ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/IRJ.2025.159.70

СЕТЕВЫЕ ПРОТОКОЛЫ ТРАНСПОРТНОГО УРОВНЯ ДЛЯ РЕАЛИЗАЦИИ ВЗАИМОДЕЙСТВИЯ В МНОГОПОЛЬЗОВАТЕЛЬСКИХ ИГРАХ

Научная статья

Галай И.П.[1, *]

[1] Wargaming Group Limited, Никосия, Кипр

* Корреспондирующий автор (igor.galaj[at]gmail.com)

Аннотация

В работе представлен комплексный анализ транспортных протоколов, применяемых в многопользовательских играх реального времени. На основе требований к минимизации задержки, обеспечению согласованности данных и защите трафика сопоставлены классические решения — TCP и UDP — с современными расширениями QUIC, SCTP и DTLS. Рассмотрены три модели распределения полномочий в клиент-серверной архитектуре (Server / Client / Hybrid Authority) и их влияние на выбор транспорта. Проведён сравнительный обзор метрик (RTT, head-of-line blocking, overhead, стойкость к DDoS) и практик компенсирования потерь UDP: дельта-репликации, выборочных подтверждений и клиентского предсказания. Выделены актуальные угрозы (перехват пакетов, читерство, DDoS) и соответствующие механизмы защиты вплоть до сквозного шифрования QUIC. Новизна исследования заключается в интегрированной Decision-Matrix, которая увязывает жанр игры, целевой RTT-порог и уровень критичности данных с оптимальным протоколом, а также в предложенном алгоритме выбора, проверенном на индустриальных кейсах. Практическая значимость работы заключается в предоставлении разработчикам пошагового руководства по проектированию сетевой подсистемы, позволяющего одновременно снижать латентность и повышать устойчивость к сетевым угрозам.

Ключевые слова: многопользовательские игры, транспортный протокол, TCP, UDP, QUIC, задержка, безопасность, Decision-Matrix, дельта-репликация, DDoS.

TRANSPORT LAYER NETWORK PROTOCOLS FOR IMPLEMENTING INTERACTION IN MULTIPLAYER GAMES

Research article

Galai I.P.[1, *]

[1] Wargaming Group Limited, Nicosia, Cyprus

* Corresponding author (igor.galaj[at]gmail.com)

Abstract

The work presents a complex analysis of transport protocols used in real-time multiplayer games. Based on the requirements for minimising delay, ensuring data consistency, and protecting traffic, classic solutions — TCP and UDP — are compared with modern extensions QUIC, SCTP, and DTLS. Three models of authority distribution in client-server architecture (Server / Client / Hybrid Authority) and their impact on transport selection are examined. A comparative review of metrics (RTT, head-of-line blocking, overhead, DDoS resistance) and practices for compensating for UDP losses is conducted: delta replication, selective acknowledgements, and client prediction. Current threats (packet interception, cheating, DDoS) and corresponding protection mechanisms, up to end-to-end QUIC encryption, are highlighted. The novelty of the research lies in the integrated Decision Matrix, which links the game genre, target RTT threshold, and data criticality level with the optimal protocol, as well as in the suggested selection algorithm, which has been tested on industrial cases. The practical significance of the work lies in providing developers with a step-by-step guide to designing a network subsystem that simultaneously reduces lag and increases resistance to network threats.

Keywords: multiplayer games, transport protocol, TCP, UDP, QUIC, delay, security, Decision-Matrix, delta replication, DDoS.

Введение

Многопользовательские игры реального времени — MMO, MOBA, соревновательные шутеры и др. — уже более десяти лет остаются одним из самых быстро-растущих сегментов индустрии. Их критическое требование — минимальная сетевая задержка: рост round-trip-time (RTT) даже на десятки миллисекунд резко снижает удовлетворённость пользователей и точность игрового взаимодействия [1]. Глобальный характер таких игр усугубляет проблему: объективные ограничения распространения сигнала повышают RTT между континентами до 100–150 мс (Европа ↔ США) и > 200 мс (Европа ↔ Австралия) [2]. Потому ведущие студии разворачивают георазпределённые кластеры; показательно, что добавление 350 edge-узлов сократило средний ping одной популярной игры с 74 до 29 мс [3]. Однако даже при оптимальной топологии окончательный баланс «задержка ↔ надёжность» определяется выбранным транспортным протоколом.

На практике компромисс сводится к двум классическим вариантам: TCP с гарантированной доставкой и эффектом *head-of-line blocking* и UDP без встроенной надёжности, но с минимальной латентностью. В динамичных жанрах UDP доминирует, а разработчики реализуют поверх него собственные механизмы подтверждений, дельта-репликацию и клиентское предсказание, достигая приемлемой *eventual consistency*. Пошаговые и квази-реальные режимы чаще

полагаются на TCP. Параллельно растут требования к безопасности: отсутствие шифрования исторически делало игровые сервисы мишенью перехватов и DDoS-атак — до 50% мировых DDoS приходится на игры [4], [5], [6]. Современные решения вроде QUIC (сквозное шифрование + 0/1-RTT) и DTLS демонстрируют, что высокая скорость и защита трафика совместимы [7].

Цель статьи — дать комплексный анализ транспортных протоколов для многопользовательских игр реального времени и предложить практическую методику их выбора с учётом жанра, технических ограничений и угроз безопасности.

Для достижения цели решаются задачи:

1. Обобщить требования игр к задержке, консистентности и пропускной способности.
2. Сравнить TCP, UDP и современные расширения (QUIC, SCTP, DTLS) по метрикам задержки, надёжности, overhead'а и защищённости.
3. Проанализировать типовые угрозы (читерство, перехват, DDoS) и соответствующие механизмы защиты.
4. Систематизировать практики компенсирования потерь UDP (дельта-репликация, Selective ACK, предсказание клиента).
5. Сформулировать алгоритм выбора протокола, валидированный кейсами из индустрии.

Новизна статьи заключается в объединении в одной модели три плоскости выбора — *задержка / надёжность / безопасность* — и предлагает Decision-Matrix для быстрого сопоставления жанра игры с оптимальным транспортом. Опираясь на актуальные исследования, работа показывает, как новые протоколы (QUIC, DTLS) позволяют одновременно снизить латентность и закрыть требования безопасности, ранее считавшиеся взаимоисключающими.

Архитектура многопользовательской игры и сетевые требования

Онлайн-игры реального времени практически всегда опираются на клиент–серверную архитектуру, которая обеспечивает централизованный контроль и масштабируемость. Альтернативные схемы (например, peer-to-peer) в коммерческих проектах используются редко по двум причинам: сложности с поддержанием единого консистентного состояния между множеством равноправных узлов и уязвимость к читерству (отсутствие доверенного центрального узла позволяет злоумышленникам легче подделывать данные) [8]. В модели клиент–сервер все игровые клиенты подключаются к одному удаленному серверу (или кластеру серверов), который хранит текущее состояние виртуального мира и авторитарно обрабатывает действия игроков. Прямые соединения *клиент–клиент* отсутствуют: взаимодействие идет только через сервер, что упрощает проверку корректности и соблюдение правил игры на центральной стороне. На рисунке 1 представлены основные сетевые архитектуры игр.

