УДК 004.728.8
DOI: http://dx.doi.org/10.21686/1818-4243-2024-5-4-12

Г.А. Звонарёва, А.С. Смирнов
Московский Авиационный Институт, Москва, Россия

Аналитическая оценка работы протоколов WebSocket и WebTransport для класса SCADA-систем, работающих в рамках интрасети

Цель исследования. *В последние годы веб-технологии стали активно применяться во многих системах по управлению технологическими процессами, в том числе и в SCADA. Подобные системы позволяют осуществлять непрерывный мониторинг за объектом испытания в режиме реального времени. В связи с этим возникает необходимость частой передачи данных для их дальнейшего отображения. Для решения этой задачи используются сетевые протоколы, которые продолжают стремительно развиваться и по сей день.*
Материалы и методы. *SCADA-системы применяются во многих отраслях промышленности: авиации, наземном транспорте, ракетно-космической технике, системах приема и обработки телеметрической информации. Среди них выделяется класс SCADA-систем, работающих в рамках интрасети. Это может быть внутренняя сеть предприятия, ангар или небольшой тестовый стенд для отслеживания показателей измерительной аппаратуры. Поэтому сегодня, как никогда, важны открытые и общедоступные материалы, для понимания современных сетевых решений, позволяющих сократить временные затраты по вопросам передачи данных и их дальнейшей визуализации. Подобные сведения будут представлять особую ценность для студентов, изучающих сетевые и веб-технологии.*
Результаты. *В результате исследований с учетом ограничений разрабатываемой SCADA-системы и сети определены временные характеристики для протоколов WebSocket и WebTransport, а также предлагается наиболее оптимальное решение для типовой нагрузки.*
В представленной работе рассматриваются вопросы передачи данных между клиентской и серверной частью для класса SCADA-систем, работающих в рамках интрасети. В связи с этим предлагается рассмотреть два наиболее подходящих для данной задачи решения. Это сетевые протоколы прикладного уровня модели OSI: WebSocket и WebTransport. Кроме того, учитываются протоколы транспортного уровня (TCP и QUIC), поскольку они также могут влиять на временные характеристики.
Вопросы передачи данных рассматриваются в контексте системы отображения и мониторинга данных SCADA-системы. Серверная часть разработана с использованием C#, а клиент – при помощи JavaScript. Проводится ряд экспериментов для граничных случаев и типовой нагрузки, на основе которых определяется зависимость времени передачи данных от их объема.
Заключение. *Приведенный в работе подход по сбору сетевого трафика и анализу временных характеристик также будет полезен в рамках учебного процесса для обучения студентов по курсам «Компьютерные сети» и «Интернет-технологии».*

Ключевые слова: *SCADA, WebSocket, WebTransport, TCP, QUIC.*

Galina A. Zvonareva, Artyom S. Smirnov
Moscow Aviation Institute, Moscow, Russia

Analytical Evaluation of the WebSocket and WebTransport Protocols for the Class of SCADA Systems Operating Within the Intranet

The purpose of research. *In recent years, web technologies have been actively used in many process control systems, including SCADA. Such systems allow continuous monitoring of the test object in real time. In this regard, there is a need for frequent data transmission for their further display. To solve this problem, network protocols are used, which continue to develop rapidly to this day.*
Materials and methods. *SCADA systems are used in many industries: aviation, ground transportation, rocket and space technology, systems for receiving and processing telemetry information. Among them, a class of SCADA systems operating within the intranet stands out. This can be an internal network of the enterprise, a hangar or a small test stand for tracking the indexes of measuring equipment. Therefore, today, more than ever, open and publicly available materials are important for understanding modern network solutions that reduce the time spent on data transmission and their further visualization. Such information will be of particular value to students studying network and web technologies.*
Results. *As a result of the research, taking into account the limitations of the SCADA system and network being developed, the time characteristics for the WebSocket and WebTransport protocols have been determined, and the most optimal solution for a typical load is proposed.*
In the presented paper, the issues of data transfer between the client and server parts for a class of SCADA systems operating within an intranet are considered. In this regard, it is proposed to consider the two most suitable solutions for this task. These are application layer network protocols of the OSI model: WebSocket and WebTransport. In addition, transport layer protocols (TCP and QUIC) are taken into account, since they can also affect time characteristics.
Data transmission issues are considered in the context of the SCADA data display and monitoring system. The server part is developed using C#, and the client is developed using JavaScript. A number of experiments are carried out for boundary cases and typical loads, on the basis of which the dependence of data transmission time on their volume is determined.
Conclusion. *The approach given in the paper on collecting network traffic and analyzing time characteristics will also be useful as part of the educational process for teaching students in the courses "Computer Networks" and "Internet Technologies".*

Keywords: *SCADA, WebSocket, WebTransport, TCP, QUIC.*

Введение

Для решения вопросов автоматического управления сложными технологическими процессами традиционно используются так называемые АСУ ТП (автоматизированная система управления технологическим процессом) системы. Одним из их компонентов являются SCADA-системы (Supervisory Control And Data Acquisition), основные концепции которых включают в себя: дружественность человеко-машинного интерфейса, полноту и наглядность представляемой информации, доступность элементов управления. Подобные системы осуществляют сбор, обработку, отображение и хранение измеренных данных. В последние годы в связи с появлением новых технологий возникла необходимость в замене традиционных АСУ ТП систем [1]. Одним из решений является внедрение современных сетевых и веб-технологий [2].

Рассматриваемый класс SCADA-систем работает в рамках интрасети и представляет собой клиент-серверное приложение, состоящее из множества функциональных модулей. Одним из таких модулей является система отображения и мониторинга измеренных данных. Серверная часть осуществляет получение подготовленных данных от других компонентов SCADA-системы и передает их клиенту. На стороне клиента полученные данные визуализируются в виде графиков и таблиц в режиме реального времени.

В связи с этим возникает необходимость частой отправки новых порций данных между клиентом и сервером. При этом данные должны быть переданы клиенту в полном объеме для их корректного отображения. В связи с этим необходимо обеспечить надежную передачу данных, то есть без потерь. Кроме того, разработка канала передачи данных для системы отображения и мониторинга ведется на прикладном уровне в сетевой модели OSI. В связи с этим необходимо выбрать такую сетевую технологию, которая позволит сократить временные затраты по передаче данных. В качестве возможных вариантов предлагается рассмотрение двух сетевых стеков: TCP|TLSv1.3|HTTP/2|Web Socket и QUIC|TLSv1.3|HTTP/3|WebTransport.

Для решения поставленной задачи был разработан канал передачи данных для каждого сетевого стека. В качестве языков программирования были выбраны C# и JavaScript. Каждый из них имеет свою реализацию сетевых протоколов и соответствующие ограничения, которые будут рассмотрены в этой статье. Однако все они опираются на спецификации RFC (Request for Comments).

Для определения времени передачи данных используются специальные программы-снифферы, которые позволяют записывать проходящие через устройство сетевые пакеты. Для более точной оценки полученных данных отправляется 2000 сообщений. А для анализа записанных файлов с сетевым трафиком был разработан скрипт на Python. Также учитываются особенности двух сетевых стеков при сборе данных.

