Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Программно-алгоритмический комплекс для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS»

Студент группы **ИУ7-83Б** _____ **П. А. Топорков**
(Подпись, дата) (И.О. Фамилия)

Руководитель ВКР _____ **Ю. И. Терентьев**
(Подпись, дата) (И.О. Фамилия)

Нормоконтролер _____ **Д. Ю. Мальцева**
(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчетно-пояснительная записка к выпускной квалификационной работе «Программно-алгоритмический комплекс для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS» содержит 66 с., 26 рис., 3 табл., 27 ист.

Ключевые слова: потоковое аудио, iOS.

Классифицированы методы воспроизведения потоковых аудиоданных в мобильном приложении. Проведён анализ программных средств воспроизведения аудиоданных в мобильном приложении на операционной системе iOS. Рассмотрены протоколы потоковой передачи данных и файлы, хранящие данные аудиосигнала. Спроектирован и разработан метод воспроизведения аудиоданных в мобильном приложении на операционной системе iOS. Произведено сравнение времени и выделения оперативной памяти для обработки аудиоданных разработанным программным комплексом и существующими аналогами.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1) Стриминг (англ. streaming) —– потоковая передача данных по сети интернет;
2) Битрейт (от англ. bitrate) —– количество бит, используемых для передачи или обработки данных в единицу времени;
3) iOS (iPhone OS) —– операционная система для мобильных устройств, выпускаемых американской компанией «Apple»;
4) Аналого-цифровой преобразователь —– устройство, преобразующее входной аналоговый сигнал в дискретный формат;

ВВЕДЕНИЕ

За последнии 10 лет рынок аудиостриминга значительно вырос и претерпел сильные изменения[1]. Всё больше людей пользуются сервисами потокового прослушивания музыки, подкастов и аудиокниг.

Иследование [2] показало, что число пользователей мобильных устройств превысило 2.5 миллиарда человек. Ежегодный прирост в $5 - 10\%$ пользователей способствет к стремительному увеличению популярности мобильных сервисов. В результате возникает потребность в поддержке воспроизведения потокововых аудиоданных на мобильных устройствах.

Для реализации мобильного программного обеспечения необходимы программные комплексы, поддерживающие современные мобильные операционные системы и предоставляющие готовые программные инструменты для потокового воспроизведения аудиоданных.

На текущий момент количество программных комплексов с открытым исходным кодом для воспроизведения потоквого аудио, поддерживающих различные протоколы передачи потоковых данных и частоты дескретизации, наложение звуковых эффектов и варьирование качества звука, для мобильных приложений на операционной системе Android значительно больше, чем для операционной системы iOS [13].

Целью работы является разработка программно-алгоритмического комплекса для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS. В рамках работы необходимо решить следующие задачи:

— ввести основные понятия предметной области;
— рассмотреть существующие средства воспроизведения аудиоданных в операционной системе iOS;
— провести анализ протоколов потоковой передачи данных;
— рассмотреть форматы хранения аудиоданных;

— спроектировать и реализовать программно-алгоритмический комплекс для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS;
— сравнить время и выделение оперативной памяти для обработки аудиоданных разработанным программным комплексом и существующими аналогами;

1 Аналитическая часть

В данном разделе проводится анализ предметной области. Рассматриваются базовые понятия цифрового представления аудиоданных, прикладные протоколы передачи потоковых данных, рассматриваются виды форматов хранения аудиоданных. Проводится краткий обзор PCM формата хранения цифрового аудио. Дается анализ существующих средств воспроизведения аудиоданных в операционной системе iOS.

1.1 Анализ предметной области

Воспроизведение потоковых аудиоданных на мобильных устройствах зависит от протоколов потоковой передачи данных, форматов хранения аудиоданных, их обработки, а также от программных интерфейсов, предоставляемых разработчикам.

Зачастую используемые решения могут быть не универсальными, а их выбор продиктован спецификой области применения и поставленными задачами. Так при построении системы транслирования радиоэфиров в реальном времени, можно принебречь качеством (разрешением звука) записи, а для системы прослушивания аудикниг, в которых важна дикция, задержка передачи данных не является важным критерием.

Системным решением для воспроизведения потокового аудио на операционной системе iOS являются два комплекса программных интерфейсов: AVFoundation [18] и AudioToolbox [19]. Они являются частью системных компонентов и позволяют воспроизводить хранимые аудиофайлы в формате ACC, а также потоковое аудио, транслируемое с помощью серверной части с использованием протоколов потоковой передачи данных HLS и HTTPS. Поддеживают высокие разрешение звука и частоту дискретизации.

Рассмотрим сторонние решения, которые частично или полностью опираются на перечисленные комплексы программных интерфейсов, а также решают задачи мультиплатформенности и поддержки несисистемных средств и

аудиоформатов для воспроизведения аудиоданных в мобильном приложении.

VLCKit [20] - библиотека для воспроизведения ACC файлов [6], использующая программный интерфейс libVLC [22], реализованный для операционных систем: Linux, MacOS, Window, iOS, Android. Данная библиотека поддерживает потоковое воспроизведение аудиоданных в ACC формате с низким разрешением звука, с поддержкой протокола RTSP [10]. На текущий момент библитека не поддерживается для последних версий перечисленных операционных систем.

SwiftAudioPlayer [21] - программное обеспечение с открытым исходным кодом, позволяющее воспроизводить потокое аудио в формате WAV [3] с низкими разрешением звука и частотой дискретизации на операционной системе iOS с использованием языка программирования Swift [26]. Поддерживает получение аудиоданных с помощью протокола потоковой передачи данных HLS [8]. За основу взят системный комплекс программных итерфейсов AVFoundation.

1.2 Цифровое представления аудиоданных

Чтобы представить звук в цифровом виде, необходимо преобразовать это изменяющееся напряжение в ряд чисел, представляющих его амплитуду. Этот процесс известен как аналого-цифровое преобразование или дискретизация звука. Числа, выдаваемые аналого-цифровым преобразователем, в общем случае произвольны.

Цифровая система, такая как компьютер, не может напрямую представлять непрерывный сигнал. Вместо этого применяется ряд процессов для оцифровки звука:

— Дискретизация — преобразование сигнала в конечное множество дискретных моментов времени;

— Разрешение звука — количество используемых дискретных уровней амплитуды сигнала;

— Квантование — округление значения сигнала к одному из конечных дискретных уровней амплитуды. Обычно выражается в битах, то есть как логарифм по основанию 2 фактического числа;

В результате вышеприведённых преобразований волна из своей непрерывной формы переходит в дискретизированную. На рис. 1 представлена непрерывная формы волны:

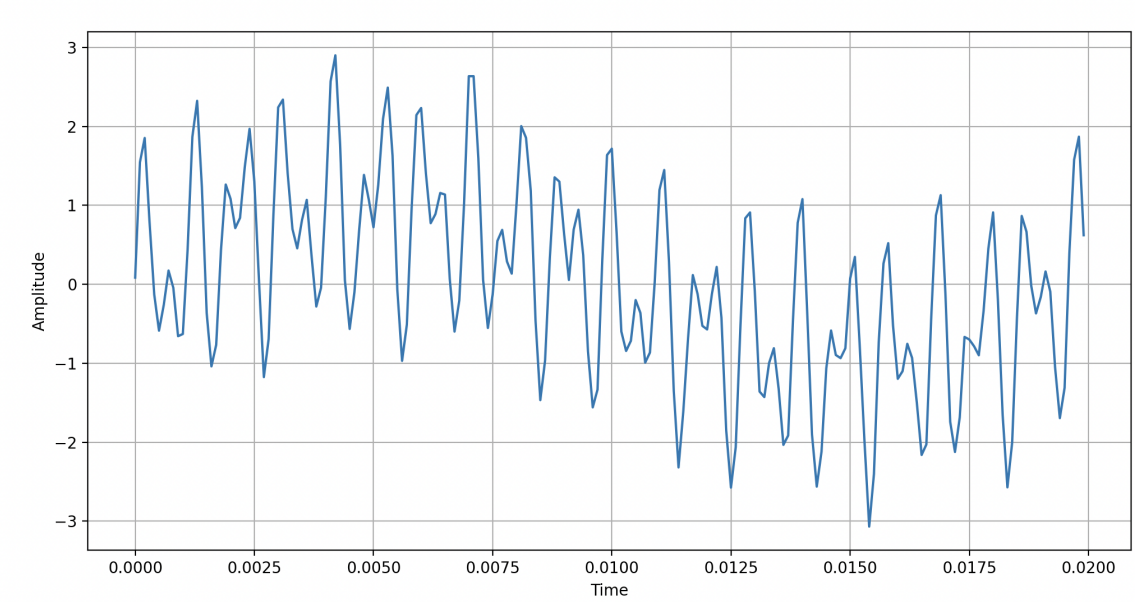

Рис. 1 – Непрерывная форма волны.

На рис. 2 представлено наложение дискретных уровней, полученных в ходе дискретизация, на непрерывную форму волны:

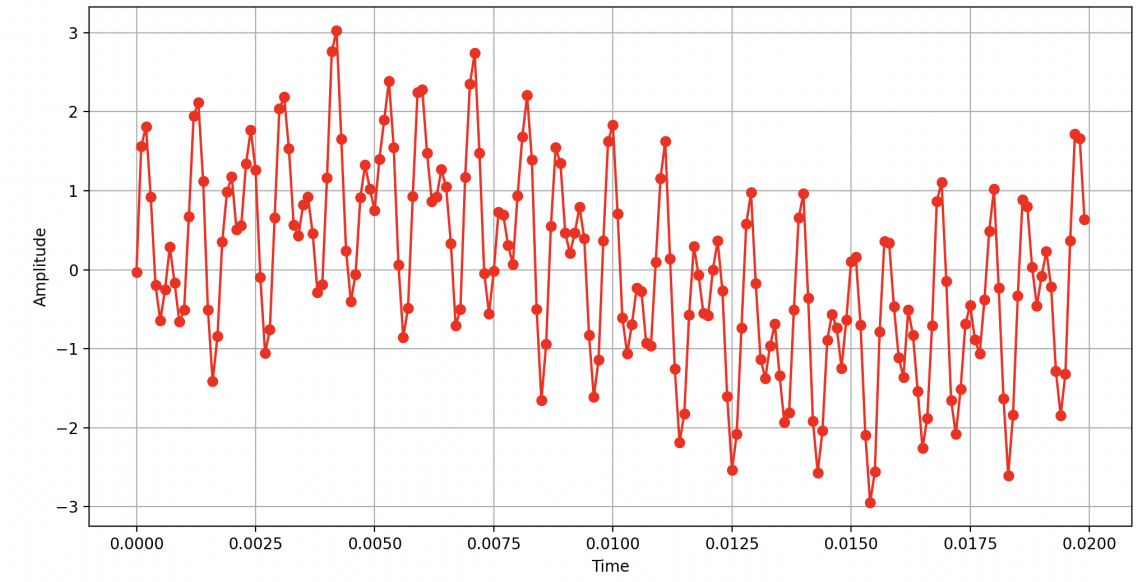

Рис. 2 – Наложение дискретных уровней на непрерывную форму волны.

Частота дискретизации

Частота дискретизации —– это количество измерений амплитуды сигнала в секунду. Чем выше частота дискретизации, тем точнее дискретизированный сигнал будет представлять исходный сигнал.

Выбор частоты дискретизации определяется теоремой Котельникова о связи непрерывных и дискретных сигналов. Данная теорема утверждает, что если максимальная частота спектра оригинального сигнала равна f_c, то при частоте дискретизации строго превышающей $2f_c$ возможно полностью восстановить исходный сигнал из дискретного вида.

Другими словами, теорема Котельникова говорит о том, что непрерывный сигнал можно представить в виде интерполяционного ряда Фурье. Данное представление отображено формулой ниже:

$$\begin{cases} x(t) = \sum\limits_{k=-\infty}^{\infty} x(k\Delta) sinc[\frac{\pi}{\Delta}(t - k\Delta)], \\ 0 < \Delta \leq \frac{1}{2f_c}, \end{cases}$$

где Δ – интервал дискретизации (сек.).

Модуляция сигнала

Модуляция сигнала представляет собой процесс изменения параметров несущего сигнала (колебания) с помощью модулирующего низкочастотного сигнала, где модулирующий сигнал хранит передаваемую информацию, а несущий сигнал отвечает за передачу.

Для цифрового представления дискретизированных аналоговых сигналов используются следующие методы модуляции:

— импульсно-кодовая модуляция (PCM);

— плотностно-импульсная модуляция (DSD);

Импульсно-кодовая модуляция [23] является стандартной формой цифрового звука в вычислительных машинах и устройствах хранения данных. В

данном методе амплитуда аналогового сигнала дискретизируется через равные промежутки времени, а затем квантуется. Для обработки аудиоданных в PCM формате необходима поддержка PCM буфера, который будет хранить пакеты аудиофайлов, полученных с помощью импульсно-кодовой модуляции.

На рис. 3 и 4 представлены квантование сигнала при импульсно-кодовой модуляции, а также представление PCM буфера соответственно:

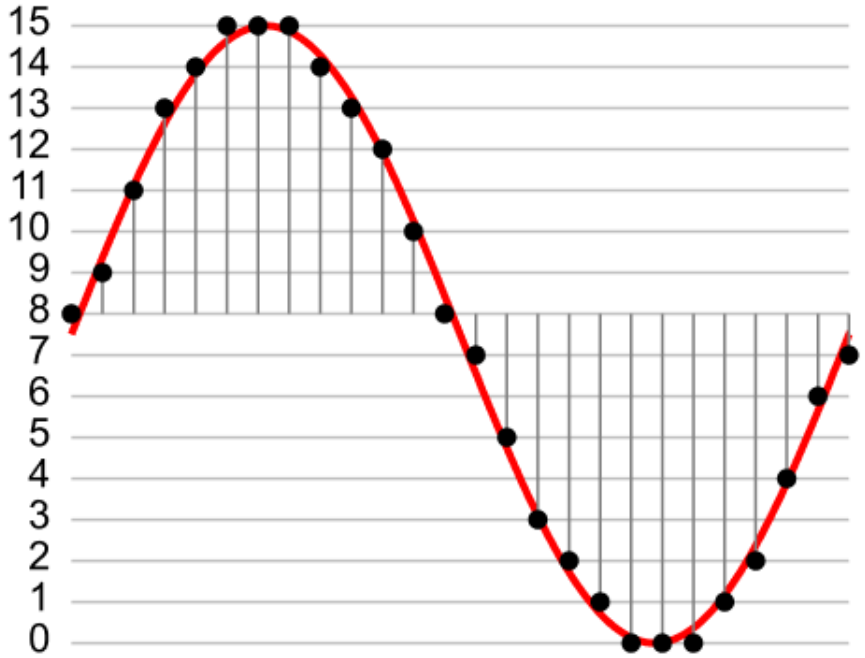

Рис. 3 – Квантование сигнала при импульсно-кодовой модуляции.

