Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2024 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Разработка подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных

090401 Информатика и вычислительная техника

09.04.01.11 «Вычислительные системы и сети»

Руководитель	_____ *подпись*	_____ *дата*	профессор, канд. техн. наук *должность, ученая степень*	О.В. Непомнящий
Выпускник	_____ *подпись*	_____ *дата*		П.Д. Неустроев
Рецензент	_____ *подпись*	_____ *дата*	доцент, канд. техн. наук *должность, ученая степень*	К.В. Раевич
Консультант	_____ *подпись*	_____ *дата*	генеральный директор ООО «ПК «Дельта»» *должность, ученая степень*	О.Г. Варыгин
Нормоконтролёр	_____ *подпись*	_____ *дата*	профессор, канд. техн. наук *должность, ученая степень*	О.В. Непомнящий

Красноярск 2024

РЕФЕРАТ

Выпускная квалификационная работа на тему «Разработка подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных» содержит 72 страницы текстового документа, 21 рисунок, 2 таблицы, 1 приложение, и 32 использованных источника.

ИКАПСУЛЯЦИЯ, ДЕКАПСУЛЯЦИЯ, МАРШРУТИЗАЦИЯ, ТЕЛЕМЕТРИЯ, ПЕРЕДАЧА ДАННЫХ, СЕТЬ, БАЗЫ ДАННЫХ, ПРОТОКОЛЫ ПЕРЕДАЧИ ДАННЫХ, WI-FI, ZIGBEE, C++, LINUX, MYSQL.

Цель работы: исследование, моделирование и разработка технических решений для создания подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных.

Задачи, решённые в процессе разработки:

- проведён аналитический обзор предметной области по теме исследования;
- предложен метод декапсуляции и инкапсуляции данных, основанный на разработанном алгоритме модуля преобразования данных;
- предложен метод маршрутизации данных, основанный на статической маршрутизации и оригинальных алгоритмах направления данных к нужному узлу сети;
- разработана программная модель, включающая в себя модули декапсуляции и инкапсуляции данных, классы Sensor и Terminal для представления в программе датчиков и исполнительных устройств, и библиотеку функций для маршрутизации данных;
- проведено тестирование разработанной системы.

В ходе выполнения ВКР предложен метод маршрутизации данных в сети программно-аппаратного комплекса управления сетью передачи телеметрических данных, а также алгоритм декапсуляции и инкапсуляции данных. Разработан программный модуль преобразования данных из состава программного обеспечения контроллера сети передачи телеметрических данных.

АННОТАЦИЯ

Магистерская диссертация на тему «Разработка подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных» выполнена в Институте космических и информационных технологий Сибирского федерального университета..

Проводятся исследования протоколов маршрутизации и протоколов, применяемых для инкапсуляции данных в системах сбора и анализа телеметрической информации. Разрабатываются модели, необходимые для организации подсистем инкапсуляции и маршрутизации данных в сети передачи телеметрических данных. Разрабатываются технические решения и программные средства из состава подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных. Проводится тестирование разработанных моделей и программных средств.

Отличительной особенностью разрабатываемой подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных является предложенный метод маршрутизации данных, основанный на статической маршрутизации и оригинальных алгоритмах направления данных к нужному узлу сети, позволяющий унифицировать данные, передаваемые при помощи различных протоколов передачи, а также предложенный алгоритм декапсуляции и инкапсуляции данных, позволяющий реализовывать инструментальные программные средства преобразования данных в сети передачи телеметрической информации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность проблемы:

В системах сбора и хранения информации для передачи данных используются различные протоколы, каждый из которых имеет свой формат, в который преобразуются данные перед отправкой. Процесс подготовки данных к передаче называется инкапсуляцией. С точки зрения семиуровневой модели OSI это процесс добавления к данным заголовков перед передачей с более высокого уровня вниз. Декапсуляцией данных называется обратный процесс – подготовка данных при движении снизу вверх [1].

При передаче данных важен не только их формат, но и верное определение маршрута в сети. Процесс выбора наилучшего маршрута с использованием некоторых заранее установленных правил [2] называется маршрутизацией.

Системы передачи телеметрических данных предназначены для сбора информации об изменениях и колебаниях значений заданных параметров на подконтрольных объектах. Передача данных при этом осуществляется посредством беспроводных или проводных сетей (оптических, электрических, компьютерных, а также радиоканалов) [3].

Примером подобной системы может служить проект системы передачи телеметрической информации посредством спутникового канала. Он представляет собой распределённый комплекс спутниковой связи, состоящий из двух больших подсистем: непосредственно системы спутниковой связи, и наземного программно-аппаратного комплекса передачи телеметрической информации, который, в свою очередь, представляет из себя совокупность серверов, контроллеров, устройств приёма и передачи телеметрической информации, и клиентских устройств. Участники системы обмениваются данными посредством беспроводных сетей. Особенностью данной системы является использование устройств с различными способами передачи данных. Для обеспечения корректного обмена данными между устройствами, составляющими программно-аппаратный комплекс передачи телеметрической

информации необходимо разработать систему инкапсуляции и маршрутизации данных. Таким образом, тема данной работы является актуальной.

Цель диссертационной работы: исследование, моделирование и разработка технических решений для создания подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных.

Для достижения поставленной цели сформулированы следующие задачи исследования:

- исследовать протоколы маршрутизации и инкапсуляции данных в системах сбора и анализа телеметрической информации;
- разработать программные модели, необходимые для организации подсистем инкапсуляции и маршрутизации в сети передачи телеметрических данных;
- разработать технические решения и программные средства;
- провести тестирование разработанных моделей и программных средств.

Предполагаемая научная новизна исследования заключается в

1. предложенном методе маршрутизации данных, основанном на статической маршрутизации и оригинальных алгоритмах направления данных к нужному узлу сети, позволяющем унифицировать данные, передаваемые при помощи различных протоколов передачи;
2. предложенном алгоритме декапсуляции и инкапсуляции данных, позволяющем реализовывать инструментальные программные средства преобразрования данных в сети передачи телеметрической информации.

1 Исследования протоколов маршрутизации и протоколов, применяемых для инкапсуляции данных в системах сбора и анализа телеметрической информации

1.1 Постановка задачи

В современном мире активно развивается цифровизация и автоматизация. Системы передачи телеметрической информации и сбора данных присутствуют во многих областях – от «интернета вещей» и «умного дома» до медицины, биологии, и автоматизации производства. Системы передачи телеметрических данных предназначены для сбора информации об изменениях и колебаниях значений заданных параметров на подконтрольных объектах. Передача данных при этом осуществляется посредством беспроводных или проводных сетей (оптических, электрических, компьютерных, а также радиоканалов).

Для развития этой отрасли в Российской федерации создаются различные проекты. К ним относится и проект системы передачи телеметрической информации посредством спутникового канала. Уникальность данного проекта состоит в том, что в качестве канала связи для передачи данных используется спутниковый канал. Такое решение позволит разворачивать систему сбора телеметрической информации в любой точке Земли, где нет интернета и любой другой связи, кроме спутниковой.

Этот проект можно разделить на две большие подсистемы: непосредственно систему спутниковой связи, и наземный программно-аппаратный комплекс. В свою очередь, наземный комплекс представляет из себя совокупность сервера, контроллера, датчиков, исполнительных устройств, а также пользовательских устройств.

На рисунке 1 представлена архитектура разрабатываемой системы.

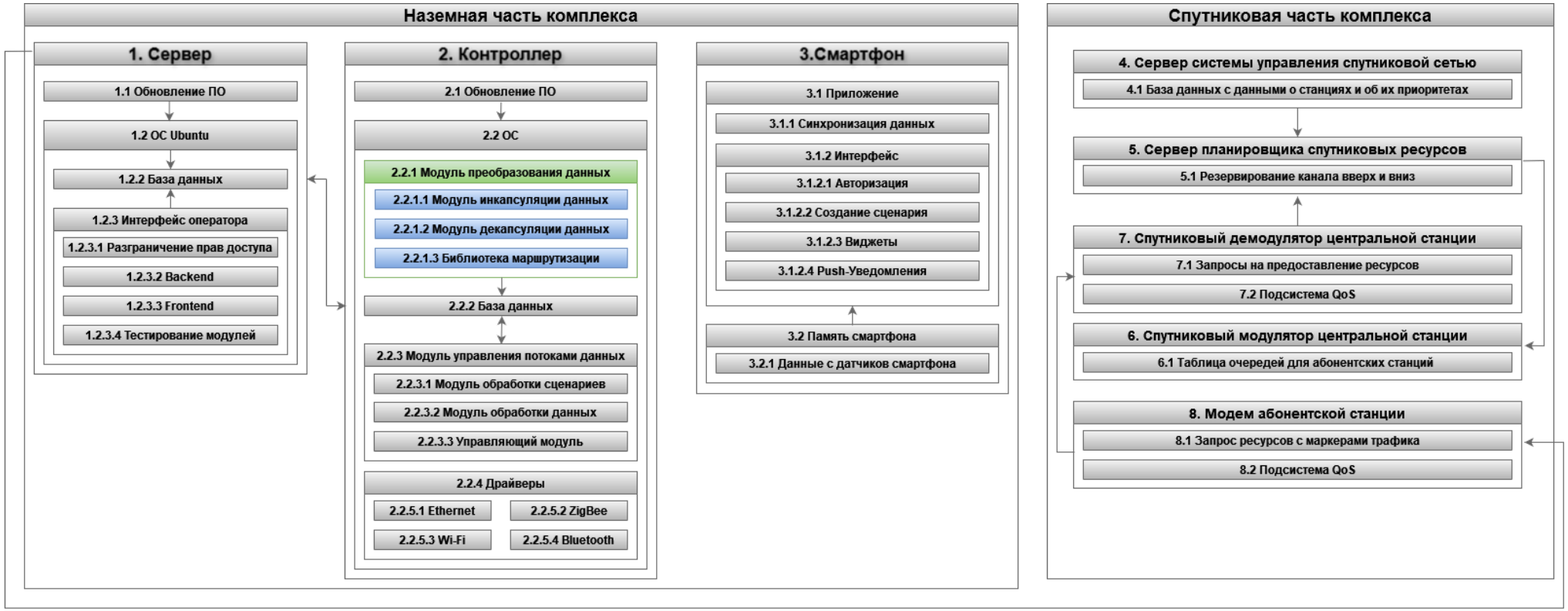


Рисунок 1 – Программная архитектура системы передачи телеметрической информации

На рисунке 1 можно увидеть, что основными компонентами программно-аппаратного комплекса передачи телеметрической информации являются контроллер, сервер и смартфон пользователя, а также спутниковая сеть передачи данных, служащая каналом связи между контроллером и сервером. Тем не менее, при подключении к сети Интернет через Wi-Fi либо по LAN-соединению, контроллер может обмениваться данными с сервером и без участия спутника. Также пользовательское приложение, установленное на смартфоне, может подключаться к контроллеру как через сервер, так и напрямую по Wi-Fi, при условии, что устройство пользователя и контроллер находятся в одной Wi-Fi сети.

Задача контроллера – собирать телеметрическую информацию с подключенных к нему датчиков, обрабатывать её и хранить, по запросу передавая пользователю через сервер, либо напрямую в приложение. Также, кроме датчиков, к контроллеру подключены и исполнительные устройства, которыми можно управлять при помощи сигналов, поступающих с контроллера.

В приложении для смартфона есть возможность просматривать поступающую на контроллер информацию с датчиков, а также создавать сценарии для контроллера по сбору данных и управлению исполнительными устройствами.

Сервер необходим для хранения системной информации, а также для обслуживания и администрирования системы.

Цель данной работы – разработка подсистем инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных. На схеме (Рисунок 1) эти подсистемы включены в состав модуля 2.2.1, входящего в состав программного обеспечения контроллера.

1.2 Сетевая маршрутизация

Маршрутизация позволяет проложить пути (маршруты) между узлами сети, не связанными напрямую, но имеющие узлы-посредники. Главная задача маршрутизации – определить наилучший путь пакетов от одного узла к другому [4].

На рисунке 2 изображена схема сети разрабатываемой системы. На ней, за исключением сервера, представлены устройства, доступные одному абоненту, а именно, два контроллера и три пользовательских устройства.

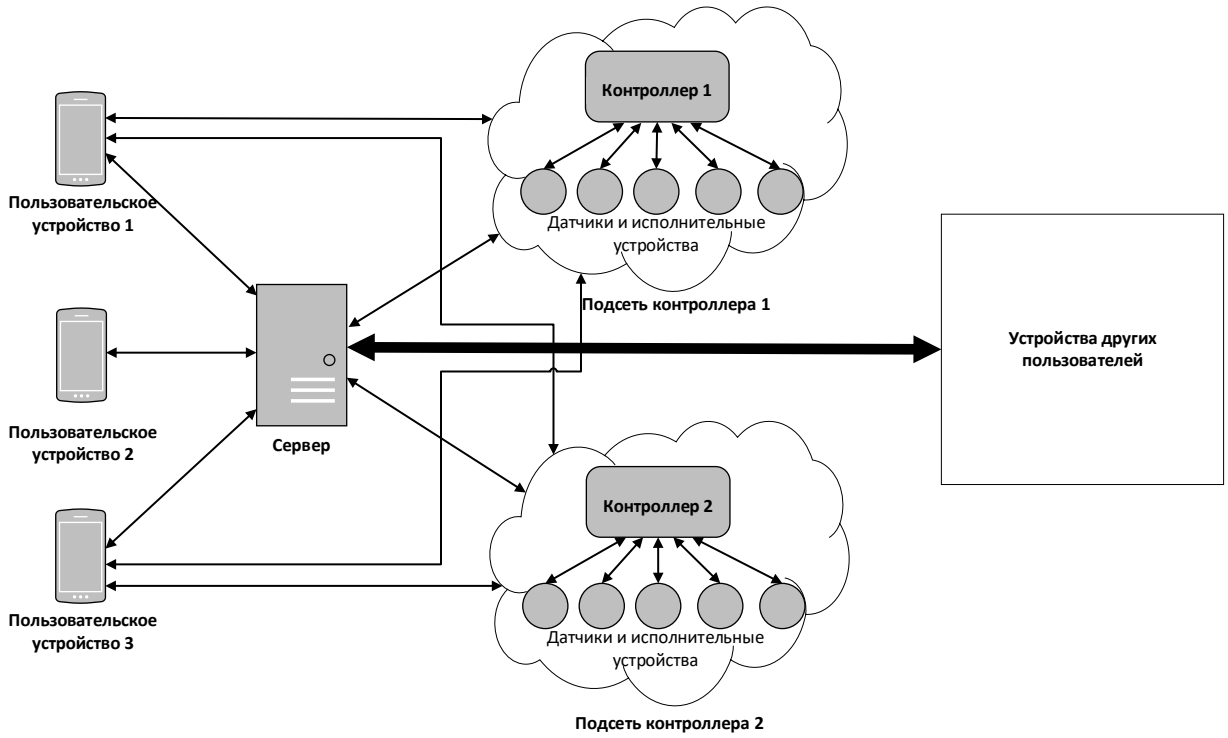


Рисунок 2 – Схема сети для одного абонента

К каждому контроллеру может быть подключено пять датчиков и исполнительных устройств. Такие требования были определены заданием на разработку системы. При этом соединение между пользовательскими устройствами и контроллером возможно как напрямую, при нахождении в одной Wi-Fi сети, так и через сервер. Также датчики и исполнительные устройства,

подключенные к контроллеру, образуют отдельную подсеть. Для взаимодействия между всеми элементами в системе необходима маршрутизация.

Существуют статические и динамические виды маршрутизации. При статической маршрутизации строится конечная таблица маршрутов при генерации сети. В динамической маршрутизации таблица маршрутов изменяется в зависимости от состояния и загрузки каналов передачи данных и узлов коммутации [5].

Существует множество протоколов динамической маршрутизации, реализующих разные алгоритмы маршрутизации. Далее рассматриваются некоторые из них. Отдельно рассматривается статическая маршрутизация.