В статье рассматриваются современные сетевые протоколы передачи данных, в том числе и те, что находятся на стадии разработки. Подобные сведения могут быть полезны для студентов, обучающихся по курсам «Компьютерные сети» и «Интернет-технологии». Кроме того, подход по сбору сетевого трафика и его анализа может использоваться не только в промышленности, но и для учебного процесса.

1. Особенности рассматриваемых сетевых протоколов

1.1 Протоколы TCP и QUIC

В основе двух сетевых стеков лежат транспортные протоколы: TCP и QUIC. Долгое время главным транспортным протоколом в сети был протокол TCP. Однако он обладает рядом недостатков, которые могут влиять на скорость передачи данных. Среди наиболее существенных недостатков можно выделить: особенности установки соединения (используется тройное рукопожатие, а при смене IP-адреса или порта соединение разрывается), проблема Head-of-line (HoL) blocking. Частично их влияние удалось снизить в протоколе HTTP/2. Однако большинство проблем TCP все еще оставались.

Поэтому был разработан новый транспортный протокол – QUIC. Изначально это была разработка компании Google, представленная в 2012 году, но уже к 2021 году QUIC был переработан и стандартизирован. Поскольку QUIC существенно отличается от TCP, то для него был изменен протокол HTTP, получивший третью версию.

Среди наиболее важных особенностей протокола QUIC выделяют [3, 4]: интеграцию с протоколом TLSv1.3, поддержку нескольких независимых потоков байт (решение проблемы Head-of-line (HoL) blocking), идентификаторы соединений и использование концепции фреймов. Кроме того, QUIC поддерживает несколько режимов передачи данных в виде однонаправленных и двунаправленных потоков, а также дейтаграмм [5]. Такой подход позволяет обеспечить как надежную, так ненадежную передачу данных.

Однако TCP и QUIC являются лишь частью сетевого стека протоколов. При разработке приложений, требующих двунаправленной передачи

данных, обычно используют либо разработанный в 2011 году протокол WebSocket, либо находящийся в процессе стандартизации протокол WebTransport.

1.2 WebSocket

До появления протокола WebSocket для организации двунаправленного канала обмена данными между клиентом и сервером использовался протокол HTTP. Этот подход приводил к ряду проблем [6]. Во-первых, необходимо было открывать несколько TCP-соединений: одно для отправки информации клиенту и по одному для каждого сообщения от клиента. Во-вторых, появляются накладные расходы из-за заголовка HTTP. В-третьих, необходимо сопоставлять множество соединений друг с другом на стороне клиента. Протокол WebSocket позволяет решить все эти проблемы путем использования одного TCP-соединения для трафика в обоих направлениях.

Работу с WebSocket можно разделить на два этапа: установку соединения (рукопожатие) и передачу данных. Этап рукопожатия необходим для того, чтобы клиент и сервер могли договориться о возможности установки нового соединения и согласовать общие параметры. На втором этапе клиент и сервер могут обмениваться данными в виде «кадров», а для снижения накладных расходов заголовки этих порций данных представлены в бинарном виде [6]. Сами «кадры» разделяются на три типа: управляющие, текстовые и бинарные.

Протокол WebSocket был стандартизирован в 2011 году и используется во многих современных приложениях. Он имеет широкую поддержку, прост в использовании и не подвержен постоянным изменениям. Кроме того, протокол WebSocket может обеспечивать как безопасную, так и не безопасную передачу данных. При безопасном варианте WebSocket работает с протоколом TLS версии 1.2 или 1.3.

Однако протокол WebSocket, используемый совместно с HTTP/2 и более ранним версиями, а также TCP, имеет ряд недостатков [7]. Одной из главных проблем является проблема Head-of-line (HoL) blocking.

1.3 WebTransport

WebTransport – это протокол, который также называют фреймворком [7], целью которого является абстрагирование от транспортного протокола, при этом предоставляя разработчикам несколько ключевых аспектов транспортного уровня. Например, дейтаграммы, потоки, сеанс.

На данный момент WebTransport является новой технологией и проходит процесс стандартизации. В описании рабочей группы указаны следующие цели [8]: разработка протокола или набора протоколов прикладного уровня, которые поддерживают ряд простых методов связи. Должна быть поддержка ненадежных сообщений, надежных сообщений и упорядоченных потоков надежных сообщений. Также особое внимание уделяется вопросам: производительности, в частности, проблеме Head-of-line (HoL) blocking. Кроме того, WebTransport должен работать совместно с уже существующими протоколами, такими как QUIC и TCP/TLS.

К протоколу WebTransport предъявляется ряд обязательных требований [7]. Например, протокол должен работать совместно с TLSv1.3 и более поздней версии. Клиент и сервер должны договориться о возможности отправки данных, например, установить соединение по TCP или QUIC. Одновременно может быть установлено несколько соединений, что решает ранее указанную проблему протокола WebSocket.

Протокол WebTransport поддерживает два вида передачи данных: через дейтаграммы и потоки. При работе с дейтаграммами основной целью было сделать их похожими по поведению на UPD. Размер дейтаграмм соотносится с MTU. Сам же протокол WebTransport должен обеспечивать отправку и получение дейтаграмм, а также определять их максимальный размер.

В свою очередь, работа с потоками должна быть похожа на потоки в QUIC. Здесь также выделяется два вида потоков: однонаправленные и двунаправленные. Данные в потоке управляются транспортным протоколом. Протокол WebTransport должен обеспечивать создание и выборку потоков, отправку и получение данных, а также прерывание этих операций.

В рамках поставленной задачи на стороне сервера используется реализация протокола WebTransport от компании Microsoft в среде .NET. Поскольку протокол все еще находится на стадии стандартизации, то на данный момент разработчики .NET еще не реализовали весь описанный в черновике функционал [9]. Так, например, ведутся работы по внедрению возможности передачи данных через дейтаграммы [10]. Поэтому в рамках данной статьи рассматривается надежная передача данных через двунаправленные потоки. Кроме того, такой подход позволяет более точно оценить временные затраты по вопросам передачи данных между WebSocket и WebTransport.

Для реализации потоков со стороны сервера используется высокопроизводительная библиотека System.IO.Pipelines [11]. Полноценно реализовать двунаправленный поток в среде .NET на данный момент не представляется возможным, поэтому для операций чтения и записи создаются два одно-

направленных потока. Каждый из них создается и обрабатывается одним из классов библиотеки System.IO.Pipelines: PipeWriter – для записи, PipeReader – для чтения [12].

Со стороны клиента используются концепция сетевых потоков или же Streams API. Это полезный набор инструментов, позволяющий JavaScript получать доступ к потокам данных, проходящих по сети. Здесь также используется два вида потоков: readable – для чтения и writable для записи [13].

2. Методика оценки работы протоколов WebSocket и WebTransport

Как упоминалось ранее, для системы отображения и мониторинга данных критическим важным является вопрос сокращения времени передачи данных. Поэтому для оценки временных интервалов был проведен ряд экспериментов для поиска наиболее подходящего технологического стека.

Для этих целей был разработан канал передачи данных, представляющий собой клиент-серверное приложение. Сервер реализован на C# в рамках среды .NET, а клиент с помощью HTML, CSS и JS. На стороне сервера формируются данные, а затем через определенные промежутки времени передаются протоколу: WebSocket или WebTransport. На стороне клиента осуществляется прием полученных данных. Для протокола WebTransport дополнительно реализуется цикл чтения и сбора данных.

