УДК 004.724

А.И. Савельев, М.В. Прищепа

Архитектура обмена данными без потерь в пиринговом веб-приложении видеоконференц-связи

Поднимается проблема соединения клиентов и передачи аудио- и видеопотоков данных в пиринговых веб-приложениях видеоконференц-связи. При взаимодействии нескольких клиентских и серверной частей приложения видеоконференц-связи на основе протокола WebRTC возможна частичная или полная потеря «сигнальных» данных, препятствующая соединению клиентов. Предложенная архитектура передачи и хранения «сигнальных» данных на клиенте и сервере обеспечивает буферизацию и последующую обработку «сигнальных» данных, исключая их потерю и поддерживая взаимодействие между группами клиентов.

Ключевые слова: пиринговые соединения, peer-to-peer протоколы, видеоконференц-связь, веб-приложения, передача мультимедийных данных.

Анализ способов передачи данных в пиринговых веб-приложениях видеоконференц-связи. В настоящее время, несмотря на большие темпы развития интернет-технологий, существует множество проблем, связанных с потоковой передачей видео- и аудиоинформации. Во многом эти проблемы возникают по причине недостаточной пропускной способности каналов связи. Поскольку системы видеосвязи требуют больших сетевых ресурсов даже для передачи видео между двумя участниками, то поддержка многопользовательских видеоконференций является крайне затруднительной задачей.

Сейчас сотни тысяч пользователей могут одновременно использовать пиринговые (peer-to-peer (P2P)) сети [1, 2]. Обычной практикой в P2P-системах потокового видео на сегодняшний день является объединение участников, просматривающих один и тот же контент в «рой» (swarm), и перераспределение частей видеоконтента исключительно между участниками этого роя. Для такой канально-изолированной структуры P2P-видеосистем характерны задержки переключения каналов и отставание воспроизведения контента, связанные с оттоком абонентов канала и дисбалансом числа принимающих и ретранслирующих узлов. В целом глобальные P2P-сети с канально-изолированной структурой в настоящее время имеют серьезные проблемы с производительностью, которые будут становиться все более серьезными с ростом числа пользователей каналов.

Кроме того, в сетях потоковых систем P2P существуют проблемы, связанные с задержками каналов коммутации и низкой производительностью систем для каналов с небольшим числом участников. В работе [3] для решения этих проблем предлагается многоканальная потоковая P2P платформа View-Upload Decoupling (VUD), разделяющая выполняемые пользователем операции загрузки и просмотра данных, а также совместного использования ресурсов перекрестных каналов. За счет этого, обеспечиваются стабильность для многоканальных систем и качественное распределение ресурсов сети. Кроме того, для повышения производительности передачи данных учитывается географическое местоположение клиентов сети, и связь устанавливается между наиболее близко расположенными пользователями, имеющих необходимые данные.

В работе [4] представлена архитектура многопользовательской распределенной пиринговой системы видеоконференций, в которой предполагается, что каждый участник может создавать, отправлять и получать видеосигнал в любой момент времени, но во время видеоконференции участник передает либо видеосигнал, создаваемый его устройством, либо ретранслирует принимаемый видеосигнал другому участнику. Таким образом, участник, который посылает свой собственный видеосигнал, не может действовать в качестве промежуточного узла для прохождения видеосигнала от другого партнера. Данная распределенная архитектура может быть использована для реализации видеоконференций системы P2P, чтобы позволить каждому участнику видеть другого участника. Архитектура данной системы видеоконференц-связи P2P определяется на основе так называемой «цепи». Конфигурация цепи во время сеанса связи формируется на основе конфигурационных (служебных) сообщений, рассылаемых между приложениями видеоконференц-связи участников.

Для уменьшения объема данных, передаваемых в ходе видеоконференц-связи, в работе [5] используется автоматический способ определения текущего говорящего, и его потокам мультимедийных данных выставляется наибольший приоритет при передаче остальным участникам. Идентификация, диаризация дикторов, а также другие методы обработки речи и анализа лица человека широко применяются для автоматизации телекоммуникационных сервисов [6–8].

В работе [9] предложена «бессерверная» архитектура распределительного узла для передачи видео высокой четкости в многопользовательской системе видеоконференц-связи. В основе архитектуры заложен многопользовательский управляющий блок, интегрированный в клиентское приложение, который служит для установления множественных канальных соединений, кодирует и декодирует клиентские видеопотоки и распределяет видео другим участникам сеанса связи.

