Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Кафедра системного программирования

Балакина Екатерина Сергеевна

Интеграция робототехнической ОС (ROS) с кибернетическим контроллером ТРИК

Выпускная квалификационная работа

Научный руководитель:
ст. преп. кафедры системного программирования СПбГУ Я. А. Кириленко

Рецензент:
генеральный директор ООО "КиберТех Лабс" Р. М. Лучин

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems Software Engineering

Balakina Ekaterina

Integration of Robot Operating System (ROS) with TRIK cybernetic controller

Graduation Thesis

Scientific supervisor:
senior lecturer Iakov Kirilenko

Reviewer:
"CyberTech Labs, LLC" CEO, Roman Luchin

Saint-Petersburg
2019

Оглавление

Введение

В программировании вообще и для робототехнических платформ в частности существует множество типичных, повторяющихся задач. Таковы сбор информации с датчиков, ее обработка и визуализация, организация межпроцессного взаимодействия, реализация многих алгоритмов робототехники. Все это ставит вопрос о переиспользовании кода с вынесением решений на отдельный уровень абстракции. Для этих целей используются различные наборы библиотек, которые называются middleware — программное обеспечение промежуточного уровня.

Одной из самых популярных библиотек промежуточного уровня для робототехники на данный момент является ROS — Robotics Operating System [8]. ROS — это распределенная система, которая выполняет функции, стандартные для операционной системы: низкоуровневый контроль устройств, передачу сообщений между процессами и другие. Кроме того, ROS предоставляет реализованные алгоритмы, часто используемые при создании робототехнических моделей – например, SLAM алгоритмы, которые вызывают особенно большой интерес в научном сообществе. Желательно иметь возможность легко тестировать такие алгоритмы – хотя бы оценивать их применимость, не вдаваясь в детали реализации под ТРИК. Кроме того, некоторые сложные алгоритмы требуют ресурсов, которыми контроллер не располагает, поэтому полезно уметь подключать более мощный вычислитель. С помощью сети ROS, например, можно получать видеоданные с контроллера, а потом использовать их на ноутбуке в алгоритмах, предоставляемых ROS. Дополнительно, в ROS интегрированы вспомогательные инструменты для визуализации и моделирования, которые помогут быстро получать обратную связь при разработке и тестировании.

Для решения задач прототипирования ROS был выбран и для платформы ТРИК. Выбор определили: хорошо развитое сообщество, документация и интеграция с множеством известных библиотек и инструментов, вроде библиотеки компьютерного зрения OpenCV, библиотеки для работы с облаками 3D точек и 3D геометрией PCL и симулятора

роботов Gazebo. Однако сейчас на контроллере ROS не используется. Причина заключается не в неуместности ROS для ТРИК, а в том, что разработки не доведены до конца, и ROS не стал частью конечного продукта.

В предыдущих разработках выяснилось, что ROS не подходит для ТРИК в качестве полноценного middleware, который позволил бы сопоставить отдельную компоненту — узел ROS — каждому сенсору или актуатору, и таким образом решил бы проблему межпроцессного взаимодействия. Тем не менее, получилось использовать ROS с персональным компьютером в контуре управления: получать данные от контроллера по сети и работать с ними с использованием стандартных инструментов ROS.

Осенью 2016 года появился ROS 2. Новый ROS заявлен как более подходящий для ряда задач и ситуаций, для которых ROS 1 не предназначен. ROS 2 намного более терпим к сетям плохого качества благодаря QoS, и он стал полностью децентрализованным, что упростит создание сетей из взаимодействующих роботов. Однако неизвестно, насколько сложным окажется процесс интеграции ROS 2 с контроллером ТРИК относительно преимуществ, которые он предоставляет, и учитывая его пока не слишком большую популярность — этот вопрос еще предстоит прояснить, поэтому фокус внимания на ROS 1 остается актуальным.

1. Постановка задачи

Таким образом, поставлена цель сделать ROS частью конечного продукта и получить возможность использовать его в экспериментах с участием контроллера ТРИК. Для этого были выделены следующие задачи:

1. проанализировать существующее решение интеграции ROS 1 с контроллером ТРИК и при наличии существенных недостатков:
    - предложить улучшенный вариант архитектуры решения;
    - реализовать соответствующий улучшенной архитектуре модуль для взаимодействия контроллера с ROS 1;
2. дополнить модуль для взаимодействия контроллера с ROS 1 работающими примерами;
3. исследовать возможность интеграции с ROS 2 и при наличии таковой интегрировать.

