МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ имени Гумарбека Даукеева»

Кафедра Телекоммуникаций и инновационных технологий

«ДОПУЩЕН К ЗАЩИТЕ»
Зав.кафедрой _PhD, доцент Қадылбекқызы Э.К._____
(ученая степень, звание, Ф.И.О.)
_____ «____» _____202___г.
(подпись)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На тему: Анализ работы протокола OpenFlow в программно-конфигурируемых сетях SDN

Специальность 6М071900 «Радиотехника, электроника и телекоммуникации»

Выполнил: _____Бак Андрей Владимирович_____Группа МРЭТн-19-2
(Ф.И.О.)

Научный руководитель: _Айтмагамбетов А.З., к.т.н., профессор_____
(ученая степень, звание, Ф.И.О.)

Консультант по технической части:_Лещинская Э.М.,к.т.н., профессор_____
(ученая степень, звание, Ф.И.О.)

Нормоконтроль:__ст. преподаватель Павлова Т.А._____
(ученая степень, звание, Ф.И.О.)
_____ «____» _____202___г.
(подпись)

Рецензент: _____
(ученая степень, звание, Ф.И.О.)
_____ «____» _____202___г.
(подпись)

Алматы 2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
имени Гумарбека Даукеева»

Институт Космической Инженерии и Телекоммуникаций

Кафедра Телекоммуникаций и инновационных технологий

Специальность 6М071900 «Радиотехника, электроника и телекоммуникации»

ЗАДАНИЕ
на выполнение магистерской диссертации

Магистранту Бак Андрею Владимировичу
(фамилия, имя, отчество)

Тема диссертации «Анализ работы протокола OpenFlow в программно-конфигурируемых сетях SDN»

Утверждена приказом по университету №263 от «30» декабря 2020 г.

Срок сдачи законченной диссертации «25» мая 2021 г.

Цель исследования состоит в определении характеристик качества работы программно-конфигурируемых сетей с использованием протокола OpenFlow

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации:

1. Анализ концепции программно-конфигурируемых сетей
2. Разработка модели для анализа величины сквозной задержки
3. Исследование величины задержки передаваемого трафика в сети SDN
4. Оценка влияния обновления TCAM на величину задержки

Перечень графического материала (с точным указанием обязательных чертежей):

1. Схема разбивки сквозной задержки
2. Задержка настройки потока
3. Зависимость величины задержки от количества коммутаторов в сети

4. Задержка с/без обращения на контроллер, когда новая скорость потока составляет 5 тысяч пакетов/с

5. Сравнение задержки трафика с/без обновления TCAM

Рекомендуемая основная литература:

1 Fei Hu. Network Innovation through OpenFlow and SDN. Principles and Design. February 2014.-520p.

2 Thomas D. Nadeau and Ken Gray. SDN Software Defined Networks. OReilly Media. September 7, 2013. – 384 p.

3 Siamak Azodolmolky. Software Defined Networking with OpenFlow. Packt Publishing. October 25, 2013. – 152 p.

4 OpenFlow Switch Specification. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

График
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1. Информационный обзор по программно-конфигурируемым сетям	01.02.2020	
2. Анализ алгоритма работы сети SDN	01.06.2020	
3. Моделирование сквозной задержки в программно-конфигурируемой сети	01.11.2020	
4. Исследование величины задержки передаваемого трафика в сети SDN	01.03.2021	
5. Анализ полученных экспериментальных и расчетных данных	01.05.2021	

Дата выдачи задания 30 сентября 2019г.

Заведующий кафедрой _____ (Кадылбекқызы Э.К.)
(подпись) (Ф.И.О.)

Научный
руководитель диссертации _____ (Айтмагамбетов А.З.)
(подпись) (Ф.И.О.)

Задание принял к исполнению
магистрант _____ (Бак А.В.)
(подпись) (Ф.И.О.)

Аңдатпа

Мақалада бағдарламалық-конфигурацияланатын желіні жасау концепциясы қарастырылған, желілік технологиялардың дамуындағы олардың маңыздылығы көрсетілген. SDN желісіндегі OpenFlow хаттамасының жұмысына талдау жасалған. Mininet-те жасалған және OpenDaylight контроллеріне қосылған бағдарламалық-конфигурацияланатын желіні басқару үдерісі көрсетілген.

Контроллер мен OpenFlow-коммутатордың өзара әрекеттесуі нәтижесінде берілетін трафик талданды. Трафик алмасу үдерісін зерттеу үшін желі тораптарының арасында Wireshark желілік дестелердің талдағышы қолданылды. SDN желісі бойынша берілетін трафиктің кешігуін қалыптастыру моделі жасалды. SDN желісіндегі трафиктің кешігу мәні зерттелді.

Аннотация

В диссертационной работе рассмотрена концепция создания программно-конфигурируемых сетей, показано их значение в развитии сетевых технологий. Выполнен анализ работы протокола OpenFlow в сети SDN. Представлен процесс управления программно-конфигурируемой сетью, созданной в Mininet, и подключенной к контроллеру OpenDaylight.

Проанализирован трафик, передаваемый в результате взаимодействия контроллера и OpenFlow-коммутатора. Для исследования процесса обмена трафиком между узлами сети был использован анализатор сетевых пакетов Wireshark. Разработана модель формирования задержки передаваемого трафика по сети SDN. Исследована величина задержки передаваемого трафика в сети SDN.

Annotation

The dissertation work discusses the concept of software-defined networks creation, shows their importance in the development of network technologies. The analysis of the operation of the OpenFlow protocol in the SDN network is carried out. The process of a software-defined network management created in Mininet and connected to an OpenDaylight controller is presented.

The traffic transmitted as a result of interaction between the controller and the OpenFlow switch has been analyzed. To study the process of traffic exchanging between network nodes, the Wireshark network packet analyzer was used. A model for the delay of transmitted traffic over the SDN network has been developed. The value of the delay in the SDN network has been investigated

Содержание

Введение

Рост спроса на телекоммуникационные услуги обуславливает новые требования к сетевым технологиям, которые должны обеспечить защищенный, надежный и качественный доступ пользователей к информационным ресурсам. Одной из основных проблем современных сетей является высокая загруженность каналов связи, что приводит к неэффективному использованию сетевой структуры в целом. В данном контексте ведущие фирмы и корпорации мира пришли к однозначному выводу – необходимо создание гибкой и надежной системы управления телекоммуникационными сетями. Стало ясно, что экстенсивный путь развития на базе специализированного оборудования является тупиковым. Необходимы новые подходы к развитию бизнес операторов и сервис-провайдеров. Это привело к появлению новой концепции построения и организации работы сетей передачи данных – программно-конфигурируемым сетям (Software Defined Network, SDN)

Отделение управления сетью от сетевых устройств является ключевой идеей программно-конфигурируемой сети (SDN). SDN - это сдвиг парадигмы в компьютерных сетях, где функциональность управления сетью (также известная как плоскость управления) отделена от функциональности пересылки данных (также известной как плоскость данных), и, кроме того, управление разделением является программируемым. Миграция управляющей логики, которая раньше была тесно интегрирована в сетевые устройства (например, коммутаторы Ethernet) в доступные и логически централизованные контроллеры, позволяет абстрагироваться от базовой сетевой инфраструктуры с точки зрения приложений. Такое разделение открывает путь к более гибкой, программируемой, независимой от производителя, экономически эффективной и инновационной сетевой архитектуре. Помимо абстракции сети, архитектура SDN предоставляет набор прикладных программных интерфейсов (API), которые упрощают реализацию общих сетевых служб (например, маршрутизация, многоадресная передача, безопасность, контроль доступа, управление пропускной способностью, управление трафиком, QoS, энергоэффективность, и различные формы управления политикой). В результате предприятия, операторы получают беспрецедентную программируемость, автоматизацию и управление сетью, что позволяет им создавать гибкие сети с высокой степенью масштабируемости, которые легко адаптируются к меняющимся потребностям бизнеса.

На сегодняшний день самым распространенным решением для реализации управления сетевых устройств в программно-конфигурируемых сетях является протокол OpenFlow. OpenFlow – это протокол взаимодействия между сетевыми устройствами, такими как коммутаторы и маршрутизаторы, и централизованным контроллером, который представляет собой сетевую операционную систему, установленную на выделенном физическом сервере, в программно-конфигурируемой сети. Такое управление может заменить или

дополнить работающую на сетевом устройстве функцию, осуществляющую построение маршрутов, создание таблицы коммутации и т.д.

Актуальность использования сетей SDN объясняется рыночными тенденциями. Технология SDN должна способствовать переориентации операторских бизнес-моделей на облачные и цифровые сервисы. Также операторы надеются освободить сети от переизбытка оборудования.

В телекоммуникационной индустрии виртуализация способна сократить операционные и капитальные затраты операторов на 60%. Такой эффект достигается за счет замены коммуникационной системы фрагментированного типа новой инфраструктурой, в основу которой положены приложения, не привязанные к конкретному серверному оборудованию. При этом существующее сетевое оборудование разгрузится в среднем на 25% за счет увеличения гибкости сети, а сроки выведения телекоммуникационных услуг на рынок значительно сократятся. В перспективе SDN позволит оперативно модернизировать конфигурацию сетей, замерять их емкость и отслеживать спектр реализуемых услуг.

Новизна работы состоит в комплексной оценке величины задержки в программно-конфигурируемой сети.

Практическая значимость работы заключается в возможности использования результатов исследования в обучающих курсах по сетям SDN.

1 Анализ концепции программно-конфигурируемых сетей

1.1 Программно-конфигурируемая сеть SDN

В современном мире, при разнообразии поставщиков телекоммуникационного оборудования, прогресс, модификация и сложность различных технологий стремятся к росту количества устройств и объемов передаваемых данных. Классическая архитектура сети, основы которой закладывались в конце 60-х годов прошлого века, в связи с быстрым ростом многообразия, сложности и важности решаемых задач, устарела и не всегда способна эффективно реагировать на новые потребности рынка. Чтобы полностью преобразовать исторически устоявшуюся архитектуру, необходимо затратить немалых средств, а модернизация отдельных элементов сети не может длиться бесконечно [1].

Сегодня инфокоммуникационные сети, несмотря на повсеместное использование и масштабное развитие, встречают на своем пути множество проблем:

– рост количества пользователей и трафика, приводящий к перегрузке сетевых устройств;

– огромное количество протоколов и их стеков, число которых с каждым днем только увеличивается;

– традиционные сети, которые проприетарны, ограничены для исследований и практически любых изменений. Интерфейс настройки сетевого оборудования иногда зависит даже от модели у одного и того же производителя, следовательно, специалисты в области сетевых технологий должны уметь анализировать системы разных вендоров и работать с большим количеством протоколов;

– оборудование разных производителей, которое довольно часто может конфликтовать между собой, несмотря на универсальность сетевых протоколов;

– настройка сети с большим количеством узлов, требующая много времени, а управление такими сетями осуществляется путем конфигурирования их элементов через специализированные интерфейсы;

– освоение системы оборудования конкретного производителя, требующее переподготовки специалистов.

Таким образом, большое количество проблемных факторов ведет к усложнению сетевого оборудования и структуры самих информационных сетей и, как следствие, удорожанию сетевого оборудования. Возможным решением этой проблемы является реализация правил обработки данных в виде программных модулей, а не встраивание их в аппаратные средства. Это позволяет сетевым администраторам лучше контролировать сетевой трафик и, следовательно, имеет большой потенциал для значительного улучшения производительности сети с точки зрения эффективного использования ресурсов и скорости. Такой подход определен в программно-

конфигурируемой сети (SDN). В SDN обработка данных изолирована от аппаратного обеспечения, а ее управление реализовано в программном модуле, называемом контроллером [2].

SDN позволяет сетевым администраторам работать с данными в сети более эффективным и инновационным способом. Используя SDN, администраторы имеют возможность контролировать поток данных, а также изменять характеристики коммутирующих устройств (устройств маршрутизации) в сети из центрального местоположения, при этом приложение управления реализовано в виде программного модуля, без необходимости работать с каждым из них. Устройство индивидуально. Это позволяет сетевым администраторам изменять таблицы маршрутизации (пути маршрутизации) в устройствах сетевой маршрутизации. Это также дает дополнительный уровень контроля над данными сети, потому что администратор может назначить высокий/низкий приоритет определенным пакетам данных или разрешить/заблокировать определенные пакеты, проходящие через сеть с различными уровнями управления. В результате сетевой трафик может эффективно контролироваться и, следовательно, может использоваться как механизм управления нагрузкой трафика в сетях. Используются несколько стандартов для внедрения SDN. OpenFlow - это один из самых популярных и общепринятых стандартов для реализации SDN. OpenFlow обеспечивает удаленное управление сетевыми устройствами маршрутизации, благодаря своей способности контролировать таблицы маршрутизации трафика в сети. Рисунок 1.1 иллюстрирует основное различие между SDN и обычными сетями.

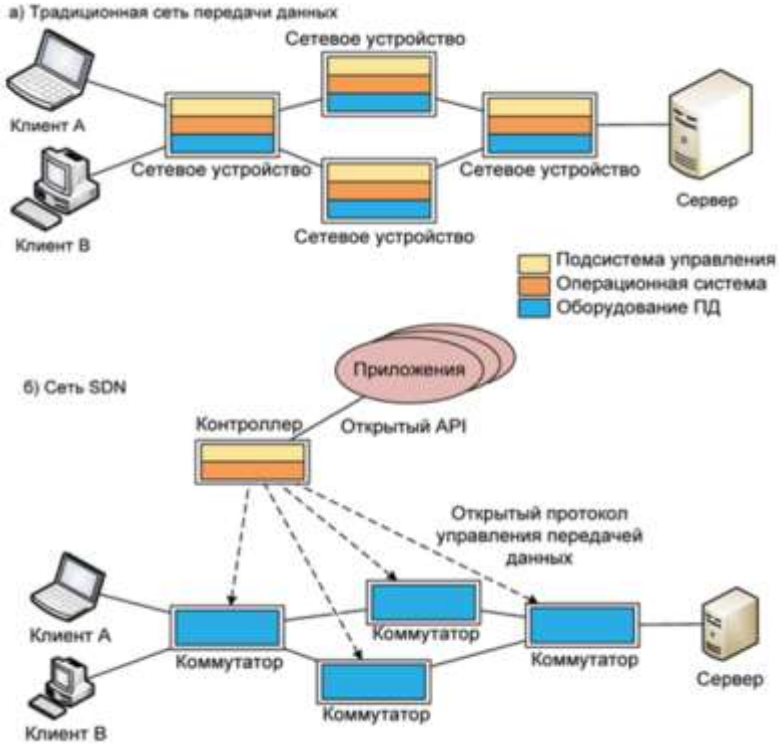

Рисунок 1.1 - Различие между традиционными сетями и SDN

Основные преимущества SDN [3]:

1) Повышение производительности. В связи с тем, что с коммутаторов снимаются нагрузки по обработке линии управления, программно-конфигурируемая сеть дает возможность этим устройствам направить все свои ресурсы на ускорение перемещения трафика.

2) Упрощение процесса администрирования. На централизованном контроллере программно-конфигурируемой сети системный администратор может наблюдать сеть в едином представлении, что способствует повышению удобства управления, обеспечения безопасности и выполнения других задач.

3) Ускорение реализации и тестирования новых сервисов. Программные средства программно-конфигурируемых сетей позволяют администраторам добавлять новые функции к уже имеющейся сетевой архитектуре.

4) Повышение безопасности. Программно-конфигурируемая сеть предоставляет возможность администратору четко видеть все потоки трафика, поэтому он будет гораздо легче замечать вторжения, определять приоритеты различных типов трафика, разрабатывать правила реагирования сети при заторах и проблемах с оборудованием.

5) Применение облачных технологий. В облаках данные и приложения размещены на компьютерах, которые взаимодействуют по сети, и технология SDN способна дать требуемый облакам уровень «интеллектуальности» сетей.

1.2 Архитектура SDN

Архитектура SDN представляет трехуровневую модель, состоящую из уровней инфраструктуры, управления и приложений. Если подробно рассмотреть информационные потоки в архитектуре SDN (рисунок 1.2), то можно увидеть два направления обмена информацией. Первый поток – между уровнем приложений и уровнем управления, второй – между уровнем физических сетевых устройств и уровнем управления. Первый поток получил название «северный интерфейс», а второй – «южный интерфейс». В качестве «северного интерфейса» выступает протокол на основе REST API (REST – общие принципы организации взаимодействия приложения с сервером, API – интерфейс программирования приложений, состоящий из набора готовых кодов, предоставляемых приложением для использования во внешних программных продуктах), а в качестве «южного интерфейса» - протокол OpenFlow [4].

Уровень приложений представляет собой совокупность приложений, которые напрямую взаимодействуют с SDN-контроллером, запрашивая необходимые ресурсы посредством API, решая высокоуровневые задачи по управлению сетью. Примером таких приложений могут служить средства мониторинга, аналитики или бизнес-приложения. Например, конкретное бизнес-приложение Microsoft Lynk и его основная роль – изменение сети в режиме реального времени под текущие нужды обслуживаемой программы

(изменение Quality of Service или создание VPN тоннеля между двумя абонентами) [5].

Уровень управления осуществляет низкоуровневое логически централизованное управление, которое обеспечивает форвардинг трафика на инфраструктурном уровне с помощью открытого интерфейса OpenFlow. Уровень управления постоянно осуществляет мониторинг всей сети и способен управлять всеми сетевыми устройствами.

Уровень инфраструктуры состоит из среды передачи данных и коммутаторов SDN (OpenFlow-коммутаторы), которые могут быть как логическими, так и физическими элементами сети.