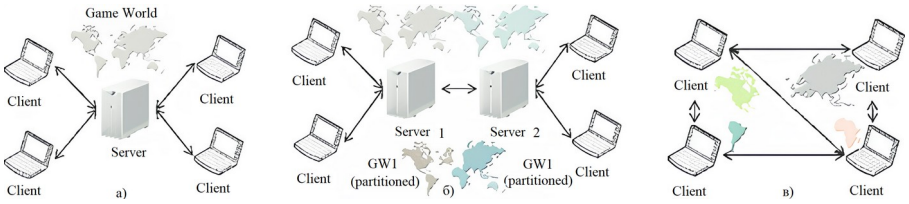

Рисунок 1 - Сетевые архитектуры игр:
DOI: https://doi.org/10.60797/IRJ.2025.159.70.1

Примечание: а – классическая модель «клиент–сервер» с единственным сервером, поддерживающим целый игровой мир; б – мультисерверная модель, при которой мир разделен на зоны или шардовые серверы (Server 1, Server 2), обслуживающие разных игроков и взаимодействующие между собой; в – peer-to-peer модель, в которой все узлы равноправны и каждый клиент обменивается данными непосредственно с другими (в современных играх почти не используется из-за проблем безопасности и консистентности)

В рамках клиент-серверной архитектуры различают варианты разграничения полномочий между сервером и клиентами. Чаще всего выделяют три модели управления игровой логикой:

- Server Authority. Клиенты отправляют только ввод (нажатия, вектор движения); вся симуляция выполняется на сервере, а клиент лишь визуализирует присланное состояние. Достоинство — максимальная защита от читерства (пользователь не может изменить логику). Недостаток — дополнительная задержка: каждое действие отображается лишь после подтверждения сервера, что заметно в высокодинамичных жанрах.
- Client Authority. Основная логика (перемещение, коллизии) считается локально, сервер периодически принимает агрегированное состояние и рассылает его другим игрокам. Это почти устраняет задержку и разгружает сервер, но открывает путь для мошенничества: модифицированный клиент может, например, «телепортироваться» или стрелять сквозь препятствия, а серверу сложно обнаружить подмену без полного пересчёта.
- Гибридная модель. Критически важные расчёты (попадания, урон, экономика) выполняет сервер, а клиент интерполирует движение и обрабатывает второстепенные эффекты. Такой компромисс снижает воспринимаемую задержку-ввода, сохраняя контроль сервера над ключевыми параметрами и существенно ограничивая возможности читерства. Поэтому гибрид сегодня наиболее распространён.

Помимо организации логики, архитектура должна удовлетворять нефункциональным требованиям сетевой игры. Главные из них: низкая задержка, консистентность данных, устойчивость к потерям и эффективность использования

сети. Низкая задержка (ping) критична для чувствительности управления: в большинстве экшен-игр превышение задержки ~100 мс вызывает заметные лаги [3].

Консистентность означает, что все игроки должны видеть более-менее единое состояние мира — допустимы небольшие расхождения (ведь идеальная синхронность недостижима из-за сетевых задержек), но ключевые события (попадания, результаты взаимодействий) должны приходить к каждому участнику достоверно. Устойчивость к потерям предполагает, что игра должна продолжать корректно работать даже если периодически теряются пакеты: например, кратковременное отсутствие обновлений о положении противника не должно приводить к фатальным сбоям (в таких случаях клиенты применяют предсказание движения или отображают последнее известное состояние). Эффективность трафика важна, поскольку у игроков разная пропускная способность сети. Разработчики стремятся минимизировать объем пересылаемых данных — например, посылая дельты изменений (разницу между предыдущим и новым состоянием), а не полное состояние объекта. Также распространены техники сглаживания сетевого трафика (чтобы посылать пакеты через равные интервалы, избегая «спайков» нагрузки) и компрессии данных.

Отдельно стоит учесть географический фактор. В идеале, чтобы сократить задержки, игроки должны подключаться к ближним региональным серверам. Так и поступают крупные игры, создавая отдельные серверные площадки для Европы, Северной Америки, Азии и т.д. Однако в некоторых случаях (например, глобальные MMO) игроки из разных регионов взаимодействуют в общем мире. При этом неизбежны задержки между отдаленными пользователями.

Одно из решений — мультисерверная архитектура (рис. 1b), в которой мир разделен на зоны, и каждый сервер отвечает за свою зону, взаимодействуя с другими. Пользователь автоматически переключается на ближайший сервер той зоны, где находится его персонаж, что локализует большую часть трафика и снижает средний ping [8]. Тем не менее границы зон вносят дополнительную сложность (переход игрока из зоны в зону, баланс нагрузки между серверами).

Другая технология — использование сетей доставки контента и edge-серверов, когда копии игровой службы запускаются динамически ближе к всплескам пользовательской активности [3]. Обратной стороной является удорожание инфраструктуры. Таким образом, топология серверов и сетевые протоколы должны проектироваться совместно, чтобы обеспечить приемлемый баланс между издержками, задержкой и консистентностью.

Наконец, архитектура игры должна предусмотреть масштабирование. В пиковые моменты сервер должен обрабатывать тысячи одновременных подключений, не допуская деградации ответа. Для этого применяются *load balancing*, разделение игроков на инстансы/шарды, оптимизация сетевых обновлений (например, отсылать обновления не всем игрокам, а только тем, кто находится в зоне видимости события) [1]. Многие протоколы предусматривают механизмы групповой адресации (multicast, broadcast) или сжатия обновлений для группы, чтобы снизить нагрузку на сервер. В итоге на этапе архитектурного проектирования формируются требования к транспортному протоколу: насколько быстро устанавливается соединение, какую обеспечивает пропускную способность и задержку, есть ли встроенные механизмы масштабирования (multicast), как реагирует на потери, поддерживает ли шифрование и пр. Далее мы рассмотрим, как классические и новые транспортные протоколы удовлетворяют этим требованиям и какие возникают компромиссы при их использовании в играх.

Транспортные протоколы в реальном времени

3.1. TCP и UDP

Протокол TCP изначально разрабатывался для надежной потоковой передачи данных в сетях с непредсказуемым поведением. Его сильные стороны — гарантированная доставка и строгий порядок байтового потока. Для игр, требующих обмена критически важными сообщениями (например, ход в шахматах, результат матча, транзакции в игровой экономике), эти свойства весьма ценны [1]. В пошаговых жанрах или текстовых MMO небольшое увеличение задержки из-за установления TCP-сессии и ожидания подтверждений не играет большой роли, зато упрощается разработка (можно полагаться, что все данные дойдут и в том же порядке, что отправлены). В частности, пошаговые стратегии и настольные игры почти всегда работают по TCP или поверх него (например, используя WebSockets) — поскольку реакция игрока измеряется секундами, а потеря пакета недопустима. *Янкевич* косвенно подтверждает это, отмечая, что в системах реального времени, где важна надежность управления, допустимо пожертвовать долями секунды на дополнительные обмены для гарантии успешного исхода команды [7].