2. Обзор

2.1. ТРИК

Контроллер ТРИК [14] оснащен центральным процессором ARM, работающим под управлением операционной системы на основе ядра Linux. Среда исполнения trikRuntime содержит набор библиотек для обработки аудио- и видеоданных, управления сервоприводами и моторами, сбора показаний с аналоговых и цифровых датчиков. Для взаимодействия с пользовательскими программами существует библиотека trikControl. Она обеспечивает доступ к периферии робота и некоторую логику обработки информации с датчиков. За доступ к периферии и ее инициализацию отвечает класс Brick, а за работу с конкретными элементами периферии – моторами и сенсорами – классы Sensor, ServoMotor и PowerMotor.

Для сборки ядра прошивки используется система сборки BitBake от OpenEmbedded, позволяющий собирать пакеты и дистрибутивы для Linux под встроенные платформы. В рецептах для BitBake указывается, как именно и в какой последовательности собирать пакеты. Для ROS существуют готовые рецепты от BMW Car IT GmbH [3], для ROS 2 рецепты находятся в разработке.

2.2. ROS

ROS представляет из себя распределенную систему, состоящую из узлов (nodes) и каналов (topics) и реализующую паттерн издатель-подписчик. Подключаясь к главному узлу (master node), остальные узлы сообщают о своем существовании. Для взаимодействия по типу запрос-ответ (request/reply) в ROS существует отдельный механизм — сервисы (services). Для сообщений в ROS существует специальные типы данных, часто нужные в робототехнике, например, облака точек, изображения или сообщения с джойстика.

С одной точки зрения, ROS можно условно разделить на 2 составляющие:

1. коммуникационная прослойка, отвечающая за взаимодействие компонент робота друг с другом и с внешней средой;

2. набор алгоритмов, для обработки получившихся в результате взаимодействия данных, а также некоторые другие инструменты, например, визуализации и симуляции.

С другой точки зрения, в ROS можно выделить следующие 3 группы:

1. клиентские библиотеки, написанные на разных языках программирования (roscpp, rospy);

2. инструменты для сборки ROS-приложений (catkin_make);

3. готовые пакеты для работы с ROS-сетью (roscore, rosgraph).

2.2.1. ROS 1

В ROS 1 разделяются узел (node), который запускается один внутри одного процесса, и нодлеты (nodelets), которые можно запускать несколько из одного процесса, что может сильно увеличить производительность за счет уменьшения издержек на пересылку информации между процессами. В работе 2016 года [11] замена nodes на nodelets позволила значительно уменьшить нагрузку системы, однако особенности Nodelet API накладывают ограничения на реализацию таких узлов: нодлеты могут быть использованы только из клиентской библиотеки C++, и они не могут предоставлять сервисы. С другой стороны, основная модель использования ROS на контроллере подразумевает более мощный вычислитель в контуре управления и не подразумевает активное использование узлов на ограниченном в ресурсах контроллере, поэтому преимущества nodelets для контроллера сомнительны.

ROS 1 разрабатывался почти полностью с нуля, в нем используется собственный формат сериализации, транспортный протокол, работающий поверх TCP или UDP и использующий протокол XML-RPC, и механизм обнаружения новых узлов сети (discovery mechanism). Для обнаружения главного узла, каждый узел, должен знать его URI.

2.2.2. ROS 2

ROS создавался в далеком 2007 году для нужд вполне конкретного робота: PR2. Последний обладал конкретными характеристиками, которые повлияли на развитие ROS в дальнейшем. ROS набрал популярность, его стали использовать для самых разных роботов, но все же с течением времени обнаружилось, что для некоторых распространенных случаев использования ROS изначально не проектировался вообще, а потому подходит не очень хорошо. Поэтому появился ROS 2.

ROS 2 заявлен как более подходящий для:

1. маленьких встраиваемых платформ, чистых "железок". Теперь они могут быть участниками сети ROS напрямую;
2. систем реального времени;
3. неидеальных сетей. Новый ROS толерантнее к перебоям связи и плохому Wi-Fi;
4. построения сетей из роботов.

В ROS 2 сериализация, механизм обнаружения и транспортный протокол представлены абстрактным интерфейсом. Разработчики ROS 2 остановили свой выбор на DDS (Data Distribution Service) [4] в качестве наиболее подходящей реализации этого интерфейса [6], однако можно реализовать его самостоятельно, комбинируя готовые библиотеки в открытом доступе, например, легковесную библиотеку nanomsg и Protobuf [7].

Использование DDS позволило ROS 2 стать децентрализованной системой: в ней, в отличие от ROS 1, отсутствует мастер-узел. Так, построение сетей из роботов должно стать более простой задачей, а падение мастер-узла больше не должно приводить к падению всей сети. Отсутствие мастер-узла, в совокупности с Quality of Service policies от DDS, делает сеть более стабильной и толерантной к плохой связи.

ROS 1 официально тестируется только на Ubuntu, но поддерживается сообществом на других дистрибутивах Linux и на OS X. ROS 2

тестируется на Ubuntu Xenial, OS X El Capitan и на Windows 10. Возможность использовать ROS на всех основных операционных системах привлекательна для ТРИК, так как большая часть его пользователей используют Windows и OS X.

Кроме того, ROS 2 имеет улучшенный API, так как не связан необходимостью поддерживать устаревшие и не самые красивые решения для обратной совместимости. В новом API можно использовать C++11, C++14, планируется добавить поддержку C++17. В ROS 2 рекомендуется любой узел писать как компоненту — примерно таким же образом, как в ROS 1 nodelet. Этот подход пришел на смену разделению узлов на ноды и нодлеты в ROS 1. Можно запускать несколько компонент из одного процесса или изолировать каждую в отдельном. В первом случае обеспечивается внутрипроцессное (intraprocess) взаимодействие, потенциально имеющее те же преимущества, что и нодлеты. Тем не менее, на 2016 год с производительностью intraprocess имелись серьезные проблемы [2], судя по более свежим обсуждениям в интернете, проблема всё ещё актуальна.

В ходе работы было принято решение отказаться от интеграции с ROS 2, так исследование показало, что пока у него нет никаких преимуществ по сравнению с ROS 1.

3. Архитектура

Обычно в построении ROS сетей каждый компонент робота: сенсор или актуатор — представлен полноценным узлом. Стоит заметить, что в условиях не очень больших вычислительных мощностей узлом приходится делать всего робота целиком. В случае ТРИК так и было сделано [10]: узел через библиотеку trikControl имеет доступ к периферии робота, получает таким образом информацию от сенсоров и публикует ее как участник сети ROS. Мастер-узел может быть запущен как на контроллере, так и на удаленном компьютере пользователя (1).

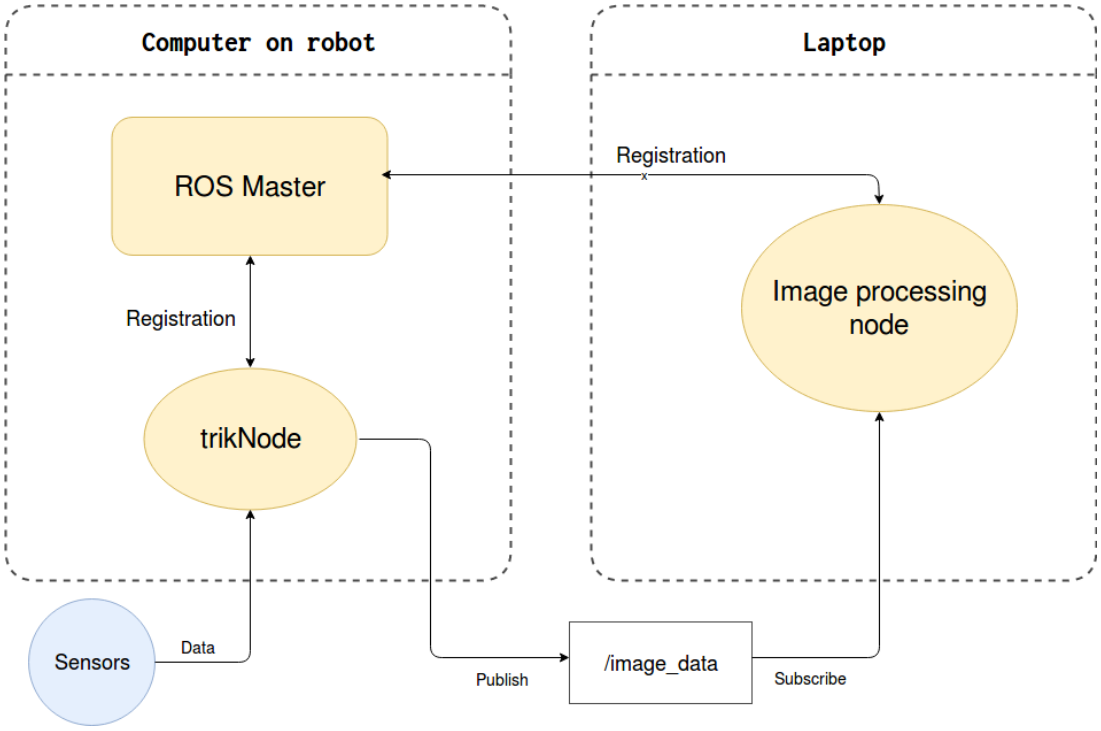

Рис. 1: Пример запущенной сети ROS

Сейчас архитектура подразумевает монолитный-блок – отдельный узел, который публикует информацию со всех датчиков и может управлять моторами. У этой архитектуры есть существенный недостаток: среда исполнения trikRuntime устроена таким образом, что в один момент времени может полноценно работать только один процесс, использующий библиотеку trikControl. Это так, потому что некоторые ресурсы неразделяемы, и процесс, который первый получил к ним доступ, блокирует его для других процессов. На контроллере по умолчанию

запущена графическая оболочка trikGui, использующая trikControl, поэтому часть ресурсов заблокирована всегда. Есть два пути реализации модуля ROS с доступом ко всем ресурсам.

Путь первый: встроить ROS в trikRuntime, то есть сделать всегда запущенную графическую оболочку trikGui узлом сети ROS (2). Это относительно несложно реализовать технически, но тогда в trikRuntime появится большое количество зависимостей. Сборка trikRuntime будет требовать установленный и настроенный ROS. Кроме того, для использования ROS необходима операционная система Linux, поэтому придется исключить ROS-компоненту для пользователей других операционных систем. Однако в описанном подходе есть существенный плюс: не нужно ни выбирать транспортный протокол, ни задумываться о формате описания сообщений и сериализации – все это уже есть в ROS.

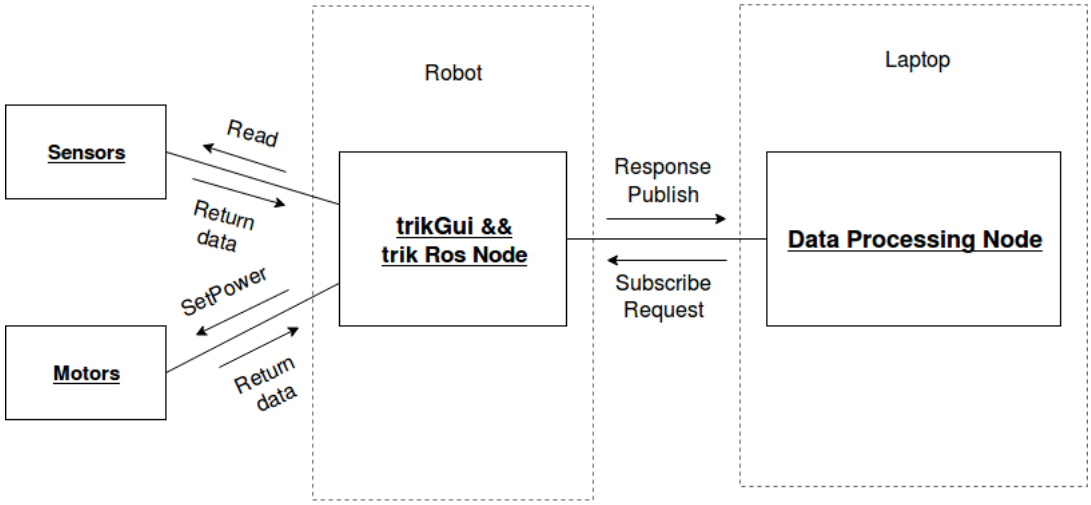

Рис. 2: Вариант новой архитектуры, ROS встроен в trikRuntime

Путь второй: отделить “сервер” с графической оболочкой, захватывающий ресурсы, и выставить интерфейс, который позволит всем другим процессам использовать эти ресурсы. В таком случае придется выбрать средство межпроцессного взаимодействия, которое позволит сообщаться этому “серверу” и другим процессам, в частности, узлу ROS. В курсовой работе [12] есть детальный обзор таких средств. За рамками курсовой была протестирована библиотека nanomsg, и оказалось, что эта библиотека намного лучше подходит для межпроцессного взаимо-

действия на контроллере, чем другие решения в обзоре. Узел сети ROS мог бы получать информацию от сервера с использованием, например, этой библиотеки, и архитектура (3) могла бы выглядеть примерно так:

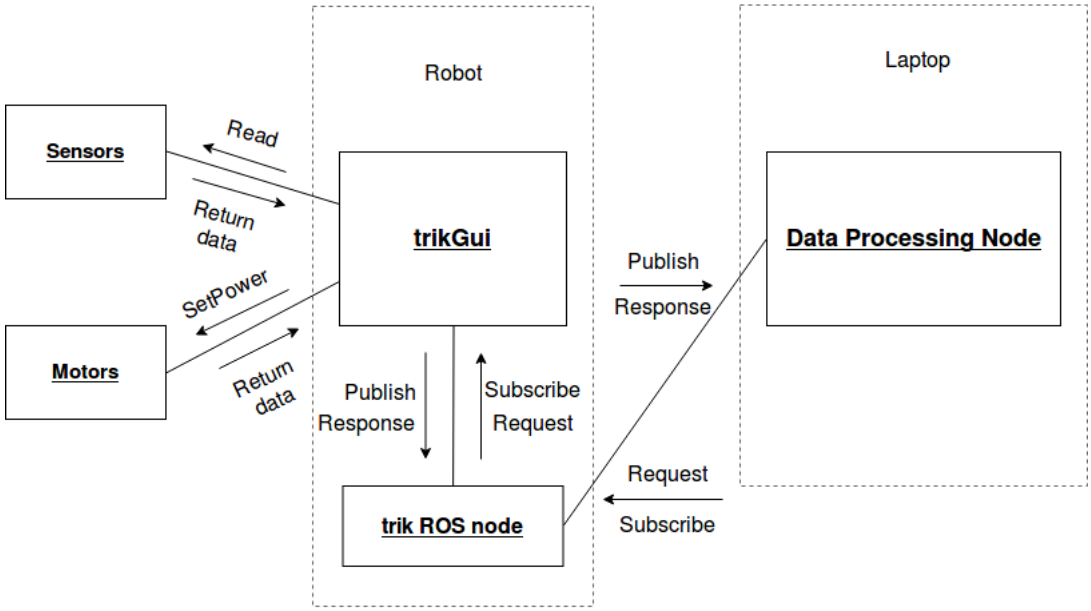

Рис. 3: Вариант новой архитектуры, узел ROS на контроллере

Беглая оценка производительности системы с запущенным на контроллере узлом ROS с помощи утилиты dstat показала, что при увеличении нагрузки, для имитации приближенной к реальной (около 100 сообщений от датчиков в секунду), время простоя (idle) процессора стремится к нулю. Поэтому ожидается, что ROS будет потреблять слишком много ресурсов на контроллере, и использовать его для обеспечения доступа к сенсорам и датчикам неоправданно дорого.

Видится логичным отказаться от использования ROS на контроллере вообще, а интегрировать контроллер с сетью ROS с помощью других средств. Тогда каждому контроллеру на компьютере пользователя можно будет сопоставить узел ROS, непосредственно связанный с контроллером. Такой узел будет получать информацию с периферии связанного с ним контроллера и дублировать его в сеть ROS для других узлов. Для конфигурирования робота и управления им во время запуска алгоритмов узел будет отправлять запросы на контроллер. При такой архитектуре (4) изначальная задача — использовать ROS для

прототипирования и быстрого тестирования алгоритмов — оказывается решенной, и при этом удается избавиться от проблемы блокирования ресурсов, уменьшить нагрузку на процессор контроллера.

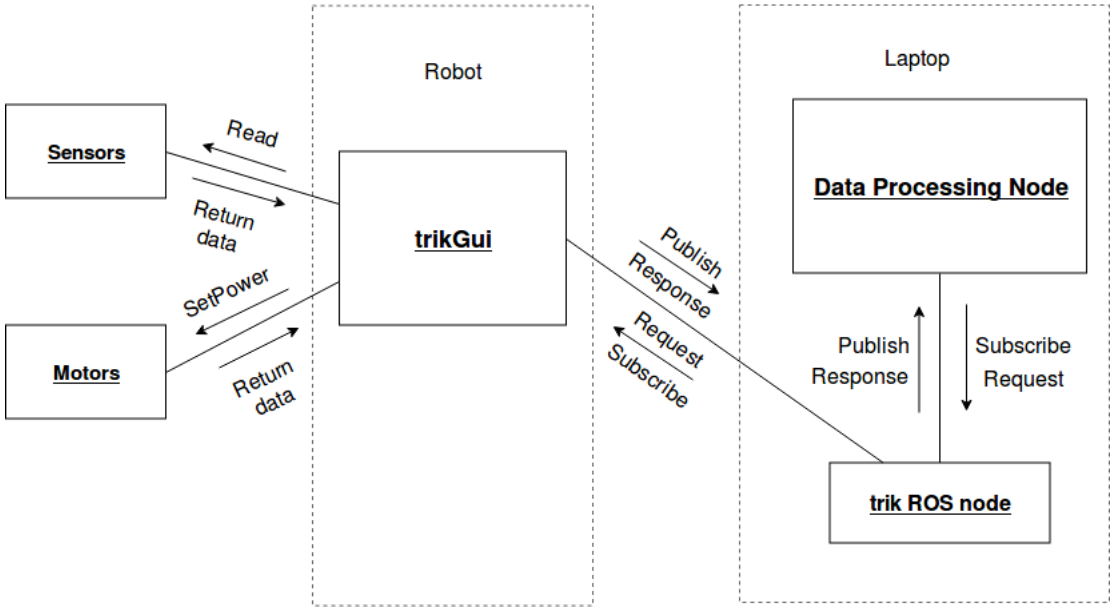

Рис. 4: Вариант новой архитектуры, без узла ROS на контроллере

Предложенная выше архитектура подразумевает использование двух паттернов: подписки на события – для получения потока данных с сенсоров, и удаленного вызова процедур (Remote Procedure Call, RPC) – для конфигурирования контроллера снаружи. Выбор решения определяется, прежде всего, наличием этих двух паттернов или легкостью их реализации. Кроме того, большое значение имеет пересылаемый по сети трафик, поэтому предпочтение отдается решениям, которые могут работать поверх UDP. Отправка сообщений с помощью UDP происходит намного быстрее, потому что сообщениям не требуется подтверждение. Потери, возникающие при использовании UDP, несущественны для большей части передаваемой контроллером информации – показаний датчиков. В редких случаях, когда гарантия доставки все-таки необходима, можно предусмотреть возможность использования TCP, либо обеспечить гарантию доставки на прикладном уровне с помощью использования различных уровней QoS. Кроме того, UDP позволяет использовать мультивещание – объединять получателей определенной

информации таким образом, что, сколько бы их ни было, пакет с информацией для них будет отправлен в сеть в единственном экземпляре.

Таким образом, появляется задача организовать сбор телеметрии для контроллера ТРИК, с использованием готовых библиотек и решений или без них. Задача сбора информации с датчиков, ее пересылки и визуализации на компьютере уже решалась в курсовых “Телеметрия роботов на базе контроллера TRIK” [15] и “Телеметрия моделей ТРИК” [13]. В этих работах не используются сторонние библиотеки, а лишь стандартные модули Qt для работы с TCP и UDP. Между тем, существуют более высокоуровневые протоколы и их реализации, удовлетворяющие требованиям задачи. С учетом вышесказанного, и того, что результаты указанных курсовых не были интегрированы с trikRuntime, было принято решение провести обзор существующих сторонних библиотек и фреймворков, релевантных задаче телеметрии.

Еще одним важным аспектом является сериализация. Многие библиотеки и фреймворки для отправки сообщений имеют свои механизмы сериализации или уже интегрированы с популярными протоколами, такими как CBOR. Возможно, понадобится выбрать конкретный протокол, например, Protobuf или MsgPack, и предусмотреть интеграцию с ним. Или же реализовать собственный формат упаковки сообщений, если это позволит выиграть в производительности. В работах по телеметрии [13], [15] используется базовая сериализация, представляемая средствами Qt с помощью класса QDataStream. Следует отметить, что при усложнении требований к типам сообщений она может оказаться неподходящей. Однако задача выбора протокола сериализации остается за рамками этой дипломной работы.

4. Реализация

Для интеграции контроллера ТРИК с ROS, требуется реализовать систему сбора телеметрии. С учетом требований, описанных в предыдущем разделе, а именно: наличие паттерна издатель-подписчик, RPC и работа поверх UDP, был проведен обзор протоколов, библиотек и фреймворков для сообщения.

4.1. nanomsg

Легковесная библиотека [1], предоставляющая абстракцию над сокетами; улучшенная версия другой популярной библиотеки сообщений – ZeroMQ. Nanomsg написана на языке программирования C и подходит для встроенных систем. Поддерживает множество паттернов взаимодействия, в том числе подписку на события (издатель-подписчик) и запрос-ответ, поверх которого легко реализовать RPC. Также позволяет использовать несколько транспортных механизмов, из которых для сетевого взаимодействия предназначены два: TCP и WS (вебсокеты поверх TCP).

4.2. gRPC

Фреймворк от Google для удалённого вызова процедур [9]. По умолчанию использует Protobuf в качестве инструмента сериализации и для описания типов данных. Использует HTTP/2, работающий поверх TCP, в качестве транспорта. Основная “идея” gRPC (как и большинства RPC вообще) базируется на предоставлении сервисов – методов, которые можно вызвать удаленно. В качестве ответа на запрос клиента может быть возвращен не только единственный результат, но и поток, из которого клиент сможет читать сообщения, пока они не закончатся. Этот механизм называется streaming RPC, и на его основе обычно реализуется паттерн подписки на события.

4.3. QtRemote

Модуль Qt, который позволяет разделить API и реализацию API между различными процессами и компьютерами. Подразумевает использование реплики – прокси к удаленному объекту – таким образом, что обращение к реплике выглядит так же, как обращение к реальному удаленному объекту, а все изменения в реальном объекте моментально отображаются в реплике. В отличие от обычного RPC, в котором ответу сервера обязательно предшествует запрос клиента, в QtRemote удаленный объект инициирует общение при любом изменении своего состояния. В качестве транспортного протокола используется TCP. Пока единственный поддерживаемый механизм сериализации – QDataStream, но разработчиками Qt обсуждается целесообразность добавления другой сериализации – CBOR или Protocol Buffers [5].

4.4. MQTT (Message Queuing Telemetry Transport)

Сетевой протокол для интернета вещей (IoT), ориентированный на обмен сообщениями между устройствами по принципу издатель-подписчик, основное предназначение которого — работа с телеметрией от различных датчиков. Является ISO стандартом. Существует огромное количество реализаций протокола, в том числе модуль Qt – Qt MQTT. Для работы необходим MQTT-брокер, к которому должны подключиться все устройства системы. Устройства регистрируют у брокера топики, информацию по которым хотят публиковать, и подписываются на топики, информацию по которым хотят получать. Топики имеют уровневую структуру; пример топика: sensors/digital/A1. Предусмотрена возможность хранения брокером самого актуального сообщения по топику – таким образом, вновь подписавшиеся на топик клиенты всегда будут иметь информацию сразу после подключения. Протокол работает поверх TCP и имеет 3 уровня QoS, а именно: “выстрелил и забыл” (fire-and-foget), “доставить хотя бы один раз” (delivered at least once) и “доставить ровно один раз” (delivered exactly once).

4.5. MQTT-SN (Message Queuing Telemetry Transport for Sensor Networks)

Еще один сетевой протокол для телеметрии. В отличие от MQTT работает поверх UDP и предназначается для систем, ограниченных в ресурсах и работающих в сетях с высокой потерей пакетов – таких системах, например, как WSN (Wireless Sensor Networks). WSN обычно состоит из автономных датчиков – вычислителей, крайне ограниченных в производительности и связанных необходимостью экономить заряд батареи. Для экономии трафика (имя топика входит в сообщение) в MQTT-SN есть предопределенные имена-id топиков и короткие двухсимвольные – например, A1, соответствующий sensors/digital/A1. MQTT-SN поддерживает те же QoS уровни, что и MQTT, и еще один дополнительный, который используется для отправки сообщений без установки соединения. Для использования MQTT-SN нужен MQTT-брокер и гейтвей – промежуточный сервер, преобразующий MQTT-SN сообщения в MQTT. MQTT-SN не очень популярен, имеет небольшое количество реализаций и не стандартизирован.

4.6. COAP (Constrained Application Protocol)

Специальный протокол, построенный с учетом ограничений архитектурного стиля REST. Спроектирован для легкой интеграции с HTTP и похож на него – клиенты получают доступ к данным сервера с помощью запросов GET, PUT, POST, и DELETE. Однако, в отличие от HTTP, COAP предназначен для ограниченных в ресурсах устройств, например, для сенсоров в WSN. Использует UDP в качестве транспортного протокола, и поддерживает два уровня QoS – для сообщений, требующих подтверждения, и нет. Таким образом, гарантия доставки и сбор датаграмм в нужном порядке обеспечиваются на прикладном, а не транспортном уровне. Стандартизирован IETF (Internet Engineering Task Force). Имеет своей основой паттерн запрос-ответ, а также позволяет наблюдать за состоянием сервера – этот механизм называет-

ся “observe”, и с его помощью COAP поддерживает паттерн подписки на события. COAP имеет значительно большее количество реализаций, чем MQTT.

4.7. Решение

Для интеграции контроллера с ROS необходимо организовать сбор телеметрии, и, поскольку одним из требованием была выдвинута возможность использовать UDP, в качестве кандидатов для решения были выбраны COAP и MQTT-SN.

По сравнительным оценкам производительности COAP и MQTT-SN, последний работает на 30% быстрее. [2]. Однако короткое знакомство с существующими реализациями MQTT-SN, в частности, с самой поддерживаемой из них – от Eclipse, показало, что библиотеки требуют доработки и исправления ошибок перед тем, как начать с ними работу. Еще одна реализация – от EMQ – была отвергнута сразу, так как она написана на языке программирования Erlang, что сильно затрудняет ее интеграцию с контроллером ТРИК. Кроме того, MQTT-SN не стандартизирован, а COAP стандартизирован IETF. Эти два обстоятельства определили выбор в его пользу.

Таким образом, был выбран протокол COAP и библиотека libcoap в качестве реализации этого протокола на языке программирования C. Библиотека libcoap может быть использована как для создания клиента, так и для сервера, и одинаково хорошо подходит и для встроенных устройств, и для обычных компьютеров.

В trikRuntime в библиотеку trikNetwork был добавлен класс trikCoapServer. Экземпляр этого класса создается в отдельном потоке в trikGui, и с помощью с ссылки на экземпляр Brick получает доступ к периферии робота. Возможности COAP позволяют сделать данные с периферии доступными снаружи для подписки, а также принимать некоторые данные извне и управлять с помощью них роботом или конфигурировать его – например, изменять мощность моторов. Также был реализован узел сети ROS, который одновременно является клиентом

COAP. Узел публикует полученную с сенсоров информацию в сеть ROS и получает команды от других узлов сети. Подобный узел, но пригодный для использования только на контроллере, уже был реализован в работе [11], при этом доступ к датчикам и моторам осуществлялся через вновь созданный экземпляр Brick. Теперь же обращение идет по протоколу COAP к контроллеру, к единожды созданному в trikGui экземпляру, и проблемы блокируемых ресурсов не возникает.

Для апробации получившейся системы был использован пакет ROS teleop_twist_keyboard, позволяющий управлять контроллером с помощью клавиатуры.

5. Заключение

В рамках работы были получены следующие результаты.

1. Проанализировано существующее решение интеграции ROS 1 с контроллером ТРИК и выявлены его недостатки.
2. Предложено новое решение, лишенное выявленных недостатков.
3. Проведен обзор существующих библиотек и фреймворки для обмена сообщениями и удаленного вызова процедур.
4. Проведен обзор протоколов обмена сообщениями IoT/M2M (MQTT, MQTT-SN, COAP) и их реализаций.
5. Реализован соответствующий улучшенной архитектуре модуль для взаимодействия контроллера с ROS 1.
6. Исследована возможность интеграции с ROS 2 и принято решение отказаться от неё.

Список литературы

[1] About Nanomsg. — Access mode: https://nanomsg.org/. (дата обращения: 8.05.2019).

[2] Maruyama Yuya, Kato Shinpei, Azumi Takuya. Exploring the Performance of ROS2 // EMSOFT '16 Proceedings of the 13th International Conference on Embedded Software, Article No. 5. — 2016.

[3] OpenEmbedded Layer for ROS Applications. — Access mode: https://github.com/bmwcarit/meta-ros. (дата обращения: 25.12.2018).

[4] Pardo-Castellote G. OMG Data-Distribution Service: Architectural Overview // Proc. of IEEE International Conference on Distributed Computing Systems Workshops. — 2003. — P. 200–206.

[5] QtCS2018 RemoteObjects. — Access mode: https://wiki.qt.io/QtCS2018_RemoteObjects. (дата обращения: 8.05.2019).

[6] ROS on DDS. — Access mode: https://design.ros2.org/articles/ros_on_dds.html. (дата обращения: 25.12.2018).

[7] ROS on ZeroMQ and Friends. — Access mode: https://design.ros2.org/articles/ros_with_zeromq.html. (дата обращения: 25.12.2018).

[8] ROS.org. — Access mode: http://www.ros.org/. (дата обращения: 25.12.2018).

[9] gRPC. — Access mode: https://grpc.io/. (дата обращения: 8.05.2019).

[10] trik-ros. — Access mode: https://github.com/auduchinok/trik-ros. (дата обращения: 25.12.2018).

[11] Евгений Аудучинок. Интеграция робототехнических библиотек ROS с контроллером ТРИК, СПбГУ. — 2016. — Access mode:

http://se.math.spbu.ru/SE/YearlyProjects/spring-2016/371/Auduchinok_report.pdf.

[12] Екатерина Балакина. Реализация межпроцессного взаимодействия на контроллере ТРИК, СПбГУ. — 2018. — Access mode: http://se.math.spbu.ru/SE/YearlyProjects/spring-2018/344/344-Balakina-report.pdf.

[13] Матвей Брыксин. Телеметрия моделей ТРИК. — 2014.

[14] Робототехнический контроллер ТРИК. — Access mode: http://www.trikset.com/. (дата обращения: 25.12.2018).

[15] Сергей Свитков. Телеметрия роботов на базе контроллера TRIK, СПбГУ. — 2016.