Производительность способов передачи потокового видео в P2P-сетях зависит также от конфигурации самой сети, ее топологии, неоднородности сетевых ресурсов абонентов, пропускной способности их каналов связи [10]. В отличие от совместной загрузки файлов, где малая пропускная способность канала просто приводит к медленной загрузке, при передаче потокового видео низкая скорость подключения становится настоящей проблемой. Также остаются актуальными вопросы компрессии видеосигнала, позволяющей снизить загруженность канала без значительного увеличения нагрузки на устройство конечного пользователя при кодировании/декодировании сигнала.

В работах [11, 12] представлены полученные авторами результаты предварительных исследований по сокращению передаваемого объема данных между пользователями в приложениях видеоконференц-связи. Описаны алгоритмы и программные средства, позволившие провести оптимизацию разработанного кроссплатформенного приложения видеоконференц-связи на этапах создания и удаления аудио- и видеопотоков данных, их передачи от сервера к клиенту и обратно, создания цепочек потоков и их поиска на сервере. В ходе исследований были выполнены упрощение клиентской части приложения и реорганизация структуры серверной части приложения.

На основе предложенных принципов оптимизации способов обмена мультимедийными данными в приложении видеоконференц-связи была разработана новая пиринговая архитектура прямой передачи аудио- и видеоданных между клиентскими частями, представленная в следующем разделе. Затем описан процесс формирования клиентских веб-страниц и установления связи с сервером по протоколу WebSocket. В последнем разделе обсуждаются разработанные алгоритмы установления соединений между клиентами с использованием протокола WebRTC.

Основные элементы разработанной пиринговой архитектуры взаимодействия модулей веб-приложения видеоконференц-связи. Разработанная архитектура, представленная на рис. 1, позволяет предотвратить потерю «сигнальных» данных при соединении трех и более участников видеоконференц-связи. Основными структурными элементами архитектуры являются: *1* – клиентская часть приложения; *2* – серверная часть приложения; *3* – блок протоколов передачи данных. Клиентская часть подразделяется на две независимые составляющие – устройство пользователя и веб-страница. Устройство пользователя в приложении необходимо для создания аудио- и видеопотоков с камеры и микрофона, подключенных или являющихся частью устройства.

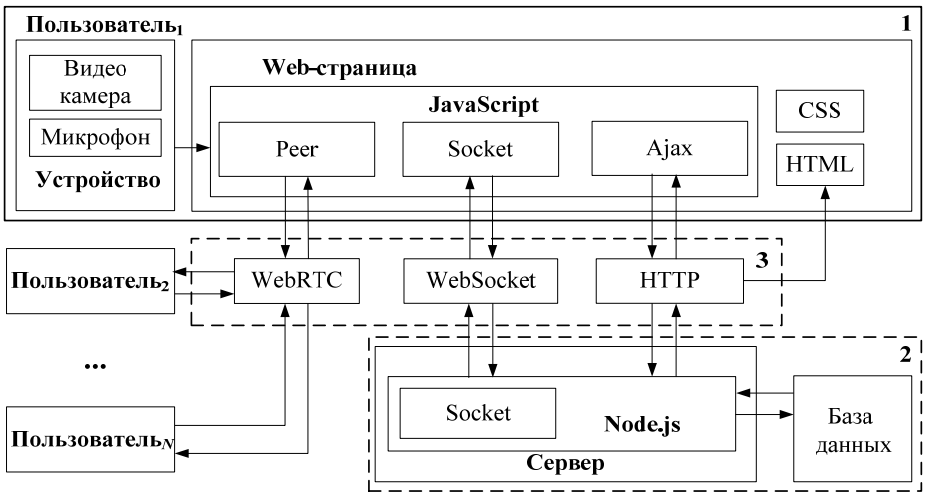

Рис. 1. Разработанная архитектура обмена данными в приложении видеоконференц-связи

Веб-страница клиентской части приложения состоит из классов, написанных на языке программирования JavaScript, необходимых для создания соединений с сервером и другими клиентами с помощью различных протоколов и обработки данных. Средства CSS и HTML служат для построения графического интерфейса, отображения данных и управления клиентской частью приложения. Средства JavaScript, использующиеся на веб-странице видеочата, включают в себя три различных типа инструкций, позволяющих организовать передачу данных по трем протоколам: WebRTC, WebSocket и HTTP. Также средства JavaScript служат для захвата и обработки потоков данных с микрофона и видеокамеры.

