

Р. В. Морозов^{1✉}, П. Ю. Бугаков¹

Сравнительный анализ методов обмена сообщениями в распределенной системе мониторинга сетевых узлов

¹Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: me@craftersmine.ru

Аннотация. В работе рассматриваются и анализируются подходы к организации обмена сообщениями между компонентами в распределенных системах мониторинга. Проведено сравнение транспортных протоколов TCP и UDP, прикладного протокола HTTP в рамках архитектурного стиля REST, а также протокола WebSocket с точки зрения их применимости в системах, требующих обмена данными в режиме, близком к реальному времени. Особое внимание уделено таким характеристикам, как задержка передачи данных, надежность доставки, поддержка двустороннего обмена, требования к вычислительным ресурсам и особенностям безопасности. На основе проведенного анализа обоснован выбор WebSocket в качестве основного механизма обмена сообщениями между координатором и агентами, а также использование HTTP для односторонней передачи данных от агентов к координатору. Результаты исследования демонстрируют, что комбинированное использование WebSocket и HTTP позволяет обеспечить баланс между производительностью, масштабируемостью и сложностью реализации распределенной системы мониторинга.

Ключевые слова: сети, обмен сообщениями, протоколы связи, tcp, udp, http, websockets

R. V. Morozov^{1✉}, P. Yu. Bugakov¹

Comparative analysis of messaging methods in a distributed system for monitoring network nodes

¹Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: me@craftersmine.ru

Abstract. The work examines and analyzes approaches to organizing message exchange between components in distributed monitoring systems. A comparison is conducted of the TCP and UDP transport protocols, the HTTP application protocol within the REST architectural style, and the WebSocket protocol in terms of their applicability in systems requiring data exchange in near real-time conditions. Special attention is given to characteristics such as data transmission latency, delivery reliability, support for bidirectional communication, computational resource requirements, and security aspects. Based on the analysis performed, WebSocket is justified as the primary mechanism for message exchange between the coordinator and agents, while HTTP is used for unidirectional data transmission from agents to the coordinator. The results of the study demonstrate that the combined use of WebSocket and HTTP provides a balance between performance, scalability, and implementation complexity in a distributed monitoring system.

Keywords: networks, message exchange, communication protocols, tcp, udp, http, websockets

Введение

В современных вычислительных и сетевых системах существует широкий спектр способов организации обмена данными между узлами, от передачи сырых

байтовых потоков до использования высокоуровневых проприетарных протоколов прикладного уровня. Особенно критичную роль выбор механизма обмена играет в системах мониторинга и управления, где требуется своевременный обмен информацией о событиях, метриках производительности, состоянии узлов, а также команд управления и сценариев реагирования на инциденты, связанные с непредвиденной остановкой работающих служб, ошибками в их работе, остановкой рабочих станций или серверов.

Главная особенность таких систем – распределенная архитектура, в которой есть отдельный координатор и множество агентов, работающих на управляемых узлах (рис. 1).

