 4.1). Кадры, относящиеся к соединению в целом, передаются в выделенном потоке управления.

Мультиплексирование запросов выполняется с использованием абстракции потока QUIC, описанной в разделе 2 [QUIC-TRANSPORT]. Для каждой пары запрос-отклик применяется один поток QUIC. Потоки не зависят один от другого, поэтому блокировка потока или потеря пакетов не препятствует работе других потоков.

Серверное выталкивание (push) - это введённый в HTTP/2 ([HTTP/2]) механизм взаимодействия, позволяющий серверы выталкивать клиенту обмен запрос-отклик в ожидании того, что клиент сделает указанный запрос. Это является компромиссом между загрузкой сети и возможным сокращением задержки. Некоторые кадры HTTP/3 (такие как PUSH_PROMISE, MAX_PUSH_ID, CANCEL_PUSH) служат для управления выталкиванием с сервера.

Как и в HTTP/2поля запросов и откликов сжимаются для передачи. Поскольку HPACK [HPACK] полагается на упорядоченную передачу разделов сжатых полей, QUIC не гарантирует этого, в HTTP/3 вместо HPACK применяется QPACK [QPACK]. В QPACK используются отдельные односторонние потоки для изменения и отслеживания состояния таблицы полей, а кодированные разделы полей ссылаются на состояние таблицы, не изменяя её.

2.1. Состав документа

Подробный обзор жизненного цикла соединений HTTP/3 включает:

- раздел 3, посвящённый обнаружению конечных точек HTTP/3 и организации соединений HTTP/3;
- раздел 4, описывающий выражение семантики HTTP с помощью кадров;
- раздел 5, описывающий прерывание соединений HTTP/3 (аккуратное и внезапное).

Детали протокола передачи и взаимодействия с транспортом включают:

- раздел 6, описывающий применение потоков QUIC;
- раздел 7, описывающий кадры, применяемые в большинстве потоков;
- раздел 8, описывающий ошибки и их обработку для отдельного потока и соединения в целом.

Финальная часть документа включает дополнительные ресурсы:

- раздел 9, описывающий добавление новых возможностей в будущих документах;
- Приложение A, сравнивающее HTTP/2 и HTTP/3.

2.2. Соглашения и термины

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.

В этом документе используется кодирование целых чисел переменного размера из [QUIC-TRANSPORT].

Ниже даны определения используемых в документе терминов.

abort - прерывание
: Внезапное завершение соединения или потока, возможно из-за ошибки.

client - клиент
: Конечная точка, инициирующая соединение HTTP/3. Клиенты передают запросы и получают отклики HTTP.

connection - соединение
: Соединение транспортного уровня между парой конечных точек, использующих транспортный протокол QUIC.

connection error - ошибка соединения
Ошибка, влияющая на всё соединение HTTP/3.

endpoint - конечная точка
Клиент или сервер в соединении.

frame - кадр
наименьшая коммуникационная единица в потоке HTTP/3, состоящая из заголовка и последовательности байтов переменного размера, структура которой определяется типом кадра.
Элементы протокола, называемые кадрами, присутствуют в этом документе и [QUIC-TRANSPORT]. При обсуждении кадров [QUIC-TRANSPORT] добавляется QUIC, например QUIC CONNECTION_CLOSE. В иных случаях имеются в виду кадры, определённые в параграфе 7.2.

HTTP/3 connection - соединение HTTP/3
Соединение QUIC с согласованным прикладным протоколом HTTP/3.

peer - партнёр
Конечная точка. При обсуждении конкретной конечной точки термин «партнёр» относится к удалённой течке.

receiver - получатель
Конечная точка, принимающая кадры.

sender - отправитель
Конечная точка, передающая кадры.

server - сервер
Конечная точка, воспринимающая соединение HTTP/3. Сервер получает запросы и передаёт отклики HTTP.

stream - поток
Односторонний или двухсторонний поток байтов, обеспечиваемый транспортом QUIC. Все потока соединения HTTP/3 могут считаться потоками HTTP/3, однако в HTTP/3 имеется несколько типов потоков.

stream error - ошибка потока
Ошибка прикладного уровня в отдельном потоке.

Термин «содержимое» (content) определён в параграфе 6.4 [HTTP], термины «ресурс» (resource), «сообщение» (message), «пользовательский агент» (user agent), «сервер-источник» (origin server), «шлюз» (gateway), «посредник» (intermediary), «прокси» (proxy) и «туннель» (tunnel) - в разделе 3 [HTTP].

Пакеты в этом документе показаны в формате, заданном в параграфе 1.3 [QUIC-TRANSPORT], для иллюстрации порядка и размера полей.

3. Организация и поддержка соединений

3.1. Обнаружение конечных точек HTTP/3

HTTP полагается на понятие полномочного отклика, т. е. отклика, сочтённого наиболее подходящим для запроса с учётом состояния целевого ресурса в момент создания сообщения с откликом сервером-источником (или по его указанию), указанным в URI цели. Определение полномочного сервера для HTTP URI описано в параграфе 4.3 [HTTP].

Схема https связывает полномочия с владением сертификатом, который клиент считает доверенным для хоста, указанного компонентом authority в URI. Получив сертификат сервера при согласовании TLS, клиент **должен** проверить, что этот сертификат приемлем для сервера источника из URI по процедуре, описанной в параграфе 4.3.4 [HTTP]. Если сертификат не удаётся проверить по отношению к серверу источнику в URI, клиенту **недопустимо** считать этот сервер полномочным для источника.

Клиент **может** попытаться обратиться к ресурсы по https URI, путём преобразования идентификатора хоста в адрес IP, организации соединения QUIC с этим адресом на указанном порту (включая проверку сертификата сервера, как указано выше) и передачи запросного сообщения HTTP/3, нацеленного URI на сервер через защищённое соединение. Если для выбора HTTP/3 не применяется иной механизм, используется маркер h3 в расширении согласования протокола прикладного уровня (Application-Layer Protocol Negotiation или ALPN, см. [RFC7301]) при согласовании TLS.

Проблемы связности (например, блокировка UDP) могут приводить к отказу при организации соединения QUIC и клиентам **следует** в таких случаях пытаться использовать основанные на TCP версии HTTP.

Серверы **могут** обслуживать HTTP/3 на любом порту UDP и анонсы дополнительных служб всегда явно включают порт, а URI содержат явный порт или используют принятый по умолчанию для схемы.

3.1.1. Дополнительные службы HTTP

Источник HTTP может анонсировать доступность эквивалентной конечной точки HTTP/3 в поле заголовка отклика HTTP Alt-Svc или в кадре HTTP/2 ALTSVC ([ALTSVC]), используя маркер ALPN h3. Например, источник может инициировать отклик HTTP о доступности HTTP/3 на порту UDP 50781 хоста с тем же именем, включая в заголовок поле

```
Alt-Svc: h3=":50781"
```

При получении записи Alt-Svc, указывающей поддержку HTTP/3, клиент **может** попытаться организовать соединение QUIC с указанным хостом и портом и при успехе - передавать запросы HTTP с использованием описанного в этом документе отображения.

3.1.2. Другие схемы

Хотя HTTP не зависит от транспортного протокола, схема http связывает полномочия (authority) с возможностью принимать соединения TCP на указанном порту любого хоста, идентифицированного компонентом authority. Поскольку HTTP/3 не использует TCP, HTTP/3 не может служить для прямого доступа к полномочному серверу ресурса, указанного http URI. Однако расширения протокола, такие как [ALTSVC], позволяют полномочному серверу указать другие службы, которые тоже имеют полномочия и могут быть доступны по протоколу HTTP/3.

Перед запросом к источнику, чья схема отличается от https, клиент **должен** убедиться в готовности сервера обслуживать эту схему. Для источников со схемой http экспериментальный метод решения этой задачи описан в [RFC8164]. В будущем могут быть определены другие механизмы для различных схем.

3.2. Организация соединения

HTTP/3 работает на основе базового транспорта QUIC версии 1. Будущие спецификации **могут** задавать другие версии транспорта QUIC для HTTP/3.

QUIC версии 1 использует TLS версии 1.3 или выше в качестве протокола согласования. Клиент HTTP/3 **должен** поддерживать механизм указания серверу целевого хоста при согласовании TLS. При указании сервера доменным именем [DNS-TERMS], клиенты **должны** передавать расширение TLS для индикации имени сервера (Server Name Indication или SNI) [RFC6066], если не применяется дополнительный механизм указания целевого хоста.

Соединения QUIC организуются в соответствии с [QUIC-TRANSPORT]. В процессе организации поддержка HTTP/3 указывается выбором маркера ALPN h3 в согласовании TLS. В том же согласовании **может** быть предложена поддержка других прикладных протоколов.

Опции на уровне соединения, относящиеся к ядру протокола QUIC, устанавливаются при начальном криптографическом согласовании, а опции HTTP/3 передаются в кадре SETTINGS. После организации соединения QUIC каждая из конечных точек **должна** передать кадр SETTINGS в качестве первого кадра в соответствующего потока управления HTTP.

3.3. Повторное использование соединения

Соединения HTTP/3 сохраняются для множества запросов. Предполагается, что для повышения производительности клиенты не будут закрывать соединения, пока не поймут, что продолжение взаимодействия с сервером больше не нужно (например, при уходе пользователя с web-страницы) или пока сервер не закроет соединения.

После организации соединения с конечной точкой сервера это соединение **можно** использовать неоднократно для запросов с разными компонентами authority в URI. Для использования имеющегося соединения с новым источником, клиент **должен** проверить предоставленный сервером сертификат нового источника с использованием процесса, описанного в параграф 4.3.4 [HTTP]. Это означает, что клиенту нужно сохранять сертификат сервера и любые дополнительные сведения, требуемые для его проверки. Не сделав этого, клиент не сможет повторно использовать соединение для дополнительных источников.

Если сертификат неприемлем для нового источника по какой-либо причине, соединение **недопустимо** использовать снова и для нового источника **следует** создавать новое соединение. Если причина невозможности проверки сертификата может относится к другим источниками, уже связанным с соединением, клиенту **следует** повторно проверить сертификат сервера для этих источников. Например, отказ при проверке сертификата из-за его старения или отзыва может служить для аннулирования всех прочих источников, для которых этот сертификат применялся для организации полномочий (authority).

Клиентам **не следует** создавать более одного соединения HTTP/3 для данного адреса IP и порта UDP, когда адрес и порт могут быть получены из URI, выбранной дополнительной службы ([ALTSVC]), настроенного прокси или распознавания имён для любого из них. Клиент **может** создать несколько соединений HTTP/3 с одним адресом IP и портом UDP, используя различные конфигурации транспорта или TLS, но **следует** избегать создания нескольких соединений с одной конфигурацией.

Серверам рекомендуется поддерживать открытые соединения HTTP/3 как можно дольше, но при необходимости можно разрывать бездействующие (idle) соединения. Когда любая из конечных точек решает закрыть соединение HTTP/3, ей **следует** сначала передать кадр GOAWAY (параграф 5.2), чтобы обе конечные точки могли надёжно определить, были ли обработаны ранее переданные кадры и прервать все оставшиеся задачи.

Сервер, не желающий, чтобы клиенты повторно использовали соединения HTTP/3 для определённого источника, может указать, что он не является полномочным для запроса, передав код статуса 421 (Misdirected Request) в отклике на запрос (см. параграф 7.4 в [HTTP]).

4. Семантика HTTP в HTTP/3

4.1. Кадрирование сообщений HTTP

Клиент передаёт запрос HTTP в потоке запросов, который является инициированым клиентом двухстороннем потоке QUIC (см. параграф 6.1). Клиент **должен** передавать в данном потоке лишь один запрос. Сервер может передавать промежуточные отклики HTTP в том же потоке, затем передаёт один финальный отклик HTTP, как описано ниже. Описание промежуточных и финальных откликов дано в разделе 15 [HTTP].

Выталкиваемые (push) отклики передаются в инициированном сервером одностороннем потоке QUIC (см. параграф 6.2.2). Сервер может передать промежуточные отклики HTTP, за которыми следует один финальный отклик HTTP, как при стандартном отклике. Выталкивание более подробно описано в параграфе (4.6).

Получение в потоке нескольких запросов или дополнительного отклика HTTP после финального отклика HTTP **должно** считаться ошибкой (некорректная форма отклика).

Сообщение HTTP (запрос или отклик) состоит из:

1. раздела заголовков, включающего данные управления и переданного в одном кадре HEADERS;
2. необязательного содержимого в виде последовательности кадров DATA;
3. необязательного раздела трейлеров в кадре HEADERS.

Разделы заголовков и трейлеров описаны в параграфах 6.3 и 6.5 [HTTP], содержимое в параграфе 6.4 [HTTP].

Получение недействительной последовательности кадров **должно** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED. В частности, это может быть кадр DATA перед кадром HEADERS, кадры HEADERS или DATA после трейлерного кадра HEADERS. Иные типы кадров (особенно, неизвестные) могут разрешаться в соответствии с правилами для них (см. раздел 9).

Сервер **может** передать один или несколько кадров PUSH_PROMISE перед, в промежутке или после кадров сообщения с откликом. Эти кадры PUSH_PROMISE не относятся к отклику (см. параграф 4.6). Кадры PUSH_PROMISE не разрешается передавать в push-потоках и вытолкнутый отклик с кадрами PUSH_PROMISE **должен** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED.

Кадры неизвестных типов (раздел 9), включая резервные (параграф 7.2.8) **можно** передавать в поток запроса или выталкивания до, после или между другими кадрами, описанными в этом разделе.

Кадры HEADERS и PUSH_PROMISE могут указывать обновления динамической таблицы QPACK. Хотя эти обновления и не являются напрямую частью обмена сообщениями, они должны приниматься и передаваться для использования сообщения (см. параграф 4.2).

Транспортное кодирование (раздел 7 в [HTTP/1.1]) не определено для HTTP/3 и заголовок Transfer-Encoding применять **недопустимо**.

Отклик **может** состоять из множества сообщений тогда и только тогда, одно или несколько промежуточных сообщений (1xx, см. параграф 15.2 в [HTTP]) предшествует финальному отклику на тот же запрос. Промежуточные отклики не имеют содержимого и разделе трейлеров.

Обмен HTTP запрос-отклик полностью занимает инициированный клиентом двухсторонний поток QUIC. После отправки запроса клиент **должен** закрыть поток для передачи. Если не применяется метод CONNECT (параграф 4.4), клиенту **недопустимо** ставить закрытие потока в зависимость от получения отклика на свой запрос. После приёма финального отклика сервер закрыть поток для передачи. С этого момента поток QUIC полностью закрыт.