1.2.1 Протокол RIP

Протокол RIP (Routing Information Protocol, Протокол маршрутной информации) – один из самых простых и старых протоколов динамической маршрутизации. Этот протокол использует два типа сообщений:

- Запрос (request) используется для запрашивания информации у соседних маршрутизаторов.

- Ответ (responce) содержит информацию о таблицах маршрутизации соседних маршрутизаторов.

Маршрутизатор, на котором настроен протокол RIP, с определённой частотой посылает широковещательные запросы со всех интерфейсов, которые поддерживают RIP. Если такой запрос поступил на маршрутизатор, он отправляет в ответ информацию о своей таблице маршрутизации. Получив эту информацию, маршрутизатор, отправивший запрос, вносит изменения в собственную таблицу маршрутизации. В качестве метрики для определения лучших маршрутов используется количество переходов. Протокол RIP имеет ограничение – максимально возможно 15 переходов. Это ограничивает размер сети [6, 7, 8].

У протокола RIP есть ещё один недостаток: он хранит только один, лучший, маршрут до каждого узла сети. Это значит, что, если один из узлов перестанет быть доступным, резервного пути не будет, и протокол начнёт работу сначала, заново выстраивая все таблицы маршрутизации, что может занять долгое время. Поэтому в настоящее время протокол RIP используют редко [6].

1.2.2 Протокол OSPF

Протокол OSPF (Open Shortest Path First, протокол маршрутизации по кратчайшему пути). Как следует из названия, этот протокол вычисляет лучшие пути между двумя нужными узлами сети. В отличие от протокола RIP, где каждый маршрутизатор запрашивает таблицу маршрутизации соседа, OSPF сначала составляет базу данных – карту сети, в которой затем находятся кратчайшие пути при помощи алгоритма Дейкстры.

База данных составляется при помощи сообщений, передаваемых между маршрутизаторами:

- LSA (link-state advertisements) – это объявления, которыми обмениваются маршрутизаторы. Есть разные типы сообщений LSA.
- LSDB (link-state database) – эти LSA формируют базу данных.

Чтобы избежать слишком большого трафика в сети, в которой все маршрутизаторы посылаю друг другу запросы, в сети выбирается основной маршрутизатор DR (Designated Router), и запасной маршрутизатор называется BDR (Backup Designated Router), через который происходит весь обмен уведомлениями.

При изменении топологии сети LSA-сообщения начинают рассылаться снова, формируя новую базу данных.

Протокол OSPF, по сравнению с RIP, требует больших ресурсов, и сложнее в настройке [6, 7].

1.2.3 Протокол EIGRP

Протокол EIGRP (Enhanced Interior Gateway Routing Protocol, усовершенствованный протокол маршрутизации внутренних шлюзов). Этот протокол долгое время был проприетарным протоколом компании Cisco.

Протокол EIGRP использует три таблицы:

- EIGRP Neighbor Table: в этой таблице представлены все напрямую соединенные маршрутизаторы.

- EIGRP Topology Table: в этой таблице представлены все изученные маршруты от соседей (с точкой назначения и метрикой).

- Global Routing Table: общая таблица, в которую записываются лучшие маршруты из предыдущей таблицы.

Как можно увидеть из описания таблиц, протокол EIGRP имеет сходства как с протоколом RIP, так и с OSPF. С RIP его объединяет изучение маршрутов соседей, а с OSPF – построение топологии сети. По этой топологии находятся кратчайшие маршруты при помощи алгоритма Беллмана-Форда.

Существенным отличием и преимуществом EIGRP от RIP является возможность хранения резервных маршрутов. Также протокол EIGRP прост в настройке, и поддерживает аутентификацию, что является важным моментом для обеспечения безопасности сети.

Из недостатков протокола EIGRP выделяют то, что он долго оставался внутренним протоколом CISCO, отчего устройства других производителей могут не поддерживать его. [6, 7, 9]

1.2.4 Статическая маршрутизация

Статическая маршрутизация – самый простой вид сетевой маршрутизации. При такой маршрутизации не используются никакие алгоритмы и протоколы, как при динамической. Чтобы настроить статическую маршрутизацию, достаточно вручную прописать все необходимые маршруты в сети.

Такой способ маршрутизации хорошо подходит для небольших сетей, где не будет происходить никаких изменений в течение долгого времени. Также он является достаточно безопасным ввиду отсутствия широковещательных запросов, которые могут рассылаться по сети при динамической маршрутизации.

Однако у статической маршрутизации есть и недостатки. Главный из них: в случае необходимости изменить маршруты в сети, или добавить новые, делать это придётся вручную, что не очень удобно. Таким образом, этот способ маршрутизации отличает плохая масштабируемость. [6, 10]

1.2.5 Анализ методов маршрутизации

Были рассмотрены три прокола динамической маршрутизации данных: RIP, OSPF и EIGRP. Все они имеют значительные отличия друг от друга, обладают своими преимуществами и недостатками. Так, протокол RIP является самым простым из рассмотренных, но у него есть существенные недостатки: долгое время построения таблиц маршрутизации, отсутствие резервных путей, ограничение размера сети.

Протокол EIGRP лишён многих недостатков протокола RIP. Имеется возможность хранить резервный путь, также кратчайшие пути определяются, исходя из топологии сети. Протокол прост в настройке, и показывает хорошие результаты работы. Однако, так как этот протокол долгое время являлся проприетарным протоколом компании Cisco, его может не поддерживать оборудование других производителей.

Протокол OSPF имеет сходства с протоколом EIGRP, но в то же время имеет ряд отличий. В частности, OSPF имеет меньшие временные задержки при работе. В таблицу маршрутизации этого протокола попадают только лучшие маршруты, то есть резервные пути не сохраняются, но благодаря скорости работы протокола новая топология сети строится достаточно быстро. Этот протокол более ресурсоёмкий, чем RIP, но его поддерживает большая часть сетевых устройств, что является несомненным преимуществом.

Статическая маршрутизация не использует никаких алгоритмов, так как подразумевает ручную конфигурацию всех доступных маршрутов в сети. Такая маршрутизация удобна для использования в небольших сетях, администраторам которых точно известны все маршруты, которые необходимы в данной сети. В случае, когда количество узлов и маршрутов между ними велико, или часто меняется, более подходящим будет использование динамической маршрутизации. Применительно к разрабатываемой системе можно сказать, что несмотря на то, что возможны разные варианты её конфигурации из-за разного количества периферийных устройств, подключенных к контроллеру, и возможных связей между ними, подходящим решением будет использование статической маршрутизации. Такой вывод можно сделать, исходя из того, что количество маршрутов в сети передачи телеметрической информации для одного абонента известно, и не будет изменяться с течением времени. Исключение составляет конфигурация подсети подключаемых к контроллеру периферийных устройств, но в этом случае могут создаваться самоорганизующиеся и самовосстанавливающиеся беспроводные сети, например, сети Zigbee [11].

Таким образом, каждый из рассмотренных протоколов имеет свои преимущества, но, исходя из предполагаемой архитектуры разрабатываемой системы, наиболее подходящим методом маршрутизации для неё является статическая маршрутизация.

1.3 Инкапсуляция данных

Для описания архитектуры и принципов работы сетей передачи данных применяется сетевая семиуровневая модель OSI (Basic Reference Model Open Systems Interconnection model). Эта модель состоит из семи уровней, где первый – физический уровень, где данный представляются в виде бит, а седьмой уровень – прикладной, тот, что видит пользователь [12, 13]. При передаче данных в сети данные переходят с одного уровня модели OSI на другой, проходя через их все

(Рисунок 3). Процесс подготовки данных к передаче называется инкапсуляцией данных. С точки зрения семиуровневой модели OSI это процесс добавления к данным заголовков перед передачей с более высокого уровня вниз. Декапсуляцией данных называется обратный процесс – подготовка данных для перехода снизу вверх.

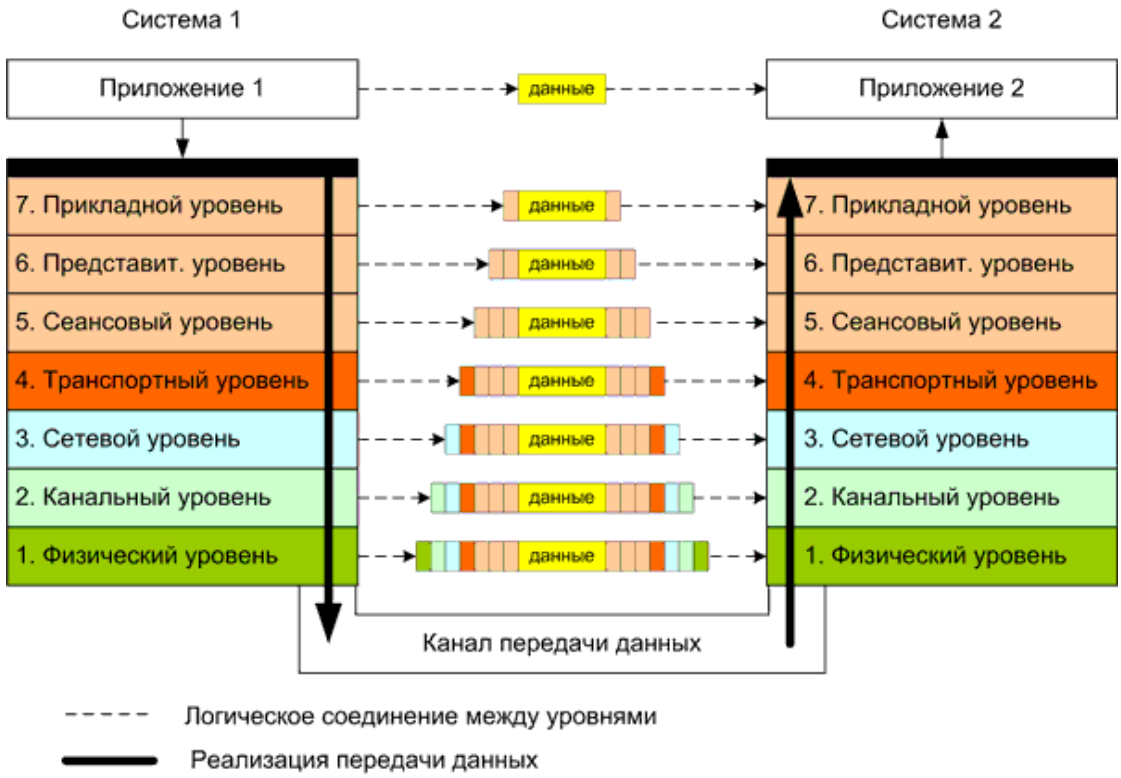


Рисунок 3 – Модель OSI

Инкапсуляция и декапсуляция данных – процессы, тесно связанные с конкретными протоколами передачи данных, так как при инкапсуляции и декапсуляции идёт работа непосредственно с пакетами данных, передаваемыми по сети. Поэтому для разработки системы инкапсуляции необходимо учитывать используемые в сети протоколы передачи данных.

В случае разрабатываемого программно-аппаратного комплекса передачи телеметрических данных, инкапсуляция и декапсуляция данных необходимы при работе с датчиками, которые могут передавать информацию контроллеру по протоколам Zigbee, Wi-Fi. Контроллеру же необходимо извлекать чистые данные для хранения их в базе данных, и добавлять полученные данные в ip-

пакеты для дальнейшей передачи. Эти задачи должен выполнять модуль инкапсуляции.

Далее будут рассмотрены протоколы, с которыми будет работать модуль инкапсуляции.

1.3.1 Zigbee

Zigbee является одним из самых популярных протоколов для подключения периферийных устройств к контроллеру различных систем автоматизации и передачи телеметрической информации. Этот протокол основан на стандарте IEEE 802.15.4. Применяется для коммерческих и жилых сетей IoT с ограничениями по стоимости, мощности и пространству. Он позволяет создавать ячеистые сети и управлять ими, обнаруживать новые устройства, обеспечивать безопасность и самовосстановление. Zigbee может работать в разных диапазонах: 868 МГц – в Европе, 915 МГц – в США, и 2,4 ГГц – в Российской Федерации. В диапазоне 2,4 ГГц предусмотрено 16 частотных каналов по 1МГц, максимальная скорость передачи в которых составляет 250 кбит/с. Максимальное расстояние связи 20-25 метров [14].

Устройства Zigbee имеют спящий режим, из которого могут переключаться в рабочее состояние за 15 мс или меньше. Наличие спящего режима обеспечивает низкое энергопотребление. Также Zigbee имеет три механизма безопасности: списки контроля доступа (ACL), 128-битное шифрование AES и таймеры обновления сообщений [15].

Протокол Zigbee использует несколько различных видов пакетов для передачи данных разного уровня [16]:

- Пакет данных: используется для передачи данных.
- Пакет подтверждения: используется для подтверждения успешной передачи данных.
- Пакет MAC команды: используется для организации пересылок управляющих команд MAC-уровня.

- Управляющий пакет (beacon frame, кадр маяка): используется для передачи сигнальных пакетов.

1.3.2 Wi-Fi

Wi-Fi относится к WLAN-протоколам, и основан на стандарте IEEE 802.11. Wi-Fi может работать на частотах 2,4 ГГц или 5 ГГц с различными скоростями для разных стандартов, например, 24 Мбит/сек, 54 Мбит/сек, или 450 Мбит/сек. Большинство Wi-Fi-маршрутизаторов обеспечивают покрытие сигналом в радиусе около 30 м [17].

Wi-Fi является очень распространённым протоколом. Он широко применяется для организации беспроводного доступа в Интернет, а также для передачи данных между различными устройствами.

Так же, как и в Zigbee, протокол Wi-Fi имеет механизм энергосбережения. Станция Wi-Fi может работать в двух режимах: в активном и спящем. В активном режиме станция передает и принимает данные в любое время. В спящем режиме происходит отключение питания, станция не может передавать и принимать данные. Включение питания происходит при необходимости передать данные с этой станции.

В Wi-Fi используется три типа кадров:

- Кадры данных.

- Кадры контроля, служебные кадры, которые необходимы для обеспечения работы Wi-Fi.

- Кадры управления, использующиеся для реализации различных сервисов Wi-Fi, например, подключение к точке доступа Wi-Fi, или аутентификация.

В случае, если требуется передать большой объём данных, который слишком велик для одного кадра, используется механизм фрагментации, при котором один большой кадр разделяется на несколько кадров нормального

размера, при этом принимающая сторона распознаёт их, как части единого целого [18].

1.3.3 IP

IP (Internet Protocol) – протокол сетевого уровня стека TCP/IP. Протокол был создан в 1981 году и описан в RFC 791. Основной задачей протокола является доставка датаграмм между хостами сетей TCP/IP через произвольное число промежуточных узлов (маршрутизаторов) [19].

Протокол IP взаимодействует с протоколами уровня сетевого интерфейса. Данные передаются по физическим соединениям в виде кадров, содержащих заголовок и данные. В заголовке указываются адреса отправителя и получателя. В IP данные передаются в виде IP-дейтаграмм, формат которых аналогичен формату кадра. У дейтаграммы также есть заголовок, в котором содержатся IP-адреса отправителя и получателя [20].

На рисунке 4 представлена структура заголовка IPv4.

Рисунок 4 – Структура заголовка IPv4

В протоколе IP определен формат всех данных, передаваемых в Internet.

Так как соединение между контроллером и сервером, а также сервером и пользовательскими устройствами осуществляется через Интернет, данные между этими элементами сети должны передаваться именно по протоколу IP, а

значит, необходима инкапсуляция обработанных данных, хранящихся в базе данных контроллера, в этот протокол.

1.4 Выводы по главе

Проведён обзор источников по теме исследования. Введены понятия маршрутизации и инкапсуляции данных в сети. Рассмотрены три протокола динамической маршрутизации: RIP, OSPF и EIGRP, а также статическая маршрутизация, выявлены их сходства и различия, достоинства и недостатки. Проведён сравнительный анализ рассмотренных протоколов и методов маршрутизации данных, в результате которого сделан вывод, что статическая маршрутизации является наиболее подходящим вариантом для применения в разрабатываемой системе передачи телеметрической информации.

