Санкт–Петербургский государственный университет

Соловьев Дмитрий Николаевич

Выпускная квалификационная работа

Система домашней автоматизации с использованием децентрализованного подхода в хранении и обработке данных.

Уровень образования: бакалавриат

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5190.2021 «Большие данные и распределенная цифровая платформа»

Научный руководитель:
доцент, кафедра компьютерного моделирования и многопроцессорных систем,
к.ф.-м.н., PhD
Корхов Владимир Владиславович

Рецензент:
доцент, кафедра компьютерного моделирования и многопроцессорных систем,
к.т.н.
Дик Геннадий Давидович

Санкт-Петербург
2025 г.

Аннотация

С увеличением количества устройств интернет вещей (ИВ) в системах домашней автоматизации (СДА) актуализируется вопрос топологий сетей, в которых находятся эти устройства, вопрос способов хранения и передачи данных между устройствами. Многим системам, использующим централизованный подход в хранении и обработке информации, свойственна существенная потеря функциональности при выходе из строя центрального узла. В ходе исследования были изучены принципы, методологии и архитектуры, применимые к программно-аппаратным комплексам для построения СДА и позволяющие повысить уровень безопасности данных, отказоустойчивость, а также увеличить скорость передачи информации в рамках системы за счет использования распределенного подхода к хранению и обработке информации, разработан прототип распределенной СДА. Результаты проведенного исследования и предложенная в нем архитектурная модель могут быть использованы не только в системах умных домов, но и в умных городах и на предприятиях.

Abstract

With the increasing number of Internet of Things devices in home automation systems (HAS), the topologies of the networks that house these devices, as well as how data is stored and transferred between devices, are becoming an issue. Many systems that use a centralized approach in storing and processing information are characterized by a significant loss of functionality when a central node fails. In the course of the research the principles, methodologies and architectures applicable to hardware-software complexes for the construction of HAS were studied, which allow to increase the level of data security, fault tolerance and increase the speed of information transfer within the system due to the use of distributed approach to information storage and processing, and a prototype of distributed HAS was developed. The results of the conducted research and the architectural model proposed in it can be used not only in smart home systems, but also in smart cities and enterprises.

Содержание

1 Введение

1.1 Актуальность темы

Актуальность и практический аспект анализа систем и выбора оптимальной для построения системы домашней автоматизации:

1. Количество устройств с возможностью интеграции в систему умного дома среди потребителей быстро возрастает год к году. Особенности построения актуальных систем накладывают ограничения на масштабируемость системы, ее надежность и безопасность.
2. В системах домашней автоматизации с появлением новых типов устройств возникают проблемы интеграции, связанные со скоростью реакции на события и с отсутствием приоритезации сообщений.

1.2 Объект исследования

Объектом научно-исследовательской работы являются системы домашней автоматизации, системы хранения, передачи и обработки информации.

1.3 Предмет исследования

Предметом научно-исследовательской работы является оптимизация обработки и передачи данных между устройствами ИВ в системе домашней автоматизации.

1.4 Постановка задачи

В рамках данной работы ставится цель провести комплексное исследование существующих СДА на рынке, проанализировать применяемые в них

протоколы и выявить основные недостатки. На основании этого выбрать оптимальный протокол и разработать прототип системы, отвечающий сформулированным требованиям и решающий обозначенный список проблем.

Для достижения этой цели были сформулированы начальные условия (требования):

1. Сеть устройств в пределах одной зоны должна строиться по одноуровневому принципу, без централизованных узлов управления.
2. Взаимодействие устройств внутри зоны осуществляется напрямую, без посредников (ЦУД).
3. Для связи устройств из разных зон используются шлюзы-ретрансляторы, объединённые в единую распределённую сеть.

Практическая значимость предлагаемого подхода:

- Уменьшение задержек, расширение сфер применения: в предлагаемой системе уменьшается количество узлов между устройством-инициатором и устройством-актуатором (при нахождении в одной сети–до нуля), за счет чего увеличивается скорость выполнения команд и повышается общая отзывчивость системы, что расширяет сферы применения устройств ИВ для построения СДА, ранее нефункциональных ввиду относительно больших задержек.
- Повышение комфорта использования: данные о состоянии устройств, как и статистика, собираемая устройствами, хранится и обрабатывается на самих устройствах, что позволяет обеспечить быстрое восстановление состояния системы после внештатных ситуаций (например, отключения питания) без длительной синхронизации.
- Упрощение настройки системы: использование планировщика, автоматически распределяющего сценарии по устройствам, не требует в ручной конфигурации каждого узла.
- Повышение отказоустойчивости: отказ сценария в одной локации возможен лишь при неисправности одного из двух устройств (инициатора и актуатора) и не зависит от иных устройств в сети. Объедине-

ние локаций, а также дистанционный мониторинг и управление системой осуществляется посредством распределенной системы шлюзов-ретрансляторов, что избавляет от единой точки отказа системы (точки входа).

- Повышение безопасности: распределенный подход для хранения сценариев и управления устройствами минимизирует получение злоумышленниками доступа ко всем устройствам. Система шлюзов-ретрансляторов избавляет от единой точки отказа системы (точки входа).

2 Глава 1. Топологии сетей и протоколы беспроводных сетей

2.1 Топологии беспроводных сетей.

Все перечисленные беспроводные сети работают в одном или нескольких вариантах топологии. На рис. 2.1 приведены топологии беспроводных сетей различных конфигураций.

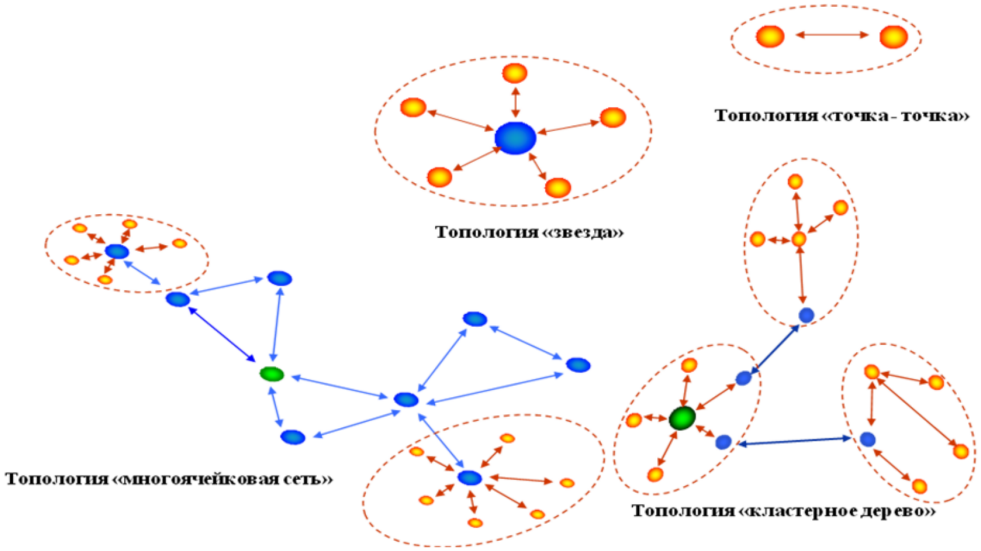


Рис. 2.1: Топологии беспроводных сетей [1]

2.1.1 Топология точка-точка

Самый простой вариант организации сети из двух устройств. Как правило, узлы этой сети являются равноправными, то есть сеть одноранговая. Эта топология характерна для Bluetooth, BLE, RFID, UWB, Wi-Fi Direct, RuBee, PDC и прочих.

2.1.2 Топология «звезда»

В этой конфигурации все устройства подключаются к центральному узлу (хабу), который управляет коммуникацией между ними. Данную топологию используют протоколы: Wi-Fi Zigbee, Zigbee Pro, Z-Wave, Bluetooth, Wi-Fi HaLow, Insteon, UWB, IDEN, CDMAOne, WIMAX, GSM, GPRS, UTMS.

2.1.3 Ячеистая топология

Ячеистая топология — базовая полносвязная топология компьютерных сетей и сетей связи, в которой каждая рабочая станция сети соединяется со всеми другими рабочими станциями этой же сети. Характеризуется высокой отказоустойчивостью, сложностью настройки и избыточным расходом кабеля в проводных сетях. Каждый узел имеет несколько возможных путей соединения с другими узлами, за счет этого такая топология очень устойчива, так как исчезновение одного из каналов не приводит к потере соединения между двумя компьютерами. Эта топология допускает соединение большого количества узлов и характерна, как правило, для крупных сетей, она строится из полносвязной путем удаления некоторых возможных связей. Топология применима для сетей с использованием протоколов UWB, Zigbee, Zigbee Pro, Thread, BLE Mesh, Z-Wave, Wi-Fi HaLow, LoRaWAN, Wirepas Mesh, IDEN, CDMAOne, WIMAX, GSM, GPRS, UMTS.

2.1.4 Топология «кластерное дерево»

Топология «кластерное дерево» образуется как комбинаций вышеупомянутых топологий. Основание дерева вычислительной сети располагается в точке (корень), в которой собираются коммуникационные линии информации (ветви дерева). Вычислительные сети с древовидной структурой строятся там, где невозможно непосредственное применение базовых сетевых структур в чистом виде.

2.2 Протоколы беспроводных сетей

В данный момент большинство производителей ИВ используют централизованный подход для построения СДА [2]. В качестве центра умного

дома (ЦУД) выступает устройство (хаб), объединяющее конечные устройства в единую сеть. Сценарии взаимодействия устройств могут быть локальными (сценарии хранятся в ЦУД) или облачными. Вариант с локальным расположением сценариев более предпочтительный, так как повышается отзывчивость системы и ее отказоустойчивость. Отдельную сложность представляет объединение устройств из разных экосистем: различные комбинации связей «устройство», «ЦУД», «облако».