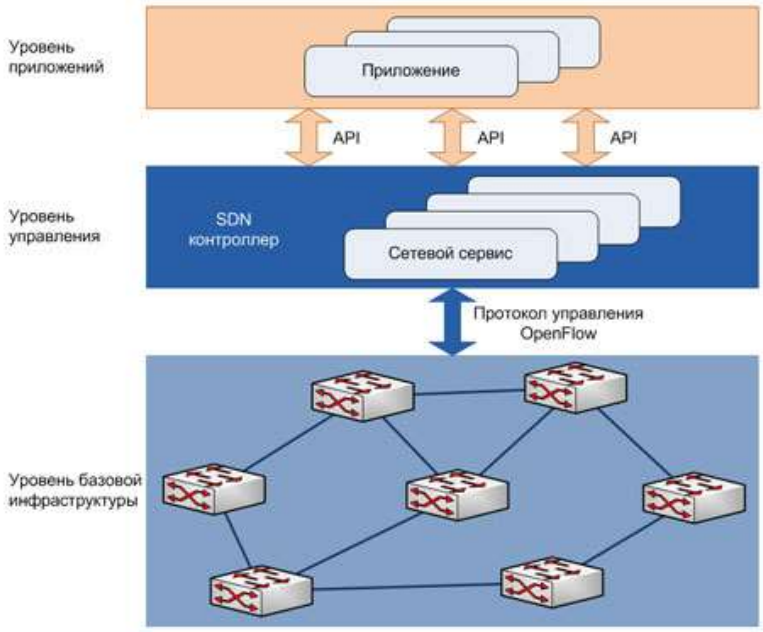

Рисунок 1.2 – Архитектура SDN

Компоненты архитектуры сети:

1. Оркестратор – представленная в виде аппаратного устройства или программного обеспечения платформа автоматизации, которая используется приложениями для конфигурирования устройств уровней управления и инфраструктуры (например, корректирует работу нескольких контроллеров) и выполняет конкретные сервисные запросы.

2. Контроллер – вычислительное устройство (например, сервер) на котором функционирует специализированная платформа, состоящая из сетевых управляющих приложений и сетевой операционной система (СОС):

– СОС – готовый фреймворк, задача которого предоставить интерфейс прикладного программирования для сетевых приложений по контролю и управлению сетью целиком, а также реализация механизмов управления таблицами одного или нескольких OpenFlow-коммутаторов (добавление, удаление, модификация правил, сбор статистики). Разработано более 30

сетевых операционнах систем для контроллеров SDN-сетей: Ryu, Maestro, MUL, Beacon, Floodlight, POX, NOX, In-kernel [6];

– SDN-приложение – программная реализация различных сетевых функций и сервисов (маршрутизация, балансировка нагрузки, фильтрация трафика, шлюзы, сетевые экраны, шифрование, DPI, NAT, DHCP, DNS).

3. OpenFlow-коммутатор – простое программируемое сетевое устройство, выполняющее лишь функции коммутации данных согласно инструкциям контроллера. Основные составляющие OpenFlow-коммутатора, принцип работы описаны в разделе 1.3.

4. Программный интерфейс (northbound API) – открытый интерфейс программирования, позволяющий программировать контроллер извне. Предназначен для создания экосистемы приложений. Пользователями данного API являются все разработчики сетевых приложений

1.3 OpenFlow коммутатор

1.3.1 Компоненты коммутатора OpenFlow. Коммутатор OpenFlow состоит из одной или нескольких таблиц потоков и групповой таблицы, которые выполняют поиск и пересылку пакетов, а также одного или нескольких каналов OpenFlow на внешний контроллер (рисунок 1.3). Коммутатор связывается с контроллером, и контроллер управляет коммутатором по протоколу OpenFlow [7].

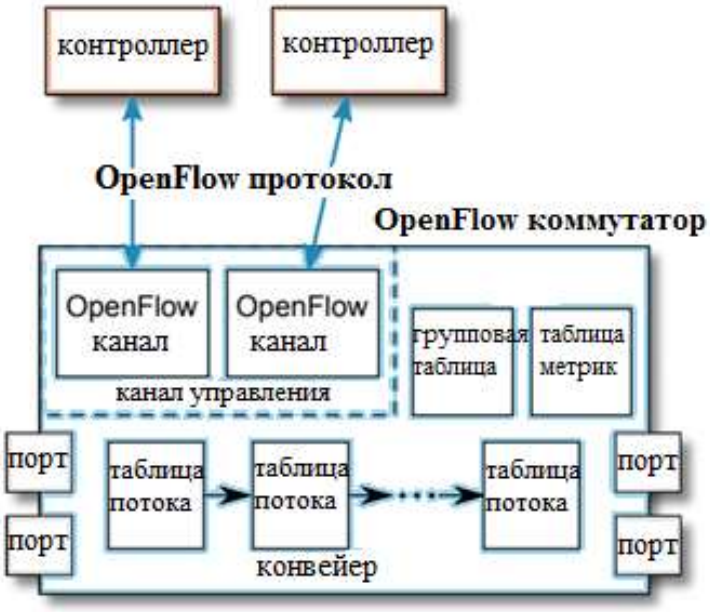

Рисунок 1.3 - Основные компоненты коммутатора OpenFlow

Используя протокол коммутации OpenFlow, контроллер может добавлять, обновлять и удалять записи потоков в таблицах потоков. Каждая таблица потоков в коммутаторе содержит набор записей потоков, каждая запись потока состоит из полей сопоставления, счетчиков и набора инструкций, которые должны применяться к сопоставляемым пакетам.

Сопоставление начинается с первой таблицы потоков и может продолжаться до дополнительных таблиц потоков конвейера. Если соответствующая запись найдена, выполняются инструкции, связанные с конкретной записью потока. Если в таблице потоков совпадение не найдено, результат зависит от конфигурации записи потока пропущенных таблиц (например, пакет может быть перенаправлен на контроллеры по каналу OpenFlow, отброшен или может перейти к следующей таблице потоков).

Инструкции, связанные с каждой записью потока, содержат действия или изменяют конвейерную обработку. Действия, включенные в инструкции, описывают пересылку пакетов, модификацию пакетов и обработку групповых таблиц. Инструкции конвейерной обработки позволяют отправлять пакеты в последующие таблицы для дальнейшей обработки и позволяют передавать информацию в виде метаданных между таблицами. Обработка конвейера таблицы останавливается, когда набор команд, связанный с соответствующей записью потока, не указывает следующую таблицу, в этот момент пакет обычно модифицируется и пересылается.

Записи потока могут пересылать в порт. Обычно это физический порт, но это также может быть логический порт, определенный коммутатором, или зарезервированный порт. Зарезервированные порты могут указывать общие действия пересылки, такие как отправка на контроллер или пересылка с использованием методов не OpenFlow, таких как «обычная» обработка коммутатора, в то время как определяемые коммутатором логические порты могут указывать группы агрегации каналов, туннели или интерфейсы обратной связи.

Действия, связанные с записями потока, могут также направлять пакеты в группу, которая определяет дополнительную обработку. Группы представляют наборы действий, а также более сложную семантику пересылки (например, многопутевой ввод-вывод, агрегация каналов). Как общий уровень косвенности, группы также позволяют нескольким записям потока пересылать на один идентификатор (например, переадресация IP на общий следующий переход). Эта позволяет эффективно изменять общие выходные действия в записях потока.

Таблица группы содержит записи группы; каждая запись группы содержит список блоков действий с определенной семантикой, зависящей от типа группы.

1.3.2 OpenFlow порты. Порты OpenFlow ничем не отличаются от портов обычного коммутатора уровня L2, каждый порт имеет входной/выходной буфер и уникальный номер, который выступает в роли имени порта.

Набор портов OpenFlow может не совпадать с набором сетевых интерфейсов, предоставляемых оборудованием коммутатора, некоторые сетевые интерфейсы могут быть отключены для OpenFlow, а коммутатор OpenFlow может определять дополнительные порты OpenFlow.

Протокол OpenFlow определяет набор стандартных портов: физические порты, логические порты и зарезервированные порты [8].

Физические порты – это порты, определенные коммутатором, которые соответствуют аппаратному интерфейсу коммутатора.

Логические порты – это определенные коммутатором порты, которые не соответствуют напрямую аппаратному интерфейсу коммутатора. Логические порты являются абстракциями более высокого уровня, которые могут быть определены в коммутаторе с использованием не-OpenFlow методов (например, групп агрегации каналов, туннелей, интерфейсов обратной связи).

Логические порты могут включать инкапсуляцию пакетов и могут отображаться на различные физические порты. Обработка, выполняемая логическим портом, зависит от реализации и должна быть прозрачной для обработки OpenFlow, и эти порты должны взаимодействовать с обработкой OpenFlow подобно физическим портам OpenFlow.

Единственные различия между физическими и логическими портами заключаются в том, что пакет, связанный с логическим портом, может иметь дополнительное поле конвейера, называемое Tunnel-ID, связанное с ним, и когда пакет, полученный на логическом порту, отправляется на контроллер, его логический порт, и его базовый физический порт сообщаются контроллеру.

Зарезервированные порты определяют общие действия пересылки, такие как отправка на контроллер, флуд или пересылка с использованием не-OpenFlow методов, таких как «обычная» обработка коммутатора.

Коммутатор не обязан поддерживать все зарезервированные порты, только те, которые помечены как «Обязательные»:

– обязательные: ALL - представляет все порты, которые коммутатор может использовать для пересылки определенного пакета. Может использоваться только в качестве выходного порта. В этом случае копия пакета начинает исходящую обработку на всех стандартных портах, исключая входной порт пакета и порты, которые настроены OFPPC_NO_FWD;

– обязательные: CONTROLLER - представляет канал управления с контроллерами SDN. Может использоваться как входной порт или как выходной порт. При использовании в качестве выходного порта пакет инкапсулируется в сообщение Packet_In и отправляется с использованием протокола OpenFlow. Когда используется как входной порт, это идентифицирует пакет, исходящий от контроллера;

– обязательные: TABLE - представляет начало конвейера OpenFlow. Отправляет пакет в первую таблицу потоков, чтобы пакет мог быть обработан через конвейер OpenFlow;

– обязательные: IN PORT - представляет входной порт пакета. Может использоваться только как выходной порт, отправляет пакет через входной порт;

– обязательные: ANY - специальное значение, используемое в некоторых запросах OpenFlow, когда порт не указан (т.е. порт подстановочный). Некоторые запросы OpenFlow содержат ссылку на определенный порт, к которому относится только запрос. Использование ANY

номера порта в этих запросах позволяет этому экземпляру запроса применяться ко всем портам. Не может использоваться ни как входной, ни как выходной порт;

– необязательные: LOCAL - представляет локальный сетевой стек коммутатора и его стек управления. Может использоваться как входной порт или как выходной порт. Локальный порт позволяет удаленным объектам взаимодействовать с коммутатором и его сетевыми службами через сеть OpenFlow, а не через отдельную сеть управления;

– необязательные: NORMAL - представляет пересылку с использованием традиционного не-OpenFlow конвейера коммутатора. Может использоваться только как выходной порт и обрабатывает пакет, используя обычный конвейер. Если коммутатор не может пересылать пакеты из конвейера OpenFlow в обычный конвейер, он должен указать, что он не поддерживает это действие;

– необязательные: FLOOD - представляет потоковую рассылку с использованием традиционного не-OpenFlow конвейера коммутатора. Может использоваться только как выходной порт, фактический результат зависит от реализации. В общем случае пакет будет отправляться на все стандартные порты, но не на входной порт или порты, находящиеся в заблокированном состоянии. Коммутатор также может использовать идентификатор VLAN пакета или другие критерии, чтобы выбрать, какие порты использовать для потоковой рассылки.

1.3.3 Конвейерная обработка. Коммутаторы, совместимые с OpenFlow, бывают двух типов: только OpenFlow и гибридные. В коммутаторах, поддерживающих только операцию OpenFlow, все пакеты обрабатываются конвейером OpenFlow и не могут обрабатываться иначе [9].

Гибридные коммутаторы поддерживают работу как OpenFlow, так и обычную коммутацию Ethernet, то есть традиционную коммутацию Ethernet L2, изоляцию VLAN, маршрутизацию L3 (маршрутизация IPv4, маршрутизация IPv6), обработку ACL и QoS. Эти коммутаторы должны обеспечивать механизм классификации вне OpenFlow, который направляет трафик либо в конвейер OpenFlow, либо в обычный конвейер. Например, коммутатор может использовать тег VLAN или входной порт пакета, чтобы решить, обрабатывать ли пакет с использованием одного конвейера или другого, или он может направлять все пакеты в конвейер OpenFlow. Гибридный коммутатор OpenFlow также может позволить пакету отправляться из конвейера OpenFlow в обычный конвейер через зарезервированные порты NORMAL и FLOOD.

Конвейер OpenFlow каждого логического коммутатора OpenFlow содержит одну или несколько таблиц потоков, каждая из которых содержит несколько записей потоков. Обработка конвейера OpenFlow определяет, как пакеты взаимодействуют с этими таблицами потоков (рисунок 1.4) [7].

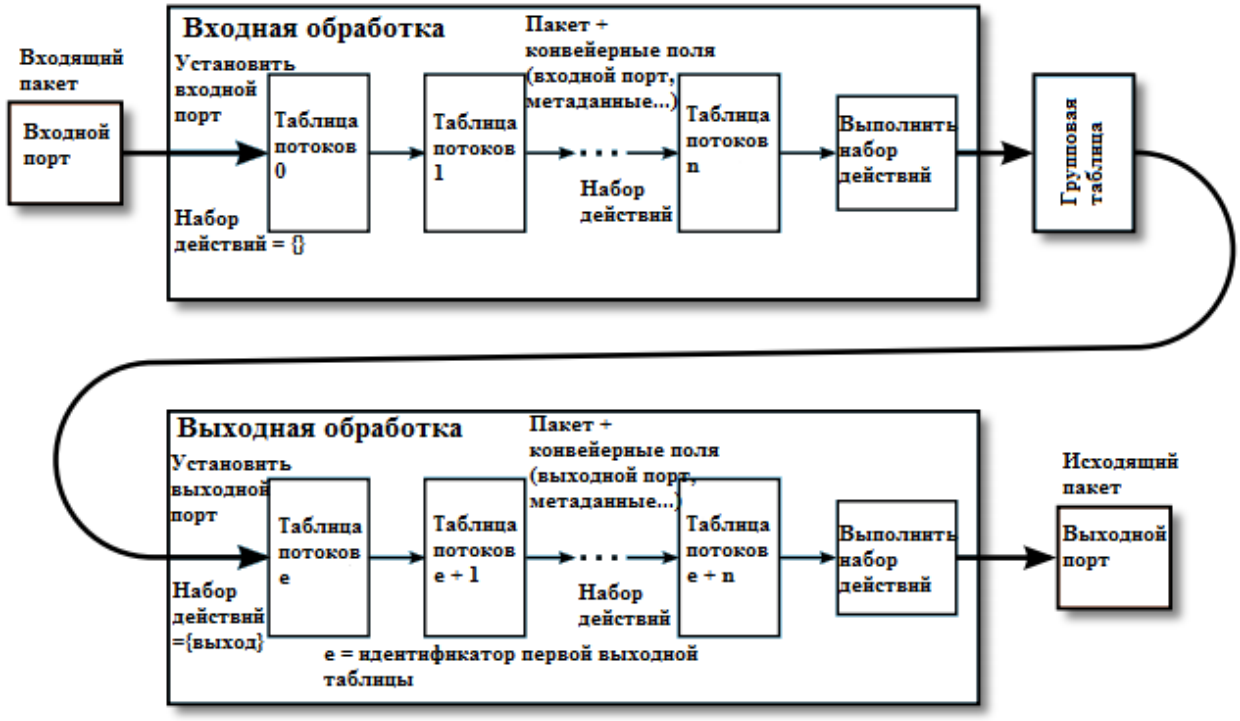

Рисунок 1.4 – Поток пакетов через конвейер обработки

Таблицы потоков коммутатора OpenFlow нумеруются в том порядке, в котором они могут пересматриваться пакетами, начиная с 0. Конвейерная обработка происходит в два этапа: входная обработка и выходная обработка.

Конвейерная обработка всегда начинается с входной обработки в первой таблице потоков: пакет должен сначала сопоставляться с записями потока в таблице потоков 0 (рисунок 1.5). Другие таблицы входного потока могут использоваться в зависимости от результата обработки в первой таблице. Если результатом входной обработки является пересылка пакета на выходной порт, коммутатор OpenFlow может выполнить выходную обработку в контексте этого выходного порта. Выходная обработка является необязательной, коммутатор может не поддерживать какие-либо выходные таблицы или может быть не настроен для их использования. Если ни одна выходная таблица не установлена в качестве первой выходной таблицы, пакет должен обрабатываться выходным портом, и в большинстве случаев пакет пересылается из коммутатора. Если выходная таблица установлена как первая выходная таблица, пакет должен быть сопоставлен с записями потока этой таблицы потоков, и могут использоваться другие таблицы выходных потоков в зависимости от результата совпадения в этой таблице потоков.

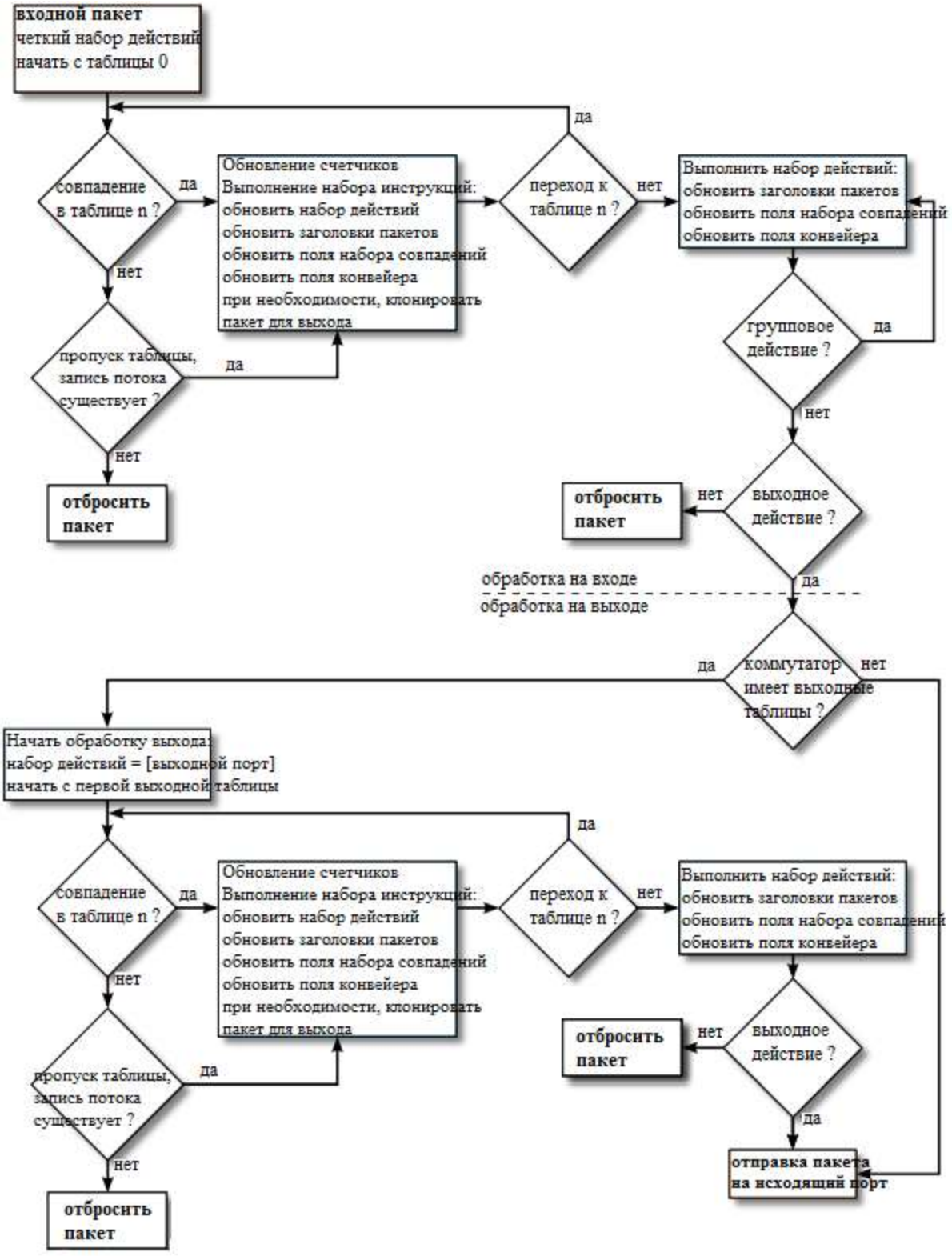

Рисунок 1.5 – Упрощенная блок-схема прохождения потока пакетов через коммутатор OpenFlow

При обработке таблицей потоков пакет сопоставляется с записями потока данной таблицы потоков. Если запись потока найдена, выполняется набор инструкций, включенный в эту запись потока. Эти инструкции могут

явно направлять пакет в другую таблицу потоков (используя инструкцию GotoTable), где тот же процесс повторяется снова. Запись потока может только направить пакет на номер таблицы потоков, который больше, чем номер его собственной таблицы потоков, другими словами, конвейерная обработка может идти только вперед, а не назад. Если соответствующая запись потока не направляет пакеты в другую таблицу потоков, текущий этап обработки конвейера останавливается на этой таблице, пакет обрабатывается с соответствующим набором действий и обычно пересылается.