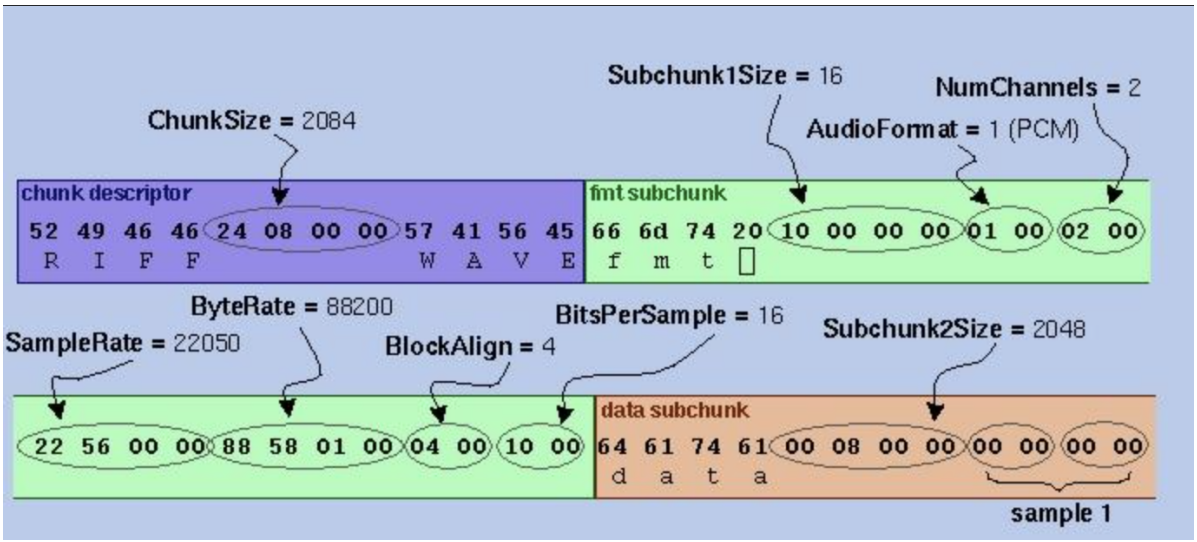

Рис. 4 – Представление PCM буфера.

DSD является однобитным аудиоформат, который получается в ходе плотностно-импульсной модуляции [23], являющийся частным случаем сигма-дельта модуляции. Данный формат устарел и применялся для оптических носителей SACD. Модуляция происходит при избыточной частоте дискретизации, а также при избыточном разрешении звука, что позволяет уменьшить шум квантования. Он происходит при частоте сигнала, амплитуда которого многократно превышает допустимое разрешение. Для обработки аудиоданных в DSD формате необходимо использовать специальный DSD буфер, хранящий однобитные пакеты аудиоданных.

На рис. 5 представлено отличие плотностно-импульсной модуляции от импульсно-кодовой модуляции:

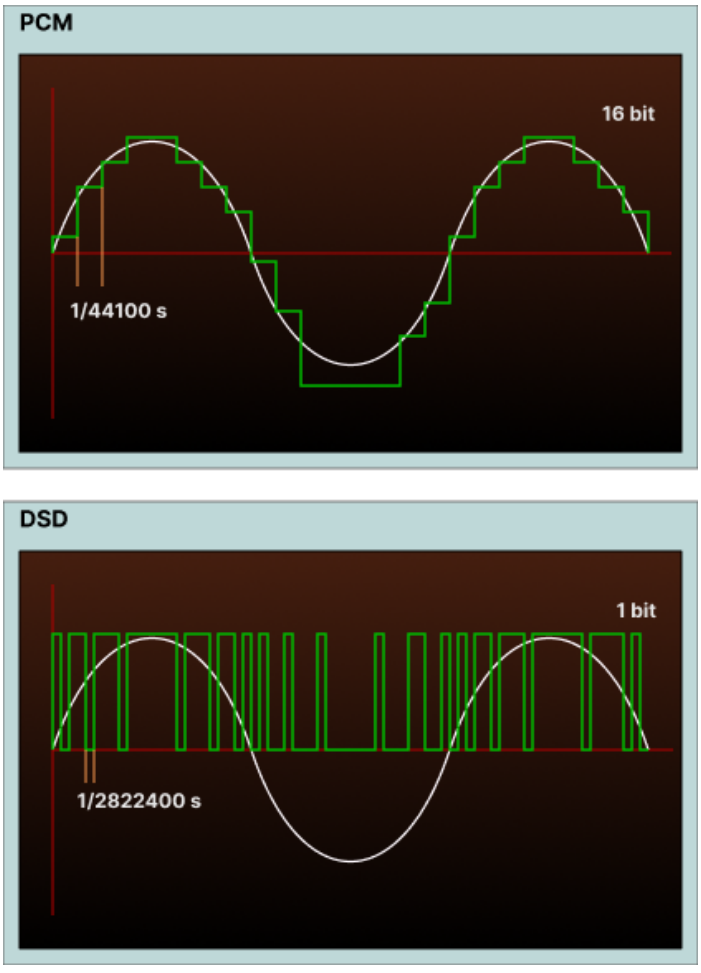

Рис. 5 – Отличие плотностно-импульсной модуляции от импульсно-кодовой модуляции.

На текущий момент оцифровка звука в PCM формат и обработка аудиоданных с помощью PCM буфера является наиболее распространенной. Большинство библиотек или программных интерфейсов, ориентированных на воспроизведение аудиофайлов или потокового аудио, поддерживают работу с ним.

1.3 Анализ форматов хранения аудиоданных

В общем случае аудиофайлы могут быть не сжаты, а также не иметь явного формата. Чаще всего аудиоданные обрабатывают для хранения по специальному формату, хранят в файле специального формата, представляющим «заголовок», а также тело — выборку целых чисел, характеризующих цифровой сигнал для воспроизведения аудио.

Заголовок может содержать:

— тип файла;
— разрешение звука;
— частота дискретизации;
— информацию о сжатие, если таковое имеется;
— количество байтов, следующих за заголовком;
— информацию о название и исполнителе;

1.3.1 Необработанные звуковые файлы

Звуковые файлы, которые не имеют заголовка, а содержат только числовую выборку, характеризующую цифровой сигнал, называются необработанными. Отсутствие явной информации о частоте дискретизации и разрешении позволяет использовать аудиоданные только по принятым соглашениям, которые принимаются между теми, кто упаковывает и использует данные аудиофайлы.

1.3.2 WAV (WAVE) формат

Для данного формата не применяется сжатие к битовому потоку, аудиозаписи хранятся с разными частотами дискретизации и битрейтами. Однако WAV файлы имеют больший размер по сравнению с другими популярными форматами, такими как MP3, в которых используется сжатие с потерями для умень-

шения размера файла при сохранении того же качества звука. Разрешение звука может быть как беззнаковым 8-битным, так и знаковым 16-битным (фрагменты дублируют информацию, найденную в других фрагментах). Засчёт дубликатов блоков данных, характеризуют цифровой сигнал в конкретный момент времени, обработка информации может занять дополнительное время.

Заголовок файла содержит:

— размер файла в байтах;
— длину блока данных;
— количество аудиоканалов;
— битрейт;
— частота дискретизации;

1.3.3 MP4 формат

MP4 [4] —— это международный стандарт кодирования аудиовизуальных материалов, из-за высокой степени сжатия, результирующие файлы имеют меньший размер с почти полным сохранением исходного качества. Данные в файле MP4 разделены на два раздела, первый из которых содержит мультимедийные данные, а второй содержит метаданные. Структуры в MP4 обычно называются атомами или блоками. Минимальный размер атома составляет 8 байт (первые 4 байта определяют размер, а следующие 4 байта определяют тип).

Первый раздел содержит следующую информаию:

— таблица выборок значений цифрового сигнала в момент времени;
— заголовок аудиоданных, содержащий частоты дискретизаии (до 48 кГц) и разрешение звука;
— количество аудиоканалов;
— битрейт;

Второй раздел содержит следующую информаию:

— дескриптор файла;
— информацию о сжатии;

1.3.4 MP3 формат

Файл MP3 [5] состоит из фреймов, каждый из которых состоит из заголовка и блока данных. Фреймы не являются независимыми и обычно не могут быть извлечены на произвольных границах переходов фреймов. Блоки данных файла содержат информацию об аудио с точки зрения частот и амплитуд. Заголовок идентифицирует начало допустимого кадра. Затем следуют 3 бита, где первый бит показывает, что это стандарт MPEG, а оставшиеся 2 бита показывают, что используется уровень MPEG-1 Audio Layer [7]. После этого значения будут различаться в зависимости от содержимого конкретного аудиофайла.

Заголовок фрейма содержит следующую информацию:

— информация о переходе на текщий фрейм;
— идентификатор версии MPEG Audio;
— описание фрейма;
— бит, который показывает, что фрейм зашифрован;
— битрейт;
— частота дискретизации (от 8 кГц до 48 кГц);
— приватный ключ шифрования;

1.3.5 AAC формат

AAC (Advanced Audio Coding) относится к стандарту цифрового кодирования звука, который представляет аудиофайлы на основе сжатия звука с потерями. Он был запущен как преемник формата файла MP3 с учетом того, что у сторонних разработчиков возникли проблемы с реализацией новых идей в процессе кодирования, основанных на развитии методов сжатия данных. AAC обеспечивает лучшее качество звука по сравнению с MP3 при той же скорости передачи данных. Основные различия между AAC и MP3 с точки зрения улучшений заключаются в поддержке более широкого диапазона и частот дискретизации (от 8 кГц до 96 кГц). Разбиение по фреймам и содержание заголовка фрейма идентичны с MP3 форматом.

1.3.6 Сравнение форматов хранения аудиоданных

Сравнение рассмотренных форматов хранения аудиоданных рассмотрены в табл. 1. Обозначения:

— 1 — поддержка частоты дискретизации более 48 кГц;

— 2 — наличие заголовка с метаинформацией;

— 3 — поддержка сжатия данных;

— 4 — фрагментация данных цифрового сигнала;

Табл. 1 – Сравнение форматов хранения аудиоданных

Формат	**1**	**2**	**3**	**4**
WAV	+	-	-	-
MP3	-	+	+	+
MP4	-	+	+	-
ACC	+	+	+	+

По результатам сравнения можно сделать вывод, что наиболее оптимальными форматами хранения аудиоданных являются ACC и MP3. Принципиальное отличие заключается в диапазоне поддерживаемых частот дискретизации. При требовании поддержки частоты дискретизации до 48 кГц форматы равноэффективны, но при необходимости в поддержке частоты до 96 кГц, необходимо использовать формат ACC. Следует отметить, что формат WAV подходит в случае заранее принятых стандартов звукозаписи, при которых метаданные, хранящиеся в заголовках других форматов, будут известны.

1.4 Анализ протоколов передачи потоковых данных

Для передачи данных при стриминге необходимо использовать прикладные протоколы передачи данных, сжимающие данные для быстрой транспортировки, а также форматирующие данные в специальный формат.

1.4.1 HTTP/HTTPS

HTTP [24] — протокол прикладного уровня для передачи данных в формате гипертекста, использующий протокол транспортного уровня TCP [11]. Передача данных осуществляется в виде сегментов (пакетов) данных. Гарантируется, что запрошенные пакеты данных будут доставлены и с необходимым порядком пакетов. Изначально при помощи HTTP протокола можно передавать данные только в формате гипертекста (HTML), но для потоковой передачи данных можно настроить специальный механизм HTTP Long Polling.

При потоковой передаче данных HTTP сервер настроен на удержание определенного запроса от клиента (HTTP Push). Сервер сохраняет соединение с клиентом открытым и отправляет обновления относящиеся к запросу при доступности. В данном подходе отсутствует необходимость в постоянном запросе к серверу, т.е. клиент отправляет единственный запрос на соединение, а также единственный запрос на разрыв соединения.

В запросе, для установки постоянного соединения между клиентом и серверов, необходимо передать заголовок $TransferEncoding : chunked$. Это позволит определить на серверной стороне, что передача данных должна проводиться с помощью HTTP Push. На рис. 6 представлена схема передачи данных с сервера на клиент с помощью HTTP Push.

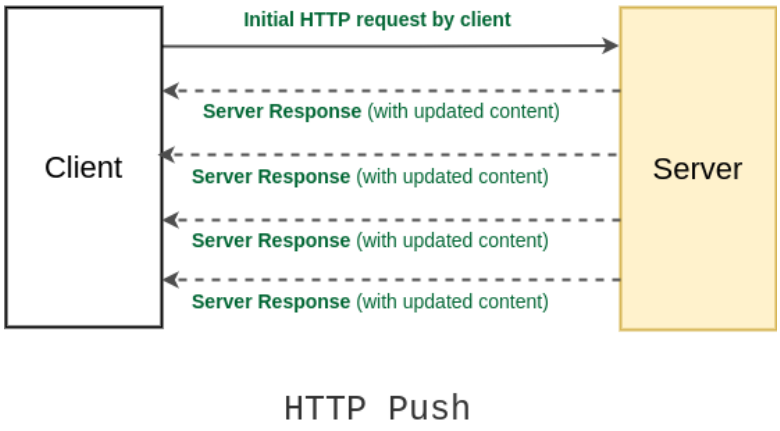

Рис. 6 – Схема передачи данных с сервера на клиент с помощью HTTP Push.

Следует отметить, что использование протокола HTTP для передачи по-

токовых аудиоданных является наиболее распространённым. При передаче пакетов аудиоданных придерживаются следующих правил:

— первые пакеты хранят данные относящиеся к метаинформации (заголовку) передаваемого аудиофайла;

— остальные пакеты хранят информацию о звуковом сигнале;

1.4.2 HLS (HTTP Live Streaming)

HLS — это протокол потоковой передачи, который набирает всё большую популярность. Данный протокол создан на основе протокола прикладного уровня HTTP, использующий протокол транспортного уровня TCP. Концептуально HTTP Live Streaming состоит из трех частей:

— серверный компонент;

— компонент доставки;

— клиентское программное обеспечение.

В классической конфигурации аппаратный кодировщик принимает на вход аудиофайл. Происходит кодировка аудио в ACC формат, затем программный сегментатор разбивает поток на серии коротких медиафайлов, размещающихся на сервере. Сегментатор создает и поддерживает файл-индекс, содержащий список медиафайлов. Клиент считывает индекс, затем запрашивает перечисленные файлы по порядку и отображает их без остановок, разрывов между сегментами. На рис. 7 изображена схема принципа работы протокола HLS.

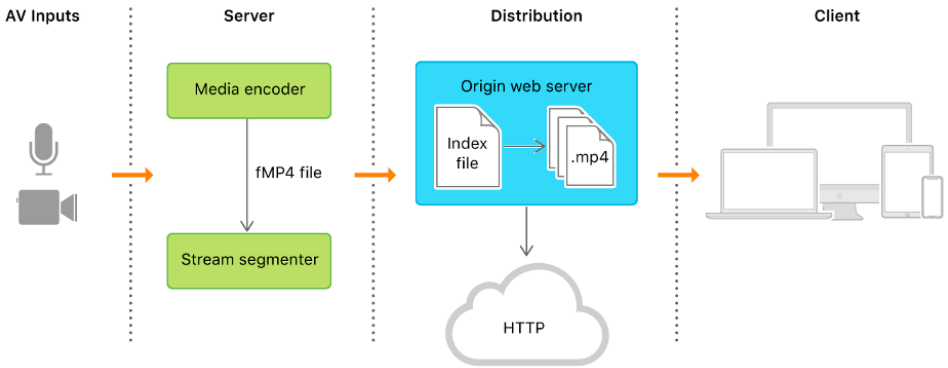

Рис. 7 – Схема принципа работы протокола HLS.

