Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Макеев Владислав Дмитриевич

Разработка веб-приложения для обмена данными с беспилотным летательным аппаратом

Курсовая работа

Научный руководитель:
ст. преп. Смирнов М. Н.

Консультант:
Пименов А. А.

Санкт-Петербург
2020

Оглавление

Введение

С каждым днём беспилотные аппараты всё больше входят в жизнь человека, проявляя отменные качества в самых разных сферах деятельности – от обнаружения очагов пожаров до исследования труднодоступных элементов строительных конструкций. Уже сейчас с помощью дронов многие работы выполняются дешевле и проще обычного – так, например, фотографам не приходится арендовать вертолёт, чтобы снять красивую панораму. В некоторых делах – таких, как сбор данных в эпицентре шторма для получения информации о тенденциях его движения – коптеры являются единственным возможным способом решения проблемы, что также говорит об огромной актуальности любых исследований, направленных на совершенствование беспилотных летательных аппаратов.

Возможности, которые дроны открывают, кажутся почти безграничными. На пути прогресса, однако же, встаёт множество экономических и технологических проблем. Кроме больших задач – необходимости снижения себестоимости производства, улучшения аэродинамических свойств и тому подобного – существует также ряд более мелких, гораздо менее предсказуемых на этапе проектирования трудностей.

Одним из лимитирующих факторов, влияющих на процесс взаимодействия с дронами, является необходимость использования предустановленного ПО для получения данных с уже запущенного аппарата. Так, например, у среднестатистического пользователя, запустившего клиент на ПК для отслеживания получаемой камерами дрона картинки, отсутствует возможность быстрой смены используемого устройства на планшет в случае, когда ему необходимо отойти от своего рабочего места вслед за беспилотным летательным аппаратом. Особо проблематично отслеживать параметры, заниматься отладкой и настройкой дрона с мобильных телефонов, а затея воспользоваться устройством коллеги в случае, когда телефон внешнего пилота сел, наверняка не увенчается успехом – опять же, ввиду отсутствия нужного ПО и возможности его оперативной установки.

Такая проблема встречается повсеместно – пусть протоколы передачи данных дронам и поддержаны многими программными продуктами, они используют собственный клиент, а кроме того зачастую разработаны для конкретного класса беспилотных летательных аппаратов.

Чтобы исключить подобные элементы в процессе использования дронов, можно использовать тонкий клиент, уже установленный почти в каждом устройстве – web-browser. Платформой для развертывания сервера в данном случае выступает сам дрон, к которому и будут подключатся пользователи. Подобное решение в виде веб-приложения внутри другого программного продукта широко распространено и применяется в таких известных программах, как µTorrent или pgAdmin.

Целью данной курсовой работы является разработка веб-приложения для Embedded Linux, интегрированного с библиотекой CoreCVS (Computer Vision Primitives Library) и способного обеспечить одновременное общение нескольких клиентов с дроном.

1. Постановка задачи

Целью данной работы является разработка интегрированного с CoreCVS веб-приложения для Embedded Linux (для таких платформ, как, например, NVidia Jetson Nano, Raspberry Pi, Orange Pi и прочие), способного обеспечить обмен данными (трансляция изображения с камеры, манипуляция параметрами и прочее) с дроном.

Для достижения цели были поставлены следующие задачи:

1. Изучение существующих веб-серверов и выбор наиболее подходящего варианта.
2. Интеграция выбранного сервера с CoreCVS.
3. Добавление необходимой функциональности (передача потока изображений, редактирование списка параметров и передача сообщений с сервера).

2. Обзор

2.1. Языки программирования

В качестве основного инструмента для решения задачи был выбран язык программирования C++. Это решение обусловлено следующими факторами (по степени значимости):

1. Является языком, на котором написан CoreCVS[1], что обеспечивает возможность тривиальной интеграции сервера с отлаженным механизмом взаимодействия с дроном.

2. Позволяет добиться достаточной производительности.

3. Имеет в своём распоряжение широкий выбор библиотек.

Дополнительным языком разработки стал JavaScript, используемый для обеспечения интерактивности клиентской части программного продукта и отдаваемый пользователям вместе с html страницами.