Если пакет не совпадает с записью потока в таблице потоков, это означает пропуск таблицы. Поведение при пропуске таблицы зависит от конфигурации таблицы. Инструкции, включенные в запись потока пропущенных таблиц, могут гибко указывать, как обрабатывать не сопоставленные пакеты. Полезные опции включают удаление пакетов, передачу пакетов в другую таблицу или отправку пакетов в контроллеры по каналу управления.

1.3.4 Таблицы OpenFlow. Каждый OpenFlow-коммутатор содержит одну или несколько уникальных таблиц, которые он заполняет на основе данных, полученных от контроллера. Так как по сети передаются не отдельные пакеты, а потоки данных, то такая таблица получила название Flow table (таблица потоков). Под потоком понимается совокупность пакетов (сеансов связи), определяемая по различным признакам и ограниченная только возможностями реализации самих таблиц (например, TCP-сессия, пакеты с определенного MAC или IP-адреса, пакеты с одинаковым номером порта OpenFlow-коммутатора).

Каждая запись таблицы потоков (таблица 1.1) содержит:

– поля соответствия: наименование поля заголовка пакета (field) и строка из двоичных символов и символа неопределенности (pattern). Запись о потоке применима к пакету, если все двоичные символы строки pattern поля field совпали с соответствующим полем пришедшего пакета и все классификаторы (список пар (field, pattern)) совместимы с заголовком пакета;

– приоритет: натуральное число из определенного диапазона, указывающее степень значимости о потоке в таблице потоков. Используется для избирательного применения записи о потоках к конкретному пришедшему пакету (для обработки выбирается запись с наибольшим приоритетом);

– счетчики: содержат статистические данные о работе записи (например, информацию о том, сколько пакетов было обработано в соответствии с данной записью о потоке);

– инструкции: для изменения набора действий или конвейерной обработки;

– таймаут: метки, которые определяют срок жизни записи в таблице потоков (максимальное время пребывания и максимальный срок простоя правила);

– cookie: набор определенных данных, выбранных контроллером. Может использоваться контроллером для фильтрации записей потока, на которые влияют статистика потока, изменения потока и запросы на удаление потока. Не используется при обработке пакетов;

– флаги: флаги предназначены для изменения записи таблицы потоков, например, флаг OFPFF_SEND_FLOW_REM - удаление записи в таблице потоков.

Таблица 1.1 – Основные компоненты записи потока в таблице потоков.

Поля соответствия	Приоритет	Счетчики	Инструкции	Таймаут	Cookie	Флаги

1.3.5 Инструкции. Инструкция – операция, которая содержит либо множество действий, чтобы добавить их в список действий, либо действия, которые немедленно применяются к пакету или модифицируют процесс обработки пакета в конвейере (рисунок 1.6).

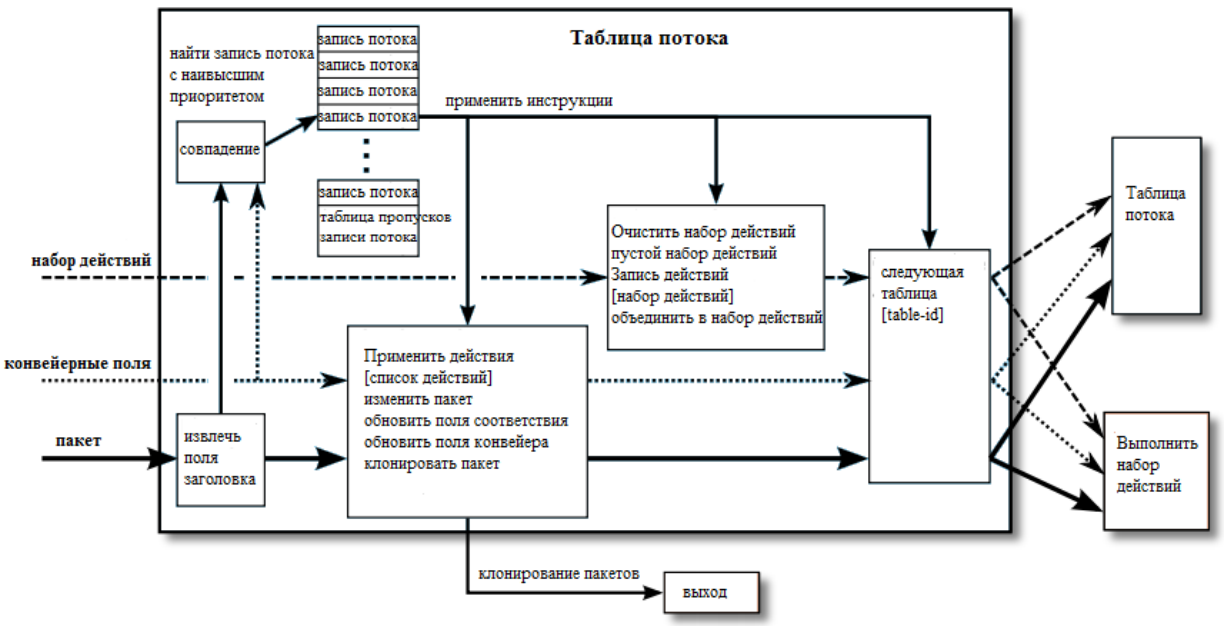

Рисунок 1.6 – Соответствие и выполнение инструкций в таблице потоков

Различаются несколько типов инструкций [7]:

– Apply-Actions – применяет конкретное действие немедленно, без каких-либо изменений в списке действий. Эта инструкция может быть использована для модификации пакета между двумя таблицами или для выполнения нескольких действий одного и того же типа.

– Clear-Actions – удаляет все действия из списка действий.

– Write-Actions – заносит действия в список действий. Если действие данного типа присутствует в списке, то оно перезаписывается, в противном случае добавляется.

– Write-Metadata metadata/mask – записывает значение метаданных в маске в поле метаданных. Маска указывает, какие биты регистра метаданных должны быть изменены.

– Goto-Table – указывает следующую таблицу в конвейере обработки. Идентификатор таблицы должен быть больше текущего идентификатора таблицы. Эта инструкция должна поддерживаться во всех таблицах потоков, кроме последней, коммутаторы OpenFlow с одной таблицей потоков не требуются для реализации этой инструкции.

1.3.6 Действия. Каждому потоку в таблице ассоциировано одно или несколько действий, которые выполняются, когда поток соответствует записи. Под действием понимается операция, которая изменяет пакет, список действий и/или процесс обработки конвейером, в зависимости от этого различаются несколько типов действий:

– Output – передать пакет на указанный порт (физический, логический, зарезервированный).
– Set-Queue – задать идентификатор очереди для пакета.
– Drop – отбросить пакет, принадлежавший потоку.
– Group – обработать пакет в соответствии с указанной группой.
– Push-Tag/Pop-Tag – работа с метками (VLAN, MPLS).
– Set-Field – изменить значение определенного поля заголовка пакета.
– Change-TTL – изменить значения поля TTL.

В OpenFlow-коммутаторе действия могут быть представлены не по отдельности, а списком действий. Список действий состоит из набора действий, связанных с пакетом, который хранится, пока идет его обработка таблицами потоков, и выполняется только при выходе пакета из конвейера обработки.

1.3.7 Групповая таблица. Группа OpenFlow – это абстракция, которая упрощает более сложные и специализированные пакетные операции, которые нелегко выполнить с помощью записи в таблице потоков. Каждая группа получает пакеты в качестве входных данных и выполняет любые действия OpenFlow с этими пакетами. Группа не может отправлять пакеты в другие таблицы потоков. Кроме того, ожидается, что пакеты были соответствующим образом сопоставлены до входа в группу, поскольку группы не поддерживают сопоставление пакетов. Группа – это просто механизмы для выполнения дополнительных действий или наборов действий.

Как показано на рисунке 1.7, преимущество группы заключается в том, что она содержит отдельные списки действий, и каждый отдельный список действий называется сегментом. Таким образом, говорят, что группа содержит список сегментов. Каждый сегмент или список сегментов может применяться к входящим пакетам (точное поведение зависит от типа группы) [10].

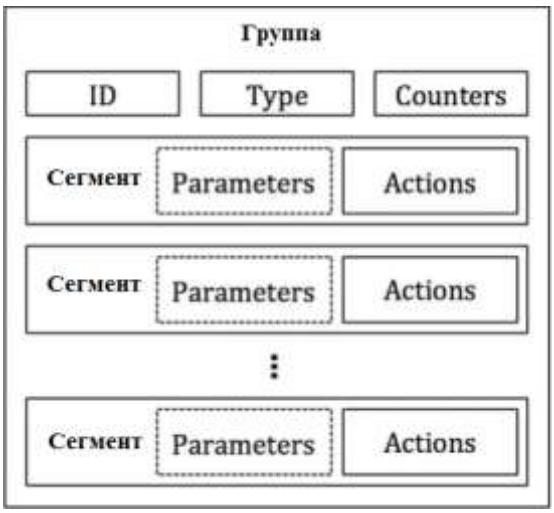

Рисунок 1.7 – Компоненты группы и сегмента

Идентификатор группы – это 32-битное поле, которое определяет номер группы. Счетчики работают аналогично счетчикам обычных таблиц пересылки, сохраняя статистику группы. Блоки действий представляют собой набор инструкций, связанный с сегментом. Это соответствует набору действий, которые изменяют заголовок пакета и/или отправляют пакет на выходной порт.

Существует четыре типа групп: ALL, SELECT, INDIRECT и FAST-FAILOVER.

Группа ALL показанная на рисунке 1.8 будет принимать любой полученный пакет в качестве входного и дублировать его, чтобы работать с ним независимо каждым сегментом в списке сегментов. Таким образом, группа ALL может использоваться для дублирования и последующей работы с отдельными копиями пакета, определяемыми действиями в каждом сегменте. В каждом сегменте могут быть различные действия, что позволяет выполнять разные операции с разными копиями пакета.

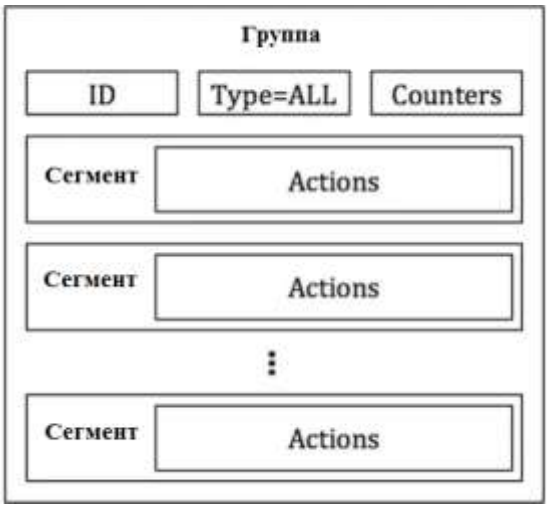

Рисунок 1.8 – Группа ALL

Группа SELECT в первую очередь предназначена для балансировки нагрузки. Как показано на рисунке 1.9, каждому сегменту в группе SELECT назначен вес, и каждый пакет, входящий в группу, отправляется в один сегмент. Алгоритм выбора сегмента не определен и зависит от реализации коммутатора, однако взвешенный циклический перебор - это, пожалуй, самый очевидный и простой выбор распределения пакетов по сегментам. Вес сегмента указывается в качестве специального параметра для каждого сегмента. Каждый сегмент в группе SELECT по-прежнему представляет собой список действий, поэтому любые действия, поддерживаемые OpenFlow, можно использовать в каждом сегменте, и как группа ALL, сегменты не обязательно должны быть единообразными.

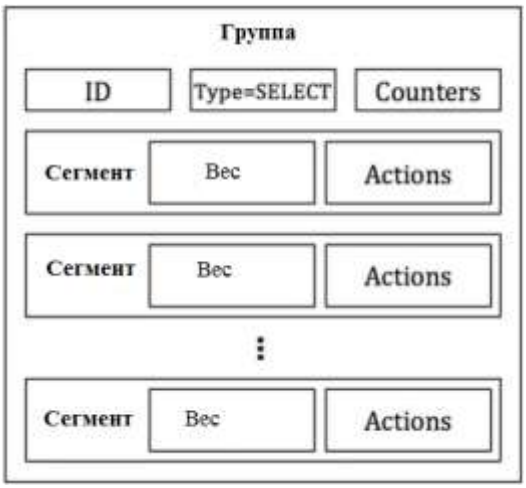

Рисунок 1.9 – Группа SELECT

Группа INDIRECT (рисунок 1.10) содержит только один сегмент, в котором все пакеты, полученные группой, отправляются в этот единственный сегмент. Цель группы INDIRECT – инкапсулировать общий набор действий, используемых многими потоками. Например, если потоки A, B и C совпадают по разным заголовкам пакетов, но имеют общий набор или подмножество действий, эти потоки могут отправлять пакеты в одну группу INDIRECT вместо того, чтобы дублировать список общих действий для каждого потока. Группа INDIRECT используется для упрощения развертывания OpenFlow и уменьшения объемов памяти, занимаемого набором аналогичных потоков.

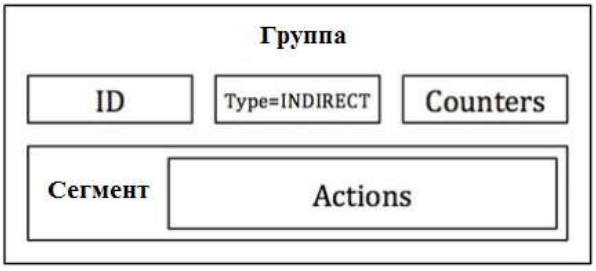

Рисунок 1.10 – Группа INDIRECT

Группа FAST-FAILOVER разработана специально для обнаружения и преодоления сбоев портов. Как и группы SELECT и ALL, группа FAST-FAILOVER как показано на рисунке 1.11 имеет список сегментов. В дополнение к этому списку действий каждый сегмент имеет порт наблюдения и/или группу наблюдения в качестве специального параметра. Наблюдаемый порт/группа будет отслеживать работоспособность или статус указанного порта/группы. Если считается, что жизнеспособность упала, сегмент не будет использоваться. Если жизнеспособность определена как повышенная, то сегмент можно использовать. Одновременно можно использовать только один сегмент, и используемый сегмент не будет изменен если живучесть текущего порта/группы отслеживания используемого сегмента не изменится с высокого на низкий. Когда такое событие происходит, группа FAST-FAILOVER быстро выберет следующий сегмент в списке сегментов с активным портом / группой наблюдения.

Нет никакой гарантии относительно времени перехода для выбора нового сегмента при возникновении сбоя. Время перехода зависит от времени поиска активного порта/группы наблюдения и от реализации коммутатора. Однако мотивация использования группы FAST-FAILOVER состоит в том, что это почти гарантированно будет быстрее, чем обращение к плоскости управления, чтобы обработать событие неработающего порта и вставить новый поток или набор потоков. С группами FAST-FAILOVER обнаружение сбоев канала и восстановление полностью происходит в плоскости данных.

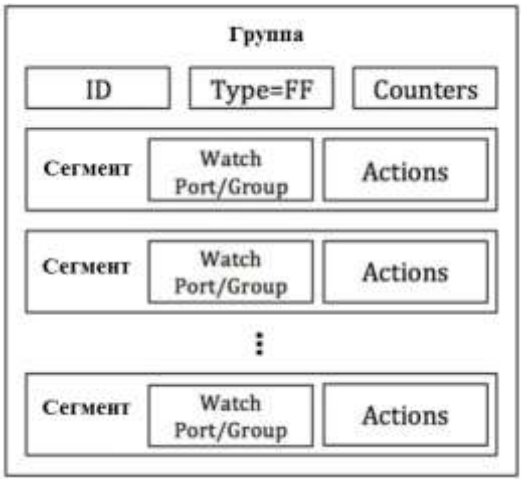

Рисунок 1.11 – Группа FAST-FAILOVER

1.4 Классификаторы протокола OpenFlow

Структура сопоставления OpenFlow используется для сопоставления пакетов с записями потока в таблицах. Каждый классификатор может состоять из заголовка и иметь от нуля до некоторого количества полей соответствия классификаторов, закодированных при помощи формата OXM TLV [11].

Поля соответствия классификатора описываются с использованием формата OpenFlow Extensible Match (OXM). В совокупности они образуют компактный формат type-length-value (TLV). Классификаторы OXM TLV могут быть от 5 до 259 (включительно) байт длинной.