Передаваемые данные представляют собой объект, содержащий информацию о параметрах измерительных каналов устройства: идентификатор канала, координаты для графиков, а также статистические данные. Объект переводится в бинарный вид для сокращения объема. Затем на стороне клиента бинарные данные обратно переводятся в объект. Для корректного отображения графиков и таблиц на стороне клиента важно получать полную информацию о всех измерительных каналах.

В рамках проводимых испытаний необходимо получить зависимость временных интервалов от объема передаваемых данных для обоих сетевых стеков и разных видов сети Ethernet: 1 Гб и 10 Мб. В качестве временных интервалов между отправляемыми сообщениями были выбраны 0,5 сек. (отсутствие перегрузок и потерь пакетов) и 0,02 сек. (типовая нагрузка). Такой подход позволяет рассмотреть граничные случаи, исходя из которых при необходимости для промежуточных значений можно определить наилучший в рамках данной задачи стек технологий.

Передаваемые данные могут иметь следующий объем: 16 КБ, 32 КБ, 64 КБ, 128 КБ, 256 КБ и 512 КБ. Поскольку размер сетевых параметров часто связан со степенью 2, то и для анализа полученных результатов были выбраны соответствующие объемы передаваемых данных. Данные больше 512 КБ в представленной работе не рассматриваются, поскольку для типовой нагрузки характерен объем от 128 КБ до 512 КБ.

Поскольку рассматриваемая SCADA-система работает в рамках интрасети, то допускается, что потери пакетов либо отсутствуют, либо являются незначительными (менее 1 %). Кроме того, из-за необходимости иметь полный набор данных для корректного отображения графиков и дополнительных сложностей с синхронизацией на клиенте, в данной работе не рассматривается подход с мультиплексированием нескольких потоков.

Испытательный стенд представляет собой два компьютера, соединенных в рамках интрасети через коммутатор с помощью Ethernet. В качестве операционной системы используется Windows 11, поскольку она содержит необходимые библиотеки для корректной работы сетевого стека QUIC|TLSv1.3|HTTP/3|WebTransport.

Для получения информации о сетевых пакетах и их временных характеристиках используются программа-сниффер Wireshark и ее упрощенный вариант Dumpcap. Wireshark – это одно из наиболее популярных решений для перехвата траффика. Данная программа позволяет отображает пакеты, проходящие через сетевой стек компьютера, на котором она установлена [14]. Структуру подобных пакетов можно подробно изучить: как заголовки с информацией от сетевых протоколов разного уровня, так и их полезную нагрузку. Также Wireshark добавляет к каждому пакету метаинформацию, например, время перехвата пакета.

Dumpcap – это утилита командой строки, которая как и Wireshark позволяет перехватывать сетевые пакеты и записывать их в файл [15]. Однако Dumpcap, в отличие от Wireshark, имеет сильно упрощенный графический интерфейс и не анализирует трафик во время его перехвата [16]. Такой подход позволяет снизить нагрузку на память и процессор устройства, на котором запущена программа-сниффер.

Wireshark и Dumpcap состоят из множества модулей, в число которых входит npcap. Это библиотека и сетевой драйвер, разработанные под операционную систему Windows [17]. Они позволяют перехватывать трафик перед его попаданием на драйвер мини-порта, то есть непосредственно перед началом работы сетевой карты.

Для анализа процессов, происходящих в операционной системе и ее сетевом стеке во время испытаний, используются программы-снифферы

Windows performance recorder (WPR) – для записи и Windows performance analyzer (WPA) – для анализа и визуализации записанной информации. В своей основе оба инструмента используют службу трассировки Windows (ETW).

ETW работает следующим образом: приложения делятся своими событиями с сессиями ETW, которые могут предоставлять их пользователю либо в режиме реального времени, либо в виде файлов логирования с форматом *.etl [18]. Таким образом, WPA через ETW записывает происходящие в системе события, а WPA визуализирует их в виде графиков и таблиц.

Процесс передачи и приема данных отличается для двух сетевых стеков: TCP|TLS|HTTP/2|WebSocket и QUIC|TLSv1.3|HTTP/3|WebTransport. Их можно представить в виде нескольких временных интервалов, пересекающихся друг с другом, что изображены на рис. 1 и рис. 2.

Таким образом, основным отличием при работе с двумя сетевыми стеками является сокрытие процесса сбора данных на клиенте в протоколе WebSocket, то есть программисту предоставляется уже готовый к работе набор данных. В противовес этому по протоколу WebTransport данные предоставляются программисту по мере их поступления. Такой подход позволяет обрабатывать порции данных, не дожидаясь окончания процесса передачи данных. Однако, если разработчику необходим полный объем данных, то он должен сам реализовать их сбор в единый массив.

Из-за различий между рассматриваемыми сетевыми стеками и наличии проблем синхронизации локального времени двух компьютеров был принят следующий подход к сбору временных характеристик. После получения полного объема данных в приложении

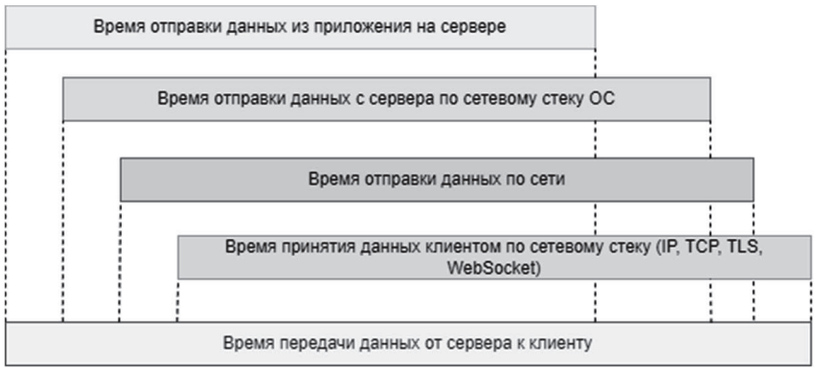

Рис. 1. Схема пересечения временных интервалов при передаче данных с использованием стека TCP|TLS|HTTP/2|WebSocket

Fig. 1. Scheme of time interval intersection during data transmission using TCP|TLS|HTTP/2|WebSocket stack

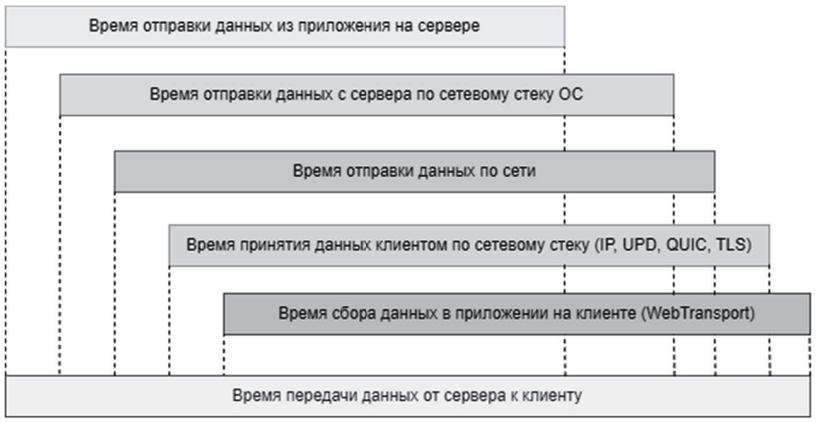

Рис. 2. Схема пересечения временных интервалов при передаче данных с использованием стека QUIC|TLSv1.3|HTTP/3|WebTransport

Fig. 2. Scheme of time interval intersection during data transfer using QUIC|TLSv1.3|HTTP/3|WebTransport stack

клиента отправляется ответное сообщение. Общее время передачи данных от сервера к клиенту определяется как разность двух временных меток. В приложении сервера фиксируется время начала отправки данных T_1, то есть время перед вызовом функции SendAsync для WebSocket и функции WriteAsync для WebTransport. А второй временной меткой T_2 является время перехвата на стороне сервера ответного сообщения.