2.2. Сервер

2.2.1. Требования

Перед непосредственным перечислением кандидатов на роль обработчика запросов следует упомянуть критерии, применявшиеся к потенциальным серверам:

1. Поддержка websocket или longpoll для синхронной отправки сообщений.

2. Поддержка достаточного количества клиентов (около десятка).

3. Наличие лицензии, совместимой с MIT[2] .

[1]CoreCVS Wiki, URL: https://github.com/PimenovAlexander/corecvs/wiki (дата обращения: 12/4/2020).

[2]Сайт Open Source Initiative, URL: https://opensource.org/licenses/MIT (дата обращения: 12/4/2020).

Требование синхронной отправки сообщений обусловлено необходимостью получения постоянного потока телеметрии, будь то данные от акселерометра, гироскопа, показателя тяги, загруженности процессора и прочего.

Лицензии типа MIT позволяют свободно использовать и модифицировать код программных продуктов.

2.2.2. Выбор протокола передачи данных

Анализ технологий websocket и longpoll в контексте рассматриваемой задачи привёл к выводу о большей рациональности поиска решения с websockets в силу того, что longpoll (рис. 1) в некоторых ситуациях вызывает большую нагрузку на сервер.

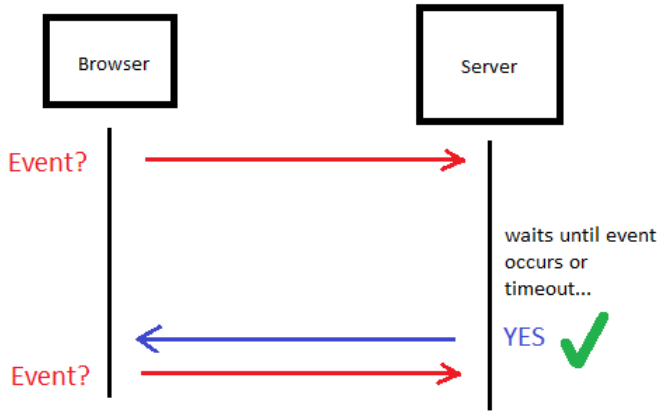

Рис. 1: Упрощённая схема работы longpoll

В свою очередь работа протокола websocket (рис. 2) имеет и свои минусы. Так, при разрыве соединения в случае нештатных ситуаций она не только не гарантирует, но даже и не пытается его восстановить, перекладывая реализацию алгоритма повторного соединения на плечи вышележащих абстракций.

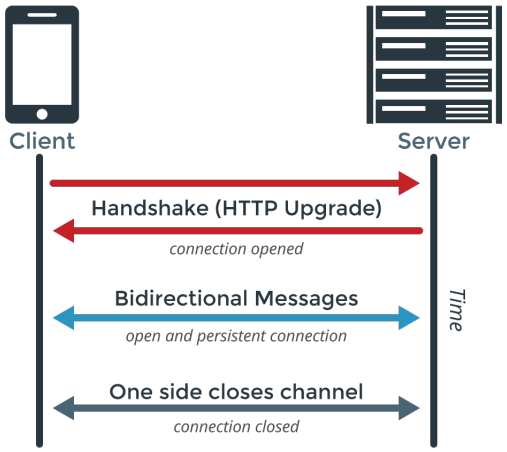

Рис. 2: Упрощённая схема работы websockets

Несмотря на это, при прочих равных websockets демонстрирует лучшее быстродействие [3], что позволяет сузить круг поиска подходящего сервера до тех вариантов, которые используют именно эту технологию.

2.2.3. Готовые решения

Итоговыми кандидатами на этом этапе стали библиотеки:

- uWebSockets[3]
- IXWebSocket[4]
- Simple-WebSocket-Server[5]

Готовые сервера			
Требования	uWebSockets	IXWebSocket	Simple-WebSocket-Server
WebSockets	✓	✓	✓
Поддержка 10 клиентов	✓	✓	✓
Лицензия	MIT	Apache-2	BSD-3

Таб. 1: Сравнение готовых решений

[3]Репозиторий uWebSockets, URL: github.com/uNetworking/uWebSockets (дата обращения: 12/4/2020).

