РАЗРАБОТКА ПРОГРАММНОЙ СИСТЕМЫ УПРАВЛЕНИЯ ПАРАМЕТРАМИ И СТРУКТУРАМИ ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЕЙ

Цель курсового проектирования: разработка программной системы для визуального проектирования структуры, измерения параметров каналов связи, исследования процессов маршрутизации и управления потоками данных в программно-конфигурируемых сетях (ПКС).

Задачи курсового проектирования:

- разработка визуальной среды для графического отображения параметров и структур ПКС, управления параметрами ПКС с помощью графического интерфейса пользователя;
- изучение возможностей протокола OpenFlow для осуществления маршрутизации и сбора информации о топологии ПКС;
- изучение структуры и возможностей контроллера Ryu;
- разработка приложения контроллера Ryu для управления параметрами ПКС.

1. ТЕМАТИКА КУРСОВОГО ПРОЕКТИРОВАНИЯ

1.1. СТРУКТУРА ПКС

ПКС – это новый архитектурный подход к построению и проектированию компьютерных сетей. В рамках этого подхода сеть становится программируемой и управляется централизованно.

Архитектура ПКС состоит из трех уровней (рис. 1.1): инфраструктурного уровня, уровня управления и уровня сетевых сервисов. На инфраструктурном уровне находятся сетевые устройства, такие как хосты и OpenFlow коммутаторы, а также каналы связи между ними. Обмен данными на этом уровне происходит при помощи различных протоколов модели OSI. На уровне управления находится контроллер, управляющий всей сетью, а также каналы связи, соединяющие контроллер с OpenFlow коммутаторами. Обмен информацией между контроллером и коммутаторами происходит с помощью протокола OpenFlow. Уровень сетевых сервисов включает в себя приложения и сервисы, функционирующие в сети.

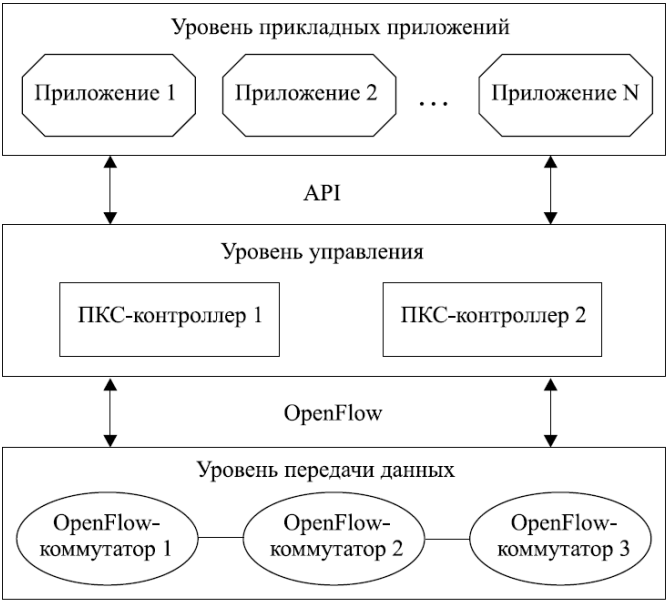

Рис. 1.1. Архитектура ПКС

На рис. 1.2 представлен пример топологии ПКС, состоящей из трех OpenFlow коммутаторов *S*1, *S*2, *S*3, хостов *H*1, *H*2 и контроллера *C*0.

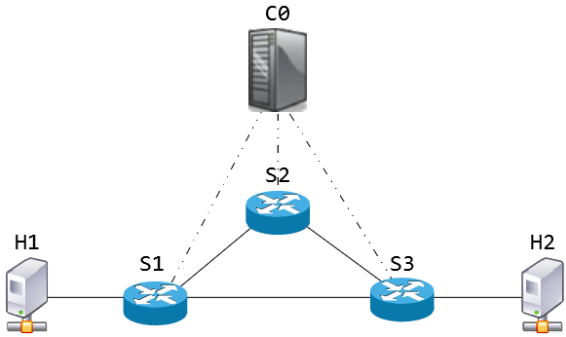

Рис. 1.2. Пример топологии ПКС

1.2. OPENFLOW КОММУТАТОР

В отличии от сетевого коммутатора, применяющегося в классической сетевой архитектуре, OpenFlow коммутатор осуществляет

передачу пакетов в соответствии с правилами, установленными контроллером, а не самостоятельно, на основе заложенных интеллектуальных алгоритмов. На рис. 1.3 приведена упрощенная логическая структура OpenFlow коммутатора.

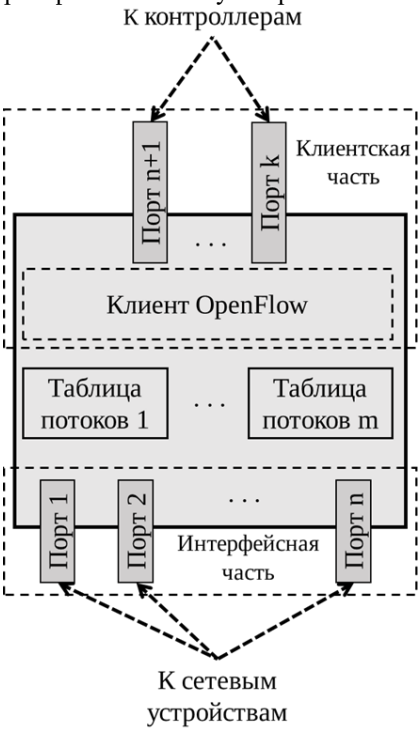

Рис. 1.3. Упрощенная логическая структура OpenFlow коммутатора

OpenFlow коммутатор имеет определенное количество сетевых интерфейсов, к которым подключаются каналы связи. Как правило, вместо термина «сетевой интерфейс» используют термин «порт». Каждый порт коммутатора имеет свой номер. Существует особый порт для обмена сообщениями с контроллером. В коммутаторе содержится несколько таблиц маршрутизации, называемых также таблицами потоков (англ. Flow Tables). В данные таблицы помещаются правила обработки пакетов. Добавлять, изменять и удалять правила может только контроллер посредством передачи в коммутатор специальных сообщений. В соответствии с правилами, хранящимися в таблицах

потоков, коммутатор обрабатывает поступающие через порты пакеты без участия контроллера.