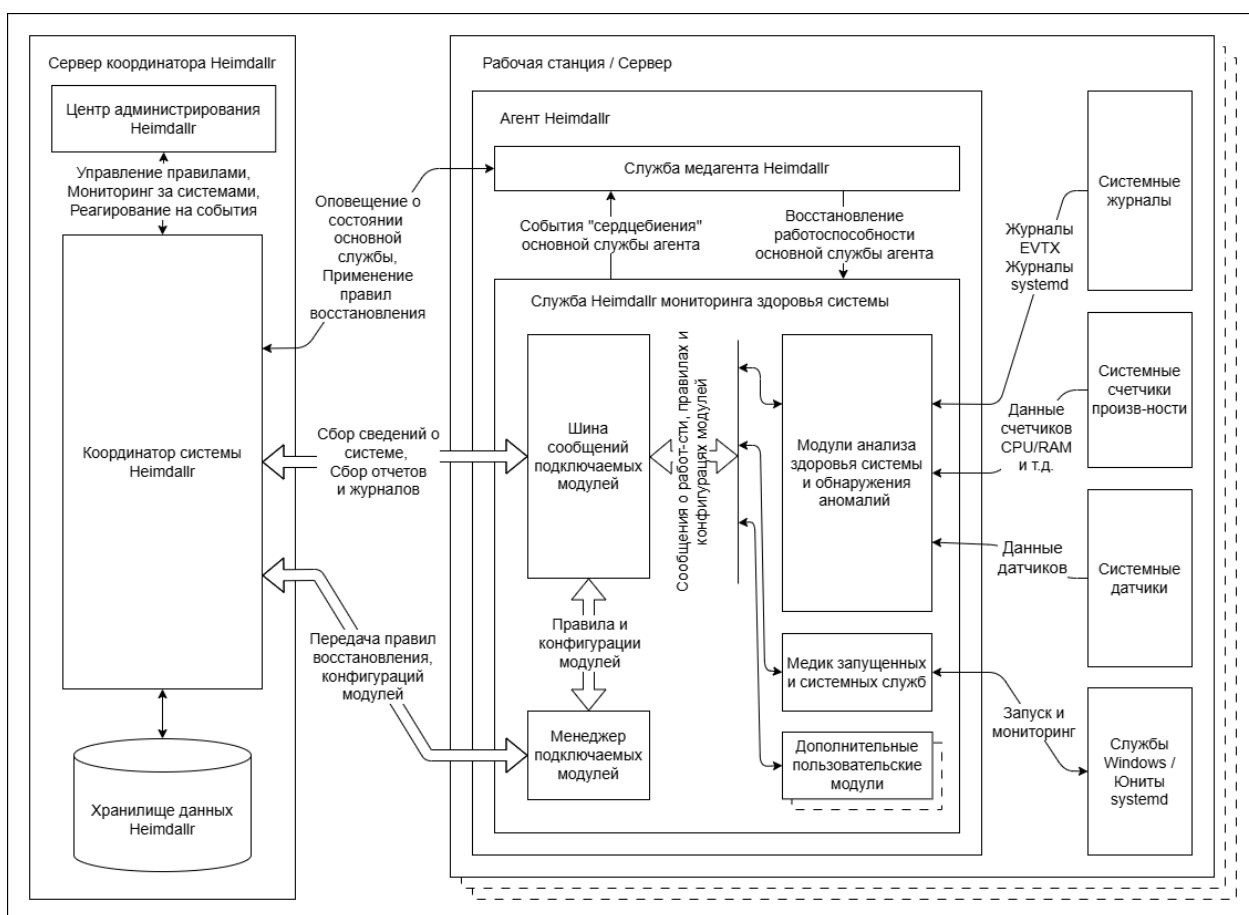


Рис. 1. Концептуальная схема архитектуры

Взаимодействие между этими компонентами должно соответствовать ряду важных требований: обеспечивать низкие задержки при передачи данных, поддерживать двусторонний обмен сообщениями, сохранять устойчивость к сбоям и эффективно использовать вычислительные ресурсы при сериализации и десериализации данных. Не менее важным является и высокий уровень защиты передаваемой информации.

Отдельное внимание уделяется безопасности, поскольку агенты обычно обладают расширенными привилегиями в системе. Ошибки или ненадежная реали-

зация механизма обмена сообщениями могут привести к компрометации всей инфраструктуры и вызвать серьезные последствия.

С учетом большого количества существующих сетевых протоколов и архитектурных подходов возникает задача выбора наиболее подходящего способа организации взаимодействия между координатором и агентами. Такой выбор должен опираться на сравнительный анализ различных технологий с точки зрения производительности, надежности, сложности реализации и уровня безопасности.

Способы сетевого взаимодействия

В рамках исследования были рассмотрены четыре распространенных подхода к организации сетевого взаимодействия: транспортные протоколы TCP и UDP, прикладной протокол HTTP, используемый в архитектуре REST, а также протокол WebSocket. Эти технологии относятся к разным уровням сетевой модели и обладают собственными особенностями, поэтому для выбора подходящего решения требуется их сравнительный анализ [1, 2].