Первые 4 байта каждого классификатора OXM TLV – заголовок (рисунок 1.12):

1) OXM class – спецификация OpenFlow разделяет два типа OXM классов:

– ONF reserved – классы, описываемые спецификацией и зарезервированные для последующих стандартизаций;

– ONF member – классы, используемые членами консорциума ONF. Позволяют членам консорциума использовать свои классификаторы и уникально идентифицироваться, благодаря им (например, Nicira).

2) OXM field – поле используется для определения конкретного поля соответствия в каждом типе класса (таблицы 1.2 и 1.3);

3) OXM hasmask – поле может принимать значения 0 или 1. Определяет наличие в поле данных OXM битовой маски;

4) OXM length – описывает длину полезной нагрузки OXM TLV в байтах, то есть все, что следует за 4-байтовым заголовком OXM TLV.

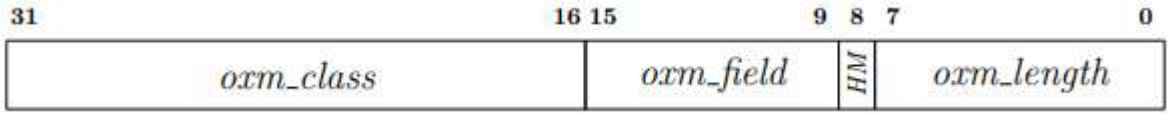

Рисунок 1.12 – Заголовок классификатора OXM TLV

1.4.1 OXM classes – reserved. Тип OXM-классов – Member – не описывается спецификациями и не требует разбора в рамках данной работы. Рассмотрим классы Reserved на примере протокола OpenFlow версии 1.5.1.

В спецификации протокола определены три класса в данном типе:

– experimentrt – класс используется для расширения стандартного набора полей соответствия в OpenFlow-коммутаторе;

– register – класс используется для хранения временных значений и информации, связанной с пакетом в процессе обработки конвейером;

– OpenFlow basic – основной класс классификаторов, который содержит базовый набор полей соответствия, однако не все из этого набора полей должны обязательно поддерживаться OpenFlow-коммутатором, но каждое обязательное match-поле должно поддерживаться не менее чем одной таблицей потоков. Кроме базовых полей соответствия основного класса таблицы потоков могут содержать match-поля (в необязательном порядке) классов experimenter и register. Поля соответствия класса OpenFlow basic разделяются на два типа. К первому типу относятся поля, которые сравниваются с данными, извлеченными из заголовка пакетов (header match

fields). Все они имеют разные размеры, обязательность/необязательность к реализации, маску и описание (таблица 1.2).

Таблица 1.2 – Поля соответствия класса OpenFlow basic (header match fields)

Math-поле	Размер, бит	Маска	Описание
1 OFPXMT_OFB_ETH_DST	48	Да	MAC-адрес получателя
2 OFPXMT_OFB_ETH_SRC	48	Да	MAC-адрес отправителя
3 OFPXMT_OFB_TYPE	16	Нет	Ethernet type
4 OFPXMT_OFB_VLAN_VID	12+1	Да	VLAN-ID по стандарту 802.1Q
5 OFPXMT_OFB_VLAN_PCP	3	Нет	VLAN-PCP по стандарту 802.1Q
6 OFPXMT_OFB_IP_DSCP	6	Нет	Diff Serv Code Point (DSCP)
7 OFPXMT_OFB_IP_ECN	2	Нет	ECN бит в заголовке IP
8 OFPXMT_OFB_IP_PROTO	8	Нет	IPv4 или IPv6-номер протокола
9 OFPXMT_OFB_IPV4_SRC	32	Да	IPv4-адрес отправителя
10 OFPXMT_OFB_IPV4_DSP	32	Да	IPv4-адрес получателя
11 OFPXMT_OFB_TCP_SRC	16	Нет	TCP-порт отправителя
12 OFPXMT_OFB_TCP_DST	16	Нет	TCP-порт получателя
13 OFPXMT_OFB_TCP_FLAGS	12	Нет	TCP-флаги
14 OFPXMT_OFB_UDP_SRC	16	Нет	UDP-порт отправителя
15 OFPXMT_OFB_UDP_DST	16	Нет	UDP-порт получателя
16 OFPXMT_OFB_SCTP_SRC	16	Нет	SCTP-порт отправителя
17 OFPXMT_OFB_SCTP_DST	16	Нет	SCTP-порт получателя
18 OFPXMT_OFB_ICMPV4_TYPE	8	Нет	ICMP-тип
19 OFPXMT_OFB_ICMPV4_CODE	8	Нет	ICMP-код
20 OFPXMT_OFB_ARP_OP	16	Нет	ARP-код отправителя/получателя
21 OFPXMT_OFB_ARP_SPA	32	Да	IPv4-адрес отправителя в payload протокола ARP
22 OFPXMT_OFB_ARP_TPA	32	Да	IPv4-адрес получателя в payload протокола ARP
23 OFPXMT_OFB_ARP_SHA	48	Да	MAC-адрес отправителя в payload протокола ARP
24 OFPXMT_OFB_ARP_THA	48	Да	MAC-адрес получателя в payload протокола ARP
25 OFPXMT_OFB_IPV6_SRC	128	Да	IPv6-адрес отправителя
26 OFPXMT_OFB_IPV4_DST	128	Да	IPv6-адрес получателя
27 OFPXMT_OFB_IPV4_FLABEL	20	Да	IPv6-метка потока
28 OFPXMT_OFB_ICMPV6_TYPE	8	Нет	ICMPv6-тип
29 OFPXMT_OFB_ICMPV6_CODE	8	Нет	ICMPv6-код
30 OFPXMT_OFB_IPV6_ND_TAR-GET	128	Нет	IPv6, сообщение NDM, адрес получателя
31 OFPXMT_OFB_IPV6_ND_SLL	48	Нет	IPv6, сообщение NDM, source link-layer address option
32 OFPXMT_OFB_IPV6_ND_TLL	48	Нет	IPv6, сообщение NDM, target link-layer address option
33 OFPXMT_OFB_MPLS_LABEL	20	Нет	MPLS-метка
34 OFPXMT_OFB_MPLS_TC	3	Нет	MPLS-класс трафика

Продолжение таблицы 1.2. 2

35 OFPXMT_OFB_MPLS_BOS	1	Нет	MPLS флаг «дно стека»
36 OFPXMT_OFB_PBB_ISID	24	Да	Технология PBB, идентификатор сервиса I-SID
37 OFPXMT_OFB_IPV6_EXTHDR	9	Да	IPv6 Extension Header pseudo-field
38 OFPXMT_OFB_PBB_UCA	1	Нет	Технология PBB, поле UCA

Второй тип полей соответствия содержит поля, которые используются в процессе обработки конвейером (pipeline match fields) и никаким образом не взаимодействуют с заголовками пакета (таблица 1.3).

Таблица 1.3 – Поля соответствия класса OpenFlow basic (pipeline match fields)

Math-поле	Размер, бит	Маска	Описание
OFPXMT_OFB_IN_PORT	32	Нет	Входящий порт. Номер логического или физического входящего порта
OFPXMT_OFB_IN_PHY_PORT	32	Нет	Физический порт. Соответствие физического и логического порта, когда сообщение приходит на логический порт
OFPXMT_OFB_METADATA	64	Да	Используется для передачи информации между таблицами
OFPXMT_OFB_TUNNEL_ID	64	Да	Метаданные, ассоциированные с логическим портом
OFPXMT_OFB_ACTSET_OUT-PUT	32	Нет	Output port from action set Metadata
OFPXMT_OFB_PACKET_TYPE	16+16	Нет	Packet type – canonical header type of outermost header

1.5 Постановка задачи исследования

Современное состояние сетей показало, что потенциал роста производительности, пропускной способности на основе традиционных технологий практически исчерпан. Это связано с ростом количества пользователей и трафика, с трудностями настройки сети и управления потоками в ней, особенно с учетом новых потребностей в качестве сервисов для высокоскоростных глобальных сетей и сетей центров обработки данных. Влияющим фактором является также рост потребностей в виртуализации сетей, т.е. отображения нескольких логически изолированных сетей с независимыми политиками обслуживания на общий набор сетевых ресурсов. Такое неудовлетворительное состояние дел может измениться из-за появления принципиально нового подхода, называемого программно-конфигурируемыми сетями. Следование такому подходу позволит повысить производительность сети, удобство конфигурирования, виртуализацию, но требует дополнительных исследований и разработок. В частности, в области

организации сетевых коммутаторов, программных приложений для управления сетью и платформ для их выполнения.

Целью исследования является определение характеристик качества работы программно-конфигурируемых сетей с использованием протокола OpenFlow.

Для достижения поставленной цели в работе решаются следующие задачи:

– анализ алгоритма работы сети SDN;
– исследование создания записей в таблице потоков;
– разработка модели для анализа величины сквозной задержки;
– исследование величины задержки передаваемого трафика в сети SDN;
– оценка влияния обновления TCAM на величину задержки.

2 Исследование работы программно-конфигурируемой сети

2.1 Инструменты исследования

2.1.1 VirtualBox. VirtualBox – программный продукт виртуализации для операционных систем Microsoft Windows, Linux, macOS и других. Виртуализация позволяет запускать несколько операционных систем на одном компьютере: при этом каждый из экземпляров таких операционных систем работает со своим набором логических ресурсов (процессорных, оперативной памяти, устройств хранения), предоставлением которых из общего пула, доступного на уровне оборудования, управляет хостовая операционная система – гипервизор. Гипервизор – программа или аппаратная схема, позволяющая одновременное, параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере. Гипервизор также обязан предоставлять работающим под его управлением на одном хост-компьютере ОС средства связи и взаимодействия между собой (например, через обмен файлами или сетевые соединения) так, как если бы эти ОС выполнялись на разных физических компьютерах [12].

2.1.2 Эмулятор сети Mininet. Сетевой эмулятор Mininet позволяет быстро развернуть сеть, состоящую из различного числа виртуальных хостов, коммутаторов, контроллеров и каналов между ними [13].

Данное программное обеспечение использует технологии виртуализации, встроенные в ядро ОС Linux. Это позволяет запускать на виртуальных хостах стандартные сетевые утилиты Linux. Эмулируемые с помощью Mininet коммутаторы имеют поддержку OpenFlow, что позволяет использовать данный продукт для исследования и обучения принципам построения программно-конфигурируемых сетей.

Большая часть кода Mininet написана с использованием языка Python. Топологии эмулируемых сетей, а также параметры сетевых устройств также могут быть описаны на языке Python. Кроме того, существуют встроенные

шаблоны топологий, например, линейная топология или топология «дерево». Также для управления эмулируемой сетью существует встроенный интерпретатор команд.

2.1.3 Контроллер сети SDN OpenDaylight. OpenDaylight (ODL) – это платформа контроллеров SDN с открытым исходным кодом для настройки и автоматизации сетей любого размера и масштаба. Проект OpenDaylight возник из движения SDN с четким упором на программируемость сетей. С самого начала он был разработан как основа для коммерческих решений, которые предназначены для различных вариантов использования в существующих сетевых средах [14].

Платформа ODL разработана, чтобы предоставить последующим пользователям и поставщикам решений максимальную гибкость в создании контроллера, отвечающего их потребностям. ODL позволяет любому пользователю использовать услуги, созданные другими, писать и включать свои собственные и поделиться своей работой с другими. ODL включает поддержку самого широкого набора протоколов на любой платформе SDN – OpenFlow, OVSDB, NETCONF, BGP и многих других, которые улучшают программируемость современных сетей и решают ряд потребностей пользователей.

2.1.4 Анализатор сетевого трафика Wireshark. Для исследования процесса обмена трафиком между узлами сети используется анализатор сетевых пакетов Wireshark. Wireshark позволяет перехватывать и распознавать сообщения протокола OpenFlow. Также сетевой анализатор может определить структуру сообщения и передаваемые в нем данные [15].

2.2 Исследование алгоритма работы сети SDN

Для анализа работы протокола OpenFlow были использованы две виртуальные машины в программе VirtualBox. Одна запускает эмулированную сеть в Mininet, а другая - контроллер OpenDaylight (рисунок 2.1). Обе виртуальные машины подключены к сети только для хоста, чтобы они могли общаться друг с другом.

В виртуальной машине под управлением операционной системы Ubuntu, был установлен контроллер SDN OpenDaylight и следующие функции (рисунок 2.2):

– odl-restconf – возможность использования restconf API для управления контроллером;

– odl-l2switch-switch – функция коммутатора l2, необходимая для коммутации пакетов через Mininet Open vSwitch, поскольку Open vSwitch является коммутатором openflow и не может пересылать пакеты без контроллера. l2switch – это модуль на контроллере, который выполняет функцию обучающего коммутатора для Open vSwitch. Контроллер автоматически отправляет потоки в Open vSwitch;

– odl-mdsal-apidocs – функция автоматического создания документации API;

– odl-dluxapps-applications – функция для графического интерфейса OpenDaylight.

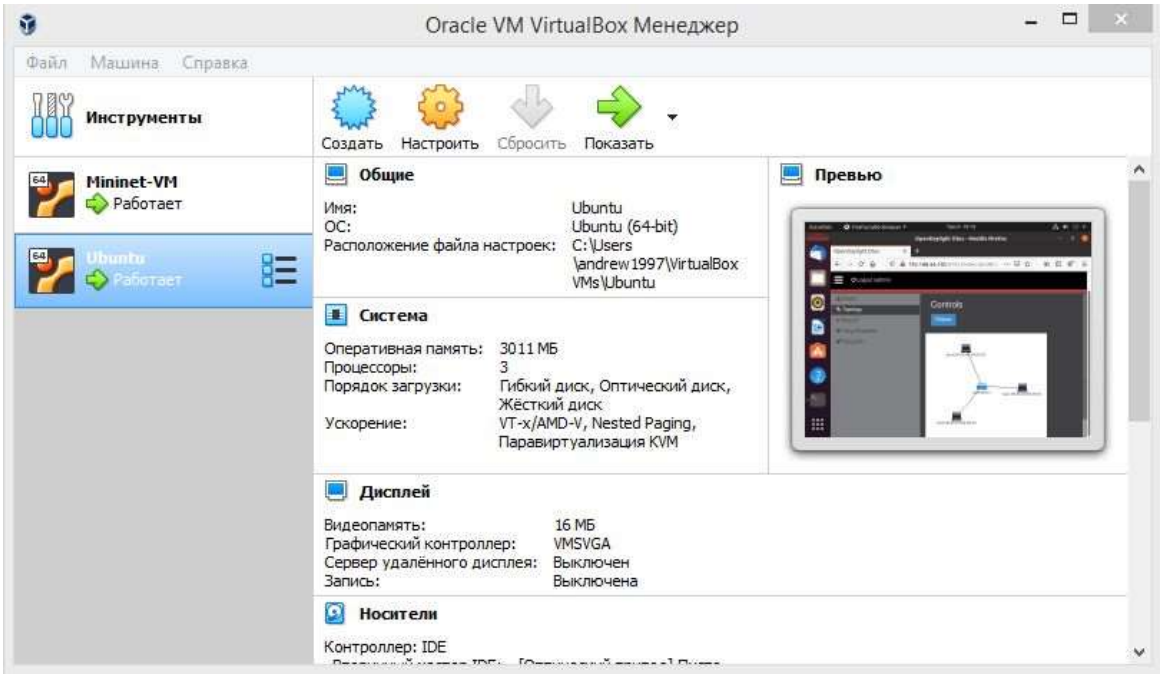

Рисунок 2.1 – Использование VirtualBox

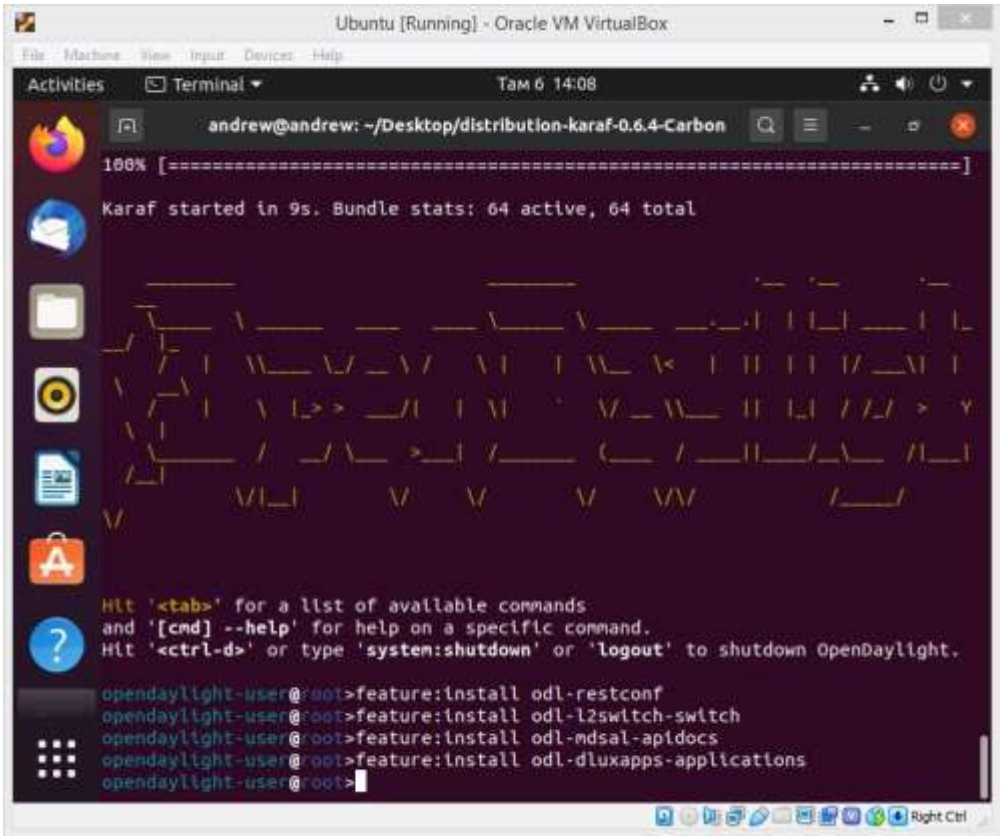

Рисунок 2.2 – Установка контроллера OpenDaylight

В виртуальной машине Mininet создана сеть из одного коммутатора Open vSwitch, который поддерживает протокол OpenFlow (ip адрес 192.168.56.101) и трех подключенных к нему хостов под управлением контроллера OpenDaylight (ip адрес 192.168.56.102), в результате ввода команды, приведенной на рисунке 2.3.

