Санкт–Петербургский государственный университет

ЖАВОРОНКОВ *Владислав Олегович*

Выпускная квалификационная работа

Анализ протокола QUIC для использования в современных веб-приложениях

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2018 «Программирование и информационные технологии»

Научный руководитель:
кандидат физ.-мат. наук, доцент
кафедры компьютерного моделирования и
многопроцессорных систем:
Корхов Владимир Владиславович

Рецензент:
Старший программист 3 категории,
ООО «ФОРА СОФТ»,
Березнев Даниил Дмитриевич

Санкт-Петербург
2022 г.

Содержание

Перечень сокращений, условных обозначений, символов, единиц и терминов

TCP — Transmission Control Protocol

UDP — User Datagram Protocol

QUIC — Quick UDP Internet Connections

IP — Internet Protocol

HTTP — HyperText Transfer Protocol

ID — Identity document

TLS — Transport Layer Security

RTT — Round Trip Time

ОС — Операционная система

DNS — Domain Name System

LTE — Long-Term Evolution

WiFi — Wireless Fidelity

ВВЕДЕНИЕ

Многие годы весь интернет использовал TCP протокол для передачи данных, который начал устаревать еще в 2000-х годах. Данный протокол изначально не был создан для эффективной работы в современных сетях. Например, TCP соединение строго зависит от IP-адресов источника и получателя, что является хорошим решением, кода оба узла обмена статичны, как изначально планировалось, но с развитием коммуникационных технологий, появились беспроводные сети, и источник перестал быть привязан к одному IP. Таким образом, пользователь, подключенный к сети по LTE, может менять свой IP-адрес просто идя по улице, таким образом вынуждая TCP создавать новое соединение, что увеличивает задержку в получении данных. Еще одной проблемой TCP является то, что все данные, передаваемые по TCP, протокол передает как один файл или поток байтов, даже если на самом деле их несколько. Это привело к одной из самых известных проблем TCP – блокировке начала очереди.

В последние годы, большинство компаний, работающих в сфере веб-технологий пытались улучшить пользовательский опыт и безопасность коммуникации между пользователями и сервисами компании. Для этого, предпринимались попытки решить проблемы как на высоких, так и на более низких уровнях стека технологий в сети интернет. Так, некоторые компании создавали свой протокол поверх UDP, чтобы решить проблемы TCP, который сейчас использует преобладающее множество клиент-серверных приложений. Одной из таких попыток создать свой протокол был QUIC, разработанный компанией Google в 2012 году. Позже стандартизированный и легший в основу HTTP/3, протокол обещал решить большинство проблем стека TCP в связке с TLS.

На данный момент, QUIC активно развивается, сформированная в 2016 году рабочая группа рассмотрела более 23 версий спецификации и продолжает

развивать ее. Сервисы Google, такие как YouTube, Search, Gmail и многие другие полностью переведены на инфраструктуру HTTP/3.

Постановка задачи

Целью работы является анализ существующих решений со стороны клиента и сервера для использования протокола QUIC в современном веб-приложении. Для достижения поставленной цели необходимо: проанализировать протокол в текущей спецификации draft-ietf-quic-transport-27, создать клиент-серверное приложение в виде текстового чата с множеством пользователей, проанализировать сложности имплементации данного протокола, а также описать пути решения тех или иных задач, возникших при использовании данного протокола. На данный момент протокол имеет малое количество ресурсов и реализаций, а также достаточно низкоуровневые API, что может стать сложностью при имплементации нового решения в существующую архитектуру, так как устоявшиеся технологии имеют гораздо более высокий уровень абстракций как на стороне сервера, так и на стороне клиента.

Также при разработке веб-приложения поставлена задача реализовать упрощенную версию технологии WebSocket, но работающую поверх HTTP/3. Для этих целей предполагается использование клиентского WebTransport API для взаимодействия с сервером по HTTP/3 на низком уровне потоков. Необходимо написать клиентский модуль, инкапсулирующий сложности при управлении потоками, дающий возможности передавать данные между клиентом и сервером с помощью простых методов, близких к методам протокола WebSocket.

Обзор литературы

В книге [1] описываются протоколы стека TCP/IP на всех уровнях, их реализация, каково назначение данных протоколов. Принцип работы данных протоколов дает представление о работе сети интернет в целом, и том, какие проблемы лежат в основе данных протоколов при использовании в современных сетях и какие улучшения протоколов возможны.

В работе [5] описываются проблемы протокола TCP связанные с выбором скользящего окна и то, как протокол QUIC может решить эту проблему, а также приводятся улучшенная конфигурация протокола QUIC для уменьшения задержки в сетях 3G и LTE. На основе полученной схемы производится анализ работы протокола в сети.

В статье [6] рассматривается процесс установки соединения QUIC и методы защиты соединения. На основе механизма установки соединения QUIC реализуется собственная версия данного процесса с помощью Python. Данная статья позволяет детальнее рассмотреть процесс установки соединения, например, каким образом протокол шифрует пакеты, в чем заключается преимущество и недостаток данного подхода по отношению к TCP+TLS. Так, в статье указано что QUIC не может полностью реализовать потенциал от 0-RTT запроса, так как в таком случае возникает проблема с безопасностью, из-за того, что сервер не сможет определить подлинность IP-адреса и, таким образом, злоумышленник может подменить IP-адрес и атаковать сервер.

В статье [9] автор рассматривает причины, по которым преимущества QUIC не могут привести к кратному увеличению производительности. Автор утверждает, что QUIC все же использует разработанные ранее методы обеспечения безопасности и контроля полосы пропускание. А также, то, что QUIC смог решить проблемы блокировки начала очереди, свойственную протоколу TCP, но способы мультиплексирования, используемые в браузерах, не позволяют получить заметное преимущество.

Глава 1. Описание основных транспортных протоколов в сети интернет

Всемирная сеть с самого своего начала построена на базе стека протоколов TCP/IP. Семейство протоколов TCP/IP получило широкое распространение благодаря способности к взаимному объединению гетерогенных локальных и глобальных сетей, а также способности создавать основы для коммуникаций peer-to-peer и базовые службы поверх такого взаимодействия. В этой главе будут рассмотрены основные протоколы транспортного уровня. Их всего два TCP и UDP.

TCP и UDP используют разные типы взаимодействия между приложениями. Существует два основных типа связи. Первый тип — связи, ориентированные на создание соединения, — применяется при работе приложения с потоком данных. Второй вариант — связи без создания соединения — предполагает обмен независимыми сообщениями и подходит для случайных взаимодействий между приложениями при небольшом объеме пересылаемых данных [4].

1.1 Транспортный протокол TCP

Transmission Control Protocol (TCP) (протокол управления передачей) — один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI [1].

TCP — это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета [2]. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализации TCP

в большинстве случаев встроены в ядро ОС. Но также существуют реализации, работающие в пространстве пользователя.

Передача TCP сегментов осуществляется в виде Internet датаграмм. Заголовок датаграммы в Internet протоколе имеет несколько информационных полей, включая адреса отправляющего и принимающего хост-компьютеров. Заголовок TCP следует за Internet заголовком и дополняет его информацией, специфической для TCP протокола. Такое деление допускает использование на уровне хост-компьютеров протоколов, иных нежели TCP. Опишем формат TCP заголовка в таблице 1.

Бит	0-3	4-6	7-15	16-31
0	Порт источника			Порт назначения
32	Порядковый номер (SN)			
64	Номер подтверждения (ACK SN)			
96	Длина заголовка	Зарезервировано	Флаги	Размер окна
128	Контрольная сумма			Указатель важности
160	Опции			
160/192+	Данные			

Таблица 1 – формат TCP заголовка

Главной целью протокола TCP является обеспечение надежного, безопасного сервиса для логических цепей или соединений между парами процессов [1]. Чтобы обеспечить такой сервис, основываясь на менее надежных коммуникациях Internet, система должна иметь возможности для работы в следующих областях: базовая передача данных, достоверность, управление потоком, разделение каналов, работа с соединениями, приоритет и безопасность. В таблице 2 представлены описание и ключевые термины для каждой области.