TCP (Transmission Control Protocol) представляет собой транспортный протокол, функционирующий поверх IP и обеспечивающий надежную передачу данных между узлами сети. Для этого в протоколе предусмотрены процедура установления логического соединения, проверка целостности передаваемых данных, подтверждение факта доставки и повторная отправка утерянных сегментов. Благодаря перечисленным механизмам TCP гарантирует корректную и упорядоченную доставку информации. Вместе с тем такая надежность достигается ценой дополнительных задержек: обработка данных становится возможной только после приема всех сегментов в правильной последовательности. В системах, где требуется обмен данными в реальном времени, данное обстоятельство может служить существенным ограничением [3, 4].

Кроме того, TCP является потоковым протоколом и не задает формат передаваемых данных. Поэтому разработчику приходится самостоятельно реализовывать протокол прикладного уровня, который определял бы структуру сообщений, правила их интерпретации и способы сериализации. Это усложняет реализацию, но одновременно предоставляет гибкость при выборе формата обмена данными в соответствии с конкретной задачей.

После установления соединения TCP обеспечивает двусторонний обмен данными между клиентом и сервером. Упрощенная схема установления соединения и процесса передачи данных с использованием TCP приведена на рис. 2.

UDP (User Datagram Protocol) представляет собой транспортный протокол, функционирующий поверх IP и, в отличие от TCP, не обеспечивающий установление соединения, гарантии доставки, упорядоченности и повторной передачи данных. Передача осуществляется в виде независимых датаграмм, которые могут быть потеряны, дублированы или доставлены вне исходного порядка.

Отсутствие механизмов контроля надежности и упорядоченности снижает накладные расходы и задержки передачи, что делает UDP предпочтительным для приложений реального времени, включая системы потоковой передачи данных и

сетевые игры. Однако обеспечение целостности, порядка доставки и обработки потерь при необходимости должно реализовываться на уровне приложения, что увеличивает сложность разработки [5].

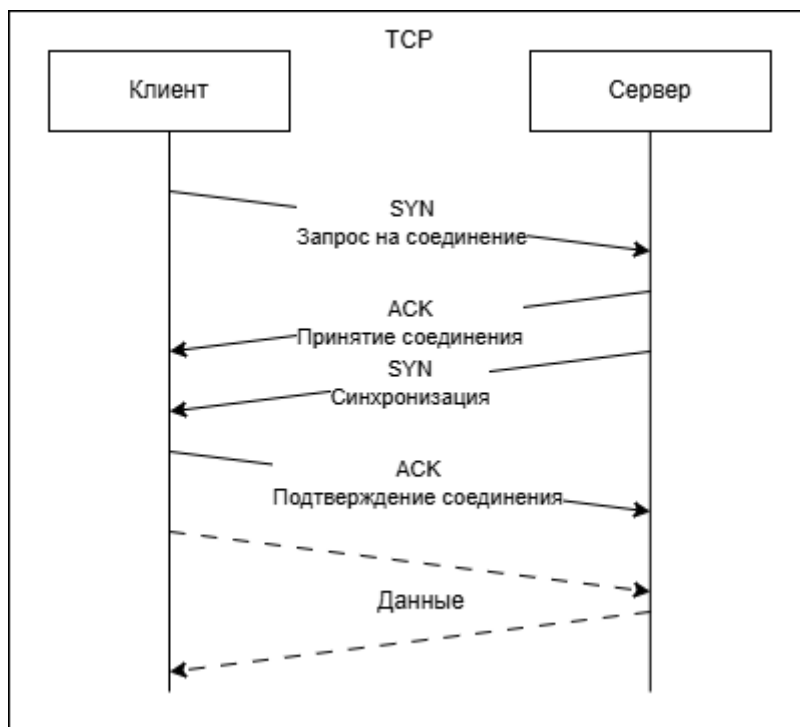


Рис. 2. Диаграмма установления соединения и передачи данных используя протокол TCP

UDP не накладывает ограничений на структуру передаваемых данных, что требует использования протокола прикладного уровня для их интеграции. Обобщенная схема передачи данных с использованием UDP представлена на рис. 3.