1.3. КОНТРОЛЛЕР ПКС

Контроллер ПКС является вычислительным комплексом или сервером, на котором работает программное обеспечение, управляющее сетью. Контроллер выражает концепцию сетевой операционной системы, управляющей ресурсами сети и являющейся прослойкой между приложениями верхнего уровня и аппаратным обеспечением сети. Значительным преимуществом данной концепции является абстрагирование управляющего программного обеспечения от конкретной физической реализации контроллера.

Технологии ПКС предполагают подчинение всех коммутаторов контроллеру, управляющему логикой работы сети. Контроллер обменивается с коммутаторами сообщениями по заранее установленному соединению в сети управления. Если в процессе работы OpenFlow коммутатор получил пакет, не соответствующий ни одному правилу в таблице потоков, то он отправляет этот пакет контроллеру, который его обработает и отправит коммутатору набор инструкций для выполнения.

1.4. ПРОТОКОЛ OPENFLOW

Протокол OpenFlow определяет типы и структуры сообщений, которыми могут обмениваться контроллеры и OpenFlow коммутаторы в процессе функционирования сети. На текущий момент существует 6 версий протокола OpenFlow – от 1.0 до 1.5. Основные особенности разных версий протоколов приведены в табл. 1. Наиболее популярной версией протокола является версия 1.3.

Таблица 1. Основные особенности версий протокола OpenFlow

Версия	Главная особенность	Причина	Применение
1.0 – 1.1	Несколько таблиц потоков		
	Группировка потоков	Применение наборов действий к группе потоков.	Балансировка нагрузки, аварийное переключение, агрегирование каналов связи
	Полная поддержка VLAN и MPLS		
1.1 – 1.2	Расширяемое поле соответствия	Гибкое изменение поля соответствий	

	Поддержка нескольких контроллеров	Увеличение надежности, улучшение балансировки нагрузки и масштабируемости	Аварийное переключение контроллера, балансировка нагрузки контроллера
1.2 – 1.3	Таблица счетчиков	Обеспечение QoS и DiffServ	
	Пропуск записей в таблице потоков	Обеспечение гибкости	
1.3 – 1.4	Синхронизированные таблицы	Улучшение масштабируемости таблиц потоков	Обучение и переадресация на основе MAC адресов
	Пакетное выполнение модификаций	Улучшение синхронизации коммутаторов	Одновременное управление несколькими коммутаторами
1.4 – 1.5	Таблицы исходящих потоков	Обработка пакетов в выходном порту	
	Планирование пакетное выполнения модификаций	Улучшение синхронизации коммутаторов	

Протокол OpenFlow стандартизирует множество типов сообщений для гибкого и эффективного управления ПКС. Далее приведены сообщения, характерные для OpenFlow версии 1.3.

1.4.1. Сообщения от контроллера к коммутатору

Множество сообщений от контроллера к коммутатору (табл. 2) используется для управления OpenFlow коммутаторами в процессе работы сети. Некоторые сообщения объединяются в пары запрос от контроллера – ответ от коммутатора.

Таблица 2. Сообщения от контроллера к коммутатору

Группа сообщений	Сообщение	Описание
Handshake	OFPFeaturesRequest	Запрос функций коммутатора после установления соединения
	OFPSwitchFeatures	Ответ коммутатора на запрос функций
Switch Configuration	OFPSetConfig	Запрос на установку параметров конфигурации коммутатора

	OFPGetConfigRequest	Запрос на получение параметров конфигурации коммутатора
	OFPGetConfigReply	Ответ коммутатора на OFPGetConfigRequest
Flow Table Configuration	OFPTableMod	Запрос на конфигурирование таблицы потоков
Modify State	OFPFlowMod	Модификация таблицы потоков
	OFPGroupMod	Модификация группы таблиц потоков
	OFPPortMod	Изменение поведения порта
	OFPMeterMod	Изменение счетчика
Multipart	OFPDescStatsRequest	Запрос статистики коммутатора
	OFPDescStatsReply	Ответ на OFPDescStatsRequest
	OFPFlowStatsRequest	Запрос статистики одного потока
	OFPFlowStatsReply	Ответ на OFPFlowStatsReply
	OFPAggregateStatsRequest	Запрос совокупной статистики потока
	OFPAggregateStatsReply	Ответ на OFPAggregateStatsReply
	OFPTableStatsRequest	Запрос статистики таблицы потоков
	OFPTableStatsReply	Ответ на OFPTableStatsReply
	OFPPortStatsRequest	Запрос статистики одного порта
	OFPPortStatsReply	Ответ на OFPPortStatsReply
	OFPPortDescStatsRequest	Запрос статистики всех портов
	OFPPortDescStatsReply	Ответ на OFPPortDescStatsReply
	OFPQueueStatsRequest	Запрос статистики очереди
	OFPQueueStatsReply	Ответ на OFPQueueStatsReply
	OFPGroupStatsRequest	Запрос статистики группы
	OFPGroupStatsReply	Ответ на OFPGroupStatsReply
	OFPGroupDescStatsRequest	Запрос статистики всех групп
	OFPGroupDescStatsReply	Ответ на OFPGroupDescStatsReply
	OFPGroupFeaturesStatsRequest	Запрос возможностей групп
	OFPGroupFeaturesStatsReply	Ответ на OFPGroupFeaturesStatsReply
	OFPMeterStatsRequest	Запрос статистики счетчика
	OFPMeterStatsReply	Ответ на OFPMeterStatsReply
	OFPMeterConfigStatsRequest	Запрос конфигурации счетчика
	OFPMeterConfigStatsReply	Ответ на OFPMeterConfigStatsReply

