

DOI: 10.15514/ISPRAS-2026-38(1)-13



Анализ и оптимизация задержек видеотрансляции удалённых рабочих столов

И.С. Проскуряков, ORCID: 0009-0005-1559-6278 <ivan.proskuryakov@ispras.ru>

Д.А. Бадалян, ORCID: 0000-0002-0642-9102 <dbadalyan@ispras.ru>

*Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. Для повышения эффективности работы удалённых рабочих столов ключевое значение имеет оптимизация передачи видеоданных, обеспечивающая стабильность и качество изображения при ограниченных ресурсах сети и клиента. Современные подходы включают использование специализированных протоколов передачи данных и видеокодеков, оптимизированных для работы в реальном времени. Целью данной работы является исследование и оптимизация задержек в процессе передачи видео в системе предоставления доступа к удалённым рабочим столам с использованием GStreamer для обработки видео и WebRTC для передачи данных по сети. В результате работы выявлены основные источники задержек внутри конвейеров подготовки видео удалённого рабочего стола. Проведён сравнительный анализ поведения различных конвейеров в различных сценариях и выявлены их преимущества и недостатки в контексте оптимизации трансляции изображения рабочего стола. В ходе анализа эффективности и предсказуемости рассматриваемых GStreamer-конвейеров иллюстрируются подходы и инструменты оценки данного класса конвейеров подготовки кадров. Отдельным результатом являются полученные методы оптимизации, перечисленные в разделе 3.

Ключевые слова: видеотрансляции; удаленные рабочие столы; видеокодеки; стандарт H.264; стандарт AV1; стандарт VP9; платформа GStreamer; анализ задержек; инструмент GstShark.

Для цитирования: Проскуряков И.С., Бадалян Д.А. Анализ и оптимизация задержек видеотрансляции удалённых рабочих столов. Труды ИСП РАН, том 38, вып. 1, 2026 г., стр. 183–200. DOI: 10.15514/ISPRAS-2026-38(1)-13.

Благодарности: авторы выражают благодарность О.Д. Борисенко, Н.А. Лазареву, А.М. Бирюкову за помощь в подготовке текста. Исследования выполнены в интересах ООО «Системные решения» и дистрибутива “Moncloud”.

Analysis and Optimization of Video Transmission Delays in Remote Desktops

I.S. Proskuriakov, ORCID: 0009-0005-1559-6278 <ivan.proskuryakov@ispras.ru>

D.A. Badalyan, ORCID: 0000-0002-0642-9102 <dbadalyan@ispras.ru>

*Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. To improve the efficiency of remote desktop operations, the optimization of video data transmission is of key importance, ensuring stability and image quality under limited network and client resources. Modern approaches include the use of specialized data transmission protocols and video codecs optimized for real-time operation. The aim of this work is to investigate and optimize latency in the video transmission process within a remote desktop access provisioning system, utilizing GStreamer for video processing and WebRTC for network data transmission. As a result of the work, the main producers of latency inside the pipelines of remote desktop video preparation have been identified. A comparative analysis of the behavior of different pipelines in different scenarios has been carried out and their advantages and disadvantages have been identified in the context of optimizing the translation of the desktop image. During the analysis of the effectiveness and predictability of the considered GStreamer pipelines, approaches and tools for evaluating this class of training pipelines are illustrated. A separate result is the optimization methods listed in section 3.

Keywords: video streaming; remote desktops; video codecs; H.264; VP9; AV1; GStreamer; latency analysis; GstShark.

For citation: Proskuriakov I.S., Badalyan D.A. Analysis and Optimization of Video Transmission Delays in Remote Desktops. *Trudy ISP RAN/Proc. ISP RAS*, vol. 38, issue 1, 2026, pp. 183-200 (in Russian). DOI: 10.15514/ISPRAS-2026-38(1)-13.

Acknowledgements. The authors express their gratitude to O.D. Borisenko, N.A. Lazarev, and A.M. Biryukov for their assistance in preparing the text. This work was conducted in the interest of System Solutions LLC and Moncloud Distribution.

1. Введение

Под системой по предоставлению доступа к виртуальным удалённым рабочим столам (Virtual Desktop Infrastructure, VDI) здесь и далее понимается система, автоматизированно размещающая среды рабочих столов на некотором наборе виртуальных или физических серверов и предоставляющая доступ к ним. Среда рабочих столов включает в себя операционную систему и прикладное ПО. Среда виртуальных рабочих столов помещается в виртуальные машины, за управление и подключение к которым отвечает система VDI за счёт наличия слоя-перехватчика между клиентом и рабочим столом.

Одним из ключевых аспектов функционирования систем по предоставлению удалённого доступа к рабочим столам является трансляция изображения. Трансляция, производимая специальным компонентом системы, должна обеспечивать приемлемое качество изображения, минимальную задержку и стабильность даже в условиях ограниченной пропускной способности сети.

Требования к качеству изображения, задержке и стабильности делают оптимизационную задачу комплексной: необходимо обеспечить компромисс между качеством изображения, уровнем сжатия, количеством затрачиваемых на кодирование и декодирование видео временных и вычислительных ресурсов с учётом пропускной способности сетевого канала между удалённым вычислителем и клиентской машиной.

Цель данной работы – исследование поведения конвейеров подготовки кадров в системе трансляции VDI для их дальнейшей оптимизации. Для этого проводится качественный анализ поведения конвейеров подготовки кадров, используемых в разрабатываемой в ИСП РАН системе трансляции удалённых рабочих столов Wormhole, в контексте реализации

выделенных методов оптимизации трансляции. Новизна полученных результатов заключается в описании узких мест данных конвейеров и побочных эффектов работы некоторых стандартных объектов конвейеров в контексте системы трансляции VDI. В ходе проведённого анализа демонстрируются оценки и инструменты, которые могут быть использованы при профилировании производительности конвейеров подготовки кадров в VDI системе.

Wormhole состоит из клиентской и серверной части: с серверной ведётся трансляция на клиент, для трансляции используется технология WebRTC [1]. Захват экрана удалённого рабочего стола и кодирование видео осуществляются инструментами GStreamer. Клиентом выступает веб-браузер на базе проекта Chromium. Обобщённая архитектура Wormhole показана на рис. 1.

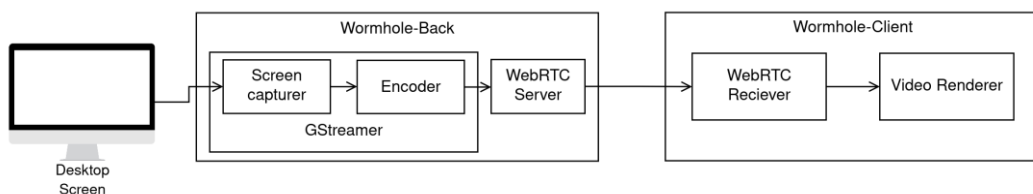


Рис. 1. Обобщённая архитектура компоненты трансляции Wormhole.
Fig. 1. Generalized architecture of the Wormhole streaming component.

2. Обзор

В данном разделе приводится краткий обзор тематик публикуемых исследований в контексте виртуальных рабочих столов, получивших название VDI систем (Virtual Desktop Infrastructure), а также справочная информация о наборе технологий, задействованном в задаче видеотрансляции удалённых рабочих столов.

