БЕСШОВНОЕ ВОСПРОИЗВЕДЕНИЕ ЖИВОЙ ПОТОКОВОЙ TCP-ТРАНСЛЯЦИИ МЕТОДОМ ОПЕРЕЖАЮЩЕЙ ЗАГРУЗКИ С НИЗКОЙ ЗАДЕРЖКОЙ

DOI: 10.36724/2072-8735-2021-15-1-11-18

Manuscript received 06 August 2020;
Revised 22 September 2020;
Accepted 28 October 2020

Лобов Игорь Викторович,
НИЦ "Курчатовский институт" ФГБУ ГНЦ РФ – Институт физики высоких энергий, Москва, Россия,
lobov@ihep.ru

Готман Владислав Георгиевич,
НИЦ "Курчатовский институт" ФГБУ ГНЦ РФ – Институт физики высоких энергий, Москва, Россия,
vladislav.gotman@ihep.ru

Ключевые слова: *бесшовная потоковая трансляция, TCP-трансляция, стриминг с низкой задержкой, опережающая загрузка, медиапоток, Ogg, Vorbis, Theora, low latency streaming, TCP-streaming, progressive download*

Рассматривается клиент-серверная технология организации бесшовного воспроизведения в сети интернет живой потоковой TCP-трансляции методом опережающей загрузки. Выявлены составляющие общей задержки воспроизведения медиа-потока на клиенте относительно реального действия. Обсуждена природа и поведение составляющих общей задержки, и проведено исследование поведения наиболее важной из них – размера приемного буфера клиента Lбуф. Обнаружено, что гипотетическое время отображения последнего принятого кадра состоит из двух компонентов: 1) общей нестабильности, имеющей характер хорошо выраженной "полосы нестабильности", и 2) эпизодических "просадок" связанных со спонтанными задержками медиа-потока в канале передачи данных. Итогом таких просадок является увеличение Lбуф, в результате чего увеличивается задержка воспроизведения. Предложен способ уменьшения этой задержки до приемлемого значения (100-300 мс), позволяющего осуществить двустороннюю связь между двумя клиентами – алгоритм плавной коррекции размера приемного буфера клиента. Также прилагаются результаты работы алгоритма плавной коррекции в реализации системы TCP-трансляций презентаций методом опережающей загрузки. Рассматриваемая реализация системы TCP-трансляций представляет собой распределенный программный комплекс, состоящий из одного или нескольких процессов-Кодировщиков медиа-потока, процесса-Ретрансмиттера, клиентской части. Кодировщик медиа-потока преобразует данные от источника медиа-данных (камера, экран компьютера, микрофон) и передаёт их подключившимся клиентам через Ретрансмиттер. В качестве контейнера медиа-потока использовался открытый формат Ogg с кодеками theora и vorbis. Клиентская часть представляет собой набор web-страниц (HTML + JS), где и реализован алгоритм плавной коррекции приёмного буфера. По результатам работы алгоритма сделан вывод о возможности использования системы TCP-трансляций презентаций методом опережающей загрузки для организации связи с низкой задержкой между двумя и более клиентами.

Информация об авторах:

Лобов Игорь Викторович, *к.ф.-м.н., с.н.с., НИЦ "Курчатовский институт" ФГБУ ГНЦ РФ – Институт физики высоких энергий, Москва, Россия. ORCID ID: 0000-0002-1542-1380*

Готман Владислав Георгиевич, *м.н.с., НИЦ "Курчатовский институт" ФГБУ ГНЦ РФ – Институт физики высоких энергий, Москва, Россия*

Для цитирования:
Лобов И.В., Готман В.Г. Бесшовное воспроизведение живой потоковой TCP-трансляции методом опережающей загрузки с низкой задержкой // T-Comm: Телекоммуникации и транспорт. 2021. Том 15. №1. С. 11-18.

For citation:
Lobov I.V., Gotman V.G. (2021) Low latency seamless live TCP-streaming by means of progressive download method . *T-Comm*, vol. 15, no.1, pp. 11-18. (*in Russian*)

Введение

В настоящее время получили широкое распространение клиент-серверные технологии организации живой потоковой трансляции медиа на базе стека HTTP [1] /TCP/IP [2]. Основные технологии в этой области: Apple HLS [3], Adobe HDS [4], Microsoft Smooth Streaming [5], MPEG DASH [6]. Особенностью этих технологий является то, что клиент не создает одну TCP-сессию с сервером для приема медиапотока, а загружает поток отдельными сегментами и воспроизводит их бесшовно. Вследствие такого воспроизведения у пользователя создается впечатление просмотра непрерывного медиапотока. Последовательность URL-адресов для загружаемых с сервера сегментов формирует клиент на основании получаемой от сервера информации (манифеста). Таким образом, воспроизведение медиапотока на клиенте производится типичным для интернета способом – путем «скачивания» отдельных частей контента так, как это делается при загрузке любой HTML-страницы [7].

Существует и другой способ организации живой потоковой трансляции медиа на базе стека HTTP/TCP/IP – путем открытия одной TCP-сессии и непрерывного приема данных медиапотока от сервера. Воспроизведение медиа производится бесшовно уже в силу самой природы TCP-соединения. Такой механизм приема и воспроизведения медиапотока уже давно существует в стандарте HTTP – это механизм опережающей загрузки (progressive download). Применительно к живой потоковой трансляции механизм опережающей загрузки разрабатывался в работах [8, 9, 10, 11], при этом передача медиаданных осуществлялась в контейнере Ogg [12] с кодеками Theora [13]/Vorbis [14]. Настоящая работа является дальнейшим развитием этого подхода и посвящена актуальной проблеме уменьшения задержки воспроизведения между самим действием и отображением этого действия в браузере клиента.