	OFPMeterFeaturesStatsRequest	Запрос функций измерительной подсистемы
	OFPMeterFeaturesStatsReply	Ответ на OFPMeterFeaturesStatsReply
	OFPTableFeaturesStatsRequest	Запрос функций таблицы потоков
	OFPTableFeaturesStatsReply	Ответ на OFPTableFeaturesStatsReply
Queue Configuration	OFPQueueGetConfigRequest	Запрос конфигураций очереди
	OFPQueueGetConfigReply	Ответ на запрос конфигураций очереди
Packet-Out	OFPPacketOut	Отправка пакета через порт коммутатора
Barrier	OFPBarrierRequest	Запрос барьера
	OFPBarrierReply	Ответ на запрос барьера
Role Request	OFPRoleRequest	Запрос на изменение роли контроллера
	OFPRoleReply	Ответ на запрос на изменение роли контроллера
Set Asynchronous Configuration	OFPSetAsync	Установить асинхронные сообщения, которые контроллер будет получать по протоколу OpenFlow
	OFPGetAsyncRequest	Запрос асинхронного сообщения
	OFPGetAsyncReply	Ответ на запрос асинхронного сообщения

1.4.2. Асинхронные сообщения

Асинхронные сообщения (табл. 3) коммутатор отправляет контроллеру в случае наступления определенных событий.

Таблица 3. Асинхронные сообщения

Группа сообщений	Сообщение	Описание
Packet-In	OFPPacketIn	Сообщение о пакете, пришедшем на порт коммутатора
Flow Removed	OFPFlowRemoved	Сообщение об удалении записи потока из таблицы потоков
Port Status	OFPPortStatus	Сообщение об изменении состоянии порта

Error	OFPErrorMsg	Сообщение об ошибке

1.4.3. Симметричные сообщения

Симметричные сообщения (табл. 4) используются для установления соединения, проверки подключения или проведения экспериментов в сети.

Таблица 4. Симметричные сообщения

Группа сообщений	Сообщение	Описание
Hello	OFPHello	Приветственное сообщение, которым обмениваются контроллер и коммутатор при установке соединения.
Echo Request	OFPEchoRequest	Эхо – запрос
Echo Reply	OFPEchoReply	Эхо – ответ
Experimenter	OFPExperimenter	Экспериментальное сообщение

1.5. ПРАВИЛА ОБРАБОТКИ ПАКЕТОВ КОММУТАТОРОМ ПКС

Таблица потоков OpenFlow коммутатора состоит из множества правил обработки пакетов, приходящих на его порты. На рис. 1.4 представлена структура таблицы потоков с одним правилом. Правило обработки пакета состоит из трех частей. Первая часть состоит из набора сопоставлений (англ. Match), вторая – из набора инструкций (англ. Instructions), третья содержит приоритет (англ. Priority). Сопоставления требуются коммутатору для того, чтобы отнести пакет к некоторой группе пакетов. В табл. 5 представлены некоторые параметры пакетов, которые могут быть использованы в качестве аргументов сопоставления.

Набор инструкций определяет то, как коммутатор должен обрабатывать пакеты, отнесенные к соответствующей группе. Коммутатор может отправить пакет на определенный порт или выполнить другие действия. Некоторые действия над пакетами представлены в табл. 6.

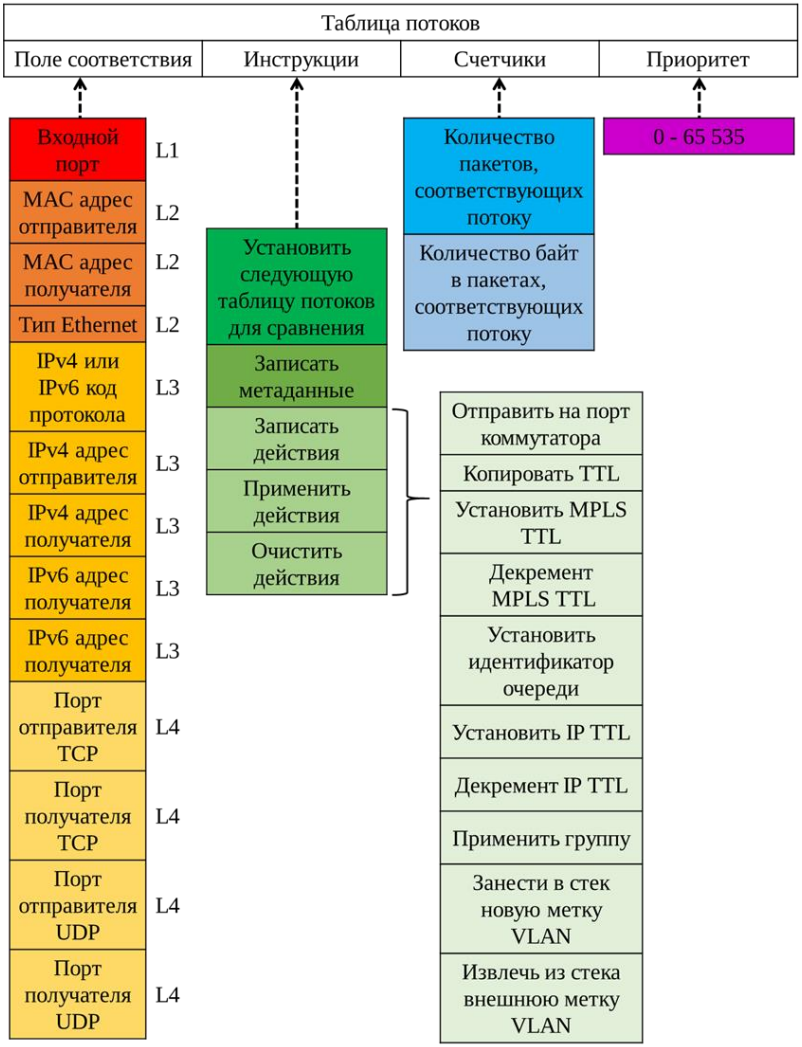

Рис. 1.4. Таблица потоков OpenFlow коммутатора

Если коммутатор получил пакет, для которого не найдено сопоставление в таблице потоков, то он отправляет контроллеру сообщение *Packet-In*. Контроллер примет решение о дальнейших действиях и отправит в коммутатор сообщение *Packet-Out* содержащее инструкции о том, на какой порт отправить этот пакет.