Ряд исследований VDI посвящён оценке эффективности наиболее известных проприетарных систем – VMWare, Citrix, Microsoft. Так, в [2] оценивается поведение данного класса VDI систем в различных сетевых условиях с точки зрения качества предоставляемого сервиса (Quality of Service, QoS). В работах [3–4] оценивается производительность VDI на основе открытых технологий с использованием графических процессоров для улучшения производительности рабочих сред. Ряд исследований затрагивают тему оптимизации доставки трафика и решают оптимизационную задачу путём оптимального размещения сред в облачной среде с помощью технологии периферийных вычислений (edge-computing) [5], с помощью пересмотра подхода к представлению и сжатию кадра для специфичного контента путём реализации собственных решений [6]. Общим у приведённых исследований является отсутствие фокуса на анализе самого процесса подготовки кадров на стороне VDI с использованием актуальных технологий видеокодирования, поэтому далее остановимся на используемых для этого технологиях подробнее.

На рис. 2 показана последовательность, в которой происходит формирование видеокадров. Первым является этап получения набора снимков рабочего стола в RGB-подобном формате. Это делается штатными средствами операционных систем.

Далее происходит этап конвертации, необходимый для представления цвета в виде одной яркостной компоненты и двух цветовых (YUV). Этот подход позволяет эффективно сжимать данные на этапе кодирования, осуществляемого кодировщиком.

Под кодированием данных понимается преобразование из исходного формата в другой, как правило, наиболее пригодный для определённого сценария взаимодействия с пользователем. Кодирование осуществляется кодировщиками – программными или программно-аппаратными реализациями стандартов представления и сжатия видео. Наиболее известные

в области стандарты – H.264, H.265, VP9, AV1, где H.265 и AV1 имеют специальные режимы для оптимизации кодирования видео экрана рабочего стола [7:Chapter 3.8, 8-9].

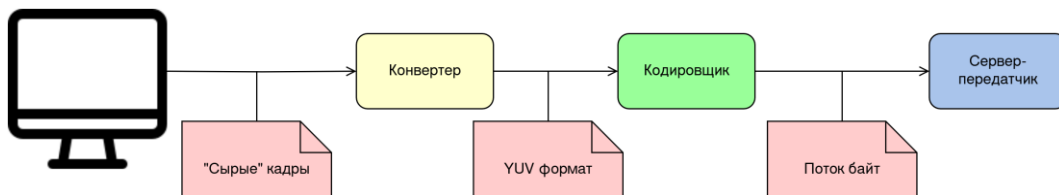


Рис. 2. Последовательность подготовки данных для видеотрансляции удалённого рабочего стола.

Fig. 2. Remote desktop video streaming data preparation sequence.

Для упрощения работы с каждым из перечисленных этапов разработаны системы обработки видео контента, позволяющие выстраивать сложные сценарии работы с медиа, в том числе с использованием программных интерфейсов. Среди данного класса систем выделяются две – FFmpeg и GStreamer. Система FFmpeg представляет собой набор библиотек по кодированию, декодированию, транскодированию, мультимплексированию, демультимплексированию, фильтрации, стримингу и воспроизведению видео. Система GStreamer имеет ту же функциональность, что и FFmpeg, однако отличается по своей структуре, предоставляя возможность конструирования графов из компонентов для обработки медиа данных и позволяя конструировать конвейеры обработки данных со стандартизированными интерфейсами.

Финальным этапом является отправка данных получателю посредством специальных сетевых технологий. В случае с системами трансляции видео удалённого рабочего стола необходимо иметь возможность трансляции в режиме, близком к режиму реального времени, то есть с задержкой около 100 мс [10]. Данным требованиям удовлетворяют 3 открытых технологии – WebRTC (Web Real-Time Communication) [10], протокол WebTransport совместно с интерфейсом WebCodecs [11], протокол QUIC [12]. WebRTC является достаточно сложной технологией и требует использования протоколов обхода NAT (Network Address Translation), таких как ICE [13], STUN [14], TURN [15]. Протокол WebTransport целиком полагается на стандарт HTTP/3 [16], из-за чего уходит необходимость в реализации механизма обхода протокола NAT [17]. Однако HTTP/3 и связанный с ним протокол QUIC ещё не получили достаточного распространения [18].

3. Описание методов оптимизации процесса видеотрансляции в задаче удалённых рабочих столов

Вынося за рамки общие методы оптимизации передачи аудио- и видеоконтента в информационной сети в режиме реального времени (видеостриминг), такие, например, как выбор оптимального профиля кодирования, размера группы изображений, а также методы оптимизации работы транслируемой операционной системы внутри VDI, можно выделить следующие подходы к оптимизации видеотрансляций в задаче VDI.

1. Использование кодировщиков видеоизображений с механизмами оптимизации кодирования изображения, характерного для рабочего стола персонального компьютера. Задача кодирования такого класса медиаданных получила название технологии кодирования содержимого экрана (экранного контента) SCC (Screen Content Coding), методы её оптимального решения предусматриваются стандартами видеокодирования AV1 и H.265 [19].
2. Использование аппаратных реализаций кодировщиков видео: стандарты H.264, H.265 и AV1 имеют аппаратные реализации на графических процессорах [20-22] и в графических компонентах процессоров [23].

3. Использование специализированных систем, таких как Microsoft DirectX [24] и Vulkan [25], для получения доступа к ресурсам графических процессоров для аппаратного декодирования.
4. Обеспечение zero-copу подхода при работе с медиаданными: копирование является операцией, затратной по времени и по памяти.
5. Обеспечение стабильного сетевого канала без перегрузок: в задаче оптимизации видеотрансляций с помощью WebRTC широко распространено использование алгоритмов контроля за недопущением перегрузки канала (Congestion Control) для корректировки частоты видеопотока по текущему состоянию сетевого канала. В данном классе алгоритмов выделяются алгоритмы GCC [26], NADA [27], SCReAM [28]. Аналогичный подход с использованием обратной связи потенциально возможно реализовать и для сценария трансляции удалённых рабочих столов.

Для оценки практической эффективности 4-х из 5 предложенных выше оптимизаций необходимо иметь инструментарий, контролирующий изменения задержек, формируемых конвейером подготовки кадров.

4. Методика сбора данных

В разделе приводится информация о структуре анализируемых GStreamer конвейеров подготовки видео, а также основная информация об используемом в работе инструменте профилирования – GstShark.

4.1 Описание структуры GStreamer конвейеров

GStreamer конвейеры состоят из объектов, проводящих манипуляции над данными. Для передачи данных из конвейера внешним потребителям используется специальный объект – ‘appsink’. В случае, если объекты работают в разном темпе, и подозревается ситуация, когда предыдущий этап конвейера данные проходят быстрее, чем следующий, во избежание потерь данных между обработчиками используются очереди (queue), осуществляющие буферизацию и выполнение политик по времени хранения буферизированных данных [29].

Данные передаются между объектами конвейеров в специальных контейнерах, так называемых буферах (buffers). Эти данные могут быть снабжены набором метаданных, такой как формат данных, длительность находящегося в буфере видеофрагмента и так далее.