Закрытие потока означает конец финального сообщения HTTP. Поскольку некоторые сообщения велики и не ограничены, конечной точке **следует** начинать обработку частичных сообщений HTTP, как получено достаточное для обработки число сообщений. Если инициированный клиентом поток прерывается без достаточного для предоставления полного отклика числа сообщений HTTP, серверу **следует** прервать поток отклика с кодом ошибки H3_REQUEST_INCOMPLETE.

Сервер может отправить полный отклик до того, как клиент полностью передаст запрос, если отклик не зависит от ещё не полученных частей запроса. Если серверу не нужна остальная часть запроса, он **может** прервать считывание потока запроса, передать полный отклик и полностью закрыть передающую часть потока. При запросе к клиенту прекратить передачу в поток запроса **следует** применять код ошибки H3_NO_ERROR. Клиентам **недопустимо** отбрасывать полные отклики в результате внезапного прерывания их запроса, хотя они всегда могут отбрасывать отклики по своему усмотрению в силу иных причин. Если сервер передал частичный или полный отклик, но не прервал чтение запроса, клиенту **следует** продолжать передачу содержимого запроса и завершить поток как обычно.

4.1.1. Отмена и отклонение запросов

После открытия потока запроса этот запрос **может** отклонить любая из конечных точек. Клиенты отклоняют запрос, если он больше не нужен, серверы - при неспособности или нежелании выполнять запрос. Серверам рекомендуется по возможности передавать отклик HTTP с подходящим кодом статуса вместо отмены запроса, обработка которого уже начата. Реализациям **следует** отклонять запросы прерыванием всех направлений потока, которые ещё открыты. Для этого реализация сбрасывает (reset) передающие стороны потоков и прерывает считывание в приёмных частых (см. параграф 2.4 в [QUIC-TRANSPORT]).

Отмена запроса сервером без какой-либо обработки приложением считается отклонением (reject) запроса. Серверу **следует** прервать свой поток отклика с кодом ошибки H3_REQUEST_REJECTED. В этом контексте обработка означает, что некоторые данные из потока были переданы программам вышележащего уровня, которые в результате могут выполнить те или иные действия. Клиент может считать запрос, отклонённый (отвергнутый) сервером, непереданным совсем, что позволяет позднее повторить запрос. Серверам **недопустимо** использовать код ошибки H3_REQUEST_REJECTED для запросов, которые были частично или полностью обработаны. При прерывании сервером отклика после частичной обработки ему **следует** прерывать свой поток откликов с кодом ошибки H3_REQUEST_CANCELLED.

Клиенту **следует** применять для отмены запроса код ошибки H3_REQUEST_CANCELLED. При получении этого кода сервер **может** резко прервать отклик, используя код ошибки H3_REQUEST_REJECTED, если обработка не выполнялась. Клиенту **недопустимо** применять код ошибки H3_REQUEST_REJECTED, за исключением случаев, когда сервер запросил закрытие потока запроса с тем же кодом.

Если поток отменен после приёма полного отклика, клиент **может** игнорировать отмену и использовать отклик. Однако при отмене потока после получения лишь частичного отклика применять такой отклик **не следует**. Безопасно можно повторять лишь идемпотентные методы, такие как GET, PUT, DELETE, и клиенту **не следует** автоматически повторять запросы с неидемпотентным методом, если у него нет возможности узнать, что семантика запроса идемпотентна независимо от метода, или определить, что исходный запрос так и не был применён (см. параграф 9.2.2 в [HTTP]).

4.1.2. Запросы и отклики некорректной формы

Запросы и отклики считаются сформированными некорректно, если в них применяется верная последовательность кадров, но встречается какая-либо из указанных ниже ошибок.

- Наличие запрещённых полей или полей псевдозаголовка.
- Отсутствие обязательных полей псевдозаголовка.
- Недействительные значения полей псевдозаголовка.
- Поля псевдозаголовка после других полей (заголовок или трейлер).
- Недействительная последовательность сообщений HTTP.
- Заглавные буквы в именах полей.
- Недопустимые символы в именах или значениях полей.

Запрос или отклик, для которого определено содержимое, включающий поле Content-Length (параграф 8.6 в [HTTP]), считается некорректно сформированным, если значение поля заголовка Content-Length не совпадает с суммой размеров полученных кадров DATA. Отклик, для которого задано отсутствие содержимого, может иметь ненулевое значения поля Content-Length даже при отсутствии содержимого в кадрах DATA.

Посредникам, обрабатывающим запросы или отклики HTTP (т. е., не выступающим лишь в качестве туннеля), **недопустимо** пересылать некорректно сформированные запросы и отклики, а их обнаружение **должно** считаться ошибкой потока типа H3_MESSAGE_ERROR.

Для некорректно сформированного запроса сервер **может** передать указывающий ошибку отклик HTTP до закрытия или сброса потока. Клиентам **недопустимо** воспринимать некорректно сформированные отклики. Отметим, что эти требования предназначены для защиты от некоторых распространённых атак на HTTP и намеренно сделаны строгими, поскольку попустительство может приводить к уязвимостям реализации.

4.2. Поля HTTP

Сообщения HTTP переносят метаданные как последовательность пар ключ-значение, называемых полями HTTP (см. параграфы 6.3 и 6.5 в [HTTP]). Список зарегистрированных полей HTTP содержится в реестре Hypertext Transfer Protocol (HTTP) Field Name Registry (https://www.iana.org/assignments/http-fields/). Как и в HTTP/2, для HTTP/3 имеются дополнительные соображения, связанные с использованием символов в именах полей, поле заголовка Connection и полях псевдозаголовка. Имена полей являются строки, содержащими подмножество символов ASCII. Свойства имён и значений полей HTTP рассмотрены в параграфе 5.1 [HTTP]. Символы в именах полей **должны** приводиться к нижнему регистру до их кодирования. Запрос или отклик с символами верхнего регистра в именах полей **должен** считаться некорректно сформированным.

В HTTP/3 поле заголовка Connection не применяется для указания относящихся к соединению полей и связанные с соединением метаданные передаются иначе. Конечной точке **недопустимо** создавать раздел полей HTTP/3 с относящимися к соединению полями, а сообщения с таким разделом **должны** считаться ошибочными. Единственным исключением является поле TE, которое **может** присутствовать в заголовке запроса HTTP/3 и в этом случае в него **недопустимо** включать что-либо, кроме trailers.

Посредники, преобразующие сообщение HTTP/1.x в HTTP/3, **должны** удалять связанные с соединением поля заголовка, как указано в параграфе 7.6.1 [HTTP], иначе конечные точки HTTP/3 сочтут их сообщения ошибочными.

4.2.1. Сжатие полей

В [QPACK] описана разновидность HPACK, обеспечивающая кодировщику некоторый контроль над блокировкой head-of-line, которая может вызываться сжатием. Это позволяет кодеру выбрать баланс между эффективностью сжатия и задержкой. QPACK применяется в HTTP/3 для сжатия разделов заголовков и трейлеров, включая данные управления в заголовке.

Для более эффективного сжатия поле заголовка Cookie ([COOKIES]) **может** быть разделено перед сжатием на несколько строк поля, каждая из которых содержит одну или несколько cookie-пар. Если после декомпрессии поле содержит несколько строк cookie, они **должны** объединяться (конкатенация) в одну строку с использованием двухбайтового разделителя «; « (ASCII 0x3b, 0x20) до передачи в контекст, отличный от HTTP/2 и HTTP/3, например, в соединение HTTP/1.1 или базовой серверное приложение HTTP.

4.2.2. Ограничение размера заголовков

Реализация HTTP/3 **может** ограничивать размер заголовка сообщения, который она будет воспринимать в отдельном сообщении HTTP. Сервер, получивший заголовок с размером больше, чем он готов обработать, может передать код статуса HTTP 431 (Request Header Fields Too Large) [RFC6585]. Клиент может отбрасывать отклики, которые он не может обработать. Размер списка полей рассчитывается без учёта сжатия и включает размер имён и значений в байтах, а также издержки в 32 байта на каждое поле.

Если реализация хочет указать партнёру своё ограничение, она может передать его как число байтов в параметре SETTINGS_MAX_FIELD_SECTION_SIZE. Получившей такой параметр реализации **не следует** передавать сообщения HTTP с заголовком больше указанного размера, поскольку партнёр, скорей всего, откажется их обрабатывать. Однако сообщение HTTP может проходить через одного или несколько посредников на пути к серверу-источнику (см. параграф 3.7 в [HTTP], поэтому ограничение применяется раздельно каждой реализацией, обрабатывающей сообщение и размер заголовка выше[1] указанного предела не гарантирует восприятие этого сообщения.

4.3. Данные управления HTTP

Как и в HTTP/2, в HTTP/3 применяется ряд полей псевдозаголовка, имена которых начинаются с символа двоеточия (:, ASCII 0x3a). Эти поля переносят данные управления для сообщения (см. параграф 6.2 в [HTTP]). Поля Поля псевдозаголовков не являются полями заголовка HTTP. Конечным точкам **недопустимо** генерировать поля псевдозаголовка, кроме заданных в этом документе, однако расширения могут менять эти ограничения (см. раздел 9).

Поля псевдозаголовка действительны лишь в контексте, где они были определены, поля псевдозаголовка откликов **недопустимо** включать в запросы и наоборот, а также **недопустимо** включать поля псевдозаголовка в трейлеры. Конечные точки **должны** считать запрос или отклик с неопределённым или недействительным полем псевдозаголовка некорректно сформированными.

Все поля псевдозаголовка **должны** включаться в блок полей до строй обычных полей. Любой запрос или отклик с полями псевдозаголовка в блоке полей после обычных полей заголовка **должен** считаться ошибочным.

4.3.1. Поля псевдозаголовка запроса

:method

Указывает метод HTTP (раздел 9 of [HTTP])

[1] В оригинале ошибочно сказано «ниже», см. https://www.rfc-editor.org/errata/eid7238. *Прим. перев.*

:scheme

Указывающая схему часть URI цели (параграф 3.1 в [URI]).

В поле :scheme можно указывать не только http и https. Прокси и шлюзы могут транслировать запросы в другие (не HTTP) схемы, обеспечивая взаимодействие HTTP с другими типами служб. Рекомендации по использованию отличных от https схем даны в параграфе 3.1.2.

:authority

Компонент полномочий из URI цели (параграф 3.2 в [URI]). **Недопустимо** включать в это поле устаревший субкомпонент userinfo для схем URI http и https.

Для обеспечения точного воспроизведения строки запроса HTTP/1.1 это поле псевдозаголовка **должно** опускаться при трансляции запроса HTTP/1.1 с целью в зависимой от метода форме (см. параграф 7.1 в [HTTP]). Клиентам, генерирующим запросы HTTP/3 напрямую, **следует** использовать поле псевдозаголовка :authority вместо поля заголовка Host. Посредник ACK, преобразующий запрос HTTP/3 в HTTP/1.1, **должен** создавать поле Host, если его нет в запросе, копируя значение из поля псевдозаголовка :authority.

:path

Компоненты пути и запроса из URI цели (path-absolute и необязательный символ ? (ASCII 0x3f) с последующим запросом, см. параграф 4.1 в [HTTP] и 3.4 в [URI][1]). Этому полю псевдозаголовка **недопустимо** быть пустым для URI http и https, которые при отсутствии компонента пути **должны** включать значение /. Запрос OPTIONS без компонента пути включает значение * (ASCII 0x2a) в поле заголовка :path (см. параграф 7.1 в [HTTP]).

Все запросы HTTP/3, кроме CONNECT (параграф 4.4), **должны** включать в точности одно значение для полей псевдозаголовка :method, :scheme, :path.

Если поле псевдозаголовка :scheme указывает схему, где компонент authority обязателен (включая схемы http и https), запрос **должен** содержать поле псевдозаголовка :authority или поле заголовка Host. При наличии этих полей им **недопустимо** быть пустыми. Если присутствуют оба поля, из значения **должны** совпадать. Если схема не требует обязательного компонента authority и он не указан в цели запроса, **недопустимо** включать в запрос :authority или Host.

Запрос HTTP, где отсутствует обязательное поле псевдозаголовка или указано непригодное значение в поле псевдозаголовка, является ошибочным.

HTTP/3 не задаёт способа передачи идентификатора версии, который включается в строку запроса HTTP/1.1. Запросы HTTP/3 неявно имеют версию протокола 3.0.

4.3.2. Поля псевдозаголовка отклика

Для откликов определено одно поле псевдозаголовка :status, передающее код статуса HTTP (см. раздел 15 в [HTTP]). Это поле **должно** включаться во все отклики, которые иначе станут некорректно сформированными (параграф 4.1.2).

HTTP/3 не задаёт способа передачи версии или фразы с причиной, включаемых в строку состояния HTTP/1.1. В откликах HTTP/3 неявно используется протокол версии 3.0.

4.4. Метод CONNECT

Метод CONNECT запрашивает у получателя организацию туннеля к серверу-источнику, указанному в request-target (см. параграф 9.3.6 в [HTTP]) и применяется в основном HTTP-прокси при организации сессии TLS с сервером-источником для взаимодействия с ресурсами https.

В HTTP/1.x метод CONNECT служит для преобразования всего соединения HTTP в туннель к удалённому хосту, в HTTP/2 и HTTP/3 метод CONNECT служит для организации туннеля через один поток.

Запрос CONNECT **должен** создаваться в соответствии с приведёнными ниже требованиями.

- В поле псевдозаголовка :method указывается CONNECT.
- Поля псевдозаголовка :scheme и :path не используются.
- Поле псевдозаголовка :authority содержит хост и порт для организации соединения (эквивалент authority-form для the request-target в запросах CONNECT, см. параграф 7.1 в [HTTP]).

Поток запроса по его завершении остаётся открытым для передачи данных. Запросы CONNECT, не соответствующие этим требованиям, считаются некорректно сформированными.

Прокси, поддерживающий CONNECT, организует соединение TCP [TCP] с хостом и портом, указанными в поле псевдозаголовка :authority. После организации соединения прокси передаёт клиенту кадр HEADERS с кодом 2xx, как указано в параграфе 15.3 [HTTP]. Все кадры DATA в потоке соответствуют данным, переданным через соединение TCP. Содержимое (payload) всех кадров DATA от клиента прокси передаёт серверу TCP, а данные от этого сервера прокси собирает в кадры DATA. Отметим, что размер и число сегментов TCP не обязательно совпадает с размером и числом кадров HTTP DATA или QUIC STREAM.