HTTP (Hypertext Transfer Protocol) представляет собой протокол прикладного уровня, широко используемый для взаимодействия компонентов распределенных систем по модели «клиент-сервер». Архитектурный стиль REST (Representational State Transfer) определяет принципы организации такого взаимодействия, включая ресурсно-ориентированный подход, унифицированный интерфейс и отсутствие сохранения состояния между запросами.

В рамках архитектуры REST обмен данными выполняется с применением текстовых форматов, например (JSON) JavaScript Object Notation или расширяемого языка разметки XML (Extensible Markup Language). Передача двоичных данных возможна как в непосредственном виде, в частности с использованием типа MIME «application/octet-stream», так и посредством их инкапсуляции в текст, например при помощи кодировки Base64. Протокол HTTP поддерживает механизмы кодирования содержимого, предназначенные для сжатия данных, к числу которых относятся алгоритмы deflate, gzip и brotli [6, 7].

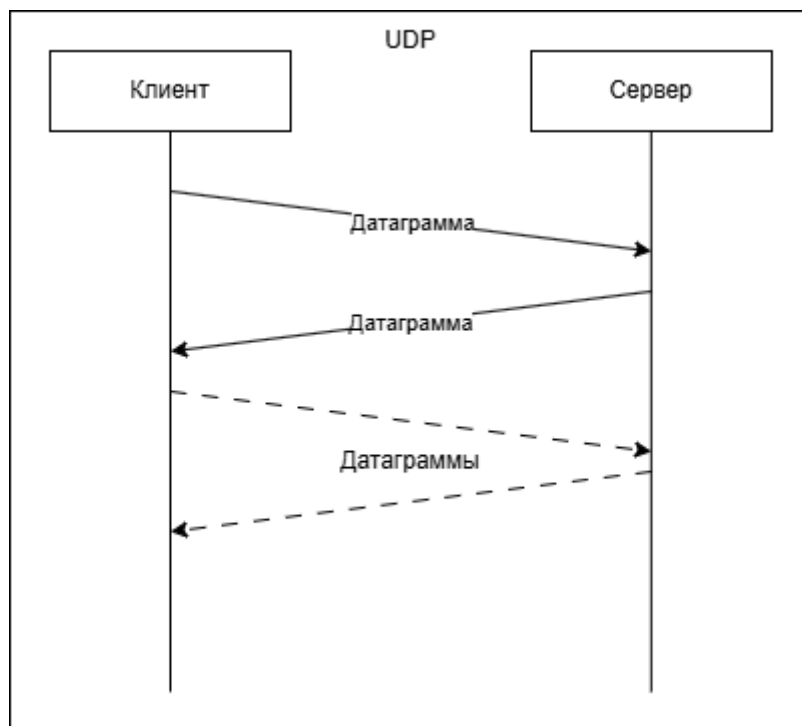


Рис. 3. Схема передачи данных с применением протокола UDP

Структура запросов и ответов в протоколе HTTP, включая методы, заголовки и коды состояния, задается явным образом, что заметно облегчает разработку как клиентской, так и серверной частей системы.

Для обеспечения защиты передаваемых данных применяется расширение HTTP, известное как HTTPS и базирующееся на протоколе TLS. Данное расширение выполняет шифрование канала связи и аутентификацию соединения. Использование готового решения позволяет не разрабатывать собственные криптографические механизмы, одновременно оставляя возможность дополнительной защиты на других уровнях при необходимости.

Существенным ограничением данного подхода в контексте рассматриваемой системы является необходимость наличия HTTP-сервера на стороне узла, предоставляющего данные. В случае двустороннего взаимодействия между координатором и агентами это может потребовать развертывания серверных компонентов на обеих сторонах. Наличие открытых сетевых портов и веб-сервисов на стороне агентов потенциально увеличивает поверхность атаки и предъявляет повышенные требования к обеспечению безопасности. Обобщенная схема выполнения HTTP/HTTPS-запроса в рамках REST-подхода представлена на рис. 4.