По результатам обзора источников по теме инкапсуляции данных в сети установлено, что процессы инкапсуляции и декапсуляции тесно связаны с протоколами, с которыми ведётся работа – следовательно, система инкапсуляции данных для программно-аппаратного комплекса передачи телеметрической информации должна разрабатываться с учётом используемых устройств и поддерживаемых ими протоколов передачи данных.

Полученные результаты аналитического обзора литературы по теме исследования позволяют перейти к следующему этапу работы: разработке моделей, необходимых для организации подсистем инкапсуляции и маршрутизации данных в сети передачи телеметрических данных.

2 Разработка методов инкапсуляции и маршрутизации данных в сети передачи телеметрических данных

2.1 Задания на разработку подсистем инкапсуляции и маршрутизации данных в сети передачи телеметрических данных

Модули инкапсуляции и маршрутизации данных являются частью программного обеспечения контроллера. В качестве контроллера используется медиаплеер Hyundai H-DMP100 на базе процессора Allwinner H616 [21], что определено заданием на ВКР.

Перед началом разработки программного обеспечения для контроллера программно-аппаратного комплекса передачи телеметрической информации были разработаны задания на разработку модулей декапсуляции, инкапсуляции и маршрутизации данных, входящих в состав модуля преобразования данных. В результате были определены требования к этим модулям.

К модулю преобразования данных предъявляются следующие требования:

1. Модуль должен осуществлять декапсуляцию данных, получаемых от датчиков, отделяя, таким образом, данные от служебной информации передающих протоколов.

1.1. Возможные протоколы передачи данных от датчиков: Zigbee, Wi-Fi.

2. Должна производиться обработка полученных данных, т.е. приведение их к удобному формату для дальнейшего занесения в базу данных и хранения, а также для отправки на сервер или пользовательские устройства.

3. Данные после декапсуляции должны быть доступны другим программным модулям для совершения необходимых действий над ними.

4. Для отправки данных на сервер или пользовательские устройства должна производиться инкапсуляция данных в IP-пакеты.

Требования, предъявляемые к модулю маршрутизации:

1. Для корректного обмена данными между устройствами в сети в контроллере должны быть указаны маршруты до всех серверов, имеющихся в системе, и подключаться к ближайшему из доступных.

2. Должны быть проложены маршруты до пользовательских устройств, если они находятся в одной Wi-Fi сети с контроллером.

3. Датчики и исполнительные устройства, подключаемые к контроллеру, образуют свою подсеть, в которой также нужно указывать все возможные маршруты.

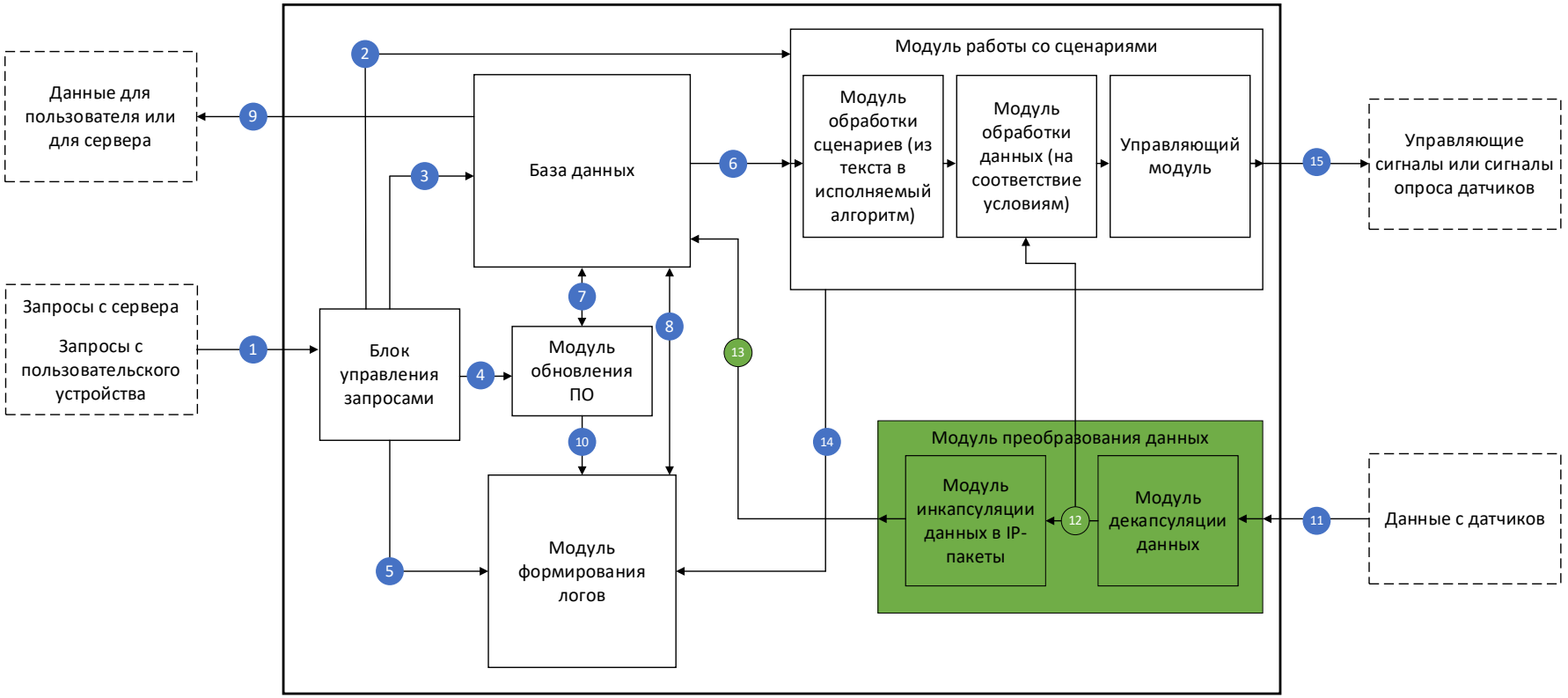
Также имеется дополнительная задача: при разработке ПО для контроллера, по возможности, сохранить изначальный функционал медиаплеера, используемого в качестве контроллера.

2.2 Структура программного обеспечения контроллера

Разрабатываемые модули являются частью программного обеспечения контроллера. В программно-аппаратном комплексе передачи телеметрической информации контроллер выступает логическим центром, на котором происходят все основные процессы системы. В связи с этим программное обеспечение контроллера состоит из многих модулей, выполняющих различные функции. На рисунке 5 представлена структурная схема ПО контроллера. Зелёным цветом на ней выделен разрабатываемый модуль преобразования данных.

Схема на рисунке 5 показывает основные модули, входящие в состав программного обеспечения контроллера, а также входные и выходные данные. Среди основных модулей можно выделить:

1. Модуль преобразования данных;
2. База данных;
3. Модуль работы со сценариями;
4. Блок управления запросами;
5. Модуль обновления ПО;
6. Модуль формирования логов.



Обозначения:

1 – Запросы с сервера или с пользовательского устройства;
2 – Запросы, адресованные модулю работы со сценариями;
3 – Запросы к базе данных;
4 – Запрос на обновление ПО контроллера;
5 – Запрос на отправку логов на сервер или пользователю;
6 – Новый сценарий из базы данных поступает в модуль работы со сценариями;
7 – Обмен данными между базой данных и модулем обновления ПО;
8 – Обмен данными между базой данных и модулем формирования логов;
9 – По запросу данные из базы данных отправляются источнику запроса;
10 – Информация об обновлении системы отправляется в модуль формирования логов;
11 – Данные от датчиков поступают в модуль преобразования данных;
12 – После удаления заголовков передающих протоколов чистые данные отправляются в модуль обработки данных;
13 – Данные, упакованные в IP-пакеты, отправляются в базу данных для хранения;
14 – Информация о работе модуля работы со сценариями отправляется в модуль формирования логов;
15 – В результате работы модуля работы со сценариями формируются управляющие сигналы для исполнительных устройств.

Рисунок 5 – Структурная схема программного обеспечения контроллера

На схеме на рисунке 5 отсутствует модуль маршрутизации данных. В дальнейших частях данной работы будет дано объяснение такому решению.

На этапе разработки структуры программного обеспечения контроллера была уточнена и структура модуля преобразования данных. В него войдут два модуля: модуль декапсуляции данных и модуль инкапсуляции данных в IP-пакеты.

Далее более подробно рассматривается структура и принципы работы модуля преобразования данных.

2.3 Принцип работы модуля преобразования данных

Модуль преобразования данных включает в себя два модуля, выполняющих разные задачи – модуль декапсуляции данных и модуль инкапсуляции данных. Основной функцией модуля преобразования данных является работа с данными, присылаемыми на котроллер различными датчиками. Эти данные нуждаются в обработке, для того чтобы другие программные модули могли использовать их. На рисунке 6 представлена более подробная структура модуля преобразования данных.

Рисунок 6 – Принцип работы модуля преобразования данных

Данные от датчиков могут поступать на контроллер в различных форматах – заданием на разработку определено, что это могут быть форматы Zigbee и WiFi. Для передачи информации по сети к битам данных присоединяются служебные заголовки указанных протоколов. Отделить данные от этих

заголовков – задача модуля декапсуляции данных. Также внутри модуля декапсуляции должна производиться обработка данных, после которой они будут приведены в формат, удобный для использования другими программными модулями. Таким образом, на выходе из модуля декапсуляции оказываются чистые данные, которые могут быть приняты в работу другим программным модулем – согласно схеме на рисунке 5, это модуль работы со сценариями.

Отправлять данные напрямую в модуль работы со сценарием представляется не оптимальным решением, поскольку в таком случае возникает необходимость в синхронизации между двумя программными модулями. Данные с датчиков используются для проверки условий сценариев, которая производится регулярно, с периодом, задаваемым пользователем. Период опроса датчиков может отличаться – для каждого датчика он определён по умолчанию, но может быть также изменён пользователем. Таким образом, в системе может возникнуть ситуация, когда датчик был опрошен незначительно короткое время назад, и нужно опросить его снова для проверки по сценарию. Чтобы не посылать лишних запросов датчикам в таких случаях, было принято решение сохранять последние показания датчиков и время, когда был произведён опрос в специально отведённой для этого области памяти. Эта область памяти также будет доступна модулю работы со сценариями. В качестве такой области памяти можно использовать базу данных на контроллере, поскольку к ней имеют доступ все программные модули, показанные на рисунке 5.

По внешнему запросу данные из БД могут быть направлены на сервер, или напрямую на пользовательское устройство. Также необходимость пересылки данных возникает при выполнении условий сценария. Тогда запускаются необходимые действия с подключенными к контроллеру датчиками и исполнительными устройствами, и требуется взаимодействовать с ними при помощи управляющих сигналов. При пересылке данных между различными устройствами в сети данные проходят через модуль инкапсуляции. Этот модуль выполняет работу, обратную предыдущему модулю: в нём данные разбиваются

на пакеты, к которым добавляются заголовки протоколов, используемых для дальнейшей пересылки данных.

На рисунке 7 представлена блок-схема алгоритма работы модуля преобразования данных.

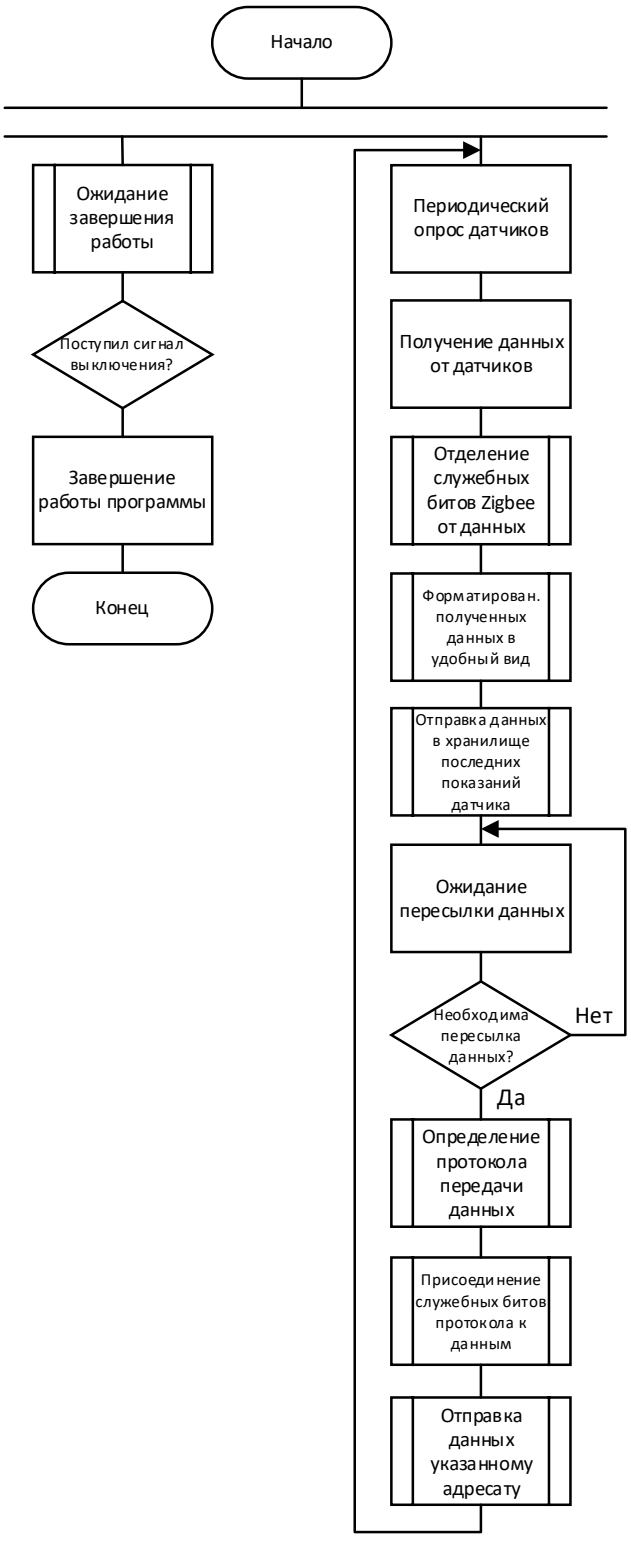


Рисунок 7 – Алгоритм работы модуля преобразования данных

Отличительной особенностью разрабатываемой системы является множество протоколов передачи данных, используемых в ней. Предложенный метод преобразования данных в сети передачи телеметрической информации позволяет обеспечить декапсуляцию данных из различных протоколов передачи данных, и инкапсуляцию их в пакеты протокола IP, что позволит обеспечить передачу данных между различными узлами сети передачи телеметрической информации.

Далее будет рассмотрен принцип маршрутизации данных в сети передачи телеметрической информации.

2.4 Маршрутизация данных в сети передачи телеметрической информации

На структурной схеме программного обеспечения контроллера (Рисунок 5) отсутствует модуль маршрутизации данных, как уже было отмечено выше. Это объясняется тем, что в результате разработки структуры программного обеспечения контроллера были пересмотрены принципы маршрутизации данных в разрабатываемой системе.

На рисунке 8 изображена схема маршрутов, возможных в системе передачи телеметрической информации с точки зрения контроллера. Маршрут между сервером и пользовательским устройством не рассматривается, так как соединение между ними происходит через Интернет при помощи приложения, устанавливающегося на пользовательское устройство.

Контроллер может подключаться к серверу, или к пользовательскому устройству. Соединение с сервером происходит через спутниковый канал, либо при помощи проводного соединения с Интернетом. Соединение с пользовательским устройством напрямую происходит по Wi-Fi, если контроллер и устройство пользователя находятся в одной сети.

Датчики и исполнительные устройства подключаются к контроллеру также посредством беспроводных сетей, например, Zigbee, как показано на рисунке 8.

На рисунке 2, демонстрирующем схему сети для одного абонента, показано три пользовательских устройства, способных соединяться с сервером и контроллером, что является верным и определено заданием на разработку системы передачи телеметрической информации. Однако в один конкретный момент времени к контроллеру может быть подключено только одно устройство, что также определено заданием на разработку, и объясняется необходимостью обеспечения безопасности и целостности данных.