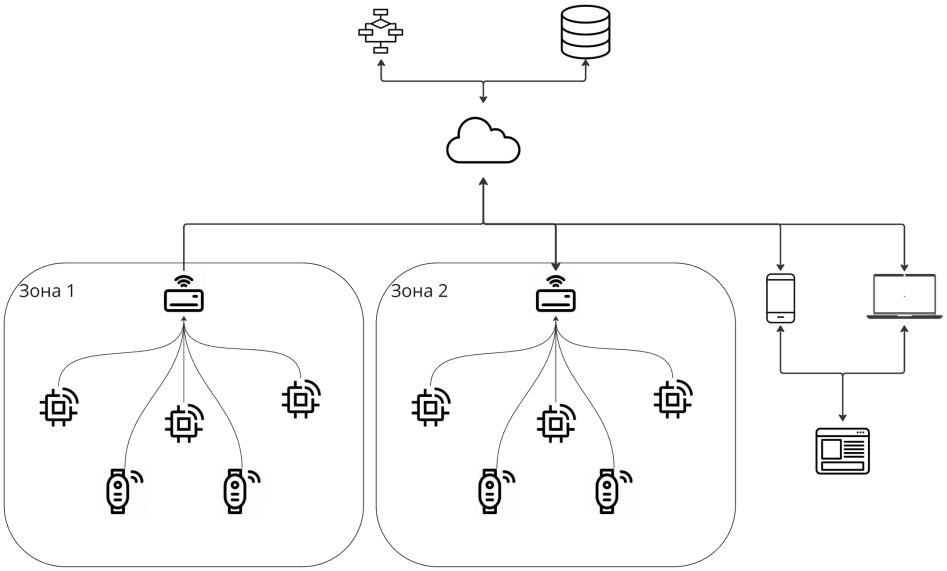


Рис. 2.2: Пример СДА с облачной обработкой событий

Большинство имеющихся на данный момент решений опираются на топологию «дерево», где в качестве корня рассматривается ЦУД или облако, что проиллюстрированно на рис. 2.2. При этом структура может подразумевать наличие единственных ветвей из корня, например «облако-ЦУД-устройства» [4]. Тенденции 2023 и 2024 годов по переносу обработки событий в ЦУД, который ранее использовался лишь для маршрутизации сообщений от устройств к облаку, позволили увеличить надежность и уменьшить задержки в системе, но не исключили имеющиеся проблемы полностью.

Стоит различать аппаратные ограничения, не позволяющие изменить топологию системы, связанные с особенностью работы протокола, например, топологию Bluetooth, позволяющую организовывать только связи типа точка-точка или точка - множество устройств, и архитектуру системы УД, подразумевающую централизованное выполнение сценария в ЦУД либо в облаке.

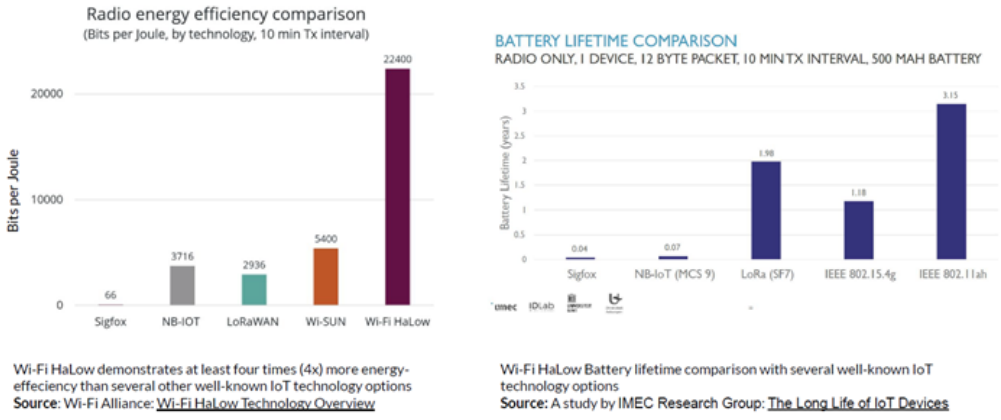
2.2.1 Wi-Fi и Wi-Fi HaLow

Протокол Wi-Fi относится к семейству стандартов IEEE 802.11, реализующих беспроводную передачу данных в различных частотных диапазонах:

- 2.4 ГГц–стандарты 802.11b/g/n;
- 5 ГГц–802.11a/ac;
- 6 ГГц–802.11ax (Wi-Fi 6E).

Эти протоколы поддерживают высокую пропускную способность и широко распространены благодаря простой интеграции в клиентские устройства и устоявшейся практике применения. Однако они изначально не предназначены для энергоэффективных решений, что ограничивает их применение в системах домашней автоматизации, особенно если подразумевается использование батарейного питания. Также различные виды конфигурации сети могут оказывать значительное влияние на время автономной работы IoT-устройства. [3]

IEEE 802.11ah (Wi-Fi HaLow) –расширение Wi-Fi в суб-ГГц диапазоне для IoT и промышленных задач. Обеспечивает дальность до километра, поддержку до тысяч устройств и низкое энергопотребление, что проиллюстрировано на рисунке 2.3, благодаря механизмам Target Wake Time и RAW. Сохраняет совместимость на уровне MAC с Wi-Fi, унаследовав безопасность и QoS (Quality of Service).



Wi-Fi HaLow demonstrates at least four times (4x) more energy-effeciency than several other well-known IoT technology options
Source: Wi-Fi Alliance: Wi-Fi HaLow Technology Overview

Wi-Fi HaLow Battery lifetime comparison with several well-known IoT technology options
Source: A study by IMEC Research Group: The Long Life of IoT Devices

Рис. 2.3: Сравнение энергоэффективности протоколов [16]

Основные элементы архитектуры и топологии

- **Диапазон частот:** 863–928 МГц (региональные поддиапазоны).
- **Узлы:**
  - STA (датчики/актуаторы) с поддержкой энергосбережения.
  - AP (Access Point)–единственный координатор сети, задаваемый при развертывании сети администратором.
  - Relay AP (опционально)–специальные узлы (ретрансляторы)
- **Топология:** поддерживаются несколько вариантов: звездная и древовидная с ретрансляцией (ограниченное дерево без полного L2-mesh) для расширения до удалённых STA.
  - Звезда.
  - Дерево: дерево ограниченной глубины, без полного L2-mesh для расширения до удалённых STA

Преимущества и недостатки: В качестве преимуществ IEEE 802.11ah можно выделить большую дальность связи (до 3 км вне помещений при использовании ретранслятора Relay AP), высокую плотность подключений (возможность обслуживания сотен устройств одной точкой доступа), а также низкое энергопотребление при малой активности. Вместе с тем, стандарту присущи: низкая пропускная способность по сравнению с Wi-Fi в диапазонах 2,4/5/6 ГГц, более высокая сложность и стоимость модулей по сравнению с устройствами на базе 802.15.4, из-за чего данный протокол не получил широкого распространения среди потребительских устройств, заняв только небольшую нишу промышленных IoT-устройств; а также сложности лицензирования различных диапазонов частот для разных стран. При отказе единственного AP вся BSS теряет синхронизацию и связь STA; стандарт не содержит механизмов внутреннего резервирования или автоматического фейловера.

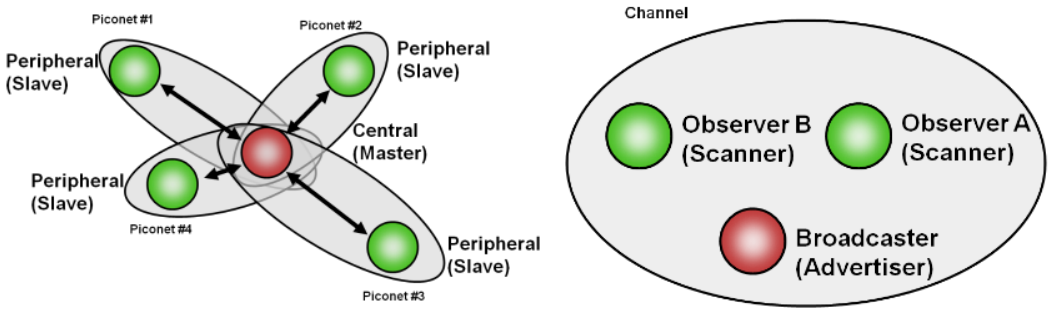
2.2.2 Bluetooth Low Energy (BLE) и Bluetooth Mesh

Bluetooth Low Energy (BLE)–низкоэнергетичная радиотехнология, оптимизированная для передачи небольших объёмов данных с минимальным расходом батареи и поддерживающая энергосберегающие режимы. BLE определяет четыре базовые роли устройств: Central, Peripheral, Broadcaster и Observer, что позволяет строить радиосеть в топологии «Звезда» с центральным узлом (Central) и периферийными устройствами (Peripheral). Частотный диапазон BLE включает 40 каналов в диапазоне 2,4–2,4835 ГГц: 37 каналов для передачи данных и 3 канала для рекламы.

Роли устройств

- **Peripheral**–Энергоэффективное устройство, периодически транслирующее (advertise) данные и ожидающее подключения от центрального узла 2.4a. Характерные примеры: пульсометры, датчики температуры, трекеры активности.
- **Central**–Стационарное устройство, выполняющее сканирование эфирных пакетов и устанавливающее соединения с одним или несколькими периферийными узлами. Часто это смартфоны, планшеты или концентраторы IoT-сетей, которым нужны данные от множества сенсоров.
- **Broadcaster**–Устройство с односторонней рекламой: без установления соединения периодически рассылает рекламные пакеты с полезной информацией (beacons) 2.4b. Используется для вещания маячков близости, рекламных кампаний, системах навигации внутри помещений (Indoor positioning system).
- **Observer**–Стационарный приёмник, которое пассивно сканирует эфир, собирая (несвязанную) информацию из рекламных пакетов Broadcaster’ов и Peripheral’ов. Применяется для мониторинга окружающей среды, подсчёта посетителей, сбора статистики о маячках.

Применимость в системах автоматизации и IoT. BLE широко используется в умных домах, торговых и продуктовых центрах, медицине и иных



(a) Peripheral/Central **(b) Broadcaster/Observer**

Рис. 2.4: Профили устройств BLE[18]

сферах ИВ благодаря низкому энергопотреблению (десятки микроампер в спящем режиме) и широкой распространенности среди потребительских устройств.

BLE Mesh. Т.к. BLE Mesh лишь расширение стандарта, использующее особым образом режим рекламы для лавинообразной передачи сообщений, по техническим характеристикам он сильно похож на BLE. Для поддержания совместимости была применена лавинообразная рассылка (flooding [8]), которая с большим количеством устройств может значительно увеличить территорию покрытия, но, с другой стороны, может привести к избыточной загруженности каналов, что при превышении порогового количества устройств может привести к потерям пакетов.

2.2.3 Zigbee и Zigbee Pro

Стандарт ZigBee основан на IEEE 802.15.4 и работает в нелицензируемых ISM-диапазонах [10, 9]: 2.4ГГц (глобально доступен и используется во всем мире), 915МГц (США, Канада, Австралия и Израиль), 868МГц (ЕС, Великобритания, Турция)

Области применения включают умные дома и городскую инфраструктуру, где используются системы управления освещением, датчики движения, решения для безопасности и «умные» счётчики в составе городских систем управления; промышленную автоматизацию с круглосуточным мониторингом состояния оборудования, предиктивным обслуживанием, сбором данных с вибрационных и температурных датчиков, дистанционным управлением приводами и оптимизацией технологических процессов; медицинский мониторинг для удалённого сбора жизненных показателей пациентов и контроля

заряда портативных медицинских устройств.