Каждый эксперимент представляет собой передачу 2000 сообщений определенного объема. Время T_1 для всех сообщений записывается в excel-файл после окончания процесса передачи данных. Для подсчета временных интервалов был разработан скрипт на Python. Он анализирует записанный в ходе эксперимента через Dumpcap файл с сетевым трафиком с помощью библиотеки pyshark и осуществляет поиск всех T_2. Затем подсчитывает разницу между T_2 и T_1, формируя время передачи данных для каждого сообщения.

3. Анализ полученных результатов

Для наиболее точной оценки времени передачи данных была рассмотрена ситуация, при которой отсутствовали перегрузки и потери данных. Для этого данные передавались каждые 0,5 сек. Полученные результаты представлены на рис. 3 и рис. 4.

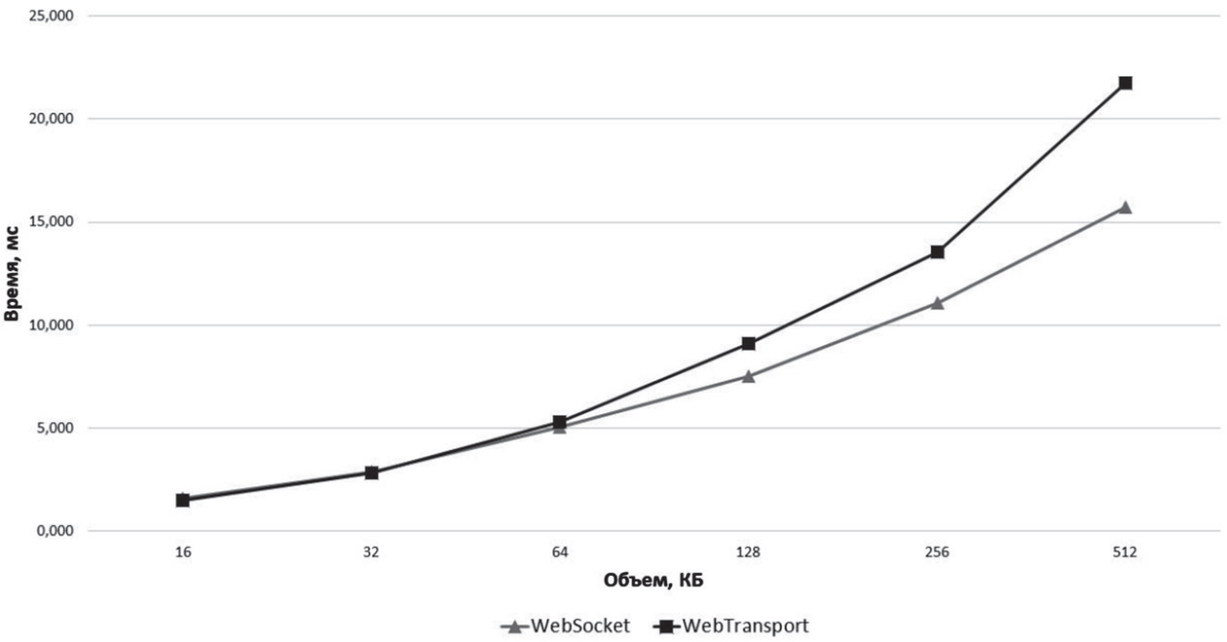

Рис. 3. Время передачи данных для 1 Гб Ethernet
Fig. 3. Data transfer time for 1 Gb Ethernet

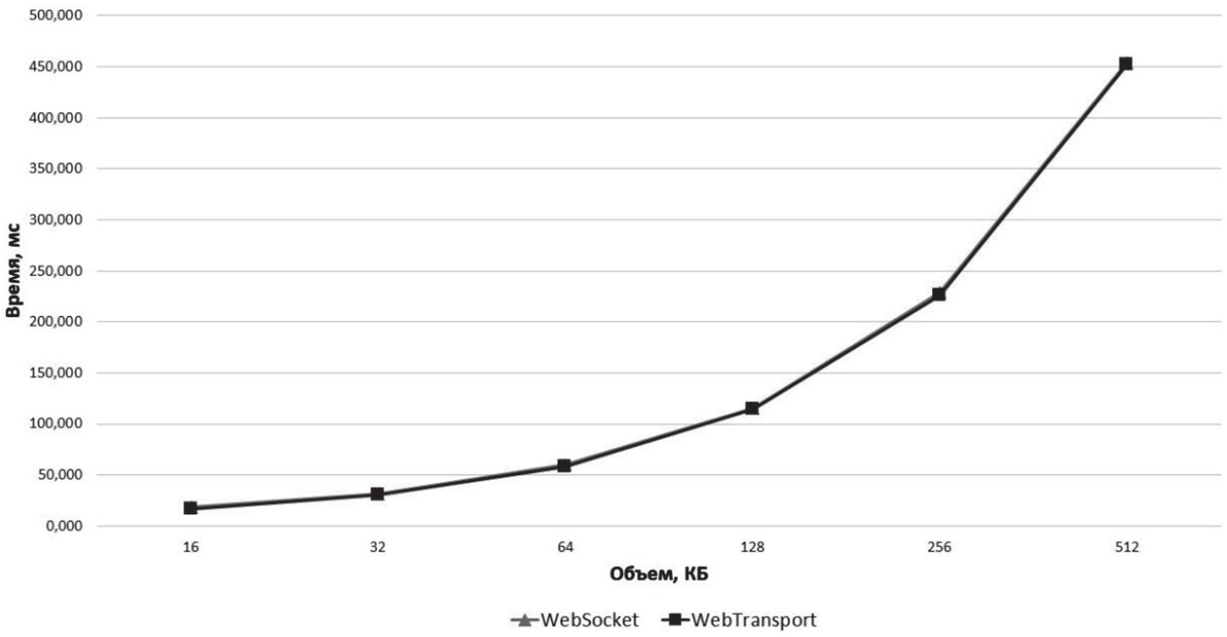

Рис. 4. Время передачи данных для 10 Мб Ethernet
Fig. 4. Data transfer time for 10 Mb Ethernet

Таким образом, реализация сетевого стека TCP|TLSv1.3|HTTP/2| WebSocket продемонстрировала меньшие затраты по времени передачи данных по мере увеличения их объема по сравнению с QUIC|TLSv1.3|HTTP/3|WebTransport в 1 Гб сети. Однако при 10 Мб сети результаты оказались практически идентичны.