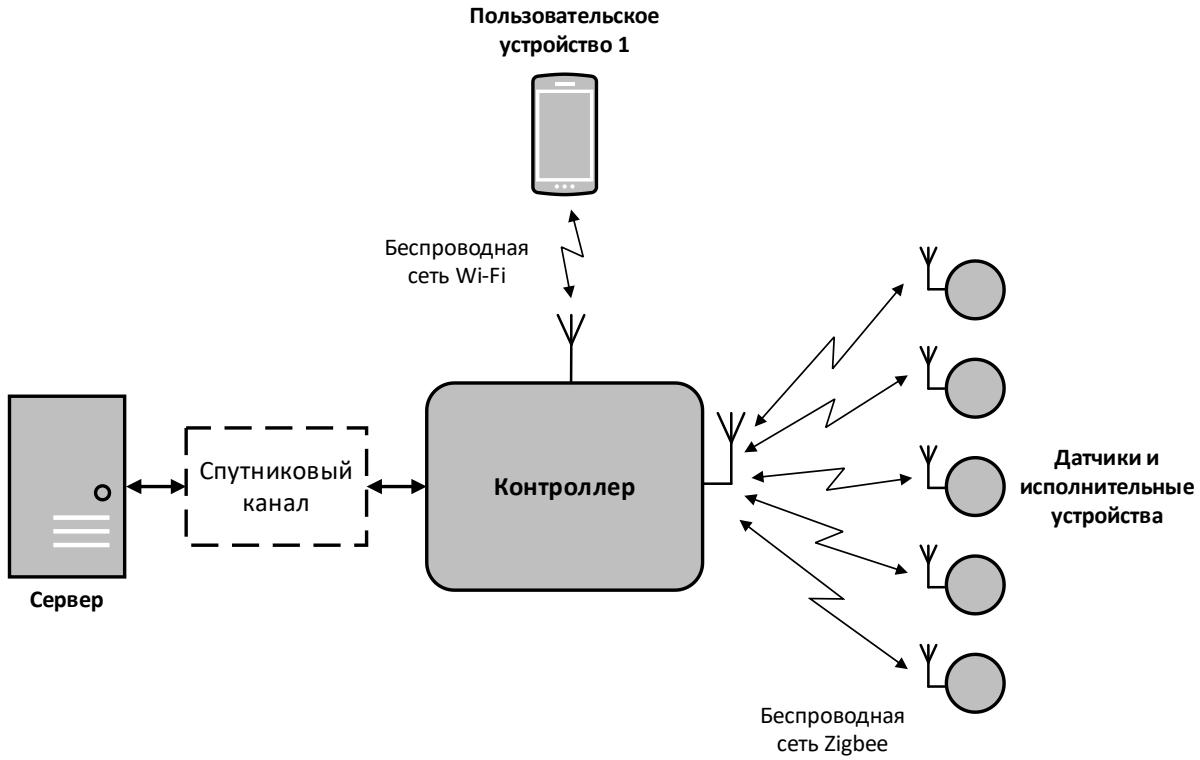


Рисунок 8 – Маршруты в сети передачи телеметрической информации

Также, согласно заданию на разработку, к контроллеру может быть подключено от одного до пяти периферийных устройств – датчиков и исполнительных устройств. Эти устройства устанавливают прямое соединение с контроллером при подключении, и в дальнейшем не устанавливают никаких новых соединений.

Таким образом, сеть, логическим центром которой является контроллер, исполняющий, в том числе, роль маршрутизатора, имеет топологию «звезда». При этом для различных устройств применяются различные типы соединений – спутниковый канал либо LAN, беспроводное соединение по протоколам WiFi, Zigbee. Маршруты в такой сети, как показано на рисунке 8, известны заранее.

Задача маршрутизации данных в таком случае сводится не к построению маршрутов и поиску наиболее выгодного из них, а к установлению первоначального соединения между устройствами и дальнейшему направлению пакетов данных по нужному маршруту. В связи с этим было принято решение не выделять маршрутизацию данных в отдельный программный модуль. Под программным модулем в данном случае подразумевается программа, выполняющаяся параллельно с другими процессами на контроллере. Вместо этого маршрутизацию данных предлагается реализовать в виде набора функций, или библиотеки. Эти функции будут доступны для вызова из других программных модулей, и будут обеспечивать отправку данных по нужному маршруту. Каждая из таких функций будет отвечать за определённый маршрут, учитывая необходимые форматы данных и особенности различных типов соединений. Таким образом, предлагается разработать следующие функции:

1. Функция передачи данных на сервер;
2. Функция передачи данных на устройство пользователя;
3. Функция передачи данных на периферийные устройства.

Для передачи данных посредством протоколов WiFi и Zigbee предлагается разработать функции-посредники между функциями передачи данных и модулем преобразования данных, так как данная задача относится одновременно и к маршрутизации, и к инкапсуляции и декапсуляции данных.

2.5 Выводы по главе

Разработаны задания для дальнейшей разработки модуля преобразования данных и системы маршрутизации данных в сети передачи телеметрической информации.

В составе рабочей группы проекта разработки системы передачи телеметрической информации разработана структура программного обеспечения контроллера, пересмотрены принципы маршрутизации данных в разрабатываемой системе.

На основе задания на ВКР, задания на разработку, и в рамках структуры программного обеспечения контроллера описана структура модуля преобразования данных, определены его функции, описан принцип его работы и предложен алгоритм работы модуля преобразования данных.

Выявлены и описаны все маршруты в подсети системы передачи телеметрической информации, центром которой является контроллер, выполняющий роль маршрутизатора, определена топология этой сети. Определена задача маршрутизации в данной сети, предложен метод реализации маршрутизации. Перечислены функции, которые необходимо разработать.

Полученные результаты разработки моделей, необходимых для организации подсистем инкапсуляции и маршрутизации данных в сети передачи телеметрических данных, позволяют перейти к разработке программных моделей и программных средств из состава подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных.

3 Разработка программных моделей и программных средств из состава подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных

3.1 Эксперименты до начала разработки подсистемы инкапсуляции и маршрутизации данных

При постановке задачи на разработку программного обеспечения контроллера была определена дополнительная задача: сохранить функционал медиаплеера Hyundai H-DMP100, взятого в качестве прототипа контроллера сети передачи телеметрических данных. Данный медиаплеер имеет предустановленную операционную систему Android с установленным программным обеспечением для воспроизведения медиафайлов и просмотра онлайн-ТВ. Для исследования возможности выполнения этой задачи был проведён ряд экспериментов по совместимости необходимого для разработки программного обеспечения с ОС Android.

Так как Android является Unix-подобной операционной системой, имеется возможность работать в ней при помощи терминала. Для этого необходимо установить приложение, эмулирующее терминал. Для проведения экспериментов было выбрано приложение Termux [22], которое является одним из самых популярных эмуляторов терминала, а также позволяет обеспечить пользователю root-права, что является важным фактором для проведения исследования.

В ходе экспериментов проверялась совместимость ОС Android с программным обеспечением Zigbee2MQTT, которое позволяет использовать устройства Zigbee, а также совместимость с MQTT-брокером Mosquitto.

Проведённые тесты показали, что Mosquitto устанавливается на Android без ошибок и функционирует исправно. Zigbee2MQTT также устанавливается на Android без ошибок, с учётом предвариательной установки недостающих в

данной ОС библиотек и программных компонентов. Однако при настройке Zigbee2MQTT возникли проблемы, связанные с подключением USB-стика Zigbee, выполняющего роль маршрутизатора сети Zigbee-устройств. Так как файловая система ОС Android отличается от файловой системы ОС Ubuntu и подобных ей, вызывает затруднение определить, как именно определяется подключаемое USB-устройство. Также после определения файла устройства из-за отсутствия root-прав у пользователя невозможно изменить пользователя – владельца файла, что необходимо для работы модуля MQTT.

Также был проведён эксперимент по установке на Android ПО для домашней автоматизации Home Assistant [23], для проверки, насколько исследуемая ОС подходит для развёртывания систем, подобных разрабатываемой наземной части системы передачи телеметрических данных. Эксперимент показал, что установка Home Assistant выполняется медленно и сопровождается множеством ошибок из-за отсутствующих библиотек, которые необходимо устанавливать дополнительно.

Таким образом, проведённые эксперименты показали, что операционная система Android подходит для использования на контроллере передачи телеметрической информациии, но только при условии предварительной установки всего необходимого программного обеспечения и его настройке. При этом процесс настройки устанавливаемого программного обеспечения на Android отличается от аналогичного на других ОС, таких, как Ubuntu, и, как показали проведённые эксперименты, занимает значительно больше времени.

По результатам проведённых экспериментов был сделан вывод, что на данном этапе разработки системы передачи телеметрической информации и, в частности, подсистемы инкапсуляции и маршрутизации данных, использование операционной системы Android усложнит и существенно замедлит процесс разработки. В связи с этим было принято решение выбрать для дальнейшей разработки другую операционную систему. Также был сформирован перечень программных и инструментальных средств для разработки подсистемы инкапсуляции и маршрутизации данных. Этот перечень рассматривается ниже.

3.2 Выбор программных и инструментальных средств для разработки подсистемы инкапсуляции и маршрутизации данных

Первым этапом в разработке системы инкапсуляции и маршрутизации данных стало создание программной модели данной системы. Программная модель разработана на основе метода инкапсуляции и маршрутизации данных, предложенного в главе 2 данной пояснительной записки. Программная модель развёрнута на виртуальной машине под управлением Unix-подобной операционной системы. Для разработки программной модели был выбран ряд программных и инструментальных средств, перечень которых представлен ниже.

1. Язык программирования C++ выбран для разработки модуля обработки преобразования данных, а также библиотеки функций маршрутизации данных. Выбор данного языка обусловлен универсальностью и кросплатформенностью, высокой скоростью выполнения программного кода, а также имеющийся опыт программирования на данном языке.

2. Среда разработки Code::Blocks [24]. Свободно распространяемая кросплатформенная среда разработки. Была выбрана для разработки программной модели из-за свободного распространения, возможности установки на Unix-подобные операционные системы и удобного интерфейса.

3. База данных MySQL [25]. Свободная реляционная система управления базами данных. Причины, по которым была выбрана данная СУБД: работа с реляционными базами данных, свободное распространение, большое количество документации по работе с MySQL.

4. Операционная система Linux Mint 21.3 «Virginia» [26]. Свободно распространяемый дистрибутив Linux, основанный на Ubuntu и Debian. Данная система была выбрана в качестве ОС для разрабатываемой программной модели ввиду свободного распространения, удобного интерфейса и стабильности работы на виртуальной машине.

5. Виртуальная машина Oracle VM VitualBox [27]. Программное обеспечение для создания виртуальных машин. Причиной, по которой была выбрана эта система виртуализации, является её бесплатное распространение и удобство интерфейса.

6. Zigbee2MQTT [28]. Программное обеспечение, которое позволяет использовать устройства Zigbee без использования моста или шлюза конкретных производителей. Это позволяет связывать события и управлять устройствами Zigbee с помощью протокола MQTT. Данная программа является бесплатной, и распространяется с открытым исходным кодом.

7. Eclipse Mosquitto [29] – брокер сообщений протокола MQTT с открытым исходным кодом, который реализует протоколы MQTT версий 5.0, 3.1.1 и 3.1. Необходим для обеспечения работы модуля Zigbee2MQTT, и для возможности отправки сообщений различным устройствам.

8. Sonoff Zigbee 3.0 USB Dongle Plus (Рисунок 9) [30]. Универсальный Zigbee USB-стик. Позволяет создавать сети из множества Zigbee-устройств. Утройство выполняет роль шлюза в Zigbee-сети. Поддерживает подключение многих типов устройств, независимо от производителя, что является важным фактором при выборе подобного устройства.

Рисунок 9 – USB-стик Sonoff Zigbee 3.0 USB Dongle Plus

9. Датчик открытия дверей и окон Aqara Door and Window Sensor (Рисунок 10) [31]. Zigbee датчик открытия, состоящий из двух частей: самого датчика, и магнитной составляющей. При закрытии двери или окна между частями происходит контакт, информация о чём передаётся на шлюз Zigbee-

сети. Данное устройство было предоставлено консультантом для экспериментов в ходе выполнения ВКР.

Рисунок 10 – Датчик открытия дверей и окон Aqara Door and Window Sensor

Выбор описанных выше программных и инструментальных средств позволил перейти к следующему этапу работы: подготовке к разработки программной модели.

3.3 Подготовка к разработке программных моделей модуля преобразования данных и библиотеки функций для маршрутизации данных

Для разработки программных моделей модуля преобразования данных и библиотеки функций для маршрутизации данных необходимо подготовить рабочее место с установленными программными средствами, и с поддержкой всех необходимых инструментальных средств.

Для создания такого рабочего места при помощи системы виртуализации VirtualBox создана виртуальная машина с операционной системой Linux Mint. Установлена среда разработки Code::Blocks. Также на виртуальной машине развёрнута база данных MySQL, где создана тестовая база данных с таблицами для хранения данных о датчиках и исполнительных устройствах, с внесённым тестовым набором данных.

Для обеспечения виртуальной машины связи с USB-стиком Sonoff настроены USB-порты в VirtualBox.

Произведена установка и настройка Zigbee2MQTT и сопутствующих программных средств: брокера MQTT Mosquitto и Node.js. Проведено тестирование подключения USB-стика и датчика открытия дверей при помощи встроенного Web-интерфейса Zigbee2MQTT. На рисунке 11 показан скриншот интерфейса, на котором видно подключенный к сети Zigbee датчик открытия двери.

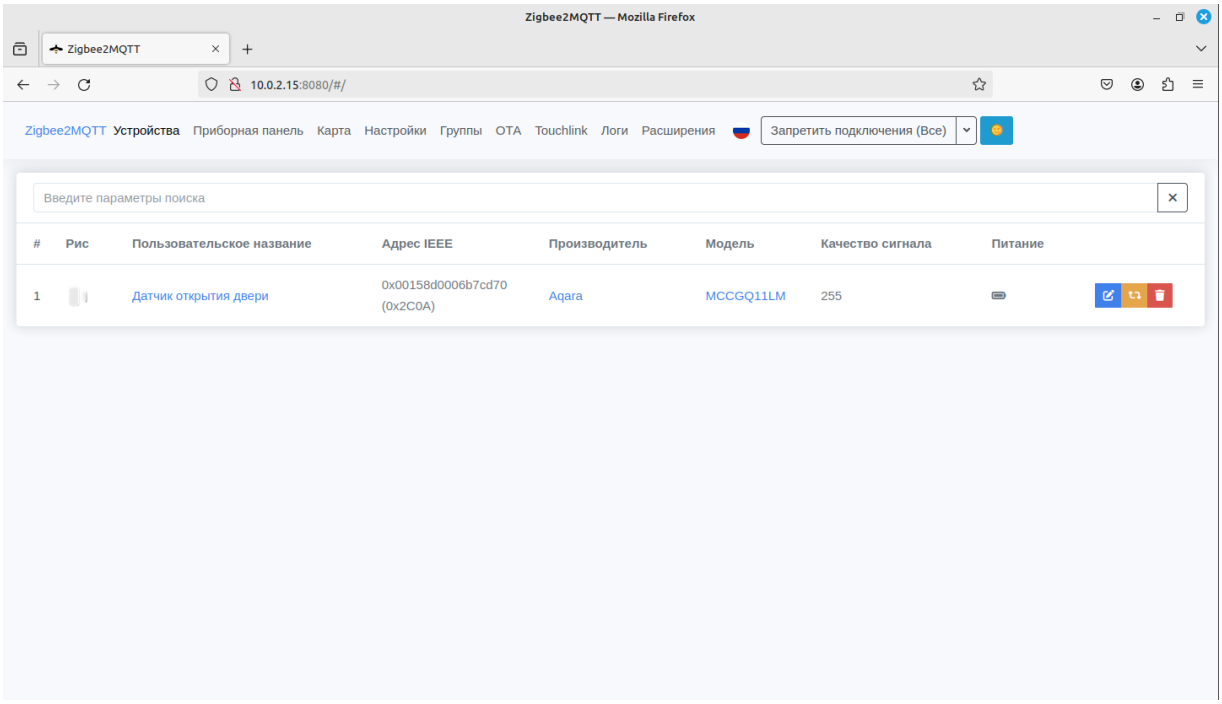


Рисунок 11 – Web-интерфейс Zigbee2MQTT

Отслеживать состояние подключенных Zigbee-устройств можно как при помощи Web-интерфейса, так и через терминал Linux Mint, куда направляется стандартный вывод Zigbee2MQTT. На рисунке 12 представлен скриншот такого вывода. По сообщениям, представленным на рисунке, можно увидеть, что сообщения от датчика открытия двери публикуются с заголовком темы «zigbee2mqtt/Датчик открытия двери». Если между двумя частями датчика установлен контакт (дверь закрыта), поле «contact» принимает значение true, если же контакта нет, в это поле устанавливается значение false.