1. Постановка задачи

Для односторонней трансляции задержка между самим действием и отображением этого действия в браузере клиента не имеет большого значения. Например, если транслируется лекция, а клиентами являются слушатели лекции и обратная связь производится в виде чата, то задержка даже в десять секунд может быть вполне допустимой. Однако представляется интересным организовать живую двустороннюю связь в виде двух трансляций в противоположные стороны (от лектора слушателю и наоборот). В этом случае величина задержки удваивается и если, к примеру, величина задержки в одну сторону будет составлять одну секунду, общая задержка в обе стороны станет равной уже двум секундам, что может быть весьма некомфортно для живого общения.

Рассмотрим общую схему возникновения задержки в технологии потоковой трансляции методом опережающей загрузки [8]. В системе трансляций [10] имеются следующие компоненты:

- источник медиаданных (камера, компьютер, микрофон и т.д.);
- сервер для перекодирования и ретрансляции живого потока медиаданных

с источника нескольким клиентам одновременно;

- браузер клиента, который получает и воспроизводит медиаданные с сервера.

При передаче данных по цепочке «источник → сервер → браузер» полная задержка воспроизведения складывается из частных задержек следующих процессов: а) оцифровки медиаданных на источнике; б) передачи данных по каналу связи от источника к серверу; в) промежуточного перекодирования на сервере; г) передачи данных от сервера клиенту; в) буферизации потока на клиенте, последующему декодированию и воспроизведению. Как будет показано в настоящей работе, наиболее существенным элементом задержки воспроизведения является буферизация на клиенте.

Рассмотрим важнейшие свойства буферизации в технологии потоковой трансляции методом опережающей загрузки. При подключении к серверу клиент по своей инициативе делает только один запрос GET на старт загрузки потока. После этого процессом передачи клиенту потока начинает управлять сервер. Клиент не может воспроизводить медиаданные сразу же по мере их поступления, т.к. ему необходимо первоначально накопить некоторое количество медиаданных в приёмном буфере. Типичная начальная длительность накопленных данных составляет приблизительно 200 мс. Эти накопленные данные необходимы для компенсации возможных кратковременных задержек передачи данных. Воспроизведение начнется только тогда, когда число медиаданных в буфере станет достаточным для гарантированного плавного воспроизведения медиапотока. Итак, браузер создает *искусственную дополнительную задержку воспроизведения вследствие создания запаса медиаданных в буфере.*

По мере дальнейшего приема медиапотока задержка воспроизведения может только расти, причем она делает это не плавно, а скачками. Причина этого состоит в следующем. Может случиться так, что из-за уменьшения эффективной пропускной способности канала «сервер-клиент» передача потока клиенту замедлилась или даже вовсе приостановилась. Поскольку воспроизведение потока браузером продолжается, то размер приемного буфера начинает уменьшаться, а неотосланные клиенту данные будут накапливаться в канале связи и на сервере. Если передача потока клиенту возобновится до того момента, как приемный буфер клиента опустеет, то все накопленные на сервере данные потока поступят в буфер клиента, воспроизведение не прервется и задержка воспроизведения не возрастет.

Если же задержка передачи данных клиенту будет не кратковременной, то по исчерпанию приёмного буфера воспроизведение потока клиентом остановится. Поскольку клиент подключён к серверу с помощью TCP-соединения, то медиаданные не могут «потеряться» (мы предполагаем, что TCP-соединение не разрывается) и после возобновления передачи все задержанные медиаданные будут в итоге переданы в приёмный буфер клиента. Воспроизведение потока клиентом возобновится с самых «старых» медиаданных в буфере. Следовательно, появится дополнительная задержка воспроизведения, равная времени, в течение которого воспроизведение останавливалось.

Задачу уменьшения задержки воспроизведения можно решить, например, так: сервер непрерывно следит за объёмом передаваемых клиенту данных и при появлении задержки сервер отбрасывает самые старые данные. В этом случае после возобновления связи сервер отправит клиенту не весь объём накопленных за время простоя данных, а только самые свежие, тем самым опосредованно уменьшив

задержку. Однако при этом воспроизводимая клиентом трансляция уже будет не бесшовной, т.к. пропадет часть предназначенных для клиента медиаданных.

Цель настоящей работы заключалась в поиске программного способа уменьшения накапливающейся задержки трансляции медиаданных до приемлемых величин (не более 300 мс), что позволило бы использовать систему односторонних трансляций конференций для организации двухсторонней связи в *реальном времени*. Главным условием искомого способа уменьшения задержки являлась бесшовность – т.е. плавность воспроизведения медиапотока на клиенте. В следующих пунктах производится анализ свойств задержки воспроизведения и на основе этого анализа предлагается способ решения поставленной проблемы.

2. Анализ свойств задержки воспроизведения

Суммируя сказанное выше, время задержки $T_{зад}$ между самим действием и воспроизведением этого действия в браузере клиента является суммой трех слагаемых:

$$T_{зад} = T_{ист} + T_{пер} + L_{буф} \quad (1)$$

где $T_{ист}$ – задержка, вызванная оцифровкой изображения и его кодированием источником медиаданных (например камерой). По сути, это разность между моментом наступления какого-либо события презентации и моментом пересылки соответствующего кадра источником медиаданных приемнику медиаданных, ближайшему по цепочке. Этот параметр зависит от скорости работы кодеков камер и имеет большой разброс для разных производителей. Выделение этой задержки в самостоятельный параметр $T_{ист}$ вызвано тем, что мы не в состоянии влиять на величину этой задержки.