4.2 Описание инструмента сбора данных

Для оценки эффективности конфигураций конвейеров подготовки видео необходимо получать информацию об этапах формирования задержек на каждом из этапов, которые проходят данные при обработке конвейером GStreamer. Данную функциональность предоставляет находящийся в открытом доступе инструмент GstShark [30]. В составе инструмента имеется ряд трассировщиков, предоставляющих данные о поведении конвейера:

1. ‘bitrate’ – измеряет и выводит частоту (количество бит в секунду) данных, выходящих из src-интерфейса объекта конвейера.
2. ‘buffer’ – выводит информацию о каждом буфере (единице-контейнере данных в библиотеке GObject, используемой системой GStreamer для реализации концепции ООП), покидающем src-интерфейс объекта конвейера.
3. ‘cpuusage’ – выводит информацию о загрузке ядер ЦПУ.
4. ‘framerate’ – измеряет и выводит частоту выводимых объектом конвейера кадров, если такая информация объектом предоставляется.
5. ‘graphic’ – визуализирует структуру GStreamer конвейера с указанием форматов данных, проходящих через объекты.

6. ‘interlatency’ – межэлементная задержка, отображающая время прохождения буфера данных между элементами конвейера.
7. ‘proctime’ – измеряет и выводит время, потраченное объектом для передачи буфера от входного интерфейса (sink) к выходному (src).
8. ‘queuelevel’ – измеряет количество данных, пребывающих в очередях, и выводит его.
9. ‘scheduletime’ – измеряет и выводит интервал между поступлением буферов данных на входной интерфейс (sink) объекта.

Для анализа собранных данных разработчики GstShark предоставляют собственный инструмент визуализации, но при его использовании мы столкнулись с проблемой в работе компонента ‘gnuplot’. Решением стала разработка собственной программы для визуализации собранных логов с использованием Python3.11 в связке с инструментами Pandas, Numpy, Matplotlib.

Данные необходимо было собирать при работе Wormhole на ОС Windows 10, в связи с чем GstShark собирался из исходных файлов в среде MSYS2 и запускался в ней же. Такой подход позволил избежать использования дополнительной виртуализации, возникающей при использовании WSL, но пришлось отказаться от использования трассировщиков ‘сruusage’ и ‘graphic’, так как они использовали Linux-специфичные вызовы и программы.

5. Анализ наборов данных

5.1 Способ сбора наборов данных

Для сбора наборов данных и воспроизводимости полученных результатов стояла задача единообразно эмулировать некоторый сценарий работы за удалённым рабочим столом. Простейшим решением этой задачи, к которому мы прибегли, является включение на рабочем столе в полноэкранный режим заготовленной видеозаписи с определёнными сценариями – данный способ эмуляции дешёв в эксплуатации и воспроизведении. Таким способом собраны все анализируемые наборы данных.

Основной сценарий используемых в заготовленных видеозаписях – сценарий “офисной работы”, то есть работа в пакете Microsoft Excel, программе проектирования Revit и просмотр веб-страниц. Также использовался сценарий с воспроизведением динамически меняющейся сложной трехмерной графики. Далее, если не сказано иное, предполагается, что данные собраны на видео “офисной работы”.

Конфигурация экспериментального стенда следующая. Гостевая система Windows 10 на основе libvirt, хост-система – Debian 12. Топология виртуальной машины: 4 виртуальных процессора, 16 Гб ОЗУ. Топология хостовой машины: NVIDIA A16, Intel Core i7-11700, 62.6 Гб ОЗУ.

Для каждой из анализируемых ниже трасс проводилось от 3 прогонов длительностью от 5 до 15 минут с целью получения качественной картины поведения конвейера. На иллюстрациях приведены графики, построенные по трассе одной трансляции – усреднения и иные модификации наборов данных для построения данных графиков не проводилось.

5.2 Анализ конвейеров, использующих ресурсы графических процессоров

Один из путей оптимизации видеотрансляции – использование аппаратно реализованных кодировщиков видео. С целью их эффективного использования проводится исследование конвейеров на основе аппаратного кодировщика NVIDIA.

5.2.1 Сравнительный анализ поведения реализации кодировщика H.264 от NVIDIA в ключевых сценариях работы

В разделе анализируются трассы, собранные при работе видео кодировщика H.264 в реализации NVIDIA в конвейере Wormhole (плагин 'nvh264') при трансляции двух качественно разных видеорядов: сценарий А – трансляция в полноэкранном режиме на рабочий стол записи мультфильма со сложной 3D графикой; сценарий Б – трансляция в полноэкранном режиме на рабочий стол записи работы с инструментами Excel.

Фрагмент графика, построенного по данным трассировщика 'proctime', показан на рис. 3. Из графика видно, что время, затрачиваемое на кодирование в сценарии А, выше, чем время, которое затрачивается при кодировании видео работы с “офисной” нагрузкой. Это доказывает, что несмотря на отсутствие заложенных в стандарт оптимизаций в задаче SCC, H.264 может справляться с кодированием контента, характерного для рабочего стола персонального компьютера, эффективнее, чем с кодированием сложных 3D сцен. Данный вывод является не очевидным, так как изначально стандарт H.264 разрабатывался для сжатия и представления “натурального” видео, характер которого отличается от видео рабочего стола [9]. В то же время развитие H.264, стандарт H.265, имеет специальные расширения профилей кодирования для повышения эффективности работы с видео рабочего стола [8].

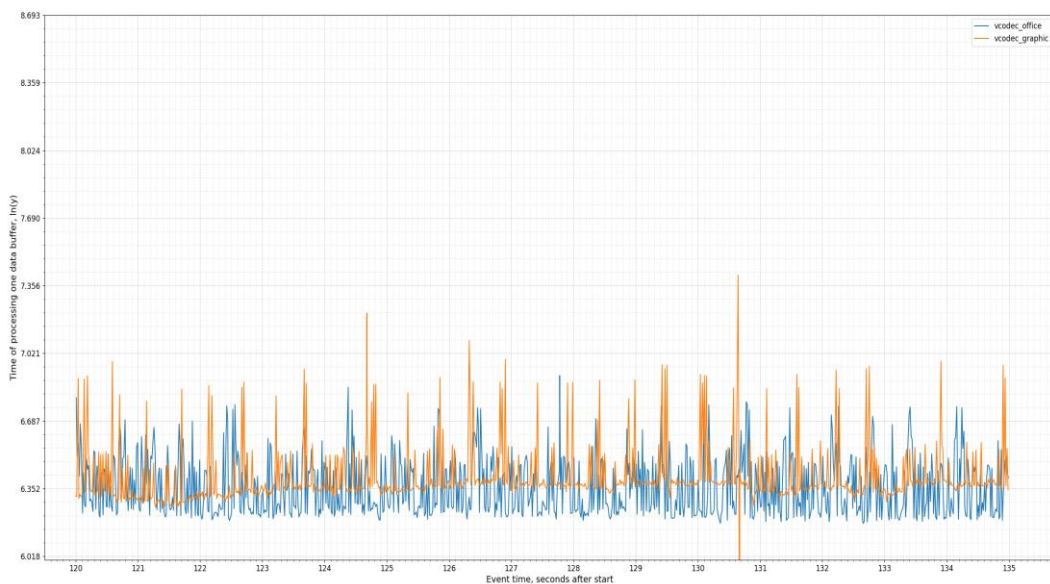


Рис. 3. Время (в логарифмической шкале), требуемое кодировщиком для передачи полученных данных для дальнейшей обработки конвейером. Оранжевая линия соответствует распределению времени кодирования трансляции сценария А, синяя – сценария Б.

Fig. 3. Encoder processing time (logarithmic scale) required to pass received data to the pipeline for further processing. The orange line represents the encoding time distribution for Scenario A, the blue line – for Scenario B.