Таблица 5. Некоторые параметры пакетов

Параметр пакета	Описание
in_port	Порт, на который пришел пакет
eth_dst	MAC адрес получателя
eth_src	MAC адрес отправителя
eth_type	Тип Ethernet
vlan_vid	Идентификатор VLAN
ipv4_src	IPv4 адрес отправителя
ipv4_dst	IPv4 адрес получателя
tcp_src	Номер TCP порта отправителя
tcp_dst	Номер TCP порта получателя
udp_src	Номер UDP порта отправителя
udp_dst	Номер UDP порта получателя
ipv6_src	IPv6 адрес отправителя
ipv6_dst	IPv6 адрес получателя

Таблица 6. Некоторые действия с пакетами

Действие над пакетом	Описание
OFPP_IN_PORT	Пакет пересылается на принимающий порт
OFPP_FLOOD	Пакет направляется на все порты VLAN, кроме заблокированных и принимающего
OFPP_ALL	Пакет направляется на все физические порты, кроме принимающего
OFPP_CONTROLLER	Пакет отправляется контроллеру в виде сообщения о приходе пакета

Приоритет для правила выбирается из диапазона [0 ... 65535].

1.6. ЭМУЛЯТОР ПКС MININET

Mininet – это эмулятор ПКС, позволяющий создать виртуальную сеть передачи данных и виртуальные конечные устройства. Запуск и управление эмулятором осуществляются с помощью командной строки. В листинге 1 приведена команда для запуска эмулятора.

Листинг 1. Запуск Mininet

```
$ sudo mn
```

При запуске Mininet без параметров будет запущена сеть, имеющая топологию в соответствии с рис. 1.5.

После успешного запуска эмулятора в интерфейсе командной строки появляется приглашение для ввода *mininet>*. Для исследования структуры сети в Mininet имеются следующие команды: *net*, *nodes* и *dump*. В листинге 2 приведены результаты выполнения этих команд.

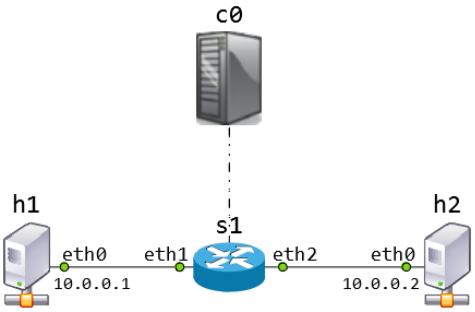

Рис. 1.5. Топология Mininet по умолчанию

Листинг 2. Результат выполнения команд net, nodes, dump

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=4167>
<Host h2: h2-eth0:10.0.0.2 pid=4170>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4176>
<Controller c0: 127.0.0.1:6653 pid=4160>
```

По умолчанию Mininet подключает к виртуальной сети встроенный контроллер (условное обозначение `c0`), позволяющий вычислять маршруты для передачи пакетов.

Каждый хост в Mininet является копией исходной файловой системы операционной системы компьютера, на котором он запущен.

Следовательно, каждый хост способен выполнять такие же действия и команды, что и реальный компьютер.

Чтобы выполнить команду на определенном хосте в Mininet, достаточно написать команду следующего синтаксиса: *hx command*, где *x* – номер хоста, на котором требуется выполнить команду, а *command* – сама команда, предназначенная для выполнения.

Для тестовой передачи пакетов между каждой парой хостов в сети используется команда *pingall*.

При запуске Mininet имеется возможность выбрать одну из стандартных топологий эмулятора. В табл. 7 приведены варианты запуска Mininet со стандартными топологиями.

Таблица 7. Запуск Mininet со стандартными топологиями

Тип топологии	**Команда**	**Пример**	**Параметр p**
Один коммутатор	sudo mn --topo=single,[p]	sudo mn --topo=single,4	Количество хостов
Линейная	sudo mn --topo=linear,[p]	sudo mn --topo=linear,4	Количество коммутаторов
Дерево	sudo mn --topo=tree,[p]	sudo mn --topo=tree,3	Количество уровней дерева

Для остановки эмулятора Mininet используется команда `exit`. Для полного завершения всех процессов Mininet используется команда *sudo mn -c*. Часто выполнение этой команды помогает избавиться от ошибок, возникающих при запуске эмулятора, поэтому рекомендуется выполнять ее каждый раз после выхода из Mininet.

Для создания более сложных топологий рекомендуется использовать сценарии на языке программирования Python. В листинге 3 приведен сценарий, описывающий создание той же топологии, что и на рис. 1.5.

Листинг 3. Сценарий на Python для создания топологии

```
# подключение необходимых модулей
from mininet.net import Mininet
from mininet.node import OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel

# функция, создающая топологию ПКС
def topology():
```

```
    net = Mininet(switch=OVSKernelSwitch)

    # добавление контроллера c0
    c0 = net.addController('c0')

    # добавление хостов h1 и h2
    h1 = net.addHost('h1')
    h2 = net.addHost('h2')

    # добавление коммутатора s1
    s1 = net.addSwitch('s1')

    # добавление физических каналов связи
    net.addLink(s1, h1)
    net.addLink(h2, s1)

    # запуск контроллера c0
    c0.start()

    # подключение коммутатора s1 к контроллеру c0
    s1.start([c0])

    # создание сети
    net.build()

   # запуск командной строки
    CLI(net)

   # остановка сети
    net.stop()

if __name__ ==  'main':
    setLogLevel('info')
    topology()
```

В листинге 5 приведена команда запуска эмулятора Mininet с топологией, описанной в файле сценария (предполагается, что файл сценария носит название *topology.py*).

Листинг 5. Запуск Mininet с пользовательской топологией