Область	Описание	Ключевые термины
Базовая передача данных	Передача непрерывных потоков октетов (сегментов) в обоих направлениях. Функция проталкивания для подтверждения отправки.	Функция проталкивания

Достоверность	Протокол TCP должен иметь защиту от разрушения данных, потери, дублирования и нарушения очередности получения. Это достигается присвоением очередного номера каждому октету, а также требованием подтверждения от (ACK) от программы TCP. Если не получено подтверждение в течение контрольного интервала времени, пакет пересылается повторно. Для получателя номера пакетов нужны для восстановления очередности. Подтверждение фиксируется посредством добавления к каждому передаваемому сегменту контрольной суммы.	Подтверждение (ACK), Контрольный интервал времени, контрольная сумма
Управление потоком	Получатель управляет количеством данных, посылаемых ему отправителем с помощью «окна» вместе с подтверждением, которое указывает диапазон приемлемых номеров [13].	«Окно»
Разделение каналов	Для использования возможностей TCP во множестве процессов, протокол TCP предоставляет на каждом хост-компьютере набор адресов и портов. Вместе с адресами сетей и хост-компьютеров они образуют сокет.	Сокет Порт
Работа с соединениями	Программы протокола TCP должны поддерживать определенную информацию о состоянии каждого потока. Включает сокет, номер очереди, размеры окон.	Хранение состояния соединения
Приоритет и безопасность	Пользователи TCP могут потребовать для своего соединения приоритет и безопасность.	

Таблица 2 – Области работы TCP

Далее рассмотрим основные этапы передачи данных по TCP, от установки соединения до его закрытия.

1.1.1 Установка соединения TCP

Рассмотрим процесс установки соединения TCP. Его можно разделить на три стадии:

- Установка соединения
- Передача данных
- Завершение соединения

В процессе установления соединения возникает необходимость в информации о том, в каком состоянии сейчас находятся узлы, чтобы выполнять дальнейшие действия, для этого предусмотрен определенный набор состояний узла. Данный набор состоит из следующих состояний:

CLOSED — Состояние полного отсутствия соединения.

LISTEN — Ожидание запроса на соединение со стороны чужих портов и программ TCP.

SYN-SENT — клиент отправил запрос серверу на установление соединения и ожидает ответа.

SYN-RECEIVED — сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения.

ESTABLISHED — соединение установлено, идёт передача данных

FIN-WAIT-1 — одна из сторон завершает соединение, отправив сегмент с флагом FIN.

CLOSE-WAIT — другая сторона переходит в это состояние, отправив, в свою очередь сегмент ACK и продолжает одностороннюю передачу.

FIN-WAIT-2 — Ожидание запроса на закрытие соединения со стороны чужой программы TCP.

LAST-ACK — Ожидание запроса на закрытие соединения, ранее отправленного чужой программе TCP.

TIME-WAIT — Ожидание, когда истечет достаточное количество времени и можно быть уверенным, что чужая программа TCP получила подтверждение своего запроса на закрытие соединения.

CLOSING — обе стороны инициировали закрытие соединения одновременно: после отправки сегмента с флагом FIN узел-1 также получает сегмент FIN, отправляет ACK и находится в ожидании сегмента ACK (подтверждения на свой запрос о разъединении).

Теперь рассмотрим непосредственно процесс установки соединения, также называемый рукопожатием. На рис.1 представлено схематическое представление данного процесса.

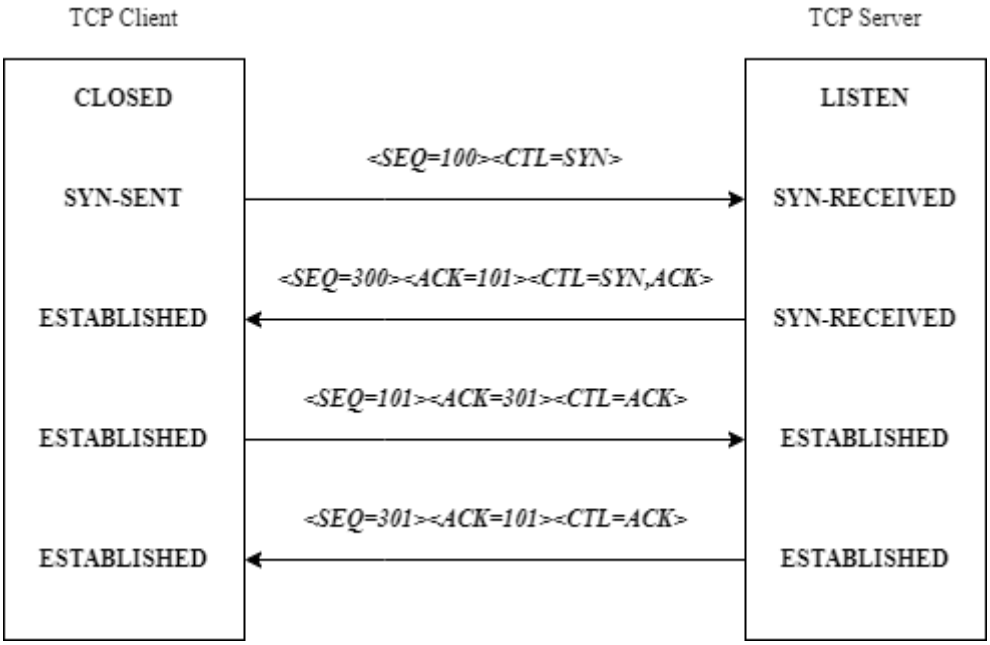

Рисунок 1 – Процесс установки TCP-соединения

На рис.1 представлены два узла TCP Client — узел, инициирующий процесс установки TCP-соединения и TCP Server — узел, ожидающий входящего соединения. В блоках под названиями узлов расположены состояния, в котором находятся узлы на определенном этапе установки соединения. Стрелками из одного блока в другой показана отправка сегмента. Над стрелкой показаны передаваемые данные.

Изначально клиент находится в состоянии CLOSED, а сервер в состоянии LISTEN. Клиент инициирует процесс соединения и отправляет серверу сегмент SYN с флагом SEQ равном 100, говорящем об использовании номеров последовательности, начиная с 100. Клиент переходит в состояние SYN-SENT. Далее, сервер передает SYN и подтверждение для принятого SYN в адрес клиента, и переходит в состояние SYN-RECEIVED. Затем, клиент отвечает пустым сегментом с подтверждением ACK для сегментов SYN, и переходит в состояние ESTABLISHED. Так как сервер в состоянии SYN-RECEIVED получил сегмент с флагом ACK, он переходит в состояние

ESTABLISHED и отправляет клиенту подтверждение [2]. TCP-соединение установлено.

1.1.2 Передача данных

Передача данных по протоколу TCP осуществляется с помощью обмена сегментами. В случае потери сегментов в результате ошибок, например, в случае ошибки в контрольной сумме или перегрузки сети, программа протокола TCP использует механизм повторной посылки по истечении определенного времени, чтобы убедиться в получении каждого сегмента.

При передаче, отправитель данных отслеживает следующий номер в очереди, подлежащий отправке с помощью переменной SND.NXT. В свою очередь, получатель данных отслеживает следующий номер, прибытие которого он ожидает с помощью RCV.NXT. Также отправитель отслеживает значение самого старого номера, который был отправлен, но еще не получил подтверждения в переменной SND.UNA.

Когда отправитель информации создает и посылает некий сегмент, он увеличивает значение переменной SND.NXT. Адресат по получении сегмента увеличивает значение переменной RCV.NXT и отправляет подтверждение. Когда программа TCP, пославшая данные, получает подтверждение, она увеличивает значение SND.UNA. Разность в значениях этих переменных является мерой, характеризующей задержку сегментов в сети. Величина, на которую надо всякий раз осуществлять приращение значения этих переменных, является длиной поля данных в сегменте. Заметим, что поскольку соединения находятся в состоянии ESTABLISHED, все сегменты, в дополнение к собственно данным, должны нести некую информацию о подтверждении ранее отправленных сегментов [2].

1.1.3 Закрытие соединения

При закрытии соединения создается сегмент с сигналом FIN и помещается в очередь сегментов, ждущих отправления. После этого программа TCP уже не будет принимать от этого клиента каких-либо команд на отправление данных по закрытому соединению, а сама переходит в состояние FIN-WAIT-1. Тем не менее, в этом состоянии еще возможно получение клиентом данных с этого соединения. Все сегменты, стоящие в очереди, и сам сегмент с сигналом FIN будут в случае необходимости посылаться напарнику вновь и вновь, пока не получат своего подтверждения.

Когда программа TCP партнера подтвердит получение сигнала FIN, и сама отправит сюда свой сигнал FIN, локальная программа может подтвердить получение последнего. Заметим, что программа TCP, получающая сигнал FIN, будет подтверждать его, но не будет посылать своего собственного сигнала FIN до тех пор, пока ее клиент тоже не закроет соединения.

1.1.4 Проблемы TCP

К основным проблемам протокола TCP относят:

- Частая повторная передача сегментов
- Размер окна — окно в TCP имеет ограниченный размер в 65535 байт, то есть если пропускная способность сети будет больше этого значения, мы не сможем использовать всю пропускную полосу, а только ограниченную размером окна в TCP.
- Обнаружение ошибки при передаче данных — алгоритм контрольной суммы считается слабым и может приводить к ошибкам.