Основные элементы архитектуры и топологии

- **Узлы:**
  - Координатор (ZC) единственный узел, создаёт PAN: сканирует каналы, выбирает свободный, задаёт PAN_ID, Extended PAN_ID, ключи безопасности, хранит таблицы маршрутизации и аутентификации. Работает также как маршрутизатор.
  - Маршрутизатор (ZR) станционарное, постоянно активеное устройство, расширяющее покрытие, ретранслирующее пакеты, обеспечивающее альтернативные пути.
  - Конечное устройство (ZED) автономное устройство, без функции ретрансляции, общающееся только с родителем (координатором или маршрутизатором), большую часть времени находящееся в режиме сна.
- **Топология:** ZigBee поддерживает три базовые топологии:
  - Звёздная топология (Star) Все узлы подключены непосредственно к координатору. Простая настройка, но ограничена радиусом и зависит от координатора.
  - Деревовидная топология (Cluster-Tree) Иерархия: координатор → маршрутизаторы уровней → конечные устройства. Адресация упрощена tree-адресацией.
  - Ячеистая топология (Mesh-network) Любой маршрутизатор может ретранслировать пакет между любыми двумя узлами. Поддерживает самовосстановление при отказах и динамическое подключение новых устройств.

Координатор задаётся программно администратором, но при запуске сети он автоматически:

- сканирует каналы ISM;
- выбирает наименее загруженный канал;

- присваивает PAN_ID;
- задаёт параметры: `NWK_MAX_CHILDREN`, `NWK_MAX_DEPTH`, `MAX_ROUTERS`.

В сетях IEEE 802.15.4 с beacon-режимом координатор через равные интервалы от 15 мс до 4 минут рассылает фрейм-маяк, который объявляет начало очередного суперфрейма и указывает доступные слоты доступа[10]; обычно именно координатор является главным источником этих маяков, хотя в древовидных (cluster-tree) конфигурациях роль ретрансляторов могут брать на себя маршрутизаторы, тогда как в обычном mesh-режиме они маяки не посылают. При этом автоматическое резервирование источников маяков не предусмотрено, поэтому отказ узла-маяка приводит к потере синхронизации всей сети и её приходится вручную восстанавливать.

Обнаружение маршрута Процесс начинается с того, что узел-инициатор рассылает по сети широковещательный запрос на установление маршрута (RREQ), при этом указывается определённая метрика, например, количество переходов или качество канала. Узел назначения, получив несколько RREQ от разных маршрутов, выбирает из них наиболее подходящий–согласно метрике–и отправляет одноадресный ответ (RREP) обратно к источнику, используя обратный путь, по которому был получен лучший запрос.

Передача и ретрансляция После получения ответа (RREP) начинается передача данных. Информация передаётся от источника к получателю через промежуточные узлы. Каждый промежуточный узел проверяет свою таблицу маршрутизации, чтобы определить, куда направить пакет дальше, и ретранслирует пакет следующему узлу по цепочке. Если ожидаемое подтверждение доставки (ACK) не поступает, инициируется сообщение об ошибке маршрута (RERR). В таком случае начинается повторное построение маршрута — запускается новый процесс рассылки RREQ.

Шаблоны устройств (Application Profiles) На уровне приложения Zigbee определяет профили устройств, что можно наблюдать на рисунке 2.5:

- **Публичные профили:** Home Automation, Commercial Building Automation и др.: лампы, выключатели, датчики, термостаты, розетки.

- **Частные профили:** Производители могут создавать собственные профили, но контроллер должен поддерживать их для взаимодействия.

Каждый профиль включает: `Profile ID`, `Device ID`, `Cluster IDs` и назначение энптойтов (адрес и адресат). Контроллер распознаёт и управляет устройством только при совпадении этих идентификаторов.

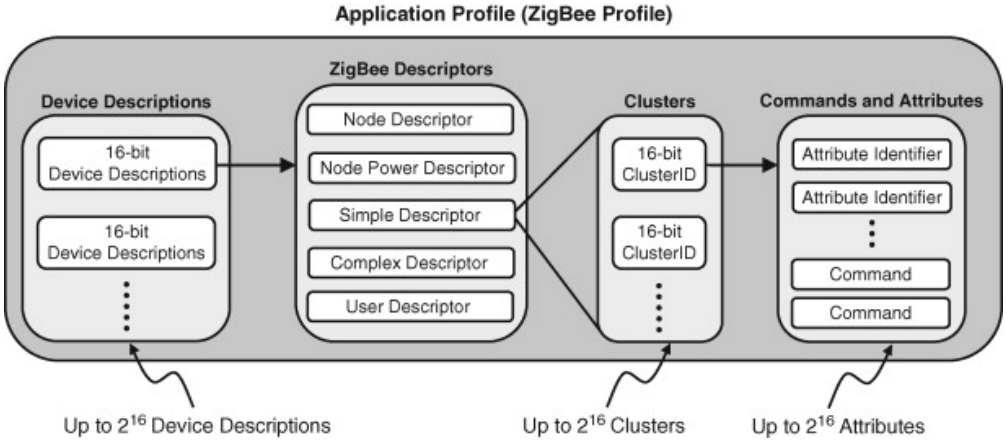


Рис. 2.5: Профили устройств Zigbee [19]

Влияние отказов

- **Координатор:** при его отказе сеть не функционирует, требует перезапуска с новым координатором [11].

- **Маршрутизатор:** потеря альтернативных путей, временная деградация доступности узлов, последующее самовосстановление [11].

- **Конечное устройство:** затрагивается только функционал этого узла, остальная сеть продолжает работу.

Zigbee Pro. Zigbee Pro расширенную реализацию базового протокола, оптимизированную для высоконагруженных и масштабируемых сетей [12]. В отличие от классического Zigbee, Pro-версия обеспечивает поддержку сетей размером до нескольких тысяч узлов, более гибкую и самовосстанавливающуюся маршрутизацию, улучшенную управляющую функциональность

(Many-to-One и Source Routing) и усиленные механизмы безопасности с отдельными ключами для транспортного и сетевого уровней.

В случае отказа координатора или маршрутизатора, сеть на базе Zigbee Pro демонстрирует лучшие возможности по сохранению связи и быстрому восстановлению работоспособности [11] благодаря:

- **Самовосстановлению маршрутов:** оставшиеся маршрутизаторы автоматически перестраивают таблицы маршрутов без участия центрального узла.
- **Source Routing и Many-to-One маршруты:** устройства могут кэшировать рабочие пути и переключаться на альтернативные узлы при недоступности основного, что снижает время простоя.
- **Гибкая архитектура Trust Center:** в Pro-режиме можно реализовать избыточные или внешние Trust Center (через облачные серверы или резервные координаторы), что позволяет продолжать авторизацию и обновление ключей даже при выходе из строя основного координатора.

Таким образом, выбор Zigbee Pro вместо базовой версии Zigbee особенно оправдан в критичных системах, где недопустима потеря управляемости и долговременная деградация сети при отказах ключевых узлов.

2.2.4 Сравнение представленных протоколов

На рис. 2.6 представлен сравнительный график различных протоколов беспроводных сетей. Была составлена таблица 2.1.

Wi-Fi Подходит, если устройства питаются от сети и требуют периодической двусторонней связи (настройка, диагностика), но энергопотребление высоко, а при отказе единственного роутера подключенные к нему устройства теряют соединение. Отсутствие готовых профилей усложняет быструю интеграцию простых сенсоров и актюаторов в экосистему без дополнительной логики на каждом устройстве.

Wi-Fi HaLow (802.11ah) Дает радиус работы в несколько сотен метров и низкое энергопотребление благодаря планируемым «пробуждениям», но требует специализированных модулей и ручной реализации логики таймингов

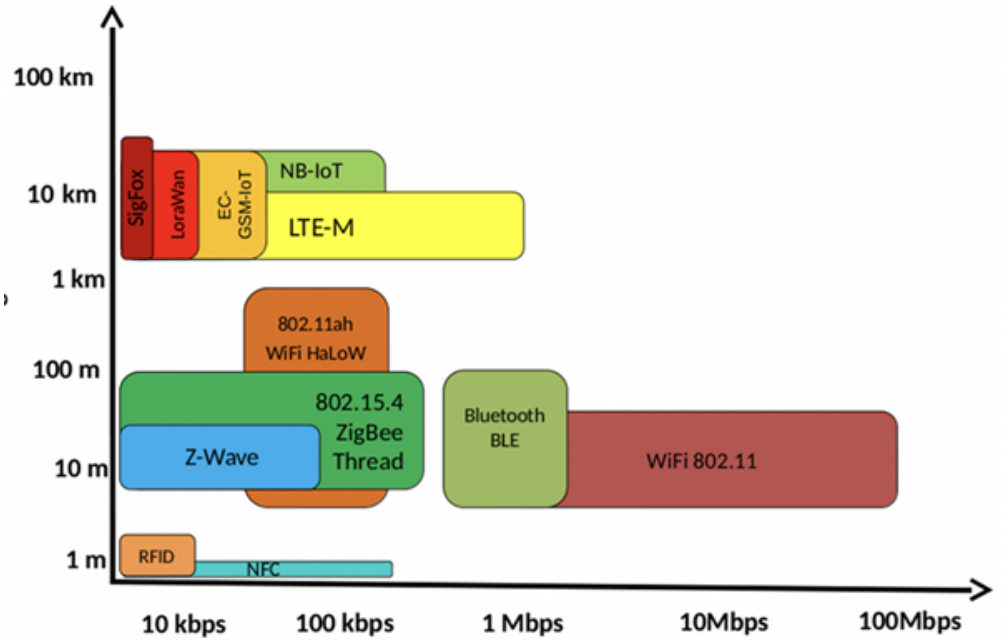


Рис. 2.6: Сравнение беспроводных протоколов передачи данных по дальности и скорости передачи. [15]

и ретрансляции. Полезен для удалённых уличных датчиков, но не для систем автоматизации помещений.

BLE Идеален для маленьких беспроводных сенсоров: потребляет микроватты в спящем режиме, а «из коробки» доступно множество стандартизованных GATT-профилей (кнопки, переключатели, датчики температуры), что ускоряет выпуск простых устройств. Единственная точка сбора (смартфон или хаб) — SPOF, что незначительно при подключении к смартфону или ПК, но становится уязвимым местом при построении системы с большим количеством устройств.