WebSocket представляет собой протокол прикладного уровня, обеспечивающий полно-дуплексный обмен данными между клиентом и сервером поверх одного установленного TCP-соединения. В отличие от HTTP, где взаимодействие строится по модели «запрос-ответ», WebSocket после первоначального HTTP-рукопожатия устанавливает постоянное соединение. Это позволяет обеим сторо-

нам обмениваться сообщениями независимо друг от друга и поддерживать двустороннюю связь в реальном времени [8].

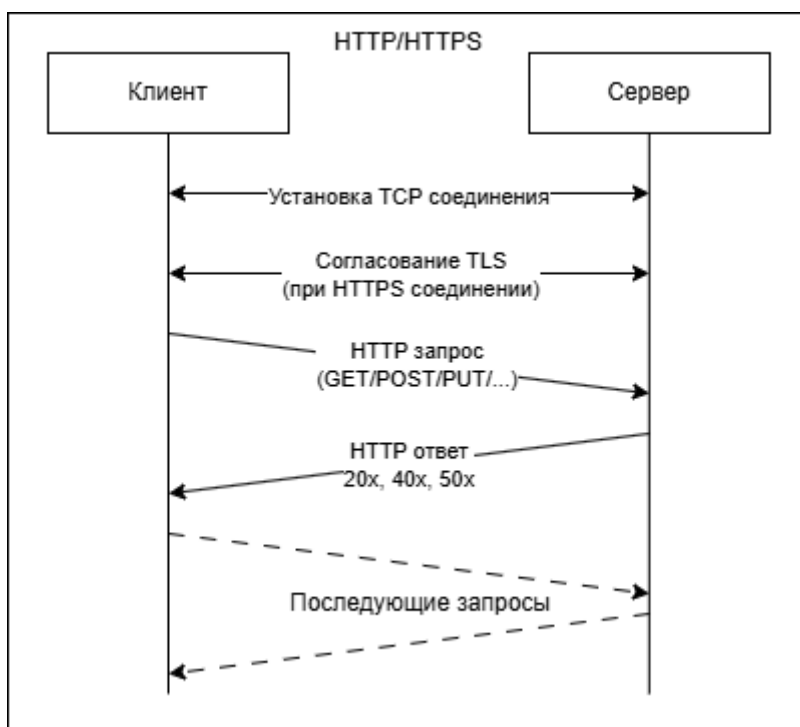


Рис. 4. Диаграмма выполнения HTTP-REST запроса

Протокол поддерживает передачу как текстовых, так и двоичных данных, благодаря чему можно использовать различные форматы сериализации, включая JSON, XML, и собственные бинарные протоколы. Это дает больше гибкости при проектировании системы обмена сообщениями и позволяет оптимизировать передачу данных с учетом требований к производительности и объему сетевого трафика.

Защищенная версия протокола – WSS- использует TLS-шифрование по аналогии с HTTPS, обеспечивая конфиденциальность и целостность передаваемых данных. За счет постоянного соединения и отсутствия необходимости заново устанавливать соединение для каждого сообщения, WebSocket обеспечивает низкие задержки передачи данных. Благодаря этому протокол хорошо подходит для систем, которым требуется взаимодействие, близкое к реальному времени [9].

WebSocket позволяет организовать эффективный двусторонний обмен данными без необходимости постоянного опроса сервера (polling). Однако для работы протокола требуется поддерживать активные соединения, что при большом количестве агентов может увеличивать нагрузку на сервер. Обобщенная схема обмена сообщениями с использованием WebSocket представлена на рис. 5.