1.2 Транспортный протокол UDP

Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Протокол UDP обеспечивает доставку датаграмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP. К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-датаграммы и контрольная сумма, позволяющие поддерживать целостность данных [3].

Протокол UDP ориентирован на транзакции, получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой.

Механизм данного протокола минимален, это можно увидеть, рассмотрев формат UDP заголовка (таблица 3) и сравнив его с TCP.

Бит	0-7	8-15	16-23	24-31
0	Порт отправителя		Порт получателя	
32	Длина		Контрольная сумма	
64	Данные			

Таблица 3 – формат UDP заголовка

В таблице 3 порт отправителя — порт процесса, посылающего датаграмму. Можно считать, что это порт, на который при отсутствии какой-либо информации следует посылать ответную датаграмму. Если данное поле не задействовано, то в него следует записать нули.

Порт получателя имеет смысл только в контексте конкретного Internet адреса получателя.

Длина — длина в октетах данной датаграммы, включая как заголовок, так и данные (Это означает, что минимальное значение поля длины равно восьми).

Контрольная сумма — 16 битное дополнение до единицы суммы дополнений UDP заголовка, данных и псевдозаголовка. Последний содержит информацию из заголовка в протоколе IP. В случае необходимости, датаграмма дополняется в конце нулевыми октетами, чтобы общее их количество стало четным [3].

1.2.1 Проблемы UDP

Основной проблемой UDP является его простота. У UDP ограничены функции применения из-за отсутствия соединения, ненадежности, отсутствия очередности пакетов. С одной стороны, это позволяет ему успешно использоваться в системах DNS, играх, видеостриминге. То есть в системах, где приоритетнее потерять пакет и получить новый, а не ждать получения потерянного. Но исключает использование в системах, в которых преобладает TCP.

Глава 2. Транспортный протокол QUIC

Предпосылок к созданию нового транспортного протокола было много. Большинство веб-платформ, таких как мессенджеры, социальные сети, стриминг платформы и другие, использовали TCP для передачи данных, так как единственный аналог UDP не гарантирует доставку. Но с развитием интернета, количество пользователей, использующих беспроводные сети, стало преобладать над пользователями в проводных сетях.

Беспроводные сети имеют ряд особенностей, таких как: динамическое изменение параметров сети, высокие показатели packet loss, jitter, reordering, фиксированные ассиметричный канал, смена IP-адреса [6]. Но в основе беспроводных сетей лежит все тот же стек TCP/IP, который изначально не был спроектирован под условия беспроводных сетей.

Очевидным вариантом решения данных проблем, является рассмотрение создания аналога TCP. Возможны несколько вариантов. Первый, в семейство TCP/IP на уровне транспортного протокола разработать свой протокол. У такого решения есть ряд проблем, основной из которых является обновление всего оборудования в сети для работы с этим протоколом, что может занять не один год. Другим решением является создание своего протокола поверх UDP. То есть мы можем взять основу UDP с его ненадежной доставкой, и на прикладном уровне написать протокол, способный работать как надежный.

До 2021 года, большие компании, имеющие достаточные ресурсы, разрабатывали свои протоколы поверх UDP для использования в своих системах. Но в мае 2021 года протокол QUIC over UDP был принят в качестве стандарта RFC 9000. Все больше компаний, для которых необходима надежная доставка данных за минимальное время начинают использовать протокол QUIC в своих продуктах.

Изначально, QUIC был разработан компанией Google в 2012 году. QUIC позволяет мультиплексировать несколько потоков данных между двумя

компьютерами, работая поверх протокола UDP, и содержит возможности шифрования, эквивалентные TLS и SSL. Имеет более низкую задержку соединения и передачи, чем TCP. Хорошо переносит потерю части пакетов путём выравнивания границ криптографических блоков по границам пакетов.

В этой главе рассматриваются основные абстракции, используемые данным протоколом, принцип работы и его преимущества перед существующими решениями.

2.1 Потоки

Потоки — это базовая абстракция, которую предоставляет протокол QUIC. Потоки в QUIC представляют собой легковесные, упорядоченные потоки байт. Потоки создаются при отправке данных. Например, один фрагмент потока может открыть, передать данные и закрыть поток. Потоки также могут быть долгоживущими и существовать на протяжении всего соединения [11].

Потоки могут быть созданы любым узлом, могут одновременно отправлять данные, чередуясь с другими потоками, и могут быть закрыты. QUIC допускает одновременную работу произвольного количества потоков и отправку произвольного количества данных в любом потоке с учетом ограничений управления потоком и ограничений потока.

Потоки могут быть однонаправленными или двунаправленными. Однонаправленные потоки передают данные в одном направлении: от инициатора потока до принимающей стороны. Двунаправленные потоки позволяют отправлять данные в обоих направления.

Потоки идентифицируются внутри соединения по числовому значению, называемому идентификатором потока. Идентификаторы потока уникальны для потока. Узел QUIC не должен повторно использовать идентификатор потока (Stream ID) в соединении. Идентификаторы потоков кодируются как целые числа переменной длины.

Внутри каждого типа соединения потоки создаются с численно увеличивающимися идентификаторами потоков. Идентификатор потока, который используется не по порядку, приводит к открытию всех потоков этого типа с идентификаторами потоков с меньшим номером.

Фрагменты потока инкапсулируют передаваемые данные. Узел использует Stream ID и смещение (offset) в фрагменте потока для упорядочивания данных [11].

Также потоки используют состояния, по типу состояний в TCP, на каждом узле. Всего есть две машины состояний, одна для отправляющей стороны, вторая для принимающей. В двунаправленном соединении, используются обе машины состояний на обоих узлах.

2.2 Соединения

Установка соединения в QUIC сочетает согласование версии с криптографическим и транспортным рукопожатиями для уменьшения задержки установления соединения. После установки, соединение может быть перенесено на другой IP-адрес или порт на любом узле. Соединение также может быть завершено на любом узле [11].

Каждое соединение имеет набор идентификаторов соединения (Connection IDs), каждый из которых определяет соединение. Причем идентификаторы соединения независимо выбираются узлами и используются в рамках узла. Основной функцией Connection ID является безопасность того, что изменения адреса на низлежащих протоколах, таких как UDP или IP не приведут к отправке данных на неверную конечную точку. Такой подход позволяет не зависеть от IP адресов клиента и сервера при установке соединения и дает возможность бесшовного переключения между сетями, например, при переходе от WiFi к LTE. Это является преимуществом QUIC в сравнении с TCP, которому при изменении адреса необходимо создавать новое соединение.

Одним из основных аспектов производительности протокола является количество круговых путей или циклов запрос-ответ, необходимых совершить для установки соединения (RTT). QUIC поддерживает 1-RTT и 0-RTT рукопожатия для уменьшения времени установки соединения, то есть при использовании протокола QUIC необходимо совершить 1, либо 0 циклов запрос-ответ для установки соединения. Чтобы уменьшить время, требуемое для установки соединения, клиент, который уже устанавливал 1-RTT соединение с конкретным сервером при повторном подключении может кэшировать некоторые параметры и впоследствии установить так называемое 0-RTT соединение с сервером [8]. Это позволит клиенту отправить данные почти мгновенно, без необходимости в ожидании окончания рукопожатия между узлами.

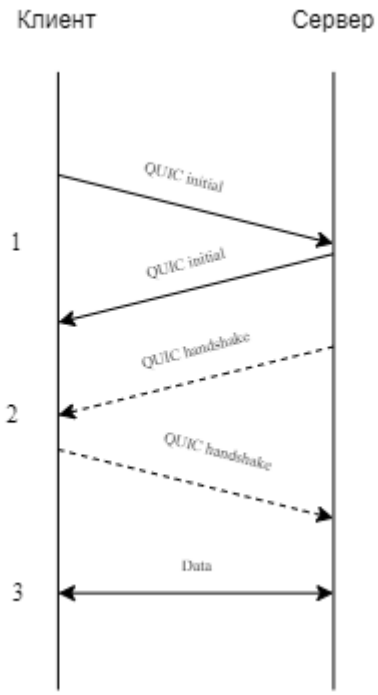

Рисунок 2 – установка 1-RTT QUIC соединения

На рисунке 2 представлен процесс установки первого соединения QUIC. Если посмотреть на цикл запрос-ответ под номером 1 QUIC initial, это является первым этапов установки соединения, когда клиент передает параметры соединения серверу и, если сервер утверждает параметры, отправляет ответ с initial пакетом. Далее сервер отправляет оставшиеся TLS сообщения, после

чего клиент подтверждает соединение и может быть начат процесс обмена данными [12].