Следующим основным элементом архитектуры приложения является серверная часть, она выполняет несколько различных функций: формирование клиентской части приложения; регистрация клиента; авторизация клиента; обмен «сигнальными» данными между клиентами; создание комнат чата и работу с базой данных. Сам сервер работает на платформе Node.js, транслирующей JavaScript в машинный код, и имеет такую же асинхронную архитектуру, как и клиентская часть, разработанная средствами языка программирования JavaScript. База данных MongoDB, расположенная в серверной части приложения, имеет NoSQL-архитектуру, которая подходит для упрощения реализации серверной части. Взаимодействие с базой данных MongoDB происходит с помощью JavaScript и специальной библиотеки драйвера, предназначенной для этой базы данных.

Третий элемент архитектуры, представленной на рис. 1, состоит из протоколов – HTTP, WebSocket и WebRTC. Эти протоколы обеспечивают обмен данными на различных этапах работы приложения, с их помощью осуществляется создание соединения клиентских частей по протоколу WebRTC для передачи потоковых аудио- и видеоданных между ними. Существуют проблемы, возникающие при создании соединения, исходящие из асинхронной архитектуры приложения и протокола WebRTC, не предусматривающего стандартную реализацию соединения множества клиентов. Также необходимо отметить сложность процедуры установления связи между клиентами по протоколу WebRTC, требующую обмена «сигнальными» данными между ними и особого внимания при создании соединения.

В данной работе описано решение вышерассмотренной проблемы потери сигнальных данных при множественном соединении клиентов видеоконференц-связи с помощью внедрения новых алгоритмов взаимодействия клиентских и серверной частей и использования различных протоколов для организации обмена данными. Такая архитектура позволяет создать полноценное пиринговое приложение видеоконференц-связи, которое может работать в режиме группового чата. В следующем разделе рассмотрены основные протоколы и программные средства, использующиеся для создания клиентской веб-страницы и ее функционирования при проведении видеоконференции.

Процесс формирования клиентских веб-страниц и установления связи с сервером по протоколу WebSocket. Чтобы ясно представлять проблему потери «сигнальных» и предложенные в данном исследовании программно-алгоритмические решения данных, вначале рассмотрим основные этапы функционирования клиентской и серверной частей разработанного приложения видеоконференц-связи.

Клиентская часть приложения начинает работу с формирования веб-страницы регистрации или авторизации, позволяющих взаимодействовать клиенту с сервером посредством отправления или получения данных по HTTP-протоколу.

HTTP – это протокол прикладного уровня для передачи произвольных данных. Данный протокол используется в приложении для передачи клиенту графического интерфейса в виде html и css данных, логики клиентской части приложения, написанной на языке программирования JavaScript, а также для обмена данными клиента с сервером, при регистрации и авторизации клиента с помощью технологии Ajax, позволяющей обмениваться данными с сервером по протоколу HTTP без перезагрузки веб-страницы.

Авторизация клиента позволяет пользователю получить доступ к персональным данным и странице видеоконференц-связи. Для авторизации пользователь вводит данные в формы: логин и пароль. Затем происходит сбор данных из форм и отправка этих данных на сервер с помощью технологии Ajax. Далее сервер обрабатывает полученные данные: проверяет на соответствие определенному набору символов и на превышение максимального размера данных в запросе, выполняет поиск пары логин–пароль по базе данных. При успешном завершении всех операций сервер сформирует страницу видеоконференц-связи с пользовательскими данными и отправит ее по протоколу HTTP-клиенту. В случае несоответствия данных определенным требованиям или возникновения

ошибки сервер отправит клиенту информационное сообщение, способствующее устранению возникшей ситуации, используя протокол HTTP.

Регистрация клиента, так же как и авторизация, предполагает заполнение форм данными и их отправку на сервер по протоколу HTTP. Далее сервер обрабатывает данные, присланные клиентом. В случае положительного результата обработки сервер сохранит все полученные данные в базе данных, проведет в автоматическом режиме регистрацию клиента, сформирует и отправит ответ на запрос клиента в виде страницы видеоконференц-связи с пользовательскими данными. В случае ошибочных данных сервер вернет уведомление об ошибке клиенту по протоколу HTTP.

Таким образом, приложение использует HTTP-протокол для надежной передачи HTML, CSS и javascript данных между сервером и клиентом. Преимущество использования этого протокола в том, что он специально предназначен для передачи веб-страниц и их логики, а также хорошо поддерживается всеми существующими браузерами. Протокол HTTP имеет набор стандартных команд, среди которых есть две основные команды: «GET» и «POST», входящие в состав Ajax технологии, соответственно позволяющие осуществлять запросы на выдачу страниц к серверу и запрос на обмен различным типом данных между сервером и клиентом при авторизации или регистрации клиента.