Параметры команды:

– topo – опция определения топологии сети. В нашем случае single,3 означает, что к сети будет подключен один коммутатор и 3 хоста;

– mac – MAC-адрес на каждом хосте будет установлен на простое число;

– controller – опция для определения контроллера SDN. В нашем случае контроллер OpenDaylight установлен на другой виртуальной машине, поэтому мы используем опцию remote с IP-адресом виртуальной машины (192.168.56.102), на котором установлен OpenDaylight;

– switch – опция для определения виртуального коммутатора. Используется Open vSwitch с Openflow версии 1.3.

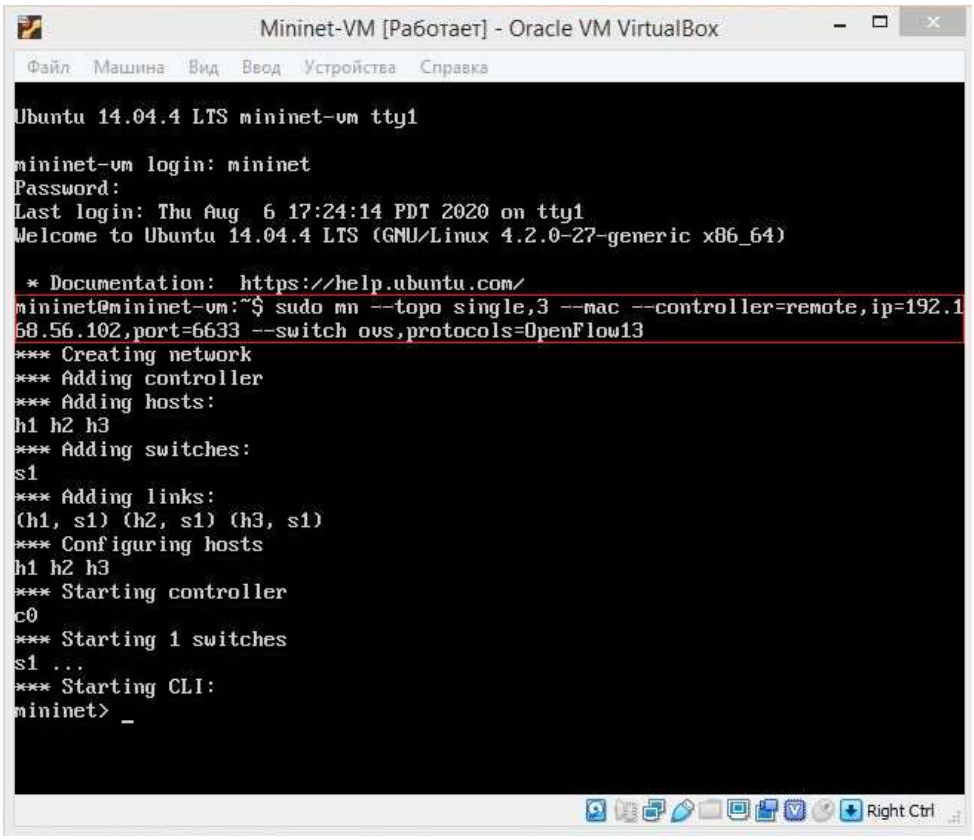

Рисунок 2.3 – Создание сети

Для просмотра топологии сети, а именно сопоставление портов коммутатора и хостов используется команда net (рисунок 2.4).

Рисунок 2.4 – Использование команды net

Для просмотра информации об узлах, коммутаторах и контроллерах в моделируемой сети используется команда dump (рисунок 2.5).

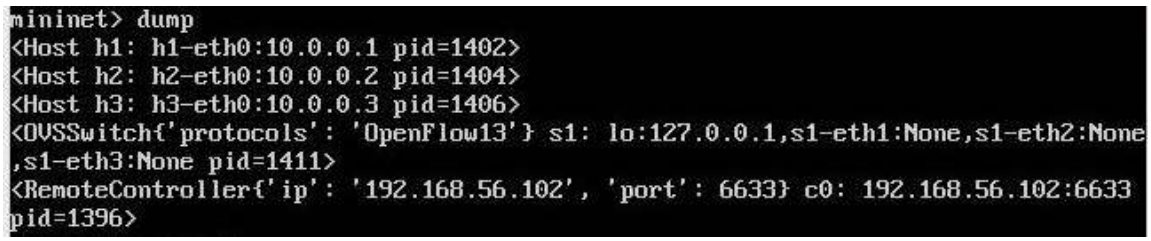

Рисунок 2.5 – Использование команды dump

Для проверки связи между всеми хостами в сети используется команда pingall. Команда pingall будет запускать команду ping на каждом хосте и проверять связь с каждым другим хостом, а затем сообщит о результатах в интерфейсе командной строки Mininet (рисунок 2.6).

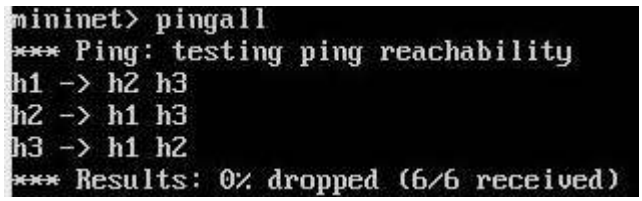

Рисунок 2.6 – Использование команды pingall

Пропингуя все узлы, убедились, что контроллер OpenDaylight работает. Теперь отобразим нашу топологию сети в графическом интерфейсе OpenDaylight. Он работает на виртуальной машине, поэтому IP-адрес – 192.168.56.102, а порт, определенный приложением – 8181 (рисунок 2.7).

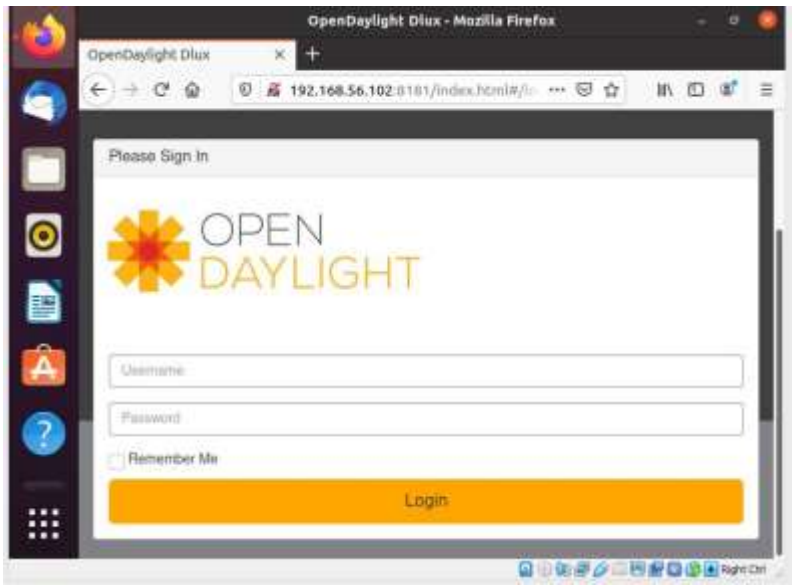

Рисунок 2.7 – Графический интерфейс OpenDaylight

Рисунок 2.8 демонстрирует сгенерированную в Mininet топология сети с помощью графического интерфейса контроллера OpenDaylight.

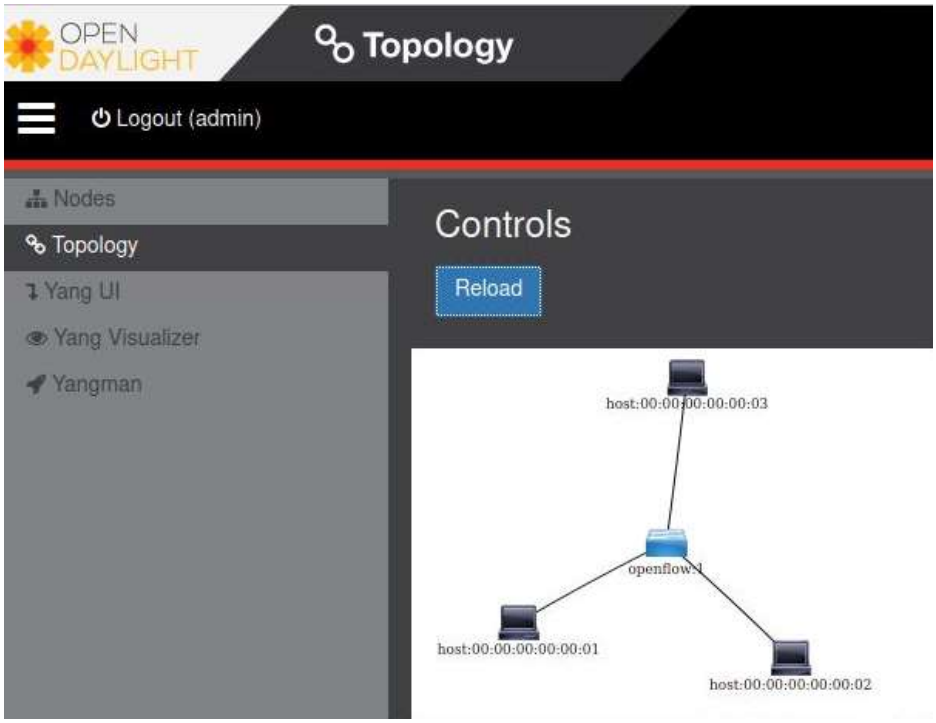

Рисунок 2.8 – Вкладка топология в контроллере OpenDaylight

Во вкладке «Узлы» можно просмотреть информацию об узлах в сети (рисунок 2.9). Видно, что хост 1 подключен к s1-eth1, хост 2 подключен к s1-eth2, хост 3 подключен к s1-eth3. Обратите внимание, что существует локальный интерфейс управления, который является локальной плоскостью управления коммутатора, используемой для отправки трафика непосредственно на плоскость управления.

Рисунок 2.9 – Вкладка «Узлы»

Чтобы просмотреть сообщения OpenFlow, которыми обмениваются контроллер и коммутатор в сети, используется Wireshark.

Каждое сообщение OpenFlow начинается с одинаковой структуры заголовка. Эта фиксированная структура выполняет три роли, которые не зависят от используемой версии OpenFlow. Во-первых, поле версии указывает версию OpenFlow, которой принадлежит это сообщение. Во-вторых, поле lenght указывает, где это сообщение закончится в потоке байтов, начиная с первого байта заголовка. В-третьих, xid или идентификатор транзакции - это уникальное значение, используемое для сопоставления запросов и ответов. Поле типа, которое указывает, какой тип сообщения присутствует и как интерпретировать полезную нагрузку, зависит от версии (рисунок 2.10).

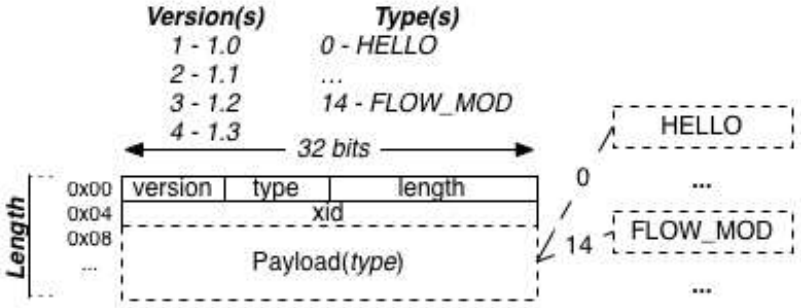

Рисунок 2.10 – Структура заголовка сообщения OpenFlow

Используя фильтр отображения Wireshark для openflow_v4, что означает OpenFlow версии 1.3, можно просмотреть пойманные пакеты (рисунок 2.11).

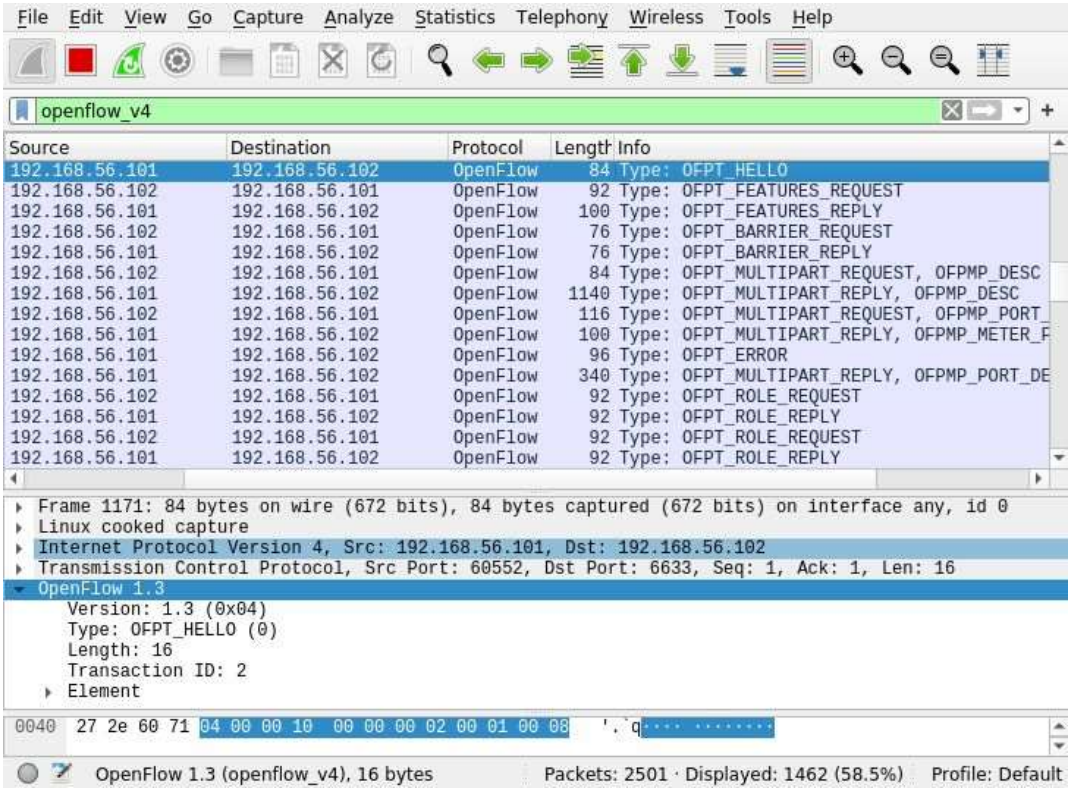

Рисунок 2.11 – Фильтрация пакетов OpenFlow

На рисунке 2.11 видно, что первый пойманный пакет - это сообщение HELLO, отправленное от коммутатора (192.168.56.101) к контроллеру (192.168.56.102). Он используется для согласования версии протокола OpenFlow. Если согласование версии протокола не удается, отправляется сообщение об ошибке с типом Hello Failed и кодом «Несовместимый». Это инициализация канала OpenFlow, который является каналом данных для протокола OpenFlow между контроллером и коммутатором.

Следующий тип сообщения - это OFPT_FEATURES_REQUEST от контроллера к коммутатору (рисунок 2.12). Когда между коммутатором и контроллером устанавливается транспортный канал OpenFlow, первое действие - определение функции. Контроллер отправит на коммутатор Feature Request по транспортному каналу. Запрос функции состоит только из сообщения заголовка OpenFlow со значением Feature Request, установленным в поле типа. Затем коммутатор ответит на этот запрос своими возможностями.

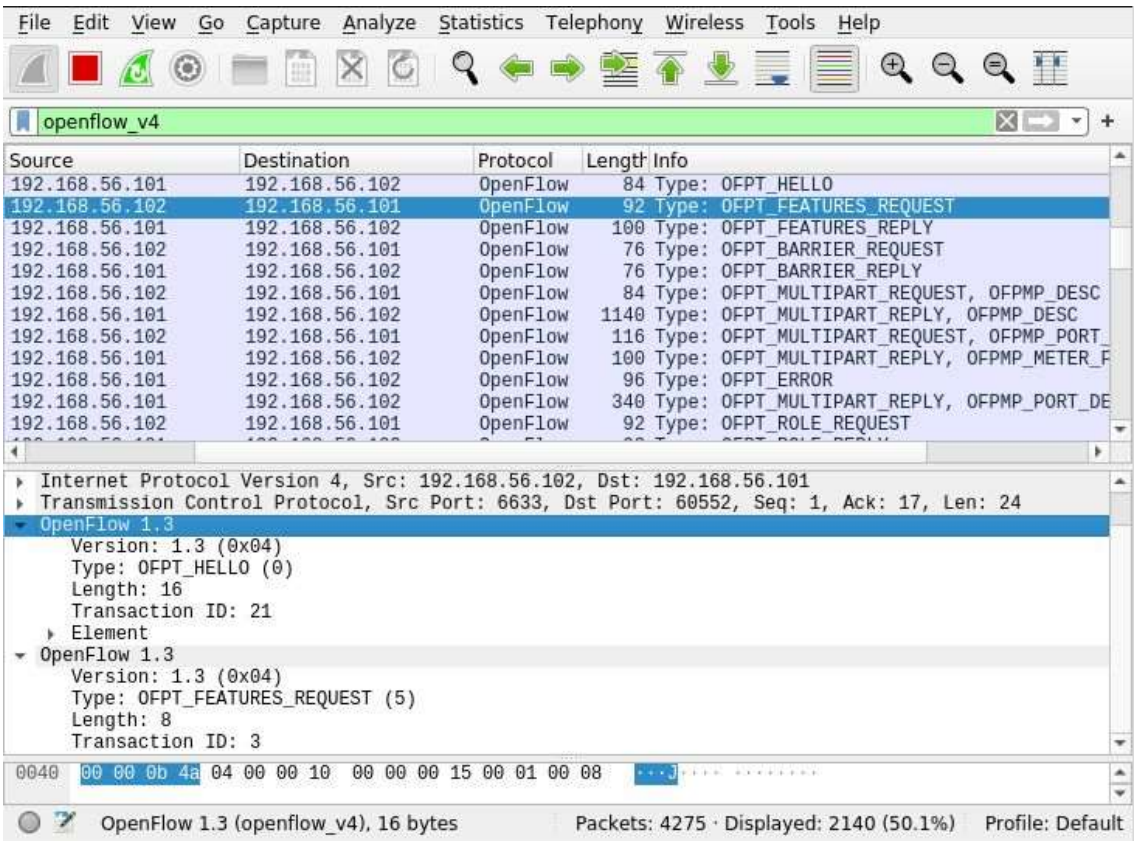

Рисунок 2.12 – Сообщение OFPT_FEATURES_REQUEST

OFPT_FEATURES_REPLY - это ответ коммутатора контроллеру с перечислением его возможностей (рисунок 2.13). Datapath-id - это 64-битное поле, которое следует рассматривать как аналог MAC-адреса моста Ethernet-коммутаторов, его уникальный идентификатор для конкретного управляемого конвейера обработки пакетов. Поле n_buffers определяет, сколько пакетов коммутатор может поставить в очередь для действий Packet_In (захват и пересылка на контроллер). Количество таблиц в коммутаторе фиксируется n_tables. Capabilities показывает какие возможности коммутатора поддерживаются коммутатором.