Во-первых, разница во времени передачи данных в 1 Гб сети может быть вызвана дорогими по времени операциями на стороне клиента при использовании WebTransport. Из-за особенностей реализации протокола WebTransport разработчик имеет доступ только к стандартным читающим и пишущим потокам, в которых нельзя задавать размер обрабатываемой порции данных. Поскольку заранее не представляется возможным указать потоку какой объем данных требуется накопить, то возникает необходимость постоянного опроса читающего потока на предмет наличия новых данных.

Кроме того, из-за необходимости иметь полный набор

данных для их корректной визуализации и дополнительных сложностей по вопросам синхронизации, в данной работе не могли быть использованы потенциальные преимущества протоколов QUIC и WebTransport по мультиплексированию потоков, а следовательно и проблема Head-of-line (HoL) blocking.

Также стоит учитывать, что работа по сбору данных в протоколе WebTransport реализована на JavaScript и может уступать по производительности протоколу WebSocket, написанному на языке C и встроенному непосредственно в браузер.

Во-вторых, на временные характеристики передаваемых данных влияют параметры сети, в частности окно отправителя (SndWnd), окно перегрузки (CWnd) и окно получателя (RcvWnd).

В стеке TCP|TLS|HTTP/2|WebSocket окно отправителя и получателя по умолчанию ограничено до 65535 КБ. Однако для эффективного использования сетей с высокой пропускной способностью оно может автоматически увеличиваться операционной системой за счет параметра масштабирования TCP. Размер самого окна рассчитывается следующим образом [19]:

$$SndWnd_{new} = 65535 * 2^{\text{параметр масштабирования } TCP}.$$

В противовес данному подходу, в стеке QUIC|TLSv1.3|HTTP/3|WebTransport окна отправителя и получателя могут по умолчанию достигать значений до нескольких МБ [20], что в теории может позволить отправлять большие объемы данных, чем в стеке TCP|TLS|HTTP/2|WebSocket.

Рассмотрим изменение данных параметров на примере передачи 128 КБ данных. Для этого были проанализированы записанные во время эксперимента файлы .etl в WPA и на основе них построены графики изменения сетевых параметров для обоих сетевых стеков, представленные на рис. 5, рис. 6.

Для 1 Гб сети параметр масштабирования окон в стеке TCP|TLS|HTTP/2|WebSocket больше 1, окно отправителя равняется 2097120 байт, а окно получателя составляет 262656 байт. При этом отправляемые данные не достигают максимального значения окна перегрузки, поэтому оно не оказывает влияния на время передачи данных. Для сетевого стека QUIC|TLSv1.3|HTTP/3|WebTransport окно отправителя и получателя имеют значения свыше 1 МБ, что так же позволяет отправлять максимальный объем данных. Таким образом, в данном случае сетевые параметры не влияют на временные характеристики. Однако из-за обозначенных ранее проблем со сбором данных на стороне клиента, после 64 КБ становится заметна разница во временных затратах.

Далее на рис. 7 и рис. 8 представлены результаты для 10 Мб сети.

Для 10 Мб сети параметр масштабирования окон для TCP|TLS|HTTP/2| WebSocket оказался равен 0, то есть окно получателя и отправителя имеет максимальный размер в 64 КБ. Из-за этого окно перегрузки вскоре достигло этого значения, а сами данные начали передаваться с задержкой. Поэтому в низкоскоростной сети стек TCP|TLS|HTTP/2|