$T_{пер}$ – задержка, связанная с передачей медиаданных по каналу связи от источника медиаданных к клиенту и (возможным) промежуточным перекодированием этих медиаданных на сервере-ретрансляторе. По сути, это разность между моментом времени начала передачи кадра от источника медиаданных на сервер и моментом времени поступления этого кадра в клиентский буфер браузера.

Выделение этой задержки в самостоятельный параметр оправдано тем, что это та часть общей задержки, которую мы уже можем контролировать. Например, используя более скоростные или менее загруженные каналы связи, выбирая более мощный сервер для перекодирования потока.

$L_{буф}$ – задержка, являющаяся результатом накопления медиаданных в буфере клиента. Эта величина равна общему времени воспроизведения всех медиаданных, находящихся в данный момент буфере клиента.

Поведение во времени трех слагаемых формулы (1) различное. Задержку $T_{ист}$ можно в дальнейшем считать константой, она не зависит от работы канала связи. Две другие могут меняться с течением времени, но характер этого изменения у них различен. Задержки $T_{пер}$ и $L_{буф}$ зависят от качества работы канала связи. Дополнительно к этому задержка $L_{буф}$ определяется логикой программного обеспечения браузера и, как будет показано ниже, имеет весьма своеобразное поведение.

Мы можем программно контролировать $L_{буф}$ следующим образом. В соответствии со стандартом HTML5 [7], тег <video> индицирует следующие параметры буферизации:

buffered.start(0) – «get the start position of a buffered range»
buffered.end(0) – «get the end position of a buffered range»
currentTime – «returns the current position (in seconds) of the audio/video playback»

Это времена начала и конца буфера, а также позиция в буфере текущего момента воспроизведения соответственно. Они измеряются в секундах и отсчитываются от момента начала воспроизведения потока клиентом. Из этих параметров нас интересует только два. Параметр currentTime есть момент отображения текущего отображаемого кадра. Его поведение во времени отражает поведение во времени воспроизведения кадров. Параметр buffered.end(0) есть гипотетическое время отображения последнего принятого кадра в буфере. Его поведение во времени отражает поведение $T_{пер} + T_{ист}$.

Параметр currentTime есть то место буфера, откуда в данный момент данные берутся проигрывателем для воспроизведения. Разность buffered.end(0)–currentTime есть не что иное, как задержка $L_{буф}$ воспроизведения медиапотока из-за буферизации на клиенте (рис. 1):

$$L_{буф} = \text{buffered.end(0)} - \text{currentTime} \quad (2)$$

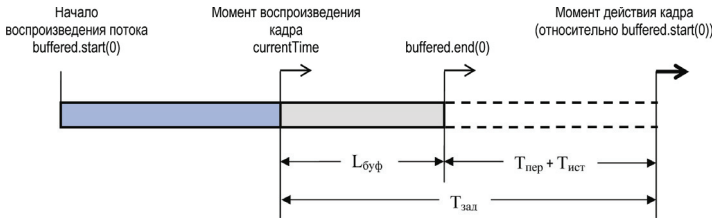

Рис. 1. Компоненты задержки воспроизведения действия в браузере клиента от самого этого действия

Рисунок 1 следует воспринимать в динамике: направленные вправо стрелки иллюстрируют, что величины currentTime и buffered.end(0) увеличиваются *в среднем* с одинаковой скоростью одна секунда за секунду.

Для определения соотношения влияния компонентов $T_{пер}$, $T_{ист}$ и $L_{буф}$ на общую задержку проводился следующий эксперимент. Источник трансляции (видеокамера или программа захвата экрана компьютера) передавал изображение экрана клиентского компьютера, на котором отображалось текущее астрономическое время и изображение результата трансляции с этой же камеры/экрана. Каждый раз регистрировались две величины – общая задержка трансляции $T_{зад}$ и размер клиентского буфера buffered.end(0) – currentTime (как часть этой задержки) при разных параметрах трансляции. Результат исследования приведен на рисунке 2. Изменение $L_{буф}$ достигалось за счет искусственной задержки перекодировки на ретрансляторе без остановки воспроизведения на клиенте, а битрейт потока не превышал эффективной пропускной способности канала передачи данных.

Красными, синими и зелёными треугольниками показаны результаты измерения задержки медиапотоков «видео+аудио», для которых источником являлась IP-камера. Минимальная задержка трансляции от реального времени в этом случае составила 350 мс при буфере клиента равным 110 мс.

Фиолетовыми кружками изображены результаты измерения задержки медиапотока «видео+аудио», в котором ис-

точником являлся захват экрана компьютера, а источником аудиоданных была звуковая карта компьютера. Минимальная задержка трансляции от реального времени в этом случае составила 250 мс при буфере клиента равным 180 мс.

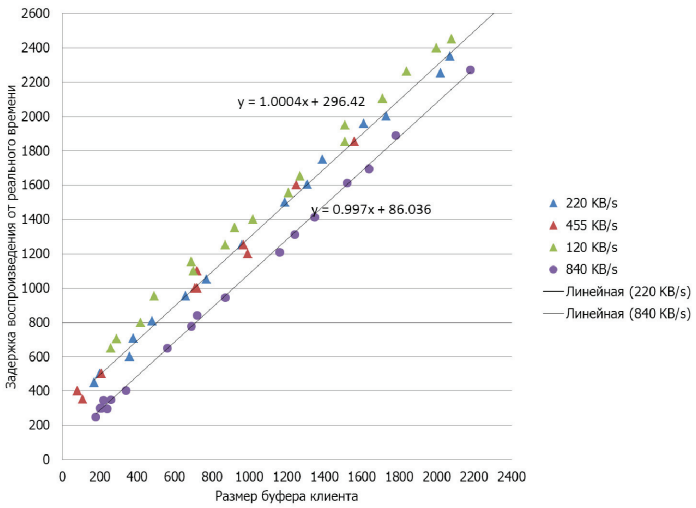

Рис. 2. Зависимость задержки $T_{зад}$ от размера буфера $L_{буф}$ для разных битрейтов потокой трансляции