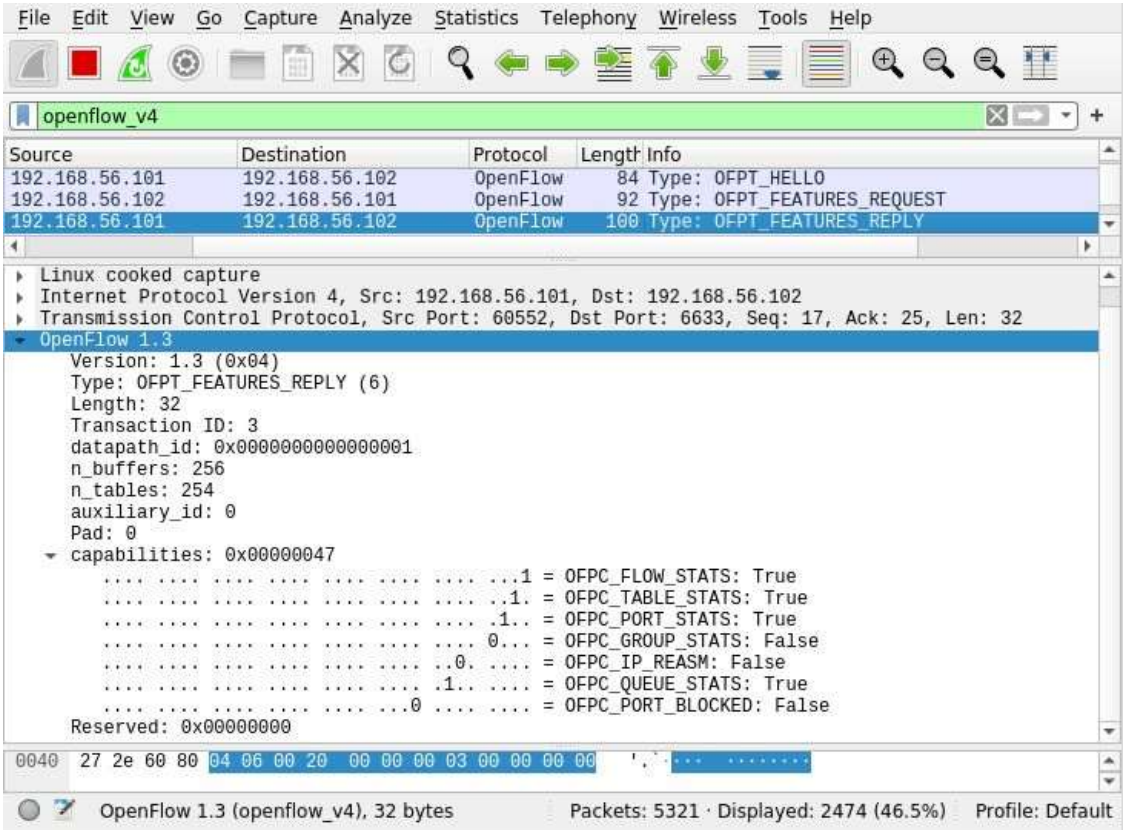

Рисунок 2.13 – Сообщение OFPT_FEATURES_REPLY

Контроллер может запросить у коммутатора состояние по каналу OpenFlow с помощью сообщения OFPT_MULTIPART_REQUEST. Типы сообщений, обрабатываемые этим сообщением, включают различную статистику (FLOW / TABLE / PORT / QUEUE / METER и т.д.) или функции описания (METER_CONFIG / TABLE_FEATURES / PORT_DESC и т.д.).

В сообщении OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC расположена информация о портах коммутатора, отправленная на контроллер. В это сообщение включено все, от типа до статуса и скорости.

В сообщении OFPT_MULTIPART_REPLY, OFPMP_DESC, можно увидеть такие атрибуты, как производитель и версии аппаратного / программного обеспечения (рисунок 2.14).

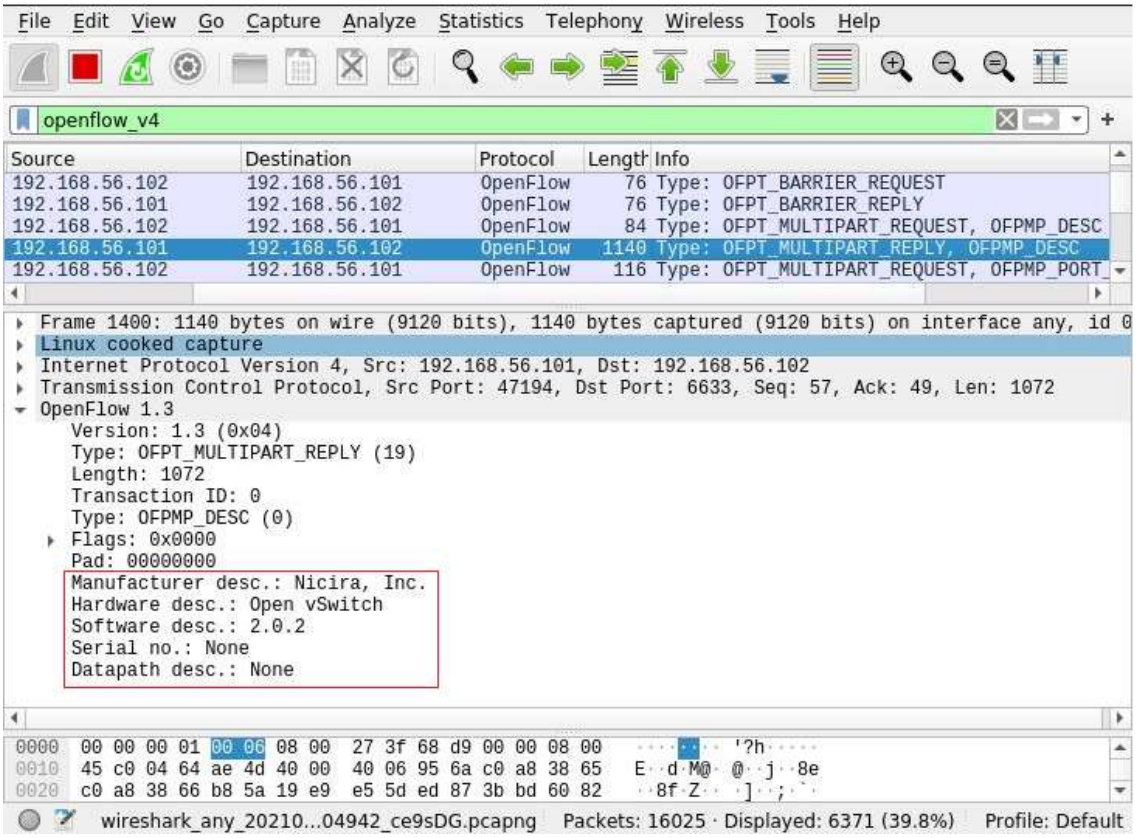

Рисунок 2.14 – Сообщение OFPT_MULTIPART_REPLY, OFPMP_DESC

Сообщение Flow_Mod позволяет контроллеру изменять состояние коммутатора OpenFlow (рисунок 2.15). Это сообщение начинается со стандартного заголовка и сопровождается файлом cookie, который является непрозрачным полем, установленным контроллером, его маской, идентификатором таблицы и командой, определяющей тип модификации таблицы потоков. Далее следует idle_timeout, hard_timeout и приоритет. Idle_timeout – это количество секунд, по истечению которых запись потока удаляется из таблицы потоков, поскольку ни один из пакетов не соответствует ей. Hard_timeout – это количество секунд, по истечению которых запись потока удаляется из таблицы потоков независимо от того, совпадают ли пакеты с ней или нет. Приоритет подразумевает порядок совпадений, которые перекрываются с более высокими числами, представляющими более высокий приоритет. Далее в Flow_Mod идут buffer_id, out_port, out_group и флаги. Buffer_id – это идентификатор буферизованного пакета. В конце добавляются совпадение и инструкции.

Рисунок 2.15 – Структура сообщения Flow_Mod

Сообщение Packet_In - это способ для коммутатора отправить захваченный пакет контроллеру. Это может произойти из-за явного действия в результате совпадения, запрашивающего такое поведение, или из-за пропуска таблицы.

Сообщение Packet_In состоит из заголовка, за которым следует идентификатор буфера. Идентификатор буфера - это уникальное значение, используемое для отслеживания буферизованного пакета. Длина захваченного пакета указывается total_len. Поле причины указывает, почему пакет был захвачен и отправлен. Table_id – это id таблицы которая просматривалась. Cookie используется для идентификации конкретного потока, соответствующего этому пакету. Наконец, захваченная часть пакета начинается сразу за pad (рисунок 2.16).

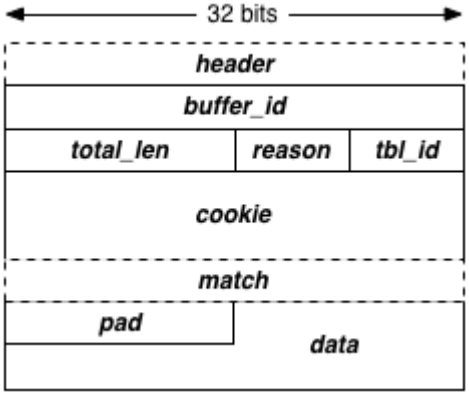

Рисунок 2.16 – Структура сообщения PacketIn

Был инициирован пинг с h1 на h2 в Mininet (рисунок 2.17).