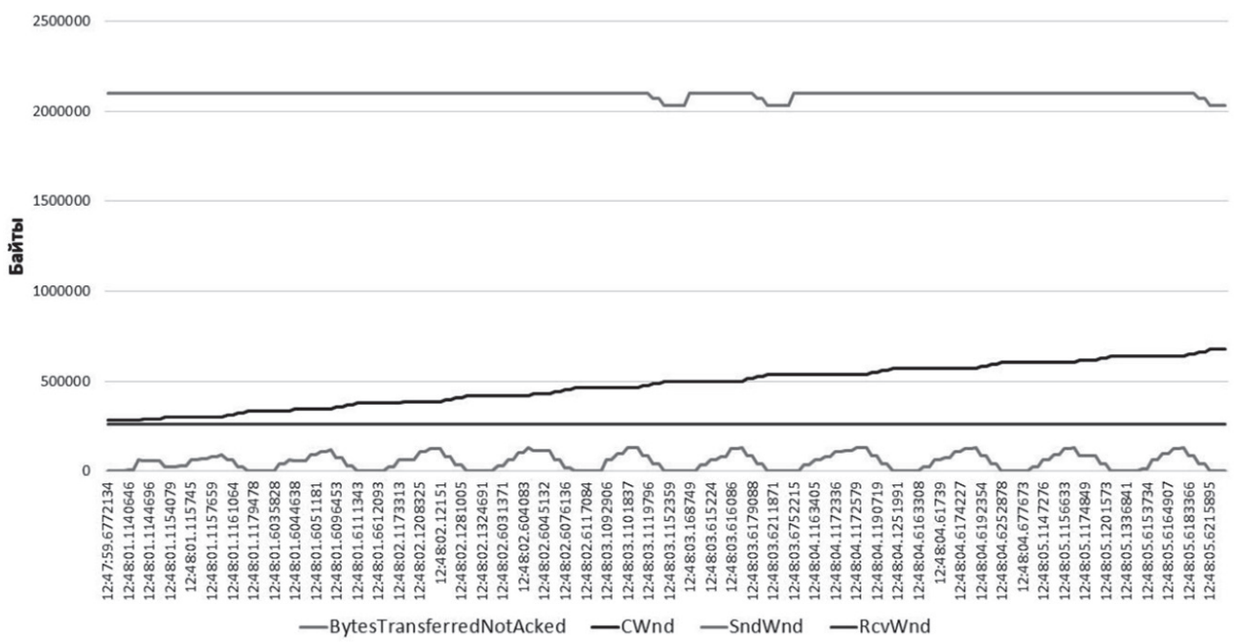

Рис. 5. Сетевые параметры для 1 Гб Ethernet в TCP|TLS|HTTP/2|WebSocket

Fig. 5. Network parameters for 1 Gb Ethernet in TCP|TLS|HTTP/2|WebSocket

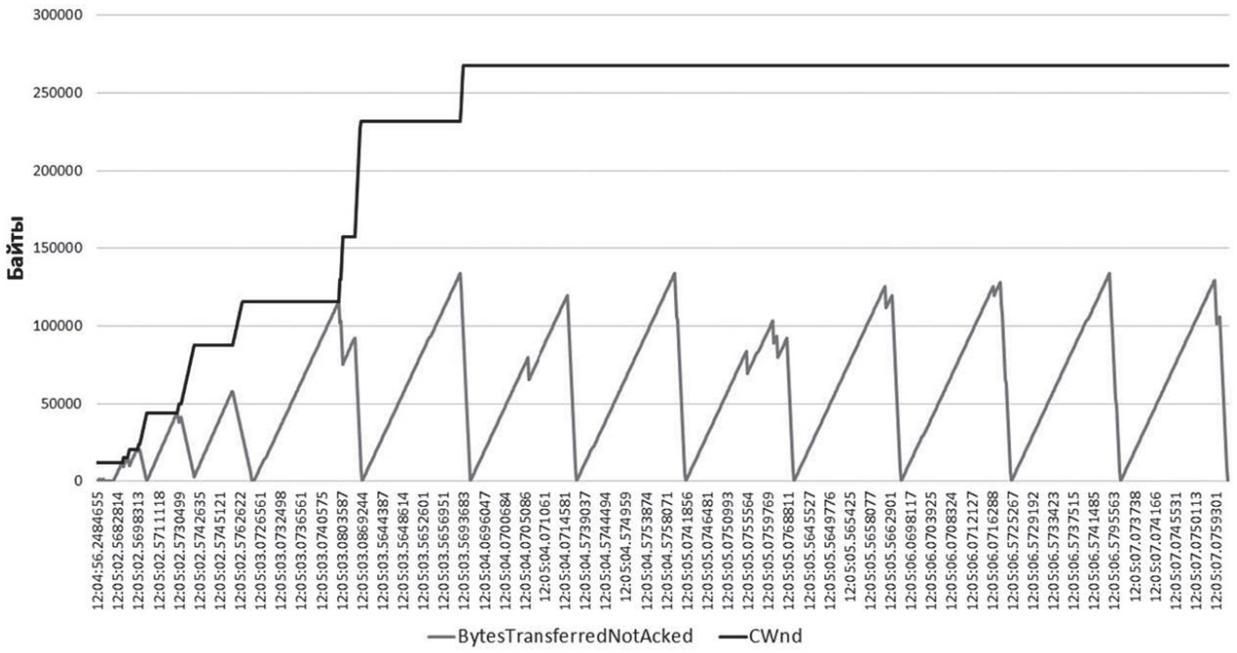

Рис. 6. Сетевые параметры для 1 Гб Ethernet в QUIC|TLSv1.3|HTTP/3|WebTransport
Fig. 6. Network parameters for 1 Gb Ethernet in QUIC|TLSv1.3|HTTP/3|WebTransport

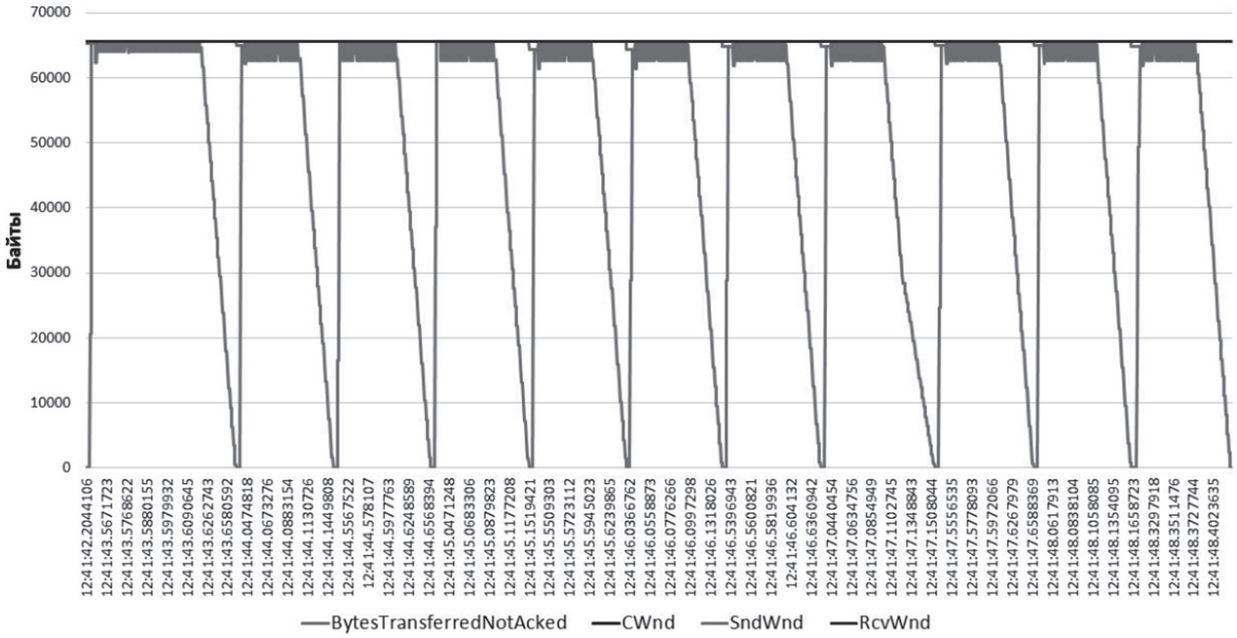

Рис. 7. Сетевые параметры для 10 Мб Ethernet в TCP|TLS|HTTP/2|WebSocket
Fig. 7. Network parameters for 10 Mb Ethernet in TCP|TLS|HTTP/2|WebSocket

WebSocket теряет преимущество в скорости отправки данных, а, следовательно, и во временных характеристиках.

Для сетевого стека QUIC| TLSv1.3| HTTP/3| WebTransport график изменения окна перегрузки оказался практически идентичен, а окно отправителя и получателя так же имеют значения свыше 1 МБ. Что позволяет данному сетевому стеку отправлять большой объем данных. Однако из-за задержек в 10 Мб сети при получении ответного сообщения и пакетов ACK от клиента, а также замедленного сбора данных на клиенте, разница во времени передачи данных между двумя сетевыми стеками практически незаметна.

Далее рассмотрим время отправки данных для типовой нагрузки. Предлагается рассмотреть 1 Гб сеть в рамках интрасети, то есть при отсутствии потерь данных. Сами данные отправляются каждые 0,02 сек. Такой временной интервал был

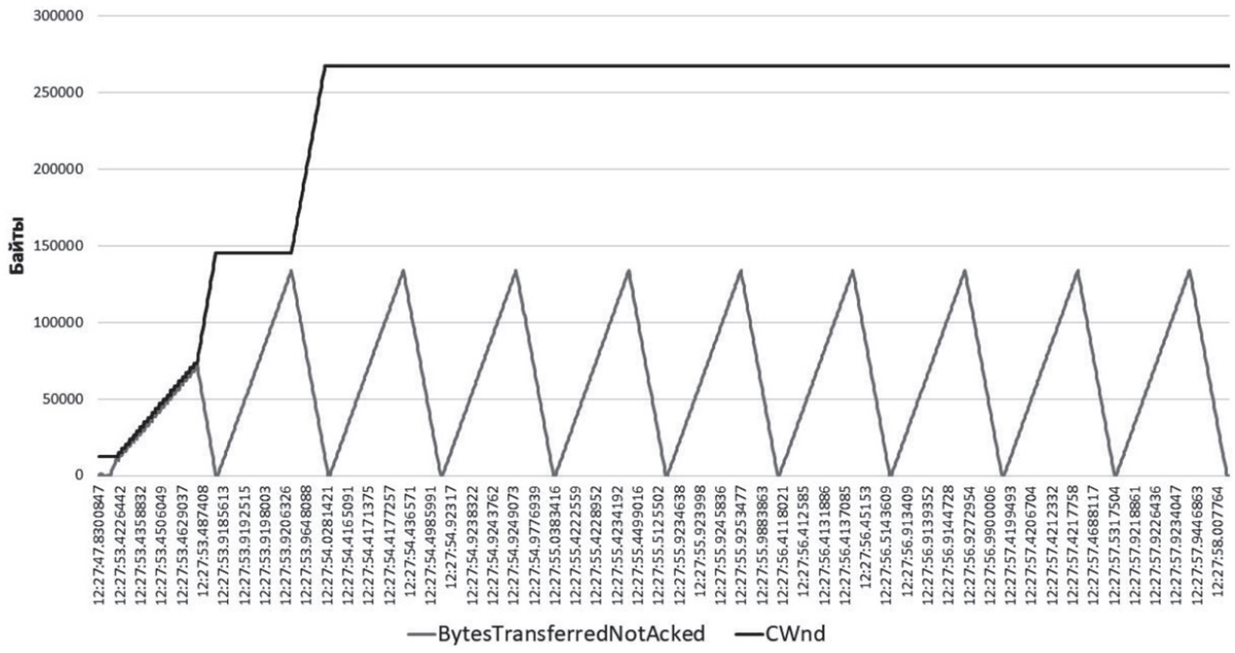

Рис. 8. Сетевые параметры для 10 Мб Ethernet в QUIC|TLSv1.3|HTTP/3|WebTransport
Fig. 8. Network parameters for 10 Mb Ethernet in QUIC|TLSv1.3|HTTP/3|WebTransport