К достоинствам данного протокола можно отнести:

— сегментацая и индексация потка файлов, полученных в результате работы сегментатора;
— поддержка ACC формата, что позволяет передавать аудиоданные с частотой дискретизации от 8 до 96 кГц;
— гарантированная последовательная передачи пакетов без потерь;

К недостаткам данного протокола можно отнести:

— предварительная обработка исходного файла, которая заключается в разбиении и сжатии на подфайлы специального формата;
— сильная задержка при передаче данных, которая не позволяет транслировать данные в режиме реального времени;

1.4.3 Low-Latency HLS

Low-Latency HLS [9] позволяет передавать медиа файлы на параллельных каналах, с помощью разделения медиафайлов на большее количество маленьких файлов CMAF Chunks. Эти файлы называются частичными сегментами HLS. Т.к. каждый частичный сегмент имеет меньшую длительность, он может быть упакован, передан или добавлен быстрее чем его родительский сегмент. Пока стандартный медиасегмент может быть длительностью 6 секунд каждый, частичный сегмент может быть до 1 секунды.

Ускорение передачи данных достигается с помощью усложнения процесса предварительной обработки файла. Происходит дробление подфайлов, полученных сегментатором, на частичные сегменты. Создаётся список контекста сегментов, которые находятся в очереди на отправку данных. Параллельно происходит фильтрация списка, которая отбрасывает сегменты, которые считаются незначительными.

По сравнению с стандартной организацией HLS, данный протокол обеспечивает более быструю передачу пакетов, но требует дополнительной фильтрации сегментов, на которые делится исходный файл. Также может происходить потеря какого-то блока данных цифрового сигнала, что приведёт к искажению воспроизведения аудиоданных.

1.4.4 RTSP (Real-Time Streaming Protocol)

RTSP – протокол прикладного уровня. Реализует потоковую передачу медиа данных в реальном времени, а также устанавливает и управляет либо одним, либо несколькими синхронизированными во времени потоками. Источники данных могут быть как источниками реального времени, так и хранимыми. RTSP поддерживает как передачу данных, сегментируемых аппаратным сегментатором, так и передачу с помощью датаграм, благодаря чему совместим и с протоколом транспортного уровня TCP, и с UDP [12].

Механизм передачи данных в реальном времени обусловлен отсутствием предварительной фильтрации сегментов аудиофайлов. Аналогово-цифровой преобразователь полностью перенаправляет поток данных на RTSP сервер, который сегментирует поступающие данные, сжимает и конвертирует их в ACC или MP3 формат.

К достоинствам данного протокола можно отнести:

— отсутствие предварительной обработки поступаемых данных;

— возможность передачи данных в режиме реального времени;

— гарантированная последовательная передачи пакетов;

К недостаткам данного протокола можно отнести:

— потеря пакетов данных при обработке аналогово-цифрового сигнала;

— сжатие данных обусловливает частоту дискретизации до 48 кГц;

1.4.5 Сравнение протоколов передачи потоковых данных

Сравнение рассмотренных протоколов передачи потоковых данных представлено в табл. 2. Обозначения:

— 1 — поддержка частоты дискретизации более 48 кГц;

— 2 — трансляция в режиме реального времени;

— 3 — гарантированная последовательная передача пакетов без потерь;

— 4 — отсутствие необходимости предварительной обработки аудиофайлов перед трансляцией;

Табл. 2 – Сравнение протоколов передачи потоковых данных

Протокол	**1**	**2**	**3**	**4**
HTTP	+	+	+	+
HLS	+	-	+	-
Low-Latency HLS	+	-	-	-
RTSP	-	+	-	+

По результатам сравнения можно сделать вывод, что для потоковой передачи аудиоданных хорошо подходит протокол HTTP, т.к. он удовлетворяет всем пунктам сравнения. Протоколы семейства HLS хорошо подходят для транслирования предварительно обработанных аудиофайлов с частотой дискретизации более 48 кГц, но не поддерживают передачу данных в режиме реального времени. Протокол RTSP, напротив, лучше подходит для трансляции в реальном времени, но плохо поддерживает файлы с большой частотой дискретизации. При использовании Low-Latency HLS и RTSP возможна потеря данных при трансляции.

1.5 Анализ существующих средств воспроизведения аудиоданных в операционной системе iOS

Воспроизведение аудиоданных зависит от программных компонентов, а также доступных ресурсов, предоставляемых разработчикам. Ниже рассмотрены доступные подходы и механизмы для воспроизведения аудио в операционной системе iOS.

1.5.1 AVPlayer

AVPlayer [14] — программный интерфейс, являющийся частью комплекса программных интерфейсов AVFoundation. Принцип работы заключается в загрузке аудиофайла целиком в оперативную память устройства. Работа происходит с аудиофайлами целиком, в оперативную память устройства выгружаются заголовки и блоки данных цифрового сигнала, что напрямую связывает

возможность проигрывания аудио с доступным размером оперативной памяти. На рис. 8 представлена схема загрузки в оперативную память и воспроизведения аудиофайла с помощью AVPlayer.

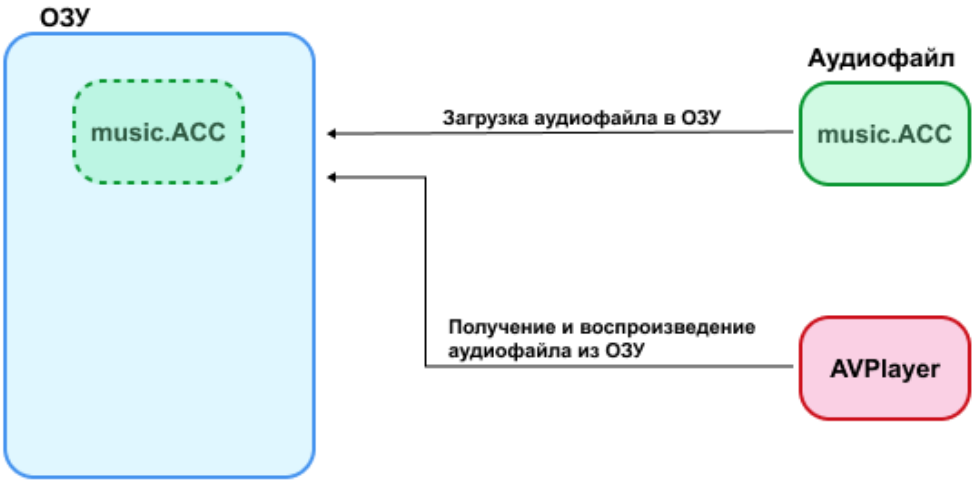

Рис. 8 – Схема загрузки в оперативную память и воспроизведения аудиофайла с помощью AVPlayer.

Данный подход позволяет воспроизводить локальные аудиофайлы, хранящиеся на устройстве и имеющие формат ACC. Поддержка частоты дискретизации вплоть до 96 кГц, а также разрешение звука до 32 бит даёт возможность воспроизведения и управления аудиоданными высокого качества.

В контексте воспроизведения потоковых аудиоданных необходим следующий сценарий использования AVPlayer:

1. предварительное разбиение аудиофайла на подфайлы того же формата с определением метаинформации воспроизводимого участка базового файла на стороне сервера;
2. поочерёдное получение и буферизация файлов (принцип очереди FIFO) на стороне клиента;
3. поочерёдное получение и воспроизведение буферизованных файлов;

На рис. 9 представлена схема буферизации подфайлов оригинального файла и их перемещение в оперативную память для воспроизведения при потоковой передаче аудиоданных.

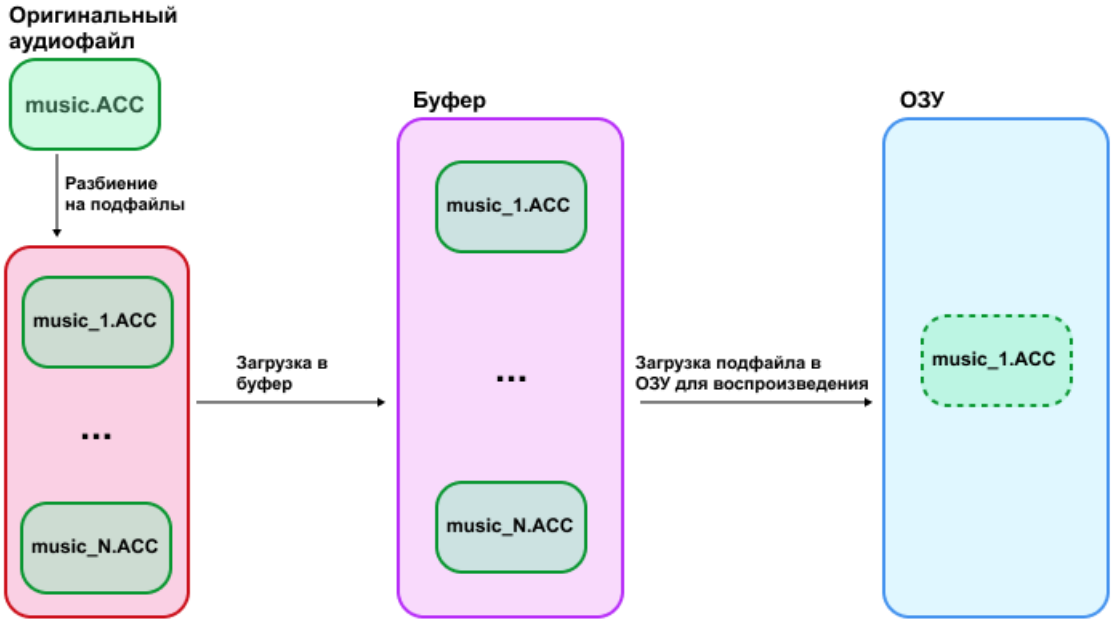

Рис. 9 – Схема буферизации подфайлов оригинального файла и их перемещение в оперативную память для воспроизведения при потоковой передаче аудиоданных.

К плюсам данного подхода можно отнести:

— поддержка аудиофайлов с частотой дискретизации до 96 кГц;
— поддержка аудиофайлов с разрешеним звука до 32 бит;

К недостаткам данного подхода можно отнести:

— полная выгрузка аудиоданных в оперативную память, отсутствие сегментации файла;
— отсутствует системная поддержка потокового воспроизведения данных;
— поддержка одного аудиоформата ACC;
— дополнительные требования к стороне сервера для воспроизведения потоковых аудиоданных;
— необходимость в реализации и поддержке буферизации файлов;

1.5.2 AVAudioPlayer

AVAudioPlayer [15] — программный интерфейс, являющийся частью комплекса программных интерфейсов AVFoundation. Он позволяет воспроизводить

потоковые аудиоданные с помощью системного сегментатора.

AVAudioPlayer поддерживает протокол потоковой передачи данных HLS, частоту дискретизации до 48 кГц, а также разрешение звука до 16 бит. Основной особенностью является управление воспроизведением аудио.

Системный сегментатор позволяет обрабатывать пакеты данных при потоковой передаче аудиоданных. Происходит обработка сегментов, сформированных протоколом потоковой передачи данных HLS, которая заключается в получении заголовка фрейма блока данных, который подаётся для воспроизведения. Использование сегментатора не позволяет накладывать звуковые эффекты на аудиопоток, а также использовать системные вызовы для работы с аудиокартой.

Для корректного воспроизведения потокового аудио с помощью программного интерфейса AVAudioPlayer, сторона сервера должна поддерживать передачу аудиоданных в формате ACC по протоколам семейства HLS.

К плюсам данного подхода можно отнести:

— системная поддержка потокового воспроизведение данных;
— поддержка системного сигментатора;

К недостаткам данного подхода можно отнести:

— поддержка аудиофайлов с частотой дискретизации до 48 кГц;
— поддержка аудиофайлов с разрешеним звука до 16 бит;
— поддержка одного аудиоформата ACC;
— дополнительные требовония к стороне сервера для воспроизведения потоковых аудиоданных;

1.5.3 AVAudioEngine

AVAudioEngine [16] — программный интерфейс, являющийся частью комплекса программных интерфейсов AVFoundation, который поддерживает системные вызовы для взаимодействия с звуковой картой с помощью комплекса программных интерфейсов AudioToolbox.

Работа с звуковой картой устройства [17] поддерживает обработку паке-

тов аудиоданных для потокового воспроизведения, различные частоты дискретизации и разрешения звука, наложение звуковых эффектов, работу с нотной матрицей и обработку выделения памяти для хранения сегментов данных цифрового сигнала. При работе с аудиокартой необходимо реализовывать собственный сегментатор, так как системная поддержка стандартов протоколов потоковой передачи данных отсутствует.

Принцип воспроизведения аудиоданных заключается в поочерёдной обработке поступающего аудиосигнала с помощью создаваемых аудиоузлов, которые изменяют звуковой сигнал заданным способом. Создаётся дерево связей аудиоузлов, пройдя по которому на аудиосигнал наложатся звуковые эффекты или изменится метаинформация для воспроизведения аудиофайла.

На рис. 10 представлена схема прохождения аудиосигнала по дереву аудиоузлов.

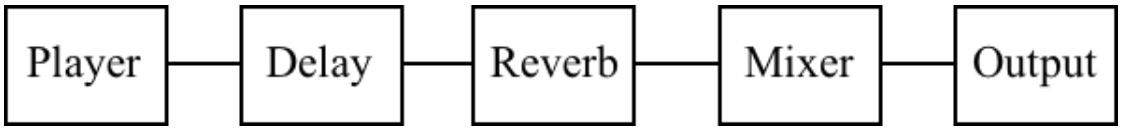

Рис. 10 – Схема прохождения аудиосигнала по дереву аудиоузлов.

AVAudioEngine поддерживает хранение аудиосигнала в PCM формате, т.к. на текущий момент он является стандартизированным решением. Работа с PCM форматом аудиосигнала достигается с помощью создания PCM буфера, который позволяет хранить поступающие потоковое аудиоданные в виде потока байт. На рис. 11 представлена схема расположения фреймов аудиофайла в PCM буфере для работы с AVAudioEngine.

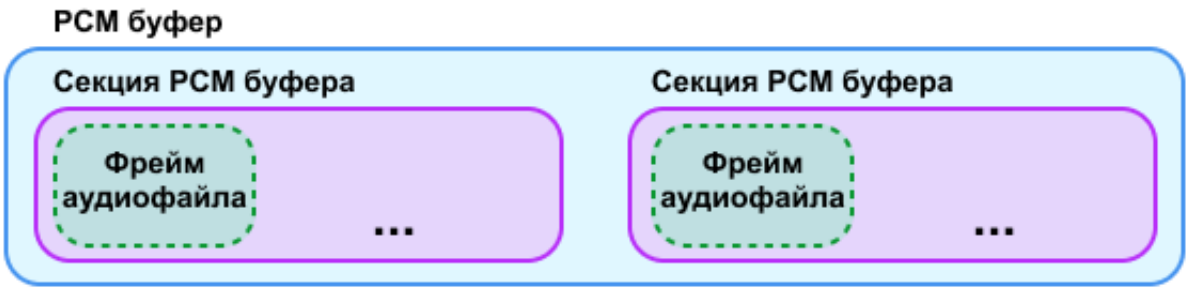

Рис. 11 – Схема расположения фреймов аудиофайла в PCM буфере.