После получения клиентом веб-страницы видеоконференц-связи средствами JavaScript, создается сокет на клиенте, который устанавливает соединение с сервером с помощью протокола WebSocket. WebSocket – это протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. Протокол WebSocket открывает сокеты на клиенте и сервере, позволяющие обмениваться любыми типами данных. В случае успешного соединения по протоколу WebSocket сервер создаст у себя сокет с данными клиента и начнет его авторизацию: попытается получить http cookie данные клиента, которые хранят необходимую информацию для авторизации, произведет распаковку cookie данных, попытается загрузить из базы данных сессию, соответствующую данным из http cookie, по загруженной сессии определит пользователя, принадлежащего сессии, привяжет пользовательские данные к сокету, создаст уникальный номер для сокета, сгенерирует и отправит событие «connection» внутри сервера. Если одно из действий при авторизации сокета сгенерирует ошибку, то сокет на сервере будет автоматически отключен и удален, а клиентский сокет получит сообщение о разрыве соединения. Событие «connection», возникающее на сервере, привязывает к сокету, созданному на сервере – «слушателей» событий, посылаемых сокетом клиента. Сокет клиента еще при его создании формирует набор «слушателей» событий, посылаемых сокетом сервера. Таким образом, устанавливается связь между клиентом и сервером через протокол WebSocket, которая позволяет им быстро обмениваться сообщениями различного типа, не требующими их идентификации, так как для каждого вида сообщений существует отдельный «слушатель».

Данный протокол при построении архитектуры обмена данными позволяет достичь высокой скорости обмена информации и уменьшение нагрузки на клиента и сервер за счет отсутствия затрат на идентификацию данных. В разработанном приложении видеоконференц-связи протокол WebSocket играет важную роль – он занимается передачей «сигнальных» данных браузеров клиентов, которые позволяют создать соединение по протоколу WebRTC. Таким образом, данный протокол является основой для создания соединения по протоколу WebRTC и упрощает процесс передачи данных, необходимых для пирингового соединения.

Алгоритмы установления соединений между клиентами по протоколу WebRTC. После установления связи с сервером по протоколу WebSocket для совершения видеозвонков пользователю необходимо включить видеокамеру и микрофон и дать к ним доступ браузеру. Браузер, получивший доступ к камере и микрофону пользователя, используя средства JavaScript, сформирует медиапотоки данных с подключенных устройств. Полученные аудио- и видеопотоки можно будет передать по протоколу WebRTC между браузерами клиентов напрямую. WebRTC – интернет-протокол, предназначенный для организации передачи потоковых данных между браузерами или другими поддерживающими его приложениями по технологии точка–точка. Для соединения двух клиентов по протоколу WebRTC необходим следующий набор JavaScript инструкций: создание пира для каждого из клиентов, назначение одного из клиентов как «вызывающего», назначение другого клиента как «отвечающего», формирование «сигнальных» данных, обмен «сигнальными» данными, завершение установки соединения.

Для передачи «сигнальных» данных между клиентами используется сервер и протокол WebSocket. Ранее созданные сокеты в клиентской части приложения позволяют передавать «сигналь-

ные» данные по определенным каналам серверу, сервер в свою очередь транслирует эти данные другим клиентам, для которых они предназначены. Для соединения клиентов по протоколу WebRTC необходимо три типа данных: «call offer», «call answer» и «candidate». «Call offer» служит для инициализации сессии WebRTC, он формируется на одном из клиентов и с помощью WebSocket-протокола пересылается серверу, сервер в свою очередь передает данное сообщение «отвечаещему» клиенту. «Call offer» имеет формат SDP (Session Description Protocol). Сообщение SDP, передаваемое от одного узла другому, может указывать: адреса места назначения, служащие для медиа-потоков мультикастинг-адресами, номера UDP портов для отправителя и получателя, медиа-форматы (например, кодеки), применяющиеся во время сессии, время старта и остановки. Сообщение SDP используется для широковещательных сессий, например телевизионных, радиопрограмм или видеоконференций. Клиент, получивший «call offer», сформирует и отправит ответ по протоколу WebSocket в виде данных «call answer», которые также имеют формат SDP. Как только клиент, отправлявший «call offer», получит SDP-сообщение типа «call answer», между клиентами начнется обмен данными типа «candidate» по протоколу WebSocket. Данные типа «candidate» имеют формат – ICE (Interactive Connectivity Establishment) Candidate. Создание интерактивного подключения (ICE) – это метод, используемый в компьютерных сетях, включающий в себя передачу сетевых адресов (NATs) в таких интернет-приложениях, как ip-телефония (VoIP), приложениях пиринговой передачи данных (peer-to-peer communications), видеоприложениях, системах мгновенного обмена сообщениями (instant messaging) и других интерактивных медиа-приложениях. Данные типа «candidate» используются для соединения клиентов, устанавливая путь между ними, по которому будут передаваться медиа-потоки. При успешном обмене данными типа «candidate» каждый из клиентов откроет канал для передачи различного типа данных по протоколу WebRTC, в том числе аудио- и видеопотоков.