Из приведенных на рисунке 2 данных следует, что задержка трансляции презентации $T_{зад}$ линейно связана с размером клиентского буфера $L_{буф}$, а суммарная задержка кодирования и передачи $T_{ист} + T_{пер}$ вносит постоянное смещение (в нашем случае 296 мс для IP-камеры и 86 мс для захвата экрана):

$$T_{зад} = L_{буф} + \text{const} \qquad (3)$$

Минимизация $T_{ист} + T_{пер}$ является технологической проблемой, она решается путем выбора более мощных компьютеров, улучшением алгоритма кодирования, увеличением пропускной способности каналов связи. А вот задача минимизации содержимого буфера клиентского проигрывателя ($L_{буф}$) привела к появлению настоящей работы.

3. Поведение зависимости размера клиентского буфера от времени

Рассмотрим, как меняется размер клиентского буфера $L_{буф}$ с течением времени по мере прохождения трансляции и проанализируем поведение каждого из компонентов $L_{буф}$. Как было сказано выше, оба компонента в среднем должны возрастать линейно и равномерно со скоростью одна секунда за секунду.

Параметр currentTime отражает процесс воспроизведения кадров из буфера и в идеале должен представляться линейной зависимостью от времени currentTime = ts, где ts – текущее время. В реальности же эта зависимость имеет некоторый разброс, его ширина определяется алгоритмом воспроизведения кадров используемого проигрывателя/браузера и теоретически не должна превышать длительности воспроизведения одного кадра.

Например, для числа кадров в секунду fps=20 разброс величины currentTime вокруг текущего времени будет приблизительно равен 50 мс.

Типичная зависимость buffered.end(0) от времени показана на рисунке 3. Для удобства, по оси ординат отображается разность buffered.end(0)-ts(timestamp).

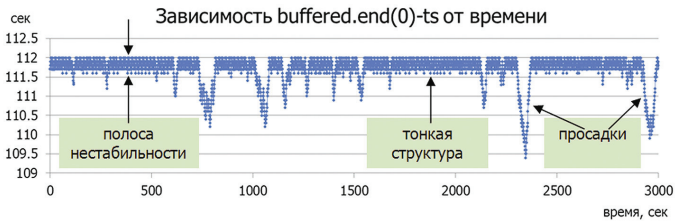

Рис. 3. Типичная зависимость buffered.end(0)-ts от времени

Зависимость buffered.end(0)-ts в целом имеет тонкую дискретную структуру из-за того, что данные в буфер поступают кадрами и правая граница буфера buffered.end(0) увеличивается каждый раз дискретно на величину, кратную длительности воспроизведения одного кадра. На рисунке 3 вертикальный шаг тонкой структуры составляет 100 мс, что соответствует частоте кадров 10 fps.

Форма зависимости buffered.end(0)-ts от времени слагается из двух вкладов – 1) горизонтальной «полосы» фиксированной ширины и 2) эпизодических «просадок». Первый вклад («полоса нестабильности») вызван неравномерностью поступления кадров в буфер клиента. Второй вклад («просадки») связан с качеством работы канала передачи данных.

Рассмотрим вначале свойства полосы нестабильности. На рисунке 4 приведены результаты нескольких экспериментов по измерению ширины полосы нестабильности для разных значений битрейта медиапотока. Мы видим, что ширина зависит от битрейта передаваемого потока: чем меньше битрейт тем она больше.

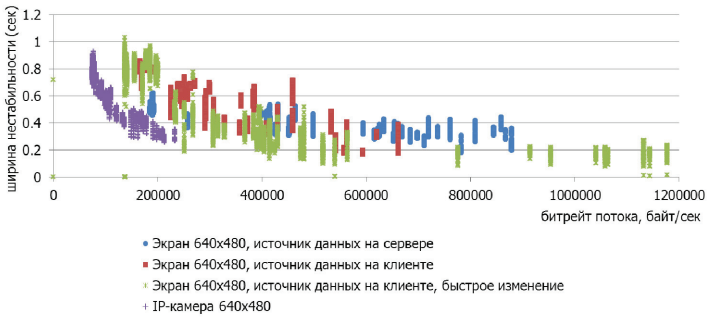

Рис. 4. Зависимость ширины полосы нестабильности и битрейта потока (для всех измерений fps=10)

Поскольку использовался кодек Theora, то кодированный поток содержал только два типа кадров: I-кадры (полное изображение) и P-кадры (изменения изображения по сравнению с последним I-кадром). На рисунке 5 приведены результаты измерения зависимости ширины полосы нестабильности от частоты следования I-кадров.

Нетрудно заметить, что чем чаще идут I-кадры, тем больше будет битрейт потока и наоборот. Поэтому поведение зависимости ширины полосы нестабильности от частоты следования I-кадров напрямую связано с ее зависимостью от битрейта потока.