Если клиент устанавливает первое соединение с сервером, рукопожатие 1-RTT используется для обмена информацией о соединении, включая идентификатор соединения и ключ шифрования/дешифрования. После успешного рукопожатия 1-RTT клиент отправляет и получает данные через рукопожатие 0-RTT; следовательно, задержка установления соединения уменьшается. Если рукопожатие 0-RTT не используется в течение определенного периода времени, информация о соединении удаляется, а соединение закрывается. Затем клиент инициализирует новое соединение, используя рукопожатие 1-RTT.

2.3 Преимущества и недостатки QUIC в сравнение с TCP + TLS

Важным отличием QUIC от TCP является его реализация в прикладном пространстве, в то время как TCP реализован в ядрах ОС. Данное отличие позволяет использовать и настраивать QUIC более гибко, подстраиваться под новые открытия в области протоколов передачи данных и улучшать его для нужд конкретного продукта. Это также является существенным недостатком, для каждого языка программирования необходима своя реализация данного протокола, что увеличивает количество возможных ошибок и расхождений с той или иной реализацией протокола и отладку этих ошибок.

В самом деле, QUIC реализует близкий аналог TCP поверх UDP. Например, метод управления полоской пропускания, реализованный в TCP и QUIC имеет одинаковую философию, оба протокола начинают с невысокой скорости и увеличивают ее по мере возможности, используя подтверждения, чтобы измерять пропускную способность сети [9].

Как упоминалось ранее, QUIC объединил транспортное и криптографическое рукопожатие, благодаря чему QUIC на один цикл круговых путей быстрее, чем его ближайший аналог TCP + TLS 1.3. При

версии TLS меньше 1.3 увеличивается количество рукопожатий на один и в таком случае, QUIC на два цикла быстрее. Данное ускорение, в действительности, заметно только в медленных сетях, на которых один цикл круговых путей занимает длительное время.

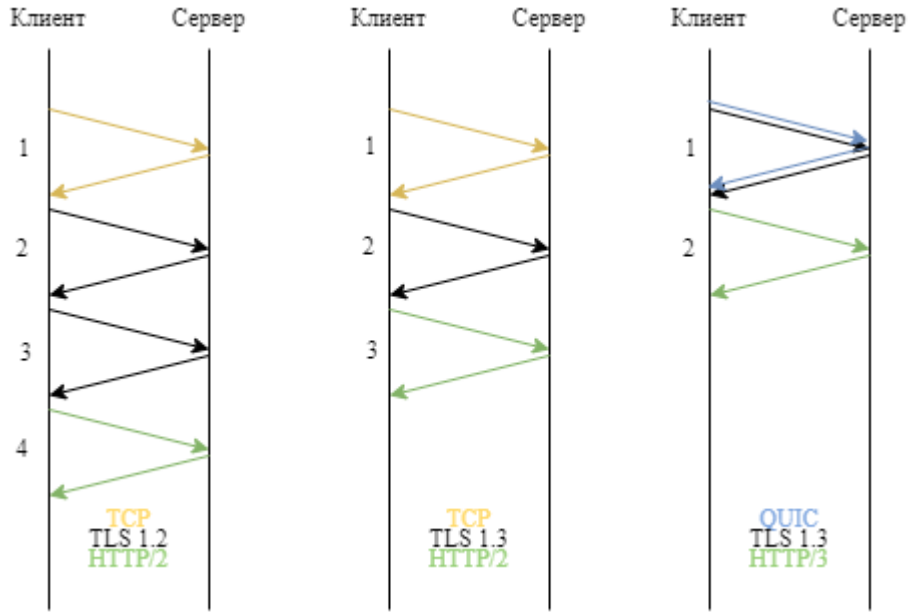

Рисунок 3 – установка соединения TCP+TLS и QUIC

На рисунке 3 представлен наглядный процесс установки соединения протоколов TCP с TLS и QUIC. Видно, что QUIC использует 1-RTT для установки соединения, в то время как TLS 1.2 имеет 3-RTT. При этом 1-RTT для QUIC актуально только для первого установления зашифрованного соединения, в последующих соединениях используется информация из первого соединения и, таким образом, достигается 0-RTT.

На текущий момент, производительность QUIC в быстрых сетях уступает производительности TCP+TLS. Связано это с тем, что TCP достаточно много лет использовался в широком наборе прикладных протоколов передачи данных в сети и оптимизировался на уровне ОС. QUIC же основан на UDP, который данных оптимизаций не получал. В последнее время UDP активно оптимизируется на уровне ОС, а также улучшается QUIC. На этом основании, и с учетом того, что в архитектуре QUIC действительно

заложены улучшения, призванные решить проблемы TCP и ускорить передачу данных, можно утверждать, что в будущем QUIC обгонит TCP по производительности.

Глава 3. Улучшения HTTP/3

На основе протокола QUIC была представлена третья версия протокола передачи гипертекста HTTP/3, известного также как HTTP-over-QUIC. Данный протокол пытается решить проблемы своего предшественника HTTP/2, с помощью объединения транспортного и криптографического рукопожатия, отсутствия блокировки начала очереди. Все преимущества HTTP/3 основаны на преимуществах QUIC, который использует в основе своей UDP.

Спецификация HTTP/3 регламентирует каким образом использовать QUIC для взаимодействия между клиентом и сервером. Так, для передачи данных, необходимо на клиенте создать двунаправленный поток, отправить данные, ожидать ответа от сервера в этом потоке и после ответа закрыть поток. Также протокол HTTP/3 использует внутренний сегментный слой похожий на слой HTTP/2 [7].

Также в каждом потоке основной единицей связи HTTP/3 является фрейм. Каждый тип фрейма служит своей цели. Например, кадры HEADERS и DATA составляют основу HTTP запросов и ответов. Кадры, относящиеся ко всему соединению, передаются в выделенном потоке управления. Мультиплексирование запросов выполняется с использованием абстракции потока в QUIC. Каждая пара запроса и ответа использует лишь один QUIC поток. Потоки независимы друг от друга, так что один поток, заблокированный потерей пакетов, не блокирует остальные потоки и таким образом, решается проблема блокирования начала очереди [10].

Также, как и в HTTP/2, в HTTP/3 представлена технология server push. Она позволяет серверу инициировать обмен запросом и ответом с клиентом, вместо ожидания запроса от клиента. Потенциально, данная технология уменьшает задержку при передаче данных, когда сервер, отвечая на запрос может включить дополнительные ресурсы, такие как заголовки запросов. Они находятся в кэше до тех пор, пока браузер не запросит ресурс,

соответствующий описанию. Повышение производительности происходит благодаря тому, что ресурсы отправляются еще до того, как браузер запросит их. В HTTP/3 данная функциональность реализована благодаря созданным сервером однонаправленным потокам [9].

Глава 4. Использование протокола QUIC в веб-приложении

Так как протокол QUIC реализован не на уровне ОС, а на прикладном уровне, для его использования необходимо использовать конкретную реализацию протокола под конкретный язык программирования.

Для перехода архитектуры на QUIC достаточно браузера, поддерживающего QUIC и сервера с поддержкой HTTP/3 соединения. Основные задачи во время перехода на HTTP/3 заключаются именно в адаптации архитектуры со стороны сервера. Со стороны браузера запросы будут использовать максимально доступный протокол вплоть до HTTP/3 без модификаций. Но существует инструмент, позволяющий использовать возможности QUIC на более низком уровне и управлять потоками вручную. WebTransport — это веб-API, использующий протокол HTTP/3 в качестве двунаправленной коммуникации [15]. Он предназначен для двусторонней связи между веб-клиентом и сервером HTTP/3. Этот интерфейс поддерживает отправку данных как ненадежно через API-интерфейсы датаграмм, так и надежно через API-интерфейсы потоков.

При использовании на стороне сервера нужно ориентироваться на язык программирования, на котором написан сервер и использовать библиотеку, написанную под конкретный язык. Например, для языка программирования GO написана библиотека quic-go. Список популярных языков и библиотек, реализующих передача данных по протоколу QUIC представлен в таблице 4.

Язык программирования	Библиотека QUIC
GO	quic-go
Node.JS	nodejs/quic
Java	kwik
Python	aioquic
C/C++	msquic

Таблица 4 – языки и библиотеки с реализацией QUIC для них

4.1 Имплементация QUIC в веб-приложение

В этой главе будет описано клиент-серверное веб-приложение, созданное в рамках этой работы и доступное по ссылке [17], которое для передачи данных использует протокол QUIC и инструмент WebTransport. Стоит упомянуть, что WebTransport поддерживается ограниченным количеством браузеров и на данный момент скорее не подходит для промышленной разработки. Для написания клиентской части будем использовать библиотеку React.js и TypeScript. Для серверной части был выбран язык Python и библиотека Aioquic.