По завершении метода CONNECT в поток можно передавать лишь кадры DATA. **Могут** использоваться кадры расширения, если это явно разрешено в их определении. Приём иного типа кадра **должен** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED.

Соединение TCP может закрыть любой из партнёров. Когда клиент завершает поток запроса (приёмный поток на прокси переходит в состояние Data Recvd), прокси устанавливает бит FIN в своём соединении с сервером TCP. При получении прокси пакета с установленным битом FIN он закрывает поток передачи, отправляемый клиенту. Соединения TCP, остающиеся полузакрытыми в одном направлении, не являются недействительными, но серверы зачастую плохо обрабатывают их, поэтому клиентам **не следует** закрывать поток для передачи, пока они ещё ожидают получения данных от цели CONNECT.

Ошибки соединения TCP указываются резким прерыванием потока. Прокси считает любую ошибку соединения TCP, включающую получение сегмента TCP с установленным битом RST, ошибкой потока типа H3_CONNECT_ERROR. Если прокси обнаруживает ошибку в потоке или соединении QUIC, от **должен** закрыть соединение TCP. Если прокси

[1] В оригинале ошибочно указаны параграфы 3.3 и 3.4 в [URI], см. https://www.rfc-editor.org/errata/eid7014. *Прим. перев.*

обнаруживает, что клиент сбросил (reset) поток или прервал чтение из него, прокси **следует** выполнить такую же операцию в другом направлении, чтобы гарантированно отменить оба направления потока. Во всех таких случаях прокси **следует** передавать сегмент TCP с установленным битом RST, если реализация TCP позволяет это.

Поскольку CONNECT создаёт туннель к произвольному серверу, прокси, поддерживающим CONNECT, **следует** разрешать применение этого метода лишь для известных портов или списка безопасных целей запроса (см. параграф 9.3.6 в [HTTP]).

4.5. Обновление HTTP

HTTP/3 не поддерживает механизм HTTP Upgrade (параграф 7.8 в [HTTP]) и код статуса 101 (Switching Protocols) (параграф 15.2.2 в [HTTP]).

4.6. Выталкивание с сервера

Выталкивание с сервера - это интерактивный режим, позволяющий серверу вытолкнуть (push) обмен запрос-отклик клиенту в расчёте на то, что клиент внесёт такой запрос. С этим связан компромисс между загрузкой сети и возможной задержкой. Выталкивание с сервера HTTP/3 похоже на описанное в параграфе 8.2 [HTTP/2], но использует другие механизмы.

Каждому выталкиванию присваивается уникальный идентификатор (push ID), служащий для указания выталкивания в разных контекстах в течение срока действия соединения HTTP/3. Пространство push ID начинается с 0 и завершается максимальным значением, заданным кадром MAX_PUSH_ID. Сервер не может выталкивать отклики до момента передачи клиентом кадра MAX_PUSH_ID, который служит для управления числом выталкиваний, которые может обещать сервер. Серверу **следует** использовать идентификаторы по порядку, начиная с 0. Клиент **должен** считать получение вытолкнутого потока ошибкой соединения типа H3_ID_ERROR, если не был передан кадр MAX_PUSH_ID или значение push ID превышает заданный максимум.

Идентификатор выталкивания используется в одном или нескольких кадрах PUSH_PROMISE, передающих данные управления и поля заголовка запросного сообщения. Эти кадры передаются в потоке запросов, вызвавшем выталкивание, что позволяет серверу связать выталкивание с запросом клиента. Когда один идентификатор выталкивания обещан в нескольких потоках запросов, распакованные разделы полей запросов **должны** содержать одни и те же поля в одинаковом порядке, а имя и значение каждого поля **должны** быть идентичны.

Идентификатор выталкивания включается в поток выталкивания, который в конечном итоге выполняет обещания. Идентификатор потока выталкивания указывает push ID выполняемого обещания и включает отклик на предсказанный (promised) запрос, как указано в параграфе 4.1. Идентификатор выталкивания может использоваться в кадрах CANCEL_PUSH (см. параграф 7.2.3), используемых клиентами. Чтобы указать нежелание получать обещанный ресурс. Сервер использует эти кадры, чтобы указать отмену исполнения предыдущего обещания.

Не все отклики пригодны для выталкивания. Сервер **может** выталкивать запросы, которые:

- являются кэшируемыми (см. параграф 9.2.3 в [HTTP]);
- безопасны (см. параграф 9.2.1 в [HTTP]);
- не включают содержимого и раздела трейлеров.

Сервер **должен** включать поле псевдозаголовка :authority, для которого сервер является полномочным. Если клиент ещё не проверил соединение на предмет источника, указанного вытолкнутым запросом, он **должен** выполнить такой же процесс проверки, который он применил бы перед отправкой запроса этому источнику (см. параграф 3.3). если проверка завершается отказом, клиенту **недопустимо** считать сервер полномочным для этого источника.

Клиентам **следует** передавать кадр CANCEL_PUSH при получении кадра PUSH_PROMISE с запросом, который не может кэшироваться, не содержит сведений о своей безопасности, указывает наличие содержимого или для которого сервер не является полномочным. Соответствующие отклики **недопустимо** использовать и кэшировать.

Каждый выталкиваемый отклик связан с одним или несколькими клиентскими запросами. Выталкивание связывается с потоком запросов, в котором был получен кадр PUSH_PROMISE. Выталкивание с сервера может быть с дополнительными клиентскими запросами с помощью кадров PUSH_PROMISE с тем же push ID в нескольких потоках запросов. Эти связи не влияют на работу протокола, но **могут** учитываться пользовательским агентом при решении вопроса об использовании вытолкнутых ресурсов.

Важен порядок PUSH_PROMISE относительно некоторых частей отклика. Серверу **следует** передавать кадры PUSH_PROMISE до отправки HEADERS или DATA, ссылающихся на обещанные отклики. Это снижает вероятность запроса клиентом ресурса, который был вытолкнут сервером. При изменении порядка данные вытолкнутого потока могут прибывать раньше соответствующего кадра PUSH_PROMISE. При получении клиентом нового push-потока с неизвестным идентификатором выталкивания связанный с ним запрос клиента и поля заголовка вытолкнутого запроса неизвестны. Клиент может буферизовать данные потока в ожидании соответствующего кадра PUSH_PROMISE. Клиент может использовать управление потоком данных в потоке (stream) (параграф 4.1 в [QUIC-TRANSPORT]) для ограничения объёма данных, которые сервер может представить в выталкиваемый поток. Клиентам **следует** прерывать чтение и отбрасывать уже прочтённые данные из push-потоков, если соответствующий кадр PUSH_PROMISE не был получен в разумное время.

Данные вытолкнутого потока могут прибывать после того, как клиент отменит выталкивание. В этом случае клиент может прервать чтение потока с кодом ошибки H3_REQUEST_CANCELLED. Это будет просить у сервера прекратить передачу дополнительных данных и указывать, что они будут отброшены при получении.

Вытолкнутые отклики, которые являются кэшируемыми (см. раздел 3 в [HTTP-CACHING]), могут сохраняться клиентом, если он реализует кэширование HTTP. Вытолкнутые отклики считаются подтверждёнными сервером-источником (например, наличием в отклике директивы no-cache, см. параграф 5.2.2.4 в [HTTP-CACHING) в момент их получения. Вытолкнутые отклики, которые не являются кэшируемыми, **недопустимо** сохранять в кэшах HTTP. Они **могут** быть сделаны доступными приложению отдельно.

5. Закрытие соединения

Организованное соединение HTTP/3 можно применять для множества запросов и откликов, пока оно не будет закрыто. Закрытие соединения может выполняться разными способами, описанными ниже.

5.1. Бездействующие соединения

Каждая конечная точка QUIC при согласовании объявляет тайм-аут простоя. Если соединение QUIC бездействует (пакеты не передаются) дольше этого времени, партнёр будет считать соединение закрытым. Реализациям HTTP/3 потребуется создать новое соединение HTTP/3 для новых запросов, если имеющееся соединение простаивало (idle) дольше тайм-аута, заданного при согласовании QUIC и это **следует** делать при приближении тайм-аута (см. параграф 10.1 в [QUIC-TRANSPORT]).

Предполагается, что клиенты HTTP будут просить у транспорта сохранять соединения открытыми, пока остаются отклики на запросы или сервер использует выталкивание, как описано в параграфе 10.1.2 [QUIC-TRANSPORT]. Если клиент не ожидает отклика от сервера, лучше позволить завершить простаивающее соединение, чем прилагать усилия по его сохранению, которые могут оказаться пустыми. Шлюз **может** поддерживать соединения в ожидании потребности в них, чтобы не возникало задержки на организацию нового соединения. Серверам **не следует** активно поддерживать соединения открытыми.

5.2. Отключение соединения

Конечная точка может остановить использование соединения и инициировать его аккуратное закрытие, даже если соединение не простаивает. Конечная точка инициирует аккуратное отключение (shutdown) соединения HTTP/3 передачей кадра GOAWAY. Такой кадр содержит идентификатор, указывающий получателю диапазон запросов или выталкиваний, которые были или могут быть обработаны в этом соединении. Сервер передаёт идентификатор инициированного клиентом двухстороннего потока. клиент передаёт идентификатор выталкивания. Запросы или выталкивания с указанным или большим идентификатором отклоняются (параграф 4.1.1) отправителем кадра GOAWAY. Идентификатор может иметь значение 0, если запросы или выталкивания не обрабатывались.

Сведения из кадра GOAWAY позволяют клиенту и серверу договориться о том, какие запросы или выталкивания были восприняты до отключения соединения HTTP/3. После отправки GOAWAY конечной точке **следует** явно отменять (см. параграфы 4.1.1 и 7.2.3) любые запросы или выталкивания, имеющие идентификатор не меньше указанного, чтобы очистить транспортное соединение от затронутых потоков. Конечной точке **следует** продолжать это делать по мере прибытия новых запросов или выталкиваний.

Конечным точкам **недопустимо** инициировать новые запросы в соединении после приёма от партнёра кадра GOAWAY. Для передачи дополнительных запросов клиенты **могут** создавать новое соединение.

Некоторые запросы или выталкивания могут оказаться уже переданными (но ещё не полученными).

- После получения кадра GOAWAY запросы клиента, переданные с идентификатором потока не меньше указанного в кадре GOAWAY, не будут обработаны. Клиенты могут безопасно повторить такие запросы в другом соединении HTTP. Клиент, не способный повторить запрос, теряет все запросы, находившиеся в пути, когда сервер закрыл соединение. Запросы с идентификатором потока меньше указанного в GOAWAY от сервера могут быть обработаны, но их статус нельзя узнать, пока не будет получен отклик, поток не будет сброшен индивидуально или не будет получен другой кадр GOAWAY с меньшим идентификатором потока, чем у рассматриваемого запроса, или соединение не будет прервано. Сервер **может** отклонять отдельные запросы в потоках с идентификатором меньше указанного, если эти запросы не были обработаны.
- Если сервер получает кадр GOAWAY после отправки обещанных выталкиваний и push ID не меньше указанного в кадре GOAWAY, эти выталкивания не будут восприняты.

Серверам **следует** передавать кадр GOAWAY, когда о закрытии соединения известно заранее, даже если это упреждение невелико, чтобы удалённый партнёр мог знать, был ли его запрос частично обработан. Например, при передаче клиентом HTTP запроса POST в то время, когда сервер закрывает соединение QUIC, клиент не может знать, началась ли обработка POST, если сервер не передал кадр GOAWAY с указанием потоков, на которые он мог влиять.

Конечная точка **может** передать несколько кадров GOAWAY с разными идентификаторами, но каждый следующий идентификатор **должен** быть не больше предшествующего, поскольку клиенты уже могли повторить необработанные запросы в другом соединении HTTP. Получение GOAWAY с большим, чем раньше значением идентификатора, **должно** считаться ошибкой соединения типа H3_ID_ERROR.

Конечная точка, пытающаяся аккуратно завершить соединение, может передать кадр GOAWAY с максимально возможным значением идентификатора (2^{62}-4 для сервера, 2^{62}-1 для клиента). Это гарантирует, что партнёр остановит создание новых запросов или выталкиваний. По истечении времени на прибытие находящихся в пути запросов или выталкиваний конечная точка может передать другой кадр GOAWAY, указывающий, какие запросы или выталкивания она может воспринять до завершения соединения. Это гарантирует завершение соединения без потери запросов.

Клиент имеет больше возможностей для выбора значения Push ID в передаваемом кадре GOAWAY. Значение 2^{62}-1 указывает, что сервер может продолжать выталкивания, которые он уже обещал. Меньшее значение указывает, что клиент будет отклонять выталкивания с Push ID не меньше указанного им значения. Как и сервер, клиент **может** передать следующие кадры GOAWAY, в которых Push ID не превышает переданное ранее значение.

Даже когда кадр GOAWAY указывает, что данный запрос или выталкивание не будет обработан или воспринят после получения, ресурсы базового транспорта ещё сохраняются. Инициировавшая такие запросы конечная точка может отменить их для очистки своего транспортного состояния.

После обработки всех воспринятых запросов и выталкиваний конечная точка может сохранять простаивающее соединение или **может** инициировать его незамедлительное закрытие. Конечное точке, завершающей аккуратное отключение, **следует** использовать при закрытии код H3_NO_ERROR.

Если клиент уже использовал все доступные идентификаторы двухсторонних потоков в запросах, серверу не требуется передавать кадр GOAWAY, поскольку клиент больше не сможет подавать запросы.

5.3. Немедленное закрытие приложения

Реализация HTTP/3 может в любой момент без промедления закрыть соединение QUIC. Это ведёт к отправке партнёру кадра QUIC CONNECTION_CLOSE, указывающего, что прикладной уровень прервал соединение. Код ошибки прикладного уровня в этом кадре говорит партнёру о причине разрыва соединения. В разделе 8 указаны коды ошибок, которые можно использовать при закрытии соединения HTTP/3.

Перед закрытием соединения **может** быть передан кадр GOAWAY, чтобы клиент мог повторить некоторые запросы. Включение GOAWAY в один пакет с кадром QUIC CONNECTION_CLOSE повышает вероятность получения кадра клиентом.