[4]Репозиторий IXWebSockets, URL: github.com/machinezone/IXWebSocket (дата обращения: 12/4/2020).

[5]Репозиторий Simple-WebSocket-Server, URL: github.com/eidheim/Simple-WebSocket-Server (дата обращения: 12/4/2020).

Удовлетворяя всем вышеописанным критериям, они все одинаково хорошо подходят для решения поставленной задачи. Вместе с тем, все вышеперечисленные сервера предоставляют чрезмерную функциональность, не требующуюся для успешного решения поставленных задач.

2.2.4. Альтернатива готовым решениям

В процессе изучения существующих библиотек с готовыми Websocket серверами гипотеза о необходимости использования таковых подверглась тщательному пересмотру – как оказалось, простые библиотеки вроде libevent[6] позволяют без особых проблем реализовать всю требуемой функциональность, ставя под сомнение необходимость использования более тяжеловесных решений. Результатом анализа этой возможности стало решение об использовании библиотеки ассинхронного уведомления о событиях libevent и встроенного в неё http-server'а для реализации требуемого функционала.

2.3. Существующее решение

На момент написания работы уже имеется готовое решение, предоставляющее значительную часть перечисленной функциональности (получение картинки, контроль параметров). Причинами, побудившими написать новый проект вместо доработки существующего, являются использование в готовом решении собственного клиента вместо браузера, а также сильная зависимость от QT framework. Существующее решение также использует QHttpServer для обмена данными, что в данном случае почти полностью исключает возможность переиспользования существующего кода.

Несмотря на перечисленные препятствия, некоторые модули всё же удалось позаимствовать. К ним относятся классы для модульного сервера и принадлежащие им объекты данных. В процессе интеграции сервера сами модули подверглись частичному переписыванию, а объекты

[6]Сайт библиотеки LibEvent, URL: https://libevent.org/ (дата обращения: 12/4/2020).

данных использовались для предоставления интерфейса при написании примеров модульного сервера.

3. Серверная часть

3.1. Анализ требований

Приложение подразумевает наличие следующего функционала:

1. Отображение картинок:
    - статических (иконки);
    - динамических (изображение с камеры).
2. Отображение данных – крен, тангаж, рыскание и т.д.
3. Редактирование данных.

Для реализации каждого пункта были выбраны соответствующие нуждам проекта технологии:

3.1.1. Отображение статических изображений

Для подгрузки статических ресурсов было решено добавлять их к бинарному файлу приложения на стадии линковки. Так как CoreCVS собирается с помощью CMake, было решено написать соответствующий скрипт, который бы обеспечивал доступ к статическим ресурсам как массивам данных внутри C++ кода.

Алгоритм упаковки ресурсов:

1. Создаётся пустой C файл для хранения ресурсов.
2. Если в выбранной директории не осталось ничего непроверенного, то алгоритм завершает работу.
3. Выбирается следующий файл/папка в текущей директории.
4. Если найдена папка, то алгоритм рекурсивно запускается для соответствующего каталога со второго шага.
5. Если файл не найден, то алгоритм переходит ко второму шагу.

6. В файле для хранения ресурсов добавляется массив с наименованием этого файла.

7. Файл ресурса считывается в виде массива символов в файл для хранения ресурсов.

8. В файл для хранения ресурсов добавляется переменная с размером считанного файла.

9. Алгоритм переходит ко второму шагу.

Таким образом любой статический контент, лежащий в указанных для скрипта директориях, оказывается доступен в коде при добавлении директивы "extern" через соответствующие символы.

3.1.2. Отображение динамических изображений

Данная задача подразумевает постоянное обновление изменяющегося на сервере изображения, представленного в CoreCVS в виде rgb24buffer. Для этого сервер должен запаковывать данные и отдавать их по соответствующему запросу. Кроме того, необходимо иметь систему оповещения клиента о появлении нового изображения. То же самое касается и изменения различных данных, хранящихся на сервере.