Однако в динамичных играх (шутеры, экшен) применение TCP может серьезно ухудшить качество геймплея. Пока потерянный сегмент не будет повторно доставлен, получатель не увидит и все последующие данные (эффект head-of-line blocking). Например, если в сетевом шутере потерялся пакет с позициями игроков, TCP приостанавливает передачу следующего обновления до получения этого пакета — в результате на экране может возникнуть скачок (телепортация персонажа) или ощутимая задержка в управлении. Как справедливо указывают многие исследователи, TCP «слишком медлителен» для массовых онлайновых игр с множеством одновременных действий [1].

Более того, TCP включает алгоритмы Congestion Control и тщательное управление потоком, рассчитанные на максимальную пропускную способность, а в играх объем данных обычно невелик и важнее регулярность обновлений, чем полное заполнение канала [2]. В итоге при перегрузке сети TCP соединение будет резко тормозить (уменьшать окно, переходить в режим медленного старта), тогда как для игровой логики лучше иногда пропустить обновление, но продолжить слать актуальные данные без пауз.

Протокол UDP, напротив, предоставляет лишь минимальный сервис доставки датаграмм без гарантий. Его главное достоинство — минимально возможные задержки. UDP-пакеты отправляются сразу, без установки соединения и без ожидания подтверждений. Если сеть позволяет доставить пакет, клиент получит его с наименьшей задержкой поверх IP (в идеале близкой к физическому round-trip времени) [1].

В случае потери пакета UDP никак не уведомляет отправителя – последний может лишь выяснить потерю косвенно (например, по таймауту ожидания ответа). Для интерактивных игр такой «ненадежный» подход зачастую

оправдан, как показывают эмпирические исследования, потеря отдельного обновления состояния (например, одного кадра позиции персонажа) почти не влияет на восприятие, если следом приходит более свежее обновление [2]. Гораздо критичнее для игрока — задержка или джиттер. Поэтому большинство быстрых игр (первые лица, королевские битвы, спортсимы) используют UDP для передачи позиций, событий, выстрелов и пр.

Компенсация отсутствия надежности возлагается на игровой движок: применяется модель «*последовательная согласованность*», при которой старые потерянные пакеты не пересылаются повторно, а их содержимое вычисляется из более новых данных. Например, если потерян пакет с координатами игрока в момент времени *t*, но затем пришел пакет на *t+50 мс*, то игра просто интерполирует траекторию и не требует именно тот потерянный пакет — состояние «догоняется» более актуальной информацией. Такой подход называют *supersedable packets* (замещаемые пакеты) [2]. Разработчики сетевого кода помечают типы сообщений, которые можно не подтверждать и не ретранслировать. Как правило, к ним относятся пакеты состояния, позиции, ориентации, которые обновляются постоянно и новейшее значение делает старое неактуальным.

Конечно, в любой игре есть и критичные события, терять которые нельзя (например, нажатие кнопки «выстрел», результат попадания, получение редкого предмета). Эти события образуют другой класс — *causal packets*, влияющие на дальнейший ход игры и требующие надежной доставки [2]. Для них даже в UDP-протоколе обычно вводятся механизмы подтверждений и повторов. Проще всего — дублировать такие пакеты несколько раз или запрашивать у получателя явное подтверждение важного события.

В более сложных случаях разрабатываются гибридные протоколы, сочетающие UDP для основных данных и отдельный надежный канал для критических (об этом ниже). Но даже при добавлении надежности «поверх UDP» важно, что потеря одного важного сообщения не тормозит всю коммуникацию: обычные пакеты состояния могут дальше свободно передаваться. Таким образом, достигается параллелизм потоков: критичные данные идут по одному логическому каналу с подтверждением, некритичные — по другому без подтверждений. В результате общая отзывчивость системы выше, чем при использовании одного монолитного TCP-потока, где все данные связаны единым порядком.

Следует упомянуть еще один нюанс: влияние конкуренции трафика TCP и UDP. Если игра пытается одновременно использовать оба протокола (например, TCP для чата, а UDP для позиций), в условиях перегрузки сети TCP-соединение благодаря встроенному контролю «выдавит» UDP-пакеты — маршрутизаторы при заполнении буферов склонны отбрасывать непотверждаемый UDP трафик раньше, чем TCP, который реагирует на потери снижением интенсивности. В итоге параллельное использование TCP+UDP может привести к тому, что при активном обмене по TCP (например, загрузка данных или обновление) все UDP-обновления начнут теряться, разрушая смысл такой гибридной схемы. Чтобы этого избежать, требуется сложно балансировать долю пропускной способности между протоколами, что на практике редко реализуется идеально [2]. По этой причине ряд движков (включая Unreal Engine, Unity) предпочитают вообще отказаться от TCP для игровых данных, оставив его лишь для служебных целей (подгрузка контента, социальные функции), а весь игровой трафик вести по UDP с собственным *reliability layer*.

Существует и противоположный подход — поверх одного TCP-соединения пускать несколько логических потоков (каналов) игровых данных. Однако стандартный TCP не поддерживает множественных независимых потоков *из коробки*: разработчики вынуждены либо открывать несколько TCP-соединений (что накладно), либо использовать протоколы-мультиплексоры (например, WebSocket + HTTP/2, где внутри одного TCP идут параллельные запросы). Подобные решения, по сути, реализуют на прикладном уровне то, что транспорт должен был бы делать сам (разделять трафик разного типа). Эта потребность и вдохновила создание новых транспортных протоколов, о которых речь пойдет далее.

Чтобы обобщить свойства основных протоколов, приведем сравнительную таблицу [9]:

Таблица 1 - Характеристики транспортных протоколов в контексте игр

DOI: https://doi.org/10.60797/IRJ.2025.159.70.2

Протокол	Состояние соединения	Доставка и порядок	Задержка (RTT)	Встроенная безопасность	Применение в играх
TCP	С установлением (3-way handshake)	Надёжная, строго упорядоченная (байтовый поток)	Средняя — +1–2 RTT на установление; возможен *head-of-line blocking*	Поддерживает TLS (шифрование поверх TCP)	Пошаговые игры, чаты, транзакции. Гарантирует консистентность, но не подходит для частых обновлений позиций из-за лагов.
UDP	Без установления (датаграммы)	Ненадёжная, без гарантии порядка (можно реализовать свой порядок)	Минимальная — нет доп. RTT; потери не блокируют последующие пакеты	Нет (можно применять DTLS отдельно)	Экшен, шутеры, MOBA. Обеспечивает мгновенную отправку обновлений; требует реализовать логику повторов и защиты от потерь на уровне игры.
SCTP	С установлением; многопотоковый (multi-stream)	Настраиваемая надежность: может быть надежным или частично надежным; порядок обеспечивается в границах каждой потока	Низкая —средняя — установление как у TCP, но параллельные потоки устраняют HOL-блокировку	Нет встроенного шифрования (обычно используют DTLS поверх)	Перспективен для игр: позволяет несколько потоков (например, отдельный для позиций, отдельный для событий) внутри одного соединения. Однако поддержка ограничена, в реальных играх почти не используется напрямую.
QUIC	С установлением на основе UDP (1-RTT handshake, возможен 0-RTT при повторном подключении)	Надёжная, многопотоковая (несколько параллельных потоков с независимым порядком); + имеется	Низкая — ускоренное установление (TLS 1.3 внутри); нет HOL-блокировки между потоками, но в пределах	Да, шифрование по умолчанию (TLS 1.3 встроен в протокол)	Новейший протокол, уже внедрён в веб (HTTP/3). Для игр интересен тем, что сочетает преимущества UDP (низкая задержка,