BLE Mesh Устраняет SPOF за счёт децентрализованного flooding-ретранслятора, повышая надёжность покрытия, но сложность прошивки растёт: нужно описывать модели сообщений и управлять подписками, кроме того, батарейные устройства-ретрансляторы разряжаются быстрее. Bluetooth Mesh не использует центрального координатора и строит сеть поверх классического BLE через управляемое «затопление» (managed flooding), что обеспечивает высокую отказоустойчивость и самовосстановление при выходе узлов из строя. В то же время это ведёт к повышенной сложности разработки (необходимость профилирования, управления ключами, контроля времени

жизни сообщений и т. д.) и к компромиссам по энергоэффективности, решаемым с помощью ролей «низкоэнергетический узел»/«друг» (Low Power Node / Friend)

Zigbee Низкое энергопотребление и простая интеграция типовых устройств через ZCL-профили (лампы, выключатели, датчики) делают его популярным в массовых установках. Однако жёстко централизованный PAN-Coordinator без автоматического резервирования превращает отключение хаба в полный отказ сети.

Zigbee Pro Добавляет самовосстановление mesh-маршрутов и резервирование Trust Center, что существенно повышает отказоустойчивость, сохраняя энергоэффективность и привычную модель ZCL-профилей. Разработка сложных нетиповых устройств усложняется из-за множества дополнительных опций маршрутизации и безопасности.

Thread Строит настоящий IPv6-mesh с динамическим лидером, без единой точки отказа, и самовосстанавливается через RPL. Отличается отличной надёжностью и низким потреблением сенсоров, но не предлагает «готовых» шаблонов устройств — для ламп и датчиков придётся проектировать обмен и структуру данных самостоятельно. Имеет высокий порог входа ввиду малой распространенности устройств для разработки (Dev-Kit).

Имя	**Пропускная способность**	**Дальность**	**Протокол маршрутизации**
Wi-Fi	до 23 Гбит/с	до 70 м в помещении и до 140 м на открытом местности	IEEE 802.11s (HWMP)
Wi-Fi HaLow	150 Кбит/с – 347 Мбит/с	до 200 м в помещении и до 1 км на открытом местности	IEEE 802.11s (HWMP)
BLE Mesh	до 125 Кбит/с	до 100 м в помещении и до 300 м на открытой местности	Flooding
BLE Mesh	до 2 Мбит/с	до 400 м	managed flooding
Zigbee	до 250 Кбит/с	10-100 м в помещении и 100–300 м на открытом местности	AODV (изменяемо)
Zigbee Pro	до 250 Кбит/с	10–100 м в помещении и 100–300 м на открытом местности	AODV (изменяемо)
Thread	до 250 Кбит/с	10–100 м в помещении и 100–300 м на открытом местности	MLE + distance-vector (RIPng)

Таблица 2.1: Сравнительная таблица параметров беспроводных сетей

3 Глава 2. Общее описание системы

3.1 Проблемы, решаемые предлагаемой системой

СДА, осуществляющие обработку сценариев при помощи облачных сервисов либо с помощью центра умного дома, обладая несомненными преимуществами, такими как простота настройки и реализации, также имеют ряд недостатков:

1. **Высокая задержка:** Событие, поступающее от сенсора, должно пройти длинный путь от устройства-инициатора к центру умного дома, пройти обработку и быть передано в виде команды устройству-актуатору. Использование шаблонов устройств позволяет упростить взаимодействие сенсоров и актуаторов в системе, так, например, системы, использующие ZigBee, могут устанавливать соединение напрямую между некоторыми устройствами используя технологию binding [9]. Такой подход может существенно снизить нагрузку на координатор сети и ретрансляторы, но не может решить проблему сложных взаимодействий. Также данный метод предполагает использование однотипных шаблонов, из-за чего устройства разных производителей зачастую всё-таки будут взаимодействовать через центр умного дома.

2. **Отказоустойчивость:** Использование центра умного дома либо облачные инфраструктуры для обработки сценариев потенциально снижает надёжность системы: отказ централизованного обработчика парализует работу СДА. Использование кадров маяка для синхронизации вносит ограничение на отказоустойчивость системы: выход из строя координатора, задающего синхронизирующий фрейм также парализует СДА.

3.2 Предлагаемые нововведения

Предлагаемый набор нововведений, позволяющих решить список потенциальных проблем:

1. Перенос обработки логики событий из центра умного дома (или облачного сервиса) в устройства-актуаторы.

2. Для взаимодействия устройств предлагается использование паттерна «наблюдатель» (Subject-Observer).

3. Для объединения устройств из разных локаций в единую сеть предлагается использование распределённого отказоустойчивого кластера, выполняющего функции моста (ретрансляция сообщений внутри сети) и координатора сети (настройки системы, добавление и удаление устройств) и позволяющего удалённо управлять и отслеживать состояние умного дома. Предполагается использование минимум одного узла кластера в каждой локации. На рис. 3.1 представлена архитектура предлагаемой гибридной системы.

4. С целью повышения отказоустойчивости предлагается использование CSMA [13] для предотвращения коллизии сообщений и отказа от фрейма синхронизации.

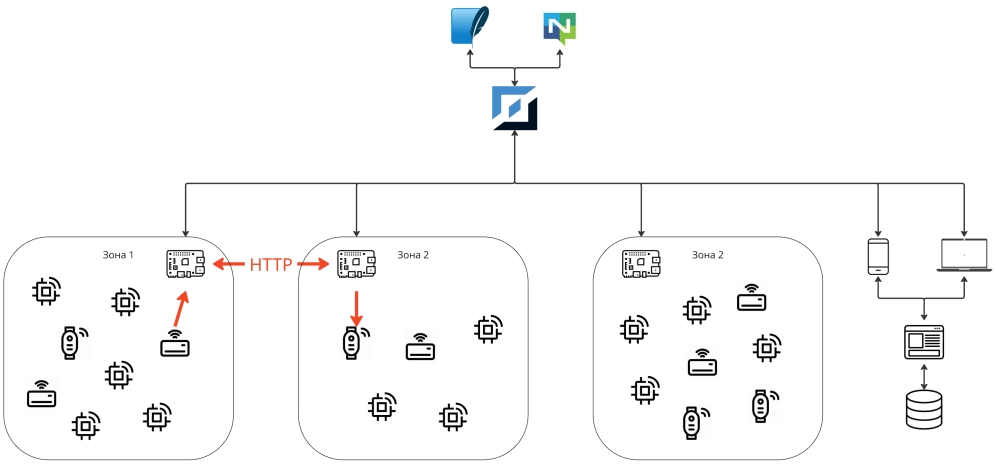


Рис. 3.1: Пример СДА с распределенной структурой

3.3 Обоснование нововведений

Данный подход позволит снизить задержки, так как вдвое сокращается количество передаваемых сообщений в режиме one-to-one и в n раз в случае передачи сообщений one-to-many. Также снижение задержек происходит благодаря отсутствию поочередной передачи команд каждому устройству, если событие требует одновременного взаимодействия нескольких устройств, например, по нажатию выключателя включение группы ламп. Данное решение также повышает отказоустойчивость сети из-за отсутствия промежуточных узлов: надёжность системы зависит только от надёжности устройства-сенсора и устройства-актуатора, а в случае использования межзональных сценариев также от узлов кластера из этих локаций.

Использование CSMA хоть и приводит к чуть большему потреблению стационарных устройств, но позволяет отказаться от генератора синхронизирующих фреймов, которым обычно выступает координатор сети. В таком случае выход из строя координатора либо его отключение ограничится только невозможностью удалять, добавлять и переконфигурировать сеть, но не повлияет на работоспособность сети.

Данное решение кроме повышения надежности системы также уменьшает задержку передачи информации, повышает отказоустойчивость системы и ее безопасность [5, 6].

4 Глава 3. Конечные устройства.

4.1 Аппаратная реализация конечных устройств.

4.1.1 Обоснование выбора аппаратной платформы

В качестве аппаратной платформы конечных устройств системы домашней автоматизации выбран микроконтроллер STM32F411 семейства STM32 (STMicroelectronics). Данное семейство отличается высоким уровнем интеграции, широкой линейкой моделей, а также возможностью масштабирования — от энергоэффективных до высокопроизводительных микроконтроллеров. В частности, совместимость пинов питания и некоторых интерфейсов (UART, SPI, I²C, ADC и др.) между различными линейками (STM32F0, F1, F3, F4 и др.) позволяет замену микросхем без необходимости внесения значительных изменений в схемотехническое решение.

Выбор серии STM32F4 обусловлен необходимостью выполнения интерпретируемого кода на конечных устройствах. В частности, в рамках данного проекта планировался запуск встроенного JavaScript-интерпретатора, который требует наличия значительного объёма оперативной памяти (не менее 64 КБ ОЗУ) и производительного ядра. STM32F411 с ядром Cortex-M4 и 128/256 КБ ОЗУ полностью удовлетворяет данным требованиям, при этом оставаясь относительно доступным и распространённым решением.

Дополнительным фактором при выборе аппаратной платформы послужила возможность разработки в среде Arduino IDE. В отличие от STM32WB, обеспечивающего полноценную поддержку беспроводных протоколов Zigbee и Thread, STM32F411 может быть полноценно использован в Arduino-среде через STM32duino core. Это особенно важно при ограниченном времени и требовании быстрой интеграции: в рамках проекта приоритетом была высокая скорость разработки и наличие широкого сообщества, предоставляющего библиотеки, примеры и поддержку. Разработка че-

рез STM32CubeIDE или использование HAL/LL библиотек требует существенно более высокого порога вхождения, что снижает доступность технологии для разработчиков с опытом, ограниченным Arduino-экосистемой.

4.1.2 Обоснование выбора радиомодуля

В качестве радиомодуля выбран микросхема SI4432 производства Silicon Labs. Данный модуль обеспечивает надёжную и энергоэффективную радиосвязь в диапазоне 433/868/915 МГц с высокой чувствительностью приёмника (до –121 дБм) и выходной мощностью до +20 дБм, что позволяет обеспечить устойчивую связь на больших расстояниях и в условиях радиопомех. Немаловажным преимуществом является буфер сообщений объёмом 64Б, что является наибольшим показателем, среди радиомодулей-претендентов. Также при выборе ключевым аспектом выбора радиомодуля была его интеграция в RadioLib–универсальную библиотеку беспроводной связи для встраиваемых устройств.

Хотя Zigbee (в частности, Zigbee Pro) признан промышленным стандартом для реализации распределённых систем автоматизации, включая системы «умного дома», его внедрение в рамках Arduino IDE представляется затруднительным. В частности, несмотря на наличие Zigbee-совместимых микроконтроллеров семейства STM32WB, полноценная работа с данным протоколом требует использования STM32CubeWB SDK и разработки в STM32CubeIDE, что влечёт за собой необходимость освоения принципиально иной среды и подхода разработки. Поддержка Zigbee в рамках STM32duino отсутствует, а готовых библиотек для Arduino IDE расширяющих функционал STM32duino нет.

Использование радиомодуля SI4432 позволяет реализовать собственный стек передачи данных, гибко настраиваемый под конкретные требования проекта. Несмотря на необходимость разработки протокольного уровня связи, данный подход обеспечивает более высокую скорость разработки, по сравнению с использованием STM32CubeWB SDK.

4.1.3 Проектирование конечного устойства.

В среде проектирования электронных схем EasyEDA была разработана принципиальная схема устройства, представленная на рисунке 4.1. В той

же среде выполнена трассировка печатной платы. Изготовление платы осуществлено по лазерно-утюжной технологии (ЛУТ), обеспечивающей высокую точность и преемлемое качество получаемого рисунка проводников.

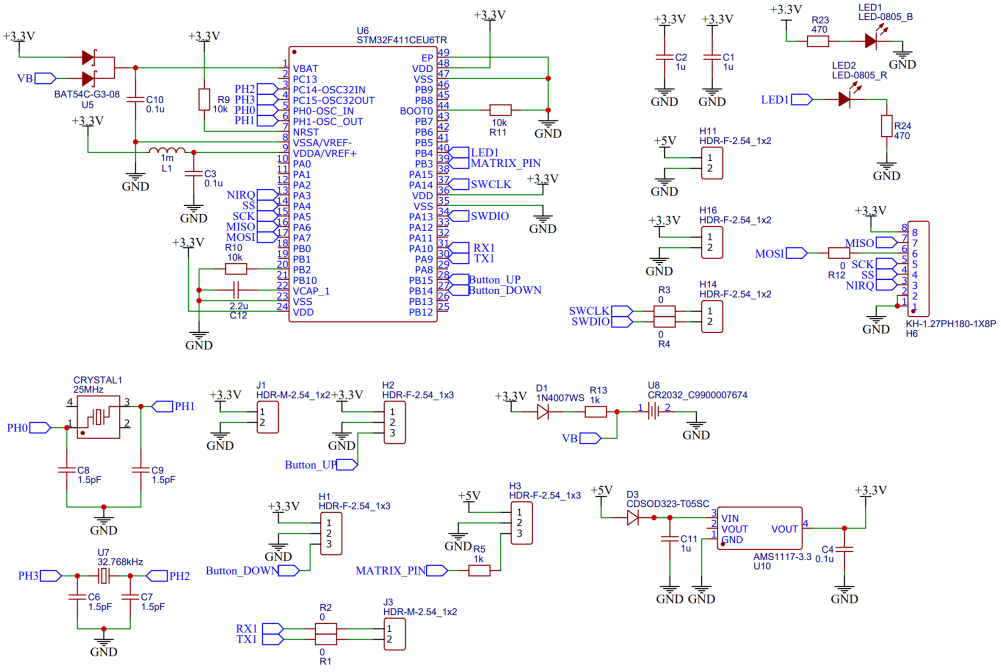


Рис. 4.1: Принципиальная схема устройства

После изготовления платы проведён монтаж электронных компонентов, включающий установку и пайку всех элементов в соответствии с разработанной схемой. Готовая плата прошла тестирование на наличие коротких замыканий и обрывов дорожек с целью обеспечения её функциональной надёжности.

Программирование и отладка микроконтроллера на плате осуществлялись с использованием устройства WeAct Mini Debugger–компактного программатора–отладчика, поддерживающего протокол StLink 2.1 для программирования микроконтроллеров STM32 и подключающегося к компьютеру через USB–порт.

4.2 Программное обеспечение конечных устройств.

Для обеспечения минимальных задержек и надёжного выполнения разнообразных задач на однокристальном контроллере STM32F411 применяется операционная система реального времени FreeRTOS, интегрированная через STM32Duino. Архитектура FreeRTOS основана на ядре планировщи-

ка, которое управляет созданными задачами (tasks), их приоритетами и выделением ресурсов процессорного времени и оперативной памяти (см. рисунок 4.2). Каждая задача располагает собственным стеком и может быть настроена на срабатывание по программируемому таймеру, что позволяет гарантировать её запуск через заданные интервалы времени. Планировщик FreeRTOS реализует приоритетный вытесняющий (preemptive) режим: если текущая задача превысила выделённый ей квант или в систему добавилась задача более высокого приоритета, управление незамедлительно передаётся вновь приоритетной задаче. Такая модель обеспечивает детерминированное поведение и высокую отзывчивость конечного устройства.

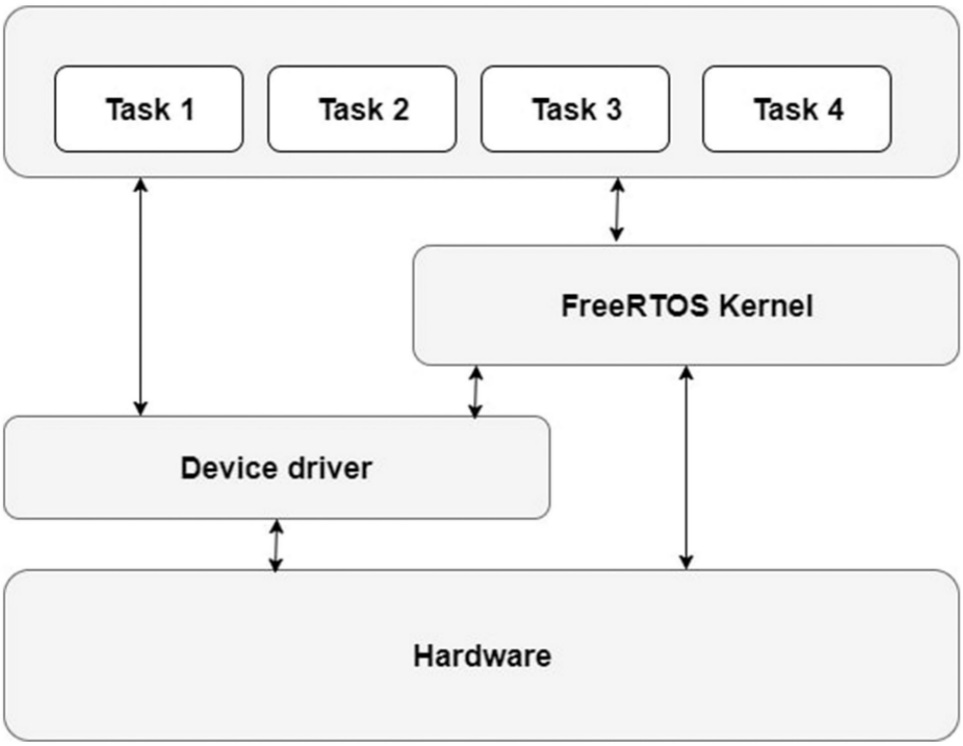


Рис. 4.2: Упрощенная архитектура FreeRTOS [14]

Для повышения отказоустойчивости и сокращения времени обработки входящих сообщений отправка и приём сообщений разбиты на множество мелких задач, каждая из которых выполняет строго определённый фрагмент обработки. Например, непосредственно чтение очередного пакета из буфера радиомодуля SI4432 обладает самым высоким приоритетом, затем следует проверка целостности данных и, лишь после этого, формирование отладочных сообщений на последовательный порт — см. рисунок 4.3. Подобное разделение предотвращает «залипание» конвейера обработки и минимизирует

вероятность образования битых пакетов.

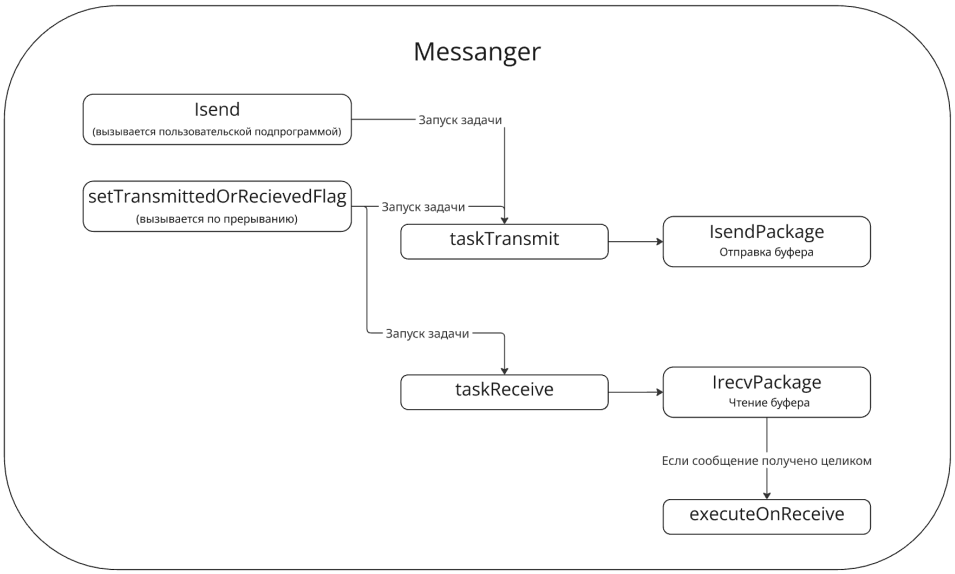


Рис. 4.3: Упрощенное взаимодействие задач в библиотеке Messanger

Программная структура конечного устройства выстроена по двухуровневой схеме (рис. 4.4). Низкоуровневая прошивка, написанная на C/C++ с использованием STM32Duino и FreeRTOS, предоставляет API для всех аппаратно-зависимых операций: настройка периферии, обмен данными с радиомодулем, синхронизация задач и работа с очередями сообщений. Высокоуровневая прошивка реализована на языке JavaScript и исполняется на встроенном интерпретаторе (например, JerryScript). Такая двухслойная архитектура позволяет оперативно изменять логику работы устройства без перепрошивки микроконтроллера: достаточно обновить JavaScript-модули через удалённую конфигурацию. Пакетный режим передачи сообщений с гибкими настройками параметров радиомодуля SI4432 доказал свою эффективность при прототипировании, сочетая низкое потребление энергии с высокой скоростью и надёжностью связи.

4.2.1 Интерпретатор JavaScript

TinyJS представляет собой компактный (приблизительно 2000 строк кода) интерпретатор JavaScript, реализованный на C++ без внешних зависимостей, что обеспечивает простую интеграцию в проекты для микроконтроллеров. Он поддерживает работу с переменными, массивами и структурами, функции (включая передачу параметров и возврат значений), а также встро-

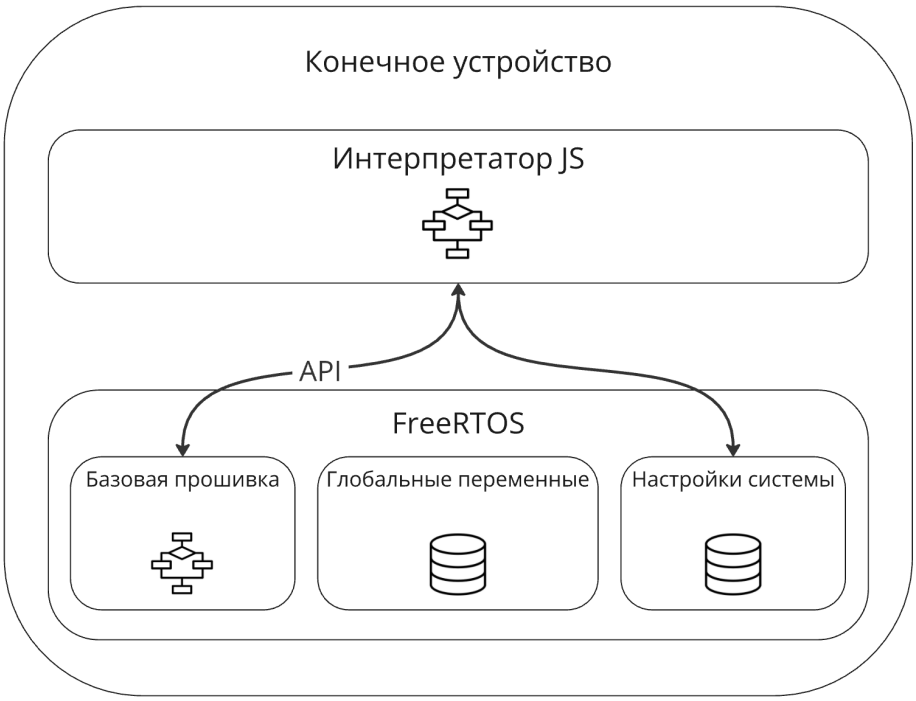


Рис. 4.4: Реализация конечного устройства

енные механизмы разбора и генерации JSON. Интерпретатор допускает вызов C/C++ кода из скриптов и обеспечивает базовую модель объектов с наследованием; выполнение производится напрямую из исходного текста с помощью рекурсивного нисходящего парсера. Благодаря отсутствию промежуточной компиляции и минимальному объёму кода TinyJS удобно использовать для настройки параметров, автоматизации простого поведения и загрузки/сохранения конфигураций на устройствах с ограниченными ресурсами. Несмотря на то, что проект не претендует на полное соответствие стандарту ECMAScript и не оптимизирован для интенсивных вычислений, его лёгкая архитектура позволяет встраивать интерпретатор в микроконтроллерные платформы с достаточным объёмом оперативной памяти.

4.3 Модель взаимодействия устройств

В настоящем разделе приводится формальная спецификация архитектуры и протокола обмена сообщениями в децентрализованной системе домашней автоматизации на основе широковещательной передачи.

4.3.1 Формат широковещательных сообщений.

- Устройства обмениваются данными по широковещательному радиоканалу без установки соединения и без участия промежуточных брокеров.
- Каждое устройство постоянно «слушает» эфир и при получении пакета выполняет фильтрацию по подписанным шаблонам, что соответствует паттерну Subject–Observer с разделяемой средой доступа (shared medium) и логикой point-to-multipoint.
- Для связи устройств в рамках сети используется пакетный режим передачи сообщений, представляющий абстракцию в виде сообщений и реализующийся в Messanger: программисту не нужно следить за длинной сообщений, по отдельности отправлять каждый пакет. В отличие от паттерна Pub–Sub, Subject–Observer не требует промежуточного звена в виде сервера. Наглядно демонстацию отличий Subject–Observer от Pub–Sub можно увидеть на рис. 4.5.
- Для проверки целостности пакетов используется алгоритм CRC-16. При общем ограничении размера пакета в 64 байта это обеспечивает высокий уровень обнаружения ошибок при относительно низкой служебной нагрузке (около 3% от общего объёма пакета).

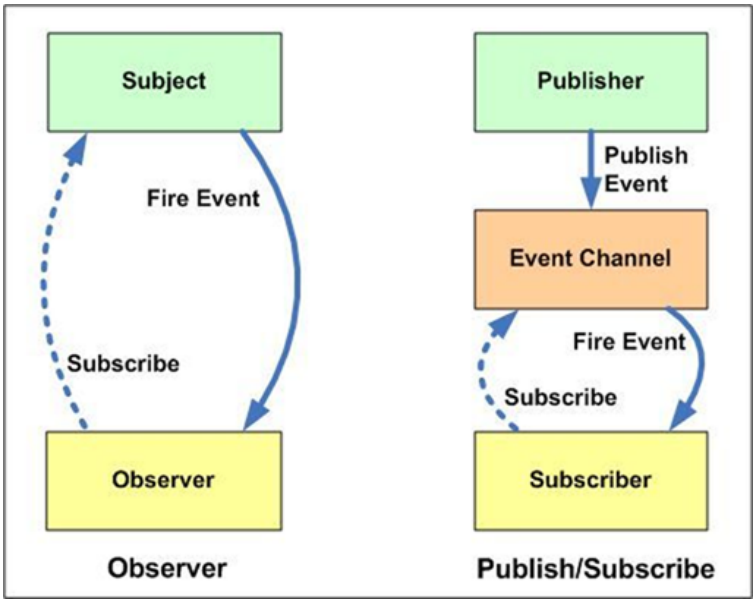


Рис. 4.5: Отличие Subject–Observer от Publisher–Subscriber [17]

4.3.2 Типы сообщений

Configuration Message (адресованное)

- `Install`:

$$(id_{\text{recv}} : \textbf{\textit{install}} : id_{\text{sender}} : topic : command)$$

  Добавляет в локальный набор правил запись $(id_{\text{sender}},\ topic \rightarrow command)$.

- `Delete`:

$$(id_{\text{recv}} : \textbf{\textit{delete}} : id_{\text{sender}} : topic : command)$$

  Удаляет из локального набора правил соответствующую запись.

- `Edit`: реализуется как последовательность `delete` старой и `install` новой записи.

Event Message (широковещательное)

$$(* : \textbf{\textit{event}} : id_{\text{sender}} : topic : payload)$$

4.3.3 Алгоритм обработки Event Message

1. **Приём и валидация.** Проверка контрольной суммы и механизма аутентификации; в случае неуспеха сообщение отбрасывается.
2. **Фильтрация.** По полям $(id_{\text{sender}},\ topic)$ отбираются все правила из локального набора.
3. **Выполнение кода.** Для каждого найденного правила последовательно выполняются:
   (a) *payload* из Event Message;
   (b) *command* из правила.

4.3.4 Формат команд и правила (Rules)

Каждое устройство хранит набор правил:

$$Rule \equiv (id_{\text{sender}},\ topic,\ command),$$

Оба фрагмента — *payload* и *command* — могут быть произвольными JS-конструкциями, например:

Пример простого payload и command

```
// payload
Switcher02 = 1;

// command
LedSwitch(Switcher02);
```

Пример с условием

```
// payload
temp = 17.3;

// command
if (temp < 18) {
  PowerSwitch(1);
} else{
  PowerSwitch(0);
}
```

5 Глава 4. Распределённый отказаустойчивый кластер.

5.1 Описание кластера.

Система представляет собой распределённый отказаустойчивый кластер, построенный на основе лёгкого Kubernetes-дистрибутива K0s и высокопроизводительной шины обмена сообщениями NATS с включённым механизмом JetStream. Архитектура обеспечивает устойчивую ретрансляцию данных между узлами, расположенными в различных географических или сетевых сегментах, даже при использовании динамических IP-адресов и NAT.

5.1.1 Общее положение.

Кластер состоит из двух узлов с ролями:

- **Контроллер-воркер (controller+worker)**: выполняет функции управляющей плоскости Kubernetes, а также может запускать пользовательские поды.
- **Рабочий узел (worker)**: предназначен исключительно для запуска сервисов ретрансляции и NATS-кластера.

Использование единственного контроллера-воркера упрощает инфраструктуру, снижает накладные расходы на управление и при этом позволяет сохранить возможность горизонтального масштабирования рабочих нагрузок. Аналогично, отдельный воркер-узел гарантирует изоляцию сервисов и высокую доступность при перезагрузке управляющего компонента.

5.1.2 Описание устройств.

В качестве физических и виртуальных хостов используются:

- **Ноутбук Lenovo G710**: процессор Intel x86_64, 8 ГБ RAM, Ubuntu 24.04 LTS. Предназначен для роли контроллера-воркера и локальной разработки.

- **Orange Pi 5**: одноплатный компьютер на базе ARM64 (Rockchip RK3588), 4 ГБ RAM, Armbian 25.2.1 (CLI-редакция на базе Ubuntu 24.04 Noble). Используется в качестве рабочего узла с целевой архитектурой ARM, что потребовало настройки сборки многоархитектурных контейнеров.

Наличие хостов разной архитектуры обуславливает необходимость мультиплатформенной сборки образов (amd64 и arm64), что реализовано в CI-пайплайне.

5.1.3 Инструменты для развертывания кластера.

Для автоматизированной установки и конфигурации кластера используются следующие решения:

K0s.

K0s представляет собой комплект единого бинарного файла, включающего все ключевые компоненты Kubernetes (API-сервер, etcd, kubelet, kube-proxy). Основные преимущества выбора K0s:

- **Лёгковесность** — отсутствие множества внешних зависимостей и компонентов, упрощённая дистрибуция.

- **Упрощённая установка** — минимальное количество команд для развёртывания управляющей плоскости и рабочих узлов.

- **Компактная конфигурация** — единый YAML-файл для настройки всех элементов кластера.

- **Гибкость** — возможность включения ролей контроллера и воркера на одном узле через флаг `--enable-worker`.

Типичная конфигурация в `k0s.yaml` содержит указание сетевого плагина (`network.provider: kuberouter`), диапазонов адресов

(`podCIDR: 10.244.0.0/16`, `serviceCIDR: 10.96.0.0/12`) и параметров хранения etcd.

K0sctl.

K0sctl — это CLI-инструмент для управления жизненным циклом K0s-кластера. Позволяет:

- Подготавливать узлы (устанавливать бинарники, настраивать службы) по SSH.
- Задавать роли `controller` и `worker` в едином файле `k0sctl.yaml`.
- Автоматически обновлять версии K0s и подключать новые узлы.

Пример раздела `hosts` в `k0sctl.yaml`:

```
hosts:

* role: controller
  ssh:
  address: 192.168.1.123
  user: root
  keyPath: \~/.ssh/id\_rsa.pub
  installFlags:

  * \--enable-worker
* role: worker
  ssh:
  address: 192.168.1.124
  user: root
  keyPath: \~/.ssh/id\_rsa.pub
```

Helm.

Для управления приложениями в кластере применяется Helm:

- **Удобство шаблонизации** — позволяет параметризовать манифесты и быстро изменять конфигурацию.
- **Управление зависимостями** — возможность агрегировать несколько чартов (например, NATS + Prometheus) одним запуском.
- **Версионирование релизов** — откат к предыдущим версиям чартов в один шаг.

В частности, чарт NATS развёрнут через Helm с параметрами реплик, JetStream и PVC:

```
config:
cluster:
enabled: true
replicas: 2
jetstream:
enabled: true
fileStore:
pvc:
size: 2Gi
```

5.1.4 Сервисы.

В кластере функционируют две основные подсистемы:

- **NATS-кластер** — обеспечивает модель publish–subscribe. JetStream гарантирует доставку и хранение сообщений на диске.
- **Relay-service** — Python-приложение, подписанное на общий топик NATS. Получив сообщение, оно фильтрует по полю `id` и перенаправляет только подходящие сообщения на локальный радиомодуль.

Архитектура шлюзов позволяет легко добавлять новые зоны и узлы без изменения кода: достаточно задать шаблоны для фильтрации и подключить новый экземпляр Relay-service.

5.1.5 Автоматизация сборки и развертывания.

CI/CD реализован на базе GitHub Actions и включает этапы:

1. **Test**: запуск `pytest` для проверки логики Relay-service.
2. **Build and Push Test Image**: сборка мультиархитектурных образов для amd64 и arm64 с помощью Docker Buildx и публикация под тегом `test`.
3. **Smoke**: базовое тестирование собранного контейнера (в перспективе — полноценные end-to-end-тесты).
4. **Publish**: вычисление следующей семантической версии из списка тегов в GitHub Container Registry и ретегирование образа под `latest` и `X.Y.Z`.

Пример фрагмента workflow для определения версии:

```
# Получение списка тегов

folders\_json=\$(gh api --paginate /users/\${OWNER}/packages/

# Фильтрация X.Y.Z

# Вычисление максимальной версии и инкремент патча
```

Благодаря такой архитектуре и автоматизации новые образы становятся доступны для развертывания сразу после успешного прохождения CI/CD, что минимизирует время вывода обновлений в продакшен.

5.1.6 Этапы выполнения работ.

Процесс построения отказоустойчивого распределённого кластера включал в себя следующие ключевые этапы:

1. **Подготовка ОС и окружения**:

- Установка Ubuntu 24.04 LTS на контроллер-воркер (Lenovo G710) и Armbian 25.2.1 (на базе Ubuntu 24.04) на Orange Pi 5.
- Настройка SSH-доступа, установка необходимых зависимостей (curl, openssh-server, rsync и пр.).

2. **Развёртывание кластера Kubernetes с помощью K0s и K0sctl**:
    - Генерация `k0sctl.yaml` с описанием ролей и SSH-доступа.
    - Развёртывание командой `k0sctl apply`.
    - Проверка статуса узлов и корректности функционирования компонентов.
3. **Развёртывание системы обмена сообщениями NATS**:
    - Подготовка `nats-values.yaml` с включением кластерного режима и JetStream.
    - Установка Helm-чарта с параметрами хранения (PVC) и количеством реплик.
    - Проверка создания StatefulSet, PVC и подключения к NATS-эндпоинтам.
4. **Разработка и контейнеризация приложения Relay-service**:
    - Написание Python-кода для фильтрации сообщений по идентификатору.
    - Создание Dockerfile с поддержкой многоплатформенной сборки.
    - Тестирование образов локально на обеих архитектурах.
5. **Настройка CI/CD на GitHub Actions**:
    - Автоматическая сборка и тестирование на каждый push в ветку.
    - Мультиархитектурная сборка через Docker Buildx.
    - Автоматическое версионирование образов и публикация в GitHub Container Registry.
6. **Проброс портов и доступ к кластеру через NAT**:

- Использование динамического DNS (no-ip) на этапе тестирования.
- Конфигурация туннелей и проброса портов для доступа к API и NATS из внешней сети.
- Планируемый переход на статический IP или VPN-сеть в финальной версии.

Этапы выстраивались по нарастающей сложности — от базового окружения до полной автоматизации, что позволило оперативно выявлять и устранять ошибки на каждом уровне развертывания.

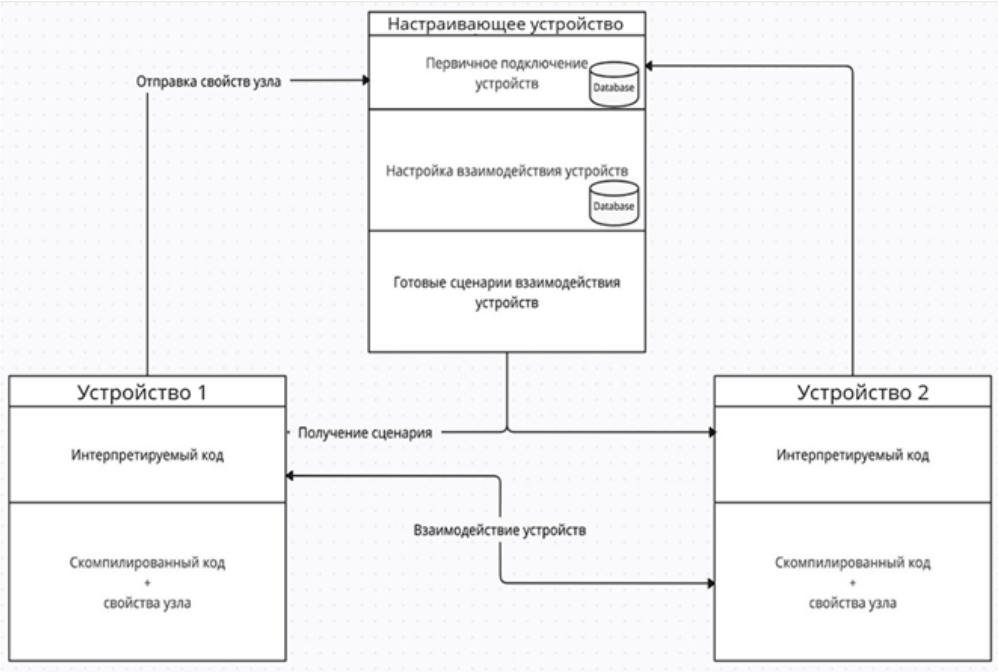


Рис. 5.1: Пример взаимодействий конечных устройств

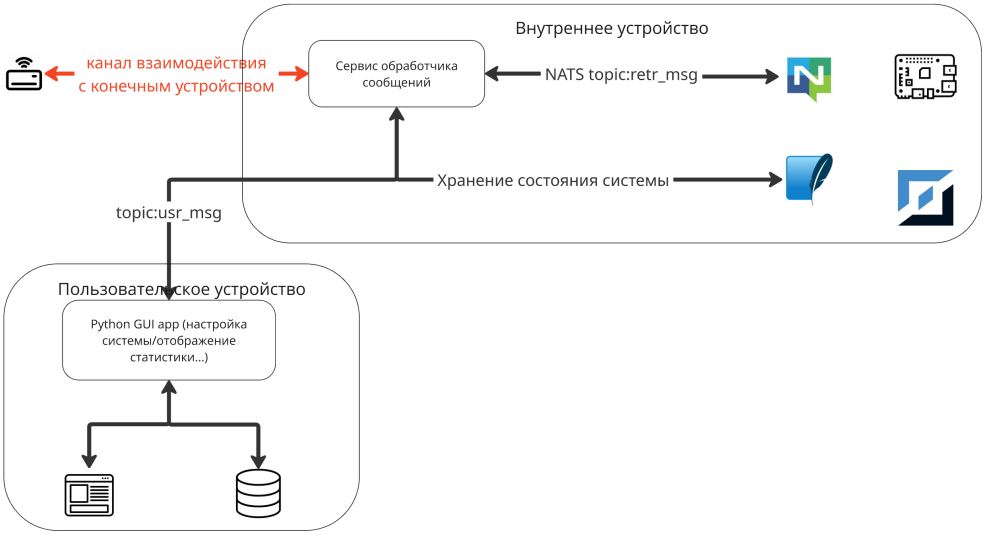


Рис. 5.2: Компонентная структура и процессы взаимодействия в кластере K0s предложеннной системы

6 Глава 5

6.1 Графический интерфейс

Был написано графическое приложение рис. 6.1 на языке Python с использованием фреймворка Qt, в котором производилась настройка системы и ее отладка.

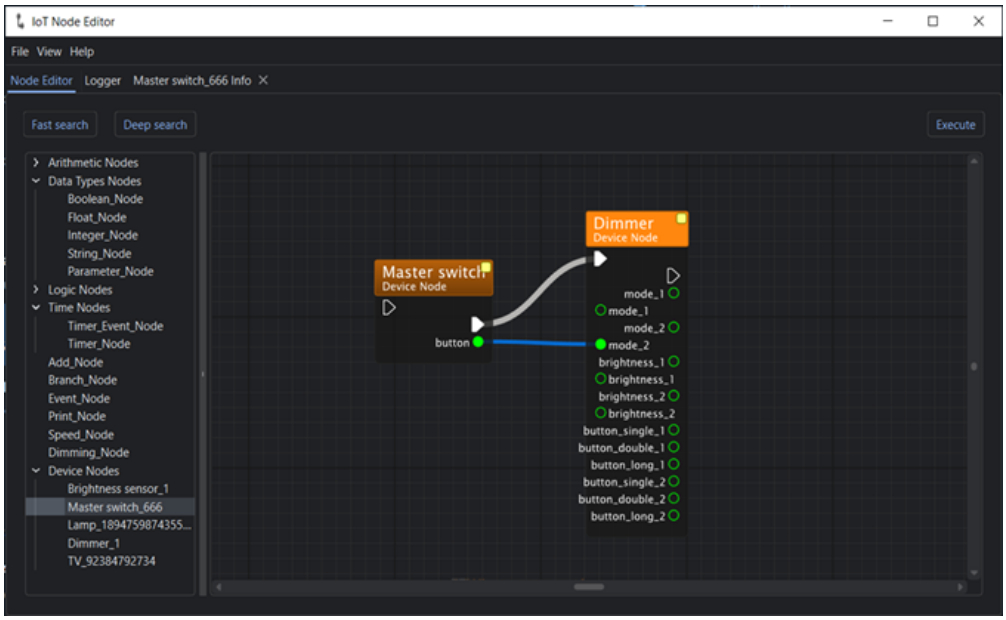


Рис. 6.1: Графическое приложение с созданным внутри сценарием для конечных устройств

Графический интерфейс позволяет прописывать и более сложные сценарии взадимодействия устройств, что можно увидеть на рис. 6.2