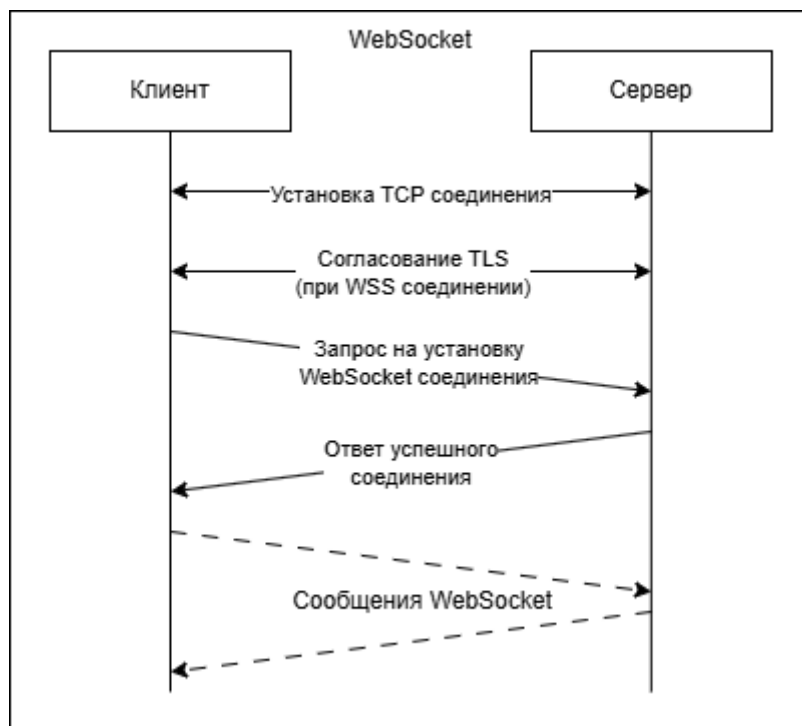


Рис. 5. Передача сообщений с использованием протокола WebSocket

Результаты сравнительного анализа

Сравнительный анализ протоколов TCP, UDP, HTTP (REST) и WebSocket позволяет выделить их ключевые различия по ряду характеристик, важных при проектировании распределенных систем: надежность доставки, задержки, модель взаимодействия, накладные расходы, сложность реализации и уровень безопасности.

Протокол TCP обеспечивает высокую надежность благодаря таким механизмам, как подтверждение доставки, контроль целостности и повторная передача данных. Это делает его пригодным для задач, в которых критически важны полнота и корректность информации. Вместе с тем указанная надежность сопровождается дополнительными задержками, возникающими как на этапе установления соединения, так и в процессе ожидания всех сегментов перед началом обработки данных. В отличие от TCP, протокол UDP снижает задержки за счет отсутствия механизмов гарантированной доставки, что делает его более эффективным для сценариев, работающих в реальном времени. Однако при использовании UDP логику обеспечения надежности приходится реализовывать на прикладном уровне. В обоих случаях разработчику зачастую требуется самостоятельно проектировать прикладные механизмы обмена сообщениями, что увеличивает сложность разработки.

Протокол HTTP в рамках REST-архитектуры, будучи протоколом прикладного уровня, предоставляет стандартизированный способ взаимодействия между компонентами системы. К его ключевым достоинствам относятся простота интеграции, широкая поддержка различными платформами и встроенные механизмы обеспечения безопасности. Тем не менее модель взаимодействия «запрос-ответ»,

а также необходимость установления соединения для каждого запроса приводят к росту накладных расходов и задержек. Данное обстоятельство ограничивает применение REST в сценариях, требующих интенсивного двустороннего обмена данными.

WebSocket сочетает в себе надежность TCP при передаче данных и возможность постоянного двустороннего обмена сообщениями. После установления соединения он обеспечивает низкие задержки и эффективную передачу данных без необходимости повторного создания соединения. Это делает его особенно подходящим для систем, где требуется обмен информацией в реальном времени. В то же время необходимость поддержания большого числа открытых соединений может приводить к дополнительной нагрузке на серверную сторону.

Сводное сравнение рассмотренных протоколов приведено в табл. 1.

Таблица 1. Сравнение функционала рассмотренных протоколов

Характеристика	TCP	UDP	HTTP (REST)	WebSocket
Уровень модели OSI	Транспортный	Транспортный	Прикладной	Прикладной
Надежность	Высокая	Отсутствует	Высокая (через TCP)	Высокая (через TCP)
Установление соединения	Требуется	Не требуется	Требуется при каждом запросе	Требуется однократно
Модель взаимодействия	Поток (Stream)	Датаграммы (Datagram)	Запрос-Ответ (Request-Response)	Сообщения (Messages)
Накладные расходы подключения	Средние	Минимальные	Высокие	Низкие после установления соединения
Сложность реализации передачи сообщений	Средняя	Высокая	Низкая	Низкая
Безопасность	Требуется дополнительных механизмов	Требуется дополнительных механизмов	HTTPS (TLS)	WSS (TLS)

Для получения практических данных о задержках было разработано приложение для тестирования, позволяющее оценить производительность каждого протокола по времени передачи одного пакета. В рамках тестирования использовались пакеты различных размеров – от 1 байта до 32768 байт (с шагом по степеням двойки). Данные передавались между компьютером и удаленным сервером в локальной сети.