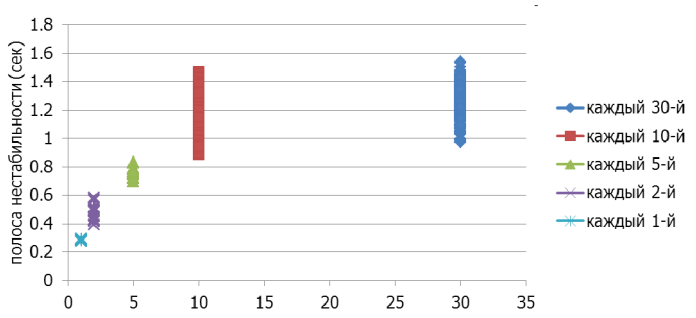

Рис. 5. Зависимость ширины полосы нестабильности от частоты следования I-кадров (для всех измерений fps=10)

В экспериментах на рисунках 4, 5 вычислялась именно ширина горизонтальной составляющей зависимости buffered.end(0)-ts, одиночные «просадки» не учитывались. Для подсчета ширины полосы нестабильности величина buffered.end(0)-ts рассматривалась как случайная с аппроксимацией распределения ее значений нормальной зависимостью. Ширина полосы нестабильности подсчитывалась как 6*σ, где σ – среднеквадратичное отклонение величины buffered.end(0)-ts.

Рассмотрим теперь второй вклад в нестабильность зависимости buffered.end(0)-ts – «просадки». «Просадки» возникают вследствие задержки кадров в канале. Кадры перестают поступать в клиентский буфер и величина buffered.end(0) некоторое время остается постоянной. На рисунке 3 видно, что после каждой «просадки» уровень buffered.end(0)-ts восстанавливался до прежнего. Этот факт свидетельствует о том, что потерь потока не было – все задержанные данные потока в итоге поступили на клиентский компьютер.

Если же кадры будут задержаны в канале настолько, что currentTime сравняется с buffered.end(0), то клиентский буфер опустеет и дальнейшее воспроизведение потока клиентом станет невозможным. На рисунке 6 приведены результаты экспериментов по опустошению клиентского буфера путем создания трех искусственных задержек передачи медиапотока клиенту длительностью 1200, 400 и 200 мс.

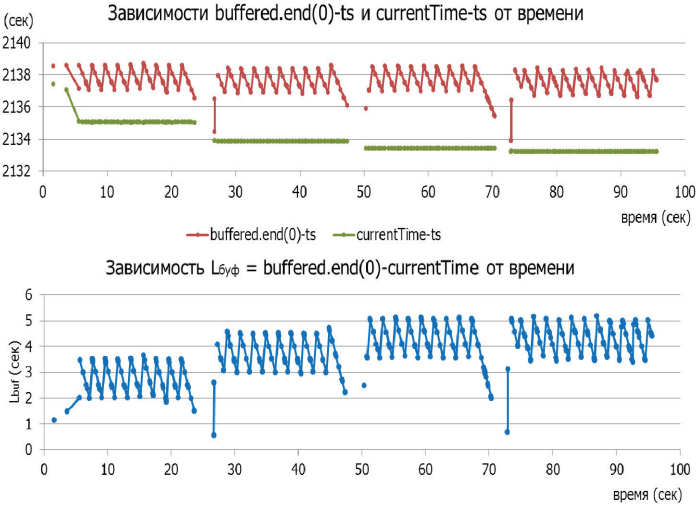

Рис. 6. Зависимости currentTime-ts, buffered.end(0)-ts и $L_{буф}$ от времени для случая медиапотока с битрейтом 20000 байт/сек

По поведению зависимости buffered.end(0)-ts мы видим, что наполнение буфера прерывалось, но потом всегда возобновлялось, причем значения buffered.end(0)-ts после искусственных задержек продолжали оставаться на прежнем уровне. Это говорит о том, что задержанные данные медиапотока не терялись и после возобновления связи приходили все. Что же касается поведения currentTime, то каждая задержка данных в канале связи приводила к увеличению задержки воспроизведения. Факт задержки воспроизведения можно видеть по общему смещению зависимости currentTime–ts вниз. Дело в том, что в случае длительной (больше чем $L_{буф}$) задержки, проигрыватель браузера обнаруживает, что ему не хватает данных для воспроизведения, и вырабатывает событие onwaiting для «подгрузки» в буфер данных, достаточных для возобновления воспроизведения. А после прихода всех задержанных данных размер клиентского буфера увеличивается как раз на величину задержки. Во всех трех ситуациях задержек (рис. 6) было выработано событие onwaiting. В моменты, когда генерировалось событие onwaiting, воспроизведение останавливалось на некоторое время до того момента, когда в приемный буфер поступит количество кадров, достаточное для возобновления воспроизведения потока. Как видно из зависимости currentTime-ts на рисунке 6, итоговая задержка воспроизведения составила величину около 2 секунд.

Возрастание задержки воспроизведения является необратимым процессом, поскольку разность buffered.end(0)-currentTime никогда не уменьшается, а может только увеличиваться. Причина этого заключается в том, что оба края буфера currentTime и buffered.end(0) возрастают во времени в среднем с одинаковой скоростью одна секунда за секунду (рисунок 1), так что currentTime в среднем никогда не догоняет buffered.end(0). Если же currentTime все-таки «догонит» buffered.end(0), то будет выработано событие onwaiting, воспроизведение потока (т.е. currentTime) остановится, проигрыватель дождется прихода задержанных в канале данных, после чего воспроизведение потока (т.е. currentTime) возобновится. Конечным результатом этого будет увеличение $L_{буф}$, т.е. увеличение задержки воспроизведения от реального процесса.

4. Алгоритм коррекции размера клиентского буфера

Кратко суммируем полученные результаты. В процессе проведения трансляции задержка воспроизведения (т.е. размер буфера проигрывателя клиента $L_{буф}$) не является константой, а имеет нестабильный характер. Эта нестабильность не произвольна, а обладает свойством постоянства ширины на отдельных временных участках. Ширина полосы нестабильности зависит от битрейта потока и может доходить до нескольких секунд (в случае потока с малым битрейтом). Длительность участков с постоянной шириной нестабильности тем больше, чем лучше качество канала передачи данных. Ситуации «просадок» – т.е. задержки данных потока в канале, могут сопровождаться увеличением размера $L_{буф}$. Следовательно, при воспроизведении потоковой трансляции задержка воспроизведения может только увеличиваться, и этот процесс увеличения имеет ступенчатый характер (см. рис. 6).