Так как протокол QUIC использует только защищенное подключение, необходимо получить сертификат и ключ для локальной разработки. Получить сертификат можно двумя способами: купить его на специализированных сайтах для конкретного хоста, либо локально создать сертификат с ключем и запустить браузер со специальными параметрами. Для процесса разработки обычно достаточно локального сертификата. Для того, чтобы его создать, средствами программы openSSL и утилиты base64 создается сертификат и кодируется в формате base64. Для корректной работы локального сертификата, используется браузер Chromium.

При написании сервера на aioquic стоит учитывать, что это не полноценная библиотека для построения веб-сервера, это низкоуровневые API, которые дают доступ к основным абстракциями QUIC и HTTP/3. То есть при работе с данной библиотекой, необходимо оперировать базовыми абстракциями QUIC такими как потоки и соединения. Для инициализации сервера необходимо создать класс, унаследованный от QuicConnectionProtocol, с методами quic_event_received и _h3_event_received. Далее, в _h3_event_received проверить заголовки на поле протокол, так как наш сервер может не поддерживать QUIC и в таком случае необходимо реализовать изменение протокола на доступный. Если в заголовке метод задан как CONNECT, то необходимо проверить адрес подключения и если на

сервере используется обработчик для данного адреса, то ответить клиенту с номером потока, который используется как поток соединения и успешным статусом, по стандарту 200.

На стороне клиента WebTransport API используется для взаимодействия с сервером. Основной функционал предоставляется методами createBidirectionalStream и createUnidirectionalStream [16]. При передаче данных по протоколу HTTP/3 клиенту необходимо открыть двунаправленный поток, записать в него данные, получить ответ от сервера и закрыть данный поток. Каждая повторная отправка данных с клиента будет сопровождаться созданием двунаправленного потока методом createBidirectionalStream, и закрытием его после получения данных. Стоит обратить внимание, что данный алгоритм действий выполняется в рамках одного соединения. Клиенту нужно открыть строго одно соединение с сервером и передавать данные с помощью потоков, используя только его.

WebTransport API может использоваться вместо WebSockets, только первый поддерживает множественные потоки, однонаправленные потоки, неупорядоченную доставку. Достаточно вспомнить, что WebSockets являются надстройкой над TCP, в которой устанавливается одно TCP соединение без тайм-аута и поддерживается на протяжении всей сессии обмена сообщениями. По тому же принципу работает двунаправленный поток в HTTP/3 [14].

Далее будет рассмотрена реализация клиент-серверного взаимодействия с использованием двунаправленных и однонаправленных потоков на примере приложения чата в реальном времени. Приложение представляет из себя одну страницу, с полем для ввода и кнопкой отправки сообщения, а также окном для вывода сообщений, причем свои сообщения будут отличаться от чужих стилистикой.

Рисунок 4 – Основной экран приложения

На рисунке 4 представлен интерфейс программы чата. В списке сообщений, сообщения 8-11 отправлены текущим пользователей, а сообщения 1-7 другими пользователями. При открытии страницы чата, пользователю присваивается уникальный идентификатор. При обновлении страницы идентификатор генерируется заново, и пользователь считается новым. При вводе сообщения в поле ввода и нажатии кнопки «SEND», сообщение появляется в списке и отправляется другим пользователям и появляется в списках у них.

Для реализации данного функционала необходимо учитывать особенности текущей реализации возможностей QUIC и HTTP/3 как в браузере, так и на серверной стороне. Алгоритм отправки сообщения всем пользователям чата можно разделить на несколько этапов:

- Установка соединения
- Создание потока (однонаправленного или двунаправленного) на клиенте
- Запись данных в поток
- Получение и сохранение данных на сервере

- Рассылка сообщения пользователям либо по существующим двунаправленным потокам, либо по созданным со стороны сервера однонаправленным потокам
- Либо чтение данных из ранее созданного двунаправленного потока, либо ожидание входящих однонаправленных потоков со стороны сервера и чтение данных из них
- Вывод сообщения в список

Оба варианта, как с использованием однонаправленных потоков, так и двунаправленных будут работать. Но, оптимально, в данном случае, использовать один двунаправленный поток, потому что такой подход позволяет экономить ресурсы на создание множества однонаправленных потоков как на клиенте, так и на сервере. А также такой подход позволяет избежать проблем с управлением или контролем множества потоков. Структура с одним двунаправленным потоком интуитивно понятнее, так как есть одно соединение и один поток, по которому передаются данные. Есть у данного подхода и недостаток, при создании однонаправленного потока под каждый процесс передачи данных, мы точно знали границы передаваемых данных. То есть каждый поток передавал некоторую законченную сущность данных, в конкретном случае поток байт, декодируемый в объект с данными. При двунаправленном потоке, если создавать лишь один поток на все соединение, данные могут прийти из нескольких запросов, например, в случае одновременной отправки множества сообщений от другого пользователя. Таким образом, для данного подхода необходим собственный механизм разграничения данных.

На сервере для отправки данных всем пользователям по двунаправленному потоку необходимо сохранить данные о пользователях. К этим данным относятся соединение и уникальный идентификатор потока, соединение необходимо для идентификации подключенного пользователя, так как один пользователь в чате создает одно соединение, а так как используется

одно двунаправленное соединение нам необходим номер потока, по которому пользователь подключался, чтобы отправить сообщения от других пользователей.

Если использовать однонаправленные соединения, то номер потока для пользователя сохранять не нужно. В таком случае, необходимо только соединение. При получении сообщения от какого-либо пользователя, в цикле создаются однонаправленные потоки в каждом сохраненном соединении и записываются в них данные, после чего потоки закрываются.

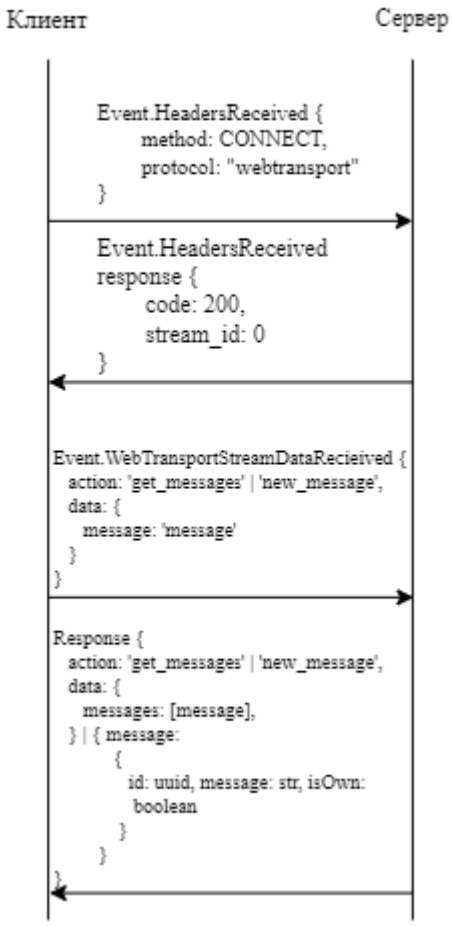

Рисунок 5 – клиент-серверное взаимодействие по HTTP/3 при передаче сообщений

На рисунке 5 представлено взаимодействие клиента с сервером при установлении соединения, а также отправке и получении сообщений. Для начала HTTP/3 клиент отправляет заголовки, с методом CONNECT и протоколом, в данном случае webtransport. В этом поле также могут быть websocket или http, с помощью данного поля на сервере можно отслеживать,

какой протокол использует клиент и использовать на одном сервере разные обработчики протоколов. Далее сервер, получает событие HeadersReceived и при возможности создать соединения для переданного протокола, создает соединение и отправляет ответ с кодом 200.

После установки соединения пользователь может отправить данные. В приложении предусмотрена отправка только по двунаправленному потоку, поэтому на сервере будет получено событие WebTransportStreamDataReceived. Данные передаются в виде последовательности байт. Стандартный запрос представляет из себя объект с двумя полями action и data. Поле action может быть только двух видов get_messages и new_message. Первый тип используется для получения всех сообщений в чате при заходе пользователя на страницу, второй тип используется для уведомления пользователей о том, что один из пользователей написал сообщение. В поле data в зависимости от типа события будет либо строковое сообщение, либо поле данных будет отсутствовать.

Ответ сервера также представляет из себя последовательность байт, который на клиенте декодируется в строку, а после с помощью утилиты JSON преобразуется в объект. В ответе, как и в запросе, два поля, один с типом, который полностью соответствует типам запроса, а второе с данными, но в поле данных может быть либо объект сообщения, если это событие получения нового сообщения, либо список сообщений.