Полученные данные о времени передачи данных приведено в табл. 2.

По получившимся данным был построен график зависимости размера пакета к длительности его передачи по локальной сети (рис. 6). Протокол HTTP не

был включен в данный график, так как задержки передачи данных при использовании данного протокола многократно превышали ожидаемые результаты.

Таблица 2. Время передачи данных по разным протоколам в локальной сети

Размер пакета (Б)	WebSocket (мс)	UDP (мс)	TCP (мс)	HTTP (мс)
1 (прогрев)	1,078	0,167	0,066	199,904
1	0,028	0,080	0,046	3,636
2	0,020	0,037	0,052	3,327
4	0,032	0,054	0,033	4,616
8	0,024	0,031	0,036	2,436
16	0,028	0,047	0,052	3,143
32	0,036	0,082	0,067	3,252
64	0,025	0,051	0,042	18,054
128	0,032	0,042	0,036	6,356
256	0,044	0,067	0,031	3,306
512	0,028	0,062	0,037	5,764
1024	0,064	0,074	0,038	3,686
2048	0,093	0,034	0,072	3,542
4096	0,120	0,086	0,037	3,540
8192	0,787	0,071	0,039	8,411
16384	0,516	0,076	0,075	3,800
32768	1,040	0,120	0,026	3,442
Общее время	3,995	1,181	0,785	280,215
Минимальное	0,020	0,031	0,026	2,436
Максимальное	1,078	0,167	0,075	199,904
Среднее	0,235	0,069	0,046	16,483

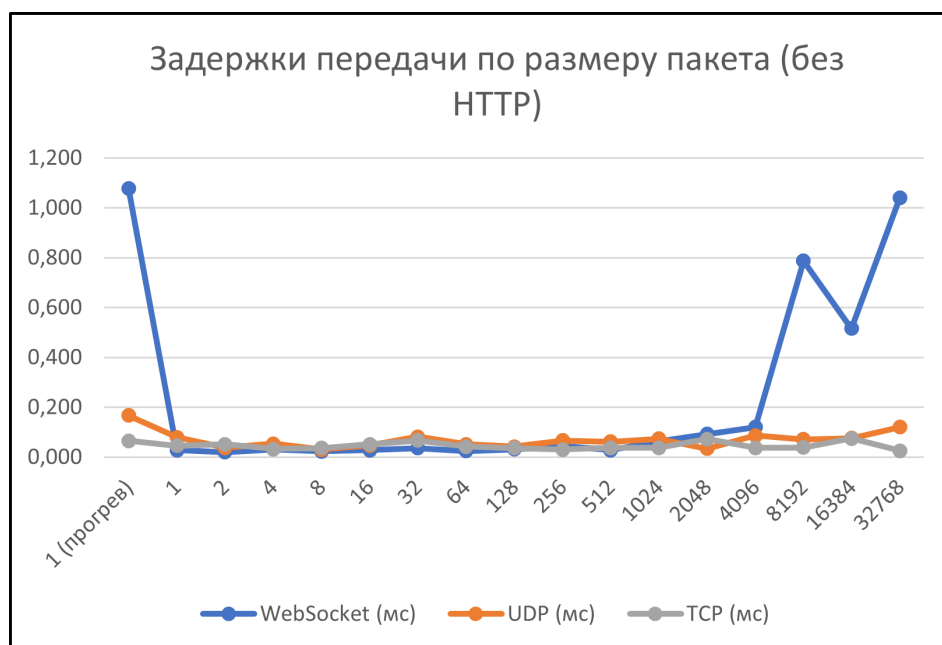


Рис. 6. График отношения размера пакета к длительности передачи пакета

По данным графика, производительность WebSocket сопоставима с TCP и UDP, за исключением пакетов крупнее 4 КБ – их передача занимает заметно больше времени, но все равно в разы быстрее, чем при использовании HTTP.