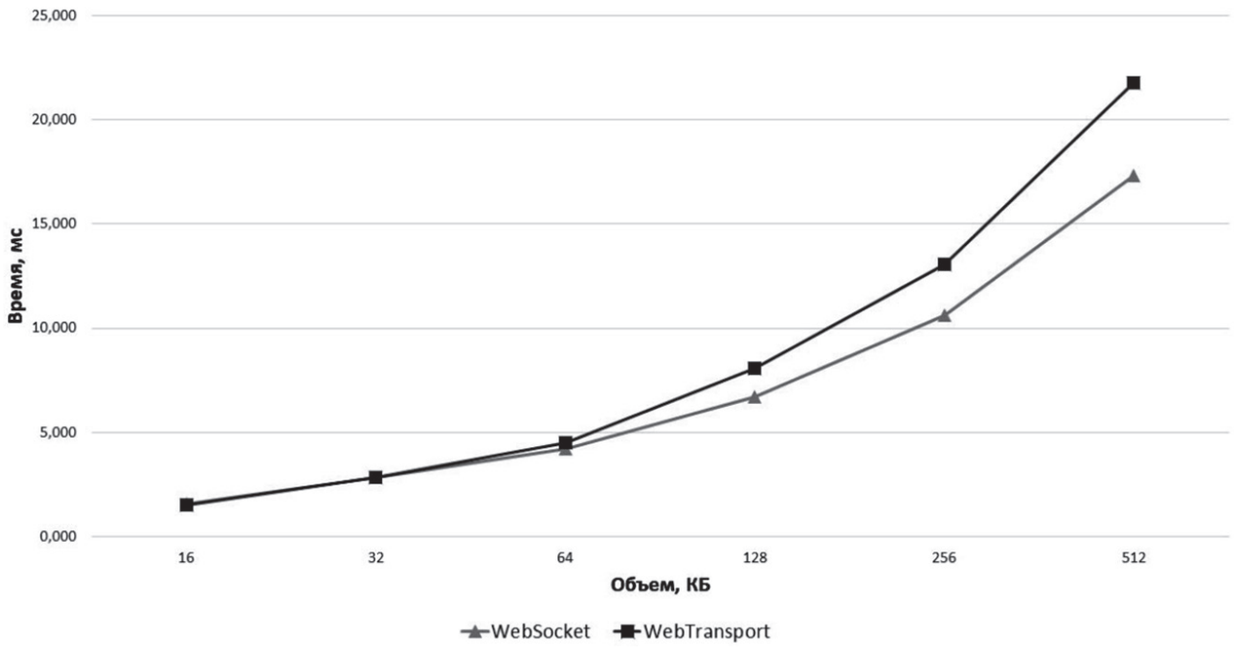

Рис. 9. Время передачи данных для типовой нагрузки
Fig. 9. Data transfer time for a typical load

выбран для наиболее плавного обновления данных на графиках, благодаря чему у пользователя создается впечатление, что график изменяется непрерывно. Полученные результаты представлены на рис. 9.

В результате график зависимости времени передачи данных от объема практически идентичен графику из рис. 3, то есть при отправке данных каждые 0,5 сек. Такой результат обусловлен обозначенными ранее причинами по замедленной сборке данных на клиенте в WebTransport и отсутствии влияния сетевых параметров в 1 Гб сети. Кроме того, исходя из графика на рис. 3, время передачи данных практически для всех представленных объемов составляет менее 0,02 сек. Поэтому дополнительных перегрузок из-за частой отправки данных не наблюдается. При этом, как упоминалось ранее, объем передаваемых данных для рассматриваемой задачи варьируется от 128 КБ

до 512 КБ, поэтому большие объемы в данной работе на учитывались.

Заключение

Таким образом, в рамках данной работы было рассмотрено два сетевых стека: TCP|TLSv1.3|HTTP/2|WebSocket и QUIC|TLSv1.3|HTTP/3|WebTransport, а также их влияние на временные затраты по передаче данных между клиентом и сервером. Использовался надежный подход, то клиент получал данные в правильном порядке и без потерь. Серверная часть была реализована в среде .NET, а клиентская с использованием JavaScript и браузера. Поскольку рассматриваемая SCADA- система работает в рамках интрасети, то потери данных либо отсутствовали, либо были незначительны. Кроме того, для данной задачи не характерен подход с мультиплексированием из-за необходимости получения полного объема данных на клиенте перед их визуализацией.

Было рассмотрено два случая: отсутствие перегрузок и типовая нагрузка. В случае отсутствия перегрузок данные передавались каждые 0,5 сек, а скорость сети составляла 1 Гб и 10 Мб. Для типовой нагрузки характерна передача данных каждые 0,02 сек и 1 Гб сеть. При такой установке до 64 КБ разница между двумя технологиями незначительна. Однако при увеличении объема передаваемых данных возникает существенный разрыв во временных затратах в пользу TCP|TLSv1.3|HTTP/2|WebSocket, вызванный более медленными операциями по сбору данных в протоколе WebTransport. С учетом того, что для типовой нагрузки объем передаваемых данных варьируется от 128 КБ до 512 КБ, наиболее предпочтительным является сетевой стек TCP|TLSv1.3|HTTP/2|WebSocket.

Однако подход с QUIC|TLSv1.3|HTTP/3|WebTransport не стоит сбрасывать со счетов. В этом сетевом стеке используются протоколы и технологии, которые на данный момент активно развиваются. Данный стек демонстрирует схожие с TCP|TLSv1.3|HTTP/2|WebSocket результаты в сетях с низкой пропускной способностью. Кроме того, все еще остается подход с дейтаграммами и ненадежной передачей данных, что может хорошо себя проявить в вопросах передачи аудио и видео.

Данный материал способствует лучшему понимаю современных сетевых протоколов, а также методов сбора и анализа сетевого трафика, и может быть использован в учебном процессе в рамках курсов «Компьютерные сети» и «Интернет-технологии».

