Санкт-Петербургский государственный университет

Кафедра системного программирования

Ковалев Марк Германович

Улучшение сетевого стека Embox для применения в устройствах, ориентированных на интернет вещей

Курсовая работа

Научный руководитель:
ассистент Козлов А. П.

Санкт-Петербург
2019

Оглавление

Введение

Развитие сетевых технологий в последние 40–50 лет подарило миру множество новых возможностей. И вместе с тем появились новые задачи и проблемы. Сейчас, к фундаментальным направлениям, таким как скорость передачи и гарантия доставки данных, добавилось обеспечение связи между разнородными устройствами.

Данная проблема не нова, но её значимость возросла с популяризацией среди обычных пользователей “умных вещей”. Умными называют вещи, обладающие вычислительными способностями для выполнения некоторых задач и имеющие возможность подключения к компьютерным сетям для взаимодействия друг с другом. И если раньше объединение множества различных ЭВМ в одну сеть тревожило в основном владельцев технологических предприятий и обширных офисов, сейчас к ним добавились производители устройств для умного дома и носимых аксессуаров. Образовался рынок интернета вещей, а вместе с тем возросли и требования к предлагаемым решениям.

Суть проблемы заключается в том, что существует система с множеством электронных устройств с различными архитектурами ЦП, способами передачи данных и техническими характеристиками. Все устройства в этой системе необходимо объединить в одну производительную сеть, удовлетворяющую многим требованиям, таким как: достаточная скорость передачи данных, доставка сообщений в рамках поставленных временных ограничений, защищённость трафика и т. д. Передача данных с помощью стека TCP/IP может удовлетворить часть этих требований, но его использование неэффективно в силу ограничений, накладываемых спецификой работы с интернетом вещей (огромное количество устройств с малыми ресурсами и пр.) [1]. Здесь необходимы более специализированные решения, протоколы и тонкая настройка всего оборудования и сети в целом.

В рамках данной курсовой работы проведено исследование области интернета вещей, обзор существующих решений и предложена реализация наиболее подходящих решений в операционной системе Embox.

1. Цель работы

Цель данной работы заключается в разработке специализированного ПО, нацеленного на оптимизацию сетевого стека открытого проекта Embox для работы с интернетом вещей. Для её достижения были сформулированы следующие задачи:

1. Изучить архитектуру проекта Embox.
2. Провести обзор технологий для интернета вещей.
3. Разработать архитектуру модуля, реализующего наиболее подходящее решение.
4. Реализовать модуль в Embox.
5. Провести тестирование работоспособности модуля.
6. Предложить примеры использования данного решения.

2. Embox

Embox — это конфигурируемая операционная система реального времени, спроектированная для работы во встраиваемых системах с ограниченными ресурсами [2].

Данная операционная система разрабатывается по модульному принципу. Все драйвера и пользовательские приложения являются отдельными модулями. Для управления ими была разработана система Mybuild. С её помощью можно выбрать только те компоненты системы, которые необходимы для решения конкретной задачи, создав таким образом минимальную прошивку, что необходимо для устройств с ограниченными ресурсами, используемых в интернете вещей. Также такая система модулей значительно упрощает разработку и внедрение в систему новых решений.

Статическая сборка системы обеспечивает дополнительную безопасность. Функциональность, которая не была включена в образ, не может быть исполнена.

В данный момент операционной системой Embox поддерживается множество архитектур, используемых в интернете вещей: ARM, x86, SPARC, Microblaze, MIPS, PPC, E2k. Архитектурно-зависимый код поставляется в отдельных модулях, что позволяет легко портировать систему на новые платформы.

Embox предоставляет слой POSIX-совместимости, что позволяет легко интегрировать Linux-приложения в систему.

В совокупности все эти свойства делают Embox отличной операционной системой для использования в области интернета вещей. Также для курсовой работы важно, что проект разрабатывается по принципу open source и есть возможность непосредственной связи с командой разработчиков.

3. Обзор существующих решений

3.1 MPLS

Multiprotocol Label Switching (MPLS) — механизм телекоммуникационной сети, основанный на передаче данных с помощью меток [3]. Когда пакет попадает в сеть MPLS, на “внешний роутер”, ему присваивается особая метка. Дальше происходит транспортировка по следующему алгоритму:

1. На “внутренний роутер” приходит пакет.
2. Роутер смотрит метку и по таблице коммутации находит следующий узел, которому надо передать пакет.
3. Метка заменяется на значение из таблицы.
4. Пакет передаётся на следующий узел.

На выходе из MPLS-сети метка снимается и пакет направляется дальше в зависимости от своего содержимого.

Преимущество такого подхода состоит в том, что все возможные пути в MPLS-сети уже известны и роутерам не надо тратить время на вычисление следующего узла, которому надо отправить данные. Таким образом, появляется возможность управлять трафиком. Например, пропускать важные пакеты по более широким каналам связи для обеспечения большей скорости доставки данных [4].

Недостатки MPLS-сети в стоимости и сложности её настройки. Высокая стоимость обусловлена дорогостоящими оборудованием и программными решениями. Также возникает проблема масштабируемости. При добавлении новых устройств в сеть, приходится всё перестраивать, что опять приводит к трате большого количества ресурсов.

MPLS является дорогой и сложной в использовании технологией, рассчитанной на компании и предприятия с обширной распределённой сетью. Поэтому данное решение не очень хорошо подходит для реализации в рамках данной курсовой работы.

3.2 SD-WAN

Software-Defined networking in a Wide Area Network (SD-WAN) стремится решить ту же проблему, что и MPLS: обеспечить крупные компании качественным сетевым соединением. Здесь происходит отделение программного обеспечения от аппаратного с последующей унификацией управления всей сетью в один контроллер [5]. Один раз настроив такую сеть, дальнейшее её сопровождение не вызывает таких трудностей, как MPLS.

Главными преимуществами такого подхода являются возможности динамического перестраивания путей для трафика, мониторинг всей сети и возможность быстрой смены конфигурации без необходимости физического контакта с конкретным оборудованием. К тому же использование SD-WAN позволяет компаниям экономить место, используемое для сетевого оборудования, потому что все вычисления переносятся в облако. Такое решение гораздо дешевле, чем MPLS, его проще внедрить и управлять сетью.

SD-WAN, как и MPLS, является комплексной и сложной технологией, направленной на улучшение сетевого взаимодействия в крупных распределенных компаниях. К тому же, хоть открытый стандарт и разрабатывается, доступен он будет только в 2019 году [6]. Поэтому такое решение также не применимо в рамках данной курсовой работы.

3.3 Связующее программное обеспечение, ориентированное на доставку сообщений.

Связующее программное обеспечение, ориентированное на доставку сообщений (Message-oriented middleware) — класс программного обеспечения, поддерживающий обмен сообщениями распределённых приложениях. Данные передаются как напрямую, так и с помощью очередей. Поддерживается как синхронная, так и асинхронная передача. Такое ПО гарантирует доставку сообщений, предоставляет сервисы по управлению, защите и администрированию [7].

Из этого класса можно выделить следующие технологии: CoAP, XMPP, AMQP и MQTT.

3.3.1 The Constrained Application Protocol (CoAP)

Протокол CoAP специализируется на использовании в среде с ограниченными узлами и ограниченной сетью. Спроектирован для машинного взаимодействия. Общение узлов происходит по REST-модели через HTTP: сервера предоставляют доступ к ресурсам по уникальным URL, клиенты обращаются к этим ресурсам с помощью запросов GET, PUT, POST, и DELETE [8].

3.3.2 Extensible Messaging and Presence Protocol (XMPP)

Протокол XMPP базируется на XML и предоставляет возможность практически в реальном времени обмениваться структурированными и легко расширяемыми данными между сетевыми узлами [9]. Изначально создавался для мгновенного обмена сообщениями (Instant messaging). Впоследствии нашлось больше применений в различных областях, в том числе и в интернете вещей. Здесь протокол используется благодаря поддержке различных способов соединения узлов (сокеты, BOSH, EXI и пр.) и шаблонов обмена сообщениями (запрос/ответ, издатель/подписчик, событийная подписка и пр.), масштабируемости и легковесности.

3.3.3 AMQP и MQTT

Advanced Message Queuing Protocol (AMQP) и Message Queuing Telemetry Transport (MQTT) — протоколы, основанные на шаблоне издатель/подписчик [10][11]. AMQP, по сравнению с MQTT, обладает большими возможностями: надежная организация очередей, гибкая маршрутизация, транзакции, повышенная защищённость и пр.

Протоколы CoAP, AMQP и MQTT — наиболее используемые в интернете вещей для передачи сообщений и управления [12]. Каждый из них обладает своими особенностями, преимуществами и недостатками. Протокол MQTT был выбран кандидатом на реализацию в рамках данной курсовой работы. Среди вышеперечисленных, это самое легковесное и простое решение, обладающее в то же время функциональностью, необходимой для реализации многих сценариев интернета вещей.

3.4 MQTT

Сеть MQTT состоит из сервера (брокера) и клиентов, которые могут быть издателями или подписчиками. Клиенты такой системы могут не знать о количестве других клиентов, их роли, местоположении и характеристиках. В то время как брокер знает о всех клиентах в такой сети. Издатель отправляет брокеру сообщение с определённой темой. Брокер пересылает сообщение всем подписчикам этой темы.

В стандартной реализации MQTT работает поверх протокола TCP. В совокупности с опциями Quality of Service это даёт надёжную гарантию доставки сообщений.

MQTT поддерживает следующие Quality of Service:

- At most once — сообщение отправляется лишь один раз и издатель с брокером не предпринимают никаких действий, чтобы убедиться в доставке.
- At least once — издатель посылает сообщение до тех пор, пока не придёт подтверждение получения.
- Exactly once — издатель и подписчик заключают двойное рукопожатие, чтобы убедиться, что доставлена только одна копия сообщения.

3.5 MQTT/UDP

MQTT/UDP — реализация протокола MQTT, работающая на UDP бродкасте [13]. В системе, построенной на данном протоколе, за ненадобностью исключается брокер, ведь благодаря UDP бродкасту все сообщения отсылаются всем участникам сети. Это ещё сильнее упрощает протокол MQTT, однако делая его более уязвимым для атак на сеть.

В стандартной реализации MQTT брокер является “single point of failure”, что означает выход из строя всей системы при отказе или неправильной работе брокера. В системе MQTT/UDP все узлы посылают сообщения всем и их возможность функционировать не зависит друг от друга.

Также в случае, когда требуется постоянное обновление данных, например с термостата, UDP справляется лучше, чем TCP. Если узлы будут

отбрасывать каждый второй пакет, TCP будет пытаться посылать уже устаревшие данные, в то время как при передаче по UDP, половина пакетов всё же доставится и при этом сохранится актуальность данных.

В случае же, если нужна гарантированная доставка сообщения, MQTT/UDP также предоставляет настройку Quality of service.

После изучения, данный протокол был выбран для реализации в операционной системе Embox вместо стандартного MQTT как более перспективное решение, применимое в сценариях с большим количеством устройств и нагруженной сетью.

4. Реализация

Для реализации протокола MQTT/UDP в Embox портирована библиотека libmqttudp, предоставляемая автором протокола [14].

Структура решения в Embox:

- scripts/mqtt_udp
  - auto_qemu, start_script и stop_script — скрипты для запуска конфигурации в эмуляторе QEMU
  - up_bridge.sh и down_bidge.sh — настройка моста для возможности объединения виртуальных машин в одну подсеть
- src/cmds/testing/mqtt_udp
  - client.c и server.c — программы, реализующие логику клиента и сервера соответственно, используя методы библиотеки libmqttudp
  - Mybuild — описание модулей клиента и сервера, а также зависимостей, необходимых для корректной сборки и работы
- templates/x86/test/mqtt
  - mods.config — описание подключаемых для сборки системы модулей. Модули, составляющие данную реализацию:

    third_party.mqtt_udp.core
    embox.cmd.testing.mqtt_udp.server
    embox.cmd.testing.mqtt_udp.client
  - rootfs/network — конфигурация сетевого интерфейса (ip, mac_address, default gateway)
- third-party/mqtt_udp
  - Makefile — описание источника для загрузки исходных файлов, опции сборки, инструкции установки библиотеки libmqttudp
  - Mybuild — описание модуля библиотеки
  - patch.txt — патч в формате diff, применяемый к исходному коду решения в целях решения проблем с совместимостью

Процесс сборки и запуска Embox с данным решением выглядит следующим образом:

1. make confload-x86/test/mqtt — загружает необходимую конфигурацию в директорию conf;
2. make — инициализирует процесс компиляции исходного кода, в рамках которого работает система сборки Mybuild. Из файла conf/mods.config она узнаёт, какие модули необходимо объединить в финальный образ системы. Здесь запускается Makefile из third-party/mqtt_udp, который загружает исходный код, применяет к нему patch.txt и компилирует библиотеку libmqttudp.a;
3. ./scripts/mqtt/up_bridge.sh — запуск и настройка моста, который будет связывать tap-интерфейсы виртуальных машин QEMU;
4. ./scripts/mqtt/auto_qemu — запуск виртуальной машины с Embox.

Решение доступно в Embox в качестве консольных утилит mqtt_udp_server и mqtt_udp_client:

mqtt_udp_server — регистрирует все получаемые MQTT/UDP пакеты и выводит данные в консоль.

mqtt_udp_client — каждую секунду посылает MQTT/UDP пакет с данными.

5. Тестирование

Модель для тестирования построена на сети из виртуальных машин QEMU, на которых установлены подготовленные образы Embox. Модель запускается с Linux-хоста, объединение машин в общую сеть происходит за счёт подключения виртуальных сетевых интерфейсов каждой машины (tapX) к интерфейсу моста (br0).

Модели тестирования MQTT/UDP представлена на рисунке 1.

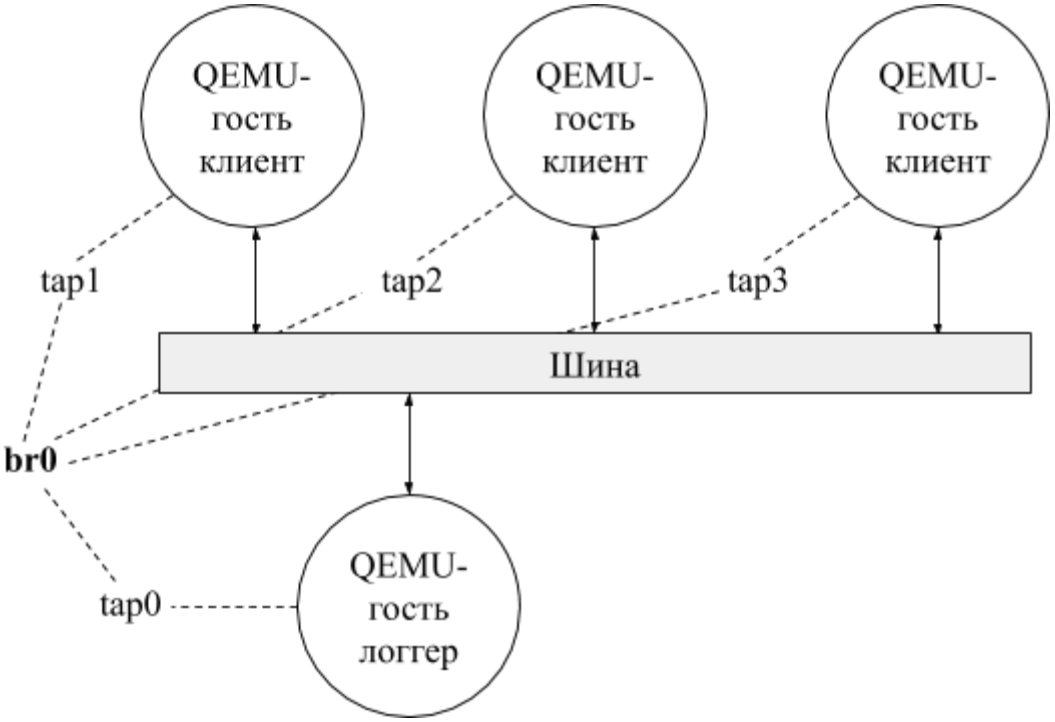

Рис. 1: Модель тестирования MQTT/UDP

Здесь один из клиентов выполняет функцию логирования. Он выводит в консоль любой полученный MQTT/UDP пакет. Это возможно в силу того, что благодаря UDP-бродкасту все сообщения в сети видны всем узлам. Образуется шина, которую и прослушивает один из клиентов.

Пакеты с каждого клиента индексируются. Таким образом можно отследить потери в сети.

Пример работы системы с тремя клиентами и одним сервером:

```
root@embox:/#mqtt_udp_server
pkt PUBLISHid 0 from 10.0.2.20 topic '/temp' = '39'
pkt PUBLISHid 1 from 10.0.2.20 topic '/temp' = '38'
pkt PUBLISHid 0 from 10.0.2.21 topic '/temp' = '39'
pkt PUBLISHid 2 from 10.0.2.20 topic '/temp' = '37'
pkt PUBLISHid 1 from 10.0.2.21 topic '/temp' = '38'
pkt PUBLISHid 3 from 10.0.2.20 topic '/temp' = '36'
pkt PUBLISHid 2 from 10.0.2.21 topic '/temp' = '37'
pkt PUBLISHid 0 from 10.0.2.22 topic '/temp' = '39'
pkt PUBLISHid 4 from 10.0.2.20 topic '/temp' = '35'
pkt PUBLISHid 3 from 10.0.2.21 topic '/temp' = '36'
pkt PUBLISHid 1 from 10.0.2.22 topic '/temp' = '38'
pkt PUBLISHid 4 from 10.0.2.21 topic '/temp' = '35'
pkt PUBLISHid 2 from 10.0.2.22 topic '/temp' = '37'
pkt PUBLISHid 3 from 10.0.2.22 topic '/temp' = '36'
pkt PUBLISHid 4 from 10.0.2.22 topic '/temp' = '35'
```

Здесь видны сообщения с трёх IP-адресов: 10.0.2.20, 10.0.2.21 и 10.0.2.22. Это адреса клиентов. Значение id показывает, какой по номеру пакет отсылает этот клиент. Из этого примера видно, что индексация для каждого IP-адреса не нарушилась, что означает отсутствие потери пакетов.

Таким образом, портированная в Embox библиотека, реализующая протокол MQTT/UDP, функционирует.

6. Примеры использования

MQTT/UDP подойдёт для системы с большим количеством устройств и постоянной нагрузкой на сеть. Самый базовый пример — умный дом, сеть которого защищена файрволом. Здесь данные не так важны и их потеря не несёт ущерба для работы всей системы. Например, для датчика температуры, делающего замеры каждую минуту, потеря даже половины пакетов незначительна. К тому же с современным оборудованием потеря UDP пакетов в локальной сети маловероятна.

Также не возникает проблема установки отказоустойчивого брокера, выход из строя которого привел бы к остановке работы всей MQTT системы (как и любой другой). В то время как MQTT/UDP будет работать, пока в сети будут оставаться узлы.

Заключение

В ходе работы получены следующие результаты:

1. Изучена архитектура проекта Embox.
2. Проведён обзор существующих технологий для интернета вещей.
3. В Embox портирована библиотека, реализующая протокол MQTT/UDP.
4. Проведено тестирование работоспособности модуля.
5. Предложены примеры использования.

Список литературы

[1] Wentao Shang, Yingdi Yu, Ralph Droms, Lixia Zhang. Challenges in IoT Networking via TCP/IP Architecture. — 2016 — URL: https://named-data.net/publications/techreports/ndn-0038-1-challenges-iot/ (дата обращения: 12.05.2019)

[2] Embox wiki. — URL: https://github.com/embox/embox/wiki (дата обращения: 12.05.2019)

[3] RFC 3031. Multiprotocol Label Switching Architecture — 2001 — URL: https://tools.ietf.org/html/rfc3031 (дата обращения: 12.05.2019)

[4] Neal Weinberg and Johna Till Johnson. MPLS explained. — 2018 — URL: https://www.networkworld.com/article/2297171/network-security-mpls-explained.html (дата обращения: 12.05.2019)

[5] Jessica Scarpati. SD-What? Understanding SD-WAN. — 2015 — URL: https://searchnetworking.techtarget.com/feature/SD-What-Understanding-SD-WAN (дата обращения: 12.05.2019)

[6] MEF Announces Industry's First SD-WAN Service Specification. — 2018 — URL: https://www.mef.net/Press-Releases/MEF-Announces-Industrys-First-SD-WAN-Service-Specification (дата обращения: 12.05.2019)

[7] University of California at Berkeley. Message-Oriented Middleware. What is it? — URL: http://www2.sims.berkeley.edu/courses/is206/f97/GroupB/mom/ (дата обращения: 12.05.2019)

[8] RFC 7252. The Constrained Application Protocol (CoAP) — 2014 — URL: https://tools.ietf.org/html/rfc7252 (дата обращения: 12.05.2019)

[9] RFC 6120. Extensible Messaging and Presence Protocol (XMPP): Core — 2011 — URL: https://tools.ietf.org/html/rfc6120 (дата обращения: 12.05.2019)

[10] OASIS Standard. OASIS Advanced Message Queuing Protocol(AMQP) Version 1.0 — 2012 — URL: http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf (дата обращения: 12.05.2019)

[11] OASIS Standard. MQTT Version 5.0 — 2019 — URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.pdf (дата обращения: 12.05.2019)

[12] Nitin Naik. Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP — 2017 — URL: https://ieeexplore.ieee.org/document/8088251 (дата обращения: 12.05.2019)

[13] Dmitry Zavalishin. Welcome to MQTT/UDP. — URL: https://mqtt-udp.readthedocs.io/en/latest/ (дата обращения: 12.05.2019)

[14] mqtt_udp. Репозиторий GitHub. — URL: https://github.com/dzavalishin/mqtt_udp (дата обращения: 12.05.2019)