Явление увеличения задержки воспроизведения совершенно неприемлемо, если речь идет не об односторонней трансляции презентации, а о беседе в реальном масштабе времени. Однако свойство постоянной ширины полосы нестабильности $L_{буф}(t)$ наводит на мысль об использовании этого факта для управления задержкой в сторону ее уменьшения. Обнаружив, что размер буфера увеличился, мы будем стараться «опустить» зависимость $L_{буф}(t)$ вниз так, чтобы нижний край полосы нестабильности не касался оси абсцисс, а отстоял от нее на величину около 1-2 размеров одного кадра (50-200 мс). Эта идея является основным принципом описываемого ниже алгоритма.

Учитывая то, что параметр currentTime позволяет себя менять, кажется заманчивым использовать простой способ компенсации возрастания $L_{буф}$ – просто будем соответственно увеличивать currentTime. Этим мы всегда будем удерживать задержку воспроизведения трансляции в требуемом диапазоне. Однако этот простой метод имеет большой минус – воспроизведения перестает быть бесшовным. При каждой коррекции величины currentTime браузер генерирует событие onwaiting, при этом происходит пропадание звука на небольшой период. Это очень неприятный эффект и если такая коррекция производится часто, то просмотр презентации или тем более беседа становится психологически некомфортными.

Ниже предлагается другой метод, который несколько более сложен, но зато является бесшовным. Вкратце его идея такова. Обнаружив возрастание $L_{буф}$, *увеличим скорость воспроизведения* playbackRate до тех пор, пока нижняя граница полосы нестабильности $L_{буф}$ не уменьшится до приемлемого значения $L_{мин}$ (например до 200 мс). После достижения $L_{мин}$ требуемого значения вернем параметру playbackRate прежнее значение 1.0. Далее алгоритм плавной коррекции излагается более подробно.

Пусть R_0 есть обычная скорость воспроизведения (=1.0). Найдем скорость R_1, при которой currentTime догонит buffered.end(0) за время, равное T. С одной стороны, за время T браузер воспроизведет $N_{воспр} = T*R_1$ кадров. С другой стороны, поскольку данные потока поступают непрерывно со средней скоростью R_0, буфер увеличится на $N_{добав} = T*R_0$ кадров. Учитывая, что до начала плавной коррекции в буфере уже имелось $N_{было}$ = (buffered.end(0) – currentTime)*R_0 кадров, то получаем следующее равенство:

$$N_{было} + N_{добав} = N_{воспр} \quad (4)$$

Отсюда получаем значение скорости воспроизведения, необходимое чтобы currentTime «догнал» buffered.end(0) за время T:

$$R_1 = 1 + (\text{buffered.end(0)} - \text{currentTime})/T \quad (5)$$

Суть алгоритма плавной коррекции размера буфера изложена в js-скрипте:

```
<video id="video"  height="640" width="480" controls
<source src =" " type='video/ogg; codecs="theora, vorbis"' />
</video>
<script>
var vid = document.getElementById('video');
function correction ()
{
var T = 1.0;   // одна секунда на коррекцию
if (критерий плавной коррекции)
vid.playbackRate = 1 + (vid.buffered.end(0) - vid.currentTime )/T;
else
vid.playbackRate  = 1;
}
setInterval (function() {correction();}, 100);   // периодически пытаемся корректировать
vid.onwaiting = function()  // защита от истощения буфера
{
if (vid.playbackRate)  vid.playbackRate = 1;
}
</script>
```

В этой программе коррекция вызывается периодически 10 раз в секунду, каждый раз скорость воспроизведения вычисляется исходя из текущих величин currentTime и buffered.end(0). Так как T равен одной секунде, то к моменту очередной коррекции проигрыватель воспроизведет приблизительно 1/10 долю накопленного буфера. Такая величина была выбрана из-за соображений плавности движений в кадре.

В процессе постоянного слежения за размером $L_{буф}$ и подстройкой скорости воспроизведения мы не должны забывать, что $L_{буф}$ изменяется внутри полосы нестабильности. Алгоритм должен постоянно контролировать нижний край полосы нестабильности с тем, чтобы он не опустился ниже $L_{мин}$. В противном случае клиентский буфер будет полностью исчерпан. В этом случае браузер решит, что данных для воспроизведения ему недостаточно, выработается событие onwaiting и возникнет задержка currentTime. Надо динамически менять скорость воспроизведения playbackRate так, чтобы она плавно уменьшалась по мере приближения currentTime к buffered.end(0) и затем «выключить» коррекцию, когда currentTime приблизится к buffered.end(0) на «приемлемое» расстояние $L_{мин}$. Если событие onwaiting все же произошло, необходимо возвращать скорость воспроизведения в единицу, в противном случае проигрыватель начинает менять размер буфера непредсказуемым образом.

Отдельно заслуживает внимания вопрос об определении критерия коррекции. Как было показано выше, если канал связи стабильный, то зависимость $L_{буф}$ (t) достаточно хорошо сформирована в виде горизонтальной полосы нестабильности. Например, на рисунках 3 и 6 зависимость $L_{буф}$ (t) имеет полосы нестабильности с ширинами W = 500 мс и 2 сек соответственно. Такой характер зависимости размера буфера позволяет провести коррекцию и попытаться «опустить» полосу пониже. Поэтому примем за критерий проведения коррекции следующее утверждение:

критерий плавной коррекции = ($L_{буф}$ (t) имеет вид явно выраженной горизонтальной полосы шириной W) && ($L_{буф}(t) - W/2 > L_{min}$).