Протокол WebRTC имеет особенности, которые создают сложности при соединении пользователей: для создания соединения между клиентами требуется выполнить операцию «handshake», которая состоит из обмена различным типом «сигнальных» данных между браузерами, но одновременно клиент может устанавливать только одно соединение по протоколу WebRTC.

Данная специфика протокола влечет за собой ряд проблем, при создании полноценной видеоконференц-связи. Проблемы возникают из-за асинхронной архитектуры приложения в ситуациях соединения одного клиента со множеством, множества с одним, множества со множеством. Такие ситуации приводят к нарушению алгоритма соединения – полной или частичной потере данных, требующихся для установления связи между клиентами. Чтобы решить сложившуюся проблему, было предложено несколько дополнительных подходов к построению архитектуры обмена данными в приложении: буферизация «сигнальных» данных протокола WebRTC на клиенте и сервере, объединение сокетов, соединяемых клиентов в «комнату» на сервере. «Комната» – это массив, находящийся на сервере, состоящий из нескольких сокетов, позволяющий обмениваться данными только с сокетами, находящимися в данном массиве. Таким образом, «комнаты» изолируют группы сокетов друг от друга, способствуя распространению данных только внутри определенных групп. Такие подходы помогают исключить потерю данных, позволяют создать все необходимые соединения между клиентами и управлять процессами соединения клиентов на различных этапах работы приложения. Далее рассмотрим алгоритмы, основанные на разработанных подходах, позволяющие создать соединение по протоколу WebRTC, контролировать обработку «сигнальных» данных, осуществлять буферизацию «сигнальных» данных и группировать сокеты клиентов в отдельные «комнаты».

Алгоритм, представленный на рис. 2, описывает этап соединения клиентов до формирования «сигнальных» данных. Сначала «вызывающий» клиент подает запрос на сервер для соединения с «отвечающим» клиентом по протоколу WebSocket. Далее сервер производит поиск «отвечающего» клиента среди подключенных. Если клиента нет, то сервер завершит звонок «вызывающего» клиента, если «отвечающий» клиент найден, то ему отправляется запрос на соединение по протоколу WebSocket. «Отвечающий» клиент формирует и отправляет ответ на запрос. Если ответ отрицательный, то сервер завершает звонок «вызывающего» клиента, в случае положительного ответа сервер получит id сокетов «вызывающего» и «отвечающего» клиентов, по id сокетов найдет «комнату», в которых сокеты находятся в данный момент. После завершения данного алгоритма происходит создание и обработка буферов сокетов на сервере посредством алгоритма, изображенного на рис. 3.

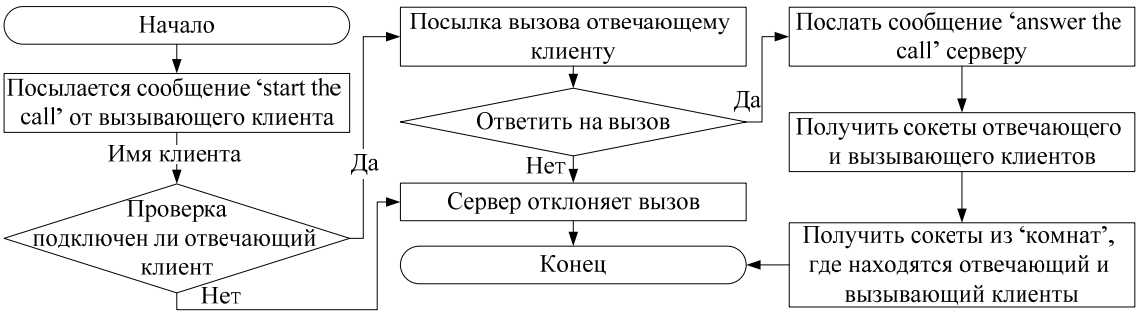

Рис. 2. Алгоритм подготовки клиентских частей перед формированием сигнальных данных