В итоге, пользователь может отправлять и получать сообщения с помощью QUIC и WebTransport в реальном времени. При написании приложения, были реализованы на стороне сервера методы отправки всем пользователям, отправка одному пользователю, кодирование и декодирование потоков, работа с JSON. На стороне клиента была написана высокоуровневая класс-обертка над WebTransport, инкапсулирующая сложности при работе с потоками и соединениями, предоставляющая метод sendData для отправки данных и метод onReceiveData для получения любых сообщений, пришедших

в рамках ранее установленного соединения. На рисунке 6 представлены классы, реализованные на стороне клиента и сервер.

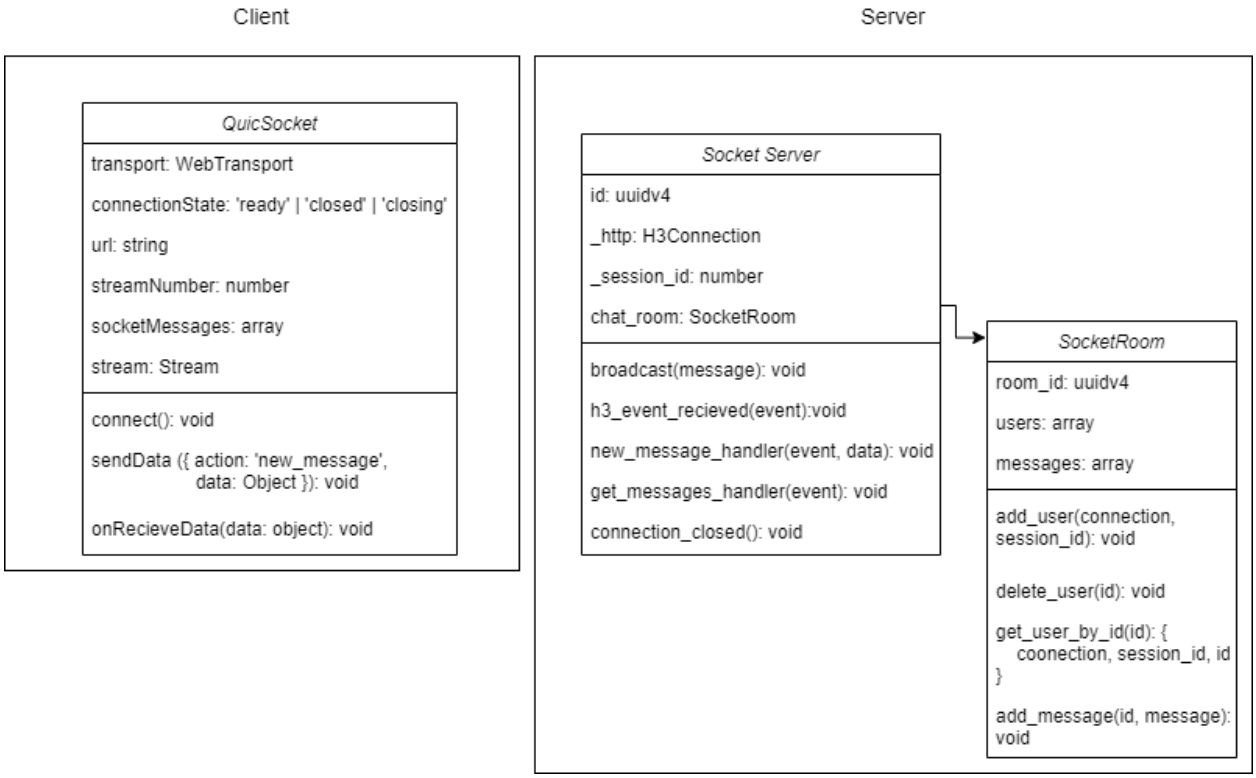

Рисунок 6 – интерфейсы клиента и сервера для передачи данных

На рисунке 6 на стороне клиента представлен основной класс QuicSocket для использования QUIC в веб-приложении. В поле url указывается адрес сервера, с которым необходимо установить соединение. При открытии клиентом сайта вызывается метод connect, который инициирует создание соединение на стороне клиента и ожидает ответа сервера о готовности соединения. Далее, при отправке сообщения вызывается метод sendData с типом события new_message, после чего происходят необходимые преобразования объекта в байтовое представление, открывается двунаправленный поток, если еще не открыт и данные записываются в этот поток. При получении клиентом нового сообщения вызывается метод onReceiveData, с полученными в потоке данными и происходит отображение сообщения на клиенте.

На рисунке 6 со стороны сервера представлены интерфейсы непосредственно обработчика всех событий на конечной точке /socket и

интерфейс комнаты сокета – абстракции, для учета всех пользователей и сообщений в чате, а также, при необходимости, создании нескольких чатов в рамках одного приложения.

В таблице 5 сравниваются возможности описанного выше решения с WebSocket over TCP.

Возможность	WebSocket TCP	Реализация с WebTransport
Отправка сообщений (клиент)	+	+
Слушатель событий (клиент)	+	+
Транслирование всем (сервер)	+	+
Поддержка JSON	+	+
Аутентификация	+	–

Таблица 5 – достоинства и недостатки собственного решения с существующим

Из таблицы 5 можно сделать вывод, что основные интерфейсы для использования реализованы в собственном решении с использованием WebTransport, при этом решение не идеально и имеет перспективы для доработки. Например, система аутентификации, необходимая всем современным приложениям.

4.2 Сравнительный анализ производительности QUIC и TCP

Для проведения сравнительного анализа QUIC с TCP в ранее разработанное приложение добавлена поддержка TCP запросов и протокола WebSocket. Таким образом, для каждой из рассматриваемых далее сетевых конфигураций использовались оба протокола, как TCP, так и QUIC.

Для анализа используется ноутбук Acer Aspire A515-55G Inter Core i5-1035G1 1.00GHz 1.19GHz Ram 8.00 GB на котором запущен клиент в браузере Chromium и сервер. Для эмуляции реальных сетей используется инструмент Netem. С помощью него добавляется задержка, потери пакетов, изменение порядка пакетов и отправка одинаковых пакетов. Для анализа сетевого

трафика используется утилита tcpdump. С помощью нее захватывается TCP и UDP трафик и сохраняется в файл для дальнейшего анализа.

Данный анализ включает в себя исследование поведения QUIC и TCP на сетях с разной пропускной способностью и потерей пакетов. В таблице 6 представлена информация о значения взятых для задержки, потерь и пропускной способности.

Задержка (мс)	25/50	25/50	25/50	25/50
Потери (%)	0.0	0.5	1	2
Пропускная способность (Мбит/с)	4	8	12	16

Таблица 6 – параметры сети, используемые в анализе

В таблице 7 представлены результаты анализа протоколов QUIC и TCP. Значения пропускной способности в данной таблице получены с помощью непрерывной отправки сообщений по разным протоколам в течение 20 секунд. Для каждого набора параметров тест запускался 3 раза и бралось среднее значение.

Пропускная способность сети, Мбит/с	Задержка, мс	Потеря пакетов, %	Пропускная способность QUIC, бит/с	Стандартное отклонение, бит/с	Стандартная ошибка среднего	Пропускная способность TCP, бит/с	Стандартное отклонение, бит/с	Стандартная ошибка среднего
4	25	0	3,879,025	47.68	15.08	3,825,464	12.85	4,86
		0.5	3,878,500	52.13	15.05	3,773,696	2,443.81	736.84
		1.0	3,878,524	65.76	18.98	3,517,576	7,366.91	2,221.21
		2.0	3,866,544	1,299.27	391.74	2,697,104	9,788.43	3,095.37
8	25	0	7,758,360	55.90	18.63	7,650,944	24.33	8.60
		0.5	7,754,824	755.27	209.47	5,542,352	34,650.91	8,946.83
		1.0	7,670,056	14,735.02	4,086.76	3,961,408	25,783.21	8,624.62
		2.0	6,596,272	107,426.67	31,011.41	2,817,920	11,004.32	3,479.87
4	50	0	3,859,072	7,098.35	2,366.12	3,825,408	12.90	4.56
		0.5	3,878,640	61.79	17.14	2,850,752	8,668.56	3,876.59
		1.0	3,871,112	1,105.36	333.28	2,079,584	14,139.02	4,713.01
		2.0	3,573,936	39,897.73	12,029.62	1,725,344	99,259.85	35,093.66
8	50	0	7,758,256	56.60	21.39	7,651,080	9.07	4.05
		0.5	7,710,728	19,503.30	5,640.12	2,282,568	16,029.88	6,544.17
		1.0	7,054,712	107,708.44	28,786.29	2,048,808	5,104.05	1,929.15
		2.0	5,211,256	209,321.01	60,245.78	1,496,152	9,393.27	3,131.09