Протокол	Состояние соединения	Доставка и порядок	Задержка (RTT)	Встроенная безопасность	Применение в играх
		необязательный ненадёжный режим (datagram)	одного потока соблюдается порядок		мультиплексирование) и TCP (надежность). Может использоваться для шифрованных игровых соединений с быстрым стартом сессии. Пока мало используется в игровой индустрии, но рассматривается для облачных игр и новых проектов.

3.2. Новые протоколы, безопасность и выбор оптимального решения

В последние годы появились расширения и новые протоколы, пытающиеся устранить ограничения классических TCP/UDP. Один из самых заметных — QUIC, разработанный Google и стандартизированный IETF (RFC 9000). QUIC работает поверх UDP, но обеспечивает надёжный поток байтов, аналогичный TCP, и шифрование трафика аналогично TLS, объединяя их в единый протокол.

Главные преимущества QUIC для игр, это ускоренное установление соединения и отсутствие межпоточной блокировки. В QUIC клиент и сервер обмениваются данными шифрования в ходе самого первого обмена пакетами (*Initial*, *Hello* и т.д.), что экономит 1–2 лишних RTT по сравнению с классической связкой TCP+TLS (рис. 2) [9].

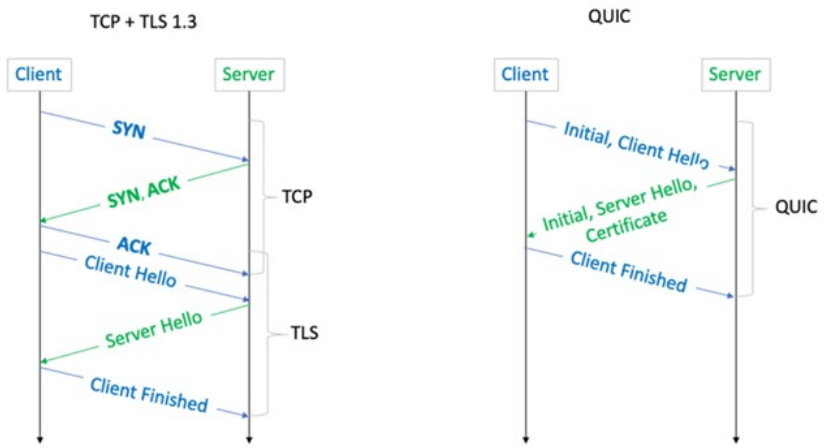

Рисунок 2 - Сравнение установления соединения:
слева – последовательность TCP + TLS 1.3 (клиент и сервер обмениваются 3 пакетами TCP и затем сообщениями TLS для согласования шифрования); *справа* – последовательность QUIC, совмещающая обмен ключами TLS и соединение в одном этапе, за счёт этого QUIC сокращает время начала передачи данных примерно на 1 RTT
DOI: https://doi.org/10.60797/IRJ.2025.159.70.3

Кроме того, QUIC мультиплексирует произвольное число логических потоков внутри соединения — каждый поток передает упорядоченный стрим данных, но потери на одном потоке не тормозят другие. Это решает проблему head-of-line blocking, которая мешала использовать один TCP-сокет для разнородных данных. Для игр это значит, что можно, к примеру, выделить отдельные потоки для позиционных обновлений, для важных событий, для чата — и задержка в одном не повлияет на остальные. Более того, спецификация QUIC недавно получила расширение *Datagram* (RFC 9221), позволяющее посылать ненадёжные датаграммы в рамках QUIC-соединения (без установки отдельного UDP-сокета) [9].

Такие датаграммы не повторяются при потере, аналогично UDP, но шифруются и интегрируются в общий поток. Потерянные датаграммы QUIC просто не учитываются, и приложение может получить уведомление о их возможной потере, чтобы при необходимости среагировать [9]. По сути, QUIC предоставляет «лучшее из обоих миров»: надёжные потоки для критичных данных и ненадёжные для остального — всё это под единой защищенной сессией.

Первые исследования производительности QUIC для игровых сценариев показывают перспективность. Например, Eghbal и Lu продемонстрировали, что использование частичного порядка доставки поверх QUIC (то есть отказ от строгой последовательности для некоторых типов обновлений) позволяет снизить задержку экранных обновлений на 30% по сравнению с TCP [10].

Особенно полезным это оказалось для рабочих нагрузок с независимыми областями данных — например, обновления разных частей игрового поля, которые не требуют строгой синхронизации между собой. Это прямой аналог ситуации в игровых движках, когда состояние мира разбивается на компоненты, и последовательность получения обновлений по одной компоненте не обязательно должна блокировать другую. QUIC с его мультиплексированием и возможностью настроить приоритеты потоков идеально подходит под такую модель [9]: можно передавать, к примеру, позиционные данные с высоким приоритетом по одному потоку, а крупные необязательные данные (звук, второстепенные события) — по другому с низким приоритетом, не опасаясь задержек первого.

Более того, все данные шифруются «по умолчанию» — это устраняет возможность читеров легко перехватывать или подделывать трафик. Стоит отметить, что QUIC шифрует не только полезную нагрузку, но и заголовки, скрывая от посредников управляющую информацию. Это мешает злоумышленникам делать некоторые виды атак (например, *packet injection* в середину потока) и затрудняет проведение DPI-атак на протокол.

С другой стороны, обязательное шифрование приводит к определенному overhead (как вычислительному, так и трафиковому из-за дополнительных байт). Поэтому интеграция QUIC в игры — вопрос баланса: в киберспортивных проектах, где ценны каждые миллисекунды, разработчики могут решить, что выигрыши QUIC в handshake недостаточны для перехода с хорошо отлаженного UDP-кода. В более казуальных или мобильных играх, напротив,

QUIC привлекателен упрощением разработки (безопасность «из коробки») и лучшей работой в сетях с высокими RTT (мобильный интернет), где ускорение старта сессии и встроенная коррекция потерь дают преимущество [11].