```
$ sudo python3 topology.py
```

1.7. КОНТРОЛЛЕР ПКС RYU

Структура приложения Ryu представляет собой множество обработчиков сообщений, отправленных от коммутаторов контроллеру. Каждый обработчик в программном коде является методом основного класса приложения, обернутым специальным декоратором. В листинге 6 приведен фрагмент кода, демонстрирующий объявление обработчика подключения OpenFlow коммутатора.

Листинг 6. Обработчик подключения OpenFlow коммутатора

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
```

При подключении OpenFlow коммутатора к контроллеру ПКС происходит процедура, называемая **рукопожатием (англ. handshake)**. Коммутатор и контроллер обмениваются приветственными сообщениями для установления связи. После этого контроллер ждет от коммутатора сообщение SwitchFeatures, сигнализирующее о готовности коммутатора к работе. Метод **switch_features_handler** является обработчиком сообщения SwitchFeatures, поскольку обернут декоратором **set_ev_cls**. Он принимает аргумент **ev (сокращение от англ. Event – событие)**. Тип события определяется аргументом **EventOFPSwitchFeatures** декоратора. Стадия конфигурирования коммутатора указана с помощью аргумента **CONFIG_DISPATCHER**.

Экземпляр сообщения, принятого контроллером, можно выделить как **ev.msg**. Он содержит в себе экземпляр класса коммутатора **ev.msg.datapath**. Основные атрибуты и методы класса datapath представлены в табл. 8.

Таблица 8. Основные атрибуты класса datapath

Название атрибута	Описание
id	Идентификатор коммутатора OpenFlow
ofproto	Указывает модуль ofproto, который поддерживает используемую версию OpenFlow
ofproto_parser	То же, что и ofproto, указывает на модуль ofproto_parser
send_msg()	Метод, используемый для отправки коммутатору сообщения от контроллера

Еще один часто используемый обработчик – это обработчик пакетов, которые получил коммутатор, но для которых в таблицах потоков нет соответствующих правил. Тип события, принимаемый обработчиком, определяется классом **EventOFPPacketIn**. Данный тип

событий обрабатывается на основной стадии работы коммутатора (**MAIN_DISPATCHER**). Данная стадия наступает после стадии конфигурирования. В листинге 7 приведен фрагмент кода, демонстрирующий объявление обработчика пакетов.

Листинг 7. Обработчик пакетов

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
```

Экземпляр класса сообщения, принятого контроллером, можно выделить как **ev.msg**. Основные атрибуты класса msg представлены в табл. 9.

Таблица 9. Основные атрибуты класса msg

Название атрибута	**Описание**
match	Метаданные принятых пакетов
data	Двоичные данные пакетов
total_len	Длина данных принятых пакетов
buffer_id	Идентификатор пакета в буфере

Поле сопоставлений для правила обработки пакетов, добавляемого в таблицу потоков коммутатора, задается с помощью метода **OFPMatch** класса **ofproto_parser** следующим образом: **ofproto_parser.OFPMatch(<Именованные параметры>)**. Некоторые именованные параметры представлены в табл. 10.

Таблица 10. Именованные параметры класса OFPMatch

Название параметра	**Описание**
in_port	Номер входного порта
eth_dst	MAC адрес получателя пакета
eth_src	MAC адрес отправителя пакета

Поле инструкций определяет то, что будет выполнять коммутатор, если примет пакет, соответствующий полю сопоставления правила. Инструкции, являются методами класса ofproto.

Инструкция **OFPInstructionActions** позволяет задавать действия над пакетами, пришедшими в коммутатор.

Пусть имеется топология ПКС, представленная на рис. 1.6. В листинге 8 представлен код для контроллера Ryu, позволяющий передавать данные между всеми хостами.

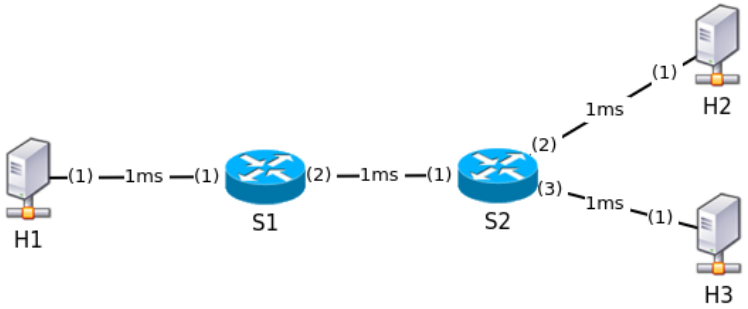

Рис. 1.6. Топология с двумя коммутаторами и тремя хостами

Листинг 8. Код контроллера Ryu

```
""" Подключение необходимых модулей """
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet

class SimpleSwitch13(app_manager.RyuApp):
    """ Класс контроллера """
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION] # Установка версии протокола OpenFlow

    def __init__(self, *args, **kwargs):
        """ Конструктор класса контроллера"""
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.logger.info("*** Ryu контроллер стартовал ***")


    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        """
        Метод добавления нового правила в таблицу потоков
        Аргументы:
        datapath -- объект коммутатора
        priority -- приоритет
```

```
        match -- шаблон соответствия пакета
        actions -- список действий
        """
        ofproto = datapath.ofproto # Получение набора данных для работы с протоколом OpenFlow
        parser = datapath.ofproto_parser # Получение объекта парсера протокла OpenFLow

        # Создание списка инструкций для коммутатора
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        # Создание объекта, представляющего OpenFlow сообщение
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
        # Отправка OpenFlow сообщения коммутатору
        datapath.send_msg(mod)


    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        """ Обработчик подключения коммутатора """

        datapath = ev.msg.datapath # Получение объекта коммутатора
        ofproto = datapath.ofproto  # Получение набора данных для работы с протоколом OpenFlow
        parser = datapath.ofproto_parser # Получение парсера протокла OpenFLow

        self.logger.info("Подключился новый коммутатор с id %s", format(datapath.id, "d").zfill(16))