Как подчёркивалось выше, AVAudioEngine поддерживает работу с системными вызовами для обработки поступающего потока аудиоданных. Ниже приведены примеры некоторых из системных вызовов:

— AudioFileStreamOpen — открывает новый анализатор потока аудиоданных;
— AudioFileStreamClose — закрывает анализатор потока аудиоданных;
— AudioFileStreamParseBytes — передает данные потока аудиоданных анализатору;
— AudioFileStreamGetPropertyInfo — получение информации о текущем потоке аудиоданных;

К плюсам данного подхода можно отнести:

— поддержка потокового воспроизведения аудиоданных;
— поддержка работы с системными вызовами;
— поддержка PCM буфера;
— поддержка наложения звуковых эффектов;
— поддержка различных частот дискретизации и разрешений звука;

К недостаткам данного подхода можно отнести:

— отсутствие поддержки системного сегментатора;

1.5.4 Сравнение средств воспроизведения аудиоданных в операционной системе iOS

Сравнение рассмотренных средств воспроизведения аудиоданных в операционной системе iOS рассмотрены в табл. 3. Обозначения:

— 1 — поддержка частоты дискретизации более 48 кГц;
— 2 — поддержка воспроизведения потоковых данных;
— 3 — воспроизведение в режиме реального времени;
— 4 — наложение звуковых эффектов;
— 5 — наличие встроенного сегментатора для обработки потоковых данных;
— 6 — поддержка различных форматов аудиофайлов отличных от ACC;
— 7 — поддержка системных вызовов;
— 8 — поддержка различных протоколов потоковой передачи данных;

Табл. 3 – Сравнение средств воспроизведения аудиоданных в операционной системе iOS

Подход	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
AVPlayer	+	-	-	-	-	-	-	-
AVAudioPlayer	-	+	-	-	+	-	-	-
AVAudioEngine	+	+	+	+	-	+	+	+

По результатам сравнения можно сделать вывод, что для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS лучше всего подходит программный интерфейс AVAudioEngine. Данный программный интерфейс поддерживает все форматы аудиофайлов, работу с системными вызовами для управления звуковым потоком, наложение звуковых эффектов, работу с PCM буфером, а также в отличии AVAudioPlayer (поддерживает только HLS) различные протоколы потоковой передачи данных, но накладывает обязательство в реализации собственного сегментатора аудиопотока.

1.6 Формализация постановки задачи

Основываясь на проведенном анализе предметной области, задачу воспроизведения потокового аудио в мобильном приложении на операционной системе iOS можно сформулировать следующим образом: необходимо спроектировать программно-алгоритмический комплекс, принимающий поток аудиоданных, которые буфиризируются в PCM формат.

В качестве программного интерфейса для обработки аудиосигнала и наложения звуковых эффектов целесообразно использовать AVAudioEngine. Для воспроизведения аудиосигнала осуществить работу с PCM буфером с помощью системных вызовов. Для получения аудиоданных с помощью сетевого запроса необходимо поддержать прикладной протокол HTTP.

Выводы

В аналитическом разделе был проведён анализ предметной области, включающий в себя обзор существующих решений для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS. В ходе анализа была сформулирована задача разработки программно-алгоритмического комплекса, проанализированы существующие методы решения поставленной задачи и предложен вариант её решения.

2 Конструкторская часть

В данном разделе разрабатывается метод воспроизведения потокового аудио в мобильном приложении на операционной системе iOS. Выделяются основные компоненты разрабатываемого программно-алгоритмического комплекса, а также описывается их взаимодействие между собою. Основные алгоритмы представлены в виде схем.

2.1 Формализация задачи

2.1.1 Требования к разрабатываемому методу

Метод воспроизведения потокового аудио в мобильном приложении на операционной системе iOS, который будет использован в разрабатываемом программном комплексе, должен удовлетворять следующим требованиям:

— на вход принимается поток пакетов аудиоданных, отправляемых стороной сервера по протоколу HTTP;
— приходящие аудиоданные должны обрабатываться в PCM формате;
— приходящие аудиоданные должны содержать метаинформацию (заголовок) воспроизводимого файла;
— для наложения звуковых эффектов строится дерево аудиоузлов, задающих правила обработки звукового сигнала;

2.1.2 IDEF0–диаграмма

На рис. 12 и 13 представлены диаграмма IDEF0 метода воспроизведения потокового аудио в мобильном приложении, а также декомпозиция диаграммы IDEF0 на подзадачи A1-A4 соответственно. Описание декомпозированных задач рассмотрено ниже.

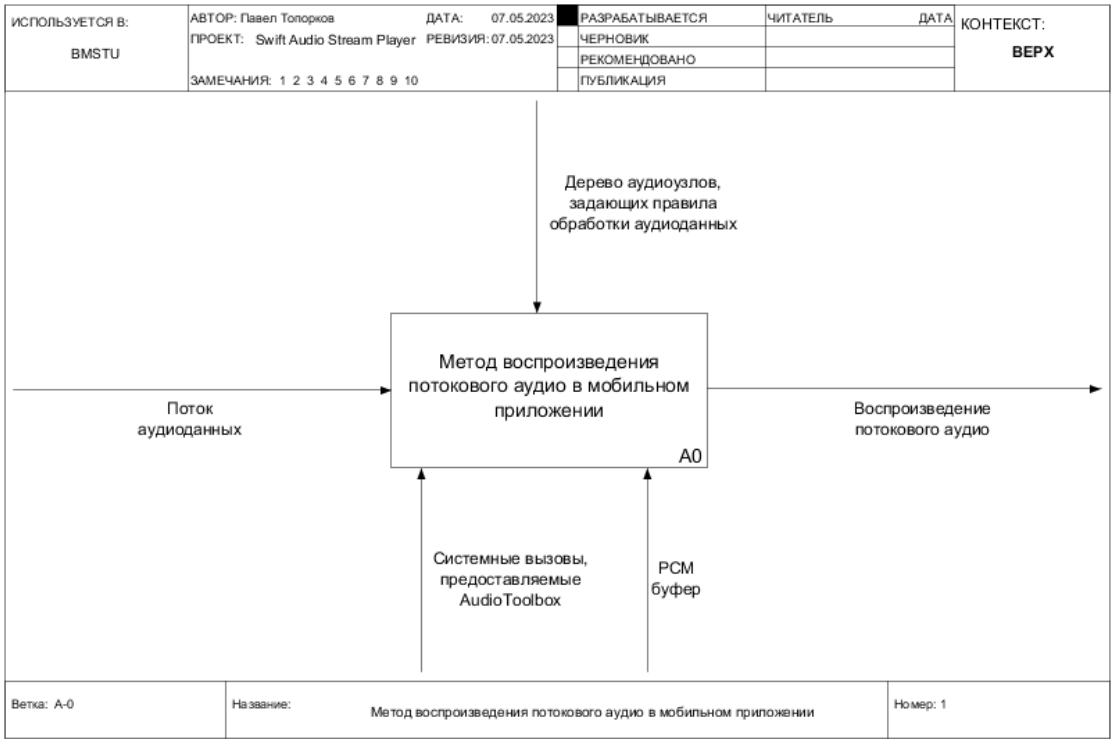

Рис. 12 – Диграмма IDEF0 метода воспроизведения потокового аудио в мобильном приложении.

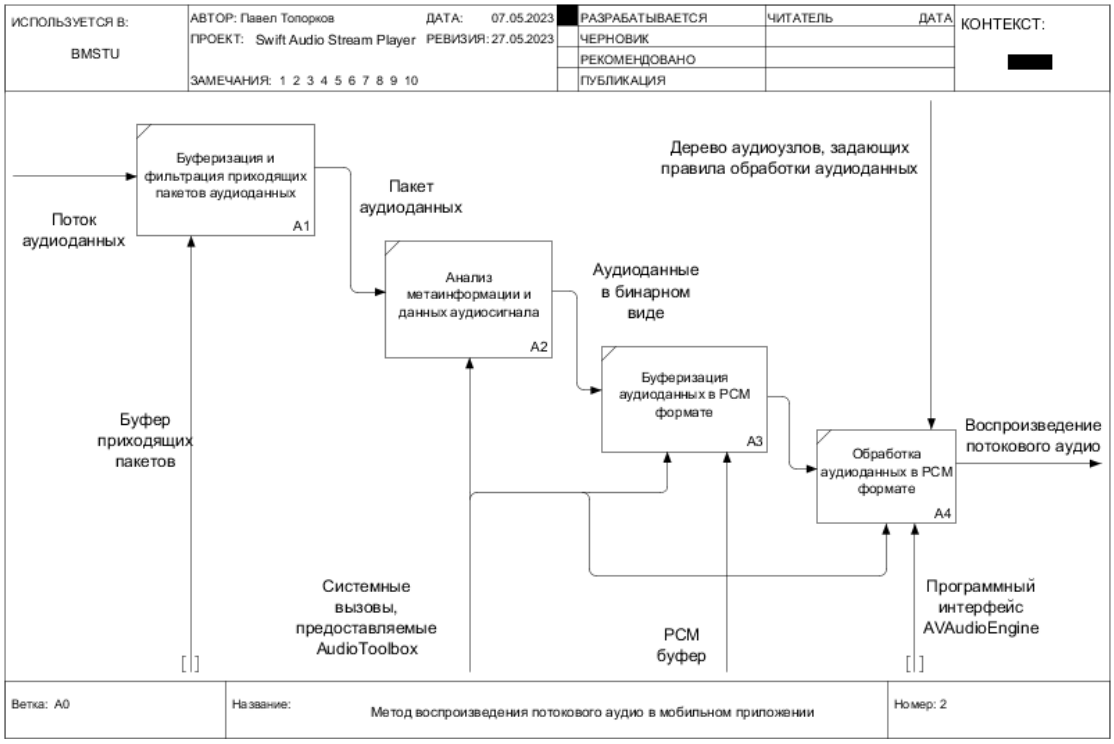

Рис. 13 – Декомпозиция диграммы IDEF0 на подзадачи A1-A4.

2.2 Буферизация и фильтрация приходящих пакетов данных

Для контроля нагрузки ЦПУ мобильного устройства необходимо ограничивать запрос потока аудиоданных. Выделяется специальный механизм буферизации приходящих пакетов данных, а также фильтруется подача аудиоданных обработчику данных. В момент переполнения буфера приходящих пакетов прекращается запрос новых данных, пока в буфере не освободится место. Данное решение продиктовано тем, что при использовании сетевых запросов мобильное устройство интенсивнее использует вычислительные мощности ЦПУ, критическая нагрузка может доходить до $90 - 100\%$ при периодичном сетевом подключении [25].

На рис. 14 представлена схема алгоритма буферизации и фильтрации приходящих пакетов данных.

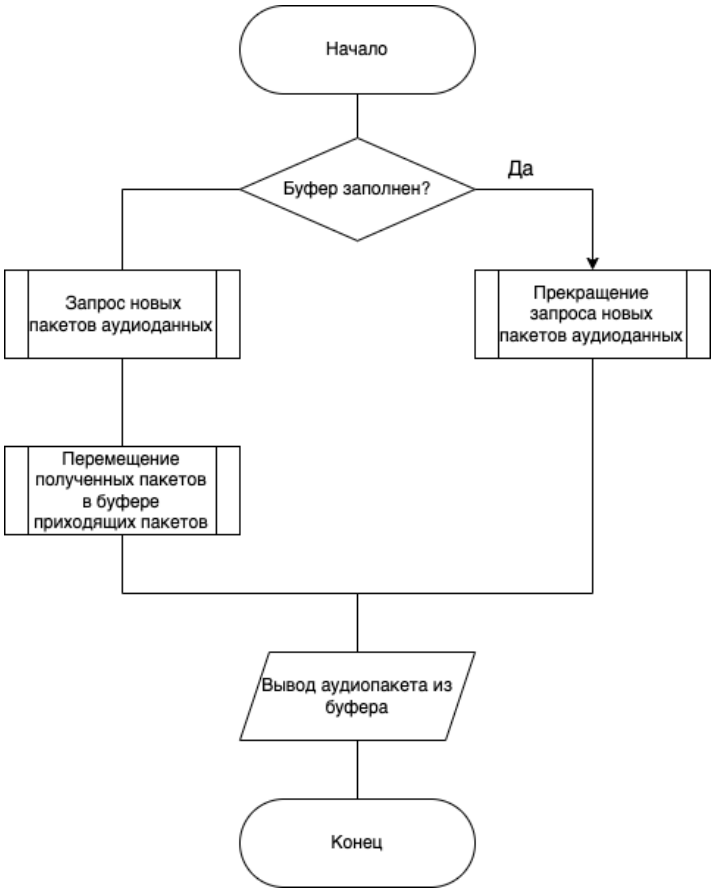

Рис. 14 – Схема алгоритма буферизации и фильтрации приходящих пакетов данных.

2.3 Анализ информации и данных аудиосигнала

Поступаемые аудиоданные необходимо проанализировать, чтобы определить формат оригинального аудиофайла, метаинформацию и данные, содержащие информацию о воспроизведении аудиосигнала.

Для обработки информации об аудиосигнале необходимо определить формат файла. Формат должен содержаться в метаинформации аудиофайла, которая передаётся в первых пришедших пакетах данных. Также необходимо определить частоту дискретизации аудиосигнала и количество фреймов в пакете. На рис. 15 представлена схема анализа информации и данных аудиосигнала.