По результатам анализа можно сделать вывод, что выбор протокола сетевого взаимодействия должен определяться требованиями конкретной системы. Для задач, где критична надежность передачи, целесообразно использовать TCP или протоколы прикладного уровня, основанные на нем (HTTP, WebSocket). В сценариях, требующих минимальных задержек и допускающих потери данных, предпочтение следует оставлять UDP. Для построения масштабируемых веб-сервисов, оптимальным является HTTP (REST), в то время как для систем с интенсивным двусторонним обменом в реальном времени, наиболее подходящим и эффективным решением является WebSocket.

Заключение

В ходе исследования были рассмотрены основные подходы к организации обмена сообщениями в распределенных системах мониторинга между координатором и агентами, включая транспортные протоколы TCP и UDP, прикладной протокол HTTP с архитектурным стилем REST, а также протокол WebSocket. Проведенный сравнительный анализ показал, что рассмотренные решения существенно различаются по таким характеристикам, как задержка передачи данных, надежность доставки, требования к вычислительным ресурсам и сложность реализации. В рамках выбранного подхода протокол WebSocket обеспечивает эффективную организацию двустороннего обмена данными благодаря поддержанию постоянного соединения и снижению накладных расходов, связанных с повторным установлением сеансов связи. Указанные свойства делают его пригодным для передачи событий мониторинга и управляющих сообщений в режиме, близком к реальному времени.

Дополнительно в архитектуре системы обосновано использование протокола HTTP в одностороннем режиме передачи данных от агентов к координатору для вспомогательных задач, к числу которых относится периодическая отправка отчетной информации. Такой подход позволяет снизить нагрузку на постоянное WebSocket-соединение и разделить каналы передачи данных в соответствии с их назначением. Следовательно, совместное применение протоколов WebSocket и HTTP позволяет достичь баланса между производительностью, масштабируемостью и сложностью разработки. Дополнительным преимуществом выступает повышение гибкости архитектуры, что особенно важно для распределенных систем мониторинга.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Таненбаум Э., Уэзеролл Д. Компьютерные сети. - 5-е изд. - СПб.: Питер, 2012. - 960 с.
2. Куроуз Дж., Росс К. Компьютерные сети: нисходящий подход. - 6-е изд. - М.: Вильямс, 2016. - 912 с.
3. Стивенс У. Р. TCP/IP Illustrated. Том 1. Протоколы. - М.: Вильямс, 2003. - 576 с.
4. Постел Дж. Transmission Control Protocol (TCP) (RFC 793). - IETF, 1981. - URL: <https://datatracker.ietf.org/doc/html/rfc793> (дата обращения: 09.04.2026).

5. Постел Дж. User Datagram Protocol (UDP) (RFC 768). - IETF, 1980. - URL: <https://data-tracker.ietf.org/doc/html/rfc768> (дата обращения: 09.04.2026).
6. Филдинг Р. Т., и др. Hypertext Transfer Protocol (HTTP/1.1: Semantics and Content) (RFC 9110). - IETF, 2022. - URL: <https://datatracker.ietf.org/doc/html/rfc9110> (дата обращения: 09.04.2026).
7. Ричардсон Л., Руби С. RESTful Web Services. - Sebastopol: O'Reilly Media, 2007. – 448 с.
8. Бугаков А. WebSocket: как это работает и зачем он нужен. - Хабр, 2018. - URL: <https://habr.com/ru/articles/346306/> (дата обращения: 09.04.2026).
9. Фетт И., Мельников А. The WebSocket Protocol (RFC 6455). - IETF, 2011. - URL: <https://datatracker.ietf.org/doc/html/rfc6455> (дата обращения: 09.04.2026).

© *Р. В. Морозов, П. Ю. Бугаков, 2026*