        """ Установка правила для отправки неизвестных пакетов в контроллер """
        match = parser.OFPMatch() # Задание пустого шаблона пакета
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)] #Действие -- отправить пакет в контроллер
        self.add_flow(datapath, 0, match, actions) # Добавить в коммутатор правило с приоритетом 0 (самый низкий)
```

```
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        """
        Обработчик получения пакета от контроллера
        """
        msg = ev.msg # Получение экземпляра сообщения
        datapath = msg.datapath # Получение объекта коммутатора
        ofproto = datapath.ofproto # Получение набора данных для работы с протоколом OpenFlow
        parser = datapath.ofproto_parser # Получение парсера протокла OpenFLow

        in_port = msg.match['in_port'] # Получение номера входящего порта

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        mac_dst = eth.dst # Получение MAC адреса получатель
        mac_src = eth.src # Получение MAC адреса отправителя

        dpid = datapath.id

        """ Костыль фильтр пакетов """
        if mac_dst[:5] == "33:33":
            return

        self.logger.info("Пришел неопознанный пакет на порт %s коммутатора %s, MAC отправителя %s, MAC получателя %s", in_port, dpid, mac_src, mac_dst)


        out_port = None # Выходной порт
        actions = [] # Список действий
        eth_dst = "" # MAC адрес получателя

        """ Определение выходного порта """
```

```
        if dpid == 1:
            if mac_dst == "00:00:00:00:00:01":
                out_port = 1

            elif mac_dst == "00:00:00:00:00:02":
                out_port = 2
            elif mac_dst == "00:00:00:00:00:03":
                out_port = 2

        elif dpid == 2:
            if mac_dst == "00:00:00:00:00:01":
                out_port = 1
            elif mac_dst == "00:00:00:00:00:02":
                out_port = 2
            elif mac_dst == "00:00:00:00:00:03":
                out_port = 3


        if out_port != None:

            match = parser.OFPMatch(eth_dst = mac_dst) # Задаем поле
сопоставления
            actions = [parser.OFPActionOutput(out_port)] # Действие --
отправить пакет на порт
            priority = 1 # Задаем приоритет
            self.add_flow(datapath, priority, match, actions) # Отправляем
правило в коммутатор

        else:
            out_port = ofproto.OFPP_FLOOD
            actions = [parser.OFPActionOutput(out_port)]