После того как «комната», где находятся сокеты каждого из клиентов, получены, начинает работать алгоритм, представленный на рис. 3. Сначала извлекается сокет из «комнаты» «отвечающего» клиента, для него создается буфер для хранения сокетов, ожидающих соединения по протоколу WebRTC. Затем сокет из «комнаты» «отвечающего» клиента добавляет в уже существующий буфер – сокет, взятый из «комнаты» «вызывающего» клиента. Если взятый сокет из «комнаты» «вызывающего» клиента был не последним, то операция извлечения сокета из «комнаты» «вызывающего» клиента и его добавление в буфер повторяется со следующим сокетом из этой комнаты. После того как будет взят последний сокет из «комнаты» «вызывающего» клиента, сокет из «комнаты» «отвечающего» клиента отсоединяется от своей «комнаты» и добавляется в «комнату» «вызывающего» клиента.

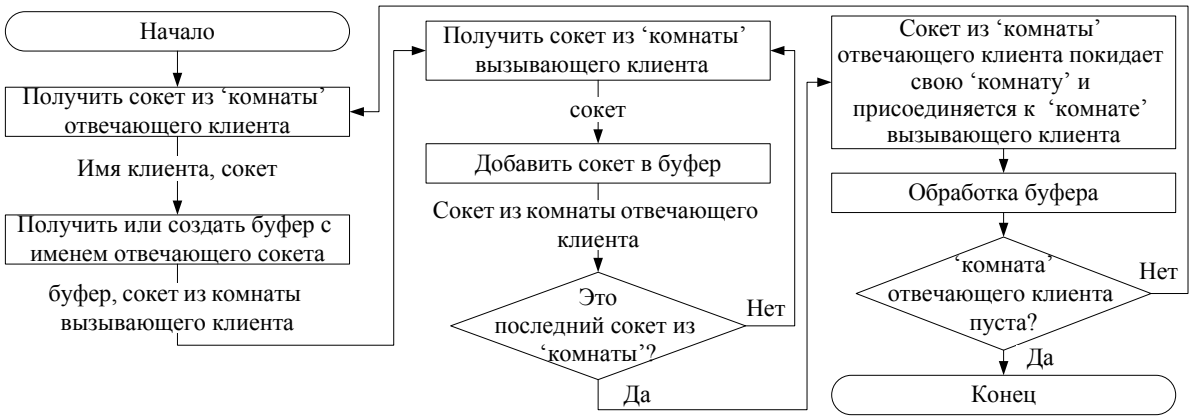

Рис. 3. Алгоритм распределения сокетов и обработки их буферов на сервере

Далее происходит вызов функции обработки буфера сокета, которая выполнит запрос на формирование сигнальных данных для всех клиентов, находящихся в очереди данного буфера. В конце алгоритма выполняется проверка на пустоту «комнаты» «отвечающего» клиента, если «комната» не пуста, то алгоритм повторит все действия с самого начала, в ином случае алгоритм считается завершенным. Как можно заметить, данный алгоритм добавляет в «комнату» «вызывающего» клиента клиентов из «комнаты» «отвечающего» клиента, и при каждом обращении к «комнате» «вызывающего» клиента в данном алгоритме «комната» должна увеличиваться. Но это не так по причине того, что перед началом алгоритма происходит дублирование всех пользователей из «комнаты» «вызывающего» клиента в отдельный массив, таким образом, алгоритм работает правильно, и каждый раз он использует не основную «комнату», а заранее подготовленный массив.

Алгоритм, представленный на рис. 4, является общим для обработки запросов от сервера на формирование сигнальных данных «call offer» или на обработку пришедших от другого клиента сигнальных данных «call answer». В клиентской части приложения видеоконференц-связи существует два отдельных буфера для каждого типа данных.

Приходящий запрос или «сигнальные» данные сначала добавляются в соответствующий им буфер. Затем происходит проверка обоих буферов, не обрабатывается ли один из них в данный момент. Если один из буферов занят обрабатывающей функцией, то алгоритм завершается, а новые данные в буфере будут обработаны позже. Если ни один из буферов не занят, они проверяются на

пустоту. В случае если оба буфера пусты, клиент по протоколу WebSocket отправит серверу уведомление, что он готов получать новые данные для установления соединения с другими клиентами. Если какой-то из буферов содержит данные, то будут извлечены первые в очереди данные и обработаны соответствующим образом для установления подключения. Как только подключение будет установлено, алгоритм продолжит свою работу с места опроса занятости буфера.