Из-за отказа от полноценных websocket серверов требовалось написать инструмент оповещения клиентов с помощью средств libevent. Тут вновь встал вопрос между довольно традиционными websocket и longpoll. Как уже было выяснено на стадии выбора сервера, разница между ними не слишком большая, а реализация longpoll несколько проще. Так как наибольшие различия технологий проявляются в сценариях, не относящихся к типовым сценариям для рассматриваемой задачи, выбор пал на longpoll.

3.1.3. Отображение данных

Для отображения данных, а конкретнее – для сборки страниц с отображаемыми данными на стороне сервера – было решено воспользоваться средствами рефлексии, уже имеющимися в CoreCVS. Такой подход,

к примеру, позволяет не изменять связанный с веб-приложением код при изменении типов передаваемых данных.

3.1.4. Редактирование данных

Как и в случае с предыдущими пунктами, редактирование данных использует систему оповещений клиентов на основе longpoll для извещения о внесенных другими пользователями изменений.

3.2. Архитектура

На рис. 3 изображена архитектура серверной части приложения. Функция processRequests отвечает за обработку всех пришедших на сервер запросов, что позволяет при необходимости запускать приложение в одном потоке с другими утилитами. На рисунке обозначены лишь основные методы и поля соответствующих классов.

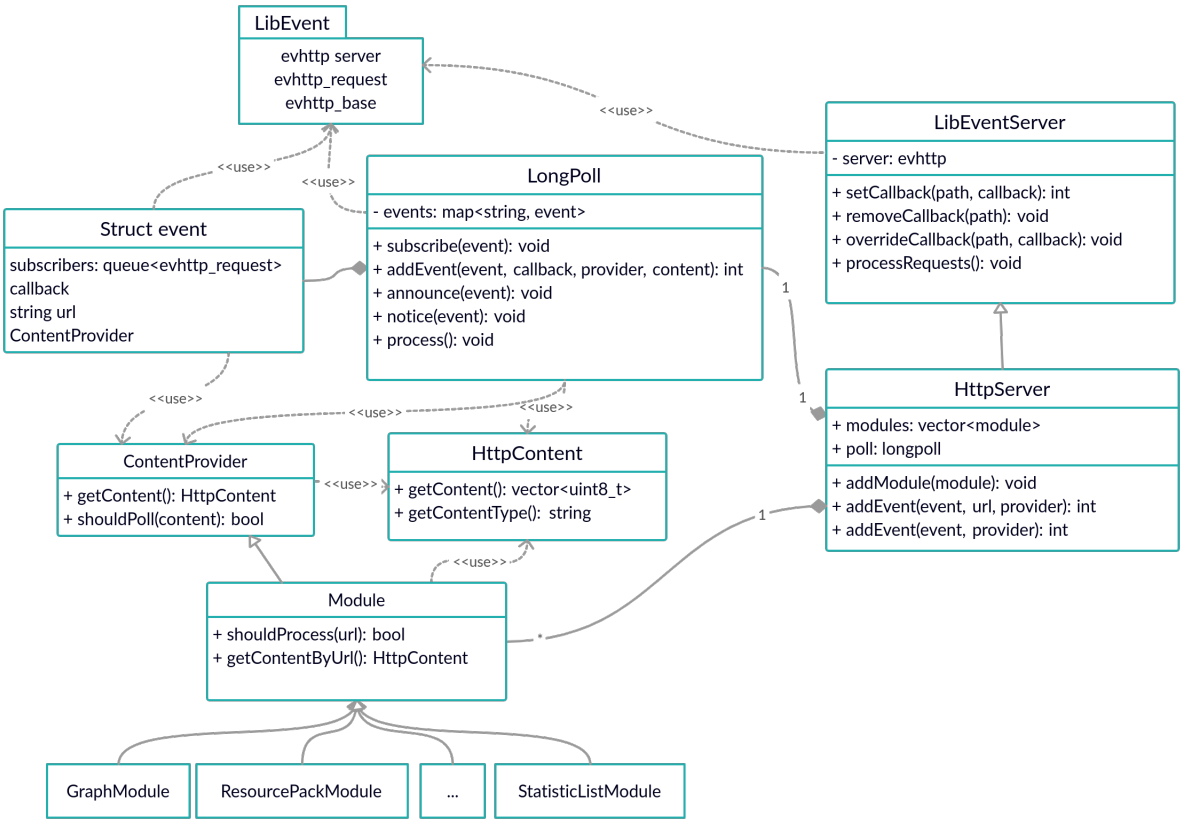

Рис. 3: Архитектура сервера

3.3. Longpoll

Разнообразие возможных запросов от клиента, требующих получения информации при её появлении, ограничивается тремя пунктами:

1. Получение обновленного изображения.
2. Получение обновленных данных.
3. Запрос на изменение данных (получение подтверждения или кода ошибки).

Для удобства работы было решено ввести систему событий, на которые клиенты могут подписываться. С учетом специфики приложения, написанная реализация longpoll (в зависимости от конфигурации сервера) поддерживает две стратегии поведения:

1. Рассылка изображения/данных при их обновлении (активации соответствующего события).
2. Отметка события (обновление изображения/данных) и рассылка всех обновившихся изображений/данных при следующем вызове функции process, отвечающей за обработку всех активных событий.

Само событие представляет из себя структуру с полями следующих типов:

- Queue<Subscriber> – Очередь подписчиков, где тип Subscriber определяется как evhttp-request – приходящий от пользователя запрос;
- string (URL) – Адрес страницы для обработки;
- callback-fn – Функция, которая должна вызываться при активации события, и принимающая в качестве параметров подписчика и контент в виде shared-ptr<HttpContent>;

- ContentProvider – Объект, предоставляющий метод для получения указателя на HttpContent;
- bool – Флаг, отображающий состояние события.

Данная реализация longpoll опирается на объекты типа ContentProvider, а те, в свою очередь, обладают методом getContent(url), что и диктует необходимость хранения URL в структуре события.

Полученные указатели на HttpContent требуется отослать подписчикам, чем и определяется наличие в структуре поля типа callback-fn, предназначенного для предоставления более гибкого метода отправки в сравнении с использованием одной функции для всех типов событий.

3.4. Обработка запросов

Алгоритм обработки запросов зависит от используемого подкласса libEventServer, однако в целом представляет собой последовательность следующих действий:

1. Сервер получает запрос.
2. Осуществляется поиск обработчика для соответсвующей URL.
3. Если обработчик найден, то он запускается, иначе запускается общий обработчик.
4. В обработчике за счет конкатенации нужных строк/файлов собирается html+css+js контент или же добавляется изображение.
5. Полученный контент добавляется в evbuffer.
6. Указываются соответствующие http заголовки.
7. Контент отправляется клиенту.

Модульный сервер (HttpServer на рис. 3) в общем обработике опрашивает все callback-модули, проверяя запрос на принадлежность к их

зоне компетенции, а в случае положительного результата проверки обрабатывает его.

Подобное поведение позволяет разделить модули на два класса: отвечающие за непосредственную обработку данных и функциональность [1].

Кроме функции shouldProcessURL модули также имеют функцию shouldPollURL, позволяющую определить тип запроса. В случае положительного результата проверки клиент добавляется в подписчики соответствующего запросу события, а в случае отрицательного результата проверки вызывается функция getContent, возвращающая указатель на передаваемый клиенту контент.

4. Клиентская часть

Здесь и далее под клиентской частью понимается JS код, отдаваемый сервером клиенту.

4.1. AJAX

Многие элементы страницы (данные и изображения) постоянно обновляются, из-за чего их получение следует осуществлять AJAX-запросами (Asynchronous JavaScript and XML – Ассинхронный JavaScript и XML) для предотвращения постоянного обновления страницы. Зачастую для посылки таких запросов используется wrapper из JQuery, однако так как анализ требований не выявил других областей применения данной библиотеки в данном проекте, было решено написать свою обёртку. Выглядит она весьма просто:

sendAjax(url, func, options), где:

- func – обработчик успешного обращения, принимающий в качестве единственного аргумента полученные от сервера данные;
- options – объект с дополнительными параметрами, такими как:
  - error – обработчик запроса, вернувшего код ошибки;
  - failureDelay – указание промежутка, за который ожидается получение данных;
  - recursive – флаг, указывающий на необходимость повторной отправки запроса после получения данных или же кода ошибки.