5.2.2 Анализ конвейера на основе CUDA

CUDA – это платформа параллельных вычислений, разработанная компанией NVIDIA для вычислений на графических процессорах (GPU). Для использования аппаратно реализованных видеокодеков в графических процессорах NVIDIA используется CUDA [21, CUDA], доступная в GStreamer через набор специализированных плагинов. Для трансляции

экрана с помощью CUDA в Wormhole используется конвейер, представленный в Приложении 1.

Иллюстрация работы конвейера на основе CUDA-плагинов посредством фрагмента графика формирования задержки, построенного по данным трассировщика ‘interlatency’, показана на рис. 4. Каждая линия графика, за исключением самой верхней, соответствует логарифму времени, прошедшего от появления данных в начале конвейера (в нашем случае – на объекте захвата картинки рабочего стола) до момента выхода этих данных из объекта конвейера, указанного в легенде. Последняя линия, соответствующая объекту appsink, показывает логарифм времени до входа в объект. По горизонтали отложено время с момента начала трансляции рабочего стола до регистрации события появления данных трассировщиком.

Из рис. 4 видно, что относительно большая разница имеется под линиями, соответствующими выходам из объектов ‘queue0’, ‘cudaupload’ и ‘vcodec’. Выход из объекта ‘queue0’ соответствует попаданию данных в объект загрузки данных в память графического процессора – ‘cudaupload’. Выход из объекта ‘cudaupload’ соответствует попаданию данных из оперативной памяти вычислителя в графический процессор и началу процесса их обработки. Выход из объекта ‘vcodec’ соответствует окончанию процесса кодирования и попаданию в очередь перед отправкой на клиент. Иными словами, график показывает два ключевых источника задержки в конвейере: объект загрузки данных в память графического процессора и объект, осуществляющий кодирование видео. Также на данном фрагменте показан момент длительного отказа в приёме данных объектом ‘cudaupload’ – он соответствует пику между 998 и 999 секундами трансляции.



Рис. 4. Фрагмент графика ‘interlatency’ для конвейера на основе CUDA. По вертикали отложен логарифм времени, которое прошло с момента появления данных в начале конвейера до момента их регистрации на некотором интерфейсе некоторого объекта. По горизонтали отмечено время в секундах, прошедшее с момента начала трансляции экрана удалённого рабочего стола.

Fig. 4. A fragment of a plot built from ‘interlatency’ tracer for a CUDA-based pipeline. The vertical axis shows the logarithm of the time elapsed from the moment the data appeared at the beginning of the pipeline to the moment it was registered on a specific interface of a specific object. The horizontal axis shows the time in seconds elapsed since the start of the remote desktop screen streaming.

Как удалось выяснить, такие пики носят периодический характер и имеют накопительный эффект. Показанный на фрагменте пик, согласно проведённому исследованию логов Wormhole, привёл к ситуации потери изображения на клиенте из-за запаздывания кадров.

Вышесказанное свидетельствует, что конвейер на основе CUDA-плагинов имеет высокие издержки, связанные с загрузкой данных в память графического процессора. Периодические отказы объекта загрузки данных в память GPU приводят к ситуациям, ухудшающим пользовательский опыт.

Выявлены три пути решения данной проблемы: понижение частоты кадров, установка меньших дедлайнов на очередях в начале и в конце конвейера и локализация данных в памяти GPU средствами DirectX. Первый подход привёл к ещё более критичному ухудшению пользовательского опыта и делает нецелесообразным использование GPU для вычислений, так как при условии незначительных компромиссов возможно добиться той же частоты кадров средствами программной реализации кодировщика. Второй подход привёл к более нестабильной работе конвейера и увеличению дисперсии времени работы кодировщика, что не может поощряться в системах, работающих в режиме реального времени. Результаты третьего метода показаны в следующем разделе.

5.2.3 Анализ конвейера на основе DirectX 11

Конфигурация конвейера с использованием абстракции памяти GPU средствами DirectX 11 показана в Приложении 2. В ходе работы конвейера кадры сразу же отправляются в буфер под управлением DirectX – это становится возможным благодаря тому, что захват изображения экрана производится средствами DirectX. Данные проходят все этапы обработки внутри памяти под контролем DirectX и выгружаются в оперативную память только в самом конце – перед передачей данных в appsink.

На рис. 5 показан фрагмент графика, построенного по данным трассировщика ‘interlatency’. Данные собирались при трансляции “офисной” нагрузки средствами Wormhole. Из графика видно, что в конвейере отсутствует этап загрузки данных из оперативной памяти в память графического процессора, благодаря чему удалось ликвидировать побочные эффекты, возникающие в результате данного класса операций.

Также можно заметить по уровню линий appsink на рис. 4 и 5, что при переходе с CUDA-конвейера на конвейер на основе DirectX 11 удалось избежать увеличения затрачиваемого на подготовку кадров времени.

5.3 Анализ конвейеров, использующих программные реализации кодировщиков

В разделе анализируется на предмет этапов формирования задержек структура конвейеров на основе программно реализованных кодировщиков видео. Также приводится сравнение кодировщиков VP9 и x264 в ходе использования Wormhole под “офисной” нагрузкой – средствами GstShark демонстрируются выявленные недостатки в работе кодировщика VP9.

5.3.1 Анализ конвейера на основе программно реализованного кодировщика на примере конвейера с x264

x264 – это программная реализация стандарта кодирования H.264 с открытым исходным кодом [31]. Для работы с ней в составе GStreamer используется плагин x264enc. Полная версия анализируемого конвейера на основе x264 приведена в Приложении 3. Для анализа формируемых в ходе работы конвейера задержек используется трассировщик ‘interlatency’. Фрагмент графика ‘interlatency’ представлен на рис. 6. Каждая линия соответствует моменту появления данных на интерфейсе объекта конвейера. Соответствующие объект и его интерфейс указаны в легенде на самом графике.



Рис. 5. Фрагмент графика, построенного по данным трассировщика 'interlatency' на основе DirectX. По вертикали: логарифм времени, которое прошло с момента появления данных в начале конвейера до момента их регистрации на некотором интерфейсе некоторого объекта. По горизонтали: время в секундах, прошедшее с момента начала трансляции экрана удалённого рабочего стола.

Fig. 5. A fragment of a plot built from 'interlatency' tracer for a DirectX-based pipeline. The vertical axis represents the logarithm of the time elapsed from the moment data enters the pipeline to its registration on a specific interface of a specific object. The horizontal axis indicates the time in seconds since the start of the remote desktop screen streaming.



Рис. 6. Фрагмент графика, построенного по данным трассировщика 'interlatency' на основе x264. По вертикали отложен логарифм времени, которое прошло с момента появления данных в начале конвейера до момента их регистрации на некотором интерфейсе некоторого объекта. По горизонтали отмечено время в секундах, прошедшее с момента начала трансляции экрана удалённого рабочего стола.

Fig. 6. A fragment of a plot built from 'interlatency' tracer for a x264-based pipeline. The vertical axis represents the logarithm of the time elapsed from the moment data enters the pipeline to its registration on a specific interface of a specific object. The horizontal axis indicates the time in seconds since the start of the remote desktop screen streaming.

Как видно из фрагмента графика на рис. 6, наибольшая задержка формируется на следующих этапах:

- перед выходом данных из очереди ‘queue0’ – это очередь перед этапом конвертации;
- перед выходом данных из объекта конвертации данных;
- перед попаданием данных в appsink, то есть на этапе кодирования.