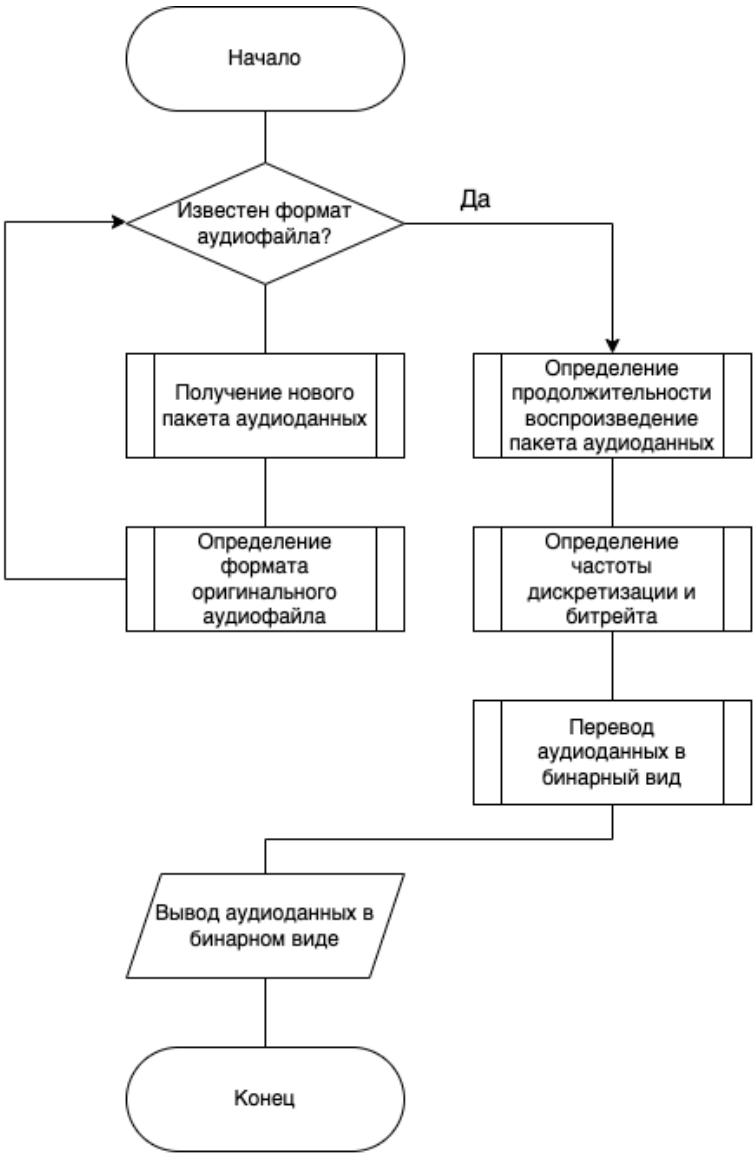

Рис. 15 – Схема анализа информации и данных аудиосигнала.

Для получения информации о звуковом сигнале необходимо из получаемого пакета аудиоданных отделить метаинформацию, а также заголовки сегментов от данных аудиосигнала. На рис. 16 представлены линейные представления аудиофайла, а также сегмента (фрейма) аудиоданных файла с расположением метаданных, заголовков и данных аудиосигнала.

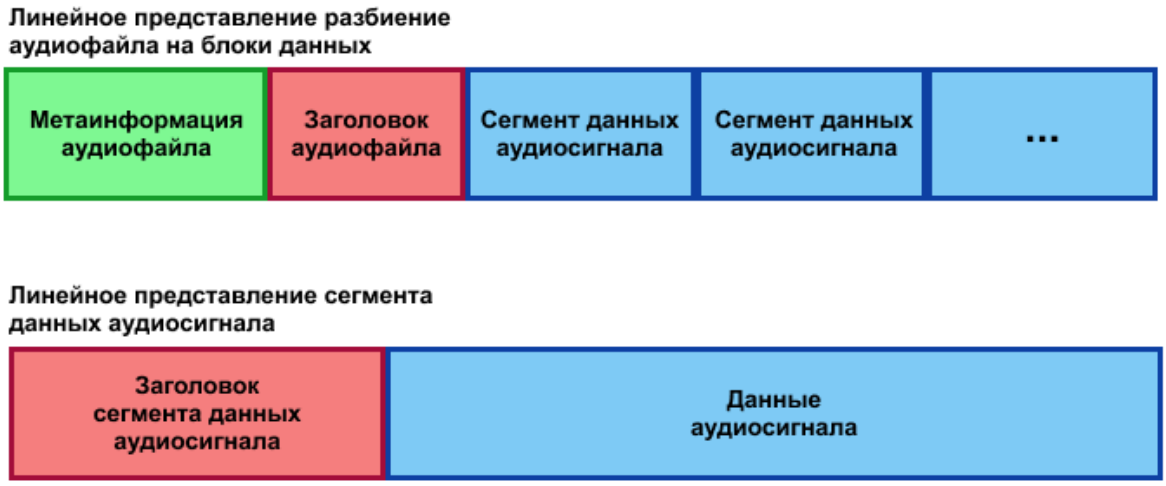

Рис. 16 – Линейные представления аудиофайла, а также сегмента (фрейма) аудиоданных файла.

Важной информацией для обработки аудио является продолжительность воспроизведения аудио, полученного из пришедшего пакета данных. Некоторые аудиофайлы могут не содержать продолжительность воспроизведения в своих метаданных. В таком случае необходимо определить продолжительность сигнала в аудиопакете самостоятельно с учётом возможного переменного битрейта. Продолжительность воспроизведения можно получить по формуле:

$$t = \frac{b}{f},$$

где

t — продолжительность воспроизведения;

n — количество фреймов в пакете аудиоданных;

f — частота дискретизации.

2.4 Буферизация аудиоданных в PCM формате

Аудиоданные в бинарном виде, полученные с помощью анализа данных аудиосигнала, буферизуются в PCM формате. Для этого создаётся специальный, системный PCM буфер, который хранит фреймы аудиоданных, предназначенных для текущего воспроизведения. После заполнения PCM буфера необходимо воспользоваться системными вызовами для воспроизведения аудиосигнала. Размер PCM буфера определяется в зависимости от разрешения звука. Ниже указано количество фреймов аудиоданных, содержащихся в PCM буфере в зависимости от разрешения звука:

— для низкого разрешения звука — 4096 фреймов;

— для высокого разрешения звука — 8192 фрейма;

На рис. 17 представлена схема буферизации аудиоданных в PCM формате.

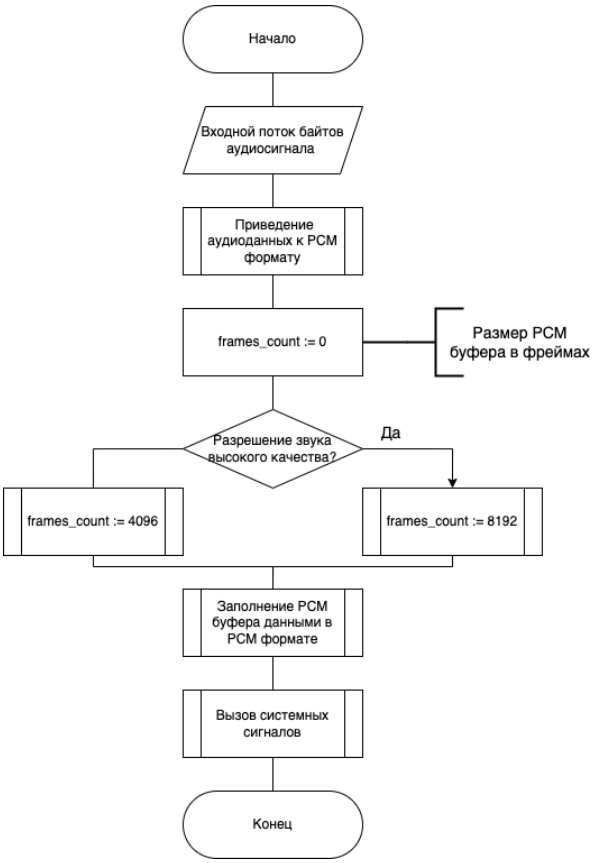

Рис. 17 – Схема буферизации аудиоданных в PCM формате.

2.5 Обработка аудиоданных в PCM формате

После помещения аудиоданных в PCM буфер необходимо начать процесс воспроизведения аудиоданных. Для этого с помощью системного вызова AudioFileStreamOpen нужно инициировать процесс анализа аудиоданных.

Также необходимо создать дерево аудиоузлов обработки аудиосигнала, которые наложат звуковые эффекты на воспроизводимый аудиосигнал. При старте воспроизведения нужно связать программный интерфейс AVAudioEngine с построенным деревом аудиоузлов. AVAudioEngine становится частью дерева аудиоузлов обработки, отвечая за определения характеристик музыкального сигнала. В случае ошибки обработки аудиосигнала вызовется системное прерывание.

На рис. 18 представлена схема обработки аудиоданных в PCM формате.

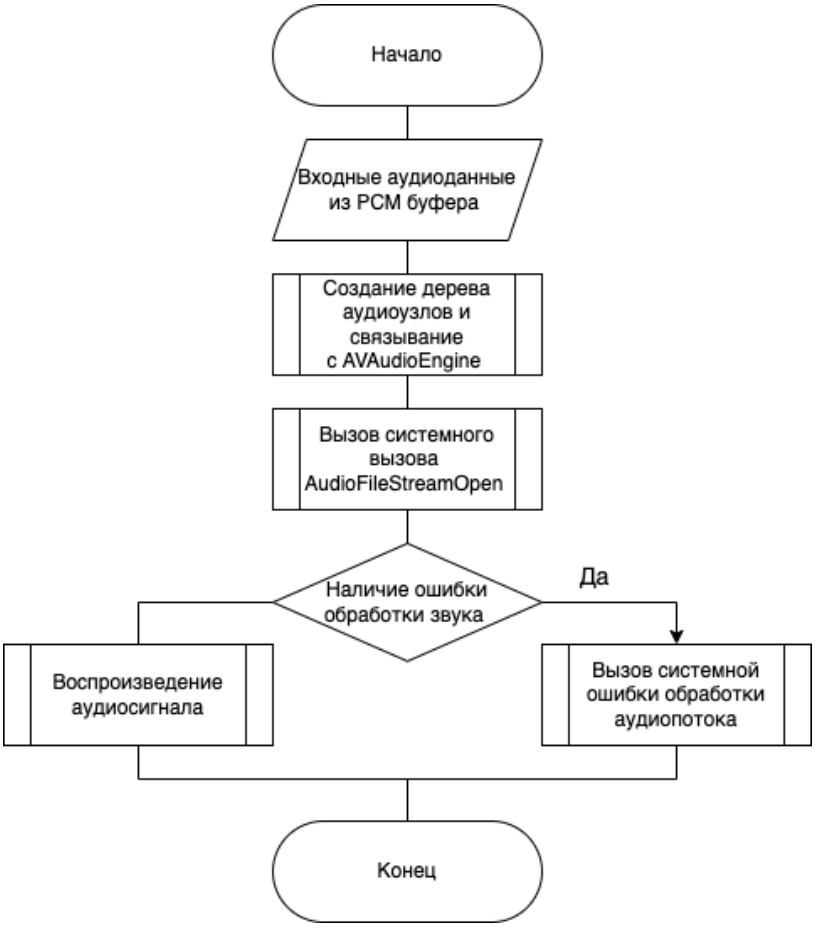

Рис. 18 – Схема обработки аудиоданных в PCM формате.

2.6 Основные компоненты разрабатываемого программного комплекса

Из вышеизложенного следует, что к основным компонентам разрабатываемого программно-алгоритмического комплекса необходимо отнести следующие:

— компонент сетевых запросов;

— компонент управления потоком данных;

— компонент обработки пакетов;

— компонент конвертации пакетов аудиоданных;

— компонент воспроизведения аудиоданных;

Компонент сетевых запросов отвечает за получение новых пакетов аудиоданных со стороны сервера. Он определяет заголовки запросов данных, устанавливает HTTP соединение.

Сетевой компонент получает данные в чистом (бинарном) виде, поддерживая постоянное соединение с сервером для быстрого получения новых пакетов аудиоданных при необходимости. При приостановке или разрыве соединения, данный компонет должен запомнить параметры, тип и статус соединения для его востановления при новом запросе аудиопакетов.

Компонент управления потоком данных получает данные от компонента сетевых запросов и определяет необходимость запроса новых аудиопакетов.

С его помощью решается задача буферизации и фильтрации приходящих пакетов данных: построение буфера приходящих пакетов данных, контроль переполнения буфера, буферизация новых пакетов данных. Данный компонет можно рассматривать как пропускающий механизм пакетов, передаваемых компоненту обработки пакетов.

Компонент обработки пакетов решает задачу анализа информации и данных аудиосигнала. При получении сетевых данных выполняется анализ следующей находящейся в них информации:

— метаинформация аудиофайла;

— заголовок аудиофайла;
— сегменты аудиосигнала;
— заголовки сегментов аудиосигнала;
— данные аудиосигнала, находящиеся в сегментах аудиофайла;

Главной задачей данного компонента является получение данных аудиосигнала, а также выявление метаинформации для корректного воспроизведения аудиосигнала.

Обработанные пакеты необходимо конвертировать в формат, подходящий для воспроизведения аудиосигнала. На основе полученных метаинформации и формата определяется размер создаваемого PCM буфера и происходит его заполнение данными аудиосигнала. Полученная метаинформация позволяет сообщить системе о характеристиках звукового сигнала, продолжительности воспроизведения, а также обработать информацию для управления потоком воспроизведения. Следует отметить, что компонент конвертации пакетов аудиоданных решает задачу буферизации аудиоданных в PCM формате.

Компонент воспроизведения аудиоданных решает задачу обработки аудиоданных в PCM формате. С помощью созданного дерева аудиоузлов, предназначенных для наложения звуковых эффектов, а также AVAudioEngine происходит настройка системы для начала воспроизведения аудиосигнала. Для анализа аудиоданных, предназначенных для воспроизведения с помощью AVAudioEngine, необходимо воспользоваться системными вызовами.

2.6.1 Взаимодействие компонентов разрабатываемого программного комплекса

Компоненты разрабатываемого программно-алгоритмического комплекса должны взаимодействовать по следующим правилам:

— компонент сетевых запросов адресует полученные аудиопакеты компоненту управления потоком данных, если буфер приходящих пакетов данных не переполнен;
— компонент управления потоком отправляет буферизированные пакеты

компоненту обработки пакетов;

— компонент конвертации пакетов аудиоданных запрашивает информацию из обработанных компонентом обработки пакетов сетевых данных;

— компонент воспроизведения аудиоданных запрашивает конвертированные аудиоданные из компонента конвертации пакетов;

На рис. 19 представлена схема взаимодействия компонентов.

Рис. 19 – Схема взаимодействия компонентов.

Вывод

В данном разделе был разработан метод воспроизведения потокового аудио в мобильном приложении на операционной системе iOS. Были выделены основные компоненты разрабатываемого программно-алгоритмического комплекса, а также описано их взаимодействие между собою. Приведены IDEF0 диаграмма, её декомпозиция, а также основные алгоритмы в виде схем.

3 Технологическая часть

В данном разделе обосновывается выбор средств программной реализации. Описываются основные особенности реализации. Приводятся примеры работы программно-алгоритмического комплекса.

3.1 Выбор средств реализации программно-алгоритмического комплекса

3.1.1 Выбор языка программирования

В качестве языка программирования для реализации программного комплекса был выбран Swift. Выбор языка обусловлен следующими причинами:

— программный интерфейс AudioToolbox написан на языке программирования Swift;

— наличие механизма подсчёта ссылок ARC, который автоматически деалоцирует неиспользуемые объекты памяти;

— наличие достаточного опыта программирования на данном языке, что позволит сократить время разработки программного комплекса;

3.1.2 Выбор среды разработки

В качестве среды разработки была выбрана интегрированная среда разработки Xcode [27] по следующим причинам:

— поддержка компиляции программного обеспечения для симуляторов мобильных устройств на операционной системе iOS для тестирования разрабатываемого программного комплекса;

— наличие отладчика, поддерживающего многопоточную работу алгоритмов;

— наличие анализатора состояния ЦПУ, ОЗУ, сетевой нагрузки мобильных устройств и симуляторов на операционной системе iOS;

— наличие механизма кэширования объектных файлов, который сокращает повторную компиляцию программного обеспечения;

— наличие встроенной системы сборки программного обеспечения;

3.2 Диаграмма классов реализуемого программно-алгоритмического комплекса

На рис. 20 представлена диаграмма классов реализуемого программного комплекса.