Другой интересный протокол — SCTP (Stream Control Transmission Protocol). Разработанный еще в 2000-х для телеком-приложений, SCTP обеспечивает мультиплексирование нескольких независимых потоков сообщений внутри одного соединения. Фактически, SCTP был предтечей многих идей QUIC: у него есть понятие *streams*, каждый из которых может быть надежным или частично надежным, с отдельной нумерацией пакетов.

Потеря в одном потоке SCTP не блокирует остальные, что устраняет HOL-блокировку на транспортном уровне. Казалось бы, идеальный вариант для игровых приложений. Почему же SCTP не получил распространения? Главная причина — плохая поддержка в сетевой инфраструктуре и ОС. Многие маршрутизаторы и NAT-ы «не знакомы» с SCTP и могут блокировать его трафик (в отличие от UDP/TCP).

Кроме того, в браузерах SCTP доступен лишь косвенно (через WebRTC Data Channels), а собственные API в популярных игровых движках отсутствуют. Разработчики привыкли работать с UDP и TCP, и переход на SCTP требовал бы глубоких изменений. Тем не менее WebRTC использует SCTP поверх UDP для передачи произвольных данных между браузерами (в сочетании с DTLS для шифрования) [12].

Это означает, что веб-игры уже сейчас косвенно эксплуатируют SCTP, получая выгоду от его многопоточности. Например, при организации p2p-соединения между клиентами через WebRTC можно передавать по отдельным *data channel* разные типы игровой информации (что, по сути, идут по разным потокам SCTP). В десктопных же играх пока нет известных случаев нативного использования SCTP — видимо, индустрия решила перескочить сразу на QUIC, как более современную альтернативу.

Отдельно следует коснуться вопроса шифрования и аутентификации трафика. Как отмечалось, внедрение TLS/DTLS в игровые протоколы шло медленно. Но ситуация меняется: рост кибератак на игры и ужесточение требований к приватности данных побуждают разработчиков интегрировать криптографическую защиту. Существует два уровня защиты:

- транспортный (например, DTLS поверх UDP или прямое использование QUIC);
- прикладной (встроенное шифрование полезных данных движком).

Транспортный удобнее, так как прозрачно шифрует весь поток; однако он также добавляет накладные расходы. Например, полное шифрование трафика лишает возможности использовать middlebox-оптимизации (проксирование, сжатие по схеме «сервер–сервер») и затрудняет диагностику сетевых проблем.

Поэтому ряд крупных MMO до сих пор передают несекретные данные (скажем, обновления позиций) в открытом виде, полагаясь лишь на собственную логику проверки достоверности (например, сервер все равно перепроверяет, не перемещается ли персонаж слишком быстро — и даже если пакет подделан, это не даст преимуществ читеру, кроме как вызвать десинхронизацию) [4].

С другой стороны, финансовые транзакции, учетные данные, чат все чаще шифруются стандартными методами. В современных игровых движках (Unity, Unreal) при использовании их сетевых API есть опция включения шифрования — фактически, это реализуется через TLS. Протокол QUIC, будучи по сути слиянием транспортного и криптоуровня, делает шифрование неотъемлемой частью соединения. Это может повысить доверие пользователей и соответствие требованиям (например, GDPR в части защиты пользовательских данных), но разработчикам надо учитывать рост трафика (шифрование обычно увеличивает объем передаваемых данных на ~5–10% за счет заголовков и выравнивания).

Помимо шифрования, важны механизмы защиты от DDoS. Для транспортных протоколов критичны методы предотвращения IP-спуфинга и усиления атак. Так, для UDP-игр желательно реализовать handshake на прикладном уровне (например, запрос-ответ перед началом постоянного трафика), чтобы отсекать ложные UDP-пакеты от неподтвержденных адресов.

QUIC включает встроенную защиту от amplication-атак: сервер перед отправкой крупных ответных данных требует от клиента специальный токен, подтверждающий его IP [9]. Это препятствует ситуации, когда злоумышленник отправляет маленький UDP-запрос с подделанного адреса жертвы, а сервер шлет на жертву большой ответ, перегружая ее (классическая UDP amplification attack).

Также QUIC шифрует номера пакетов и не позволяет просто *угадывать* последовательности для вставки поддельных пакетов. Все эти новшества дают основания полагать, что переход на QUIC в будущем станет стандартом и для игровых протоколов – по аналогии с тем, как веб трафик уже мигрирует на HTTP/3 (на базе QUIC) ради скорости и безопасности [13].

Однако следует помнить, что никакие улучшенные транспорты не отменяют ограничений физической сети и логики игры. Протокол — лишь основа, а ключевое слово в сетевом коде игр – это адаптация. Хороший протокол должен поддерживать разные классы трафика, чтобы разработчики могли настроить поведение под свой жанр. В этом смысле QUIC и SCTP предоставляют значительно более богатый инструментарий, чем TCP или UDP.

С другой стороны, если игра относительно проста по сетевому трафику (например, пошаговая или с малым числом игроков), то задействовать сложный протокол нет смысла – достаточно стандартного TCP с включенным шифрованием TLS 1.3, задержка от которого минимальна. Далее мы предлагаем методику выбора протокола для различных типов игр, обобщающую рассмотренные данные.

3.3. Методика выбора транспортного протокола по жанру и требованиям

Универсального решения, подходящего всем играм, не существует — каждый жанр предъявляет свои требования. На основе анализа свойств протоколов и опыта индустрии можно сформулировать алгоритм принятия решения о выборе транспортного уровня для новой игры.

Сначала требуется оценить требуемую чувствительность к задержкам. Если игровая механика предполагает мгновенную реакцию (action/FPS, файтинг, гоночная игра), порог чувствительности ~50–100 мс. Если геймплей более

умеренный (MOBA, RPG) — допустимы задержки 100–200 мс без критического ущерба. Пошаговые и асинхронные игры терпимы к секундам задержки. Чем выше требования к *real-time*, тем больше вес имеют характеристики протокола, обеспечивающие минимальный ping (отказ от лишних рукопожатий, отсутствие накопления очереди при потерях).

Далее определить критичность доставки каждого типа сообщений, так как не все игровые данные равнозначны. Выделите классы:

- *Обновления состояния* (позиции, ориентации, вспомогательные эффекты) — обновляются часто, устаревают быстро. Такие сообщения можно делать ненадежными и невпарными: потерянный апдейт заменится следующим. Пример: координаты игрока в FPS, обновляющиеся 60 раз/с — потеря одного кадра не должна тормозить игровой процесс [2].

- *Критические события* (выстрел, удар, использование способности, результаты матча) — влияют на игровую логику и должны дойти гарантированно. Они требуют надежной доставки и зачастую сохранения порядка относительно друг друга. Пример: команда на выстрел должна обработаться до команды на попадание или перезарядку.

- *Большие данные* (загрузка карты, ресурсы) — передаются редко и целиком, можно использовать надежную передачу, но не обязательно вписывать в общий поток обновлений.

- *Чат и социальные сообщения* — обычно не зависят от состояния мира, могут идти отдельным каналом, допускающим небольшие задержки, но требующим гарантированной доставки (никто не любит, когда чат-сообщения «теряются»).