5. Проверка алгоритма плавной коррекции

Предложенный выше алгоритм тестировался на платформе Windows 7/10 в браузерах Google Chrome, Mozilla Firefox, Opera и Yandex. Трансляция медиапотока включала в себя видео и аудио.

На рисунке 7 представлен пример работы алгоритма коррекции в процессе воспроизведения презентации.

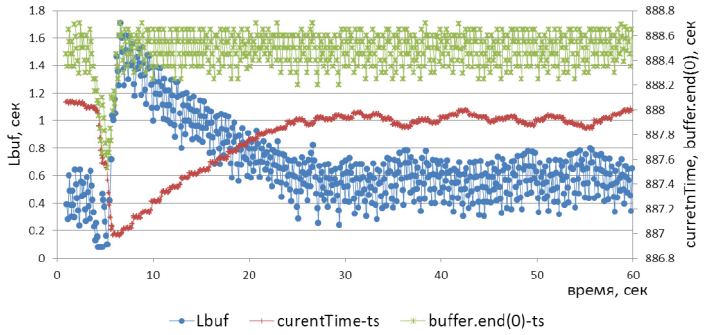

Рис. 7. Пример работы алгоритма плавной коррекции

Вначале трансляции имела задержку (см. синий график L_{buf}) в диапазоне 0.2-0.6 сек. В момент t=3.5 произошла «просадка», прием кадров клиентом прекратился и L_{buf} начал уменьшаться. В момент времени t=4.2 нижний край полосы нестабильности $L_{buf}(t)$ коснулся оси абсцисс, приемный буфер опустел и воспроизведение презентации прекратилось. При этом браузер выработал событие onwaiting и перешел в ожидание прихода медиаданных трансляции. После того, как приемный буфер браузера набрал достаточное количество кадров, воспроизведение трансляции продолжилось. При этом размер приемного буфера возрос до 1.3-1.7 сек, и у трансляции появилась соответствующая задержка. Затем включился алгоритм коррекции скорости воспроизведения. Коррекция проводилась каждую секунду, постепенно доведя задержку воспроизведения до первоначального интервала 0.3-0.7 сек.

На рисунке 8 приведен результат длительного тестирования работы механизма плавной коррекции.

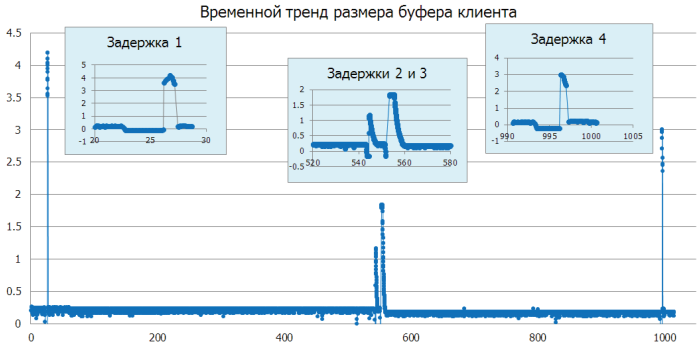

Рис. 8. Пример работы алгоритма плавной коррекции в течение 1000 сек

Отдельно изображены увеличенные участки областей просадок.

Заключение

В работе исследованы свойства задержки воспроизведения браузером клиента во время потоковой трансляции в реальном времени методом опережающей загрузки [8]. Предложено решение задачи уменьшения задержки воспроизведения до приемлемой величины, при которой возможна организация двусторонней связи в режиме online.

Выяснено, что основной вклад в общую задержку воспроизведения и в ее поведение от времени вносит накопление кадров в приемном буфере проигрывателя браузера. Проведено исследование трендов гипотетического времени отображения последнего принятого кадра buffered.end(0) относительно текущего времени.

В результате выявлены два компонента тренда:

1) общая нестабильность, имеющая характер хорошо выраженной «полосы нестабильности».

2) эпизодические «просадки», связанные с задержкой данных потока в канале передачи данных. Итогом просадок является увеличение $L_{буф}$ приемного буфера, в результате чего увеличивается и задержка воспроизведения.

По результатам проведенного исследования поведения $L_{буф}(t)$ разработан алгоритм плавной коррекции размера буфера клиента. Алгоритм заключается в слежении за поведением $L_{буф}(t)$ и постепенном его уменьшении за счет увеличения скорости воспроизведения, плавность воспроизведения медиа-потока при этом не утрачивается. Итоги тестирования работы алгоритма плавной коррекции позволяют сделать вывод о принципиальной возможности использования технологии организации живой потоковой трансляции методом опережающей загрузки [8] для организации двусторонней связи в реальном времени между двумя или более клиентами.