В случае превышения указаного интервала ожидания запрос отправляется повторно, что позволяет клиенту автоматически восстанавливать потерянное соединение.

4.2. Графики

Часть данных удобно отображать в виде графиков, а значит требуется каким-то образом выводить их на экран. Для этого существуют готовые библиотеки, из которых были выбраны следующие кандидаты:

Библиотеки для отображения графиков		
Библиотеки	Локальный запуск	Бесплатная лицензия
D3.js	✓	✓
JSCharting	✓	✗
Highcharts	✓	✗
amCharts	✓	✗
Google Charts	✗	✓
KoolChart	✓	✗
ChartJS	✓	✓
FusionCharts	✓	✗
Canvas	✓	✗

Таб. 2: Сравнение библиотек для отображения графиков

Выдвинутые критерии оставляют двух кандидатов: D3.js и Chart.js. Библиотеки показывают схожую производительность [2], однако D3.js не является библиотекой исключительно для построения графиков – это обширная графическая библиотека для визуализации данных и манипулирования SVG. Из-за этого для отрисовки графиков было решено использовать специализированный инструмент в виде Chart.js.

Для взаимодействия с Chart.js написан модуль graph.js, умеющий запрашивать и отображать полученные от сервера JSON файлы в виде графиков.

5. Апробация

Для проверки соответствия проекта поставленным ранее требованиям были проведены замеры производительности. В отсутствии доступа к целевым платформам (Raspberry Pi, Jetson Nano) замеры производились на персональном компьютере с операционной системой Ubuntu 18.04, процессором AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx 2.30 GHz и 16 GB ОЗУ.

Тестировался одновременный запуск двух экземпляров HttpServer в одном потоке с последующим подключением к серверам до десяти устройств, находящихся в одной WiFi сети.

Замеры памяти производились с помощью команды "pmap PID | tail -n 1", замеры CPU – с помощью монитора процессов "htop". Указанные в таблице проценты ЦПУ считаются от одного ядра. FPS (количество кадров в секунду) подсчитывался JS-кодом страницы.

Сервер передавал сгенерированные изображения со средним весом в 800 Кб и 17.4 Кб, а также тестировался без нагрузки, передавая только числовые данные с "Датчиков".

Замеры производительности			
Тест	pmap PID | tail -n 1	htop CPU%	FPS
Нет клиентов, изображение не передаётся	501252K	5.3%	-
1 клиент, изображение 800 Кб	370180K	30.4%	7
1 клиент, изображение 17.4 Кб	435716K	5.7%	48
10 клиентов, изображение не передаётся	501252K	10.5%	-
10 клиентов, изображение 800 Кб	370180K	37.1%	1
10 клиентов, изображение 17.4 Кб	435716K	12%	45

Таб. 3: Замеры производительности

Результаты тестирования показали возможность применения серве-

ра с 10 одновременно подключенными клиентами. Ручное тестирование заявленного функционала не выявило никаких багов.

Заключение

В ходе работы были выполнены следующие задачи:

1. Выполнен анализ существующих веб-серверов и подобран наиболее подходящий вариант.
2. Выполнена интеграция libevent http-server с CoreCVS.
3. Добавлен необходимый функционал – передача потока изображений, редактирование списка параметров и передача сообщений с сервера.

Список литературы

[1] Best practices for creating modular Web applications — Дата обращения : 28.11.20. — Режим доступа: https://www.slideshare.net/peychevi/best-practices-for-creating-modular-web-applications

[2] D3 or Chart.js for Data Visualisation — Дата обращения : 28.11.20. — Режим доступа: https://www.createwithdata.com/d3js-or-chartjs/

[3] Rasmus Appelqvist, Oliver Örnmyr, "Performance comparison of XHR polling, Long polling, Server sent events and Websockets", 2017 — Дата обращения : 28.11.20. — Режим доступа: http://www.diva-portal.se/smash/get/diva2:1133465/FULLTEXT01.pdf