Литература

1. Долганов А. В., Минигалиев Г.Б., Елизаров В.В. Интерактивные системы проектирования и управления. Нижнекамск: Нижнекамский химико-технологический институт (филиал) ФГБОУ ВПО «КНИТУ», 2014. 196 с.
2. Варламов И. Г. SCADA нового поколения. Эволюция технологий – революция системостроения // Автоматизация и IT в энергетике. 2016. № 2(79). С. 2–6.
3. Dellaverson J., Li T., Wang Y., Iyengar J., Afanasyev A., Zhang L., A Quick Look at QUIC, UCLA Computer Science, 2021.
4. Marx R., HTTP/3 From A To Z: Core Concepts [Электрон. ресурс]. 2021. Режим доступа: https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/.
5. Iyenger J., Thomson M., QUIC: A UDP-Based Multiplexed and Secure Transport. [Электрон. ресурс]. 2021. Режим доступа: https://datatracker.ietf.org/doc/html/rfc9000.
6. Melnikov A., Fette I., The WebSocket Protocol [Электрон. ресурс]. 2020. Режим доступа: https://datatracker.ietf.org/doc/rfc6455/.
7. Vasiliev V., The WebTransport Protocol Framework [Электрон. ресурс]. 2024. Режим доступа: https://datatracker.ietf.org/doc/draft-ietf-webtrans-overview/.
8. Charter for Working Group [Электрон. ресурс]. 2024. Режим доступа: https://datatracker.ietf.org/wg/webtrans/about/.
9. Genkin D., WebTransport PR follow-ups [Электрон. ресурс]. 2022. Режим доступа: https://github.com/dotnet/aspnetcore/issues/42788.
10. Genkin D., Adding datagrams support to WebTransport [Электрон. ресурс]. 2022. Режим доступа: https://github.com/dotnet/aspnetcore/issues/42784.
11. Fowler D., System.IO. Pipelines: High performance IO in .NET [Электрон. ресурс]. 2018. Режим доступа: https://devblogs.microsoft.com/dotnet/system-io-pipelines-high-performance-io-in-net/.
12. System.IO.Pipelines [Электрон. ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.io.pipelines?view=dotnet-plat-ext-8.0&viewFallbackFrom=net-7.0.
13. Streams API concepts [Электрон. ресурс]. Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Streams_API/Concepts.
14. Sharpe R., Warnicke E., Lamping U., What is Wireshark? [Электрон. ресурс]. Режим доступа: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs.
15. Sharpe R., Warnicke E., Lamping U., D.4. dumpcap: Capturing with “dumpcap” for viewing with Wireshark [Электрон. ресурс]. Режим доступа: https://www.wireshark.org/docs/wsug_html_chunked/AppToolsdumpcap.html.
16. Sharpe R., Warnicke E., Lamping U., dumpcap (1) Manual Page. [Электрон. ресурс].

Режим доступа: https://www.wireshark.org/docs/man-pages/dumpcap.html.

17. Npcap internals [Электрон. ресурс]. Режим доступа: https://npcap.com/guide/npcap-internals.html.

18. Suvorov N., Изучаем Event Tracing for Windows: теория и практика [Электрон. ресурс]. 2018. Режим доступа: https://habr.com/ru/articles/502362/.

19. Описание функций Windows TCP [Электрон. ресурс]. 2023. Режим доступа: https://learn.microsoft.com/ru-ru/troubleshoot/windows-server/networking/description-tcp-features.

20. MsQuic Settings [Электрон. ресурс]. 2023. Режим доступа: https://github.com/microsoft/msquic/blob/main/docs/Settings.md.

References

1. Dolganov A.V., Minigaliyev G.B., Yelizarov V.V. Interaktivnyye sistemy proyektirovaniya i upravleniya = Interactive design and control systems. Nizhnekamsk: Nizhnekamsk Chemical-Technological Institute (branch) of the Federal State Budgetary Educational Institution of Higher Professional Education «KNITU»; 2014. 196 p. (In Russ.)

2. Varlamov I. G. New generation SCADA. Evolution of technologies – revolution of system engineering. Avtomatizatsiya i IT v energetike = Automation and IT in energy. 2016; 2(79): 2-6. (In Russ.)

3. Dellaverson J., Li T., Wang Y., Iyengar J., Afanasyev A., Zhang L., A Quick Look at QUIC, UCLA Computer Science; 2021.

4. Marx R., HTTP/3 From A To Z: Core Concepts [Internet]. 2021. Available from: https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/.

5. Iyenger J., Thomson M., QUIC: A UDP-Based Multiplexed and Secure Transport [Internet]. 2021. Available from: https://datatracker.ietf.org/doc/html/rfc9000.

6. Melnikov A., Fette I., The WebSocket Protocol [Internet]. 2020. Available from: https://datatracker.ietf.org/doc/rfc6455/.

7. Vasiliev V., The WebTransport Protocol Framework [Internet]. 2024. Available from: https://datatracker.ietf.org/doc/draft-ietf-webtrans-overview/.

8. Charter for Working Group [Internet]. 2024. Available from: https://datatracker.ietf.org/wg/webtrans/about/.

9. Genkin D., WebTransport PR follow-ups [Internet]. 2022. Available from: https://github.com/dotnet/aspnetcore/issues/42788.

10. Genkin D., Adding datagrams support to WebTransport [Internet]. 2022. Available from: https://github.com/dotnet/aspnetcore/issues/42784.

11. Fowler D., System.IO.Pipelines: High performance IO in .NET [Internet]. 2018. Available from: https://devblogs.microsoft.com/dotnet/system-io-pipelines-high-performance-io-in-net/.

12. System.IO.Pipelines [Internet]. Available from: https://learn.microsoft.com/ru-ru/dotnet/api/system.io.pipelines?view=dotnet-plat-ext-8.0&viewFallbackFrom=net-7.0.

13. Streams API concepts [Internet]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Streams_API/Concepts.

14. Sharpe R., Warnicke E., Lamping U., What is Wireshark? [Internet]. Available from: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs.

15. Sharpe R., Warnicke E., Lamping U., D.4. dumpcap: Capturing with "dumpcap" for viewing with Wireshark [Internet]. Available from: https://www.wireshark.org/docs/wsug_html_chunked/AppToolsdumpcap.html.

16. Sharpe R., Warnicke E., Lamping U., dumpcap (1) Manual Page [Internet]. Available from: https://www.wireshark.org/docs/man-pages/dumpcap.html.

17. Npcap internals [Internet]. Available from: https://npcap.com/guide/npcap-internals.html.

18. Suvorov N. Изучаем Event Tracing for Windows: теория и практика = Learning Event Tracing for Windows: Theory and Practice [Internet]. 2018. Available from: https://habr.com/ru/articles/502362/.

19. Opisaniye funktsiy Windows TCP = Description of Windows TCP Features [Internet]. 2023. Available from: https://learn.microsoft.com/ru-ru/troubleshoot/windows-server/networking/description-tcp-features. (In Russ.)

Сведения об авторах

Галина Александровна Звонарёва
К.т.н. доцент кафедры Вычислительные машины, системы и сети
Московский Авиационный Институт,
Москва, Россия
Эл. почта: zvonarevagal@yandex.ru

Артём Сергеевич Смирнов
Студент
Московский Авиационный Институт,
Москва, Россия
Эл. почта: artyom.artem2000@yandex.ru

Information about the authors

Galina A. Zvonareva
Cand. Sci. (Technical), Associate Professor of the Department of Computing Machines, Systems and Networks
Moscow Aviation Institute, Moscow, Russia
E-mail: zvonarevagal@yandex.ru

Artyom S. Smirnov
Student
Moscow Aviation Institute,
Moscow, Russia
E-mail: artyom.artem2000@yandex.ru