Задержка на выход из очереди ‘queue0’ говорит о том, что объект захвата кадров работает быстрее, чем обработчики данных – конвертер и кодировщик. Также видно, что в данном конвейере большая задержка формируется этапом конвертации – это характерная особенность конвейеров на основе программно реализованных кодировщиков и, как можно видеть из рис. 5 и 6, в конвейерах на основе аппаратно реализованных кодировщиков этап конвертации не занимает столь существенную долю времени работы конвейера.

В подтверждение того, что этап конвертации занимает больше времени, чем этап кодирования, может служить факт, что, согласно графику, линия выхода данных из ‘queue1’, соответствующей очереди между конвертером и кодировщиком, практически совпадает с линией выхода данных из объекта конвертера – это значит, что кодировщик практически мгновенно потребляет поступившие на вход данные.

5.3.2 Сравнительный анализ VP9 и x264 в сценарии трансляции рабочего стола в FullHD

Один из основных сценариев работы Wormhole заключается в предоставлении доступа к удалённому рабочему столу без подключенного графического процессора. Для эффективного выполнения поставленной задачи, помимо программной реализации x264 стандарта H.264, необходимо рассмотреть альтернативы. Альтернативой является кодировщик VP9 – это широко используемый открытый стандарт кодирования, разрабатывавшийся как альтернатива H.264 в задаче трансляции видео [32]. Для работы с GStreamer используется реализация ‘libvpx’. Для сравнения этих кодировщиков собраны и проанализированы данные трассировщиков, полученные в ходе трансляции через Wormhole “офисной” нагрузки через заранее записанные видеозаписи – аналогично сбору статистики в разделах выше.

Используемые для сравнения GStreamer конвейеры представлены в Приложении 4.

Для выявления особенностей работы конвейеров использовались трассировщики ‘buffer’, ‘interlatency’, ‘proctime’. Трассировщик ‘buffer’ предоставляет, в том числе, данные о длительности видеофрагментов, передаваемых конвейером в appsink. Для частоты 30 кадров в секунду (30 FPS) ожидаемая длительность передаваемого буфера данных составляет около 33 мс. Однако при просмотре статистики работы VP9 обнаружены обладающие периодичностью эпизоды, когда кодировщик передавал на выход буферы с около нулевыми длительностями. Пример такого поведения показан на 575-й секунде трансляции в фрагменте графика на рис. 7.

Для объяснения произошедшего рассматривается график по данным трассировщика ‘interlatency’ в окрестности зафиксированного падения длительности буфера данных – этот график представлен на рис. 8. На рис. 8 показано, что в окрестности 575 секунды формируется пик, соответствующий повышению времени на выход данных из очереди перед этапом конвертации – это означает, что последующие элементы не готовы принять данные для обработки. По расположению пиков на графике можно предположить, что причина в резко повысившейся задержке кодировщика – пик, соответствующий задержке кодировщика, расположен левее, чем пики конвертера и очереди. Чтобы подтвердить увиденное, обратимся к данным трассировщика ‘proctime’.

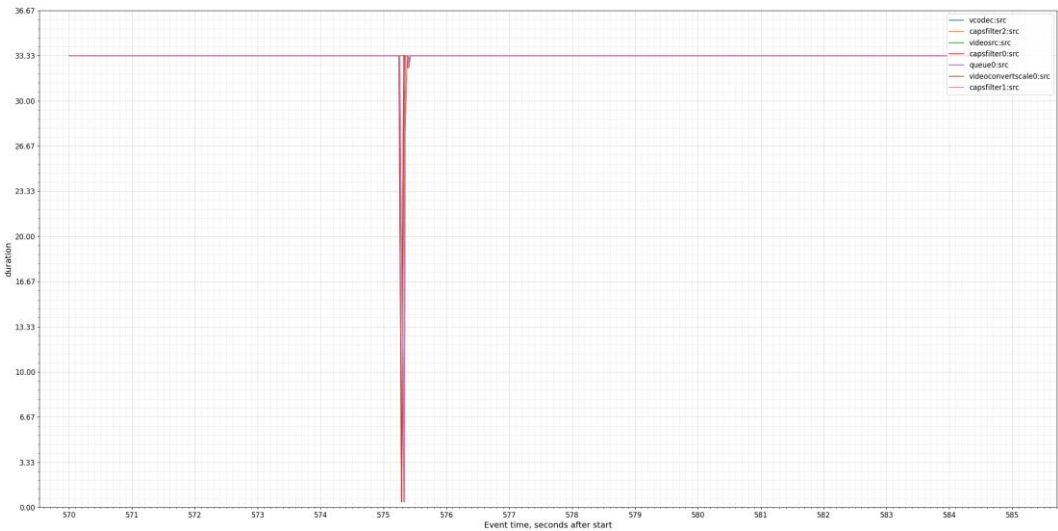


Рис. 7. Фрагмент графика по данным статистики “duration” трассировщика “buffer” по данным конвейера на основе VP9. По вертикальной оси расположены данные о длительности видеофрагмента, находящегося в *gst*-буфере данных (в мс). По горизонтальной оси отмечено время (в секундах) с момента начала трансляции рабочего стола. Падение между 575 и 576 секундами трансляции говорит о наличии аномалии в *GStreamer* конвейере: по некоторой причине длительность передаваемого видеофрагмента упала до около нулевых значений.

Fig. 7. Fragment of a plot based on “duration” statistics from the “buffer” tracer of a VP9-based pipeline. The vertical axis shows the duration of the video fragment in the *GStreamer* data buffer (in ms). The horizontal axis shows time (in seconds) since the start of desktop streaming. The drop between 575 and 576 seconds indicates an anomaly in the *GStreamer* pipeline: for some reason, the duration of the transmitted video fragment dropped to near zero values.



Рис. 8. Фрагмент графика, построенного по данным трассировщика ‘interlatency’ по данным конвейера на основе VP9. Между 575 и 576 секундами трансляции видны аномальные задержки: левее расположен пик, соответствующий задержке выхода данных из кодировщика, далее – задержки объекта конвертации форматов (зелёный) и очереди перед ним (оранжевый).

Fig. 8. Fragment of a plot based on data from the “interlatency” tracer of a VP9-based pipeline. Anomalous delays are visible between 575 and 576 seconds: to the left is a peak corresponding to the delay of data exiting the encoder, followed by the format conversion element (green) and the queue preceding it (orange).

На рис. 9 показан график трассировщика 'proctime' для соответствующего временного участка – по нему видно, что скачок времени работы наблюдается лишь у кодировщика (сначала) и очереди 'queue0' (после кодировщика). Это доказывает, что увиденное на графике 'duration' на рис. 7 падение спровоцировано именно кодировщиком.