Далее выбрать базовый протокол, исходя из первых двух факторов:

- Для пошаговых и требовательных к надежности игр: целесообразно использовать TCP. Он обеспечит доставку всех команд в строго порядке. Задержка в десятки миллисекунд здесь некритична, а упрощение разработки — плюс. Рекомендуется сразу использовать TLS 1.3 поверх TCP, чтобы шифровать передаваемые данные (учетные записи, ходы и т.п.) — современные библиотеки позволяют сделать это с минимальным overhead. *Пример:* цифровая карточная игра может работать по единому защищенному TCP-соединению, передавая ходы, результаты матчей и чат.

- Для динамичных action-игр: основываясь на опыте отрасли, предпочтителен UDP с собственным надстроенным протоколом. В этом протоколе необходимо реализовать два канала: ненадежный (для постоянных обновлений состояния) и надежный (для критических событий). Можно воспользоваться готовыми решениями вроде ENet, KCP, RakNet — они работают поверх UDP и дают механизмы реквестов-повторов для важной информации. Шифрование в таких играх — тонкий момент: если игра соревновательная, то трафик нередко шифруется, чтобы предотвратить читерство через анализ пакетов (например, *wallhack* на основе сетевых пакетов). Подходит DTLS 1.3 — по сути TLS для UDP. DTLS добавит примерно 1 RTT при старте и незначительно увеличит каждый пакет, но существенно повысит безопасность (шифрование и проверка целостности). Впрочем, некоторые разработчики могут решить не шифровать позиционные обновления, а ограничиться авторитетностью сервера для борьбы с читами (чтобы даже перехватив пакеты, злоумышленник не смог ничего «выиграть», так как сервер все перепроверяет). *Пример:* в соревновательном шутере сервер принимает UDP-пакеты с координатами, а важные события (выстрел, попадание) клиент посылает с флагом надежности — до получения подтверждения от сервера он может повторять отправку. Сервер шлёт обновления позиций всех игроков ненадежно 20–30 раз/с, а события убийств – надежно. При этом весь трафик может быть обернут в DTLS для скрытия от внешнего наблюдения.

- Для смежных случаев (умеренная динамика): можно рассмотреть гибридные подходы. Если игра сочетает экшен и значимые события (например, MMO RPG с экшен-боевкой), возможны два варианта. Первый — как и для экшена, полностью на UDP с выделением надежных подпротоколов. Второй — разделить типы трафика: игровые действия по UDP, а менее срочные данные (например, инвентарь, торговля, чат) по отдельному TCP-соединению. Такой подход применяется, например, в некоторых MMO: клиент держит два соединения — UDP для боевых событий и TCP для фоновых сервисов. Разработчикам приходится следить, чтобы TCP-трафик не перегружал канал и не вызывал потери UDP (см. шаг 2). Поэтому обычно объем через TCP минимален. Альтернатива — использовать SCTP или QUIC, которые внутри одного соединения имеют несколько потоков. QUIC в этом случае выглядит предпочтительнее из-за встроенного шифрования и более дружелюбного отношения сетевого оборудования к UDP (SCTP-пакеты могут фильтроваться, а QUIC-пакеты с точки зрения сети — это обычный UDP). *Пример:* в мобильной онлайн-игре, где сети подвержены высоким RTT и потерям, внедрение QUIC позволило бы упростить взаимодействие с сервером: клиент устанавливает одно зашифрованное соединение и открывает в нем стрим 1 для непрерывных обновлений позиций (ненадежный через DATAGRAM), стрим 2 для отправки пользовательских команд (надежный), стрим 3 для чата (надежный, низкий приоритет). Потери в стриме 1 не мешают своевременно доставлять команды по стриму 2, а все вместе идет по единому порту (UDP/443, например), что облегчает прохождение через NAT и фаерволы.

- Для браузерных игр: выбор ограничен веб-стеком. Традиционно WebSocket (поверх TCP) — но это влечет проблемы с задержками. Сейчас все большей поддержкой пользуется WebRTC Data Channels — фактически это SCTP поверх UDP + DTLS. Для браузерного реального времени определенно стоит выбирать WebRTC, так как он позволяет приблизиться по характеристикам к UDP-протоколу с шифрованием и надежностью при необходимости. Настройка WebRTC сложнее, но на стороне клиента (браузера) многое сделает встроенная реализация. *Пример:* HTML5-игра использует peer-to-peer соединение между игроками через WebRTC: браузеры обмениваются позициями по ненадежным сообщениям SCTP, а события по надежным, и весь трафик шифруется DTLS автоматически.

- Для облачных игр и стриминга: когда игровой процесс рендерится на сервере, а игроку передается видеопоток, требования несколько иные. Основной трафик — видео/аудио — идёт по UDP с механизмами компенсации потерь (например, протокол RTP с Forward Error Correction). Здесь QUIC тоже начинает применяться: ведутся работы по интеграции RTP поверх QUIC для облачных игр, чтобы воспользоваться быстрым соединением и обходом потерь [14],

[15]. Управляющие команды игрока (нажатия клавиш) передаются надежно, но они очень маленькие и хорошо вписываются в тот же QUIC/UDP поток. Таким образом, для cloud gaming вероятен переход на QUIC как единый транспорт для и медиа, и команд.

При этом также стоит учитывать масштабы и массовость игры. Если игра рассчитана на небольшие сессии (скажем, матч 5v5), можно позволить более «жадный» к ресурсу протокол, так как нагрузка на сервер не столь велика. Если же речь о MMO с тысячами одновременных игроков, важна эффективность: протокол должен минимизировать overhead на одного клиента. TCP в этом отношении не всегда оптимален — например, тысячи TCP-соединений создают нагрузку на серверную память (буферы, окна). UDP более легковесен, но потребует, возможно, реализации свойств, которые TCP дал бы из коробки.

Кроме того, в массовых играх иногда применяются UDP-мультикаст или broadcast внутри кластеров — TCP такого не умеет. Следовательно, для *массовых* миров чаще выбирают UDP-базированную схему. Отчасти поэтому крупные MMO (WoW, EVE Online) исторически работают на UDP со своим сетевым движком. Исключения — разве что браузерные MMO, которым пришлось использовать WebSocket (TCP) до появления WebRTC. Сегодня, с развитием WebTransport (перспективы QUIC в браузере), вероятно массовые веб-игры тоже перейдут на UDP-подобные транспорты.