При наличии открытых потоков, которые не были закрыты явно, они неявно закрываются при закрытии соединения (см. параграф 10.2 в [QUIC-TRANSPORT]).

5.4. Закрытие транспорта

По разным причинам транспорт QUIC может сообщать прикладному уровню о разрыве соединения. Это может быть результатом явного закрытия партнёром, ошибки на транспортном уровне или изменения топологии сети, прерывающего связность. Если соединение прерывается без передачи кадра GOAWAY, клиенты **должны** полагать, что любые переданные запросы обработаны полностью или частично.

6. Сопоставление и использование потоков

Поток QUIC обеспечивает надёжную, упорядоченную доставку байтов, но не даёт гарантий порядка доставки байтов относительно других потоков. В QUIC версии 1 поток данных с кадрами HTTP передаётся в кадрах QUIC STREAM, но это кадрирование невидимо для уровня кадрирования HTTP. Транспортный уровень буферизует и упорядочивает полученные данные потока, предоставляя приложению надёжный поток байтов. Хотя QUIC допускает доставку с нарушением порядка в потоке, HTTP/3 эту возможность не использует.

Потоки QUIC могут быть односторонними, передающими лишь данные от инициатора к получателю, или двухсторонними с передачей данных в обоих направлениях. Инициатором потока может быть клиент или сервер. Дополнительные сведения о потоках QUIC приведены в разделе 2 [QUIC-TRANSPORT].

Когда поля и данные HTTP передаются через QUIC, уровень QUIC обеспечивает большую часть управления потоком (stream). HTTP не требуется какое-либо мультиплексирование при использовании QUIC, данные, переданные в поток QUIC, всегда сопоставляются с конкретной транзакцией HTTP или контекстом всего соединения HTTP/3.

6.1. Двухсторонние потоки

Все инициируемые клиентом двухсторонние потоки применяются для запросов и откликов HTTP. Двухсторонний поток гарантирует надёжное сопоставление откликов с запросами. Такие потоки называются потоками запросов. Это означает, что первый запрос клиента выполняется в потоке QUIC 0, в последующие в потоках 4, 8 и т. д. Чтобы разрешить создание этих потоков серверу HTTP/3 **следует** задать отличные от 0 значения для числа разрешённых потоков и исходного размера окна управления потоком данных в потоке (stream). Чтобы не вносить избыточных ограничений для параллельной работы **следует** разрешать не менее 100 потоков запросов.

В HTTP/3 не применяются инициируемые сервером двухсторонние потоки, хотя могут задаваться расширения с такими потоками. Клиенты **должны** считать инициированный сервером двухсторонний поток ошибкой соединения типа H3_STREAM_CREATION_ERROR, если не было согласовано применяющее такие потоки расширение.

6.2. Односторонние потоки

Односторонние потоки обоих направлений применяются для разных целей. Назначение потока определяет тип, указываемый целым числом переменного размера в начале потока. Формат и структура данных в потоке определяются типом потока.

```
Unidirectional Stream Header {
  Stream Type (i),
}
```

Рисунок 1. Заголовок одностороннего потока.

Этот документ определяет два типа - потоки управления (control stream, параграф 6.2.1) и потоки выталкивания (push stream, параграф 6.2.2). В [QPACK] определены два дополнительных типа, а расширения HTTP/3 могут задавать свои типы (см. раздел 9). Некоторые типы потоков зарезервированы (параграф 6.2.3).

Производительность соединения HTTP/3 в его ранней фазе зависит от создания и обмена данными через односторонние потоки. Конечные точки, чрезмерно ограничивающие число потоков или размер окна управления потоком данных для этих потоков, повышают вероятность того, что удалённый партнёр быстрее достигнет предела и будет заблокирован. В частности, реализациям следует учитывать, что удалённые партнёры могут захотеть использовать поведение зарезервированного потока (параграф 6.2.3) с некоторыми из односторонних потоков, которые им разрешены.

Каждая конечная точка должна по меньшей мере создать один односторонний поток для управления HTTP. Для QPACK нужны два дополнительных односторонних потока, а другим расширениям могут требоваться свои потоки. Поэтому параметры транспорта, переданные клиентом и сервером, **должны** разрешать партнёру создание не менее трёх односторонних потоков. Этим параметрам **следует** предоставлять кредит управления потоком данных не менее 1024 байтов для каждого одностороннего потока. Отметим, что конечная точка не обязана предоставлять дополнительный кредит для создания большего числа односторонних потоков, когда партнёр уже использовал весь выделенный кредит до создания критически важных односторонних потоков. Конечным точкам **следует** сначала создавать поток управления HTTP, а также односторонние потоки, требуемые для обязательных расширений (таких как потоки кодера и декодера QPACK), а затем - дополнительные потоки, разрешённые партнером.

Если заголовок потока указывает не поддерживаемый получателем тип, остальная часть потока не может быть воспринята, так как её семантика неизвестна. Получатель потока неизвестного типа **должен** прервать чтение из потока

или отбросить входные данные без дальнейшей обработки. При прерывании чтения получателю **следует** использовать код ошибки H3_STREAM_CREATION_ERROR или резервный код (параграф 8.1). Получателю **недопустимо** считать неизвестный тип потока ошибкой соединения.

Поскольку некоторые типы потоков могут влиять на состояние соединения, получателю **не следует** отбрасывать данные из входящего одностороннего потока до прочтения типа потока. Реализации **могут** передавать потоки, не зная, поддерживается ли тип потока партнёром. Однако типы потока, которые могут менять статус и семантику имеющихся компонентов протокола, включая QPACK и другие расширения, **недопустимо** передавать, пока неизвестно об их поддержке партнёром.

Отправитель может закрыть или сбросить (reset) односторонний поток, если не указано иное. Получатель **должен** допускать закрытие или сброс односторонних потоков до получения заголовка потока.

6.2.1. Поток управления

Поток управления имеет тип 0x00, а данными этого потока являются кадры HTTP/3, определённые в параграфе 7.2.

Каждая сторона **должна** инициировать один поток управления в начале соединения и первым передать в него кадр SETTINGS. Приём первым кадра иного типа **должно** считаться ошибкой соединения типа H3_MISSING_SETTINGS. Для партнёра создаётся лишь один поток управления и получение другого потока этого типа **должно** считаться ошибкой соединения типа H3_STREAM_CREATION_ERROR. Отправителю **недопустимо** закрывать поток управления, а получателю **недопустимо** запрашивать у отправителя закрытие такого потока. Закрытие любого из потоков управления любой точкой **должно** считаться ошибкой соединения типа H3_CLOSED_CRITICAL_STREAM. Ошибки соединений описаны в разделе 8.

Поскольку содержимое потока управления служит для управления поведением других потоков, конечным точкам **следует** предоставлять достаточный кредит управления потоком данных, чтобы поток управления не блокировался.

Для управления используется пара односторонних потоков и не один двухсторонний. Это позволяет любому из партнёров передавать данные, как только он сможет. В зависимости от доступности 0-RTT в соединении QUIC клиент или сервер сможет первым начать передачу данных в поток.

6.2.2. Push-потоки

Выталкивание (push) с сервера, введённое в HTTP/2, позволяет серверу инициировать отклик до получения запроса (см. параграф 4.6). Выталкиваемый поток указывается типом 0x01, за которым следует идентификатор выталкивания (push ID) для выполняемого обещания (promise) в виде целого числа с переменным размером. Данными в этом потоке являются кадры HTTP/3 (см. параграф 7.2), выполняющие обещанное сервером выталкивание с помощью необязательных промежуточных откликов HTTP, за которыми следует финальный отклик HTTP, как указано в параграфе 4.1. Выталкивание с сервера и его идентификатор описаны в параграфе 4.6.

Выталкивание разрешено только серверу и получение сервером инициированного клиентом push-потока **должно** считаться ошибкой соединения типа H3_STREAM_CREATION_ERROR.

```
Push Stream Header {
  Stream Type (i) = 0x01,
  Push ID (i),
}
```

Рисунок 2. Заголовок потока Push.

Клиенту **не следует** прерывать чтение push-потока, пока не прочитан его заголовок, поскольку это может приводить к разному восприятию клиентом и сервером уже использованных push ID. Каждый идентификатор потока **должен** использоваться только один раз в заголовке push-потока. Если клиент обнаруживает в заголовке push-потока значение push ID, которое уже использовалось в заголовке другого push-потока, он **должен** считать это ошибкой соединения типа H3_ID_ERROR.

6.2.3. Зарезервированные типы потоков

Потоки типов 0x1f * N + 0x21 с неотрицательными значениями N зарезервированы для выполнения требования игнорировать неизвестные типы. Эти потоки не имеют семантики и могут передаваться при необходимости заполнения на прикладном уровне. Они **могут** также передаваться в соединениях, где в настоящее время не передаются данные. Конечным точкам **недопустимо** придавать таким потокам какое-либо значение.

Содержимое и размер потока выбираются произвольным способом по выбору передающей реализации. Передав поток зарезервированного типа, реализация **может** полностью прервать или сбросить его. При сбросе **следует** использовать код H3_NO_ERROR или зарезервированный код ошибки (параграф 8.1).

7. Уровень кадрирования HTTP

Кадры HTTP передаются в потоках QUIC, как описано в разделе 6. В HTTP/3 определено 3 типа потоков - управление (control stream), запрос (request stream) и выталкивание (push stream). В этом разделе описаны форматы кадров HTTP/3 и разрешённые для них типы потоков (см. таблицу 1). Сравнение кадров HTTP/2 и HTTP/3 приведено в Приложении A.2.

Таблица 1. Типы кадров и потоков HTTP/3.

Кадр	*Поток управления*	*Поток запросов*	*Поток выталкивания*	*Параграф*
DATA	нет	да	да	7.2.1
HEADERS	нет	да	да	7.2.2
CANCEL_PUSH	да	нет	нет	7.2.3
SETTINGS	да (1)	нет	нет	7.2.4
PUSH_PROMISE	нет	да	нет	7.2.5
GOAWAY	да	нет	нет	7.2.6
MAX_PUSH_ID	да	нет	нет	7.2.7
Reserved	да	да	да	7.2.8

Кадр SETTINGS может передаваться лишь в качестве первого кадра в потоке управления, это отмечено в таблице 1 как (1). Конкретные рекомендации приведены в соответствующем параграфе.

Отметим, что в отличие от кадров QUIC, кадры HTTP/3 могут разделяться между несколькими пакетами.

7.1. Схема кадра

Формат кадров показан на рисунке 3.

```
HTTP/3 Frame Format {
  Type (i),
  Length (i),
  Frame Payload (..),
}
```

Рисунок 3. Формат кадра HTTP/3.

Ниже описаны поля кадра.

Type
Целое число переменного размера, указывающее тип кадра.

Length
Целое число переменного размера, указывающее число байтов в Frame Payload.

Frame Payload
Содержимое, семантика которого определяется полем Type.

В содержимом (payload) каждого кадра **должны** присутствовать именно те поля, которые указаны в описании кадра. Кадр, содержимое ккоторого включает байты после указанных полей или завершается до конца указанных полей, **должен** считаться ошибкой соединения типа H3_FRAME_ERROR. Избыточное значение размера в поле Length **должно** проверяться на самосогласованность (см. параграф 10.8).

При полном (чистом) завершении потока отсечка **должна** считаться ошибкой соединения типа H3_FRAME_ERROR. При внезапном завершении поток может прерываться в любом месте кадра.

7.2. Определения кадров

7.2.1. DATA

Кадры DATA (type=0x00) передают последовательности байтов переменного размера, связанных с содержимым запроса или отклика HTTP.

```
DATA Frame {
  Type (i) = 0x00,
  Length (i),
  Data (..),
}
```

Рисунок 4. Кадр DATA.

Кадры DATA **должны** быть связаны с запросом или откликом HTTP. На приём кадра DATA в потоке управления получатель **должен** отвечать ошибкой соединения типа H3_FRAME_UNEXPECTED.

7.2.2. HEADERS

Кадр HEADERS (type=0x01) служит для передачи раздела заголовков HTTP, кодируемого с использованием QPACK. Дополнительные сведения представлены в [QPACK].

```
HEADERS Frame {
  Type (i) = 0x01,
  Length (i),
  Encoded Field Section (..),
}
```

Рисунок 5. Кадр HEADERS.

Кадры HEADERS могут передаваться только в потоке запроса или выталкивания. Получение HEADERS в потоке управления **должно** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED.

7.2.3. CANCEL_PUSH

Кадр CANCEL_PUSH (type=0x03) служит для запроса отмены выталкивания с сервера до получения push-потока и указывает идентификатор выталкиваемого потока (параграф 4.6), указанный целым числом с переменным размером.

Передача клиентом кадра CANCEL_PUSH указывает его нежелание принимать обещанный (promise) отклик. Серверу следует прервать передачу ресурса, но механизм прерывания зависит от состояния соответствующего push-потока. Если сервер ещё не создал поток выталкивания, он просто не создаётся. Если такой поток создан, серверу **следует** резко прервать его. Если поток уже завершён, сервер **может** резко прервать его или не делать ничего.

Сервер передаёт кадр CANCEL_PUSH, чтобы указать отказ от выполнения переданного ранее обещания. Клиент не может ожидать соответствующего обещания, если он уже не получил и не обработал обещанный отклик. Независимо от того, был ли создан push-поток, серверу **следует** передавать кадр CANCEL_PUSH, когда он понимает, что обещание не будет исполнено. Если поток создан, сервер может прервать его передачу с кодом ошибки H3_REQUEST_CANCELLED.

Передача CANCEL_PUSH не оказывает прямого воздействия на имеющиеся потоки выталкивания. Клиенту **не следует** передавать кадр CANCEL_PUSH, если он уже получил соответствующий push-поток. Вытолкнутый поток может прибыть после отправки кадра CANCEL_PUSH, поскольку сервер мог не обработать этот кадр. Клиенту в таком случае **следует** прервать считывание потока с кодом ошибки H3_REQUEST_CANCELLED.

Кадр CANCEL_PUSH передаётся в потоке управления и его получение в другом потоке **должно** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED.

```
CANCEL_PUSH Frame {
  Type (i) = 0x03,
  Length (i),
  Push ID (i),
}
```

Рисунок 6. Кадр CANCEL_PUSH.

Кадр CANCEL_PUSH содержит идентификатор выталкивания в форме целого числа с переменным размером. Поле Push ID указывает отменяемое выталкивание с сервера (см. параграф 4.6). Получение CANCEL_PUSH с идентификатором выталкивания больше текущего значения в соединении **должно** считаться ошибкой соединения типа H3_ID_ERROR.