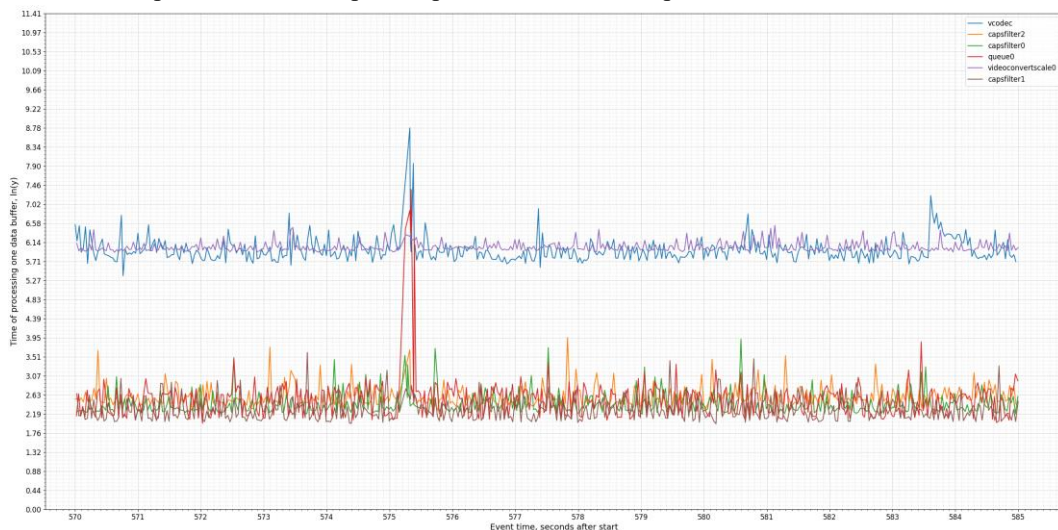


Рис. 9. Фрагмент графика, построенного по данным трассировщика 'proctime' по данным конвейера на основе VP9. По вертикали расположен логарифм времени обработки данных объектами GStreamer конвейера. По горизонтальной оси отмечено время (в секундах) с момента начала трансляции рабочего стола. Между 575 и 576 секундами отмечены два аномальных пика – время работы кодировщика (синим) и время, проведённое данными в очереди 'queue0' (красным).

Замедлению данных в очереди предшествует замедление обработки данных кодировщиком.

Fig. 9. Fragment of a plot based on data from the 'proctime' tracer of a VP9-based pipeline. The vertical axis shows the logarithmic scale of the processing time for data by GStreamer pipeline elements. The horizontal axis shows time (in seconds) since the start of desktop streaming. Two anomalous peaks are visible between 575 and 576 seconds: the encoder processing time (blue) and the time spent by data in the 'queue0' queue (red). The delay in the queue is preceded by a slowdown in the encoder's data processing.

5.3.3 Оценка потенциала AV1 в задаче трансляции удалённых рабочих столов

AV1 (AOMedia Video 1) – это открытый, свободный от выплаты лицензионных платежей правообладателям стандарт видеокодирования, разработанный организацией Alliance for Open Media (AOMedia) [33]. AV1 является преемником стандарта VP9 и ответом сообщества на эффективные, но дорогостоящие стандарты H.264 и H.265. Как сказано выше, стандарт AV1 определяет методы оптимального решения задачи SCC, поэтому является перспективным решением в задаче трансляции удалённых рабочих столов.

В Wormhole для использования AV1 сконструировано 2 GStreamer-конвейера: на основе референсной реализации libaom и на основе альтернативной – SVT – реализации. До внедрения поддержки AV1 в используемом в Wormhole медиасerverе проведён эксперимент по сбору информации о времени кодирования и размеров закодированных видео с помощью GStreamer-конвейеров. В ходе эксперимента сравнивались показатели конвейеров на основе реализаций x264, libaom и SVT-AV1 на 5 сценариях кодирования "офисной" нагрузки – заготовленных тематических видео длительностью до 2 минут. На каждый сценарий проводилось 25 прогонов видео, в ходе которых замерялись время кодирования и размер закодированного файла. Конфигурации конвейеров, используемые в экспериментах, помещены в Приложении 5. Результаты замеров и анализа данных представлены в табл. 1. Результаты не приводятся для libaom реализации, так как его использование в составе черес

GStreamer не актуально в рассматриваемой задаче: соответствующий GStreamer-плагин не предоставляет возможность гибкой настройки этого кодировщика, что ведёт к его низкой производительности в задаче кодирования видео рабочего стола. Конфигурация тестового стенда следующая: процессор Intel i7-3770, 24 Гб ОЗУ, ОС Manjaro Linux x86_64.

Как видно из табл. 1, в ряде сценариев SVT-AV1 показал превосходство над x264 по размеру итогового размера файла. Тем не менее внедрение AV1 в Wormhole ещё не произошло: SVT реализация показала себя как нестабильная – в мажорной версии 3 представленная в Приложении 4 конфигурация не работает из-за неспособности кодировщика работать в режиме константного битрейта. Плагин для работы с libaom в GStreamer требует доработок для возможности более гибкой настройки кодировщика.

Табл. 1. Статистические данные, полученные в ходе эксперимента по замеру размеров файлов и времени кодирования SVT-AV1 и x264.

Table 1. Statistical data obtained during the experiment measuring file sizes and encoding times for SVT-AV1 and x264.

Сценарий	Медианное время кодирования, SVT-AV1, с.	Медианный размер файла, SVT-AV1, МБ	Медианное время кодирования, x264, с.	Медианный размер файла, x264, МБ	Отношение времени кодирования, x264 / SVT-AV1	Отношение размеров файлов, x264 / SVT-AV1
Просмотр веб-страниц	116.183284	73.918007	109.746559	150.387691	0.944599	2.034520
Использование Microsoft Word	32.669588	8.386386	20.687374	12.312542	0.633230	1.468158
Работа в Word параллельно с работой в браузере без переключения вкладок	213.348808	31.453123	210.866439	47.955088	0.988365	1.524653
Процесс написания программного кода – IDE и браузер	118.456091	28.278339	124.338585	45.958537	1.049660	1.625221
Работа в Excel без создания диаграмм	44.977812	12.529974	29.433932	14.782100	0.654410	1.179739

6. Направления дальнейших работ

Дальнейшие работы могут быть разделены на 2 группы: улучшение прослеживаемости задержки и внедрение выделенных оптимизационных механизмов с последующей их оценкой.

В контексте улучшения прослеживаемости следующим этапом является создание инструментария прослеживания задержки от этапа появления кадра в начале конвейера до его отрисовки на клиенте. Также важным является направление по разработке на основе написанного парсера сервиса по предоставлению статистики работы конвейера в режиме реального времени.

В контексте внедрения выделенных оптимизационных механизмов важную роль играет модернизация плагинов GStreamer. Для работы с референсной реализацией AV1 необходимо внести изменения в плагин ‘av1enc’ – это позволит использовать SCC-специфичные настройки при использовании в конвейере подготовки кадров Wormhole. Изменения в плагины для работы с Vulkan и CUDA потенциально могут позволить добиться той же локализации данных, которую удалось получить средствами DirectX 11. Также перспективным направлением в контексте внедрённого инструментария является изучение влияния семантики ‘zero-copy’ на работу конвейера: необходимо рассмотреть влияние ускорения процесса передачи данных без копирований из конвейера. Наконец важными являются работы оптимизацией доставки данных через WebRTC, но для оценки влияния

этого фактора должны использоваться другие инструменты отслеживания поведения системы трансляции.

7. Выводы

В ходе исследования конвейеров обработки видео, включающих этапы кодирования, конвертации и трансляции, выявлен ряд особенностей функционирования различных конвейеров подготовки кадров, важных для разработки и оптимизации систем трансляции рабочих столов. Полученные результаты имеют практическую ценность не только для разрабатываемого в ИСП РАН решения, но и для широкого класса систем видеотрансляции, где критически важны низкие задержки и стабильность видеопотока.