        data = None

        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id =
msg.buffer_id, in_port = in_port, actions = actions, data = data)
        datapath.send_msg(out)
```

2. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

Работа над курсовым проектом состоит из следующих этапов.

1. Анализ задания, разработка модели программной системы в виде UML диаграмм, разработка алгоритма в соответствии с выданным заданием в виде схемы алгоритма.

2. Реализация алгоритма в соответствии с выданным заданием на языке программирования высокого уровня.

3. Реализация программной системы на языке программирования высокого уровня.

4. Тестирование и отладка работы программной системы и алгоритма.

5. Экспериментальный анализ работы алгоритма, определение его вычислительной сложности.

6. Оформление пояснительной записки и подготовка к защите.

Результаты курсового проектирования представляются в виде пояснительной записки, которая должна включать:

1) задание на курсовое проектирование;

2) титульный лист;

3) содержание;

4) основные разделы пояснительной записки;

5) приложения.

Рекомендуется следующая последовательность изложения материала в основной части пояснительной записки:

1) введение;

2) анализ задания;

3) разработка схемы алгоритма;

4) разработка моделей программной системы;

5) реализация алгоритма;

6) реализация визуальной среды;

7) реализация приложения для контроллера Ryu;

8) экспериментальный анализ работы программной системы и алгоритма;

9) руководство оператора;

10) заключение;

11) библиографический список.

3. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА ПРОГРАММНУЮ СИСТЕМУ УПРАВЛЕНИЯ ПАРАМЕТРАМИ И СТРУКТУРАМИ ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЕЙ

3.1. Назначение программной системы

Программная система предназначена для исследования работы сетевых алгоритмов управления ПКС с помощью графического пользовательского интерфейса.

3.2. Структура программной системы

Программная система должна состоять из следующих программ:
1) визуальная среда для графического отображения структуры сети и результатов выполнения сетевых алгоритмов;
2) одно или несколько приложений для контроллера Ryu.

3.3. Требования к визуальной среде

3.3.1. Требования к графическому интерфейсу

3.3.1.1. Визуальная среда должна представлять собой оконное приложение с графическим интерфейсом пользователя.

3.3.1.2. Основное окно визуальной среды должно иметь область топологии и несколько меню инструментов (рис. 3.1). Область топологии должна занимать основную часть окна. Меню инструментов могут располагаться справа, слева или сверху области топологии.

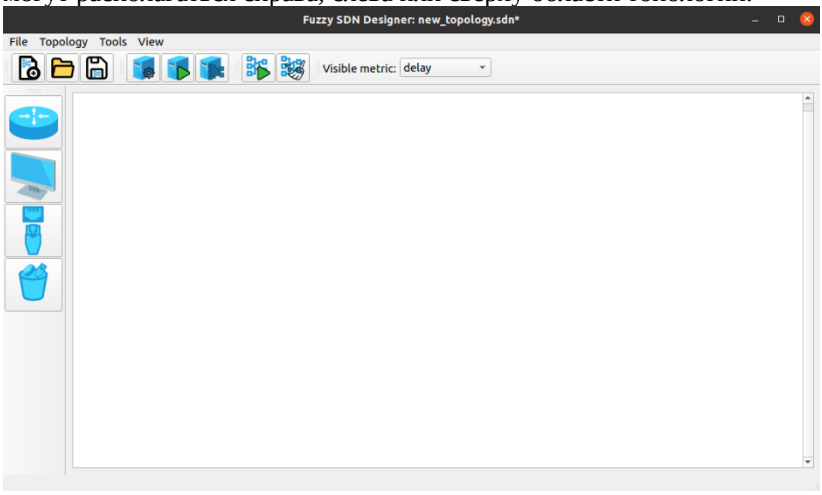

Рис. 3.1. Пример интерфейса визуальной среды

3.3.1.3. Область топологии предназначена для отображения сетевой топологии ПКС. Она должна иметь возможность добавления, удаления и перемещения элементов топологии (рис. 3.2), таких как OpenFlow коммутаторы, хосты, контроллеры и другие элементы в соответствии с заданием, а также каналы связи между ними.

а) б) в)

Рис. 3.2. Пример изображений узловых элементов топологии
а) контроллер, б) OpenFlow коммутатор, в) хост

3.3.1.4. Визуальная среда должна иметь меню инструментов топологии. Данное меню предназначено для добавления или удаления элементов, расположенных в области топологии. В меню топологии должны быть инструменты, позволяющие добавлять и удалять узловые элементы, такие как OpenFlow коммутаторы и хосты, добавлять и удалять каналы связи между узловыми элементами. В зависимости от выданного задания в меню топологии допускается добавлять другие инструменты для работы с областью топологии.

3.3.1.5. Каждый узловой элемент топологии, добавляемый в область топологии, должен иметь уникальное обозначение. OpenFlow коммутаторы могут обозначаться с помощью уникального номера коммутатора datapath id. Хосты могут обозначаться с помощью MAC и (или) IP адреса. Допускается обозначение OpenFlow коммутаторов в виде *sn*, где *n* – порядковый номер коммутатора в топологии. Допускается обозначение хостов в виде *hn*, где *n* – порядковый номер хоста в топологии.

3.3.1.6. Узловые элементы топологии могут иметь свойства, изменяемые пользователем. В качестве таких свойств у хостов должны выступать IP и MAC адреса. Свойства узловых элементов должны изменяться пользователем в отдельном окне (рис. 3.3), которое должно появляться при двойном нажатии левой кнопкой мыши по соответствующему узловому элементу.

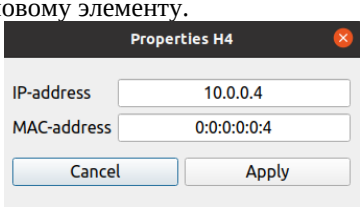

Рис. 3.3. Пример окна свойств хоста

3.3.1.7. Узловые элементы топологии должны соединяться между собой с помощью каналов связи. Канал связи должен иметь вид прямой линии, соединяющей середины изображений узловых элементов (рис. 3.4). По середине линии канала связи должно находиться поле с некоторой числовой характеристикой канала связи.

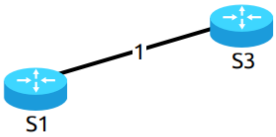

Рис. 3.4. Пример отображения канала связи между OpenFlow коммутаторами

3.3.1.8. Каналы связи должны иметь свойства, изменяемые пользователем: задержка, пропускная способность, процент потерь пакетов. В зависимости от выданного задания допускается добавлять другие свойства. Свойства каналов связи должны изменяться пользователем в отдельном окне (рис. 3.5), которое должно появляться при двойном нажатии левой кнопкой мыши по соответствующему каналу связи.

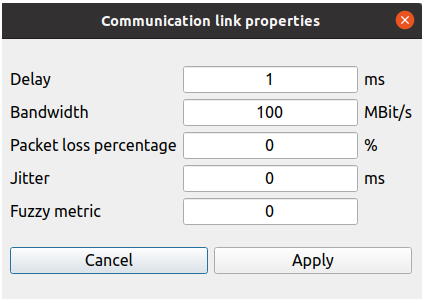

Рис. 3.5. Пример окна свойств канала связи

3.3.1.9. Визуальная среда должна иметь возможность переключения текущего свойства канала связи, значение которого отображается в поле числовой характеристики канала связи.

3.3.2. Требования к функциям визуальной среды