Полученный клиентом кадр CANCEL_PUSH может указывать идентификатор выталкивания, ещё не принятый в кадре PUSH_PROMISE по причине нарушения порядка. Если сервер получает CANCEL_PUSH для идентификатора выталкивания, ещё не указанного в кадре PUSH_PROMISE, он **должен** считать это ошибкой соединения типа H3_ID_ERROR.

7.2.4. SETTINGS

Кадр SETTINGS (type=0x04) передаёт параметры концигурации, влияющие на взаимодействие конечных точек, например, предпочтения и ограничения в поведении партнёра. Параметры SETTINGS по-отдельности называют также установками (setting), а идентификатор и значение «идентификатором» и «значением» установки.

Кадры SETTINGS всегда применяются к соединению HTTP/3 в целом, а не к отдельным потокам. Кадр SETTINGS **должен** быть первым кадром в каждом потоке управления (см. параграф 6.2.1) каждого партнёра, а последующая передача таких кадров **недопустима**. Получив второй кадр SETTINGS в потоке управления, конечная точка **должна** ответить ошибкой соединения типа H3_FRAME_UNEXPECTED.

Кадры SETTINGS **недопустимо** передавать в каких-либо потоках, кроме потока управления. Получив такой кадр в ином потоке, конечная точка **должна** возвращать ошибку соединения типа H3_FRAME_UNEXPECTED.

Параметры SETTINGS не согласуются, они описывают характеристики передающего партнёра, которые могут использоваться принимающим партнёром. Однако применение кадров SETTINGS может предполагать согласование - каждый партнёр применяет SETTINGS для анонсирования набора поддерживаемых им значений. Определение установки описывает, как каждый из партнёров комбинирует свои и партнерские параметры для выбора используемого значения. Кадр SETTINGS не предоставляет механизма оповещения о вступлении настроек в силу.

Каждый партнёр может анонсировать своё значение конкретного параметра. Например, клиент может быть готов использовать большой раздел заголовков, а сервер может более осторожно подойти к выбору размера запросов. Один идентификатор установки (параметра) **недопустимо** включать в кадр SETTINGS более одного раза. Получатель **может** считать наличие дубликатов идентификаторов ошибкой соединения типа H3_SETTINGS_ERROR.

Содержимое (payload) кадра SETTINGS может включать параметры, каждый из которых состоит из идентификатора и значения, представляемых в формате QUIC для целых чисел с переменным размером.

```
Setting {
  Identifier (i),
  Value (i),
}

SETTINGS Frame {
  Type (i) = 0x04,
  Length (i),
  Setting (..) ...,
}
```

Рисунок 7. Кадр SETTINGS.

Реализации **должны** игнорировать параметры с неизвестными (непонятными) идентификаторами.

7.2.4.1. Параметр SETTINGS

В HTTP/3 определена одна настройка (параметр).

SETTINGS_MAX_FIELD_SECTION_SIZE (0x06)

По умолчанию значение не ограничено. Применение описано в параграфе 4.2.2.

Идентификаторы настроек в формате 0x1f * N + 0x21 с неотрицательными значениями N зарезервированы для выполнения требования игнорировать неизвестные идентификаторы. Такие настройки не имеют значения. Конечной точке **следует** включать хотя бы одну такую настройку в кадр SETTINGS. Конечным точкам **недопустимо** придавать смысл зарезервированным настройкам. Поскольку эти настройки не имеют смысла, реализация может выбирать для них произвольные значения.

Идентификаторы настроек из [HTTP/2], для которых нет соответствующей настройки HTTP/3, также являются зарезервированными (параграф 11.2.2). Эти настройки **недопустимо** передавать, а их получение **должно** считаться ошибкой соединения типа H3_SETTINGS_ERROR.

В расширениях HTTP/3 могут определяться дополнительные настройки (см. раздел 9).

7.2.4.2. Инициализация

Реализации HTTP **недопустимо** передавать кадры или запросы, непригодные в соответствии с текущим пониманием настроек партнёра.

Все установки начинаются с исходного значения. Каждой конечной точке **следует** применять исходные значения при отправке сообщений до получения кадра SETTINGS, поскольку пакеты с настройками могут быть потеряны или задержаны. При поступлении кадра SETTINGS для всех настроек устанавливаются новые значения. Это избавляет от необходимости ждать приёма кадра SETTINGS перед отправкой сообщений. Конечным точкам **недопустимо**

требовать получения от партнёра каких-либо данных до отправки кадра SETTINGS, установки **должны** передаваться, как только транспорт будет готов к передаче данных. Для сервера начальным значением каждой установки клиента является принятое по умолчанию значение. Для клиентов, использующих соединение QUIC 1-RTT, начальным значением каждой установки сервера является принятое по умолчанию значение. Ключи 1-RTT всегда становятся доступными до того, как пакет с кадром SETTINGS будет обработан QUIC, даже если сервер передаст SETTINGS незамедлительно. Клиентам **не следует** дожидаться получения SETTINGS перед отправкой запросов, но **следует** обрабатывать полученные дейтаграммы, чтобы повысить вероятность обработки SETTINGS до отправки первого запроса. При использовании соединения QUIC 0-RTT начальным значением каждой установки сервера является значение из предыдущей сессии. Клиентам **следует** сохранять установки, представленные сервером в соединении HTTP/3, где были предоставлены сведения о восстановлении, но в некоторых случаях (например, при получении сеансовой квитанции до кадра SETTINGS) установки **можно** не сохранять. Клиент **должен** соблюдать сохранённые установки (или принятые по умолчанию значения, если установки не сохранены) при попытке 0-RTT. Если сервер предоставил новые установки, клиент **должен** соблюдать их.

Сервер может запоминать анонсированные установки или сохранять копию значений с защитой целостности в квитанции и восстанавливать сведения при получении данных 0-RTT. Сервер использует значения установок HTTP/3 при решении вопроса о восприятии данных 0-RTT. Если сервер не может определить, что запомненные клиентом установки совместимы с его текущими установками, серверу **недопустимо** воспринимать данные 0-RTT. Запомненные установки считаются совместимыми, если соблюдающий их клиент не нарушает текущих установок сервера.

Сервер **может** воспринять 0-RTT и впоследствии предоставить другие установки в своём кадре SETTINGS. Если данные 0-RTT восприняты сервером, в его кадре SETTINGS **недопустимо** снижать какие-либо пределы или менять значения, которые могут быть нарушены клиентом в соответствии с его данными 0-RTT. Сервер **должен** включить все установки, которые отличаются от принятых по умолчанию значений. Если сервер воспринимает 0-RTT, а затем передаёт установки, не совместимые с заданными ранее, это **должно** считаться ошибкой соединения типа H3_SETTINGS_ERROR. Если сервер воспринимает 0-RTT, а затем передаёт кадр SETTINGS, где отсутствует понятное клиенту значение (помимо зарезервированных идентификаторов установок), для которого было установлено отличное от принятого по умолчанию значение, это **должно** считаться ошибкой соединения типа H3_SETTINGS_ERROR.

7.2.5. PUSH_PROMISE

Кадр PUSH_PROMISE (type=0x05) служит для передачи раздела заголовков предсказанного (promise) запроса от сервера клиенту в потоке запросов.

```
PUSH_PROMISE Frame {
  Type (i) = 0x05,
  Length (i),
  Push ID (i),
  Encoded Field Section (..),
}
```

Рисунок 8. Кадр PUSH_PROMISE.

Поля содержимого (payload) кадра описаны ниже.

Push ID
: Целое число с переменным размером, указывающее операцию выталкивания с сервера. Идентификатор применяется в заголовках push-потоков (параграф 4.6) и кадрах CANCEL_PUSH.

Encoded Field Section
: Поля заголовка с кодированием QPACK [QPACK] для предсказанного (promised) запроса.

Серверу **недопустимо** использовать Push ID больше значения, указанного клиентом в кадре MAX_PUSH_ID (параграф 7.2.7). Клиент **должен** считать получение кадра PUSH_PROMISE с таким значением Push ID ошибкой соединения типа H3_ID_ERROR.

Сервер **может** использовать одно значение Push ID в нескольких кадрах PUSH_PROMISE. В этом случае распакованные наборы заголовков запросов **должны** содержать те же поля в том же порядке и все значения полей **должны** совпадать. Клиентам **следует** сравнивать разделы заголовков для ресурсов, обещанных неоднократно. Если клиент получает уже обещанный Push ID и обнаруживает несоответствие, он **должен** отвечать ошибкой соединения типа H3_GENERAL_PROTOCOL_ERROR. Если распакованные разделы полей совпадают, клиенту **следует** связывать вытолкнутое содержимое с каждым потоком, где был получен кадр PUSH_PROMISE.

Разрешение дублирования ссылок на один Push ID предназначено в первую очередь для сокращения дублирования, вызываемого одновременными запросами. Серверу **следует** избегать долгосрочного использования Push ID. Клиенты, скорей всего, будут потреблять вытолкнутые сервером отклики, не сохраняя их для повторного использования в течение долгого времени. Клиенты, встречающиеся с кадром PUSH_PROMISE, содержащим уже использованный и отброшенный Push ID, вынуждены будут игнорировать это обещание.

При получении кадра PUSH_PROMISE в потоке управления клиент **должен** возвращать ошибку соединения типа H3_FRAME_UNEXPECTED.

Клиенту **недопустимо** передавать кадр PUSH_PROMISE, а сервер **должен** считать получение PUSH_PROMISE ошибкой соединения типа H3_FRAME_UNEXPECTED.

Общее описание механизма выталкивания с сервера приведено в параграфе 4.6.

7.2.6. GOAWAY

Кадр GOAWAY (type=0x07) служит для инициирования аккуратного завершения соединения HTTP/3 любой из конечных точек. GOAWAY позволяет конечной точке прекратить восприятие новых запросов или выталкиваний, сохраняя обработку полученных ранее. Это позволяет выполнять административные операции, такие как обслуживание сервера. Кадр GOAWAY сам по себе не закрывает соединение.

```
GOAWAY Frame {
  Type (i) = 0x07,
  Length (i),
  Stream ID/Push ID (i),
}
```

Рисунок 9. Кадр GOAWAY.

Кадр GOAWAY всегда передаётся в пакете управления. В направлении от сервера к клиенту кадр содержит идентификатор потока QUIC, инициированного клиентом двухстороннего потока, в виде целого числа с переменным размером. Клиент **должен** считать получение GOAWAY с идентификатором потока иного типа ошибкой соединения типа H3_ID_ERROR. В направлении от клиента к серверу кадр GOAWAY содержит идентификатор выталкивания в виде целого числа с переменным размером.

Кадр GOAWAY относится ко всему соединению, а не к отдельному потоку. Клиент **должен** считать получение GOAWAY в потоке, отличном от потока управления, ошибкой соединения типа H3_FRAME_UNEXPECTED.

Дополнительные сведения о кадрах GOAWAY приведены в параграфе 5.2.

7.2.7. MAX_PUSH_ID

Кадр MAX_PUSH_ID (type=0x0d) используется клиентом для управления числом выталкиваний (push), которые может инициировать сервер, путём задания максимального значения Push ID, разрешённого серверу в кадрах PUSH_PROMISE и CANCEL_PUSH. Это ограничивает число инициируемых сервером потоков выталкивания в дополнением к ограничению, задаваемому транспортном QUIC.

Кадр MAX_PUSH_ID всегда передаётся в потоке управления и его получение в ином потоке **должно** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED. Серверу **недопустимо** передавать кадры MAX_PUSH_ID и клиент **должен** считать получение такого кадра ошибкой соединения типа H3_FRAME_UNEXPECTED.

```
MAX_PUSH_ID Frame {
  Type (i) = 0x0d,
  Length (i),
  Push ID (i),
}
```

Рисунок 10. Кадр MAX_PUSH_ID.

Максимальный идентификатор выталкивания ещё не задан в момент организации соединения HTTP/3 и сервер не может применять выталкивание до получения кадра MAX_PUSH_ID. Клиент, желающий управлять числом предсказанных сервером выталкиваний, может увеличивать максимальный идентификатор, передавая кадры MAX_PUSH_ID по мере получения выталкиваний от сервера или их отмены сервером.

Кадр MAX_PUSH_ID содержит одно целое число с переменным размером, указывающее максимальное значение Push ID, разрешённое для сервера (см. параграф 4.6). Кадр MAX_PUSH_ID не может снижать значение максимального идентификатора, установленное ранее для соединения, и получение такого кадра MAX_PUSH_ID **должно** считаться ошибкой соединения типа H3_ID_ERROR.

7.2.8. Зарезервированные типы кадров

Кадры типов 0x1f * N + 0x21 с неотрицательными значениями N зарезервированы для выполнения требования игнорировать неизвестные типы (раздел 9). Эти кадры не имеют семантики и **могут** передаваться в любом потоке, где передача кадров разрешена. Конечным точкам **недопустимо** придавать таким кадрам какое-либо значение. Содержимое (payload) и размер кадров определяется реализацией.

Кадры типов, использованных в HTTP/2, для которых нет соответствующего кадра HTTP/3, также являются зарезервированными (параграф 11.2.1). Кадры этих типов **недопустимо** передавать, а их получение **должно** считаться ошибкой соединения типа H3_FRAME_UNEXPECTED.

8. Обработка ошибок

Когда поток не удаётся закрыть успешно, QUIC разрешает приложению резко прервать (сбросить - reset) поток и сообщить причину этого (см. параграф 2.4 в [QUIC-TRANSPORT]). Это называется ошибкой потока (stream error). Реализация HTTP/3 может принять решение о закрытии соединения QUIC и указать тип ошибки. Формат передачи кодов ошибок описан в параграфе 8.1. Ошибки потока отличаются от кодов статуса HTTP, которые указывают условия ошибки. Ошибка потока говорит, что её отправитель не передал или не воспринял запрос или отклик, а код статуса HTTP указывает результат успешно полученного запроса.

Для прерывания соединения целиком QUIC предоставляет похожие механизмы указания причины (см. параграф 5.3 в [QUIC-TRANSPORT]). Это называется ошибкой соединения (connection error). Как и при ошибках потока, реализация HTTP/3 может разорвать соединение QUIC и сообщить причину в коде ошибки (параграф 8.1).

Хотя причины закрытия потоков и соединений называются ошибками, они не всегда говорят о проблеме с соединением или реализацией. Например, поток может быть сброшен, если запрошенный ресурс больше не нужен.