Далее необходимо определиться, какие угрозы актуальны. Если игра киберспортивная, защита трафика от перехвата и модификации — приоритет (иначе профессионалы могут пытаться получить любым путем преимущество). Здесь без шифрования не обойтись: DTLS или QUIC. Также стоит внедрить аутентификацию пакетов (например, HMAC на уровне приложения для критических команд, чтобы сервер подтверждал подлинность отправителя помимо IP). Если же игра кооперативная PvE или мобильная, где ставки ниже, можно сэкономить на полном шифровании, но хотя бы контроль целостности команд и проверку на сервере надо иметь. *Пример:* Гибадуллин и др. предложили метод ассоциативной стеганографии для защиты передаваемых данных — по сути, это внедрение секретных меток в поток байт, затрудняющее его анализ сторонними наблюдателями [16]. Подобные научные методы пока редко используются на практике в играх, но сам тренд на совмещение безопасности с передачей данных очевиден. В этой связи QUIC выглядит привлекательно, так как обеспечивает шифрование без дополнительных усилий со стороны разработчика, а его использование в игре неочевидно для злоумышленника (трафик неотличим от обычного HTTPS).

После выбора протокола необходимо провести нагрузочное тестирование в условиях, приближенных к реальным: эмулировать потери, высокий пинг, ограниченную пропускную способность. Возможно, выяснится, что, скажем, выбранный алгоритм ретрансляции для UDP вызывает лаги при 5% потери пакетов — тогда надо его дорабатывать (например, внедрять *Selective Acknowledgments* или Forward Error Correction для критичных данных) [10]. Архитектура игры должна оставаться гибкой: не «зашивать» жестко зависимость от одного транспорта. В идеале движок должен поддерживать быстрый переключатель (например, fallback на TCP при недоступности UDP — некоторые движки делают это автоматически, хотя такая замена никогда не равнозначна по опыту игрока). Мир сетевых протоколов развивается: уже сейчас появляются реализации HTTP/3 (QUIC) для игровых серверов, и возможно, что через пару лет это станет дефолтом. Поэтому закладывать поддержку новых протоколов — дальновидно.

Применение вышеприведенной методики обеспечит обоснованный выбор транспорта. Подводя итог, можно сказать, что для каждой категории игр оптимальны разные решения. Пошаговые и медленные игры отлично работают на надёжных потоковых протоколах (TCP/TLS), тогда как высокоскоростные экшены требуют гибкости UDP. Протоколы нового поколения — QUIC, SCTP — обещают объединить преимущества, но их внедрение должно быть тщательно проверено в реальных условиях конкретной игры. Главная цель — достичь *баланса между надежностью, скоростью и безопасностью*, учитывая ожидания игроков и технические ограничения. Именно от правильного выбора транспортного протокола во многом зависит плавность геймплея и успех сетевого проекта на конкурентном рынке.

Заключение

Проведенный анализ показал, что выбор транспортного протокола для многопользовательской игры — многогранная задача, требующая учета жанровых особенностей, сетевых условий и угроз безопасности. TCP обеспечивает гарантированную доставку и упорядоченность, что целесообразно для игр без жестких ограничений по задержке (пошаговые стратегии, экономические симуляторы) и для вспомогательных сервисов (чаты, транзакции). Однако его применение в реактивных играх реального времени ограничено из-за эффекта блокировки по очереди и агрессивного контроля перегрузки, не учитывающего специфики игрового трафика. UDP стал де-факто базой для сетевого кода экшен-игр благодаря минимальным задержкам и свободе разработки собственных протоколов поверх него. Его использование требует внедрения на прикладном уровне механизмов надежности, фильтрации пакетов и защиты, но взамен дает максимальную отзывчивость — критичный параметр для соревновательных жанров. Появление протокола QUIC и аналогичных решений знаменует новую веху: игры получают возможность использовать одновременно несколько потоков данных с различными требованиями, не жертвуя ни скоростью, ни безопасностью. QUIC уже доказал эффективность в веб-среде, и ожидается расширение его применения в игровой индустрии, особенно по мере роста требований к шифрованию трафика и поддержки сложных взаимодействий (например, облачные игры, где нужны и быстрые медиа-каналы, и надежные управляющие каналы). SCTP, хоть и не получил широкого распространения, послужил источником ценных идей (multi-streaming), которые теперь реализованы в QUIC и даже в WebRTC для браузерных игр.

Важным выводом работы является понимание, что одним протоколом часто невозможно удовлетворить все потребности игры. Лучшие результаты достигаются при сочетании нескольких подходов: разделении трафика по категориям, гибридных схемах и использовании продвинутых библиотек сетевого кода. Представленная методика выбора протокола по жанру и техническим требованиям обобщает новейшие рекомендации: от указания Янкевича о критичности быстрого обмена для систем реального времени до современных практик защиты трафика, описанных

Гибадуллиным и соавт.. Следуя ей, разработчики могут обоснованно принять решение и при необходимости аргументировать его с точки зрения академических знаний и опыта индустрии.

Научный и практический анализ транспортных протоколов демонстрирует, что сетевой код игры — это область постоянных компромиссов. Надежность против задержки, безопасность против производительности, простота реализации против гибкости настройки — эти дилеммы приходится решать в каждом проекте. Современные тенденции (глобализация игровых сообществ, повышение значимости кибербезопасности, развитие мобильного гейминга) диктуют всё более строгие требования к сетевым протоколам. Поэтому разработчикам необходимо следить за эволюцией транспортного уровня (новые RFC, результаты исследований) и вовремя адаптировать свои технологии. Оптимальный выбор протокола и грамотная реализация сетевого взаимодействия в игре напрямую влияют на впечатления пользователей. Можно заключить, что будущее — за адаптивными, многофункциональными транспортными протоколами, способными обеспечивать низкие задержки, высокую пропускную способность и встроенную защиту. Правильно балансируя эти компоненты, индустрия игр сможет предоставить игрокам все более гладкий, честный и безопасный опыт сетевого взаимодействия.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы / References

1. Huh J.H. Reliable user datagram protocol as a solution to latencies in network games / J.H. Huh // Electronics. — 2018. — Vol. 7. — № 11. — P. 295.

2. The Ideal Game Network Protocol. — 2020. — URL: https://paytonturnage.com/writing/ideal-game-network-protocol/#:~:text=The%20big%20picture%20of%20game,light%20traveling%20through%20optic%20cable (accessed: 05.04.2025).

3. Edgegap Learner Series : What is latency in gaming? — 2024. — URL: https://edgegap.com/blog/edgegap-learner-series-what-is-latency-in-gaming (accessed: 05.04.2025).

4. Sinclair K. Challenges of Securing Massively Multiplayer Online Games / K. Sinclair [et al.] // arXiv preprint arXiv:2311.07887. — 2023. — 18 p.

5. 49% of DDoS attacks targeted gaming organizations. — 2024. — URL: https://www.securitymagazine.com/articles/100943-49-of-ddos-attacks-targeted-gaming-organizations#:~:text=49,the%20same%20period%20in%202023 (accessed: 05.04.2025).

6. Gavrić N. Security Concerns in MMO Games—Analysis of a Potent Application Layer DDoS Threat / N. Gavrić, Ž. Bojović // Sensors. — 2022. — Vol. 22. — № 20. — P. 7791. — DOI: 10.3390/s22207791.