Литература

1. Hypertext Transfer Protocol – HTTP/1.1, Request for Comments: 2616, Network Working Group, 06.1999.
2. A TCP/IP Tutorial, Request for Comments: 1180, Network Working Group, 01.1991.
3. *Pantos R., May W.* HTTP Live Streaming. draft-pantos-http-live-streaming-18. Apple Inc. 2015. 49 p.
4. HTTP Dynamic Streaming Specification Version 3.0 FINAL. Adobe Systems Incorporated. 2013. 31 p.
5. Smooth Streaming Protocol. [MS-SSTR] – v20160714. Microsoft Corporation. 2016. 64 p.
6. ISO/IEC 23009-1. Information technology – Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. Second edition. 2014. 144 p.
7. HTML Living Standard, Web Hypertext Application Technology Working Group, 24.07.2020.
8. *Лобов И.В., Готман В.Г.* Трансляция мультимедиа в реальном времени над протоколом HTTP методом опережающей загрузки // Технологии и средства связи. 2016. №5. С. 36-40.
9. *Лобов И.В., Готман В.Г.* Использование контейнера Ogg для организации потоковой трансляции в реальном времени над протоколом HTTP методом опережающей загрузки // Информационные технологии. 2018. №2. С. 87-96.
10. *Лобов И.В., Готман В.Г.* Система трансляций презентаций в реальном времени над протоколом HTTP // Известия Института инженерной физики (ИИФ). 2018. №3. С. 60-66.
11. *Лобов И.В., Готман В.Г.* Адаптивная бесшовная потоковая трансляция в реальном времени над протоколом HTTP методом опережающей загрузки // Информационные технологии. 2020. №3. С. 177-184.
12. *Pfeiffer S.* The Ogg Encapsulation Format Version 0. Request for Comments: 3533, 2003. 15 p.
13. Theora Specification. Xiph.Org Foundation, 2011. 196 p.
14. Vorbis I specification. Xiph.Org Foundation, 2015. 74 p.

LOW LATENCY SEAMLESS LIVE TCP-STREAMING BY MEANS OF PROGRESSIVE DOWNLOAD METHOD

Igor V. Lobov, *Institute for High Energy Physics, National Research Center "Kurchatov Institute", Moscow, Russia,* ***lobov@ihep.ru***
Vladislav G. Gotman, *Institute for High Energy Physics, National Research Center "Kurchatov Institute", Moscow, Russia,* ***vladislav.gotman@ihep.ru***

Abstract
The article discusses the client-server technology for organizing seamless live TCP-streaming with the use of progressive download method. Main factors which have a major impact on the overall playback latency have been identified. A study on the most important factor - the size of the client's receiving buffer Lbuf - was carried out. It was found that the hypothetical display time of the last frame taken consists of two components: 1) the general instability, which has the character of a pronounced "band of instability", and 2) episodic "drawdowns" associated with spontaneous media flow retentions in data transmission channel.The result of these drawdowns is an increase in Lbuf, resulting in increased playback delay. The article proposes a method to reduce the client's receive buffer latency downto acceptable level (100-300 ms), allowing two-way communication between clients - an algorithm for smooth correction of client receiving buffer size. The article includes the results of the smooth correction algorithm in the implementation of TCP-Streaming by means of progressive downloading method. The implementation is a distributed software package consisting of one or more encoding processes, retransmitting process, client-side software. The media-flow Encoders converts data from the media-data sources (camera, computer screen, microphone) and transmits them to the clients via Retransmitter. The client-side software is a set of web pages (HTML + JS) where the receive buffer smooth correction algorithm is implemented. It was concluded that live TCP-streaming by means of progressive download method is quite suited to organize the communication with low latency between clients.

Keywords: *seamless live TCP streaming, low latency TCP streaming, progressive download, Ogg streaming, Vorbis, Theora.*

References

1. Hypertext Transfer Protocol - HTTP/1.1, Request for Comments: 2616, Network Working Group, 06.1999.
2. A TCP/IP Tutorial, Request for Comments: 1180, Network Working Group, 01.1991.
3. Pantos R., May W. (2015). HTTP Live Streaming. draft-pantos-http-live-streaming-18. Apple Inc. 49 p.
4. HTTP Dynamic Streaming Specification Version 3.0 FINAL. Adobe Systems Incorporated. 2013. 31 p.
5. Smooth Streaming Protocol. [MS-SSTR] - v20160714. Microsoft Corporation. 2016. 64 p.
6. ISO/IEC 23009-1. Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats. Second edition. 2014. 144 p.
7. HTML Living Standard, Web Hypertext Application Technology Working Group, 24.07.2020.
8. Lobov I., Gotman V. (2016). Translyatsiya multimedia v realnom vremeni nad protokolom HTTP metodom operezhayuschey zagruzki [Real time media streaming over HTTP using progressive download method]. *Tehnologii i sredstva svyazi.* No. 5. P. 36-40.
9. Lobov I., Gotman V. (2018). Ispol'zovanie kontejnera Ogg dlya organizatsii potokovoj translyatsii v real'nom vremeni nad protokolom HTTP metodom operezhayushhej zagruzki [The Utilization of Ogg Multimedia Container Format for Live Streaming over HTTP Using Progressive Download Method]. *Informatsionnye tekhnologii.* No. 2. P. 87-96.
10. Lobov I., Gotman V. (2018). Sistema translyacij prezentacij v real'nom vremeni nad protokolom HTTP [Live streaming system over http]. *Izvestiya Instituta inzhenernoj fiziki (IIF).* No. 3. P. 60-66.
11. Lobov I., Gotman V. (2020). Adaptivnaya besshovnaya potokovaya translyatsiya v realnom vremeni nad protokolom HTTP metodom operezhayushchey zagruzki [Adaptive bitrate seamless live streaming over HTTP by progressive download method]. *Informatsionnye tekhnologii.* No. 3. P. 177-184.
12. Pfeiffer S. (2003). The Ogg Encapsulation Format Version 0. Request for Comments: 3533. 15 p.
13. Theora Specification. Xiph.Org Foundation, 2011, 196 p.
14. Vorbis I specification. Xiph.Org Foundation, 2015, 74 p.

Information about authors:
Igor V. Lobov, *Senior researcher, Institute for High Energy Physics, National Research Center "Kurchatov Institute", Moscow, Russia*
Vladislav G. Gotman, *Junior researcher, Institute for High Energy Physics, National Research Center "Kurchatov Institute", Moscow, Russia*