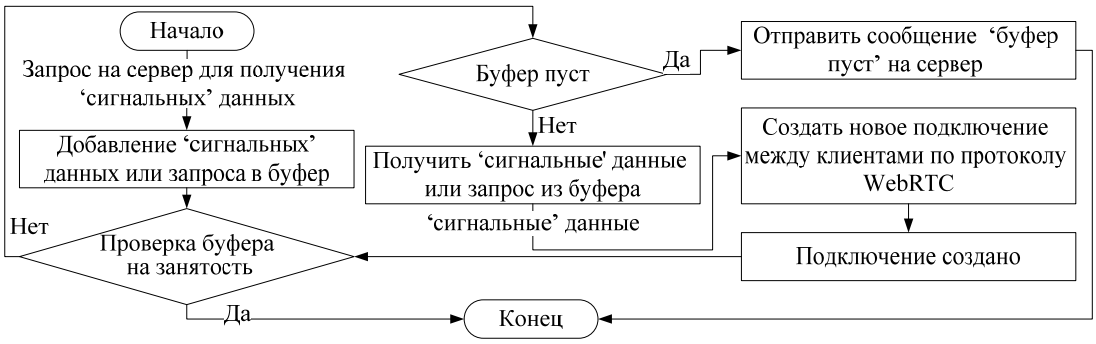

Рис. 4. Алгоритм работы с буфером данных на клиенте

Стоит отметить, что сигнальные данные типа «candidate» не имеют специальных буферов для их хранения ни на клиенте, ни на сервере, так как их обработка происходит сразу же, как только клиент их получит. Во время обработки «сигнальных» данных типа «candidate» буферы, принадлежащие данным типа «call answer» или «call offer», в зависимости от ситуации обработки продолжают быть заняты обрабатывающей функцией. Таким образом, остальные данные типа «call answer» или «call offer» продолжают добавляться в буферы, не нарушая алгоритма соединения клиентов по протоколу WebRTC.

Три приведенных выше алгоритма составляют одну из главных частей приложения, которая осуществляет создание видеоконференц-связи с другими пользователями посредством peer-to-peer протокола WebRTC. Алгоритмы позволяют контролировать асинхронную архитектуру клиентской и серверной частей приложения, осуществляя обработку данных по необходимости, препятствуя возникновению ситуаций перемешивания данных и прерывания выполняющихся соединений по протоколу WebRTC. Таким образом, в приложении достигается возможность создавать групповые видеоконференции и контролировать их состояние с помощью объединенных в «комнаты» клиентов на сервере.

Заключение. Предложенная архитектура обмена данными в приложении видеоконференц-связи позволяет распределить каналы передачи данными по соответствующим протоколам и разделить задачи между ними. Так как изначально архитектура приложения видеоконференцсвязи является асинхронной, то были выполнены разработка и внедрение новых алгоритмов по управлению данными, необходимых для соединения клиентских приложений по протоколу WebRTC. В итоге было получено полноценное приложение видеоконференц-связи, способное производить групповые звонки и контролировать состояние клиентов, связанных в общие группы – «комнаты» на сервере во время проведения группового чата.

Стоит отметить, что на данный момент существуют еще некоторые проблемы, ограничивающие использование протокола WebRTC на устройствах: 1) всего три браузера (Opera, Mozilla Firefox, Google Chrome) поддерживают данный протокол; 2) требуется наличие мощного процессора и достаточного количества памяти для обработки аудио- и видеопотоков данных. Кроме того, указанные браузеры не работают с графическими сопроцессорами, в результате нагружается основной процессор. Дальнейшие исследования будут направлены на упрощение способа передачи и улучшение обработки аудио- и видеоданных с использованием пиринговых связей для распределения нагрузки по обработке данных между клиентами.

Работа выполнена при частичной финансовой поддержке РФФИ (проект № 13-08-0741-а).

Литература

1. X. Hei, A measurement study of a large-scale P2P IPTV system / X. Hei, C. Liang, J. Liang et al. / IEEE Transactions on Multimedia, 2007.