Конечная точка **может** в некоторых случаях считать ошибку потока ошибкой соединения, закрывая соединение целиком при возникновении ошибки в отдельном потоке. Реализации нужно учитывать влияние этого на оставшиеся запросы.

Поскольку новые коды ошибок могут быть заданы без согласования (см. раздел 9), применение кода ошибки в неожиданном контексте или неизвестный код ошибки **должны** считаться эквивалентом H3_NO_ERROR. Однако закрытие потока может иметь и другие последствия, независимо от кода ошибки (см., например, параграф 4.1).

8.1. Коды ошибок HTTP/3

Перечисленные ниже коды ошибок определены для использования при резком прерывании потоков, прерывании чтения из потока и незамедлительного закрытия соединений HTTP/3.

H3_NO_ERROR (0x0100)
: Нет ошибок. Этот код применяется, когда нужно закрыть поток или соединение, но нет ошибки для её указания.

H3_GENERAL_PROTOCOL_ERROR (0x0101)
: Партнёр нарушил требования протокола, но нет более конкретного кода для указания этого или конечная точка отказывается использовать такой код.

H3_INTERNAL_ERROR (0x0102)
: Внутренняя ошибка в стеке HTTP.

H3_STREAM_CREATION_ERROR (0x0103)
: Конечная точка обнаружила, что партнёр создал неприемлемый для неё поток.

H3_CLOSED_CRITICAL_STREAM (0x0104)
: Поток, требуемый соединением HTTP/3, был закрыт или сброшен.

H3_FRAME_UNEXPECTED (0x0105)
: Получен кадр, не разрешенный в текущем состоянии или потоке.

H3_FRAME_ERROR (0x0106)
: Получен кадр, нарушающий требования к компоновке или размеру.

H3_EXCESSIVE_LOAD (0x0107)
: Конечная точка обнаружила, что поведение партнёра может вызывать чрезмерную нагрузку.

H3_ID_ERROR (0x0108)
: Некорректное использование идентификатора потока или выталкивания, например, превышение предела, снижение предела или повторное использование.

H3_SETTINGS_ERROR (0x0109)
: Конечная точка обнаружила ошибку в содержимом кадра SETTINGS.

H3_MISSING_SETTINGS (0x010a)
: В начале потока управления не был получен кадр SETTINGS.

H3_REQUEST_REJECTED (0x010b)
: Сервер отклонил запрос без обработки в приложении.

H3_REQUEST_CANCELLED (0x010c)
: Запрос или отклик на него (включая выталкиваемый) был отменен.

H3_REQUEST_INCOMPLETE (0x010d)
: Поток клиента прервался без получения полного запроса.

H3_MESSAGE_ERROR (0x010e)
: Сообщение HTTP имеет некорректную форму и не может быть обработано.

H3_CONNECT_ERROR (0x010f)
: Соединение TCP, созданное в ответ на запрос CONNECT, сброшено или аварийно закрыто.

H3_VERSION_FALLBACK (0x0110)
: Запрошенную операцию невозможно выполнить по протоколу HTTP/3. Партнёру следует повторить её по протоколу HTTP/1.1.

Коды ошибок 0x1f * N + 0x21 с неотрицательными значениями N зарезервированы для выполнения требования считать неизвестные коды эквивалентом H3_NO_ERROR (раздел 9). Реализации **следует** с некоторой вероятностью выбирать коды из этого пространства, когда можно было передать H3_NO_ERROR.

9. Расширения HTTP/3

Протокол HTTP/3 допускает расширения. В рамках заданных этим разделом ограничений расширения протокола могут применяться для предоставления дополнительных услуг или изменения любых аспектов протокола. Расширение действует лишь в рамках конкретного соединения HTTP/3. Это относится к элементам протокола, заданным в этом документе и не влияет на имеющиеся возможности расширения HTTP, такие как задание новых методов, кодов статуса и полей.

Расширениям разрешается применять новые типы кадров (параграф 7.2), установки (параграф 7.2.4.1), коды ошибок (раздел 8), типы односторонних потоков (параграф 6.2). Для точек расширения созданы реестры типов кадров (параграф 11.2.1), установок (параграф 11.2.2), кодов ошибок (параграф 11.2.3) и типов потоков (параграф 11.2.4).

Реализации **должны** игнорировать неизвестные и неподдерживаемые значения во всех расширяемых элементах протокола, а также **должны** отбрасывать данные или прерывать чтение односторонних потоков неизвестных и неподдерживаемых типов. Это означает, что любая точка расширения может безопасно использоваться расширениями без предварительной договорённости или согласования. Однако если известный тип кадр должен размещаться в определённом месте (как кадр SETTINGS, являющийся первым в потоке управления, см. параграф 6.2.1), неизвестный тип кадра, нарушающий это требование, **следует** считать ошибкой.

Расширения, способные менять семантику имеющихся компонентов протокола, **должны** согласовываться перед использованием. Например, расширение, меняющее структуру кадра HEADERS, не может применяться без согласования с партнёром. Координация вступления в силу такого изменения структуры может оказаться сложной, поэтому выделение новых идентификаторов для новых определений будет, вероятно, более эффективной мерой.

Этот документ не задаёт конкретного метода согласования применения расширений, но указывает, что для этого можно использовать настройки (параграф 7.2.4.1). Если оба партнёра устанавливают значение, указывающее желание применять расширения, это расширение может использоваться. Если для согласования расширения применяется настройка, принятое по умолчанию значение **должно** быть задано так, чтобы расширение было отключено при отсутствии этой установки.

10. Вопросы безопасности

Соображения безопасности для HTTP/3 сопоставимы с принятыми для HTTP/2 с TLS. Многие из вопросов, рассмотренных в разделе 10 [HTTP/2], применимы к [QUIC-TRANSPORT] и рассматриваются в этом документе.

10.1. Полномочия сервера

HTTP/3 полагается на определение полномочности HTTP. Вопросы безопасности, связанные с полномочиями, рассмотрены в параграфе 17.1 [HTTP].

10.2. Кросс-протокольные атаки

Применение ALPN в согласовании TLS и QUIC устанавливает целевой транспортный протокол до начала обработки байтов прикладного уровня. Это обеспечивает конечным точкам уверенность в том, что они используют один протокол. Однако это не обеспечивает защиты от кросс-протокольных атак. В параграфе 21.5 [QUIC-TRANSPORT] описаны некоторые способы использования открытого текста (plaintext) из пакетов QUIC для подделки запросов к конечным точкам, не использующим транспорт с проверкой подлинности.

10.3. Атаки на инкапсуляцию в посредниках

Кодирование полей HTTP/3 позволяет выражать имена, не соответствующие синтаксису имён полей HTTP (параграф 5.1 в [HTTP]). Запросы и отклики с недействительными именами полей **должны** считаться ошибочными. Поэтому посредники не могут преобразовать запрос или отклик HTTP/3 с недействительными именами полей в сообщение HTTP/1.1.

HTTP/3 может передавать значения полей, которые не являются действительными. Хотя большинство значений, которые могут быть закодированы, не повлияют на разбор поля, символы возврата каретки (ASCII 0x0d), перевода строки (ASCII 0x0a) и null-символ (ASCII 0x00) могут использоваться атакующим, если они будут транслироваться буквально. Любой запрос или отклик с недействительным символом в значении поля **должен** считаться ошибочным. Действительные символы определяет правило ABNF field-content из параграфа 5.5 и [HTTP].

10.4. Кэшируемость вытолкнутых откликов

Вытолкнутые отклики не имеют явного запроса от клиента, запрос предоставляется сервером в кадре PUSH_PROMISE.

Кэширование вытолкнутых откликов возможно на основе рекомендаций от сервера-источника в поле заголовка Cache-Control. Однако это может вызывать проблемы при размещении на сервере более одного арендатора (например, сервер может предоставлять нескольким пользователям по небольшой части своего пространства URI). При использовании серверного пространства несколькими арендаторами этот сервер **должен** гарантировать, что арендаторы не могут выталкивать представления ресурсов, для которых у них нет полномочий. Отказ от выполнения этого правила позволит арендатору выдавать представление, которое будет обслуживаться в кэше, переопределяя фактическое представление от полномочного арендатора.

Клиенты должны отклонять вытолкнутые отклики, для которых сервер-источник не имеет полномочий (параграф 4.6).

10.5. DoS-атаки

Для работы соединения HTTP/3 может запрашиваться больше ресурсов, чем для HTTP/1.1 или HTTP/2. Использование сжатия полей и управления потоком данных зависит от выделения ресурсов для хранения большого объёма состояний. Настройки для этих функций обеспечивают строгое ограничение используемого объёма памяти.

Число кадров PUSH_PROMISE ограничивается похожим способом. Клиенту, воспринимающему выталкивание с сервера, **следует** ограничивать число идентификаторов (Push ID), выталкиваемых за один раз.

Пропускную способность при обработке невозможно защитить также эффективно, как объем хранимых состояний.

Возможностью отправлять неопределённые элементы протокола, которые партнёр не обязан игнорировать, может быть использована для того, чтобы вынудить партнёра затрачивать дополнительное время на обработку. Это можно сделать путём передачи множества неопределённых параметром SETTINGS, неизвестных типов кадров или потоков. Следует отметить, что некоторые из таких случаев вполне корректны, например, необязательные для понимания расширения или заполнение для повышения устойчивости к анализу трафика.

Сжатие раздела полей также предоставляет некоторые возможности дополнительного расхода ресурсов при обработке, более подробно описанные в разделе 7 [QPACK].

Все эти возможности - выталкивание с сервера, неизвестные элементы протокола, сжатие полей - могут применяться корректно и становятся вредными лишь при ненужном или избыточном применении. Конечные точки, не отслеживающие такое поведение, подвергают себя риску DoS-атак. Реализациям **следует** отслеживать и ограничивать использование таких возможностей. Конечная точка **может** считать подозрительную активность ошибкой соединения типа H3_EXCESSIVE_LOAD, но ложные срабатывания будут нарушать действительные соединения и запросы.

10.5.1. Ограничения размера раздела заголовков

Большой раздел полей (параграф 4.1) может вынудить реализацию зафиксировать большой объем состояния. Важные для маршрутизации строки полей могут размещаться в конце блока, что будет мешать передаче потока полей конечному адресату. Такое упорядочивание и другие причины, например, обеспечение корректности кэша, означают, что конечной точке может потребоваться целиком буферизовать блок полей. Поскольку размер блока полей жёстко не ограничен, некоторым конечным точкам может потребоваться большой объём памяти для блока полей.

Конечная точка может использовать установку SETTINGS_MAX_FIELD_SECTION_SIZE (параграф 4.2.2) для информирования своих партнёров о возможных ограничениях размера несжатого раздела полей. Эта настройка является лишь рекомендацией, поэтому конечные точки **могут** передавать блок полей, размер которого превышает этот предел, рискуя тем, что запрос или отклик будет сочтён ошибочным. Этот параметр относится к соединению HTTP/3 и любой запрос или отклик может столкнуться в пути с более низким и неизвестным ограничением. Посредник может попытаться избежать этой проблемы, передавая значения, представленные разными партнёрами, но не обязан делать этого.

Сервер, получивший блок полей, размер которого превышает тот, который сервер готов обработать, может передать код статуса HTTP 431 (Request Header Fields Too Large) [RFC6585]. Клиент может отбрасывать отклики, которые он не может обработать.

10.5.2. Проблемы CONNECT

Метод CONNECT можно использовать для создания непропорциональной нагрузки на прокси, поскольку создание потока относительно недорого по сравнению с созданием и поддержкой соединения TCP. Поэтому прокси с поддержкой CONNECT может быть более консервативным в части числа воспринимаемых одновременно запросов. Прокси может также поддерживать некоторые ресурсы для соединения TCP после закрытия потока, в котором передан запрос CONNECT, поскольку исходящее соединение TCP остаётся в состоянии TIME_WAIT. С учётом этого прокси может задерживать повышение пределов для соединения QUIC в течение некоторого времени послезакрытия соединения TCP.

10.6. Использование сжатия

Сжатие может позволить атакующему восстановить секретные данные, если они сжаты в одном контексте с данными, контролируемыми им. HTTP/3 разрешает сжатие строк полей (параграф 4.2), а приведённые ниже соображения относятся также к использованию в HTTP кодирования содержимого со сжатием (параграф 8.4.1 в [HTTP]).

Существуют наглядные атаки на сжатие, использующие свойства Web (например, [BREACH]). Атакующий создаёт множество запросов с различными открытыми данными (plaintext), наблюдая для каждого размер полученного шифротекста (ciphertext), который будет более коротким при угаданном секрете.

Реализациям, обменивающимся данными по защищённому каналу, **недопустимо** сжимать содержимое, включающее как конфиденциальные, так и контролируемые злоумышленником данные, если только для каждого источника данных не применяются свои словари. **Недопустимо** сжимать данные, если их источник невозможно определить достоверно.

Дополнительные соображения безопасности применительно к сжатию полей представлены в [QPACK].

10.7. Заполнение и анализ трафика

Заполнение может служить для сокрытия точного размера содержимого кадра и предназначено для смягчения определённых атак в HTTP, например, атак, где сжатое содержимое включает контролируемый злоумышленником открытый текст и секретные данные (например, [BREACH]).

Если HTTP/2 использует поля Padding[1] в некоторых кадрах для повышения устойчивости соединения к анализу трафика, то HTTP/3 может полагаться на заполнение транспортного уровня или использовать зарезервированные типы кадров и потоков, описанные в параграфах 7.2.8 и 6.2.3. Эти методы заполнения дают разные результаты с точки зрения детализации заполнения, его размещения относительно защищаемой информации, применения заполнения в случае потери пакетов и способа управления заполнением в реализации.

Зарезервированные типы кадров и потоков можно применять для создания видимости передачи трафика при простое соединения. Поскольку трафик HTTP зачастую имеет характер всплесков (burst), видимый трафик может служить для маскировки времени и продолжительности таких всплесков, вплоть до создания иллюзии постоянного потока данных. Однако для такого трафика все равно применяется управление потоком данных со стороны получателя и неспособность своевременно «осушить» фиктивный трафик и предоставить дополнительный кредит на передачу может помешать отправке реального трафика.

Для смягчения атак, основанных на сжатии, запрет или ограничение сжатия может оказаться более предпочтительной мерой, нежели заполнение.