12	25	0	11,637,664	54.00	19.09	11,476,680	29.06	11.86
		0.5	10,982,328	277,611.83	71,679.07	5,517,464	24,667.95	8,721.42
		1.0	11,169,384	111,782.29	29,875.07	3,979,040	19,857.92	8,880.73
		2.0	10,414,520	286,574.33	95,524.71	2,753,000	10,853.31	4,853.75
16	25	0	15,516,808	26.65	8.43	15,302,168	15.28	6.84
		0.5	15,360,648	59,970.22	17,311.21	5,815,536	51,609.90	25,804.96
		1.0	14,547,232	236,119.23	71,192.52	3,949,176	21,747.24	8,219.88
		2.0	12,730,592	486,936.33	140,566.21	2,739,704	7,183.25	2,932.55
12	50	0	11,637,496	69.78	26.37	11,476,496	11.71	5.24
		0.5	11,637,040	46.84	15.61	2,936,712	22,170.68	9,051.14
		1.0	10,208,680	272,519.02	86,178.08	2,114,544	6,038.02	2,134.76
		2.0	9,380,888	422,657.23	127,435.64	1,465,448	6,494.77	2,454.79
16	50	0	15,516,976	64.62	22.84	15,302,344	24.18	12.09
		0.5	15,090,864	150,824.43	41,831.57	3,047,072	21,015.21	7,430.28
		1.0	13,200,880	527,822.72	175,940.11	2,153,712	9,698.57	3,428.99
		2.0	13,448,920	456,912.71	144,488.09	1,488,560	4,493.22	1,420.98

Таблица 7 – Средняя пропускная способность, стандартное отклонение и стандартная ошибка среднего для протоколов QUIC и TCP с задержкой и потерей пакетов

Как видно из таблицы 7, для любой пропускной способности и задержки, при отсутствии потерь пакетов QUIC и TCP используют пропускную способность близкую к максимальное и, при этом, в некоторых экспериментах TCP имеет даже большую, чем QUIC пропускную способность. Но при увеличении процента потерянных пакетов TCP сильно теряет пропускную способность, и она менее стабильна. В то время как QUIC, при увеличении процента потери пакетов, сохраняет пропускную способность на близком к максимальному уровню. Это связано с архитектурой протокола QUIC, который не имеет проблемы с блокировкой начала очереди и адаптируется к потерям пакетов, сохраняя при этом пропускную способность. По данным из таблицы построим графики зависимости пропускной способности протокола от количества потерянных пакетов.

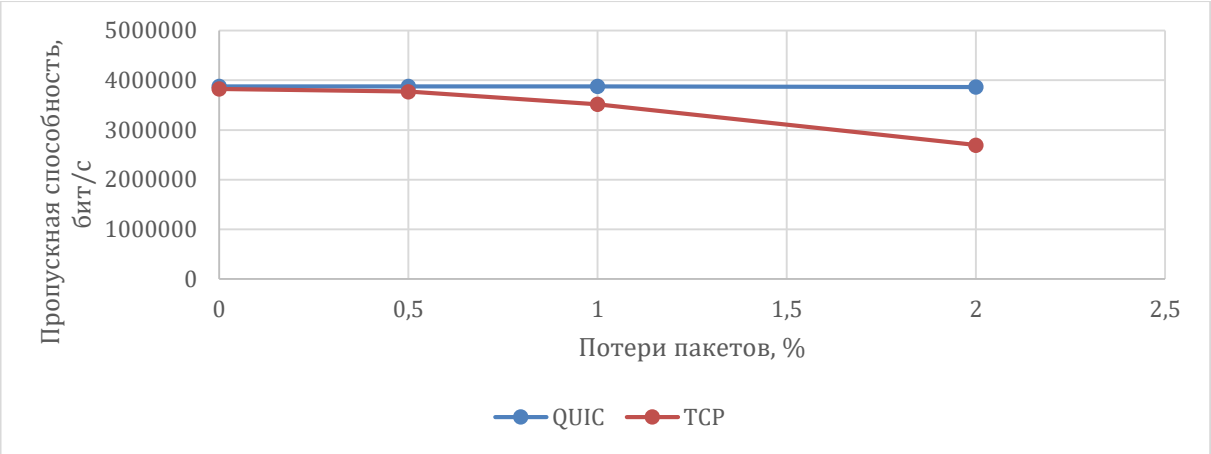

Рисунок 7 – пропускная способность QUIC и TCP при пропускной способности сети 4 Мбит/с и задержке 25 мс

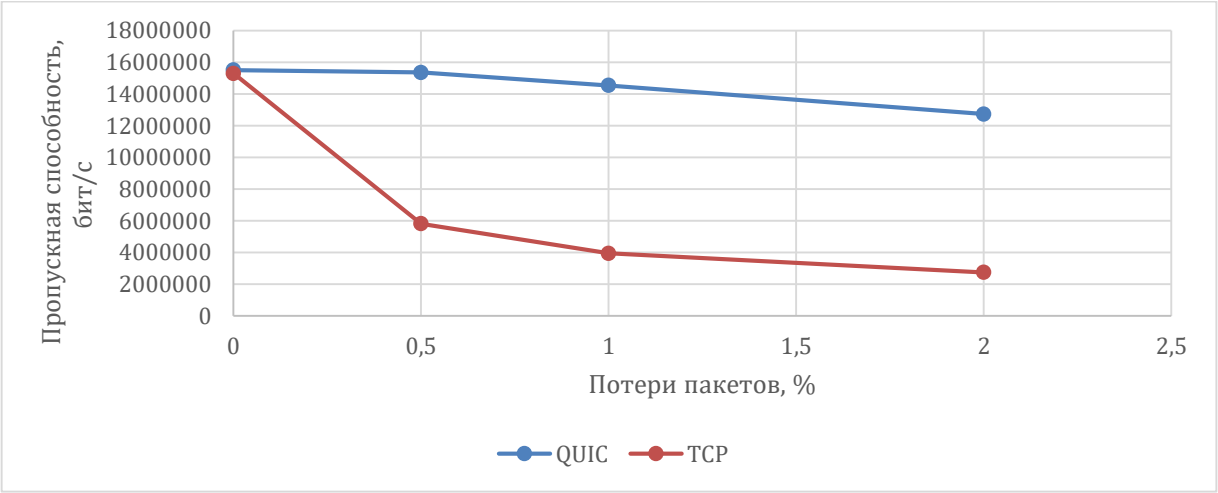

Рисунок 8 – пропускная способность QUIC и TCP при пропускной способности сети 16 Мбит/с и задержке 25 мс

Из рисунка 7 и рисунка 8 можно сделать вывод, что чем больше пропускная способность сети, тем сильнее TCP уменьшает пропускную способность в связи с потерей части пакетов, в то время как QUIC тоже уменьшает пропускную способность, но гораздо меньше чем это делает TCP. Так, при увеличении потерь пакетов с 0% до 0.5% в сети с 16 Мбит/с, пропускная способность TCP уменьшается с 15302168 бит/с до 5815536 бит/с, то есть падение пропускной способности на 62%. QUIC при тех же условиях уменьшает пропускную способность с 15516808 бит/с до 15360648 бит/с, что на 1% меньше изначального значения.

Также можно рассмотреть изменение стандартного отклонения при увеличении потерь пакетов.

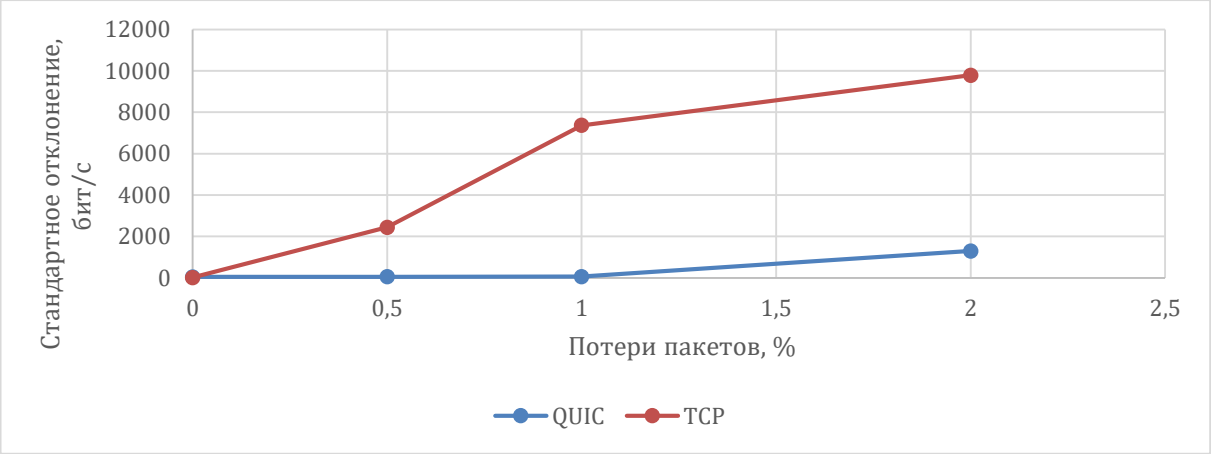

Рисунок 9 – значения стандартного отклонения QUIC и TCP при пропускной способности сети 4 Мбит/с и задержке 25 мс