3.3.2.1. Визуальная среда должна иметь возможность сохранения топологии в файл проекта с расширением *.sdn*. В файле проекта структура топологии может быть описана в формате XML или JSON.

3.3.2.2. Визуальная среда должна иметь возможность создания топологии путем открытия файла проекта, описанного выше.

3.3.2.3. Визуальная среда должна иметь возможность создания Python скрипта для запуска эмулятора Mininet с сетевой топологией ПКС, собранной в области топологии. Расширение скрипта – *.sdn.py*.

3.3.2.4. Визуальная среда должна иметь возможность запуска Mininet Python скрипта в отдельном окне терминала. Запуск должен производиться нажатием кнопки в меню инструментов.

3.3.2.5. Визуальная среда должна иметь возможность запуска контроллера Ryu в отдельном окне терминала. Запуск должен производиться нажатием кнопки в меню инструментов.

3.3.2.6. Визуальная среда должна иметь возможность изменения цвета каналов связи (рис. 3.6) для отображения деревьев и маршрутов, а также возможность изменения цвета области вокруг изображений узловых элементов топологии (рис. 3.7) для отображения сетевых сегментов или слайсов.

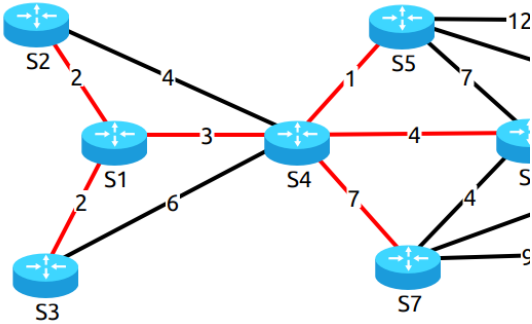

Рис. 3.6. Пример выделения каналов связи

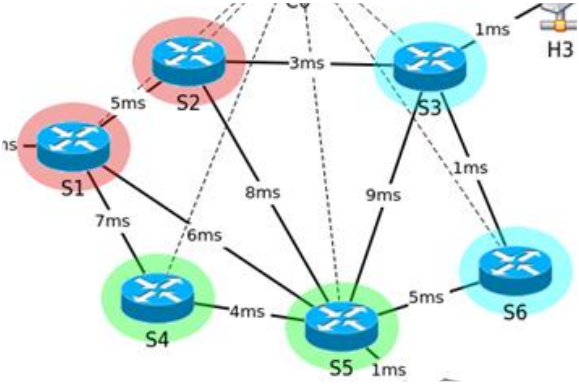

Рис. 3.7. Пример выделения узловых элементов

3.3.2.7. Визуальная среда должна включать в себя TCP-сервер для взаимодействия с контроллером Ryu.

3.4. Требования к приложению контроллера Ryu

3.4.1. Приложение для контроллера Ryu должно получать и обрабатывать сообщения от OpenFlow коммутаторов, запущенных в эмуляторе Mininet.

3.4.2. Приложение должно выполнять алгоритм (алгоритмы), в соответствии с выданным заданием. Алгоритм должен выполняться на основе матрицы смежности, получаемой приложением от визуальной среды или получаемой самостоятельно на основе анализа информации от OpenFlow коммутаторов.

3.4.3. Алгоритм должен быть реализован отдельно от приложения контроллера и должен подключаться к нему в качестве внешнего модуля.

3.4.4. Если в выданном задании подразумевается выполнение маршрутизации по определенным каналам связи, то по результатам работы алгоритма приложение контроллера должно отправлять в OpenFlow коммутаторы соответствующие сообщения для установки правил маршрутизации.

3.4.5. Приложение должно включать в себя HTTP сервер для взаимодействия с визуальной средой.

3.4.6. Приложение должно реагировать на события изменения сетевых параметров или топологии в визуальной среде. В качестве таких событий могут выступать: изменение параметров каналов связи, добавление или удаление OpenFlow коммутатора или хота и т.д.

3.5. Требования к взаимодействию визуальной среды контроллера Ryu и эмулятора Mininet

3.5.1. Управление ПКС должно производиться посредством взаимодействия пользователя с визуальной средой. Визуальная среда должна иметь средства для взаимодействия с контроллером Ryu и эмулятором Mininet.

3.5.2. Визуальная среда должна взаимодействовать с эмулятором Mininet посредством создания Python скрипта, предназначенного для эмуляции сетевой топологии, и последующего его запуска в эмуляторе Mininet.

3.5.3. Визуальная среда должна обмениваться данными с контроллером Ryu посредством сетевого взаимодействия. Передача данных из визуальной среды в контроллер Ryu должна производиться с помощью HTTP запросов, которые будут обработаны HTTP-сервером контроллера. Передача данных из контроллера Ryu в визуальную среду должна производиться посредством установки TCP-соединения с последующей отправкой необходимых данных.

3.5.5. Модель взаимодействия визуальной среды, контроллера Ryu и эмулятора Mininet представлен на рис. 3.8.

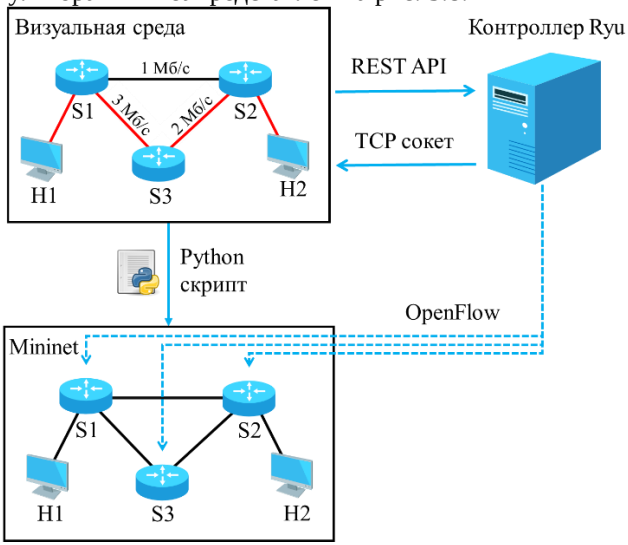

Рис. 3.8. Модель взаимодействия визуальной среды, контроллера Ryu и эмулятора Mininet

В рамках приведенной модели визуальная среда должна позволять проектировать сеть и отображать ее состояние. Эмулятор

Mininet должен виртуализировать работу элементов сети. Контроллер должен управлять сетью посредством взаимодействия с Mininet, производить мониторинг и передавать данные о состоянии сети в визуальную среду.

В результате выполнения работы должна получиться целостная программная система для исследования алгоритмов управления потоками данных в ПКС.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Корячко В.П., Перепелкин Д.А. Программно-конфигурируемые сети. Учебник для вузов. М.: Горячая линия-Телеком, 2020. 288 с.

2. Смелянский Р.Л., Антоненко В.А. Концепции программного управления и виртуализации сетевых сервисов в современных сетях передачи данных: учебное пособие – Москва: КУРС, 2022. – 160 с.

3. Перепелкин Д.А. Основы сегментирования структур программно-конфигурируемых сетей. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2023. – 120 с: ил.

4. Корячко В.П., Перепелкин Д.А., Иванчикова М.А. Основы проектирования мультипровайдерных компьютерных сетей. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2024. – 144 с: ил.

5. Тарасов А.Е., Ушаков Ю.А., Полежаев П.Н., Коннов А.Л., Шухман А.Е. – Программно-конфигурируемые сети в центрах обработки данных. – Самара: Самарский научный центр РАН, 2015. – 193 с.

6. Ли П. Архитектура интернета вещей / пер. с анг. М. А. Райтмана. – М.: ДМК Пресс, 2019. – 454 с.: ил.

7. Big Data and Software Defined Networks. – The Institution of Engineering and Technology (IET), London? United Kingtom, 2018 – 478 p.

8. Mininet Documentation // GitHub URL: https://github.com/mininet/mininet/wiki/Documentation (дата обращения: 10.05.2024).

9. RYU SDN Framework // Ryubook 1.0 documentation URL: https://osrg.github.io/ryu-book/en/html/index.html# (дата обращения: 10.05.2024).

10. Welcome to RYU the Network Operating System (NOS) // Read the Docs URL: https://ryu.readthedocs.io/en/latest/index.html (дата обращения: 10.05.2024).