Использование заполнения может предоставлять более слабую защиту, чем кажется на первый взгляд, а избыточное заполнение даже может быть вредным. В лучшем случае заполнение лишь усложняет для атакующего получение сведений о размере за счёт увеличения числа кадров, которые тому приходится наблюдать. Некорректно реализованные схемы заполнения легко преодолеваются. В частности, случайное заполнение с предсказуемым распределением обеспечивает лишь очень слабую защиту, а заполнение до определённого размера раскрывает сведения о размере, когда кадр превосходит заданный предел, что может быть полезно для злоумышленника, контролирующего открытый текст.

10.8. Разбор кадров

Некоторые элементы протокола содержат поля размера (обычно явное указание размера в кадрах с целыми числами переменного размера). Это может создавать риск при неосторожной реализации. Реализация **должна** гарантировать точное совпадение указанного в кадре размера с фактическим размером содержащихся в кадре полей.

10.9. Ранние данные

Использование 0-RTT с HTTP/3 открывает возможность для атак с воспроизведением (replay). **Должны** применяться средства смягчения, описанные в [HTTP-REPLAY], при использовании HTTP/3 с 0-RTT. При использовании [HTTP-REPLAY] для HTTP/3 ссылки на уровень TLS относятся к согласованию, выполняемому в QUIC, а ссылки на данные приложения - к содержимому потоков.

10.10. Смена адреса

Некоторые реализации HTTP используют адрес клиента для протоколирования (log) или контроля доступа. Поскольку адрес клиента в QUIC может меняться в процессе действия соединения (а в будущих версиях может поддерживаться использование нескольких адресов), таким реализациям потребуется активно извлекать текущий адрес (или адреса) клиента, когда это актуально, или явно разрешать возможность смены исходного адреса.

10.11. Вопросы приватности

Некоторые особенности HTTP/3 дают наблюдателю возможность составить временную картину действий клиента или сервера. Это включает значения настроек, время реакции на воздействия и работу функций, управляемых

[1] В оригинале ошибочно указан ещё кадр PADDING, см. https://www.rfc-editor.org/errata/eid7702. *Прим. перев.*

настройками. По мере возникновения наблюдаемых различий в поведении эти свойства могут стать основой для «отпечатка» конкретного клиента

Предпочтение HTTP/3 использовать одно соединение QUIC позволяет отслеживать активность пользователя на сайте. Повторное использование соединений для других источников позволяет отслеживать эти источники.

Некоторые функции QUIC требуют немедленного отклика и могут использоваться конечной точкой для измерения задержки до своего партнёра, что может в некоторых случаях влиять на приватность.

11. Взаимодействие с IANA

Этот документ регистрирует новый идентификатор протокола ALPN (параграф 11.1) и создаёт новые реестры для кодов HTTP/3.

11.1. Регистрация строки идентификации HTTP/3

Этот документ задаёт новую регистрацию для идентификации HTTP/3 в реестре TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs, созданном в [RFC7301].

```
Протокол:  HTTP/3
Строка идентификации:  0x68 0x33 ("h3")
Спецификация:  Этот документ
```

11.2. Новые реестры

В новых реестрах, созданных этим документом, применяется политика регистрации QUIC, заданная в параграфе 22.1 [QUIC-TRANSPORT]. Все эти реестры включают общий набор полей, указанный в параграфе 22.1.1 [QUIC-TRANSPORT] и собраны по общим заголовком Hypertext Transfer Protocol version 3 (HTTP/3).

Все исходные значения в этих реестрах имеют статус постоянных (permanent) и включают IETF в качестве контролёра изменений, а также контактный адрес рабочей группы HTTP (ietf-http-wg@w3.org).

11.2.1. Типы кадров

Этот документ создаёт реестр кодов типа кадров HTTP/3 Frame Types с 62-битовым пространством с политикой регистрации QUIC (см. параграф 11.2). Постоянные регистрации в этом реестре выполняются по процедуре Specification Required ([RFC8126]), за исключением диапазона 0x00 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфами 4.9 и 4.10 в [RFC8126].

Хотя этот реестр отделен от реестра HTTP/2 Frame Type, заданного в [HTTP/2], предпочтительно выполнять назначения параллельно при перекрытии пространств кодов. Если запись присутствует лишь в одном реестре, **следует** приложить все усилия, чтобы избежать соответствующего значения для несвязанной операции. Эксперты **могут** отклонять несвязанные регистрации, конфликтующие с таким же значением в соответствующем реестре.

В дополнение к общим полям, указанным в параграфе 11.2, постоянные записи в этом реестре **должны** включать указанное ниже поле.

Frame Type
: Имя или метка для типа кадра.

Спецификация типа кадра **должна** содержать описание структуры кадра и его семантики, включая любые части кадра с условным присутствием.

Этот документ регистрирует записи, приведённые в таблице 2.

Таблица 2. Исходные типы кадров HTTP/3.

Тип кадра	***Значение***	***Спецификация***
DATA	0x00	параграф 7.2.1
HEADERS	0x01	параграф 7.2.2
Reserved	0x02	этот документ
CANCEL_PUSH	0x03	параграф 7.2.3
SETTINGS	0x04	параграф 7.2.4
PUSH_PROMISE	0x05	параграф 7.2.5
Reserved	0x06	этот документ
GOAWAY	0x07	параграф 7.2.6
Reserved	0x08	этот документ
Reserved	0x09	этот документ
MAX_PUSH_ID	0x0d	параграф 7.2.7

Коды вида 0x1f * N + 0x21 для неотрицательный целых значений N (0x21, 0x40, ..., 0x3fffffffffffffe) **недопустимо** выделять агентству IANA и **недопустимо** включать в список выделенных значений.

11.2.2. Установки

Этот документ создаёт реестр установок (параметров) HTTP/3 Settings с 62-битовым пространством с политикой регистрации QUIC (см. параграф 11.2). Постоянные регистрации в этом реестре выполняются по процедуре Specification Required ([RFC8126]), за исключением диапазона 0x00 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфами 4.9 и 4.10 в [RFC8126].

Хотя этот реестр отделен от реестра HTTP/2 Settings [HTTP/2], предпочтительно выполнять назначения параллельно при перекрытии пространств кодов. Если запись присутствует лишь в одном реестре, **следует** приложить все усилия, чтобы избежать соответствующего значения для несвязанной операции. Эксперты **могут** отклонять несвязанные регистрации, конфликтующие с таким же значением в соответствующем реестре.

В дополнение к общим полям, указанным в параграфе 11.2, постоянные записи в этом реестре **должны** включать указанные ниже поля.

Setting Name
Символьное имя параметра (необязательно).

Default
Значение параметра при отсутствии явного указания. По умолчанию **следует** задавать наиболее ограничительное из возможных значений.

Этот документ регистрирует записи, приведённые в таблице 3.

Таблица 3. Исходные установки HTTP/3.

Имя параметра	*Значение*	*Спецификация*	*Значение по умолчанию*
Reserved	0x00	этот документ	N/A
Reserved	0x02	этот документ	N/A
Reserved	0x03	этот документ	N/A
Reserved	0x04	этот документ	N/A
Reserved	0x05	этот документ	N/A
MAX_FIELD_SECTION_SIZE	0x06	параграф 7.2.4.1	Unlimited

В целях форматирования префикс SETTINGS_ в именах можно не указывать.

Коды вида 0x1f * N + 0x21 для неотрицательный целых значений N (0x21, 0x40, ..., 0x3fffffffffffffe) **недопустимо** выделять агентству IANA и **недопустимо** включать в список выделенных значений.

11.2.3. Коды ошибок

Этот документ создаёт реестр кодов ошибок HTTP/3 Error Codes с 62-битовым пространством с политикой регистрации QUIC (см. параграф 11.2). Постоянные регистрации в этом реестре выполняются по процедуре Specification Required ([RFC8126]), за исключением диапазона 0x00 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфами 4.9 и 4.10 в [RFC8126].

Регистрация должна включать описание кода ошибки. Экспертам рекомендуется проверять новые регистрации на предмет дублирования имеющихся кодов. Применение имеющихся кодов рекомендуется, но не требуется. Использовать значения из реестра HTTP/2 Error Code не рекомендуется и эксперты **могут** отклонять такие регистрации.

В дополнение к общим полям, указанным в параграфе 11.2, постоянные записи в этом реестре **должны** включать указанные ниже поля.

Name
Имя кода ошибки.

Description
Краткое описание семантики кода.

Этот документ регистрирует записи, приведённые в таблице 4. Эти коды были выбраны из диапазона, требующего использовать процедуру Specification Required для исключения конфликтов с кодами ошибок HTTP/2.

Таблица 4. Исходные коды ошибок HTTP/3.

Имя	*Значение*	*Описание*	*Спецификация*
H3_NO_ERROR	0x0100	Нет ошибок	параграф 8.1
H3_GENERAL_PROTOCOL_ERROR	0x0101	Протокольная ошибка общего типа	параграф 8.1
H3_INTERNAL_ERROR	0x0102	Внутренняя ошибка	параграф 8.1
H3_STREAM_CREATION_ERROR	0x0103	Ошибка при создании потока	параграф 8.1
H3_CLOSED_CRITICAL_STREAM	0x0104	Закрыт критически важный поток	параграф 8.1
H3_FRAME_UNEXPECTED	0x0105	Кадр не разрешён в текущем состоянии	параграф 8.1
H3_FRAME_ERROR	0x0106	Кадр нарушает требования к схеме или размеру	параграф 8.1
H3_EXCESSIVE_LOAD	0x0107	Партнёр создаёт чрезмерную нагрузку	параграф 8.1
H3_ID_ERROR	0x0108	Идентификатор применён некорректно	параграф 8.1
H3_SETTINGS_ERROR	0x0109	Непригодное значение в кадре SETTINGS	параграф 8.1
H3_MISSING_SETTINGS	0x010a	Не получен кадр SETTINGS	параграф 8.1
H3_REQUEST_REJECTED	0x010b	Запрос не обработан	параграф 8.1
H3_REQUEST_CANCELLED	0x010c	Данные больше не требуются	параграф 8.1
H3_REQUEST_INCOMPLETE	0x010d	Поток закрыт рано	параграф 8.1
H3_MESSAGE_ERROR	0x010e	Некорректно сформированное сообщение	параграф 8.1
H3_CONNECT_ERROR	0x010f	Сброс TCP или ошибка в запросе CONNECT	параграф 8.1
H3_VERSION_FALLBACK	0x0110	Повтор через HTTP/1.1	параграф 8.1

Коды вида 0x1f * N + 0x21 для неотрицательный целых значений N (0x21, 0x40, ..., 0x3fffffffffffffe) **недопустимо** выделять агентству IANA и **недопустимо** включать в список выделенных значений.

11.2.4. Типы потоков

Этот документ создаёт реестр типов односторонних потоков HTTP/3 Stream Types с 62-битовым пространством с политикой регистрации QUIC (см. параграф 11.2). Постоянные регистрации в этом реестре выполняются по процедуре Specification Required ([RFC8126]), за исключением диапазона 0x00 0x3f (включительно), где применяется процедура Standards Action или IESG Approval в соответствии с параграфами 4.9 и 4.10 в [RFC8126].

В дополнение к общим полям, указанным в параграфе 11.2, постоянные записи в этом реестре **должны** включать указанные ниже поля.

Stream Type
Имя или метка типа потока.

Sender
Конечная точка, которая может инициировать поток этого типа (Client, Server, Both).

Спецификация для постоянной регистрации **должна** содержать описание типа потока, включая структуру и семантику его содержимого.

Этот документ регистрирует записи, приведённые в таблице 5.

Таблица 5. Исходные типы потоков.

Тип потока	***Значение***	***Спецификация***	***Отправитель***
Control Stream	0x00	параграф 6.2.1	Оба
Push Stream	0x01	параграф 4.6	Сервер

Коды вида 0x1f * N + 0x21 для неотрицательный целых значений N (0x21, 0x40, ..., 0x3fffffffffffffe) **недопустимо** выделять агентству IANA и **недопустимо** включать в список выделенных значений.

12. Литература

12.1. Нормативные документы

[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>.

[COOKIES] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[HTTP-REPLAY] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/info/rfc8470>.

[QPACK] Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Field Compression for HTTP/3", RFC 9204, DOI 10.17487/RFC9204, June 2022, <https://www.rfc-editor.org/info/rfc9204>.

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

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7301] 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>.

[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>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

12.2. Дополнительная литература

[BREACH] Gluck, Y., Harris, N., and A. Prado, "BREACH: Reviving the CRIME Attack", July 2013, <http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf>.

[DNS-TERMS] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/info/rfc6585>.

[RFC8164] Nottingham, M. and M. Thomson, "Opportunistic Security for HTTP/2", RFC 8164, DOI 10.17487/RFC8164, May 2017, <https://www.rfc-editor.org/info/rfc8164>.

[TFO] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[TLS] 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>.

Приложение A. Вопросы перехода с HTTP/2

HTTP/3 в значительной степени опирается на HTTP/2 и протоколы имеют много сходств. В этом приложении описан подход к разработке HTTP/3, указаны важные отличия от HTTP/2 и описано, как сопоставить расширения HTTP/2 с HTTP/3.

HTTP/3 исходит из того, что сходство с HTTP/2 предпочтительно, но не является жёстким требованием. HTTP/3 отличается от HTTP/2 там, где QUIC отличается от TCP, нужно воспользоваться преимуществами QUIC (например, потоками) или нужно учесть важные недостатки (например, отсутствие полного упорядочивания). Хотя HTTP/3 похож на HTTP/2 в основных аспектах, таких как связь запросов и откликов с потоками, детали устройства HTTP/3 существенно отличаются от HTTP/2.

Некоторые важные отличия рассматриваются в следующих параграфах.

A.1. Потоки

HTTP/3 позволяет использовать большее число потоков (262-1), чем HTTP/2. Применяются такие же соображения в части исчерпания пространства идентификаторов потоков, хотя пространство значительно больше и, скорей всего, раньше будут достигнуты ограничения QUIC, такие как ограничения для окна управления потоком данных.

В отличие от HTTP/2 управление одновременными потоками HTTP/3 осуществляется в QUIC. Поток в QUIC считается закрытым, когда все данные были получены, а все передачи подтверждены партнёром. HTTP/2 считает поток закрытым, когда транспорту представлен кадр с флагом END_STREAM, в результате чего поток для эквивалентного обмена может сохранять состояние active достаточно долго. Серверы HTTP/3 могут разрешать большее число инициированных клиентом одновременных двухсторонних потоков по сравнению с HTTP/2, в зависимости от ожидаемой картины использования.