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.489 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.062 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.062/0.275/0.489/0.214 ms
```

Рисунок 2.17 – Пинг с h1 на h2

На рисунке 2.18 показан вывод сообщения PACKET_IN от коммутатора к контроллеру, поскольку для входящего пакета на коммутаторе не найдено совпадений в наборе записей и h1 еще не знает, как добраться до h2. Это сообщение ARP, инкапсулированное OpenFlow через TCP. Контроллер с помощью OpenFlow-сообщения типа контроллер-коммутатор Flow_mod дает команду коммутатору запомнить расположение узла h1. Следующим шагом контроллер рассылает ARP-запрос, инкапсулированный в сообщение Packet_out, только на те порты коммутатора, к которым подключены узлы.

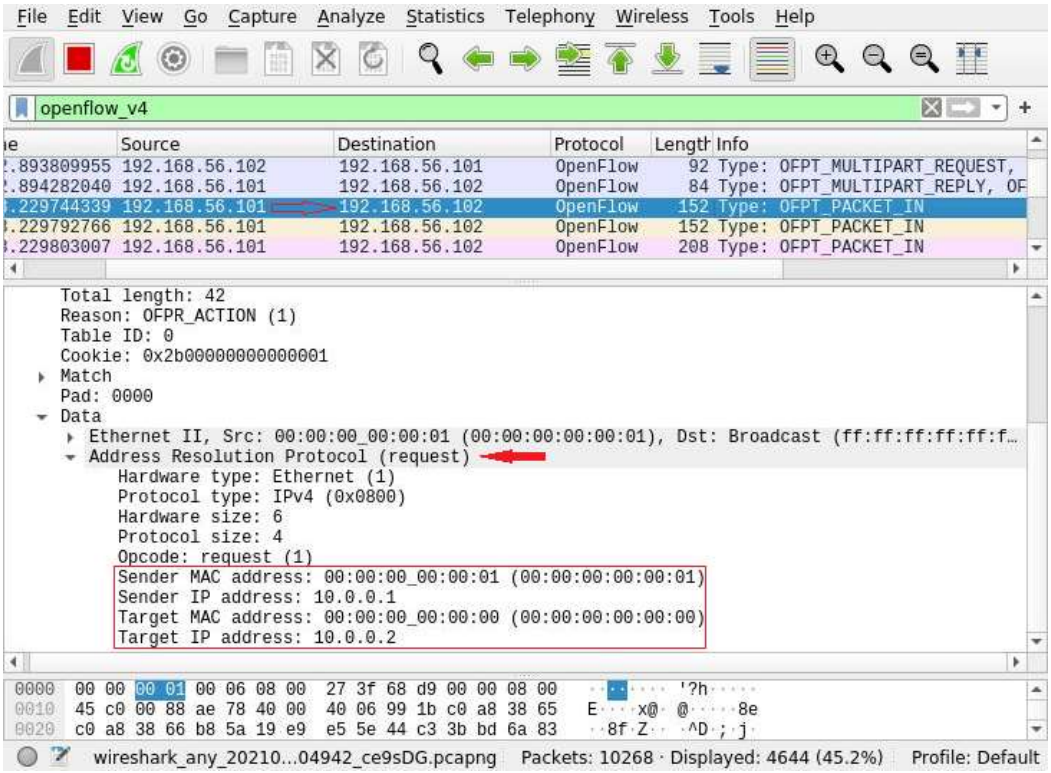

Рисунок 2.18 – ARP запрос от h1 к h2

Узел h2, обнаруживший совпадение искомого MAC-адреса с собственным, отправляет в сеть ARP-ответ, который ретранслируется коммутатором на контроллер. Контроллер теперь знает расположение узла h2

и устанавливает соответствующее правило в таблице потоков коммутатора, после чего ARP-ответ отправляется узлу h1 (рисунок 2.19).

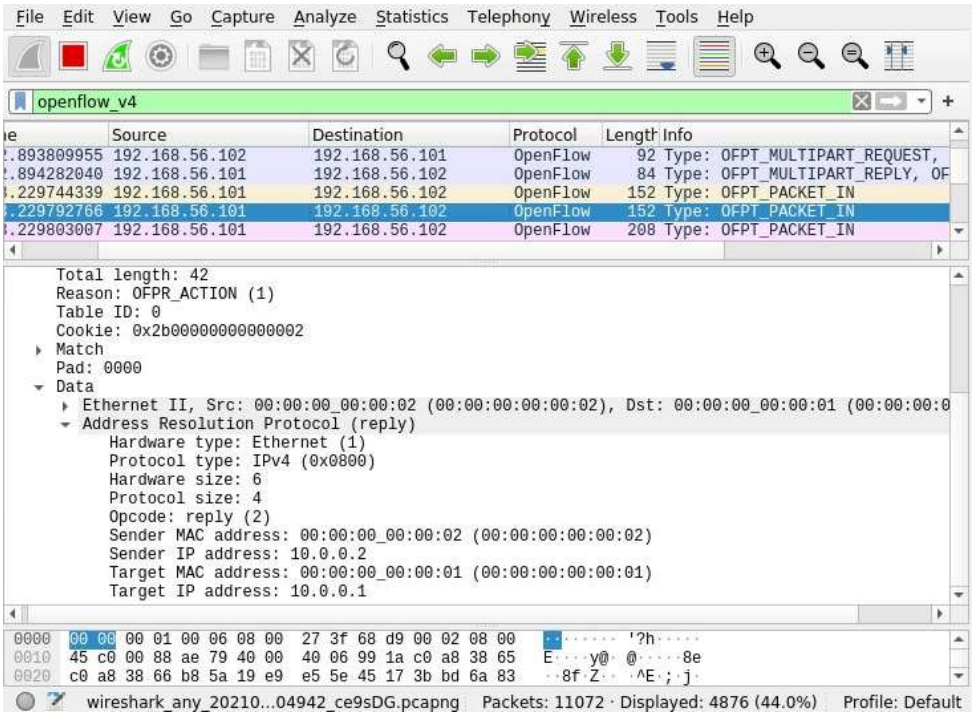

Рисунок 2.19 – ARP ответ от h2

После добавления записи в таблицу потоков, узел h1 может отправить ICMP пакет узлу h2 (рисунок 2.20).

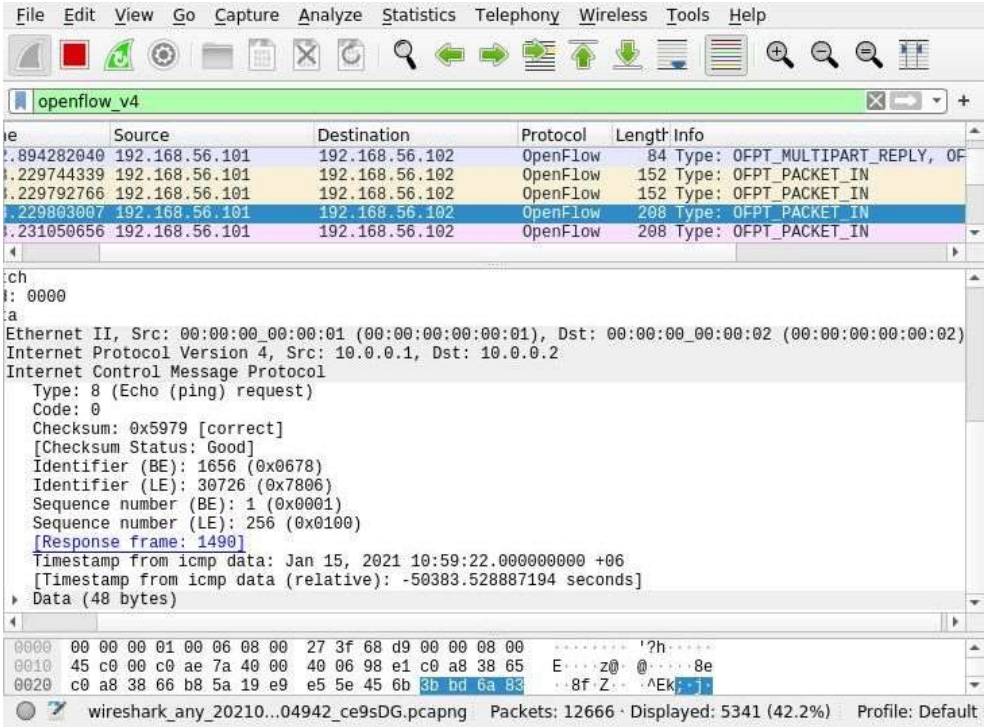

Рисунок 2.20 – ICMP запрос

Далее узлу h2 требуется отправить ответный ICMP пакет узлу h1 (рисунок 2.21).

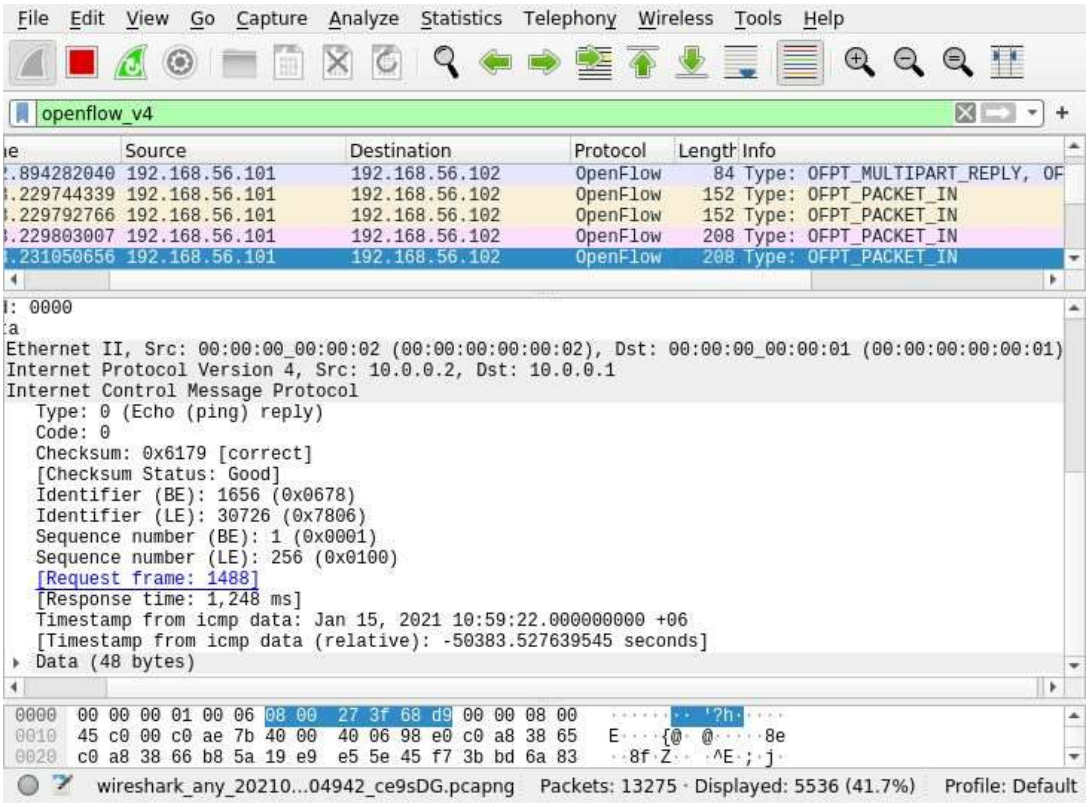

Рисунок 2.21 – ICMP ответ

2.3 Создание записей в таблице потоков

Для исследования воздействия записей потока на трафик в виртуальной машине mininet создана топология с одним коммутатором и 3-мя хостами в результате ввода команды (sudo mn --topo=single,3 --controller=none --mac), показанной на рисунке 2.22.

Параметр -- controller=none исключает OpenFlow-контроллер из топологии в данном примере. Параметр topo определяет топологию сети, например, если изменим значение параметра (--topo=2,3), то топология будет состоять из двух OpenFlow-коммутаторов и каждый из коммутаторов будет иметь по 4 хоста. Команда dump позволяет вывести информацию о всех узлах сети и хостах с параметрами интерфейсов. Команда net демонстрирует как соединены хосты и узлы.

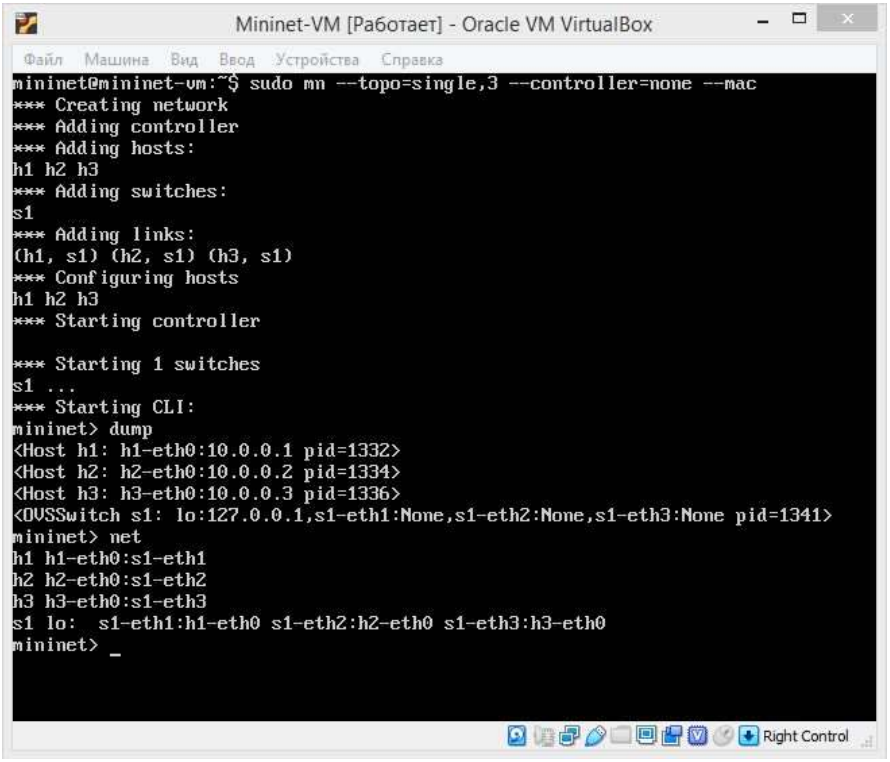

Рисунок 2.22 – Создание простейшей топологии сети с OpenFlow-коммутатором

Команда (sh ovs-ofctl show s1) позволяет просмотреть параметры OpenFlow-коммутатора (рисунок 2.23).

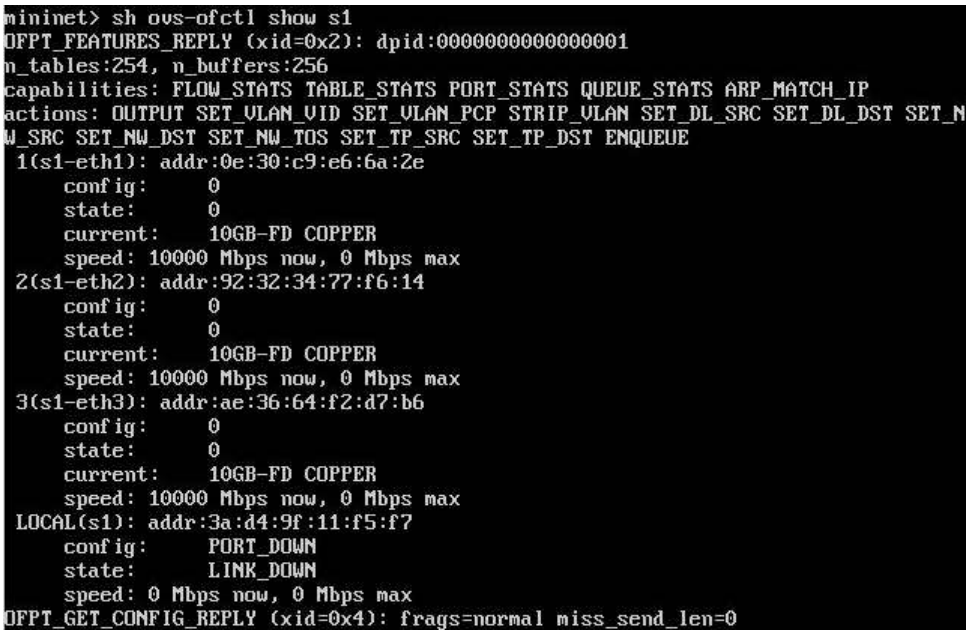

Рисунок 2.23 – Параметры OpenFlow-коммутатора

Для проверки связи между хостами нужно добавить OpenFlow-записи. Из рисунка 2.24 видно, что если не задать ни одного правила на OpenFlow-

коммутаторе, то связи в локальной сети не будет. Для того, чтобы создать стандартную OpenFlow-запись, нужно воспользоваться командой (sh ovs-ofctl add-flow s1 action=normal).

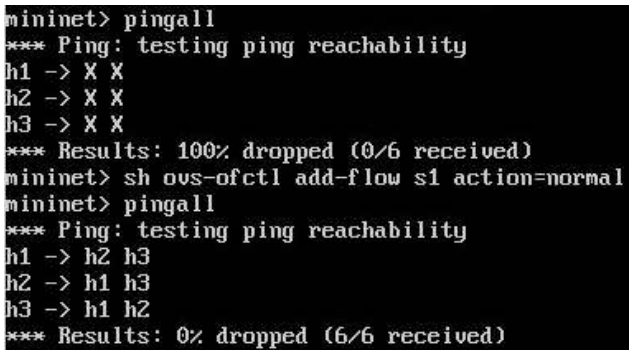

Рисунок 2.24 – Проверка связи между хостами

Для того чтобы получить данные о записях потока, используется следующая команда (sh ovs-ofctl dump-flows s1), рисунок 2.25.

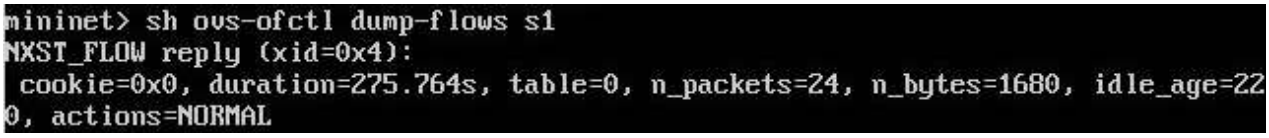

Рисунок 2.25 – Таблица потоков

Чтобы удалить все записи для потоков, необходимо воспользоваться командой (sh ovs-ofctl del-flows s1).

Создадим записи в таблице потоков, которые будут указывать коммутатору на какой из портов отправлять трафик, который пришел с указанного порта. Такая запись называется Layer 1 matching (рисунок 2.26).

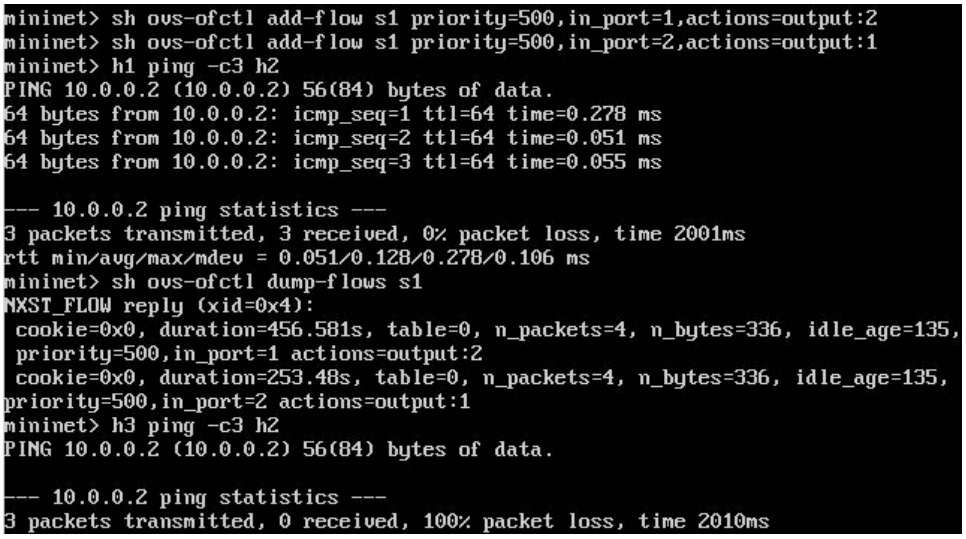

Рисунок 2.26 – Создание записей в таблице потоков

Теперь OpenFlow-коммутатор может применить определенные ему правила при передаче трафика с порта 1 на порт 2 и наоборот. Как видно из рисунка 2.26, связь с хостом h3 нет как раз из-за отсутствия правил для порта, к которому он подключен. После применения команды (sh ovs-ofctl dump-flows s1) видно, как изменилась таблица потоков OpenFlow-коммутатора.

Управлять правилами в таблице потоков позволяет поле приоритета (priority). Посмотрим, как влияет добавление правила с большим значением priority на передачу трафика, а затем удалим его, используя параметр --strict (рисунок 2.27).

```
mininet> sh ovs-ofctl add-flow s1 priority=32768,actions=drop
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 1999ms

mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1375.973s, table=0, n_packets=4, n_bytes=336, idle_age=105
4, priority=500,in_port=1 actions=output:2
 cookie=0x0, duration=1172.872s, table=0, n_packets=4, n_bytes=336, idle_age=105
4, priority=500,in_port=2 actions=output:1
 cookie=0x0, duration=162.691s, table=0, n_packets=6, n_bytes=420, idle_age=99,
actions=drop
mininet> sh ovs-ofctl del-flows --strict
ovs-ofctl: 'del-flows' command requires at least 1 arguments
mininet> sh ovs-ofctl del-flows s1 --strict
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1765.764s, table=0, n_packets=4, n_bytes=336, idle_age=144
4, priority=500,in_port=1 actions=output:2
 cookie=0x0, duration=1562.663s, table=0, n_packets=4, n_bytes=336, idle_age=144
4, priority=500,in_port=2 actions=output:1
```

Рисунок 2.27 – Приоритет записей в таблице потоков

Следующее правило будет сравнивать MAC-адрес источника трафика и MAC-адрес назначения, и отправлять трафик, согласно значению параметра output. Такая запись называется Layer 2 matching (рисунок 2.28).

```
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00
:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00
:01,actions=output:1
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
```

Рисунок 2.28 – Layer 2 matching

Для корректной работы L2 необходимо прописать правило, которое будет контролировать ARP-запросы. Первые 2 строчки команды описывают

передачу трафика согласно физическим адресам, но для того, чтобы OpenFlow-коммутатор получил сведения о том, за каким IP закреплен нужный MAC-адрес, нужно описать правило, которое разрешает широковещательный трафик. В данном примере действие flood дает возможность отправлять ARP-REQUEST на все порты, кроме того, с которого запрос был сгенерирован.

Третья запись в таблице потоков описывает политику передачи трафика на основе IP адресации и ToS – Layer 3 matching. Запись (sh ovs-ofctl add-flow s1 priority=500,ip,nw_src=10.0.0.3,actions=mod_nw_tos:184,normal). В данной записи описывается ToS, который позволяет маркировать трафик специальной меткой DSCP (Differenciated Services Code Point), что в свою очередь определяет его приоритет. Если обратит внимание на саму запись, то параметр priority отличается от DSCP тем, что задает порядок сравнения пакетов передаваемого трафика с записями в таблице потоков. Если повысить значение priority в записи для ToS, то пакет сначала пройдет сравнение по этой записи, а потом по всем остальным. На рисунке 2.29 показан пример создания Layer 3 matching. Такая структура таблиц очень похожа на работу списков доступа в процессе маршрутизации трафика в традиционных сетях – ACL.