7. Янкевич Н.С. Применение теории игр при разработке алгоритма управления транспортными потоками в интеллектуальной транспортной системе / Н.С. Янкевич // Доклады БГУИР. — 2024. — Т. 22. — № 1. — С. 74–81.

8. Yahyavi A. Peer-to-peer architectures for massively multiplayer online games: A survey / A. Yahyavi, B. Kemme // ACM Computing Surveys (CSUR). — 2013. — Vol. 46. — № 1. — P. 1–51. — DOI: 10.1145/2522968.2522970.

9. Comparing TCP and QUIC. — 2022. — URL: https://blog.apnic.net/2022/11/03/comparing-tcp-and-quic/#:~:text=A%20QUIC%20connection%20starts%20with,significant%20improvement%20in%20session%20performance (accessed: 05.04.2025).

10. Eghbal N. Lower-Latency Screen Updates over QUIC with Forward Error Correction / N. Eghbal, P. Lu // Future Internet. — 2025. — Vol. 17. — № 2. — P. 45. — DOI: 10.3390/fi17020045.

11. Amet M. A Performance Evaluation of QUIC in Real-Time Networks / M. Amet, L. Thomas, Y.Q. Song // Proceedings of the 32nd International Conference on Real-Time Networks and Systems. — 2024. — P. 255–265. — DOI: 10.1145/3648400.3648425.

12. WebRTC vs. WebSockets for multiplayer games. — 2023. — URL: https://developers.rune.ai/blog/webrtc-vs-websockets-for-multiplayer-games#:~:text=Since%20WebRTC%20originated%20from%20the,Luckily%2C (accessed: 05.04.2025).

13. Kempf M. Quic on the fast lane: Extending performance evaluations on high-rate links / M. Kempf [et al.] // Computer Communications. — 2024. — Vol. 223. — P. 90–100. — DOI: 10.1016/j.comcom.2024.03.012.

14. QUIC matches TCP's efficiency, says our research // Fastly. — 2023. — URL: https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency#:~:text=QUIC%20vs%20TCP%3A%20Which%20is,streams%2C%20efficient%20congestion%20control%2C (accessed: 05.04.2025).

15. RTP over QUIC. — 2023. — URL: https://www.ietf.org/archive/id/draft-ietf-avtcore-rtp-over-quic-04.html#:~:text=RTP%20over%20QUIC%20,RTCP%29%20packets (accessed: 05.04.2025).

16. Гибадуллин Р.Ф. Разработка декоратора stegostream для ассоциативной защиты байтового потока / Р.Ф. Гибадуллин, Д.А. Гашигуллин, И.С. Вершинин // Моделирование, оптимизация и информационные технологии. — 2023. — Т. 11. — № 2. — С. 41.

Список литературы на английском языке / References in English

1. Huh J.H. Reliable user datagram protocol as a solution to latencies in network games / J.H. Huh // Electronics. — 2018. — Vol. 7. — № 11. — P. 295.

2. The Ideal Game Network Protocol. — 2020. — URL: https://paytonturnage.com/writing/ideal-game-network-protocol/#:~:text=The%20big%20picture%20of%20game,light%20traveling%20through%20optic%20cable (accessed: 05.04.2025).

3. Edgegap Learner Series : What is latency in gaming? — 2024. — URL: https://edgegap.com/blog/edgegap-learner-series-what-is-latency-in-gaming (accessed: 05.04.2025).

4. Sinclair K. Challenges of Securing Massively Multiplayer Online Games / K. Sinclair [et al.] // arXiv preprint arXiv:2311.07887. — 2023. — 18 p.

5. 49% of DDoS attacks targeted gaming organizations. — 2024. — URL: https://www.securitymagazine.com/articles/100943-49-of-ddos-attacks-targeted-gaming-organizations#:~:text=49,the%20same%20period%20in%202023 (accessed: 05.04.2025).

6. Gavrić N. Security Concerns in MMO Games—Analysis of a Potent Application Layer DDoS Threat / N. Gavrić, Ž. Bojović // Sensors. — 2022. — Vol. 22. — № 20. — P. 7791. — DOI: 10.3390/s22207791.

7. Yankevich N.S. Primenenie teorii igr pri razrabotke algoritma upravlenija transportnymi potokami v intellektual'noj transportnoj sisteme [Application of game theory in developing a traffic flow control algorithm in an intelligent transportation system] / N.S. Yankevich // Doklady BGUIR [Proceedings of BSUIR]. — 2024. — Vol. 22. — № 1. — P. 74–81. [in Russian]

8. Yahyavi A. Peer-to-peer architectures for massively multiplayer online games: A survey / A. Yahyavi, B. Kemme // ACM Computing Surveys (CSUR). — 2013. — Vol. 46. — № 1. — P. 1–51. — DOI: 10.1145/2522968.2522970.

9. Comparing TCP and QUIC. — 2022. — URL: https://blog.apnic.net/2022/11/03/comparing-tcp-and-quic/#:~:text=A%20QUIC%20connection%20starts%20with,significant%20improvement%20in%20session%20performance (accessed: 05.04.2025).

10. Eghbal N. Lower-Latency Screen Updates over QUIC with Forward Error Correction / N. Eghbal, P. Lu // Future Internet. — 2025. — Vol. 17. — № 2. — P. 45. — DOI: 10.3390/fi17020045.

11. Amet M. A Performance Evaluation of QUIC in Real-Time Networks / M. Amet, L. Thomas, Y.Q. Song // Proceedings of the 32nd International Conference on Real-Time Networks and Systems. — 2024. — P. 255–265. — DOI: 10.1145/3648400.3648425.

12. WebRTC vs. WebSockets for multiplayer games. — 2023. — URL: https://developers.rune.ai/blog/webrtc-vs-websockets-for-multiplayer-games#:~:text=Since%20WebRTC%20originated%20from%20the,Luckily%2C (accessed: 05.04.2025).

13. Kempf M. Quic on the fast lane: Extending performance evaluations on high-rate links / M. Kempf [et al.] // Computer Communications. — 2024. — Vol. 223. — P. 90–100. — DOI: 10.1016/j.comcom.2024.03.012.

14. QUIC matches TCP's efficiency, says our research // Fastly. — 2023. — URL: https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency#:~:text=QUIC%20vs%20TCP%3A%20Which%20is,streams%2C%20efficient%20congestion%20control%2C (accessed: 05.04.2025).

15. RTP over QUIC. — 2023. — URL: https://www.ietf.org/archive/id/draft-ietf-avtcore-rtp-over-quic-04.html#:~:text=RTP%20over%20QUIC%20,RTCP%29%20packets (accessed: 05.04.2025).

16. Gibadullin R.F. Razrabotka dekoratora stegostream dlya assotsiativnoj zashhity bajtovogo potoka [Development of the stegostream decorator for associative byte stream protection] / R.F. Gibadullin, D.A. Gashigullin, I.S. Vershinin // Modelirovanie, optimizacija i informacionnye tehnologii [Modeling, Optimization and Information Technologies]. — 2023. — Vol. 11. — № 2. — P. 41. [in Russian]