6.2 Планировщик

Было опробовано два варианта распределения сценариев: их хранение и исполнение на устройстве-отправителе, порождающем событие и хранение сценариев на устройстве-исполнителе. В рамках проведенных экспериментов было установлено, что некоторые сложные команды, последовательный

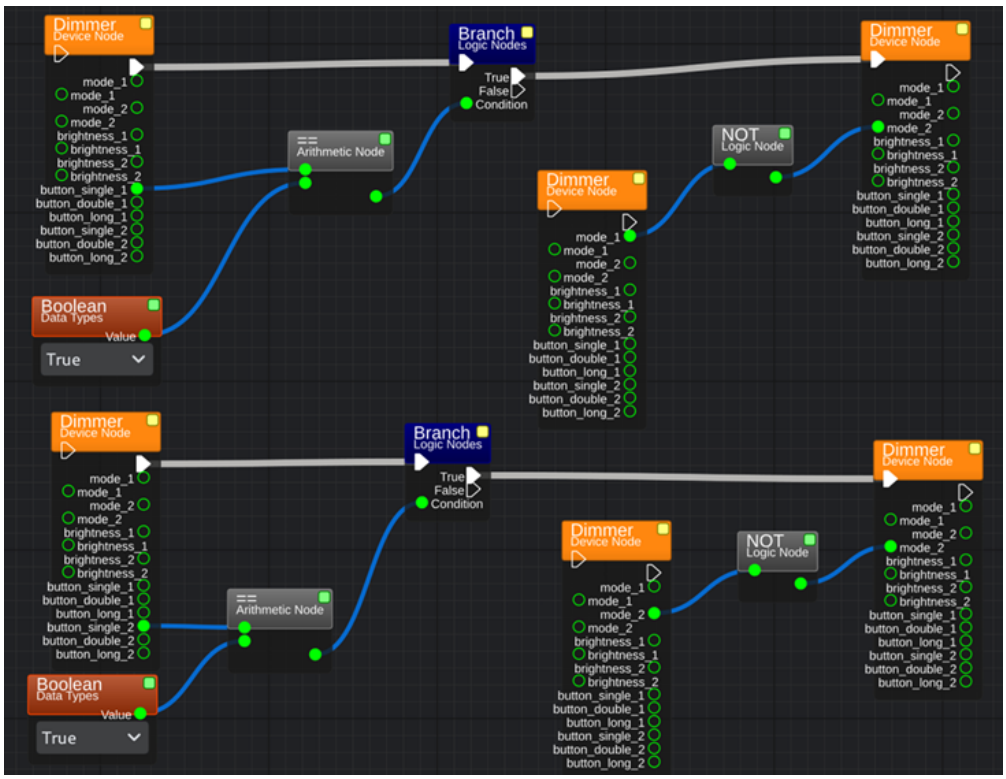


Рис. 6.2: Пример более сложного сценария взаимодействия устройств

набор действий для первого варианта проводят к значительному увеличению времени обработки и излишней загруженности радиоканала, что потенциально уменьшает максимальное число устройств в системе. Планировщик осуществляет преобразование визуальных узлов и связей в JS код, выполняя синтаксический анализ, преобразованием узлов в лексемы с последующем объединением их в единый исполняемый файл. Так же в задачи планировщика входит разбиение файла настроек на компоненты связанности и определение целевого устройства, на которое будет отправлен сценарий. Также для дальнейшего тестирования система была сконфигурирована для работы с использованием ЦУД и для распределенной работы.

7 Глава 6

7.1 Тестирование и полученные оценки

Для корректности тестирования и сравнимости подходов, ввиду отличия протоколов передачи данных, использования разных микроконтроллеров в существующих системах и иных отличий, исходная система была переконфигурирована для работы как с ЦУД, так и без ЦУД. Таким образом, проведя замеры задержек от возникновения события до реакции, можно оценить численно преимущество предложенной архитектуры. В таблице приведены результаты замеров на каждом этапе: время передачи пакета сообщения, время обработки события для каждой системы, что позволило использовать эти данные без привязки к конкретной реализации. Были проведены замеры времени задержек выполнения программы на микроконтроллере STM32F411 (25МГц), выступавшим в качестве конечного устройства, и одноплатным компьютером Raspberry pi 3b+. Тестирование одноплатного компьютера проводилось на системе Raspberry Pi OS 19.11, базирующейся на Debian 12. Использовался GCC 15 с опциями: «-mcpu=native, -O2». При тестировании микроконтроллеров использовалась STM32duino FreeRTOS v10.3.2 с библиотекой RadioLib v7.1.2. Радиомодули, представленные SI4432, использовались со стандартными параметрами.

Использовалось готовое решение на базе протокола Zigbee с ЦУД в виде YNDX-00510 [7] и лампой YNDX-00558. Лучшим способом сокращения задержек в СУД является уменьшение задержек в передаче информации.Предложенный выше подход позволяет почти в вдвое сократить задержку (в рамках тестирования спроектированной СУД) и более чем втрое по сравнению с представленными на рынке решениями.

п/п	Тип операции	Время операции
1	Передача в рамках одной зоны	197 мс
2	Выполнение 1 примитива интерпретатора на микроконтроллере	18 мс
3	Выполнение 1 примитива интерпретатора на одноплатном компьютере	0,24 мс
4	Общая задержка системы при использовании ЦУД	407 мс
5	Общая задержка системы без использования ЦУД	231 мс
6	Задержка СУД «Умный дом с Алисой» от Яндекс	∼470 мс

Таблица 7.1: Среднее время выполнения операций

7.2 Вывод

Предложен подход к построению распределенной системы домашней автоматизации, проведено сравнение с решением, представленным на рынке. В дальнейшем предполагается исследовать возможные уязвимости данной системы, построить и протестировать прототип распределенной системы с использованием шлюзов-ретрансляторов, объединенных в единую сеть посретсвом VPN.

8 Литература

[1] Колыбельников А. И. Обзор технологий беспроводных сетей // ТРУДЫ МФТИ. 2012. Т. 4. № 2 С. 3–29.

[2] Шерчков А. В. Анализ существующих технических решений «Умный дом» // Молодежный вестник Уфимского государственного авиационного технического университета. 2020. № 1 С. 153–155.

[3] Montori F., Contigiani R., Bedogni L., Is WiFi suitable for energy efficient IoT deployments? A performance study // 2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI). 2017. pp. 1-5, doi: 10.1109/RTSI.2017.8065943.

[4] Тукмачева Ю. А. Обзор и анализ автоматизированных систем «Умный дом», представленных на российском сегменте рынка // Вестник науки. 2022. Т. 4. № 10 С. 139–144.

[5] Некрасов П. В., Жариков А. М., Козин Д. А. Обеспечение безопасности беспроводных каналов связи киберфизических систем типа «Умный дом» // Безопасность информационных технологий. 2024. Т. 31. № 1 С. 54–62.

[6] Vardakis G., Hatzivasilis G., Koutsaki E., Papadakis N. Review of smart-home security using the internet of things // Electronics. 2024. No 13 P. 33–43.

[7] Центр умного дома «Яндекс Хаб» модель YNDX-00510. Инструкция по эксплуатации [Электронный ресурс]: URL:https://yastatic.net/s3/doc-binary/src/support/smart-home/hub/hub-manual.pdf (дата обращения: 17.03.2025).

[8] Specifications and Documents Mesh Protocol [Электронный ресурс]: URL:https://www.bluetooth.com/specifications/specs/html/?src=MshPRT_v1.1/out/en/index-en.html (дата обращения: 11.05.2025).

[9] Zigbee Specification Revision 23 [Электронный ресурс]: https://csa-iot.org/wp-content/uploads/2023/04/05-3474-23-csg-zigbee-specification-compressed.pdf (дата обращения: 20.05.2025).

[10] IEEE Std 802.15.4-2020: IEEE Standard for Low-Rate Wireless Personal Area Networks (LR-WPANs) [Электронный ресурс]: URL:https://standards.ieee.org/standard/802_15_4-2020.html (дата обращения: 20.05.2025).

[11] Richard Alena, Ray Gilstrap, Jarren Baldwin, Thom Stone, Pete Wilson. Fault Tolerance in ZigBee Wireless Sensor Networks // NASA Ames Research Center, Moffett Field [Электронный ресурс]: URL:https://ntrs.nasa.gov/api/citations/20110012426/downloads/20110012426.pdf?utm_source=chatgpt.com (дата обращения: 20.05.2025).

[12] Zigbee Pro Stack User Guide (JN-UG-3101), NXP Semiconductors [Электронный ресурс]: URL:https://www.nxp.com/docs/en/user-guide/JN-UG-3101.pdf (дата обращения: 20.05.2025).

[13] Simon S. Lam, A Carrier Sense Multiple Access Protocol for Local Networks // Computer Networks. 1976. Vol. 4 Issue 1 P. 21–32,

[14] Qutqut M., Al-Sakran A., Almasalha F., Hassanein H. // Comprehensive Survey of the IoT Open Source OSs. IET Wireless Sensor Systems. 2018.

[15] Сравнительный анализ стандартов связи для сетей IoT [Электронный ресурс]: URL:https://habr.com/ru/companies/msw/articles/720518/ (дата обращения: 11.05.2025).

[16] Wi-Fi HaLow™ and LoRaWAN: How do the technologies compare? [Электронный ресурс]: URL:https://dev.

wi-fi.org/zh-hans/beacon/y-zachary-freeman/wi-fi-halow-and-lorawan-how-do-the-technologies-compare (дата обращения: 11.05.2025).

[17] Observer vs Pub-Sub [Электронный ресурс]: URL:https://habr.com/ru/articles/270339/ (дата обращения: 11.05.2025).

[18] Generic Access Profile (GAP) Roles [Электронный ресурс]: URL:https://developerhelp.microchip.com/xwiki/bin/view/applications/ble/introduction/bluetooth-architecture/bluetooth-host-layer/gap/roles/ (дата обращения: 19.05.2025).

[19] Bachelor Seminar Paper [Электронный ресурс]: URL:https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0148.pdf (дата обращения: 20.05.2025).

9 Приложение

1. Solovjev D. Distributed smart home cluster [Электронный ресурс]: URL:https://github.com/Dmitrij-Solovjev/distributed-smart-home-cluster (дата обращения: 11.05.2025).
   Примечание: Кластер на базе K0s с сервисами для ретрансляции сообщений и общей настройки системы.

2. Solovjev D. IoT Base Device [Электронный ресурс]: URL:https://github.com/Dmitrij-Solovjev/IoT-Base-Device (дата обращения: 11.05.2025).
   Примечание: Прошивка для микроконтроллера на FreeRTOS, реализующая логику домашней автоматизации.