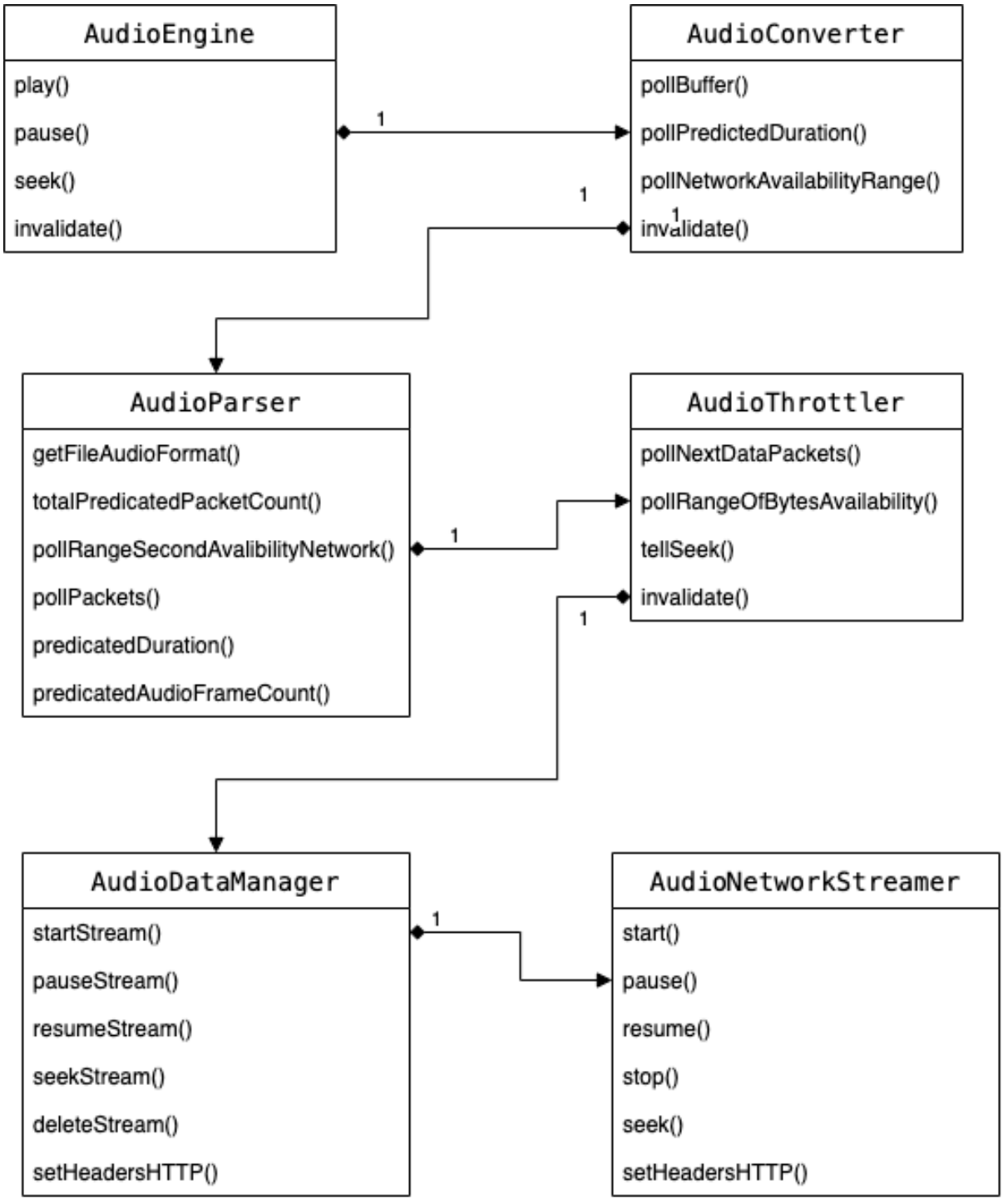

Рис. 20 – Диаграмма классов реализуемого программного комплекса.

3.3 Получение сетевых данных

Класс AudioDataManager реализует компонент сетевых запросов. С его помощью конфигурируются заголовки HTTP запроса, определяется директория файловой системы устройства для сохранения полученных данных в временных файлах, а также создаётся, прерывается, восстанавливается и очищается поток сетевых данных. Интерфейс данного класса представлен в листинге 1.

Листинг 1: Интерфейс класса AudioDataManager.

```
protocol AudioDataManager {
    var numberOfQueued: Int { get }
    var allowCellular: Bool { get set }

    func setHTTPHeaderFields(_ fields: [String: String]?)
    func setDownloadDirectory(_ dir: FileManager.SearchPathDirectory)
    func attach(callback: @escaping (_ id: ID, _ progress: Double)->())

    func startStream(withRemoteURL url: AudioURL, callback: @escaping
        (StreamProgressPTO) -> ())
    func pauseStream(withRemoteURL url: AudioURL)
    func resumeStream(withRemoteURL url: AudioURL)
    func seekStream(withRemoteURL url: AudioURL, toByteOffset offset:
        UInt64)
    func deleteStream(withRemoteURL url: AudioURL)
}
```

Для создания и обработки сетевого запроса применяется композиционный подход: создаётся класс AudioNetworkStreamer, являющийся частью класса AudioDataManager. Класс AudioNetworkStreamer получает сетевые данные, проводит валидацию создаваемого объекта запроса, устанавливает допустимое время обработки запроса, а также проксирует полученные данные классу AudioDataManager. Создание и отправка запроса стороне сервера для получения потоковых аудиоданных происходит в методе start.

Интерфейс класса AudioNetworkStreamer, а также реализация метода start представлены в листингах 2 и 3 соответственно.

Листинг 2: Интерфейс класса AudioNetworkStreamer.

```
protocol AudioNetworkStreamer {
    var HTTPHeaderFields: [String: String]? { get set }
    func start(withID id: ID, withRemoteURL url: URL, withInitialData data:
        Data?, andTotalBytesExpectedPreviously previousTotalBytesExpected:
        Int64?)
    func pause(withId id: ID)
    func resume(withId id: ID)
    func stop(withId id: ID)
    func seek(withId id: ID, withByteOffset offset: UInt64)
}
```

Листинг 3: Реализация метода start класса AudioNetworkStreamer.

```
func start(withID id: ID, withRemoteURL url: URL, withInitialData data:
    Data? = nil, andTotalBytesExpectedPreviously previousTotalBytesExpected:
    Int64? = nil) {
    killPreviousTaskIfNeeded()
    guard let data = data else {
        var request = URLRequest(url: url)
        HTTPHeaderFields?.forEach { request.setValue($1,
            forHTTPHeaderField: $0) }
        task = session.dataTask(with: request)
        task?.taskDescription = id
        return task?.resume()
    }
    var request = URLRequest(url: url, cachePolicy:
        .useProtocolCachePolicy, timeoutInterval: TIMEOUT)
    HTTPHeaderFields?.forEach { request.setValue($1, forHTTPHeaderField:
        $0) }
    request.addValue("bytes=\(data.count)-", forHTTPHeaderField: "Range")
    task = session.dataTask(with: request); task?.taskDescription = id
    initialDataBytesCount = Int64(data.count)
    totalBytesExpectedForWholeFile = previousTotalBytesExpected
    let progress = previousTotalBytesExpected != nil ?
        Double(initialDataBytesCount)/Double(previousTotalBytesExpected!) : 0
    let dto = StreamProgressDTO(progress: progress, data: data,
        totalBytesExpected: totalBytesExpectedForWholeFile)
    progressCallback(id, dto); task?.resume()
}
```

3.4 Управление потоком сетевых данных

Для управления потоком аудиоданных определён класс AudioThrottler. Интерфейс класса AudioThrottler предстален в листинге 4. Его интерфейс определяет следующие методы:

— pullNextDataPacket — отправка буферизированных данных компоненту обработки пакетов;

— tellSeek — определение размера и валидация полученных аудиопакетов;

— pollRangeOfBytesAvailable — вычисление смещения аудиоинформации в битовой форме, находящийся в полученных данных;

— invalidate — очистка буфера приходящих аудиопакетов и прекращение прослушивания сетевой информации;

Листинг 4: Интерфейс класса AudioThrottler.

```
1 protocol AudioThrottler {
2     func pullNextDataPacket(_ callback: @escaping (Data?) -> ())
3     func tellSeek(offset: UInt64)
4     func pollRangeOfBytesAvailable() -> (UInt64, UInt64)
5     func invalidate()
6 }
```

Логика получения сетевых данных вынесена за пределы публичного интерфейса и определяется при инициализации класса. Создаётся связь между компонентом управления потоком сетевых данных (AudioThrottler) и компонентом получения данных (AudioDataManager) с помощью поведенческого шаблона проектирования «Издатель-подписчик».

Обработчиком данных, полученных из класса AudioThrottler является метод handlerStream. Его реализация представлена в листинге 5. Алгоритм данного метода включает пять пунктов:

1. установка связи (подписка) с классом AudioDataManager для получения сетевых данных;
2. определение ожидаемого количества байт сетевой информации, если она пришла;

3. проверка размера информации (из первого пункта), а также переполнения буфера приходящих аудиопакетов;
4. заполнение буфера новыми сетевыми данными;
5. уведомление компонента обработки пакетов о получении новых сетевых данных;

Листинг 5: Реализация метода handlerStream.

```
1  func handlerStream(withRemoteUrl url: AudioURL) {
2      AudioDataManager.shared.startStream(withRemoteURL: url) { [weak self]
           pto in
3          guard let self = self else { return }
4          if let totalBytesExpected = pto.getTotalBytesExpected() {
5              self.totalBytesExpected = totalBytesExpected
6          }
7
8          self.queue.async { [weak self] in
9              guard isAvailablePull(pto) else { return }
10             self?.networkData.append(pto.getData())
11             StreamingDownloadDirector.shared.didUpdate(url.key,
                   networkStreamProgress: pto.getProgress())
12         }
13     }
14 }
```

3.5 Обработка аудиоданных

Для получения метаинформации и данных аудиосигнала определён класс AudioParser. Его интерфейс представлен в листинге 6. Данный класс получает и обрабатывает информацию о формате аудиофайла, количестве полученных пакетов, продолжительности воспроизведения аудиосигнала, пришедшего в сетевых данных, количестве фреймов аудиофайла.

Для вычисления продолжительности воспроизведения аудиосигнала и количества фреймов, находящихся в аудиопакетах, а также для их приведения к системным типам программного интерфейса AudioToolbox определены вычисляемые свойства predictedDuration и totalPredictedAudioFrameCount. Реализации данных методов представлены в листинге 7.

Листинг 6: Интерфейс класса AudioParser.

```
protocol AudioParser {
    var fileAudioFormat: AVAudioFormat? { get }
    var totalPredictedPacketCount: AVAudioPacketCount { get }
    var totalPredictedAudioFrameCount: AUAudioFrameCount?
    var predictedDuration: Duration?

    func pollRangeOfSecondsAvailableFromNetwork() -> (Needle, Duration)
    func pullPacket(atIndex index: AVAudioPacketCount) throws ->
        (AudioStreamPacketDescription?, Data)
    func invalidate()
}
```

Листинг 7: Реализация методов вычисления продолжительности воспроизведения и количества фреймов.

```
var predictedDuration: Duration? {
    guard let sampleRate = fileAudioFormat?.sampleRate else {
        return nil
    }

    guard
        let totalPredictedFrameCount = totalPredictedAudioFrameCount
    else {
        return nil
    }

    return Duration(totalPredictedFrameCount)/Duration(sampleRate)
}

var totalPredictedAudioFrameCount: AUAudioFrameCount? {
    let framesPerPacket =
        fileAudioFormat?.streamDescription.pointee.mFramesPerPacket
    guard let framesPerPacket = framesPerPacket else {
        return nil
    }

    return AVAudioFrameCount(totalPredictedPacketCount) *
        AVAudioFrameCount(framesPerPacket)
}
```

Для конвертации и буферизации аудиоданных в PCM буфере определён класс AudioConverter. Процесс буферизации данных происходит в методе класса pullBuffer. Реализация метода pullBuffer представлена в листинге 8. Рассмотрим алгоритм метода:

1. создание PCM буфера, основанное на формате аудиофайла и количестве фреймов;
2. определяется количество фреймов для каждого аудиопакета;
3. количество пакетов, передающихся в буфер высчитывается как отношение общего количества фреймов к количеству фреймов в одном пакете данных;
4. данные приводятся в поток байт;
5. проверяется ошибка буферизации;

Листинг 8: Реализация метода pullBuffer.

```
1  func pullBuffer() throws -> AVAudioPCMBuffer {
2      guard let converter = converter else {
3          throw ConverterError.cannotCreatePCMBufferWithoutConverter
4      }
5      let pcmBuffer = AVAudioPCMBuffer(pcmFormat: engineAudioFormat,
           frameCapacity: pcmBufferSize)
6      pcmBuffer.frameLength = pcmBufferSize
7      return try queue.sync { _ in
8          let framesPerPacket =
               engineAudioFormat.streamDescription.mFramesPerPacket
9          var countPacketsForBufferToFill = pcmBuffer.frameLength /
               framesPerPacket
10         let context = unsafeBitCast(self, to: UnsafeMutableRawPointer.self)
11         let status = AudioConverterFillComplexBuffer(converter,
               ConverterListener, context, &countPacketsForBufferToFill,
               pcmBuffer.mutableAudioBufferList, nil)
12         guard status == noErr else { throw status.error }
13         return pcmBuffer
14     }
15 }
```

3.6 Воспроизведение аудиоданных

Класс AudioEngine позволяет инициировать воспроизведение аудиоданных. В результате вызова методов данного класса происходит инициализация и настройка AVAudioEngine. Для поддержки эффекта ускоренного вопроизведения создаётся список программных аудиоузлов, которые связываются с программным интерфейсом. Данная логика представлена в листинге 9.

Корректная обработка аудиоданных со стороны операционной системы осуществляется определением состояния механизма аудиосессии. Для его настройки используется класс AVAudioSession, предоставляемый комплексом программных интерфейсов AVFoundation. Это позволяет сообщить операционной системе о необходимости воспроизведения аудиосигнала независимо от состояния жизненного цикла программного обеспечения, которое инициировало операцию воспроизведения.