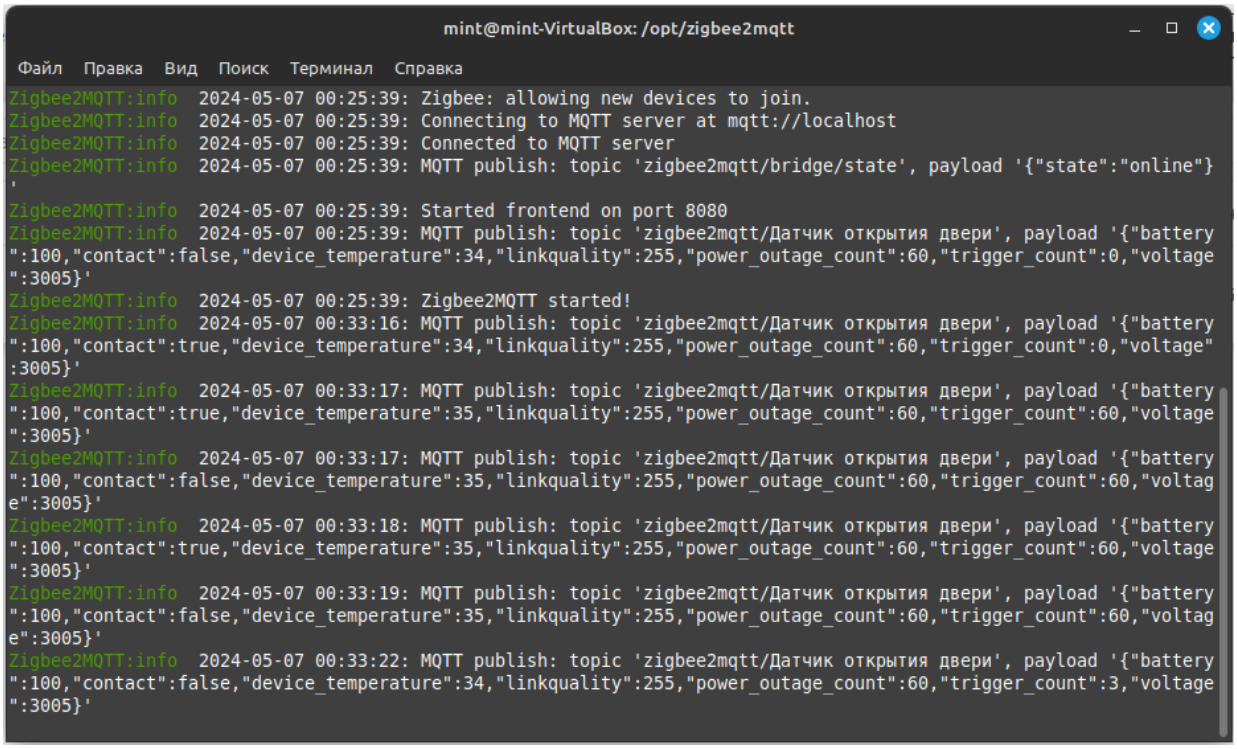


Рисунок 12 – Вывод программы Zigbee2MQTT в окно терминала Linux Mint

Проведённая подготовка является достаточной для перехода к непосредственно разработке программных моделей модуля преобразования данных и библиотеки функций для маршрутизации данных.

3.4 Разработка программных моделей модуля преобразования данных и библиотеки функций для маршрутизации данных

Для создания подсистем инкапсуляции и маршрутизации данных в составе системы передачи телеметрических данных необходимо разработать модуль преобразования данных, а также библиотеку функций, необходимых для маршрутизации данных. Модуль преобразования, в свою очередь, состоит из модуля декапсуляции и модуля инкапсуляции данных. Рассмотрим процесс разработки программной модели каждого из них.

3.4.1 Разработка модуля декапсуляции данных

Как было отмечено в главе 3.1 данной работы, при разработке программной модели систем инкапсуляции и маршрутизации данных используется Zigbee-датчик открытия двери. Соответственно, на этом этапе разработки идёт работа с декапсуляцией данных, передаваемых по протоколу Zigbee.

Программный модуль Zigbee2MQTT, установленный и настроенный на этапе подготовки рабочего места, позволяет передавать и принимать информацию от устройств Zigbee по протоколу MQTT. MQTT – это протокол обмена сообщениями по шаблону издатель-подписчик. Каждое устройство, подключенное к контроллеру (в случае программной модели – к виртуальной машине), подписывается на сообщения с определённым заголовком, публикуемым брокером. Устройства могут и самостоятельно публиковать сообщения с заголовками по имени устройства [32].

Таким образом, на этапе подготовки все сообщения от подключенного к виртуальной машине датчика проходили через модуль Zigbee2MQTT. Проведённые тесты показали, что такое подключение стабильно, и потерь информации при получении сообщений не происходит. В связи с успешными результатами тестов на подготовительном этапе было принято решение использовать Zigbee2MQTT в качестве декапсулятора данных, передаваемых по протоколу Zigbee. Этот программный модуль работает с пакетами данных Zigbee, обрабатывает их, проводя декапсуляцию, и представляет данные в формате JSON.

Таким образом, в разрабатываемой программной модели модуль декапсуляции данных является связующим звеном между модулем Zigbee2MQTT, базой данных, где хранятся данные об устройствах, модулем обработки данных, куда отправляются данные с датчиков, и модулем инкапсуляции данных, где эти данные проходят преобразование перед отправкой в базу данных, или на другие устройства.

Для представления датчиков и исполнительных устройств в программной модели используются классы Sensor и Terminal.

Класс Sensor является программным представлением датчиков, подключаемых к контроллеру. Каждый из них представляется в программе в виде объекта данного класса. Класс Sensor имеет следующие поля:

- name – имя датчика;
- id – идентификатор;
- dataType – тип данных, получаемых от датчика;
- lastData – последние значение, полученное от датчика;
- protocolType – протокол, при помощи которого происходит обмен данными с датчиком.

Класс Sensor содержит методы, позволяющие установить значение для каждого поля, и методы, возвращающие текущие значения полей. Для класса Sensor разработаны методы для записи данных датчика в базу данных, и их чтения из БД.

Класс Terminal служит для представления исполнительных устройств, подключаемых к контроллеру. Каждый объект этого класса является представлением одного исполнительного устройства. Данный класс содержит поля name, id и protocol, в которых хранятся соответственно имя, идентификатор устройства и вид протокола. Этот класс содержит методы, позволяющие установить значение для каждого поля, и методы, возвращающие эти значения.

На рисунке 13 представлен алгоритм работы разработанного модуля декапсуляции данных.

При запуске разработанной программной модели сначала происходит инициализация устройств: данные о них считываются из базы данных. Для каждого устройства создаётся объект класса Sensor или Terminal, который заносится в список (list) объектов каждого класса. После инициализации в бесконечном цикле начинается взаимодействие модели с модулем Zigbee2MQTT.

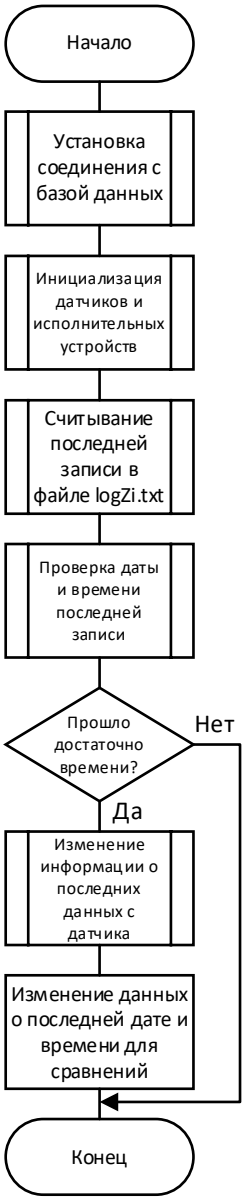


Рисунок 13 – Алгоритм работы модуля декапсуляции данных из протокола Zigbee

Так как Zigbee2MQTT является сторонним модулем, и не имеет API для работы с языками C/C++, получить данные от этого модуля возможно считыванием его стандартного вывода.

Для удобства считывания вывода Zigbee2MQTT его вывод при запуске модуля перенаправляется в текстовый файл при помощи команды echo. Модуль декапсуляции считывает этот файл, и по дате и времени последней записи определяет, были ли внесены изменения в файл после последней проверки. Если были внесены изменения, определяется тема сообщения, то есть его заголовок.

Так как у каждого Zigbee-устройства имеется уникальная тема, в которой оно публикует сообщения, по теме возможно определить устройство, от которого получены данные. Эти данные записываются в поле lastData объекта класса Sensor, а также направляются в базу данных и модуль инкапсуляции, о котором речь пойдёт ниже.

3.4.2 Разработка модуля инкапсуляции данных

Модуль инкапсуляции данных необходим для подготовки данных, полученных от датчиков, к дальнейшей отправке на сервер, или на исполнительные устройства в качестве управляющих сигналов. Так как в разрабатываемой системе передачи телеметрической информации планируется использовать несколько протоколов связи с подключаемыми устройствами, для модуля инкапсуляции важно определять, с каким протоколом будет идти работа в конкретный момент времени.

Для каждого протокола имеется свой способ обработки данных, поскольку различными будут и способы отправки данных на различные устройства. Рассмотрим используемые методы подключения устройств.

В разрабатываемой программной модели используется протокол Zigbee. Кроме него, рассмотрим также работу с устройствами, подключаемыми по Wi-Fi.

При подключении Zigbee-устройства работа с ним идёт через модуль Zigbee2MQTT. Кроме возможности декапсулировать данные, такое подключение имеет средства и для инкапсуляции данных и отправки сообщений на Zigbee-устройства с контроллера. Это возможно благодаря тому, что вместе с модулем Zigbee2MQTT установлен MQTT-брокер Mosquitto. Он позволяет публиковать сообщения в любых темах, что позволит как организовывать широковещательные рассылки на всю Zigbee-сеть, так и управлять исполнительными устройствами, принимающих управляющие сигналы по этому протоколу. Передача сообщений в этом случае происходит при помощи

вызова команды «mosquitto_pub», в аргументах которой необходимо указать тему сообщения и его содержание в формате JSON.

В случае подключения устройств по Wi-Fi данные будут отправляться при помощи стека протоколов TCP/IP, как и в случае связи с сервером. Для отправки данных таким образом необходимо составить пакет для передачи данных, и передать его принимающей стороны, например, при помощи сокетов. В случае связи с сервером эта задача осложняется ограниченностью размера пропускного канала из-за используемого канала спутниковой связи. Следовательно, необходимо минимизировать количество передаваемых данных, оставив только самые важные из них.

Таким образом, главной задачей модуля инкапсуляции данных становится правильное определение способа передачи данных для каждого конкретного случая, и правильное составление пакета данных или команды для передачи.

На рисунке 14 представлен алгоритм работы модуля инкапсуляции данных.

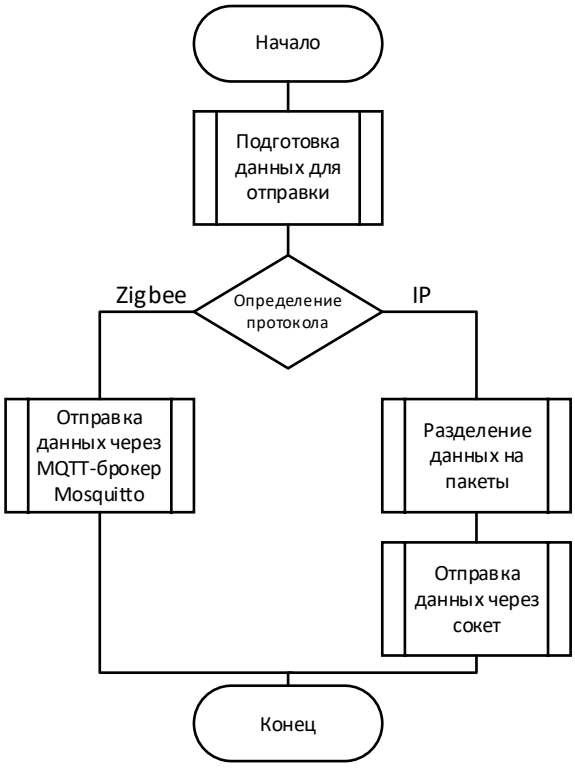


Рисунок 14 – Алгоритм работы модуля инкапсуляции данных

В разрабатываемой программной модели модуль инкапсуляции данных реализован в виде функций switch_encapsulation, encap_Zigbee и incap_IP. Первая функция принимает в качестве аргументов передаваемые данные и информацию о том, куда их необходимо передать. Задача этой функции: определить, по какому протоколу должна осуществляться передача данных, и, в зависимости от протокола, вызвать одну из функций encap_Zigbee и incap_IP. Эти функции осуществляют процесс инкапсуляции данных в разные протоколы: Zigbee и IP соответственно.

Функция encap_Zigbee принимает в качестве аргумента строку передаваемых данных, которая хранит команду для MQTT-брокера Mosquitto. Функция направляет эти данные брокеру.

Функция incap_IP использует сокет для передачи данных нужному адресату в сети. Так как в разрабатываемой модели используются только устройства, подключаемые по протоколу Zigbee, данная функция находится в стадии доработки.

Важной частью модуля инкапсуляции данных является библиотека функций для маршрутизации данных. Рассмотрим её подробнее.

3.4.3 Разработка библиотеки функций для маршрутизации данных

Библиотека функции для маршрутизации данных является вспомогательным элементом в разрабатываемой программной модели. Она тесно связана с модулем преобразования данных, а также с модулем обработки данных, входящим в состав модуля работы со сценариями. Оба этих модуля используют функции маршрутизации для отправки данных с контроллера на разные устройства.

В состав библиотеки входят следующие функции:

- Функция включения исполнительных устройств;
- Функция выключения исполнительных устройств;

- Функция отправки данных указанному адресату в сети по протоколу Zigbee;

- Функция отправки данных указанному адресату в сети при помощи стека протоколов TCP/IP;

- Функция отправки сообщения на указанный номер телефона. Одно из возможных действий, доступных для выполнения по сценарию.

Все эти функции являются действиями, способными выполняться при выполнении сценариев, но могут вызываться и по иным событиям.

Для каждого способа передачи данных необходима своя функция ввиду больших различий между ними.

На данный момент библиотека находится на этапе реализации. Полностью реализованы функции включения и выключения исполнительных устройств по протоколу Zigbee.

Полученные в ходе разработки программной модели результаты позволяют перейти к этапу тестирования.

3.5 Выводы по главе

До начала процесса разработки проведены эксперименты с целью определить, подходит ли операционная система Android в качестве ОС контроллера сети передачи телеметрических данных. По результатам экспериментов установлено, что использование ОС Android усложняет и замедляет процесс разработки системы инкапсуляции и маршрутизации данных.

На основе результатов проведённых исследований выбраны программные и инструментальные средства для разработки программной модели системы синхронизации данных в сети передачи телеметрической информации, в т.ч. язык программирования C++, среда разработки Code::Blocks, база данных MySQL, операционная система Linux Mint 21.3 «Virginia», виртуальная машина Oracle VM VitualBox, модуль Zigbee2MQTT, брокер Eclipse Mosquitto, Zigbee

USB-стик Sonoff Zigbee 3.0 USB Dongle Plus, датчик открытия дверей и окон Aqara Door and Window Sensor.

Проведена установка и настройка выбранных программных средств в рамках подготовки к процессу разработки программной модели систем инкапсуляции и маршрутизации данных.

Разработана программная модель модуля преобразования данных, в состав которого вошли:

- модуль декапсуляции данных;
- модуль инкапсуляции данных;
- библиотека функций для маршрутизации данных.

Разработанная программная модель реализует предложенный метод инкапсуляции и маршрутизации данных. Разработанные модули позволяют работать с протоколом Zigbee. Частично реализованы функции для работы с протоколом IP.

Полученные результаты позволяют перейти к тестированию разработанных моделей и программных средств.

4 Тестирование разработанных моделей и программных средств

Проведено тестирование разработанной программной модели модуля преобразования данных. Для тестирования использовался набор тестовых данных, включающий в себя четыре датчика и два исполнительных устройства. Информация об устройствах, в виде, внесённом в базу данных, приведена в таблицах 1 и 2.

Из четырёх датчиков, указанных в таблице 1, Датчик открытия двери является устройством, действительно подключенным к виртаульной машине, а DAT1, D2 и Sens3 являются тестовыми значениями.

Таблица 1 – Тестовый набор датчиков

id	Имя	Тип данных	Последние данные	Протокол
1	DAT1	int	0	Zigbee
2	D2	float	0	Wi-Fi
3	Датчик открытия двери	bool	0	Zigbee
4	Sens3	float	0	Wi-Fi

Таблица 2 – Тестовый набор исполнительных устройств

№	id	Имя	Протокол
1	11	term1	Zigbee
2	12	TERM2	Wi-Fi

Тестирование проводилось поэтапно, сначала в отдельности для каждого модуля, а после совместно. Все этапы тестирования описаны ниже.

4.1 Тестирование алгоритма работы модуля декапсуляции данных

На начальном этапе проводилось тестирование правильности работы алгоритма модуля декапсуляции данных. На этом этапе не использовалась база данных, а данные о датчиках и исполнительных устройствах явно задавались в коде программы.

Для обеспечения возможности считывать вывод Zigbee2MQTT модулем декапсуляции данных стандартный вывод этого модуля был перенаправлен в текстовый файл logZi.txt. На рисунке 15 представлен скриншот такого вывода.

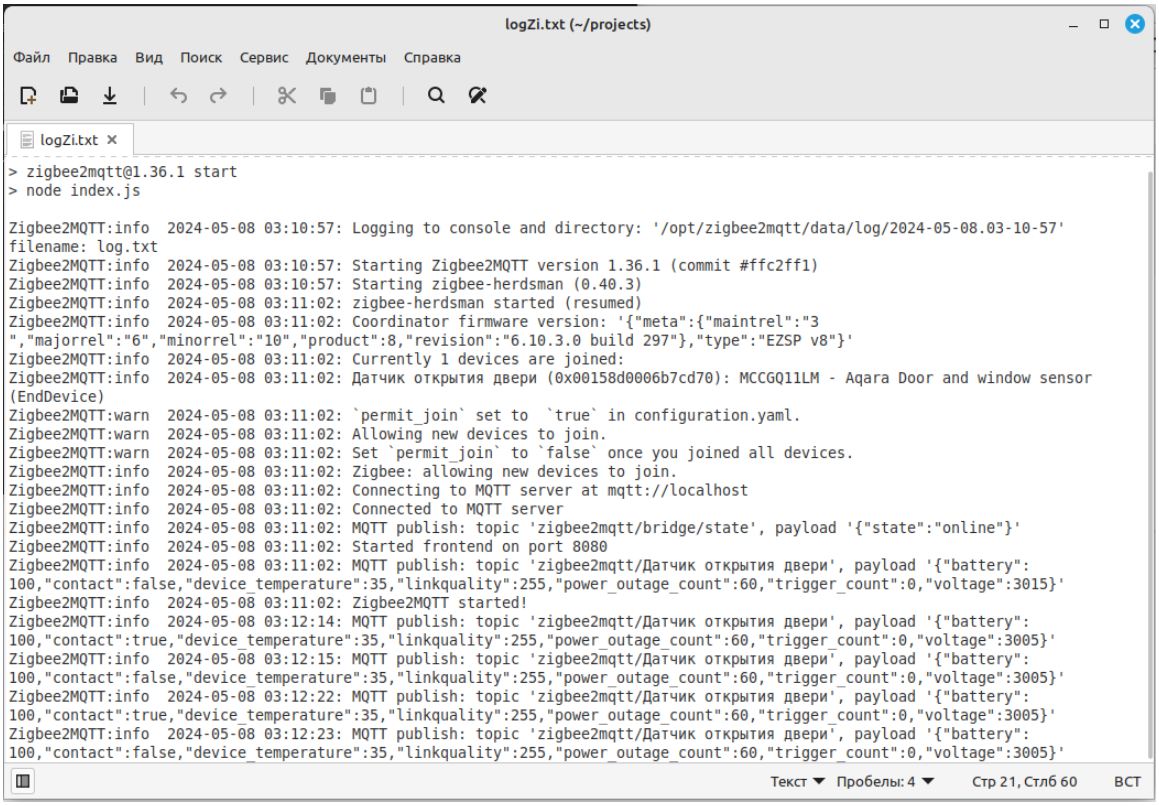


Рисунок 15 – Вывод программы Zigbee2MQTT в текстовый файл

После начала выполнения программы, содержащей модуль декапсуляции данных, происходит инициализация датчиков и исполнительных устройств. На рисунке 16 представлен скриншот вывода программы.

При обнаружении новой записи в файле logZi.txt программа в целях тестирования выводит эту запись в терминал, что можно увидеть на рисунке. Строка «Последние данные датчика «Датчик открытия двери»: 1» служит для удобства отладки, и содержит имя и последние данные устройства, связанного с последней записью в файле logZi.txt.

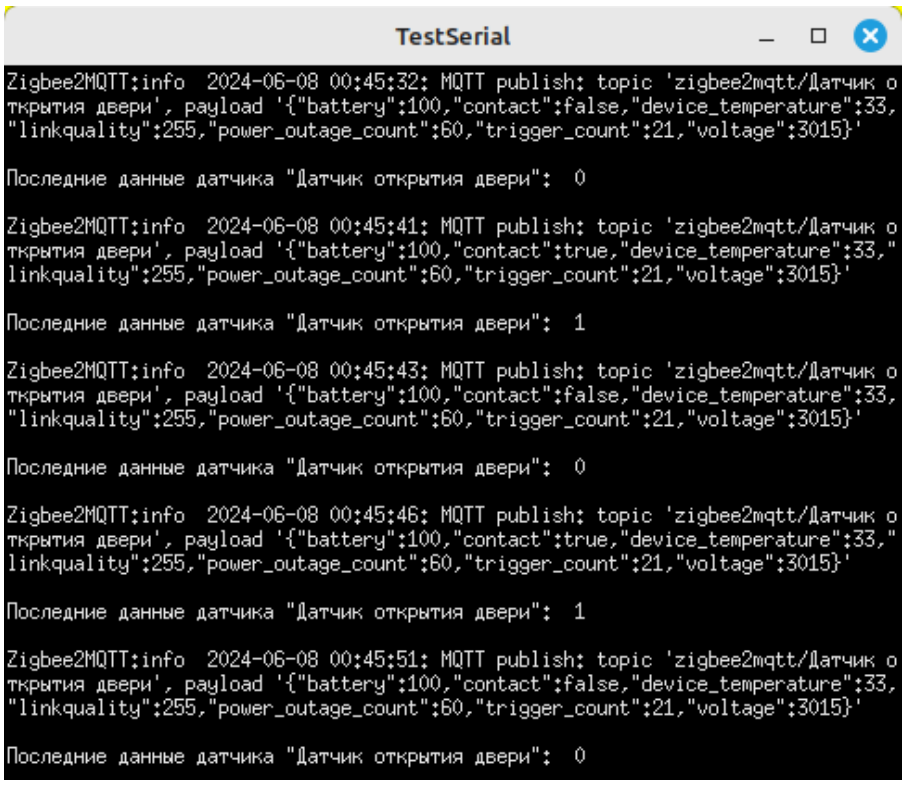


Рисунок 16 – Вывод модуля декапсуляции данных

В случае, если модуль Zigbee2MQTT не запущен, или остановлен, программа также остановится, что видно на рисунке 17.

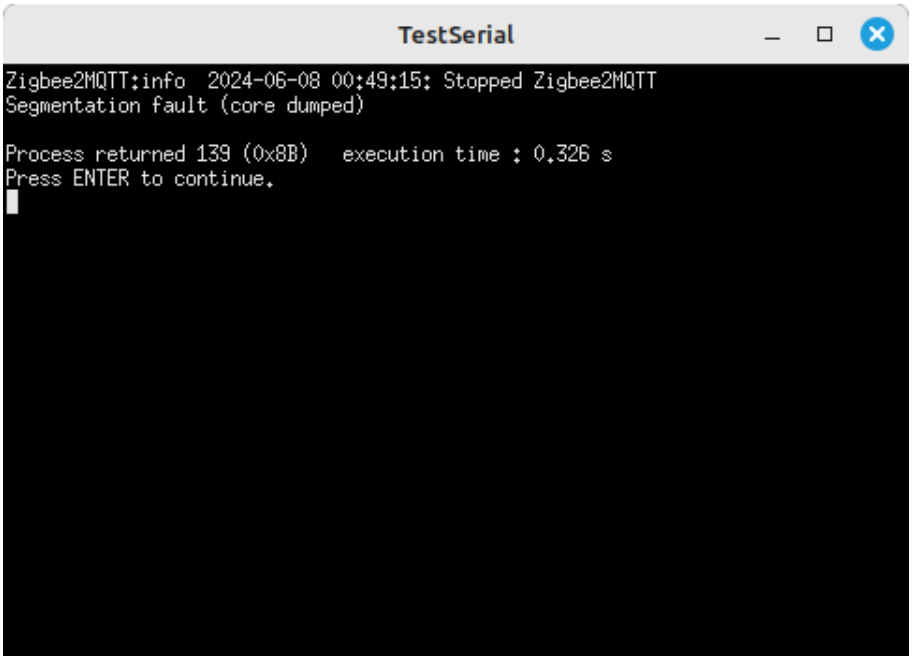


Рисунок 17 – Запуск модуля декапсуляции данных при выключенном Zigbee2MQTT, но существующем файле logZi.txt

На рисунке 18 представлена ситуация, когда файл logZi.txt не существует. В этом случае программа также прекращает работу, выводя на экран сообщение об ошибке.

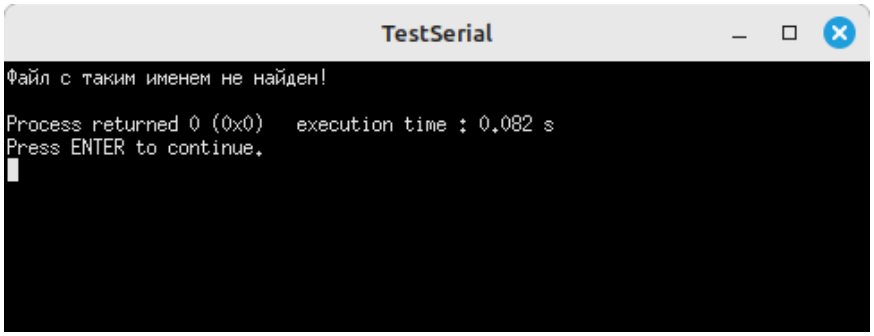


Рисунок 18 – Запуск модуля декапсуляции данных при выключенном Zigbee2MQTT и не существующем файле logZi.txt

На первом этапе тестирования модуля декапсуляции данных определено, что разработанная программная модель данного модуля работает в соответствии с алгоритмом, описанным в главе 3 данной пояснительной записки.

4.2 Тестирование программной модели модуля декапсуляции данных совместно с базой данных

На втором этапе в разрабатываемую программную модель была внедрена база данных. На рисунке 19 представлен скриншот вывода программы.

Две строки «Success!» в начале вывода говорят о том, что дважды произошло успешное подключение к базе данных – для считывания данных о датчиках и исполнительных устройствах соответственно. Также для удобства отладки и тестирования выводится SQL-запрос к базе данных на обновление информации о последних данных датчика.

При обнаружении в файле logZi.txt новой записи информация о последних данных датчика, выводимая на экран, не считывается из базы данных, а берётся из объекта класса Sensor, соответствующего нужному устройству. Как можно

увидеть на скриншоте, эти данные всегда соответствуют информации, вносимой в базу данных.

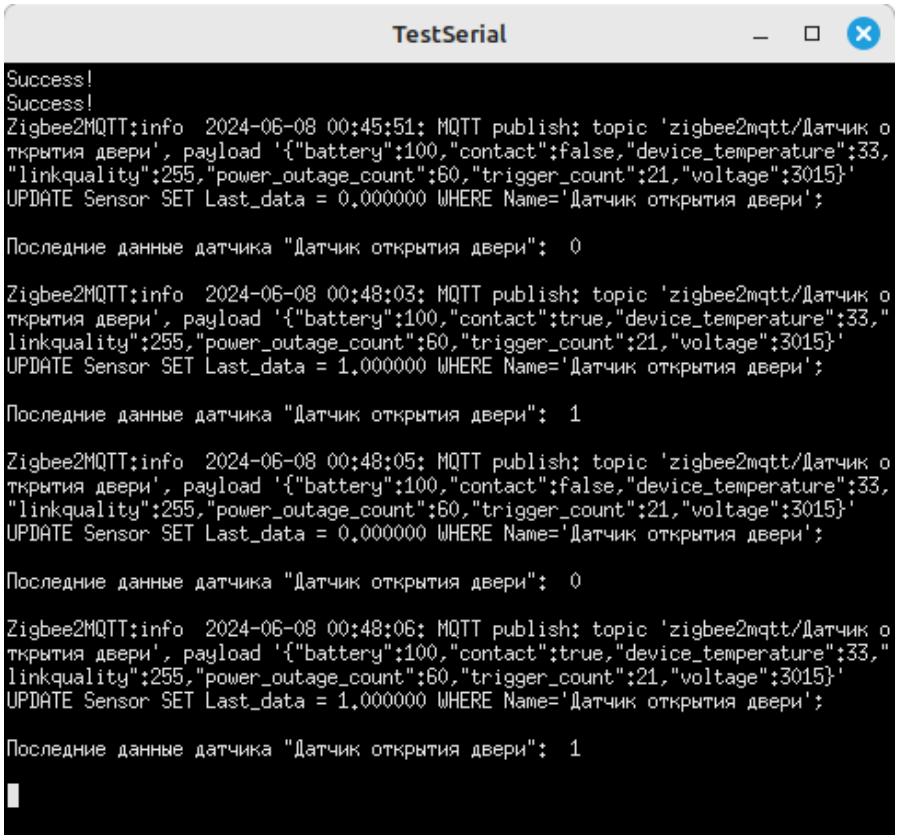


Рисунок 19 – Работа модуля декапсуляции данных с использованием БД

Результаты, полученные на втором этапе тестирования, оказались положительными, что позволило перейти к следующему этапу.

4.3 Тестирование модуля декапсуляции данных совместно с модулем управления потоками данных

На третьем этапе тестирования проводилась проверка совместимости программной модели модуля декапсуляции данных с программной моделью модуля управления потоками данных, который осуществляет работу со сценариями. Для проверки условий исполнения сценария используются последние данные с датчиков.

На рисунке 20 можно увидеть результат совместной работы двух модулей.

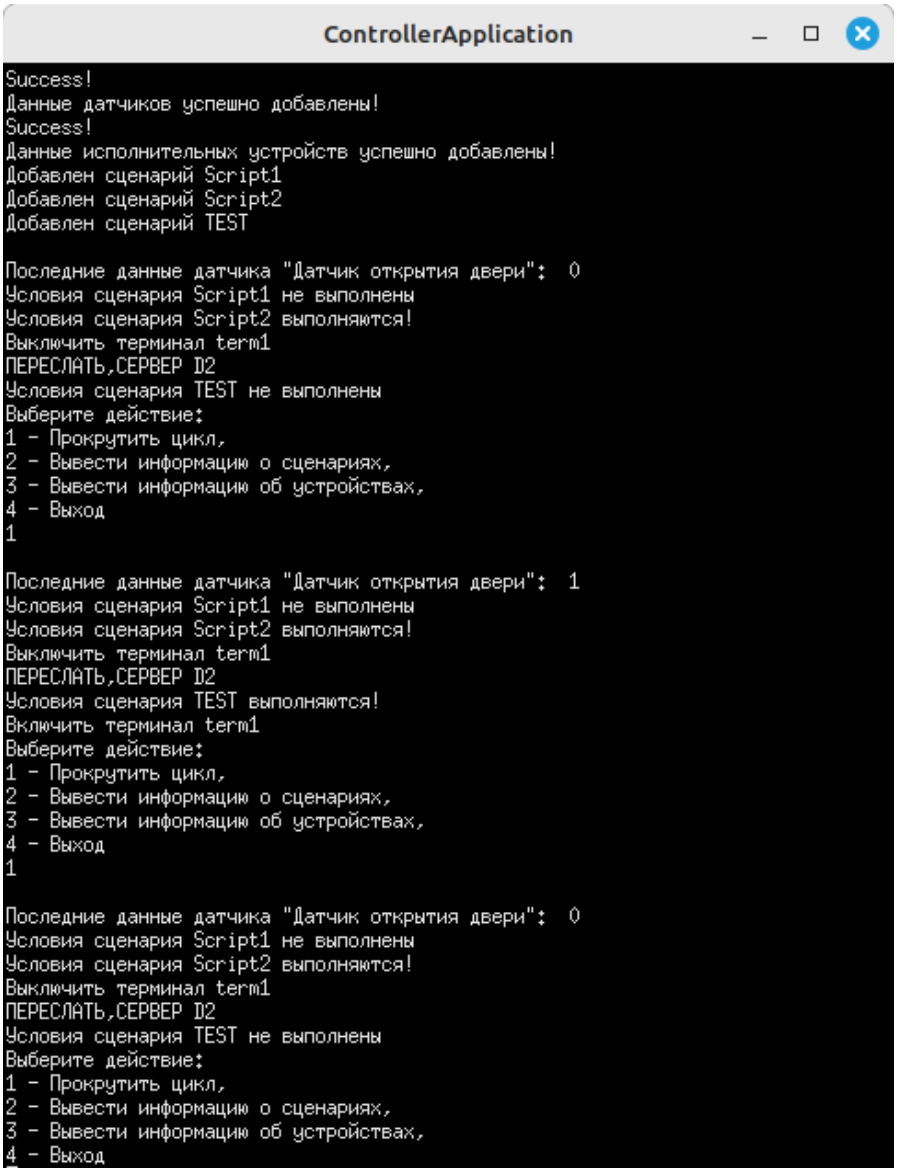


Рисунок 20 – Взаимодействие модулей преобразования данных и управления потоками данных

Данные с датчика, принимаются модулем декапсуляции данных в режиме рельного времени и заносятся в базу данных. Информация о последних прочитанных данных с датчика выводится на экран. В зависимости от изменения данных, меняется результат работы модуля управления потоками данных, проверяющего условия выполнения сценария. В тесте на рисунке 20 сценарий TEST выполняется, если значение данных датчика открытия двери равняется 1.

Полученные в ходе тестирования результаты являются положительными. Разработанная программная можель модуля преобразования данных совместима

с модулем управления потоками данных, и позволяет осуществлять работу сценариев.

4.4 Тестирование работы модуля инкапсуляции данных

На следующем этапе проводилось тестирование модуля инкапсуляции данных. Этот модуль тестировался совместно с модулем декапсуляции данных и модулем управления потоками данных. Также были протестированы разработанные функции из состава библиотеки для маршрутизации данных. Результат работы программной модели представлен на рисунке 21.

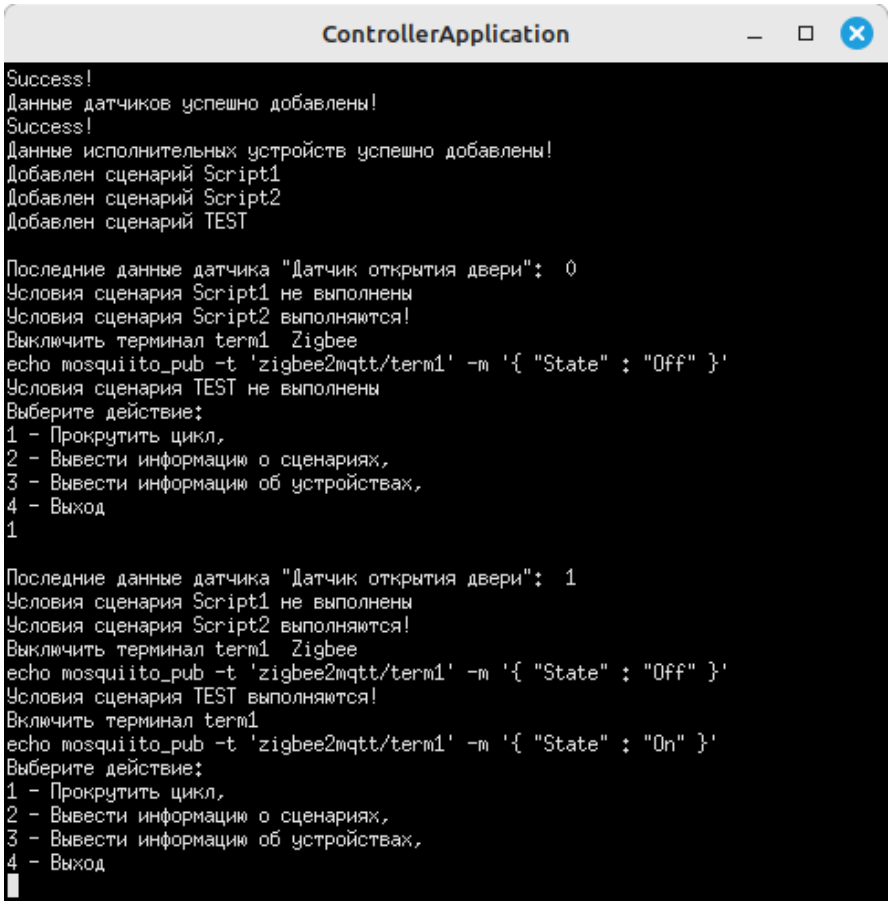


Рисунок 21 – Результаты работы модуля преобразования данных

Как можно увидеть на рисунке, при выполнении условий сценария запускаются некоторые действия, информация о которых выводится в терминал. Запуск этих действий производится путём вызова функций из библиотеки

маршрутизации: в ситуации на рисунке 21 это функции включения и выключения исполнительных устройств по протоколу Zigbee. Эти функции формируют данные, которые нужно отправить на устройства, и вызывают функцию инкапсуляции, которая производит подготовку данных к отправке и отправляет их нужным адресатам в сети. Так, строка «echo mosquiito_pub -t 'zigbee2mqtt/term1’ -m ‘{ “State” : “Off” }’ является командой, отправляемой MQTT-брокеру Mosquitto.

Результаты этого этапа тестирования показали, что разработанная программная модель работает корректно и соответствует требованиям, предъявляемым к подсистеме инкапсуляции и маршрутизации данных, описанным в главе 2 данной работы.

4.5 Выводы по главе

Проведено тестирование разработанной программной модели системы инкапсуляции и маршрутизации данных. Тестирование проводилось при помощи набора тестовых данных.

Проведено несколько этапов тестирования, в том числе:

1. тестирование алгоритма работы модуля декапсуляции данных;
2. тестирование программной модели модуля декапсуляции данных совместно с базой данных;
3. тестирование модуля декапсуляции данных совместно с модулем управления потоками данных;
4. тестирование работы модуля инкапсуляции данных.

На первом этапе тестирования установлено, что предложенный метод декапсуляции данных является эффективным для создания программного модуля.

На втором этапе установлено, что разработанный модуль декапсуляции данных корректно взаимодействует с базой данных, что позволяет обеспечить

хранение декапсулируемых данных, а также их передачу между различными программными модулями.

На следующем этапе было установлено, что модуль декапсуляции данных совместим с модулем управления потоками данных, что позволяет обеспечить передачу данных от датчиков модулю работы со сценариями, тем самым, запуская работу одной из важнейших функций контроллера сети передачи телеметрической информации.

На заключительном этапе тестирования были проверены алгоритмы работы модуля инкапсуляции данных и разработанных функций из состава библиотеки для маршрутизации данных. Установлено, что предложенные методы инкапсуляции и маршрутизации данных эффективны для разрабатываемой программной модели.

Таким образом, разработанная программная модель отвечает требованиям, предъявляемым к подсистеме инкапсуляции и маршрутизации телеметрической информации.

Тем не менее, некоторая часть разработанной модели нуждается в доработке:

1. необходимо доработать декапсуляцию и инкапсуляцию данных, передаваемых при помощи стека протоколов TCP/IP, согласно предложенным методам декапсуляции и инкапсуляции данных;
2. необходимо закончить разработку библиотеки функций для маршрутизации данных.

Выявленные недостатки могут быть исправлены на следующих этапах работы над системой инкапсуляции и маршрутизации данных. Полученные выводы позволяют сформировать перечень основных требований по дальнейшему развитию разрабатываемой системы.

ЗАКЛЮЧЕНИЕ

В процессе реализации проекта поэтапно решались определенные по результатам анализа задания на ВКР задачи. На начальном этапе был проведён аналитический обзор предметной области исследования. Рассмотрены протоколы динамической маршрутизации и статическая маршрутизация, выявлены их сходства и различия, достоинства и недостатки. Проведён сравнительный анализ рассмотренных протоколов и методов маршрутизации данных, в результате которого сделан вывод, что статическая маршрутизации является наиболее подходящим вариантом для применения в разрабатываемой системе передачи телеметрической информации.

По результатам обзора источников по теме инкапсуляции данных в сети установлено, что процессы инкапсуляции и декапсуляции тесно связаны с протоколами, с которыми ведётся работа – следовательно, система инкапсуляции данных для программно-аппаратного комплекса передачи телеметрической информации должна разрабатываться с учётом используемых устройств и поддерживаемых ими протоколов передачи данных.

На втором этапе работы предложен метод работы модуля преобразования данных, включающего в себя модули декапсуляции и инкапсуляции данных, а также вспомогательную библиотеку функций для маршрутизации данных. Предложенный метод декапсуляции и инкапсуляции данных основан на разработанном алгоритме модуля преобразования данных, а предложенный метод маршрутизации данных основан на статической маршрутизации и оригинальных алгоритмах направления данных к нужному узлу сети. Эти методы позволяют обеспечить передачу данных в сети передачи телеметрических данных, а также унифицировать данные, передаваемые при помощи различных протоколов передачи данных.

На третьем этапе работы над ВКР по результатам проведённых экспериментов и исследований выбраны программные и инструментальные средства для разработки программной модели системы синхронизации данных

в сети передачи телеметрической информации. Разработана программная модель, включающая в себя модуль декапсуляции данных, модуль инкапсуляции данных, классы Sensor и Terminal для представления в программе датчиков и исполнительных устройств, и библиотеку функций для маршрутизации данных.

Проведено тестирование разработанной программной модели, включившее в себя четыре этапа: тестирование алгоритма работы модуля декапсуляции данных, тестирование программной модели модуля декапсуляции данных совместно с базой данных, тестирование модуля декапсуляции данных совместно с модулем управления потоками данных и тестирование работы модуля инкапсуляции данных. Каждый из этапов показал соответствие разработанной программной модели требованиям, определённым заданием на ВКР и заданиями на разработку. Полученные результаты тестирования можно использовать при дальнейшей модификации системы инкапсуляции и маршрутизации данных.

Таким образом, все задачи, поставленные на первом этапе выполнения ВКР решены, что позволяет сделать вывод о достижении цели работы.

Предполагаемой научной новизной магистерской диссертации является

1. предложенный метод маршрутизации данных, основанный на статической маршрутизации и оригинальных алгоритмах направления данных к нужному узлу сети, позволяющий унифицировать данные, передаваемые при помощи различных протоколов передачи;

2. предложенный алгоритм декапсуляции и инкапсуляции данных, позволяющий реализовывать инструментальные программные средства преобразрования данных в сети передачи телеметрической информации.

СПИСОК СОКРАЩЕНИЙ

БД – База данных
ОС – Операционная система
ПО – Программное обеспечение
СУБД – Система управления базами данных
ACL – Access Control List
AES – Advanced Encryption Standard
API – Application Programming Interface
BDR – Backup Designated Router
DR – Designated Router
EIGRP – Enhanced Interior Gateway Routing Protocol
IEEE – Institute of Electrical and Electronics Engineers
IoT – Internet of things
IP – Internet Protocol
LAN – Local area network
LSA – Link-state advertisements
LSDB – Link-state database
MAC – Media Access Control
MQTT – Message queuing telemetry transport
OSI – Open Systems Interconnection model
OSPF – Open Shortest Path First
RIP – Routing Information Protocol
RFC – Request for Comments
SIM – Subscriber Identification Module
SQL – Structured Query Language
TCP – Transmission Control Protocol
USB – Universal Serial Bus
VM – Virtual machine
WLAN – Wireless local area network

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. 1.16 Принцип инкапсуляции данных, фрагментации и декапсуляции в моделях TCP/IP и OSI 7. – URL: https://zametkinapolyah.ru/kompyuternye-seti/inkapsulyaciya-dannyx.html (дата обращения: 07.07.2023).

2. Что такое маршрутизация? – URL: https://aws.amazon.com/ru/what-is/routing/ (дата обращения: 07.07.2023).

3. Системы сбора телеметрических данных. – URL: https://esseo.ru/sistemy-sbora-telemetricheskih-dannyh (дата обращения: 06.07.2023).

4. Семенов, Ю. А. Алгоритмы телекоммуникационных сетей. Часть 2. Протоколы и алгоритмы маршрутизации в Internet / Ю. А. Семенов // М.: Бином. Лаборатория знаний, 2016. – 829 с.

5. Руководство Cisco по междоменной многоадресатной маршрутизации. – М.: «Вильямс», 2004. – 320 с.

6. Основы компьютерных сетей. Тема №9. Маршрутизация: статическая и динамическая на примере RIP, OSPF и EIGRP. – URL: https://habr.com/ru/articles/335090/ (дата обращения: 09.07.2023)

7. Vetriselvan V., Patil P. R., Mahendran M. Survey on the RIP, OSPF, EIGRP routing protocols / V. Vetriselvan, P. R. Patil, M. Mahendran //International Journal of Computer Science and Information Technologies. – 2014. – Т. 5. – №. 2. – С. 1058-1065.

8. Black U. D. IP routing protocols: RIP, OSPF, BGP, PNNI, and Cisco routing protocols. / U. D. Black //Prentice Hall Professional. – 2000.

9. Rakheja P. et al. Performance analysis of RIP, OSPF, IGRP and EIGRP routing protocols in a network / P. Rakheja //International Journal of Computer Applications. – 2012. – Т. 48. – №. 18. – С. 6-11.

10. Статическая и динамическая маршрутизация. – URL: https://mikrotik-training.ru/kb/staticheskaya-i-dinamicheskaya-marshrutizatsiya/ (дата обращения: 07.03.2024)

11. Сетевая инфраструктура системы РТЛС. – URL: http://www.rtlsnet.ru/technology/view/3 (дата обращения: 07.03.2024).

12. Простое пособие по сетевой модели OSI для начинающих. – URL: https://selectel.ru/blog/osi-for-beginners/ (дата обращения: 10.07.2023).

13. Модели OSI/ISO и TCP/IP. – URL: https://testengineer.ru/modeli-osi-iso-i-tcp-ip/ (дата обращения: 07.03.2024).

14. Прасолов А.А., Бабаев Н.В., Мошков В.В. Исследование качественных параметров технологии ZigBee в реалиях концепции «Умный дом». / А.А. Прасолов, Н.В. Бабаев, В.В. Мошков. // «Экономика и качество систем связи» – 2019. – №4. – стр. 38 – 45.