В HTTP/2 управление потоком данных применяется только для тела запросов и откликов (содержимое кадров DATA), а в HTTP/3 - для всех потоков QUIC (следовательно, для всех кадров HTTP/3).

Из-за наличия других типов односторонних потоков, HTTP/3 не полагается исключительно на число одновременных односторонних потоков при контроле числа одновременных выталкиваний, находящихся в пути. Вместо этого клиенты HTTP/3 используют кадр MAX_PUSH_ID для управления числом выталкиваний с сервера HTTP/3.

A.2. Типы кадров HTTP

Многие концепции кадрирования HTTP/2 могут быть исключены, поскольку в QUIC этим занимается транспорт. Кадры уже находятся в потоке и номер потока можно не указывать. Поскольку кадры не блокируют мультиплексирование (в QUIC это происходит на нижележащем уровне), поддержка пакетов с переменным максимальным размером не нужна. Прерывание потоков обрабатывается в QUIC и флаг END_STREAM не требуется, что позволило исключить поле Flags из базового формата кадров.

Содержимое (payload) кадров в значительной мере взято из [HTTP/2]. Протокол QUIC включает многие функции (например, управление потоком данный), присутствующие в HTTP/2 и в таких случаях отображение HTTP не реализует их заново. В результате некоторые типы кадров HTTP/2 не нужны в HTTP/3. Идентификаторы таких кадров были зарезервированы для обеспечения максимальной переносимости между реализациями HTTP/2 и HTTP/3. Однако все кадры, присутствующие в обоих отображениях, имеют идентичную семантику.

Многие различия связаны с тем, что в HTTP/2 применяется абсолютное упорядочивание кадров между всеми потоками, а QUIC обеспечивает такую гарантию лишь в рамках потока. В результате, если тип кадра предполагает, что кадр из другого потока будет получен в порядке передачи, HTTP/3 нарушает это.

Ниже приведены некоторые примеры адаптации, а также общие рекомендации для реализации кадров расширения, преобразующих расширения HTTP/2 в HTTP/3.

A.2.1. Различия в приоритизации

HTTP/2 задаёт приоритет в кадрах PRIORITY и (опционально) HEADERS, а в HTTP/3 нет сигналов приоритета. Однако отсутствие явного указания приоритета не означает, что приоритизация не важна для достижения высокой производительности.

A.2.2. Различия в сжатии полей

При разработке HPACK предполагалась упорядоченная доставка. Последовательность закодированных разделов полей должна приходить (и декодироваться) в конечной точке в том же порядке, в котором поля кодировались. Это гарантирует синхронизацию динамического состояния двух конечных точек.

Поскольку в QUIC не поддерживается полного упорядочивания, HTTP/3 использует модифицированную версию HPACK, называемую QPACK. В QPACK используется один односторонний поток для внесения всех изменений в динамическую таблицу, что гарантирует общий порядок обновлений. Все кадры с закодированными полями просто ссылаются на состояние таблицы в данный момент, не изменяя его. Дополнительные сведения приведены в [QPACK].

A.2.3. Различия в управлении потоком данных

HTTP/2 задаёт механизм управления потоком данных в потоках (stream). Хотя все кадры HTTP/2 доставляются в потоках, управление потоком данных применяется лишь для содержимого кадров DATA. QUIC обеспечивает управление потоком данных для потоков и все кадры HTTP/3, заданные в этом документе, передаются через потоки, поэтому подвержены управлению потоком данных.

A.2.4. Рекомендации для определения новых типов кадров

В кадрах HTTP/3 часто применяется кодирование QUIC для целых чисел с переменным размером. В частности, такое кодирование применяется для идентификаторов потока, что позволяет использовать больше значений, чем в HTTP/2. В некоторых кадрах HTTP/3 используются идентификаторы, отличные от идентификатора потока (например Push ID).

Переопределение кодирования типов кадров расширения может потребоваться, если оно включает идентификатор потока.

Поскольку поле Flags не применяется в базовом формате кадров HTTP/3, кадры, зависящие от наличия флагов, должны предусматривать место для них как часть своего содержимого (payload).

За исключением отмеченного выше, расширения HTTP/2 для типов кадров обычно переносятся в QUIC простой заменой потока 0 в HTTP/2 потоком управления HTTP/3. Расширения HTTP/3 не предполагают упорядочивания, но оно не повредит, и предполагается, что они будут переносимы в HTTP/2.

A.2.5. Сравнение кадров HTTP/2 и HTTP/3

DATA (0x00)
Заполнение не определено для HTTP/3 (параграф 7.2.1).

HEADERS (0x01)
Область PRIORITY и заполнение не определены в HTTP/3 (параграф 7.2.2).

PRIORITY (0x02)
Как указано в Приложении A.2.1, в HTTP/3 не поддерживается сигнализация приоритета.

RST_STREAM (0x03)
Кадров RST_STREAM нет в HTTP/3, поскольку потоками управляет QUIC. Тот же код применяется для кадров CANCEL_PUSH (параграф 7.2.3).

SETTINGS (0x04)
Кадры SETTINGS передаются лишь в начале соединения (параграф 7.2.4 и Приложение A.3).

PUSH_PROMISE (0x05)
Кадр PUSH_PROMISE не указывает поток, а вытолкнутый поток ссылается на кадр PUSH_PROMISE в Push ID (параграф 7.2.5).

PING (0x06)
Кадров PING нет в HTTP/3, поскольку QUIC обеспечивает эквивалентную функциональность.

GOAWAY (0x07)
GOAWAY не включает код ошибки. В направлении от клиента к серверу он передаёт Push ID вместо идентификатора созданного сервером потока (параграф 7.2.6).

WINDOW_UPDATE (0x08)
Кадров WINDOW_UPDATE нет в HTTP/3, поскольку управление потоком данных обеспечивает QUIC.

CONTINUATION (0x09)
Кадров CONTINUATION нет в HTTP/3 и вместо этого разрешены более крупные кадры HEADERS и PUSH_PROMISE.

Типы кадров, определённые расширениями HTTP/2, не требуется отдельно регистрировать для HTTP/3, если они применимы. Идентификаторы кадров из [HTTP/2] для простоты зарезервированы. Отметим, что пространство типов кадров в HTTP/3 значительно шире (62 бита вместо 8), поэтому многие типы кадров HTTP/3 не имеют эквивалентных кодов HTTP/2 (параграф 11.2.1).

A.3. Параметры HTTP/2 SETTINGS

Важным отличием от HTTP/2 является то, что настройки передаются один раз как первый кадр потока управления и в дальнейшем не могут меняться. Это исключает множество случаев, связанных с синхронизацией изменений.

Некоторые опции транспортного уровня, задаваемые в HTTP/2 через кадры SETTINGS, заменены транспортными параметрами QUIC в HTTP/3. Параметры уровня HTTP, сохраняющиеся в HTTP/3, имеют такое же значение, как в HTTP/2. Переопределенные параметры зарезервированы и их получение считается ошибкой (см. параграф 7.2.4.1).

Ниже приведено сопоставление параметров HTTP/2 SETTINGS.

SETTINGS_HEADER_TABLE_SIZE (0x01)
См. [QPACK].

SETTINGS_ENABLE_PUSH (0x02)
Удалено с заменой кадром MAX_PUSH_ID, обеспечивающим более детальное управление выталкиванием с сервера. Указание параметра с идентификатором 0x02 (SETTINGS_ENABLE_PUSH) в кадре HTTP/3 SETTINGS является ошибкой.

SETTINGS_MAX_CONCURRENT_STREAMS (0x03)
QUIC контролирует наибольший идентификатор потока в логике управления потоком данных. Указание параметра с идентификатором 0x03 (SETTINGS_MAX_CONCURRENT_STREAMS) в кадре HTTP/3 SETTINGS является ошибкой.

SETTINGS_INITIAL_WINDOW_SIZE (0x04)
QUIC требует задания размера окна управления потоком данных на уровне соединения и потока при начальном согласовании транспорта. Указание параметра с идентификатором 0x04 (SETTINGS_INITIAL_WINDOW_SIZE) в кадре HTTP/3 SETTINGS является ошибкой.

SETTINGS_MAX_FRAME_SIZE (0x05)
This setting has no equivalent in HTTP/3. Указание параметра с идентификатором 0x05 (SETTINGS_MAX_FRAME_SIZE) в кадре HTTP/3 SETTINGS является ошибкой.

SETTINGS_MAX_HEADER_LIST_SIZE (0x06)
Переименован в SETTINGS_MAX_FIELD_SECTION_SIZE.

В HTTP/3 значения настроек являются целыми числами переменного размера (6, 14, 30 или 62 бита) в отличие от 32-битовых полей в HTTP/2. Это часто обеспечивает более компактное кодирование, но может удлинять процесс для установок, использующих все 32-битовое пространство. Настройки, перенесённые из HTTP/2, могут переопределять своё значение с ограничением размера 30 битами для более эффективного кодирования или использовать 62 бита, если 30 не достаточно.

Настройки для HTTP/2 и HTTP/3 должны определяться раздельно. Идентификаторы настроек из [HTTP/2] для простоты зарезервированы. Пространство настроек HTTP/3 значительно шире (62 бита вместо 16 bits) и для многих настроек HTTP/3 нет эквивалентного кода HTTP/2 (см. параграф 11.2.2).

Потоки QUIC могут доставляться с нарушением порядка, поэтому конечным точка рекомендуется не ждать получения установок от партнёра перед ответом на другие потоки (см. параграф 7.2.4.2).

A.4. Коды ошибок HTTP/2

В QUIC используется такая же концепция ошибок потока и соединения, как в HTTP/2. Однако различия между HTTP/2 и HTTP/3 не позволяют напрямую переносить коды ошибок между версиями. Ниже приведено логическое сопоставление кодов ошибок HTTP/2 из раздела 7 в [HTTP/2] с кодами ошибок HTTP/3.

NO_ERROR (0x00)
: H3_NO_ERROR (параграф 8.1).

PROTOCOL_ERROR (0x01)
: H3_GENERAL_PROTOCOL_ERROR за исключением случаев наличия более конкретного кода (H3_FRAME_UNEXPECTED, H3_MESSAGE_ERROR, H3_CLOSED_CRITICAL_STREAM, см. параграф 8.1).

INTERNAL_ERROR (0x02)
: H3_INTERNAL_ERROR (параграф 8.1).

FLOW_CONTROL_ERROR (0x03)
: Не применим, поскольку потоком данных управляет QUIC.

SETTINGS_TIMEOUT (0x04)
: Не применим, поскольку подтверждение SETTINGS не определено.

STREAM_CLOSED (0x05)
: Не применим, поскольку потоками управляет QUIC.

FRAME_SIZE_ERROR (0x06)
: H3_FRAME_ERROR (параграф 8.1).

REFUSED_STREAM (0x07)
: H3_REQUEST_REJECTED (параграф 8.1) применяется для указания того, что запрос не был обработан. В иных случаях код не применим, поскольку потоками управляет QUIC.

CANCEL (0x08)
: H3_REQUEST_CANCELLED (параграф 8.1).

COMPRESSION_ERROR (0x09)
: Несколько кодов ошибок, заданных в [QPACK].

CONNECT_ERROR (0x0a)
: H3_CONNECT_ERROR (параграф 8.1).

ENHANCE_YOUR_CALM (0x0b)
: H3_EXCESSIVE_LOAD (параграф 8.1).

INADEQUATE_SECURITY (0x0c)
: Не применим, поскольку в QUIC предполагается достаточный уровень защиты для всех соединений.

HTTP_1_1_REQUIRED (0x0d)
: H3_VERSION_FALLBACK (параграф 8.1).

Коды ошибок для HTTP/2 и HTTP/3 должны определяться раздельно (см. параграф 11.2.3).

A.4.1. Сопоставление ошибок HTTP/2 и HTTP/3

Посредник, преобразующий между HTTP/2 и HTTP/3, может столкнуться с ошибками в любом восходящем потоке. Полезно сообщать об ошибках нисходящему потоку, но коды ошибок в основном отражают локальные проблемы соединения, о которых нет смысла сообщать.

Посредник, обнаруживший ошибку от восходящего источника, может указать её отправкой кода статуса HTTP, такого как 502 (Bad Gateway), который подходит для широкого класса ошибок.

Имеется несколько редких случаев, когда полезно сообщить об ошибке, отображением её на тип, наиболее соответствующий получателю. Например, посредник, получивший ошибку потока HTTP/2 типа REFUSED_STREAM от источника, имеет чёткий сигнал, что запрос не был обработан и его можно безопасно повторить. Передача этих сведений клиенту как ошибки потока HTTP/3 типа H3_REQUEST_REJECTED позволяет клиенту выполнить действия, которые он сочтёт подходящими. В обратном направлении посредник может счесть уместным передавать отмену клиентского запроса, которая указывается прерыванием потока с ошибкой H3_REQUEST_CANCELLED (параграф 4.1.1).

Преобразование ошибок описаны выше (логическое сопоставление). Коды определены в неперекрывающихся пространствах для защиты от случайного преобразования, способного привести к использованию неподходящих или неизвестных кодов. Посреднику разрешено преобразовывать ошибки потока в ошибки соединения, но ему следует учитывать стоимость организации соединения HTTP/3 в случаях, когда ошибка может быть временной или нерегулярной.

Благодарности

Robbie Shade и Mike Warres были авторами draft-shade-quic-http2-mapping, предшественника этого документа.

Рабочая группа IETF QUIC получила огромную поддержку от многих людей. Ниже перечислены некоторые люди, внёсшие существенный вклад в этот документ.

Bence Beky
Daan De Meyer
Martin Duke
Roy Fielding
Alan Frindell
Alessandro Ghedini
Nick Harper
Ryan Hamilton
Christian Huitema
Subodh Iyengar

Robin Marx
Patrick McManus
Luca Niccolini
奥 一穂 (Kazuho Oku)
Lucas Pardue
Roberto Peon
Julian Reschke
Eric Rescorla
Martin Seemann
Ben Schwartz
Ian Swett
Willy Taureau
Martin Thomson
Dmitri Tikhonov
Tatsuhiro Tsujikawa

Работа Mike Bishop частично поддерживалась компанией Microsoft, пока он был её сотрудником.

Предметный указатель

Адрес автора

Mike Bishop (editor)
Akamai
Email: mbishop@evequefou.be

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru