

Санкт-Петербургский государственный университет

Тагильцев Семен Вадимович

Выпускная квалификационная работа

Улучшение сетевых возможностей ОСРВ Embox

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2020 «Технологии программирования»*

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н. Д. В. Луцев

Консультант:
генеральный директор ООО «Ембокс» А. В. Бондарев

Рецензент:
старший преподаватель кафедры ИАС К.К. Смирнов

Санкт-Петербург
2024

Saint Petersburg State University

Semyon Tagiltsev

Bachelor's Thesis

Improvement of RTOS Embox network ability

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2020 "Programming Technologies"*

Scientific supervisor:
C.Sc., docent D.V. Lutsiv

Consultant:
CEO at "Embox ltd." A.V. Bondarev

Reviewer:
senior lecturer AIS chair K.K. Smirnov

Saint Petersburg
2024

Оглавление

1. Введение	4
2. Постановка задачи	6
3. Обзор	7
3.1. Сетевые возможности Embox	7
3.2. Современные подходы к построению сетевой инфраструк- туры	8
4. Архитектура	14
5. Реализация	16
6. Тестирование	22
7. Заключение	25
Список литературы	26

1. Введение

Сетевые технологии сегодня играют ключевую роль в повседневной жизни человека, обеспечивая связь и взаимодействие между людьми, компьютерами и устройствами в реальном времени. Сетевые технологии включают в себя различные протоколы, стандарты и устройства, направленные на оптимизацию эффективности сетей, обеспечение безопасности и надежности передачи данных.

Вместе с постоянным ростом объема передаваемой информации развивается и сетевая инфраструктура. Появляются новые сетевые протоколы и устройства, а также подходы к построению сетей. Примером, который активно набирает популярность у сетевых вендеров, является SDN [8]. Программно-определяемая сеть¹ (Software-defined networking, SDN; также программно-конфигурируемая сеть) – подход к построению и управлению сетями, в котором уровень управления сетью (control plane) реализуется программно и находится отдельно от уровня передачи данных (data plane).

Корпоративные сети являются важной отраслью сетевых технологий и предназначены для успешного функционирования современных компаний. Построение локальных корпоративных сетевых инфраструктур является приоритетным направлением для организаций, которые стремятся защитить конфиденциальность своей информации, а также иметь полный контроль над администрированием. В наше время, когда ушло большое количество сетевых вендеров и идет активное импортозамещение, российские компании все чаще и чаще обращаются к отечественным технологиям, в том числе и в области построения инфраструктуры корпоративных сетей.

Одним из таких отечественных решений является ОС Embox², которая долгое время разрабатывалась на кафедре системного программирования СПбГУ. ОС Embox представляет собой открытую платформу с уклоном в минимализм и гибкость, и она может стать хорошим ин-

¹Software-Defined Networking (SDN): https://en.wikipedia.org/wiki/Software-defined_networking

²Embox: <http://emboxing.ru/>

струментом для российских компаний в области сетевых технологий. Несомненным достоинством данной ОС является небольшая кодовая база, а также то, что данная ОС имеет развитый сетевой стек и уже запускалась на сетевом оборудовании для построения традиционных сетей российскими компаниями.

Исходя из всего выше сказанного, одним из дальнейших перспективных направлений развития сетевых возможностей в ОС Embox и целью данной работы ставится обеспечить начальную поддержку технологий для построения SDN, не претендуя, однако, на полную поддержку SDN в связи с объёмами необходимой для этого работы.

2. Постановка задачи

Целью работы является реализация начальных сетевых возможностей в ОС Embox для поддержки программно-определяемых сетей (Software-defined networking, SDN).

Для достижения данной цели были поставлены следующие задачи.

1. Выполнить обзор текущих сетевых возможностей ОС Embox, современных подходов к построению сетевой инфраструктуры и выделить основную функциональность для SDN.
2. Спроектировать решение, обеспечивающее базовую поддержку SDN в ОС Embox.
3. Улучшить текущие сетевые возможности ОС Embox, реализовав спроектированное решение.
4. Выполнить тестирование реализованных компонент.

3. Обзор

3.1. Сетевые возможности Embox

Embox является кроссплатформенной операционной системой, в основе которой лежит принцип модульности. Возможность конфигурирования конечного образа Embox позволяет добиться низких требований к аппаратным ресурсам. Появляется возможность создания ОС нацеленной на решение конкретных задач.

В ОС Embox реализовано большое количество драйверов для различных сетевых контроллеров для наиболее известных архитектур. В частности, в ОС поддерживаются такие сетевые карты как e1000, lan9118, lan91c111, rtl8139.

Также в Embox широко развит сетевой стек, который включает в себя различные протоколы. Так есть поддержка протоколов для разных уровней сетевой модели OSI таких, как ethernet, ipv4, ipv6, arp, icmp, ndp, tcp, udp и другие.

Архитектура сетевой подсистемы ОС Embox похожа на аналогичную в ОС Linux и включает в себя стандартный интерфейс Berkly Sockets, модель сетевых устройств, таблицу интерфейсов, таблицу маршрутизации и так далее.

Присутствует и различные утилиты для управления сетевым трафиком, поддержка межсетевых экранов, а также сетевые интерфейсы.

Все выше перечисленное позволяет использовать ОС Embox в сфере Интернет вещей (IoT) и различных сетевых устройствах. Несмотря на то, что сетевые возможности достаточно развиты, в ОС отсутствует поддержка программно-конфигурируемых сетей (SDN), хотя и есть все предпосылки для этого.

3.2. Современные подходы к построению сетевой инфраструктуры

3.2.1. Традиционная сеть

В начале эры компьютерных технологий традиционные сети, основанные на классических протоколах, казались достаточным решением для обмена информацией. Однако с течением времени и стремительным ростом зависимости от сетевых технологий, стали выявляться ряд серьезных проблем, которые существенно затрудняли эффективное построение и управление инфраструктурой.

Одной из основных проблем традиционных сетей стало ограничение в гибкости и масштабируемости. В мире, где объемы данных растут в геометрической прогрессии, сети сталкиваются с необходимостью обеспечения высокой пропускной способности и мгновенной реакции на изменяющиеся требования бизнеса. Ручная настройка каждого сетевого устройства, сложные правила маршрутизации и ограниченные возможности адаптации стали препятствием для обеспечения эффективности и гибкости сетевой инфраструктуры.

Еще одной серьезной проблемой является сложность управления традиционными сетями. Распределенные механизмы принятия решений в каждом сетевом устройстве приводили к тому, что внедрение новых политик безопасности, отслеживание сетевых событий и мониторинг становились непростыми задачами. Отсутствие централизованного контроля затрудняло быстрые изменения и адаптацию к современным требованиям.

Следующей проблемой традиционных сетей была безопасность. С появлением новых угроз и методов атак, классические сети становились уязвимыми, так как большинство защитных механизмов оказывались разбросанными по различным устройствам, что делало сложным обеспечение единой политики безопасности.

Этот набор проблем вынудил индустрию искать инновационные решения. Так появилась концепция программно-определяемых сетей.

На Рис. 1 представлено главное отличие традиционных сетей от

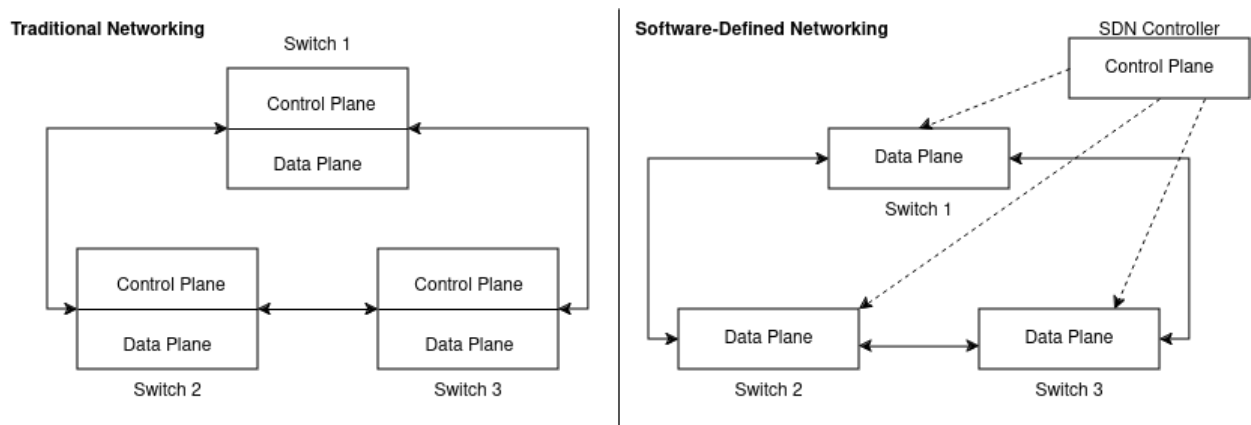


Рис. 1: Традиционная сеть vs SDN

программно-определяемых, заключающееся в отделении уровня управления сетью (control plane) от уровня передачи данных (data plane).

3.2.2. Software-Defined Networking (SDN)

Как уже упоминалось ранее, SDN (Software-Defined Networking) является подходом к дизайну и управлению сетями, в котором управление и конфигурация сетевого оборудования разделяются от передачи данных. Вместо того, чтобы иметь жестко заданные и настроенные сетевые устройства, SDN предоставляет централизованное управление, которое осуществляется программным обеспечением, обычно через SDN-контроллер. Основная идея заключается в том, чтобы сделать сеть более гибкой, масштабируемой и легкой в управлении.

Основные составляющие SDN перечислены ниже.

- SDN Controller — централизованное программное управление сетью, обеспечивающее принятие решений по маршрутизации и управлению трафиком.
- Southbound APIs — интерфейсы связи между SDN-контроллером и устройствами передачи данных, такие как OpenFlow, которые позволяют программному обеспечению на SDN-контроллере управлять потоками данных на устройствах.
- Northbound APIs — интерфейсы для взаимодействия между SDN-контроллером и прикладными программами или приложениями

высокого уровня, позволяя им использовать возможности SDN.

- SDN Application — набор программ, работающих поверх SDN-контроллера и использующих его возможности для реализации конкретных сетевых функций, предназначенные для решения различных задач, таких как мониторинг сети, обеспечение безопасности и т.д.

Все перечисленные компоненты вместе взятые изображены на Рис.2.

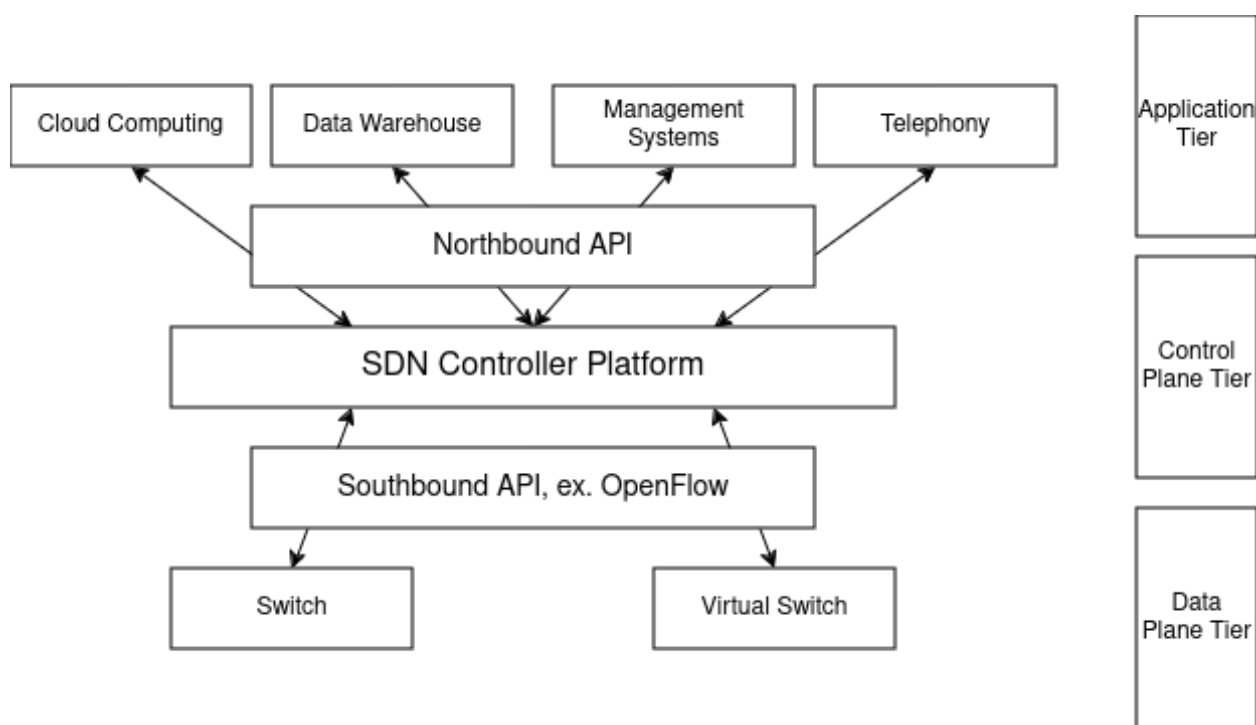


Рис. 2: SDN инфраструктура

Стоит заметить из Рис. 2, что на ранних этапах развития SDN Southbound APIs был жестко ограничен одним протоколом – OpenFlow, который и по сей день является де-факто стандартом. Но по мере развития технологий и сфер использования программно-определяемых сетей, различные вендоры стали разрабатывать новые протоколы для решения своих проблем. Так, например, появились такие протоколы, как OpFlex [1] от компании Cisco³ и Protocol Oblivious Forwarding (POF) [4] от Huawei⁴.

³Cisco: <https://www.cisco.com/>

⁴Huawei: <https://www.huawei.ru/w>

Наряду с созданием новых протоколов, в 2014 году группой исследователей и разработчиков был описан абсолютно новый проект P4 (Programming Protocol-Independent Packet Processors), о котором пойдет речь в следующей главе. [2]

3.2.3. Проект P4

P4 (Programming Protocol-Independent Packet Processors)⁵ — это язык программирования, созданный для описания протоколов обработки пакетов на сетевых устройствах, таких как коммутаторы (switches) и маршрутизаторы (routers).

Основными идеями, которые лежат в основе P4, являются независимость от целевой реализации, используемых протоколов и реконфигурируемость полей.

Иначе говоря, программы, написанные на языке программирования P4, могут быть скомпилированы под различные типы исполнительных машин, такие как ASIC, FPGA, процессоры общего назначения и др. В P4 нет встроенной поддержки сетевых протоколов. Вместо этого разработчик в своей программе описывает необходимые ему заголовки и поля, а также правила обработки, пересылки и изменения пакетов. Вместе с этим, язык программирования P4 дает возможность, в отличие от реализации конкретного протокола, изменять логику работы с пакетами в уже запущенной системе.

Также несомненным плюсом P4 выступает то, что это проект с открытым исходным кодом, который полностью расположен на GitHub⁶.

Разберем основные части проекта P4, к которым относятся:

- p4c — модульный компилятор для языка программирования P4, состоящий из фронтендов, миддлэнда и бэкендов. Фронтенд выполняет преобразование исходного P4-кода в промежуточное представление (IR) и имеет две реализации P4_14 и P4_16. Миддлэнд включает в себя оптимизации и преобразования промежуточ-

⁵P4: <https://p4.org/>

⁶P4lang: <https://github.com/p4lang>

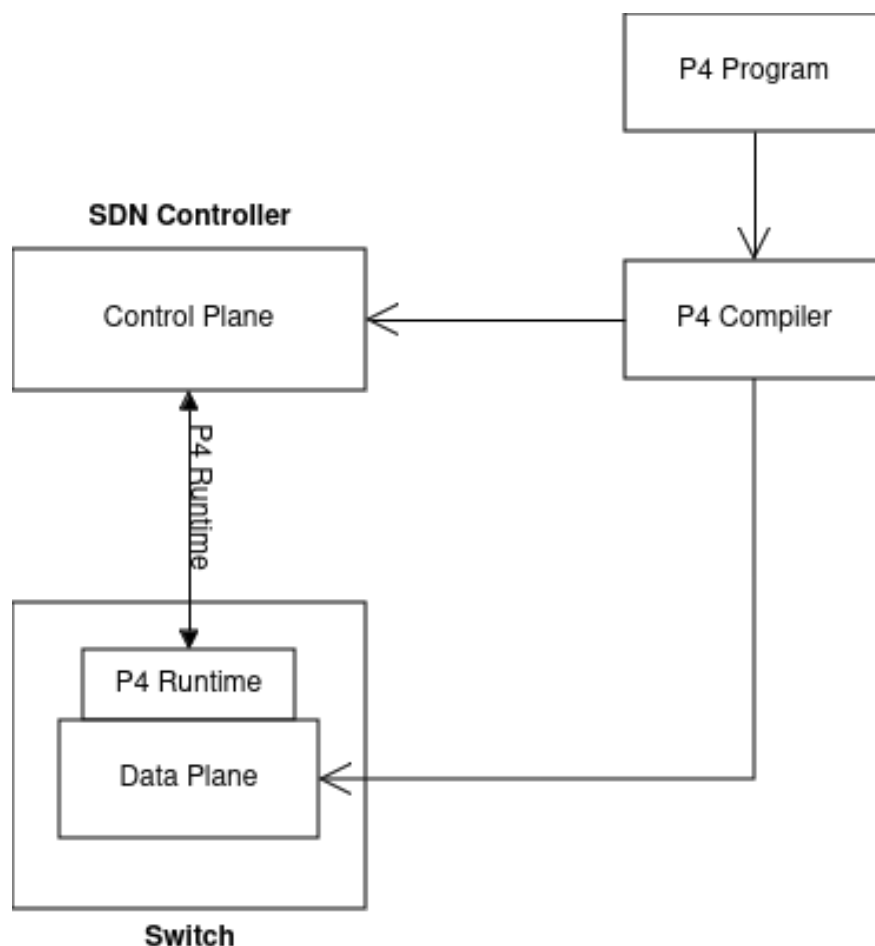


Рис. 3: Устройство проект P4 на примере коммутатора

ного представления. В конечном итоге, бэкенд генерирует код для целевой платформы.

- **PI (Protocol Independent)** — содержит реализацию абстрактного интерфейса для взаимодействия между P4-программируемыми сетевыми устройствами и SDN-контроллерами. Имеет клиент-серверную архитектуру.
- **behavioral-model (bmv2)** — включает в себя несколько реализаций программных коммутаторов, которые принимают на вход JSON-файлы, полученные при помощи компилирования P4-программы компилятором и интерпретирует его для реализации поведение обработки пакетов, заданными этими программами.

Одними из основных назначений bmv2 является отладка и тести-

рования P4-программ на эмулированных сетевых устройствах. Также bmv2 может быть доработана и запущена на реальном сетевом оборудовании для прототипирования и демонстрации SDN сетей.

Учитывая все выше сказанное, P4, в отличие от реализации конкретных протоколов, является возможным первым шагом на пути к SDN в ОС Embox.

4. Архитектура

В данной главе представлено описание архитектуры решения, позволяющего улучшить сетевые возможности ОС Embox.

Решением является портирование одной из частей проекта P4 в Embox. Это позволит получить представление работоспособного коммутатора на основе ОС Embox, который поддерживает программы написанные и скомпилированные при помощи проекта P4. Также в Embox появится несколько сетевых библиотек, которые необходимы как для интегрирования других частей проекта P4, так и для написания пользовательских приложений.

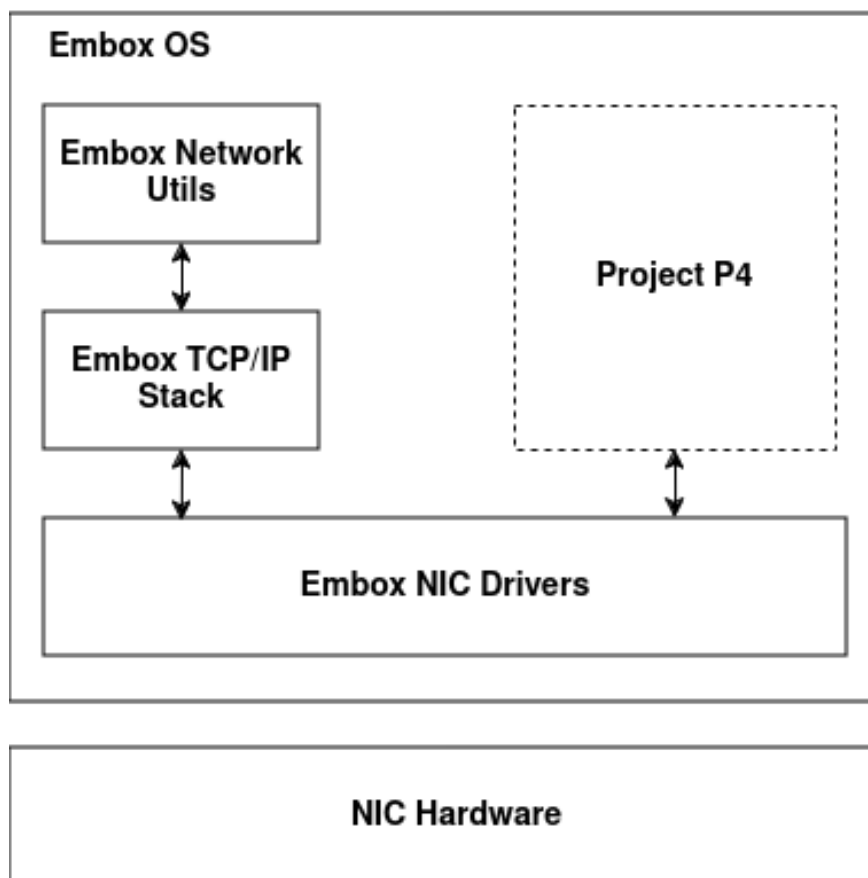


Рис. 4: ОС Embox и проект P4

После портирование в Embox всех необходимых зависимостей вместе с программным коммутатором P4, получаем ОС с виртуальным свитчем. На основе таких свитчей можно строить программно-конфигурируемые сети. В последствие может быть использован не

только для тестирования и разработки программ, но и для дальнейшего портирование на реальное оборудование. При этом для его настройки будут использоваться программы написанные на языке программирования P4 и скомпилированные под среду bmv2.

5. Реализация

Добавить поддержку проекта P4 в ОС Embox целиком является большой задачей. Поэтому логично это делать по частям и начать с передающего уровня (data plane), а именно с программного коммутатора P4 [3]. У него есть несколько основных реализаций: `simple_switch`, `simple_switch_grpc`, `psa_switch`, `simple_router` и `l2_switch` [9]. В связи с тем, что на начальном этапе поддержки P4 в ОС Embox должны получить минимальную рабочую версию, выберем наиболее простую реализацию из них – `l2_switch`.

Добавление внешнего проекта в ОС Embox происходит в несколько шагов.

1. Выявление зависимостей у проекта для портирование.
2. Конфигурирование проекта, отключение лишней функциональности.
3. Локальная кроссплатформенная сборка проекта.
4. Создание конфигурационных файлов для подключения сборки проекта в составе Embox.
5. Отладка сборки ОС Embox.

Прежде чем портировать выбранный коммутатор необходимо поддерживать его зависимости. Основными зависимостями, которые отсутствуют в Embox, являются библиотеки `thrift` [10], `nanomsg` [12], `boost` [11], `pcap` [7] и `gmp` [6]. Стоит отметить, что `boost` уже есть в Embox, но давно не обновлялся. Также, в отличие от остальных библиотек, `thrift` имеет свою зависимость в виде библиотеки `openssl` [13].

Все перечисленные зависимости вместе взятые изображены на Рис.5.

Рассмотрим интегрирование проекта P4, а точнее части с программируемым коммутатором `l2_switch`, в ОС Embox более детально.

Изначально необходимо было ознакомиться с библиотекой на локальной машине. Скачать нужную версию, затем сконфигурировать

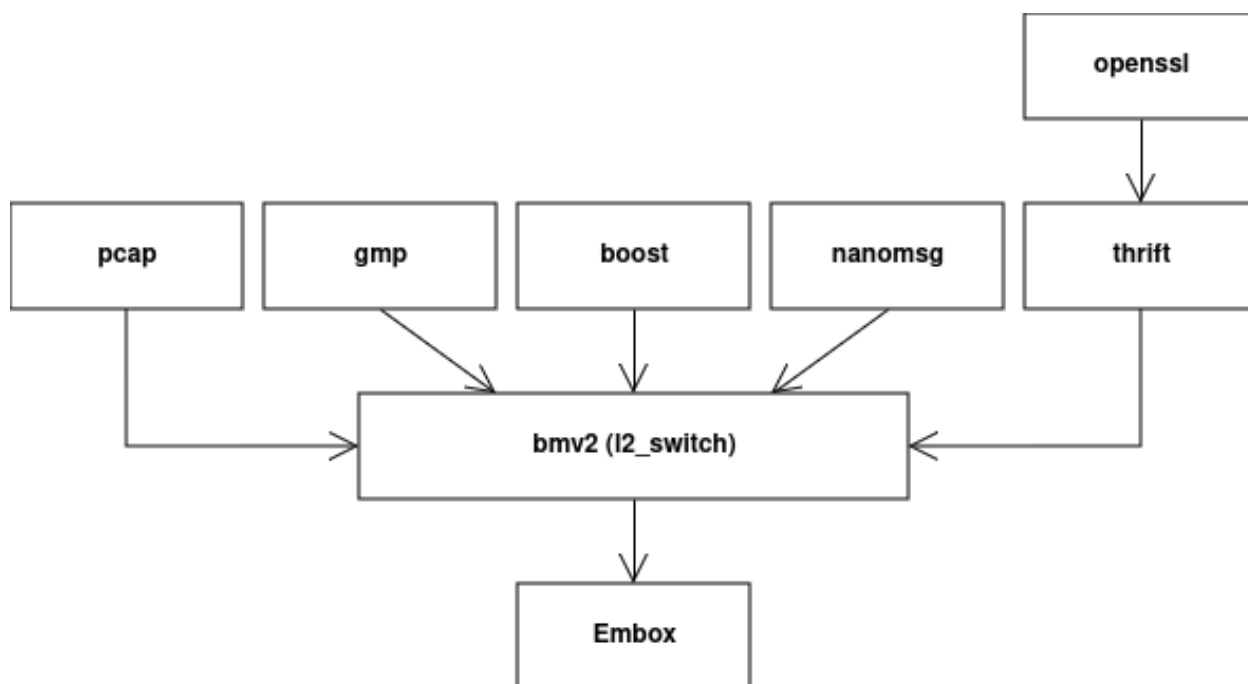


Рис. 5: Схема зависимостей

проект так, чтобы обязательно было включена статическая линковка и кроссплатформенная сборка. Также при конфигурировании необходимо отключить лишнюю функциональность, сделать минимальную рабочую версию проекта. После чего выполнить кроссплатформенную сборку.

Получив положительный результат при локальной сборке проекта, можно переходить к его интегрированию в ОС Embox. Обычно внешние библиотеки располагаются в директории `third-party/lib`, поэтому заведем в ней еще одну папку, где будут располагаться конфигурационные файлы для системы сборки Mybuild.

Сначала добавим файл описания модуля интегрируемого проекта. В нем указываются имя модуля, этап сборки ОС Embox на котором необходим данный проект. Также здесь присутствуют список зависимостей вместе с конечным исходным файлом, который используется для линковки с конечным образом ОС.

Листинг 1: Mybuild файл для проекта P4

```
package third_party.cmd
```

```

@App
@AutoCmd
@BuildDepends(embox.lib.libsupcxx)
@BuildDepends(embox.lib.libstdcxx)
@BuildDepends(embox.lib.libgcc)
@BuildDepends(third_party.boost.libs)
@BuildDepends(third_party.lib.libpcap)
@BuildDepends(third_party.lib.libgmp)
@BuildDepends(third_party.lib.openssl)
@BuildDepends(third_party.lib.thrift)
@Build(stage=2,script="(EXTERNAL_MAKE)")
@Cmd(name = "l2_switch")
module l2_switch {
    @AddPrefix("^BUILD/extbld/^MOD_PATH/install")
    source "l2_switch.o"

    depends embox.compat.posix.util.gettimeofday
    depends embox.compat.atomic.pseudo_atomic
    depends embox.compat.libc.math
    @NoRuntime depends embox.lib.libsupcxx
    @NoRuntime depends embox.lib.libstdcxx
    @NoRuntime depends embox.lib.libgcc
    @NoRuntime depends third_party.boost.libs
    @NoRuntime depends third_party.lib.libpcap
    @NoRuntime depends third_party.lib.libgmp
    @NoRuntime depends third_party.lib.openssl
    @NoRuntime depends third_party.lib.thrift
    @NoRuntime depends embox.compat.posix.pthread_rwlock
    @NoRuntime depends embox.compat.posix.sys.mman.mmap_api
    @NoRuntime depends embox.compat.posix.util.nanosleep
    @NoRuntime depends embox.compat.libc.all
    @NoRuntime depends embox.compat.posix.pthread_key
}

```

С помощью аннотации @Build можно задавать свои правила для сборки. В том числе можно указать часто используемый Makefile, который позволяет задавать три цели: configure, build, install, что является аналогом общеизвестных configure, make, make install. Для его использования в качестве опции к аннотации нужно использовать script="(EXTERNAL_MAKE)".

Опишем более подробно файл сборки. Вначале нам необходимо скачать исходный код интегрируемого проекта. Для этого необходимо задать несколько переменных:

- PKG_SOURCES – название пакета;
- PKG_SOURCES – url для скачивания;
- PKG_MD5 – контрольная сумма md5;
- PKG_PATCHES – список файлов с патчами для наложения на оригинальные исходники.

После чего описываются три цели. Цель configure переходит в директорию с распакованными и пропатченными файлами проекта, где вызывается ./configure с указанными параметрами. После вызываются цели build и install, которые вызывают make и make install с переданными параметрами.

Листинг 2: Makefile файл для проекта P4/behavioral-model

```
PKG_NAME := behavioral-model
```

```
PKG_VER := 1.15.0
```

```
PKG_SOURCES := https://github.com/p4lang/(PKG_NAME)/archive/refs/tags/(PKG_VER).tar.gz
```

```
PKG_MD5      := dc9d34e33daf40e54d9d71a69aece2dc
```

```
PKG_PATCHES := only_l2_switch.patch
```

```
PKG_PATCHES += disable_lib_check.patch
```

```
PKG_PATCHES += fix.patch
```

```
include (EXTBLD_LIB)
```

```
(CONFIGURE) :
```

```
export EMBOX_GCC_LINK=full; \  
cd (PKG_SOURCE_DIR) && ./autogen.sh && ( \  
./configure --prefix=(PKG_INSTALL_DIR) \  
CC=(EMBOX_GCC) \  
CFLAGS="(BM_CXXFLAGS)" \  
CXX=(EMBOX_GXX) \  
CXXFLAGS="(BM_CXXFLAGS)" \  
LDFLAGS=(BM_LDFLAGS) \  
--host=(AUTOCONF_TARGET_TRIPLET) \  
--with-thrift=(ROOT_DIR)/build/extbld/third_party/lib/  
thrift/install/include \  
--with-nanomsg=no \  
--enable-static \  
--disable-shared \  
--disable-undeterministic-tests \  
--with-stress-tests=no \  
)  
touch @
```

```
(BUILD) :
```

```
cp (ROOT_DIR)/third-party/p4/behavioral-model/patches/config.h  
(PKG_SOURCE_DIR)/config.h  
cp (ROOT_DIR)/third-party/p4/behavioral-model/patches/thrift_src  
/Makefile (PKG_SOURCE_DIR)/thrift_src/Makefile  
cd (PKG_SOURCE_DIR) && ( \  
(MAKE) MAKEFLAGS='(EMBOX_IMPORTED_MAKEFLAGS)'; \  
)  
touch @
```

(INSTALL) :

```
cp (PKG_SOURCE_DIR)/targets/l2_switch/l2_switch  
(PKG_INSTALL_DIR)/l2_switch.o  
touch @
```

Перед портированием проекта P4, в ОС Embox аналогичным образом были сконфигурированы и добавлены все необходимые зависимости.

В результате получилось поддержать в ОС Embox программный коммутатор вместе с его зависимостями, который используется для построения программно-конфигурируемых сетей и настраивается при помощи P4-программ.

6. Тестирование

Для тестирования полученного решения на локальной машине была поднята виртуальная сеть состоящая из двух хостов, а также l2_switch на базе ОС Embox сконфигурированного при помощи программы на языке P4. Топология данной сети представлена на Рис.6.

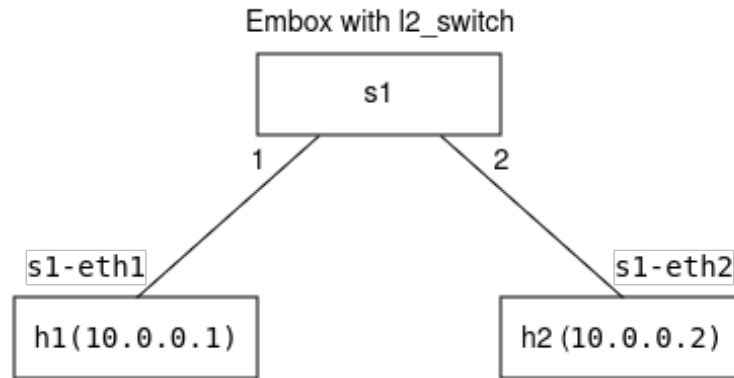


Рис. 6: Топология сети

Данная сеть была построена при помощи эмулятора сетей mininet [5]. Для этого был создан скрипта на Python, который представлен ниже.

Листинг 3: Создание сети

```
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import Topo

class SingleSwitchTopo(Topo):
    def build(self):
        switch = self.addSwitch('s1')
        host1 = self.addHost('h1', ip='10.0.0.1')
        host2 = self.addHost('h2', ip='10.0.0.2')
        self.addLink(host1, switch)
        self.addLink(host2, switch)

def run():
```

```

topo = SingleSwitchTopo()
net = Mininet(topo=topo, controller=None)

net.start()
CLI(net)
net.stop()

if __name__ == '__main__':
    run()
}

```

После запуска данного скрипта и l2_switch на ОС Embox вместе со скомпилированной программой P4, успешно выполнен базовый ping между хостами, который выведен в консоль mininet.

Листинг 4: Ping между хостами

```

mininet> pingall
*** Ping: testing ping reachability
h1 → h2
h2 → h1
*** Results: 0% dropped (2/2 received)

```

В свою очередь в логах коммутатора можно увидеть пошаговый процесс получения, парсинга, обработки и передачи пакета. Основные этапы включают извлечение заголовков, применение таблиц для маршрутизации и фильтрации, репликацию пакета и его передачу на выходной порт.

Листинг 5: Логи коммутатора

```

[10:00:59.031] [bmv2] [D] Processing packet received on port 1
[10:00:59.031] [bmv2] [D] Parser 'parser': start
[10:00:59.031] [bmv2] [D] Parser 'parser' entering state 'start'
[10:00:59.031] [bmv2] [D] Parser state 'start' has no switch,
going to default next state
[10:00:59.031] [bmv2] [T] Bytes parsed: 0

```

```

[10:00:59.031] [bmv2] [D] Parser 'parser' entering state
'parse_ethernet'
[10:00:59.031] [bmv2] [D] Extracting header 'ethernet'
[10:00:59.031] [bmv2] [D] Parser state 'parse_ethernet'
has no switch, going to default next state
[10:00:59.031] [bmv2] [T] Bytes parsed: 14
[10:00:59.032] [bmv2] [D] Parser 'parser': end
[10:00:59.032] [bmv2] [D] Pipeline 'ingress': start
[10:00:59.032] [bmv2] [T] Applying table 'smac'
[10:00:59.032] [bmv2] [D] Looking up key:
* ethernet.srcAddr      : be21346419d1

[10:00:59.032] [bmv2] [D] Table 'smac': miss
[10:00:59.032] [bmv2] [D] Action entry is mac_learn —
[10:00:59.032] [bmv2] [T] Action mac_learn
[10:00:59.032] [bmv2] [T] Primitive (no source info)
[10:00:59.032] [bmv2] [T] Primitive (no source info)
[10:00:59.032] [bmv2] [T] Applying table 'dmac'
[10:00:59.032] [bmv2] [D] Looking up key:
* ethernet.dstAddr      : 333300000002
...

```

По результатам тестирования можно сделать вывод, что реализованное решение корректно работает.

7. Заключение

В рамках выполнения данной работы были достигнуты следующие результаты.

1. Произведен обзор текущих сетевых возможностей ОС Embox и современных подходов к построению сетевой инфраструктуры. Выявлены места улучшения текущих сетевых возможностей. Выбран подход, основанный на создании программно-конфигурируемых сетей (SDN).
2. Спроектировано решение для построения программно-конфигурируемых сетей, основанное на проекте P4.
3. Спроектированное решение реализовано. Интегрированы необходимые компоненты проекта P4 вместе с зависимостями в Embox.
4. Выполнено тестирование решения на базе ОС Embox и проекта P4.

Код проекта Embox доступен на GitHub⁷. Имя пользователя: semtagg.

⁷Исходный код ОС Embox: <https://github.com/embox/>

Список литературы

- [1] Introducing OpFlex – A new standards-based protocol for Application Centric Infrastructure. — URL: <https://blogs.cisco.com/datacenter/introducing-opflex-a-new-standards-based-protocol-for-applications> (online; accessed: 2024-05-23).
- [2] P4: Programming Protocol-Independent Packet Processors. — URL: <http://www.sigcomm.org/node/3503> (online; accessed: 2024-05-23).
- [3] P4 software switch [Электронный ресурс]: GitHub. — URL: <https://github.com/p4lang/behavioral-model> (online; accessed: 2024-05-23).
- [4] Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability. — URL: <https://ieeexplore.ieee.org/document/7884951> (online; accessed: 2024-05-23).
- [5] mininet. — URL: <http://mininet.org/> (online; accessed: 2024-05-23).
- [6] Библиотека gmp [Электронный ресурс]: Википедия. — URL: https://ru.wikipedia.org/wiki/GNU_Multi-Precision_Library (online; accessed: 2024-05-23).
- [7] Библиотека pcap [Электронный ресурс]: GitHub. — URL: <https://github.com/the-tcpdump-group/libpcap> (online; accessed: 2024-05-23).
- [8] Мировой рынок использования SDN инфраструктуры. — URL: [https://www.tadviser.ru/index.php/Статья:SD-WAN_\(Software_Defined\)_Программно-определяемая_WAN-сеть](https://www.tadviser.ru/index.php/Статья:SD-WAN_(Software_Defined)_Программно-определяемая_WAN-сеть) (online; accessed: 2024-05-23).
- [9] Описание реализаций P4 software switch [Электронный ресурс]:

GitHub. — URL: <https://github.com/p4lang/behavioral-model/blob/main/targets/README.md> (online; accessed: 2024-05-23).

- [10] Проект Apache Thrift. — URL: <https://thrift.apache.org/> (online; accessed: 2024-05-23).
- [11] Проект Boost. — URL: <https://www.boost.org/> (online; accessed: 2024-05-23).
- [12] Проект Nanomsg. — URL: <https://nanomsg.org/> (online; accessed: 2024-05-23).
- [13] Проект OpenSSL. — URL: <https://www.openssl.org/> (online; accessed: 2024-05-23).