- 1. Составлен и обоснован перечень основных направлений оптимизации видеотрансляции удалённых рабочих столов.**
- 2. Проанализирована работа конвейеров подготовки кадров в составе системы трансляции VDI.** Проведён анализ работы конвейеров, использующих аппаратные (CUDA, DirectX 11) и программные (x264, libvpx) реализации кодировщиков. Установлен ряд фактов о работе анализируемых конвейеров.
 - Конвейеры на основе CUDA демонстрируют значительные задержки, связанные с загрузкой данных в память GPU, что может приводить к периодическим отказам и ухудшению пользовательского опыта.
 - Конвейеры на основе DirectX 11 эффективно устраняют проблему загрузки данных в GPU за счёт использования буферов под управлением DirectX, что повышает стабильность работы системы.
 - Программные реализации, такие как x264, показывают стабильную работу, но имеют высокие задержки на этапе конвертации форматов, в отличие от аппаратных решений.
 - Реализация VP9 в сценарии трансляции FullHD демонстрирует аномалии, связанные с передачей буферов с около нулевыми длительностями, что делает его менее предпочтительным для высоких частот кадров.
 - Реализация AV1 показывает потенциал в уменьшении размера выходного видеопотока, но страдает от нестабильной функциональности (SVT) и ограничений в настройке в контексте GStreamer (libaom).
- 3. Получен ряд практических рекомендаций.**
 - Для минимизации задержек в аппаратных конвейерах предпочтительно использовать DirectX 11 вместо CUDA.
 - Программные реализации, такие как x264, подходят для систем без GPU, но актуальным направлением работ является оптимизация этапа конвертации представлений.
 - VP9, как и x264, является приемлемым вариантом в сценарии трансляции VDI без GPU, но при увеличении частоты кадров возможно ухудшение пользовательского опыта из-за потери кадров.
 - Внедрение AV1 требует доработки плагинов GStreamer для повышения гибкости конфигурации.
 - Разработанные парсеры логов GstShark на Python (с использованием Pandas, NumPy, Matplotlib) доказали свою эффективность для визуализации данных и будут использоваться в дальнейшем.

4. Внедрён инструментарий для анализа задержек конвейера подготовки кадров.

- Разработана и внедрена методология сбора и анализа данных о задержках в GStreamer-конвейерах с использованием инструмента GstShark. Это позволило повысить прозрачность влияния параметров кодирования и выявить основные источники задержек на различных этапах обработки видеоданных.

Проведённое исследование позволило не только выявить ключевые узкие места в процессе формирования кадров видеотрансляции, но и предложить конкретные подходы к их устранению, что способствует повышению качества и стабильности работы системы Wormhole.

8. Приложения

8.1 Приложение 1 – Конвейер на основе CUDA плагинов GStreamer

```
d3d11screencapturesrc ! queue ! cudaupload ! cudaconvertscale ! video/x-raw(memory:CUDAMemory) ! queue !  
nvh264enc name=nvcodec ! video/x-h264,profile=<profile> ! queue ! appsink name=appsink
```

8.2 Приложение 2 – Конвейер на основе DirectX 11 плагинов GStreamer

```
d3d11screencapturesrc ! video/x-raw(memory:D3D11Memory) ! queue ! d3d11convert ! video/x-raw(memory:D3D11Memory) ! queue !  
nvd3d11h264enc name=nvcodec ! video/x-h264,profile=<profile> ! queue ! appsink name=appsink
```

8.3 Приложение 3 – Пример конвейера на основе x264

```
d3d11screencapturesrc ! queue ! videoconvertscale ! video/x-raw ! queue ! x264enc name=nvcodec ! video/x-h264,profile=<profile> ! queue ! appsink name=appsink
```

8.4 Приложение 4 – Конвейеры для работы с VP9 и x264

Листинг А – Конвейер с x264

```
d3d11screencapturesrc ! queue ! videoconvertscale ! video/x-raw ! queue ! Vp9enc name=nvcodec ! video/x-vp9 ! queue ! appsink name=appsink
```

Листинг Б – Конвейер с VP9

```
d3d11screencapturesrc ! queue ! videoconvertscale ! video/x-raw ! queue ! x264enc name=nvcodec ! video/x-h264,profile=<profile> ! queue ! appsink name=appsink
```

8.5 Приложение 5 – Конфигурации конвейеров, используемые для эксперимента по замеру времени и размеру закодированного файла

Листинг А – Конфигурация кодировщика x264 для кодирования видео

```
x264enc threads=8 aud=false b-adapt=false key-int-max=0 sliced-threads=true byte-stream=true  
tune=zerolatency speed-preset=veryfast bitrate=20000 pass=17 ! video/x-h264,stream-format=byte-stream,profile=high
```

Листинг Б – Конфигурация кодировщика SVT-AV1 2.2.1 для кодирования видео

```
svtav1enc target-bitrate=20000 preset=11 parameters-string=':lp=0:rc=2:pred-struct=1:overshoot-pct=10:scm=1:tile-rows=2:tile-columns=2:color-format=1:buf-sz=22:buf-initial-sz=21:buf-optimal-sz=20:' ! video/x-av1,parsed=True,stream-format=obu-stream
```

Листинг В – Конвейер для записи кадров, которые должны быть закодированы в ходе эксперимента

```
gst-launch-1.0 imagesrc name=videosrc remote=0 blocksize=16384 use-damage=0 show-pointer=true ! video/x-raw,framerate=60/1 ! queue leaky=downstream flush-on-eos=true max-size-time=40000000 max-size-buffers=0 max-size-bytes=0 ! identity sync=true ! videoconvertscale ! video/x-raw,format=i420,width=2560,height=1440,framerate=60/1 ! rawvideoparse format=i420 width=2560 height=1440 framerate=60/1 ! filesink location=/mnt/mini-demo/demo_desktop_mini_01.yuv
```

Листинг Г – Конвейер для чтения записанных “сырых” кадров

```
gst-launch-1.0 filesrc location=/mnt/mini-demo/demo_desktop_mini_03.yuv ! queue ! "video/x-raw, format=i420, width=2560, height=1440, framerate=30/1" ! rawvideoparse use-sink-caps=true ! timeoverlay ! autovideosink
```