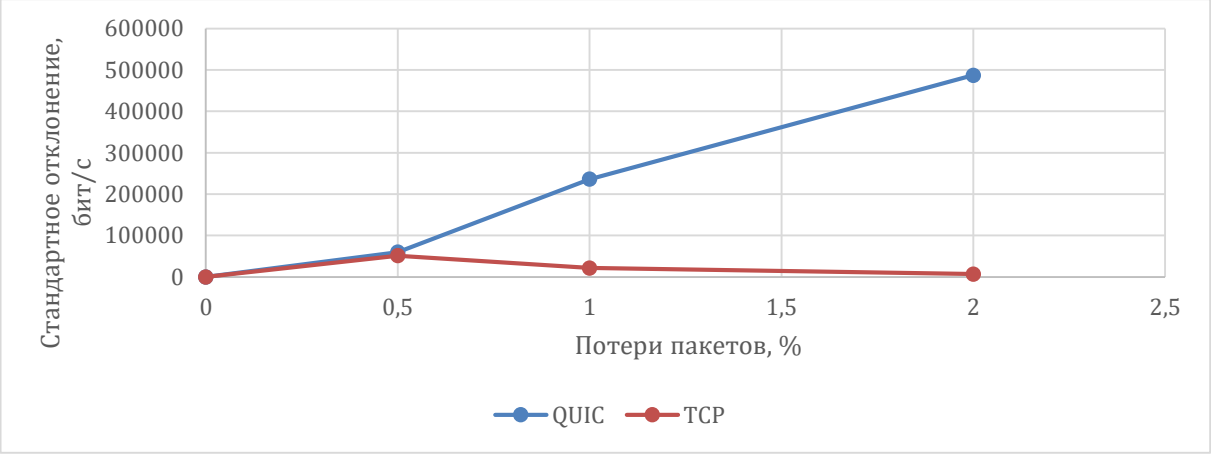

Рисунок 10 – значения стандартного отклонения QUIC и TCP при пропускной способности сети 16 Мбит/с и задержке 25 мс

На рисунке 9 видно, что при низкой пропускной способности сети, при увеличении потерь пакетов, пропускная способность TCP изменяется в большем интервале, чем при аналогичных потерях пакетов в QUIC, что ухудшает устойчивость соединения. При этом, при рассмотрении рисунка 10, где пропускная способность сети заметно выше, стандартное отклонение пропускной способности QUIC заметно больше, чем у TCP. То есть, хотя QUIC и сохраняет высокую пропускную способность при высоком проценте потерь пакетов, но при этом жертвует стабильностью соединения.

Таким образом, сравнительный анализ двух протоколов показал, что QUIC лучше использует пропускную способность сети при потерях пакетов в данной сети, чем TCP. При этом, стандартное отклонение, при достаточной пропускной способности и увеличивающихся потерях пакетов, в протоколе QUIC высокое, в сравнении с TCP, при тех же условиях. В QUIC высокое стандартное отклонение может ухудшать стабильность соединения. Также анализ показал, что QUIC и TCP достигают почти одинаковых пропускных способностей при различных значениях задержки пакетов. В итоге, потенциально QUIC позволяет отправлять в 2 раза больше данных в единицу времени в определенных условиях, чем TCP в аналогичных условиях.

Выводы

В рамках данной работы было проведено исследование протокола QUIC и использование его в современном веб-приложении. На данный момент, протокол QUIC не дает больших преимуществ при миграции приложений с HTTP/2, так как протокол UDP, являющийся основой QUIC недостаточно оптимизирован под текущие потребности современных веб-приложений. Также, при внедрении протокола в свои продукты компаниям придется столкнутся с отсутствием высокоуровневых библиотек способных поддерживать QUIC, поэтому компаниям придется самим писать данные библиотеки и возможности, что потребует значительных затрат в рабочих часах. Текущие низкоуровневые реализации QUIC также имеют достаточное количество проблем и требуют доработок, что может замедлить переход компаний на новую технологию.

На данный момент, переход на QUIC имеет смысл только в компаниях, работающих с большим количеством данных и клиентов из разных точек планеты, а также имеющих возможность исследовать и разрабатывать технологии не дающих бизнес-результата, но дающих потенциальный прирост производительности. Так как, на самом деле, QUIC экономит только 1-RTT по сравнению с реализацией HTTP/2, что заметно только на сетях с очень медленным интернетом, на которых время одной итерации запрос-ответ будет больше 200 мс.

При этом, QUIC имеет потенциал стать успешной заменой стека TCP+TLS. Благодаря его реализации в пользовательском пространстве и показателях сравнимых с TCP+TLS, он может быстрее развиваться, обрастать расширениями и улучшаться для каждой задачи индивидуально, таким образом, обгоняя стек TCP в развитии.

Заключение

В данной работе проведен анализ существующих решений со стороны клиента и сервера для использования протокола QUIC в современном веб-приложении. Проанализирован протокол в текущей спецификации draft-ietf-quic-transport-34, создано клиент-серверное приложение в виде текстового чата с множеством пользователей, проанализирована сложность имплементации данного протокола, а также описаны пути решения тех или иных задач, возникших при использовании данного протокола.

Клиентская часть приложения написана на React.js и Typescript. На стороне клиента был реализован модуль для высокоуровневого взаимодействия с сервером с помощью двунаправленных потоков и модуль, инкапсулирующий однонаправленные потоки. Благодаря чему, взаимодействие с сервером было похоже на связь с помощью вебсокетов. Оба модуля используют WebTransport API. Серверная часть приложения написана на Python с использованием библиотеки Aioquic. Поддерживает протокол QUIC и WebTransport, имеет возможность расширения при добавлении новых конечных точек, сохраняет соединения и может отправлять сообщения всем пользователям по двунаправленным потокам.

Список литературы

1 Лора А. Чеппел, Эд Титтел. TCP/IP. Учебный курс. 1-е изд. СПб: БХВ-Петербург, 2003, 976 с.

2 Спецификация TCP. – URL: http://citforum.ru/nets/tcp/tcpspec.shtml (дата обращения: 15.04.2022).

3 Спецификация UDP. – URL: http://citforum.ru/nets/tcp/udpspec.shtml (дата обращения: 15.04.2022).

4 Фейт Сидни. TCP/IP Архитектура, протоколы, реализация. 2-е изд. Лори, 2014, 448 с.

5 Jung, J., An, D. Access Latency Reduction in the QUIC Protocol Based on Communication History. Electronics 2019, 8, 1204. DOI: 10.3390/electronics8101204

6 Gagliardi, E., Levillain, O. Analysis of QUIC Session Establishment and Its Implementations. In: Laurent, M., Giannetsos, T. (eds) Information Security Theory and Practice. WISTP 2019, 16 с. DOI: 10.1007/978-3-030-41702-4_11

7 Hypertext Transfer Protocol Version 3 (HTTP/3). – URL: https://quicwg.org/base-drafts/draft-ietf-quic-http.html (дата обращения: 18.05.2022)

8 HTTP/3: Performance Improvements. – URL: https://www.smashingmagazine.com/2021/08/http3-performance-improvements-part2/

9 HTTP/3: Practical Deployment Options. – URL: https://www.smashingmagazine.com/2021/09/http3-practical-deployment-options-part3/

10 Multipath QUIC: Design and Evaluation. – URL: https://conferences2.sigcomm.org/co-next/2017/presentation/S4_2.pdf

11 QUIC IETF спецификация. – URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34 (дата обращения: 22.04.2022)

12 Hasselquist, D., Lindström, C., QUIC Behavior over Dual Connectivity. IEEE MASCOTS 2021, 19 с. DOI: 10.48550/arXiv.2112.14328
13 Robert Morris. Scalable TCP Congestion Control. IEEE INFOCOM 2000, 8 с. – URL: http://nil.lcs.mit.edu/rtm/papers/thesis.pdf
14 The WebSocket Protocol. – URL: https://datatracker.ietf.org/doc/html/rfc6455
15 Using WebTransport. – URL: https://web.dev/i18n/en/webtransport/
16 WebTransport specification. – URL: https://www.w3.org/TR/webtransport/
17 Кодовая база приложения. – URL: https://github.com/vladKrk/quic_chat