Листинг 9: Создание списка программных аудиоузлов.

```
func initAudioNodes(engine: AVAudioEngine, _ format: AVAudioFormat) {
    engine.attach(playerNode)
    audioModifiers = SAPlayer.shared.audioModifiers
    defer { engine.prepare() }
    guard let audioModifiers, audioModifiers.count > 0 else {
        engine.connect(playerNode, to: engine.mainMixerNode, format: format)
        return
    }
    audioModifiers.forEach { engine.attach($0) }
    let node = audioModifiers[i]; var i = 1
    engine.connect(playerNode, to: node, format: engineAudioFormat)
    while i < audioModifiers.count {
        let lastNode = audioModifiers[i - 1]
        let currNode = audioModifiers[i]
        engine.connect(lastNode, to: currNode, format: engineAudioFormat)
        i += 1
    }
    let finalNode = audioModifiers[audioModifiers.count - 1]
    engine.connect(finalNode, to: engine.mainMixerNode, format: format)
}
```

3.7 Пример работы программно-алгоритмического комплекса

Для демонстрации спроектированного и разработанного программного комплекса был реализован пользовательский интерфейс, позволяющий воспроизвести потоковые аудиоданные, предоставляемые серверной частью с использованием прикладного протокола передачи данных HTTP. Пользовательский интерфейс представлен на рис. 21.

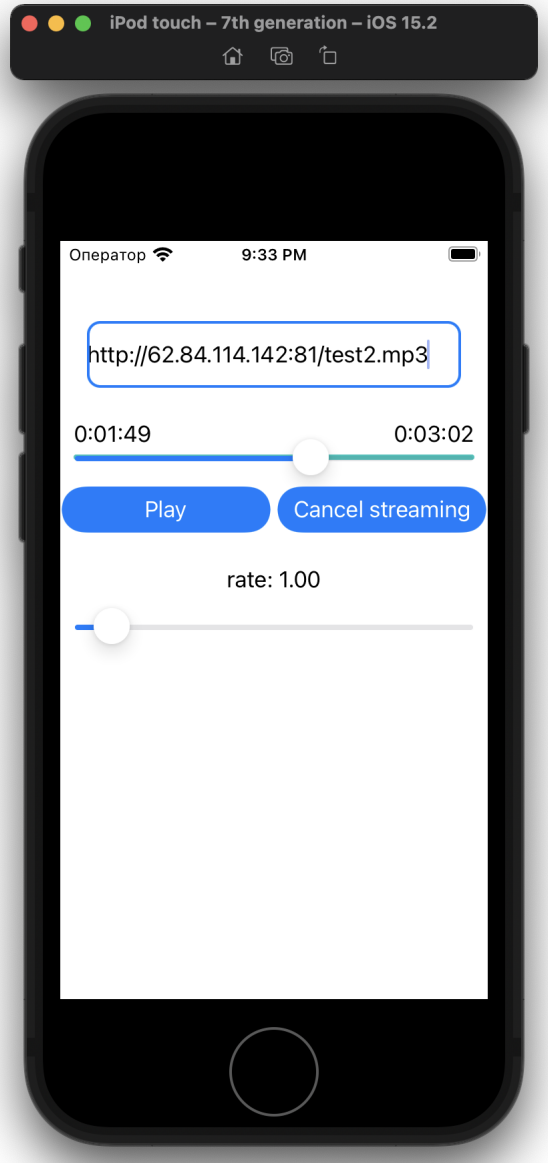

Рис. 21 – Пользовательский интерфейс, для демонстрации функционирования разработанного программного комплекса.

Для воспроизведения потоковых аудиоданных необходимо в поле ввода передать url адрес, по которому хранится аудиофайл. В процессе получения потока данных становится доступно воспроизведение аудиосигнала с возможностью изменения скорости воспроизведения: коэффициент ускорения может принимать значения от 0.1 до 16.

Реализованы возможность изменять смещение воспроизведения аудиосигнала (изменять начальное время воспроизведения), возможность остановки и восстановления воспроизведения.

В процессе получения и обработки сетевых данных происходит логирование процессов и состояний программного комплекса. Пример логирования представлен в листинге 10.

Листинг 10: Логирование процессов и состояний программного комплекса.

```
[AudioEngine] init system
[AudioEngine] success make AudioNodes
[AudioDataManager] fetch data bytes: 256
[AudioThrottler] pull throttler buffer
[AudioEngine] start stream network data
[AudioParser] get metadata
[AudioParser] get fileformat: mp3
[AudioParser] get duration: 12987
[AudioParser] get metadata
[AudioParser] get fileformat: mp3
[AudioParser] get duration: 11487
[AudioConverter] success convert packets to PCM format
[AudioConverter] pull PCM buffer
[AudioConverter] pull PCM buffer
[AudioConverter] pull PCM buffer
[AudioEngine] play audio data
[AudioEngine] edit rate: 0.5x
[AudioEngine] edit rate: 0.52x
[AudioEngine] edit rate: 0.73x
[AudioEngine] edit rate: 1.43x
[AudioEngine] seek to 1:12
[AudioEngine] seek to 0:32
[AudioEngine] pause audio data
```

Вывод

В данном разделе были описаны средства программной реализации, представлены листинги реализации компонентов программного комплекса и отмечены особенности их реализации. Приведён пример работы программного комплекса.

4 Исследовательская часть

В данном разделе проводится оценка результатов обработки потоковых аудиоданных с помощью разработанного программного комплекса. Исследуется зависимость времени обработки аудиоданных, количества выделенной оперативной памяти для их обработки и вычислительной нагрузки ЦПУ мобильного устройства, от наличия буферизации компонентом управления потоком данных. Сравнивается количество выделенной оперативной памяти и затраченного времени для обработки аудиоданных с существующими аналогами.

4.1 Исследование влияния буферизации компонентом управления потоком данных

В конструкторской части было рассмотрено, что при периодическом сетевом подлючении критическая нагрузка ЦПУ может достигать $90 - 100\%$. Для решения данной проблемы был выделен компонент управления потоком данных, который фильтрует и буферизует сетевые пакеты данных. Также должно снизиться количество выделяемой оперативной памяти и время для обработки аудиопакетов. Чтобы потвердить данное утверждение необходимо провести соответствующий эксперимент.

Получение объёма выделенной памяти, времени обработки данных и вычислительной нагрузки ЦПУ производилось с помощью Xcode Report Debuger. Для каждой из вышеперечисленных характеристик было произведено 10 замеров и взято среднее значение. Замеры проводились с использованием компонента управления потоком данных и без его участия.

Для потоковой передачи аудиоданных был выбран аудиофайл со следующими характеристиками:

— MP3 формат аудиофайла;

— продолжительность аудио 14400 сек.;

— разрешение звука 16 бит;

— частота дискретизации 48 кГц;

На рис. 22 представлено сравнение объёма выделенной оперативной памяти при потоковом получении аудиоданных с использованием компонента управления потоком данных и без его участия.

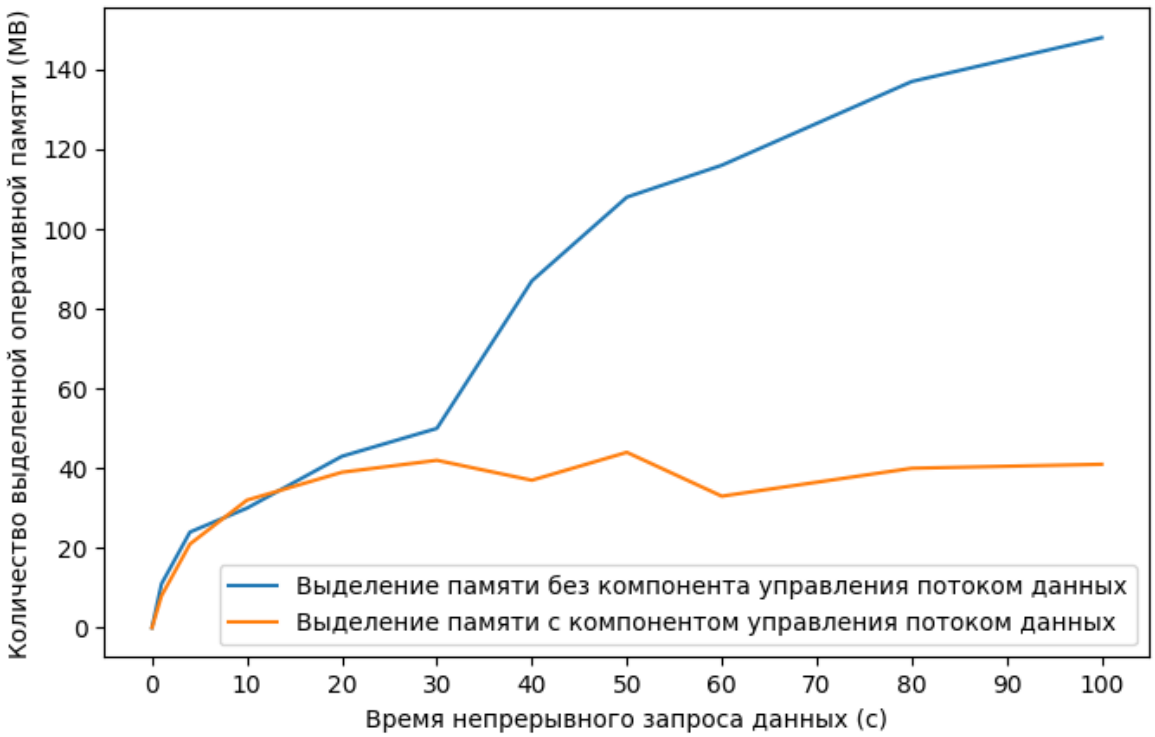

Рис. 22 – Сравнение объёма выделенной оперативной памяти при потоковом получении аудиоданных с использованием компонента управления потоком данных и без его участия.

На графике видно, что зависимость выделения оперативной памяти для обработки аудиоданных от времени без использования компонента управления потоком данных при периодическом сетевом подключении неубывающая. При использовании буферизации и фильтрации компонента объём выделенной памяти не привышает 44 Мб. Следует, что использование механизмов буферизации и фильтрации значительно снижает объём аудиоданных в оперативной памяти мобильного устройства.

Для замеров времени обработки сетевых аудиоданных необходимо также замерять время нахождения (простоя) сетового пакета после его получения.

На рис. 23 представлен график сравнения времени обработки сетевых аудиоданных с использованием компонента управления потоком данных и без его участия.

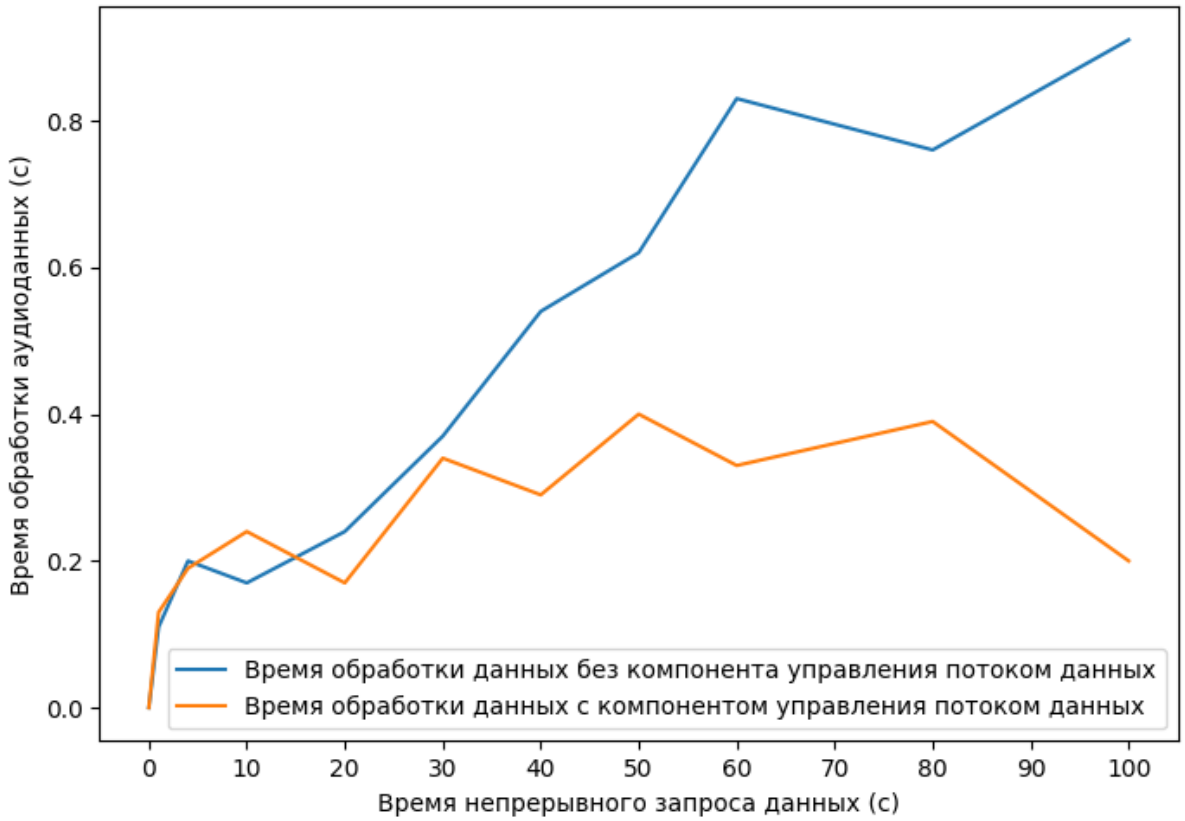

Рис. 23 – Сравнение времени обработки сетевых аудиоданных с использованием компонента управления потоком данных и без его участия.

Время обработки сетевых аудиоданных без использования компонента управления потоком возрастает. Это связано с простоем сетевых пакетов в очереде на обработку данных аудиосигнала. При использовании буферизации и фильтрации время обработки не превышает 0.4 секунд.

Для замеров нагрузки ЦПУ необходимо взять критическое значение (максимальную нагрузку) в процентах, которая потребляется мобильным приложением на операционной системе iOS, использующим разработанный программный комплекс. На рис. 24 представлен график сравнения максимальной нагруз-

ки ЦПУ с использованием компонента управления потоком данных и без его участия.

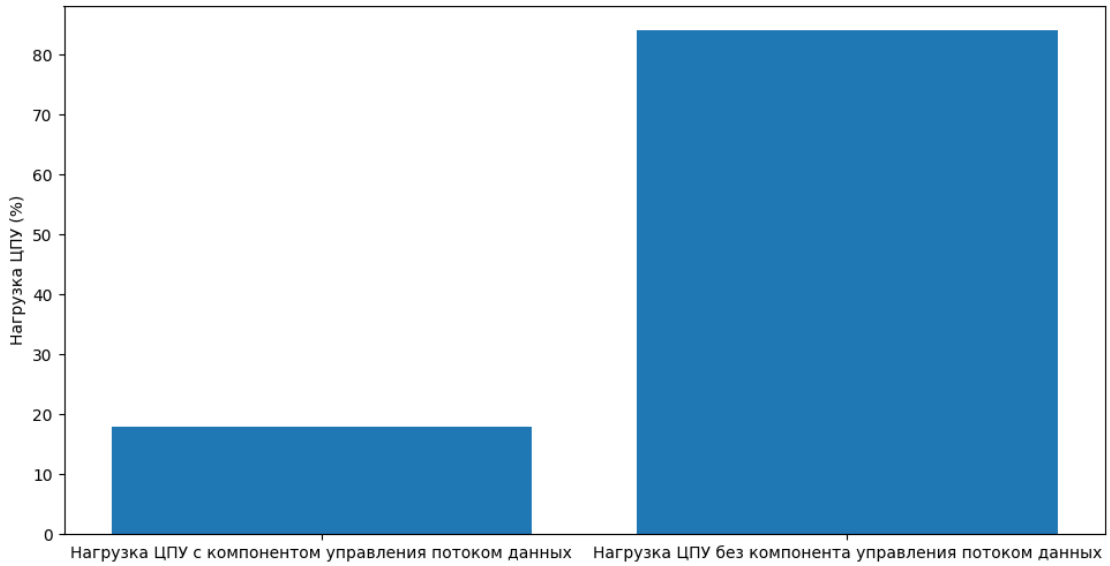

Рис. 24 – График сравнения максимальной нагрузки ЦПУ с использованием компонента управления потоком данных и без его участия.