Список литературы / References

- [1]. World Wide Web Consortium (W3C). WebRTC 1.0: Real-Time Communication Between Browsers. Available at: <http://www.w3.org/TR/webrtc/>, accessed 13.05.2025.
- [2]. Sheikholeslami A., Graffi K. A systematic quality analysis of virtual desktop infrastructure technologies // Euro-Par 2015: Parallel Processing Workshops. Lecture Notes in Computer Science, vol. 9523. Cham: Springer, 2015, pp. 335-346. DOI: 10.1007/978-3-319-27308-2_26.
- [3]. Chang C.-H., Yang C.-T., Lee J.-Y., Lai C.-L., Kuo C.-C. On construction and performance evaluation of a virtual desktop infrastructure with GPU accelerated // IEEE Access. 2020, vol. 8, pp. 170162-170173. DOI: 10.1109/ACCESS.2020.3023924.
- [4]. Li S., Xiao L., Shi C. et al. Boosting performance of virtualized desktop infrastructure with physical GPU and SPICE // Science China Information Sciences. 2020, vol. 63. Art. 179102. DOI: 10.1007/s11432-018-9914-5.
- [5]. Blakley J. R., Haas S., Firoiu V., Iyer M. B., Beveridge D., Narkhede G., George S. A., Eiszler T., Harkes J., Scott J. R., Satyanarayanan M. Deploying edge-based virtual desktop infrastructure. 2023.
- [6]. Simoens P., De Turck F., Dhoedt B., Demeester P. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices // Proceedings of the Australasian Telecommunication Networks and Applications Conference (ATNAC), 2008, pp. 391-396.
- [7]. Alliance for Open Media. AV1 Tool Description. Version 1.1. Available at: https://aomedia.org/docs/AV1_ToolDescription_v11-clean.pdf, accessed: 19.08.2025.
- [8]. ITU-T and ISO/IEC JTC 1. Recommendation ITU-T H.265 (09/2023): High Efficiency Video Coding | ISO/IEC 23008-2:2020, Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High Efficiency Video Coding. Version 9 (Edition 9). Geneva: International Telecommunication Union (ITU) and International Organization for Standardization (ISO), 2023.
- [9]. Xu X., Liu S. Overview of Screen Content Coding in Recently Developed Video Coding Standards. IEEE Transactions on Circuits and Systems for Video Technology, 2022, vol. 32, no. 2, pp. 839-852. DOI: 10.1109/TCSVT.2021.3064210.
- [10]. Benteleb A., Lim M., Akcay M. N., Begem A. C., Hammoudi S., Zimmermann R. Toward One-Second Latency: Evolution of Live Media Streaming. IEEE Communications Surveys & Tutorials, DOI: 10.1109/COMST.2025.3555514.
- [11]. WebRTC Hacks. WebCodecs, WebTransport, and WebRTC: The Future Is Now. Available at: [http://webrthacks.com/webcodecs-webtransport-and-webrtc](http://webrtc hacks.com/webcodecs-webtransport-and-webrtc), accessed: 19.08.2025.
- [12]. Iyengar J., Thomson M. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, Internet Engineering Task Force (IETF), 2021.
- [13]. Internet Engineering Task Force (IETF). Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT). Available at: <http://datatracker.ietf.org/doc/html/rfc8445>, accessed: 03.05.2025.
- [14]. Internet Engineering Task Force (IETF). Session Traversal Utilities for NAT (STUN). Available at: <http://datatracker.ietf.org/doc/html/rfc8489>, accessed: 03.05.2025.
- [15]. Internet Engineering Task Force (IETF). Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Available at: <http://datatracker.ietf.org/doc/html/rfc8656>, accessed: 03.05.2025.
- [16]. Frindell A., Kinnear E., Vasiliev V. WebTransport over HTTP/3. Internet-Draft draft-ietf-webtrans-http3-12, Internet Engineering Task Force (IETF). Available at: <http://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-12>, accessed: 19.08.2025.
- [17]. Google Chrome Developers. WebTransport API Overview. Available at: <http://developer.chrome.com/docs/capabilities/web-apis/webtransport>, accessed: 19.08.2025.
- [18]. Cloudflare, Inc. Radar 2024 Year in Review. Available at: <https://blog.cloudflare.com/radar-2024-year-in-review/>, accessed: 19.08.2025.
- [19]. Li, B., Zou, J. (2019). Screen Content Coding: A Survey. In: Sun, S., Fu, M., Xu, L. (eds) Signal and Information Processing, Networking and Computers. ICSINC 2018. Lecture Notes in Electrical Engineering, vol 550. Springer, Singapore. DOI: 10.1007/978-981-13-7123-3_68.
- [20]. Advanced Micro Devices, Inc. AMF Video Encode API Documentation. Available at: http://github.com/GPUOpen-LibrariesAndSDKs/AMF/blob/master/amf/doc/AMF_Video_Encode_API.md, accessed: 19.08.2025.

- [21]. NVIDIA Corporation. NVENC Video Encoder API Programming Guide. Available at: <http://docs.nvidia.com/video-technologies/video-codec-sdk/12.2/nvenc-video-encoder-api-programming-guide/index.html>, accessed: 19.08.2025.
- [22]. Matrox Graphics Inc. Matrox M264 Family: Multi-Channel 8- and 10-Bit H.264 Codec Cards. Available at: <http://video.matrox.com/en/products/broadcast-media/H.264-codec-cards/m264>, accessed: 19.08.2025.
- [23]. Intel Corporation. Intel Media Driver for VAAPI. Available at: <https://github.com/intel/media-driver>, accessed: 19.08.2025.
- [24]. Microsoft Corporation. Getting Started with DirectX Graphics. Available at: <http://learn.microsoft.com/en-us/windows/win32/getting-started-with-directx-graphics>, accessed: 19.08.2025.
- [25]. Khronos Group. An Introduction to Vulkan Video. Available at: <http://www.khronos.org/blog/an-introduction-to-vulkan-video>, accessed: 19.08.2025.
- [26]. Carlucci G., De Cicco L., Holmer S., Mascolo S. Analysis and design of the google congestion control for web real-time communication (WebRTC). Proceedings of the 7th International Conference on Multimedia Systems, 2016, pp. 1-12.
- [27]. Zhu X., Pan R., Ramalho M.A., Mena S. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. RFC 8698, IETF, 2020. Available at: <https://datatracker.ietf.org/doc/rfc8698/>, accessed: 19.08.2025.
- [28]. Johansson I., Westerlund M. Self-Clocked Rate Adaptation for Multimedia (Internet-Draft). IETF, 2024. Available at: <https://datatracker.ietf.org/doc/draft-johansson-ccwg-rfc8298bis-screamv2/>, accessed: 19.08.2025.
- [29]. GStreamer. queue: Data queue. GStreamer Documentation. Available at: <https://gstreamer.freedesktop.org/documentation/coreelements/queue.html>, accessed 20.08.2025.
- [30]. RidgeRun. GstShark: Performance Analysis & Tracing for GStreamer Developers. Available at: <https://developer.ridgerun.com/wiki/index.php/GstShark>, accessed 20.08.2025.
- [31]. VideoLAN. x264: Free software library and application for encoding video streams into the H.264/MPEG-4 AVC compression format. Available at: <https://www.videolan.org/developers/x264.html>, accessed: 20.08.2025.
- [32]. YouTube Engineering. VP9: Faster, better, buffer-free YouTube. Available at: <https://web.archive.org/web/20150410001205/https://youtube-eng.blogspot.com/2015/04/vp9-faster-better-buffer-free-youtube.html>, publication date: april 2015, archiving date: 10.04.2015, accessed: 20.08.2025.
- [33]. Alliance for Open Media. AV1 Bitstream and Decoding Process Specification / Alliance for Open Media. — Wakefield, MA, USA: Alliance for Open Media, 2019. — Version 1.0.0 with Errata 1. Available at: <https://aomediacodec.github.io/av1-spec/av1-spec.pdf>, accessed: 13.05.2025.

Информация об авторах / Information about authors

Иван Сергеевич ПРОСКУРЯКОВ – программист отдела информационных систем Института системного программирования РАН. Сфера научных интересов: протоколы и стандарты передачи видео и аудио в режиме реального времени, стандарты кодирования видео, разработка методик отладки сложных информационных систем.

Ivan Sergeevich PROSKURIAKOV is a programmer at the Information Systems Department of the Institute of System Programming of the Russian Academy of Sciences. Research interests: protocols and standards for real-time video and audio transmission, video coding standards, development of debugging techniques for complex information systems.

Давид Арменович БАДАЛЯН – научный сотрудник Института системного программирования РАН. Сфера научных интересов: распределенные системы, облачные системы, системы передачи видео.

David Armenovich BADALYAN – research fellow at Institute for System Programming of the RAS. Research interests: distributed systems, cloud systems, video transmission systems.