15. Stepanov M.S., Poskotin L.S., Shishkin D.V., Timur Turgut, Muzata A.R. The using of ZigBee protocol to organize the "Smart Home" system for aged people. / M.S. Stepanov, L.S. Poskotin, D.V. Shishkin, T. Turgut, A.R. Muzata //T-Comm. – 2021. – vol. 15. – no.10. – pp. 64-70. (in Russian).

16. Андреев Ф.И., Трегубов С.В. Как работает Wi-Fi. / Ф.И. Андреев, С.В. Трегубов // Научно-образовательный журнал для студентов и преподавателей «StudNet». –2020. – №3.

17. Семенов Ю.А. Беспроводные сети ZigBee и IEEE 802.15.4. / Ю.А. Семенов //ГНЦ ИТЭФ. – URL: http://book.itep.ru/4/41/zigbee.htm (дата обращения: 20.10.2023).

18. Формат кадра Wi-Fi: как происходит передача кадра в распределительную систему. – URL: https://zvondozvon.ru/tehnologii/kompyuternye-seti/format-kadra-wi-fi (дата обращения: 20.10.2023).

19. IP. Университет ИТМО. – URL: https://neerc.ifmo.ru/wiki/index.php?title=IP (дата обращения: 21.10.2023).

20. Internet Protocol. IBM. – URL: https://www.ibm.com/docs/ru/aix/7.2?topic=protocols-internet-protocol (дата обращения: 21.10.2023).

21. Allwinner : сайт. – URL: https://www.allwinnertech.com/index.php?c=product&a=index&id=90 (дата обращения: 20.10.2023).

22. Termux : сайт. – URL: https://termux.dev/en/ (дата обращения: 20.10.2023).

23. Home Assistant : сайт. – URL: https://www.home-assistant.io/ (дата обращения: 14.03.2024).

24. Code::Blocks : сайт. – URL: https://www.codeblocks.org/ (дата обращения: 11.04.2024).

25. MySQL : сайт. – URL: https://www.mysql.com/ (дата обращения: 20.04.2024).

26. Linux Mint : сайт. – URL: https://linuxmint.com/ (дата обращения: 11.04.2024).

27. VirtualBox : сайт. – URL: https://www.virtualbox.org/ (дата обращения: 11.04.2024).

28. Zigbee2MQTT : сайт. – URL: https://www.zigbee2mqtt.io/ (дата обращения: 22.02.2024).

29. Eclipse Mosquitto : сайт. – URL: https://mosquitto.org/ (дата обращения: 22.02.2024).

30. Стик SONOFF Zigbee 3.0 USB Dongle Plus-E. – URL: https://www.sonoff.ru/product/stik-sonoff-zigbee-30-usb-dongle-plus-e (дата обращения: 07.03.2024).

31. Датчик открытия дверей и окон Aqara Door and Window Sensor. – URL: https://aqara.ru/product/aqara-door-and-window-sensor/ (дата обращения: 11.04.2024).

32. Протокол MQTT: концептуальное погружение. – URL: https://habr.com/ru/articles/463669/ (дата обращения: 07.03.2024).

ПРИЛОЖЕНИЕ А

Листинги разработанных программ

Sensor.h

```
#ifndef SENSOR_H
#define SENSOR_H

#include <string>

using namespace std;

class Sensor {
public:
	Sensor();
	Sensor(string n, int i, string dt, int ld);

	string getName();
	int getId();
	string getDataType();
	float getLastData();

	void setName(string n);
	void setId(int i);
	void setDataType(string dt);
	void setLastData(float ld);

   float getLastDataFromDB();
	void setLastDataToDB(float lD);

private:
	string name;
	int id;
	string dataType;
	float lastData;

};

#endif // SENSOR_H
```

Sensor.cpp

```
#include "Sensor.h"
#include <string>
#include <iostream>
#include <mysql/mysql.h>

Sensor::Sensor()
{

}
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
Sensor::Sensor(string n, int i, string dt, int ld)
{
	name = n;
	id = i;
	dataType = dt;
	lastData = ld;
}

string Sensor::getName()
{
	return name;
}

int Sensor::getId()
{
	return id;
}

string Sensor::getDataType()
{
	return dataType;
}

float Sensor::getLastData()
{
	return lastData;
}

void Sensor::setName(string n)
{
	name = n;
}
void Sensor::setId(int i)
{
	id = i;
}
void Sensor::setDataType(string dt)
{
	dataType = dt;
}
void Sensor::setLastData(float ld)
{
	lastData = ld;
}

float Sensor::getLastDataFromDB()
{
   MYSQL *conn;
   MYSQL_RES *res;
   MYSQL_ROW row;
```

```
            conn = mysql_init(NULL);

            if(conn == NULL)
            {
               fprintf(stderr, "Error: can't create MySQL-descriptor\n");
            }

            if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
            {
               fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
            }

            mysql_set_character_set(conn, "utf8");

            char query_[200];
            sprintf(query_, "SELECT Last_data FROM Sensor WHERE Name=\'%s\';", this-
>getName().c_str());

            mysql_query(conn, query_);

            if (res = mysql_store_result(conn))
            {
               while(row = mysql_fetch_row(res))
               {
                  for (int i=0 ; i < mysql_num_fields(res); i++)
                  {
                     this->setLastData(atof(row[i]));
                  }
               }
             }
             else
               fprintf(stderr, "%s\n", mysql_error(conn));

            // Закрываем соединение с сервером базы данных
            mysql_close(conn);

            return lastData;
         }

         void Sensor::setLastDataToDB(float lD)
         {
            lastData = lD;

            MYSQL *conn;
            MYSQL_RES *res;
            MYSQL_ROW row;

            conn = mysql_init(NULL);
            if(conn == NULL)
```

```
        {
            fprintf(stderr, "Error: can't create MySQL-descriptor\n");
        }

        if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
        {
            fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
        }

        mysql_set_character_set(conn, "utf8");

        char query_[200];
        sprintf(query_, "UPDATE Sensor SET Last_data = %f WHERE Name=\'%s\';", lD, this-
>getName().c_str());
        cout << query_ << endl;
        mysql_query(conn, query_);

        if (res = mysql_store_result(conn))
        {
            /*while(row = mysql_fetch_row(res))
            {
                for (int i=0 ; i < mysql_num_fields(res); i++)
                {
                    cout << row[i] <<"\n"; //Выводим все что есть в базе через цикл
                }
            }*/
         }
         else
            fprintf(stderr, "%s\n", mysql_error(conn));

        // Закрываем соединение с сервером базы данных
        mysql_close(conn);
    }
```

Terminal.h

```
#ifndef TERMINAL_H
#define TERMINAL_H

#include <string>

using namespace std;

class Terminal {
public:
        Terminal();
        Terminal(string n, int i, string p);

        string getName();
```

```
        int getId();
        string getProtocol();

        void setName(string n);
        void setId(int i);
        void setProtocol(string p);

private:
        string name;
        int id;
        string protocol;
};

#endif // TERMINAL_H
```

Terminal.cpp

```
#include "Terminal.h"
#include <string>

Terminal::Terminal()
{

}

Terminal::Terminal(string n, int i, string p)
{
        name = n;
        id = i;
        protocol = p;
}

string Terminal::getName()
{
        return name;
}

int Terminal::getId()
{
        return id;
}

string Terminal::getProtocol()
{
        return protocol;
}

void Terminal::setName(string n)
{
        name = n;
```

```
}
void Terminal::setId(int i)
{
	id = i;
}

void Terminal::setProtocol(string p)
{
	protocol = p;
}
```

DataProcessingModule.cpp

```
//Функции для выполнения сценария
//Включить
//Формируем команду для передачи данных в зависимости от протокола
//Передаём команду в инкапсулятор
void TerminalOn(Terminal T)
{
	cout << "Включить терминал " << T.getName() << endl;
	if(T.getProtocol() == "Zigbee")
	{
	string command = "mosquiito_pub -t 'zigbee2mqtt/'" + T.getName() + "' -m '{ \"State\"
: \"On\" }'";
	//cout << command << endl;
	encap_Zigbee(command);
	}
	else if(T.getProtocol() == "IP")
	{
	//передаём данные через сокет
	}
}

//Выключить
//Формируем команду для передачи данных в зависимости от протокола
//Передаём команду в инкапсулятор
void TerminalOff(Terminal T)
{
	cout << "Выключить терминал " << T.getName() << "  " << T.getProtocol() << endl;
	if(T.getProtocol() == "Zigbee")
	{
	string command = "mosquiito_pub -t 'zigbee2mqtt/'" + T.getName() + "' -m '{ \"State\"
: \"Off\" }'";
	encap_Zigbee(command);
	}
	else if(T.getProtocol() == "IP")
	{
	//передаём данные через сокет
	}
}
```

```
//Переслать
//Формируем команду для передачи данных в зависимости от протокола
//Передаём команду в инкапсулятор
void SendTo(string param, string protocol, string data)
{
        cout << "Переслать " << data << " " << param << " по " << protocol << endl;
        if(protocol == "Zigbee")
        {
      string command = "mosquiito_pub -t 'zigbee2mqtt/" + param + "' -m '{ " + data + " }'";
      encap_Zigbee(command);
        }
        else if(protocol == "IP")
        {
      //передаём данные через сокет
        }
}
```

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include "stdio.h"
#include "stdlib.h"
#include <clocale>
#include <list>
#include <mysql/mysql.h>
#include "Sensor.h"
#include "Terminal.h"

using namespace std;

//Функция для инициализации датчиков
list<Sensor> sensInit()
{
   MYSQL *conn;
   MYSQL_RES *res;
   MYSQL_ROW row;

   list<Sensor> sensList;

   conn = mysql_init(NULL);
   if(conn == NULL)
   {
      fprintf(stderr, "Error: can't create MySQL-descriptor\n");
   }
```

```
    if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
    {
      fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
    }

    else
    {
      //fprintf(stdout, "Success!\n");
    }

    mysql_set_character_set(conn, "utf8");

    char query_[200];
    sprintf(query_, "SELECT Name FROM Sensor;");

    mysql_query(conn, query_);

    if (res = mysql_store_result(conn))
    {
      while(row = mysql_fetch_row(res))
      {
        for (int i=0 ; i < mysql_num_fields(res); i++)
        {
          //cout << row[i] <<"\n"; //Выводим все что есть в базе через цикл
          sensList.push_back(Sensor(string(row[i]), i + 1, string("float"), 0));
        }
      }
     }
     else
      fprintf(stderr, "%s\n", mysql_error(conn));

    // Закрываем соединение с сервером базы данных
    mysql_close(conn);

    return sensList;
  }

  //Функция для инициализации терминалов
  list<Terminal> termInit()
  {
    list<Terminal> termList;
    MYSQL *conn;
    MYSQL_RES *res;
    MYSQL_ROW row;

    conn = mysql_init(NULL);
    if(conn == NULL)
    {
      fprintf(stderr, "Error: can't create MySQL-descriptor\n");
```

```
    }

    if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
    {
       fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
    }

    else
    {
       //fprintf(stdout, "Success!\n");
    }

    mysql_set_character_set(conn, "utf8");

    char query_[200];
    sprintf(query_, "SELECT Name FROM Terminal;");

    mysql_query(conn, query_);

    if (res = mysql_store_result(conn))
    {
       while(row = mysql_fetch_row(res))
       {
          for (int i=0 ; i < mysql_num_fields(res); i++)
          {
             //cout << row[i] <<"\n"; //Выводим все что есть в базе через цикл
             termList.push_back(Terminal(string(row[i]), i + 1, "Zigbee"));
          }
       }
     }
     else
       fprintf(stderr, "%s\n", mysql_error(conn));

    // Закрываем соединение с сервером базы данных
    mysql_close(conn);

    return termList;
}

//Функция для перевода даты из формата строки в целое число
int dateToInt(string str)
{
    string date;
    int i_date;

    for (int i = 18; i < 28; i++)
    {
       date.push_back(str[i]);
    }
```

```
        i_date = (atoi(date.c_str()) * 100 + atoi(date.c_str() + 5)) * 100 + atoi(date.c_str() + 8);

        return i_date;
    }

    //Функция для перевода времени из формата строки в целое число
    int timeToInt(string str)
    {
        string log_time;
        int i_log_time;

        for (int i = 29; i < 37; i++)
        {
            log_time.push_back(str[i]);
        }
        i_log_time = (atoi(log_time.c_str()) * 100 + atoi(log_time.c_str() + 3)) * 100 + atoi(log_time.c_str() + 6);

        return i_log_time;
    }

    Sensor setData(Sensor sen, string str)
    {
        string substr_;
        int pos = str.find("contact") + 9;
        for (int i = pos; i < str.find(",", pos); i++)
        {
            substr_.push_back(str[i]);
        }
        if(substr_ == "true")
            sen.setLastDataToDB(1.0);
        else if (substr_ == "false")
            sen.setLastDataToDB(0.0);
        return sen;
    }

    //Выбираем, по какому протоколу отправлять, в зависимости от чего вызываем нужную функцию (маршрутизации)
    void switch_encapsulation(string data, int target)
    {
        //Проверяем протокол передачи данных
        //В зависимости от протокола вызываем нужную функцию
        switch (target) {
            case 1:
            {
                encap_Zigbee(data);
                break;
            }
            case 2:
            {
```

```
            encap_IP(data);
            break;
        }
        default:
            break;
    }
}

void encap_Zigbee(string data)
{
    char[200] com;
    sprintf(com, "echo %s", data.c_str());
    cout << com;
    //system(com);
}

void encap_IP(string data)
{
    //передаём данные через сокеты
}

int main()
{
    setlocale(LC_ALL, "rus");

    list<Sensor> sensors = sensInit();
    list<Terminal> terminals = termInit();

    string str, buf;
    string name = "//home//mint//projects//logZi.txt";
    ifstream in;
    int prev_date = 0, prev_time = 0;

    //connect_database();

    //Работаем в бесконечном цикле
    while(1)
    {
        //Открываем лог-файл Zigbee2MQTT
        in.open(name);
        if (!(in.is_open()))
        {
            cout << "Файл с таким именем не найден!\n";
            return 0;
        }

        //Читаем до последней строки
        while(getline(in, buf))
        {
            str = buf;
```

```
            }
            //Закрываем файл
            in.close();

            //Смотрим дату и время
            int i_date = dateToInt(str);
            int i_log_time = timeToInt(str);
            //Если изменились данные
            if((i_date > prev_date) || (i_date == prev_date && i_log_time > prev_time))
            {
               cout << str << endl;
               //Смотрим на название топика, выбираем датчик и вносим информацию о последних данных
               string substr_;
               for (int i = str.find("topic") + 7; i < str.find("""", str.find("topic") + 7); i++)
               {
                  substr_.push_back(str[i]);
               }
               for(Sensor sen : sensors)
               {
                  if(substr_.find(sen.getName()) != std::string::npos)
                  {
                     sen = setData(sen, str);
                     cout << "Последние данные датчика \"" << sen.getName() << "\":  " << sen.getLastData() << endl << endl;
                  }
               }

               //Устанавливаем предыдущие значения
               prev_date = i_date;
               prev_time = i_log_time;
            }
         }
         return 0;
      }
```

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«14»_____06_____2024 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Разработка подсистемы инкапсуляции и маршрутизации данных программно-аппаратного комплекса управления сетью передачи телеметрических данных

090401 Информатика и вычислительная техника

09.04.01.11 «Вычислительные системы и сети»

Руководитель	_____ подпись	19.06.24 дата	профессор, канд. техн. наук должность, ученая степень	О.В. Непомнящий
Выпускник	_____ подпись	19.06.24 дата		П.Д. Неустроев
Рецензент	_____ подпись	19.06.24 дата	доцент, канд. техн. наук должность, ученая степень	К.В. Раевич
Консультант	_____ подпись	19.06.24 дата	генеральный директор ООО «ПК «Дельта»» должность, ученая степень	О.Г. Варыгин
Нормоконтролёр	_____ подпись	19.06.24 дата	профессор, канд. техн. наук должность, ученая степень	О.В. Непомнящий

Красноярск 2024