Из графика видно, что без использования буферизации и фильтрации максимальная нагрузка ЦПУ доходит до 84%. Использование компонента управления потоком данных позволило снизить максимальную нагрузку ЦПУ до 18%.

Опираясь на результаты проведённых замеров, можно сделать вывод, что с помощью компонента управления потоком данных удалось:

— снизить нагрузку ЦПУ в 4.7 раз;
— избавиться от возрастающего времени обработки данных;
— избавиться от возрастающей зависимости выделения оперативной памяти от времени;
— ограничить время обработки данных и количество выделенной оперативной памяти до 44 Мб и 0.4 сек. соответсвенно;

4.2 Сравнение времени и выделения оперативной памяти для обработки данных с существующими аналогами

Произведён сравнительный анализ разработанного программного комплекса с существующими аналогами. В качестве аналогов были взяты: программный интерфейс AVAudioPlayer, программное обеспечение SwiftAudioPlayer. Данный выбор обусловлен следующими причинами:

— поддержка воспроизведения потокового аудио;
— полностью или частично использован комплекс программных итерфейсов AVFoundation;
— поддержка разрешения звука 16 бит;

В качестве сравниваемых характеристик были выбраны:

— среднее время обработки аудиоданных;
— среднее количество выделяемой оперативной памяти для обработки и воспроизведения аудиосигнала;

Для потоковой передачи аудиоданных был выбран аудиофайл со следующими характеристиками:

— MP3 формат аудиофайла;
— продолжительность аудио 360 сек.;
— разрешение звука 16 бит;
— частота дискретизации 48 кГц;

На рис. 25 и 26 представлены графики сравнения среднего времени обработки аудиоданных и среднее количество выделяемой оперативной памяти разработанного программного комплекса и вышеперечисленных аналогов.

На основании графиков можно сделать вывод, что среднее время обработки аудиоданных разработанного программного комплекса и AVAudioPlayer практически одинаково. Прирост времени при использовании AVAudioPlayer составляет 4%, что является незначительным. Среднее время обработки аудиоданных с помощью SwiftAudioPlayer больше, чем у разработанного программ-

ного комплекса на 20%.

Разработанный программный комплекс имеет наименьшее потребление оперативной памяти для обработки сетевых аудиоданных. AVAudioPlayer выделяет на 14% больше памяти, а SwiftAudioPlayer на 21%. Это связано с тем, что они используют системный сегментатор аудиопакетов, который хранит приходящие аудиопакеты даже после воспроизведения содержащегося в них аудиосигнала.

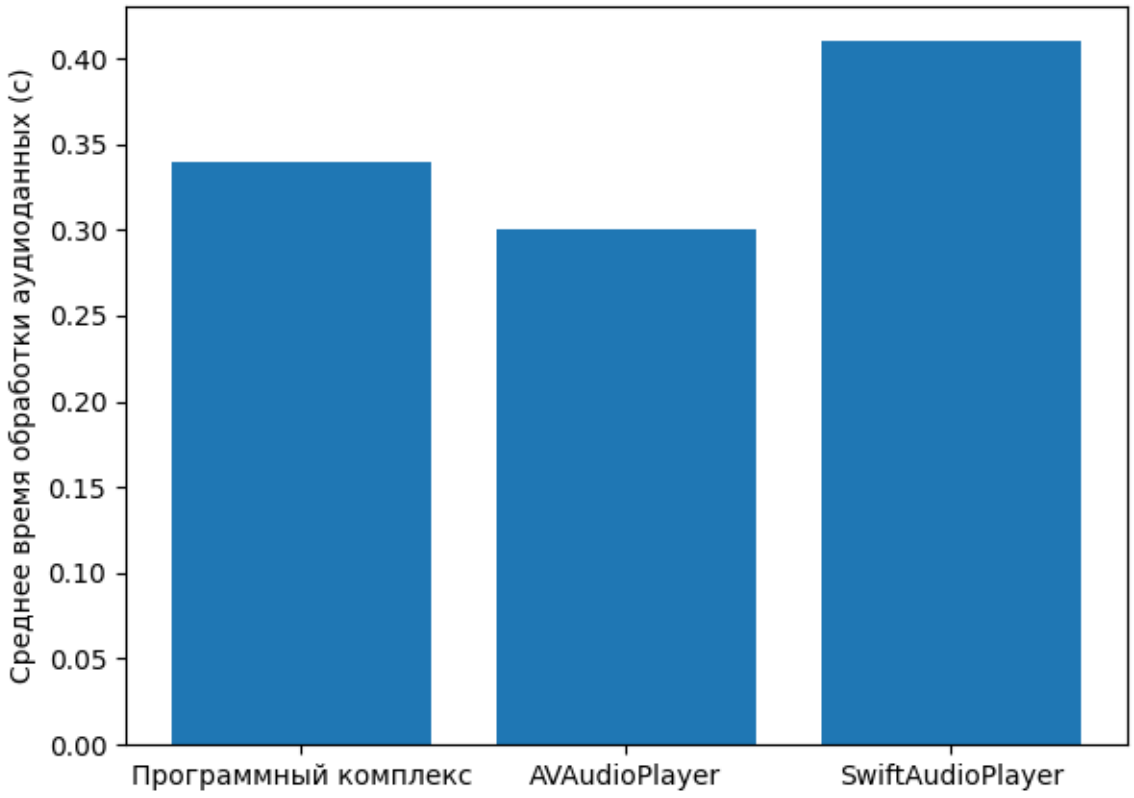

Рис. 25 – График сравнения среднего времени обработки аудиоданных разработанного программного комплекса, AVAudioPlayer и SwiftAudioPlayer.

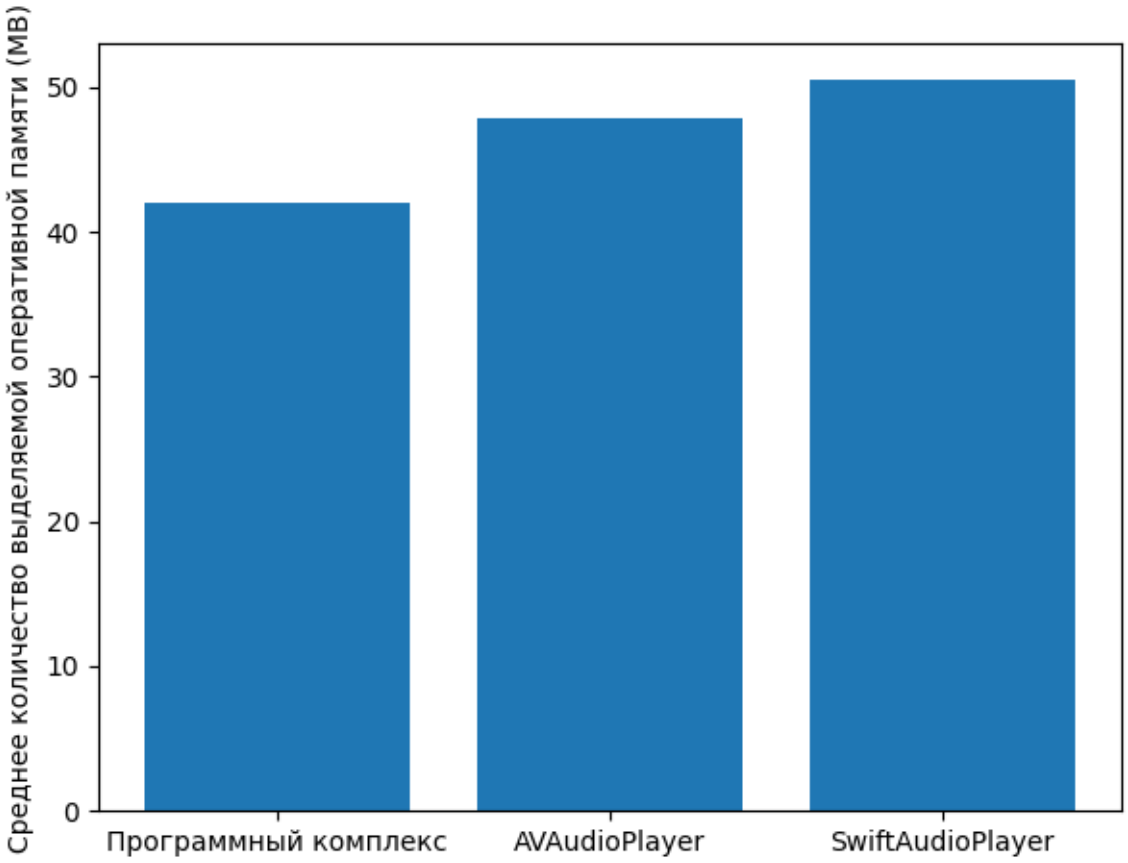

Рис. 26 – График сравнения среднего количества выделяемой оперативной памяти разработанного программного комплекса, AVAudioPlayer и SwiftAudioPlayer.

Вывод

В данном разделе была произведена оценка результатов обработки потоковых аудиоданных с помощью разработанного программного комплекса. Исследовано влияние буферизации и фильтрации компонентом управления потоком данных на время обработки данных, количество выделяемой памяти, а также нагрузку ЦПУ. Был произведён сравнительный анализ разработанного программного комплекса с существующими аналогами. В результате исследований было установлено:

— использование компонента управления потоком данных значительно снижает нагрузку ЦПУ, время и количество выделенной оперативной памяти

для обработки аудиоданных;

— разработанный программный комплекс на 20% быстрее обрабатывает аудиоданные, чем SwiftAudioPlayer и на 4% медленее, чем AVAudioPlayer;

— разработанный программный потребляет меньшее количество оперативной памяти, чем его аналоги;

ЗАКЛЮЧЕНИЕ

В рамках выполнения работы были решены следующие задачи:

— введены основные понятия предметной области;
— рассмотрены существующие средства воспроизведения аудиоданных в операционной системе iOS;
— проведён анализ протоколов потоковой передачи данных;
— рассмотрны форматы хранения аудиоданных;
— спроектирован и реализован программно-алгоритмический комплекс для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS;
— произведено сравнение времени и выделения оперативной памяти для обработки аудиоданных разработанным программным комплексом и существующими аналогами;

Таким образом цель работы —— разработать программно-алгоритмический комплекс для воспроизведения потокового аудио в мобильном приложении на операционной системе iOS была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Streaming is the future of media [Электронный ресурс]. Режим доступа URL: https://www.robeco.com/en-int/insights/2020/07/streaming-is-the-future-of-media (Дата обращения: 18.10.2022).

2. Mobile APPs and Global Markets [Электронный ресурс]. Режим доступа URL: https://www.scirp.org/journal/paper-information-aspx?paperid=85209 (Дата обращения: 13.10.2022).

3. Wav/wave audio format [Электронный ресурс]. Режим доступа URL: https://docs.fileformat.com/audio/wav/ (Дата обращения: 18.10.2022).

4. MP4 audio format [Электронный ресурс]. Режим доступа URL: https://docs.fileformat.com/video/mp4/ (Дата обращения: 18.10.2022).

5. MP3 audio format [Электронный ресурс]. Режим доступа URL: https://docs.fileformat.com/audio/mp3/ (Дата обращения: 18.10.2022).

6. ACC audio format [Электронный ресурс]. Режим доступа URL: https://docs.fileformat.com/audio/aac/ (Дата обращения: 18.10.2022).

7. MPEG Audio [Электронный ресурс]. Режим доступа URL: https://sound.media.mit.edu/resources/mpeg4/audio/mpeg1 (Дата обращения: 18.10.2022).

8. HTTP Live Streaming [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/streaming/ (Дата обращения: 18.10.2022).

9. HTTP Live Streaming Low Latency [Электронный ресурс]. Режим доступа URL: https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis (Дата обращения: 18.10.2022).

10. Real Time Streaming Protocol [Электронный ресурс]. Режим доступа URL: https://www.ietf.org/rfc/rfc2326.txt (Дата обращения: 18.10.2022).

11. TRANSMISSION CONTROL PROTOCOL [Электронный ресурс]. Режим доступа URL: https://www.ietf.org/rfc/rfc0793.txt (Дата обращения: 18.10.2022).

12. User Datagram Protocol [Электронный ресурс]. Режим доступа URL: https://www.ietf.org/rfc/rfc768.txt (Дата обращения: 18.10.2022).

13. iOS mobile operating system [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/ios/ (Дата обращения: 18.10.2022).

14. AVPlayer [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/documentation/avfoundation/avplayer/ (Дата обращения: 18.10.2022).

15. AVAudioPlayer [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/documentation/avfaudio-avaudioplayer (Дата обращения: 18.10.2022).

16. AVAudioEngine [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/documentation/avfaudio-avaudioengine (Дата обращения: 18.10.2022).

17. Core Audio [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/documentation/coreaudio (Дата обращения: 18.10.2022).

18. AVFoundation Documentation [Электронный ресурс]. Режим доступа URL: https://developer.apple.com/av-foundation/ (Дата обращения: 14.03.2023).

19. AudioToolbox Documentation [Электронный ресурс]. Режим доступа URL:

https://developer.apple.com/documentation/audiotoolbox/ (Дата обращения: 14.03.2023).

20. VLCKit Source [Электронный ресурс]. Режим доступа URL: https://github.com/videolan/vlckit (Дата обращения: 14.03.2023).

21. SwiftAudioPlayer Source [Электронный ресурс]. Режим доступа URL: https://github.com/tanhakabir/SwiftAudioPlayer (Дата обращения: 14.03.2023).

22. libVLC Documentation [Электронный ресурс]. Режим доступа URL: https://www.videolan.org/vlc/libvlc.html (Дата обращения: 14.03.2023).

23. Панфилов И.П., Дырда В.Е. «Теория электрической связи.» — М.: Радио и связь, 1991. — 344 с.

24. HTTP 1.1 Protocol Режим доступа URL: https://datatracker.ietf.org/doc/html/rfc2616 (Дата обращения: 14.03.2023).

25. Troubleshooting High CPU Utilization Режим доступа URL: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750/software/troubleshoot
util.html (Дата обращения: 14.03.2023).

26. The Swift Programming Language Режим доступа URL: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/ (Дата обращения: 14.03.2023).

27. Apple Xcode IDE Режим доступа URL: https://developer.apple.com/xcode/ (Дата обращения: 14.03.2023).

ПРИЛОЖЕНИЕ А