2. Hei X. Inferring network-wide quality in P2P live streaming systems / X. Hei, J. Liang, Y. Liu, K.W. Ross / IEEE Journal on Selected Areas in Communications, 2007.

3. Di Wu. Redesigning multi-channel P2P live video systems with View-Upload Decoupling / Di Wu, C. Liang, Y. Liu, K.W. Ross / Computer Networks. – 2010, №54. – P. 2007–2018.

4. Civanlar M.R. et al. Peer-to-peer multipoint videoconferencing on the Internet // Signal Processing: Image Communication. – 2005, №20. – P.743–754.

5. Volfin I. Dominant speaker identification for multipoint videoconferencing / I. Volfin, I.l Cohen / Computer Speech and Language. – 2013. – Vol. 27. – P. 895–910

6. Ronzhin A. Speaker Turn Detection Based on Multimodal Situation Analysis / A. Ronzhin, V. Budkov // Springer International Publishing Switzerland. M. Zelezny et al. (Eds.): SPECOM–2013, LNAI 8113. – 2013. – P. 302–309.

7. Ronzhin A. PARAD-R: Speech Analysis Software for Meeting Support / A. Ronzhin, V. Budkov, I. Kipyatkova / In Proc. of the 9th International Conference on Information, Communications and Signal Processing ICICS-2013. Tainan, Taiwan. – 2013.

8. Ronzhin A. Context-Aware Mobile Applications for Communication in Intelligent Environment / A.L. Ronzhin, A.I. Saveliev, V.Yu. Budkov // Springer-Verlag Berlin Heidelberg, S. Andreev et al. (Eds.): NEW2AN/ruSMART–2012, LNCS 7469. – 2012. P. 307–315.

9. Meshcheryakov R.V. Dialogue as a basis for construction of speech systems / Meshcheryakov R.V., Bondarenko V.P. // Cybernetics and Systems Analysis. – 2008. – T. 44. № 2. – C. 175–184.

10. Бондаренко В.П. Обработка речевых сигналов в задачах идентификации / В.П. Бондаренко, А.А. Конев, Р.В. Мещеряков / Изв. высш. учебных заведений. Физика. – 2006. – Т. 49, № 9. – С. 207.

11. Мещеряков Р.В. Структура систем синтеза и распознавания речи // Известия Томского политехнического университета. – 2009. – Т. 315, № 5. – С. 127–132.

12. Vishnu Monn. Software-based serverless endpoint video combiner architecture for high-definition multiparty video conferencing / Vishnu Monn Baskaran, Yoong Choon Chang, Jonathan Loo, KokSheik / Journal of Network and Computer Applications. – 2013. – Vol. 36. – P. 336–352.

13. Ramzan N. et al. Video streaming over P2P networks: Challenges and opportunities // Signal Processing: Image Communication. – 2012. – № 27. – P. 401–411.

14. Савельев А.И. Оптимизация алгоритмов распределения потоков мультимедийных данных между сервером и клиентом в приложениях видеоконференц-связи // Труды СПИИРАН. – 2013. – Вып. 31. – С. 61–79.

15. Ronzhin A.L. Context-Aware Mobile Applications for Communication in Intelligent Environment / A.L. Ronzhin, A.I. Saveliev, V.Yu. Budkov // Springer-Verlag Berlin Heidelberg, S. Andreev et al. (Eds.): NEW2AN/ruSMART. – 2012. – LNCS 7469. – P. 307–315.

Савельев Антон Игоревич
Мл. науч. сотр. лаборатории речевых и многомодальных интерфейсов
Санкт-Петербургского института информатики и автоматизации РАН (СПИИРАН)
Тел.: 8 (812) 328-70-81
Эл. почта: saveliev@iias.spb.su

Прищепа Мария Викторовна
Канд. техн. наук, науч. сотр. лаборатории речевых и многомодальных интерфейсов СПИИРАН
Тел.: 8 (812) 328-70-81
Эл. почта: prischepa@iias.spb.su

Saveliev A.I., Prischepa M.V.

Architecture of lossless data exchange in pear-to-pear web application of videoconference

The problem of client connection and audiovisual data transmission in pear-to-pear web application of videoconference are discussed. At the interaction of several clients and server parts of the videoconference application based on WebRTC protocol the partial or complete loss of signal data that hampers the client connection. The proposed architecture of data transmission and storage on client and server provides buffering and following processing of the signal data with the exception of their loss and support of interaction between client groups.

Keywords: peer-to-peer connection, peer-to-peer protocols, videoconferencing, web applications, media stream distribution.