```
mininet> sh ovs-ofctl add-flow s1 priority=500,ip,nw_src=10.0.0.0/24,nw_dst=10.0
.0.0/24,actions=normal
mininet>
mininet> sh ovs-ofctl add-flow s1 priority=500,ip,nw_src=10.0.0.3,actions=mod_nw
_tos:184,normal
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.1,actions=output:1
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.2,actions=output:2
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.3,actions=output:3
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=80.825s, table=0, n_packets=4, n_bytes=168, idle_age=52, a
rp,arp_tpa=10.0.0.3 actions=output:3
 cookie=0x0, duration=181.265s, table=0, n_packets=4, n_bytes=168, idle_age=52,
arp,arp_tpa=10.0.0.2 actions=output:2
 cookie=0x0, duration=259.772s, table=0, n_packets=4, n_bytes=168, idle_age=52,
arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=376.809s, table=0, n_packets=0, n_bytes=0, idle_age=376, p
riority=500,ip,nw_src=10.0.0.3 actions=mod_nw_tos:184,NORMAL
 cookie=0x0, duration=1306.425s, table=0, n_packets=12, n_bytes=1176, idle_age=5
7, priority=500,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=NORMAL
```

Рисунок 2.29 – Layer 3 matching

3 Моделирование сквозной задержки в программно-конфигурируемой сети

3.1 Проактивный и реактивный режим работы

В SDN, когда контроллер заполняет таблицы потоков, существует два основных режима работы: проактивная установка правил и реактивная установка правил [16].

В реактивном режиме таблица потоков вначале пуста. Когда пакет прибывает на коммутатор, никакое правило не выполняется, и пакет будет инкапсулирован как сообщение PACKET_IN и отправлен на контроллер центральным процессором уровня данных в коммутаторе. В то же время коммутатор буферизует пакет, ожидая команды контроллера. Контроллер загружает соответствующие правила в коммутатор. Последующие пакеты того же потока затем соответствуют вновь установленным правилам и передаются на полной скорости линии, не нарушая работу контроллера.

В проактивном режиме таблица потоков устанавливается в коммутаторе заранее, до прибытия пакета. Для достижения производительности поиска на проводной скорости современные коммутаторы обычно используют TCAM для хранения этих правил. TCAM может сопоставлять каждый входящий пакет со всеми правилами параллельно и выводить результат за один такт. Однако в коммутаторе SDN может не хватить места для выполнения всех правил, особенно в крупномасштабных сетях, по следующим причинам:

– TCAM стоит дорого и потребляет большое количество энергии. Современные наборы микросхем TCAM обычно поддерживают всего несколько тысяч или десятки тысяч записей, что далеко от удовлетворения текущих требований к управлению сетью [17, 18];

– разрыв между значительным увеличением трафика и медленным увеличением емкости TCAM увеличивается. Масштаб таблицы потоков постоянно расширяется;

– размер таблицы SDN также увеличивается. Возьмем к примеру, OpenFlow, один из четко определенных открытых южных API, количество полей соответствия составляет 12 в версии 1.0, но 45 в версии 1.5. Это означает, что каждое правило будет занимать больше места для хранения.

Когда TCAM заполнен и необходимо вставить новое правило, нужно удалить старое правило, чтобы освободить место для нового правила. Независимо от того, какой алгоритм замены принят, он неизбежно приведет к удалению некоторых активных правил из таблицы потоков. Последующие пакеты, соответствующие этим удаленным правилам, должны вызвать генерацию входящих пакетов на контроллер.

Следующие два сценария подчеркивают, что даже если коммутатор SDN настроен в проактивном режиме, все же возможно, что будет активирована настройка потоков:

1) Некоторые исследователи такие как DevoFlow [18], предложили использовать грубые правила, чтобы максимально сократить количество обращений к контроллеру. Однако это дилемма: агрессивное использование подстановочных знаков соответствия потока подрывает способность контроллера эффективно управлять сетевым трафиком, а также не может производить точные измерения и сбор статистики. Фактически чтобы улучшить использование сети и производительность приложений и удовлетворить такие потребности, как безопасность, измерение и справедливость в различных сетевых сценариях, предлагается все больше и больше политик управления трафиком. Эти политики обычно преобразуются в детальные правила (например, правило контроля доступа, правила ограничения скорости и маршрутизации политик) [19]. Количество детализированных правил, вероятно может достигать сотен тысяч или даже миллионов [19], которые вряд ли могут быть сохранены в текущем TCAM. Это неизбежно приведет к замене правил и настройки потока.

2) Из-за ограничений ЦП, памяти и изменений трафика виртуальные машины часто требуются для миграции как внутри, так и между центрами обработки данных [20]. Однако виртуальные машины обычно имеют тесную связь с сетью. Например, сетевые политики такие как QOS, ACL и маршруты политик, в коммутаторах обычно зависят от виртуальных машин. Следовательно, миграция виртуальных машин часто вызывает изменение конфигурации сетевых политик. Соответствующие правила коммутатора должны быть удалены, и этот процесс вызывает обновление таблицы потоков. Между тем, коммутатор связанный с новым сервером динамически запускает вставку правила для перенаправления трафика на новый физический интерфейс.

3.2 Модель сквозной задержки

Сквозная задержка каждого пакета – это сумма задержек, испытываемых в последовательности промежуточных узлов на пути к месту назначения. Каждая задержка представляет собой сумму двух частей: фиксированной части, такой как задержка передачи и задержка распространения, и переменной части, такой как задержка обработки и задержка постановки в очередь в узлах.

На рисунке 3.1 показана разбивка задержек при сквозном прохождении пакетов. Пакеты начинаются с исходного хоста, проходят через несколько коммутаторов и достигают своего конечного хоста. Общая задержка D включает в себя задержку обработки протокола (StkD), которая представляет собой задержку обработки, связанную со стеком протоколов исходного и конечного хоста; задержку передачи (TD) на исходном и конечном хостах; задержку распространения (PD) в среде передачи; задержку коммутатора (SD), вызванную пересылкой пакетов в коммутаторах. Сквозную задержку можно представить следующей формулой:

$$D = StkD_{src} + TD_{src} + PD + SD + TD_{dst} + StkD_{dst}. \quad (3.1)$$

Сквозная задержка начинается с момента, когда приложение на исходном хосте отправляет сообщение в интерфейс сокета. Затем сообщение обрабатывается стеком протоколов, и карта сетевого интерфейса извлекает пакет из памяти хоста и отправляет его на физический уровень. Затраты времени в описанной выше процедуре представлены $StkD_{src}$. Процесс приема на хосте назначения – процесс, обратный отправке пакета. $StkD_{dst}$ – это величина задержки, с которой пакет обрабатывается стеком протоколов на хосте назначения. $StkD_{dst}$ – это временной интервал между моментом, когда сетевая карта начинает копировать пакет в память с использованием DMA, и моментом, когда приложение на хосте назначения получает этот пакет.

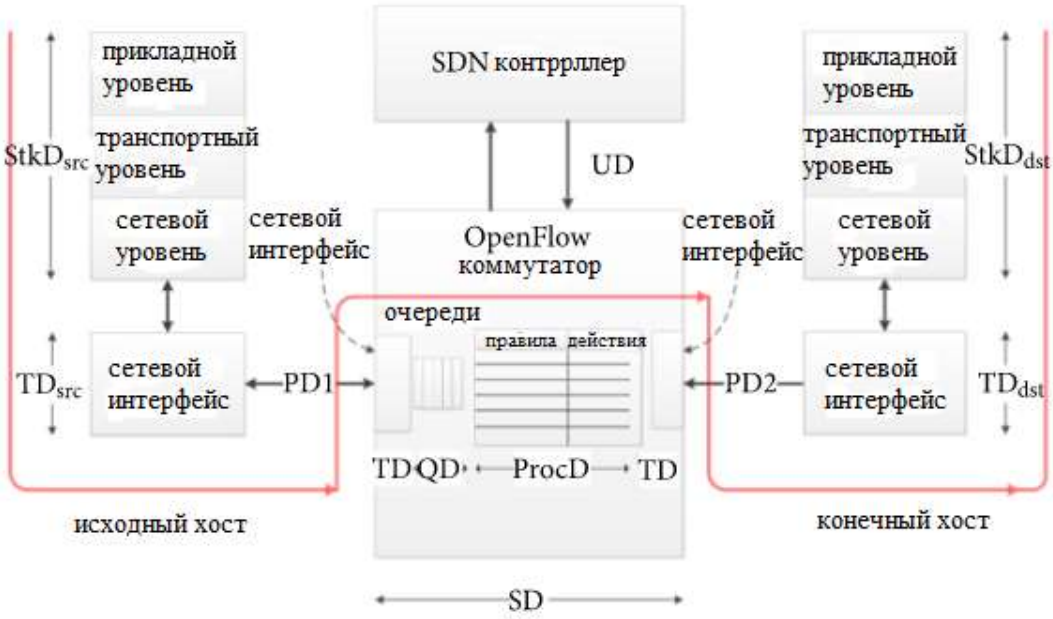

Рисунок 3.1 – Схема разбивки сквозной задержки

TD_{src} – это задержка, необходимая для передачи всех битов пакета от хоста источника к исходящему каналу. Точно так же TD_{dst} – это количество времени, необходимое для приема всех битов пакета от канала до хоста назначения. TD можно рассчитать по следующему уравнению:

$$TD = \frac{s}{r}, \quad (3.2)$$

где s – размер данных;
r – скорость передачи.

PD относится к задержке распространения пакетов в среде связи (медь, оптическое волокно или беспроводной канал) по каждому каналу. Он пропорционален расстоянию между хостом источником и хостом получателем

и может быть представлен следующей формулой для каждого сегмента канала:

$$PD = \frac{l}{p}, \quad (3.3)$$

где l – расстояние;

p – скорость распространения.

Скорость распространения варьируется в разных средах связи. Например, скорость медного провода составляет примерно $2.3 \cdot 10^5$ км/с, а в оптоволоконном кабеле – примерно $2.0 \cdot 10^5$ км/с.

SD – общая задержка, вызванная коммутаторами SDN.

3.3 Модель задержки для одного SDN коммутатора

На рисунке 3.2 показана задержка настройки потока на одном коммутаторе S_i. Настройка потока может быть далее разделена на следующие подпроцедуры:

1) Когда пакеты прибывают пачками, они не могут быть немедленно обработаны и должны быть помещены в очередь в буфере. Время ожидания – это задержка в очереди (q_i).

2) Коммутатор извлекает заголовок из входящего пакета и сравнивает его с таблицей потоков. Назовем временные затраты на этот процесс задержкой обработки (t_i).

3) Если пакет не соответствует ни одной записи в таблице потоков, он будет передан от механизма пересылки к главному процессору в плоскости данных через PCI/PCIe. Затем хост-процессор генерирует соответствующее сообщение Packet_In. Задержка определяется как h_i^{up}.

4) Сообщение Packet_In передается в контроллер через безопасный канал OpenFlow, и эта задержка определяется как w_i^{up}.

5) Контроллер анализирует сообщение Packet_In и генерирует управляющие сообщения, включая flow_mod и packet_out, на основе своих локальных политик. Задержка определяется как C.

6) Сообщение контроллера загружается в коммутатор через безопасный канал OpenFlow. Задержка определяется как w_i^{down}.

7) После получения управляющего сообщения главный процессор анализирует и преобразует логическое представление инструкций в инструкции, специфичные для оборудования. Инструкции передаются механизму пересылки коммутатора через PCI/PCIe. Инструкции выполняются оборудованием, а новые правила вставляются в поисковую систему (например, TCAM). Задержка определяется как h_i^{down}.

D_i^{setup} – это общая задержка вышеуказанных подпрограмм, которую можно выразить следующей формулой:

$$D_i^{setup} = q_i + t_i + h_i^{up} + w_i^{up} + C + w_i^{down} + h_i^{down}, \quad (3.4)$$

В подпроцедуре 3, если пакет соответствует записи в таблице потоков, настройка потока пропускается, и модель задержки коммутатора может быть упрощена как:

$$D_i^{bypass} = q_i + t_i. \quad (3.5)$$

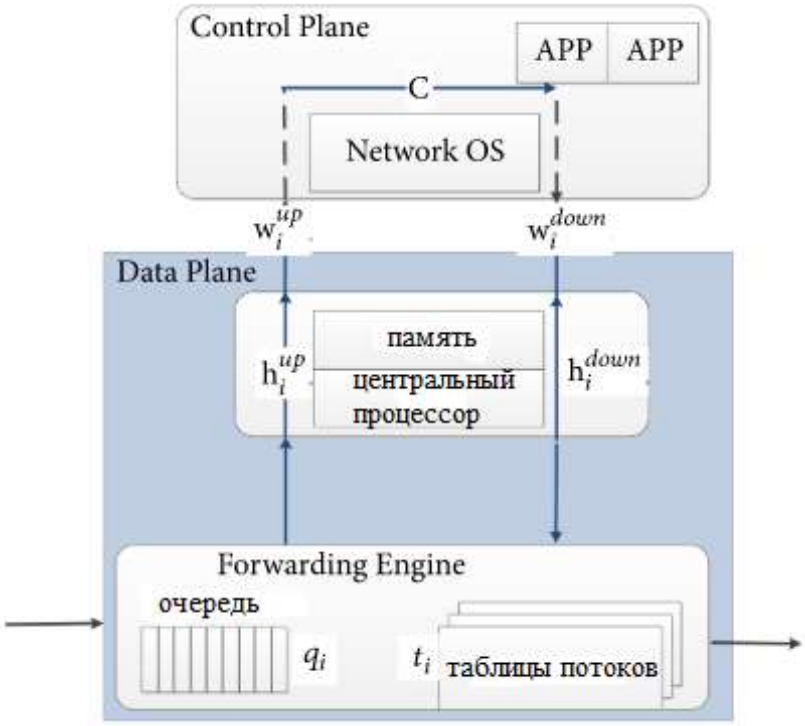

Рисунок 3.2 – Задержка настройки потока

3.4 Модель задержки с несколькими коммутаторами SDN

По сравнению со сценарием с одним коммутатором, модель задержки с несколькими коммутаторами более сложна. Когда контроллер отправляет сообщение flow_mod, задержка распространения между контроллером и разными коммутаторами различается. Более того, затраты времени на установку правил на разных коммутаторах также различается в зависимости от структуры таблицы потоков в TCAM и зависимостей между новыми правилами поступления и существующими элементами таблицы в TCAM. Эти два вида задержки могут повлиять на порядок установки правил на разных коммутаторах, а также могут повлиять на согласованность логики управления.

По-прежнему используются переменные из раздела 3.3. Определены новые переменные, $D_{i,j}^{reactive}$, представляет собой задержку начала с момента получения пакета S_i до момента, когда пакет покидает S_j в реактивном сценарии, и $D_{i,j}^{proactive}$, представляет собой соответствующую задержку в

проактивном сценарии. Параметр $w_{i,j}$ определяется как задержка распространения между S_i и S_j.

Сначала изучаем простую топологию, которая состоит только из двух коммутаторов, S_1 и S_2. Эти два коммутатора управляются одним и тем же контроллером SDN.

Используем диаграмму последовательности для описания задержки прохождения пакета в этой топологии (рисунок 3.3). В реактивном режиме таблица потоков вначале пуста. Настройка потока запускается во входном коммутаторе S_1, а затем контроллер загружает соответствующие правила в S_1 и все нижестоящие коммутаторы, S_2 в этом примере. Коммутаторам, расположенным ниже по потоку, больше не требуется запуск настройки потока. Задержка составляет ($w_1^{down} + h_1^{down} + w_{1,2} + q_2$) от момента времени в который контроллер отправляет правила, до момента времени, когда пакет покидает S_1 и достигает S_2. Тем временем S_2 потратит ($w_2^{down} + h_2^{down}$) на получение и установку правил. Когда пакет приходит на S_2 и готов к поиску, если соответствующие правила уже были загружены в TCAM, пакет не обязательно должен ждать на S_2, и общая задержка составляет:

$$D_{1,2}^{reactive} = D_1^{setup} + w_{1,2} + D_2^{bypass}. \quad (3.6)$$

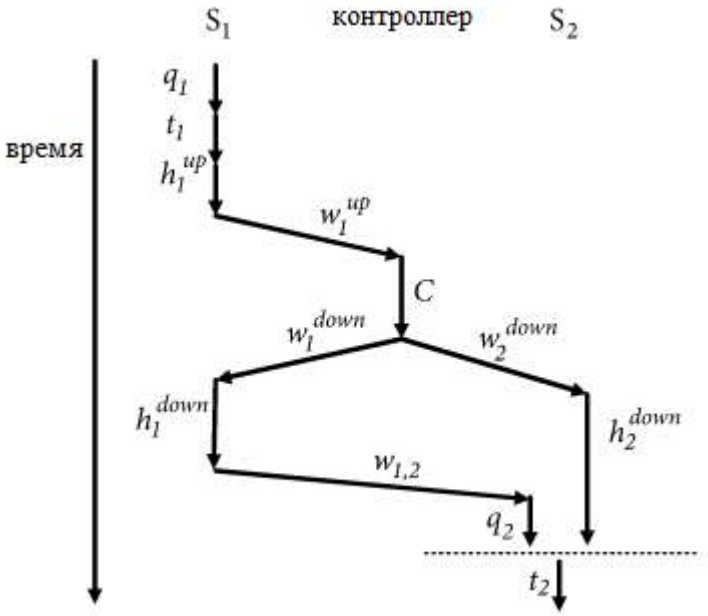

Рисунок 3.3 – Диаграмма последовательности прохождения пакетов в примере топологии, состоящей из двух коммутаторов S_1 и S_2

В противном случае пакет должен быть приостановлен до конца реорганизации TCAM в S_2. Полная задержка может быть представлена следующим уравнением:

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + w_2^{down} + h_2^{down} + t_2, \quad (3.7)$$

Два приведенных выше уравнения можно объединить в одно:

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + t_2 + \left(w_1^{down} + h_1^{down} + w_{1,2} + q_2 + w_2^{down} + h_2^{down}\right), \quad (3.8)$$

Определим новый параметр M_j – это задержка от момента времени, в который контроллер отправляет правила, до момента времени, когда пакет готов к поиску в таблице потоков S_j. M_j можно представить следующим рекурсивным уравнением:

$$M_j = \left(M_{j-1} + w_{j-1,j} + q_j + w_j^{down} + h_j^{down}\right), \quad (3.9)$$

Начальное значение M_1 равно:

$$M_1 = w_1^{down} + h_1^{down}, \quad (3.10)$$

Тогда формулу (3.8) можно упростить как:

$$D_{1,2}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + M_2 + t_2, \quad (3.11)$$

Точно так же, когда есть три коммутатора, общая задержка $D_{1,3}^{reactive}$ может быть представлена:

$$D_{1,3}^{reactive} = q_1 + t_1 + h_1^{up} + w_1^{up} + C + M_3 + t_3, \quad (3.12)$$

Кроме того, дадим обобщение вышеупомянутой модели, и общая задержка прохождения пакета от S_i к S_j может быть сформулирована следующим уравнением:

$$D_{i,j}^{reactive} = q_i + t_i + h_i^{up} + w_i^{up} + C + M_j + t_j. \quad (3.13)$$

В проактивном режиме, как упоминалось в разделе 3.1, хотя правила предустановлены в коммутаторе, пакет все равно может не найти соответствующую запись потока и с определенной вероятностью будет отправлен на контроллер. Параметр p_i обозначает эту вероятность. Таким образом, в проактивном сценарии задержка коммутатора S_i, $D_i^{proactive}$ может быть рассчитана как:

$$D_i^{proactive} = p_i \cdot D_i^{setup} + (1 - p_i) \cdot D_i^{bypass}. \quad (3.14)$$

Чтобы вывести формулу сквозной задержки, сосредоточимся на простой топологии, которая включает два коммутатора S_1 и S_2, управляемые одним и тем же контроллером SDN. В этом сценарии есть три случая:

1) Правила в S_1 не найдены, в S_2 найдены.
2) Правила в S_1 найдены, в S_2 не найдены.
3) Правила в S_1 и S_2 найдены.

Вероятность и соответствующая задержка для каждого случая перечислены в таблице 3.1. Таким образом, общая задержка $D_{1,2}^{proactive}$ в проактивном режиме может быть рассчитана как:

$$\begin{aligned} D_{1,2}^{proactive} &= p_1 \cdot D_{1,2}^{reactive} + (1-p_1) \cdot p_2 \cdot \left(D_1^{bypass} + x_{1,2} + D_2^{setup}\right) \\ &+ (1-p_1) \cdot (1-p_2) \cdot \left(D_1^{bypass} + x_{1,2} + D_2^{bypass}\right) \\ &= p_1 \cdot D_{1,2}^{reactive} + (1-p_1)\left(D_1^{bypass} + x_{1,2} + D_2^{proactive}\right), \end{aligned} \tag{3.15}$$

Таблица 3.1 – Вероятность и соответствующая задержка трех случаев в приведенном выше примере

Случаи	Детали	Вероятность	Задержка
Случай 1	S_1 пропуск, S_2 совпадение	p_1	$D_{1,2}^{reactive}$
Случай 2	S_1 совпадение, S_2 пропуск	$(1 - p_1) \cdot p_2$	$D_1^{bypass} + x_{1,2} + D_2^{setup}$
Случай 3	S_1 совпадение, S_2 совпадение	$(1 - p_1) \cdot (1 - p_2)$	$D_1^{bupass} + x_{1,2} + D_2^{bypass}$

Приведенное выше уравнение можно упростить как:

$$D_{1,2}^{proactive} = p_1 D_{1,2}^{reactive} + (1-p_2)\left(D_2^{bypass} + x_{1,2} + D_1^{proactive}\right) \tag{3.16}$$

Аналогично, общая задержка $D_{i,j}^{proactive}$ может быть представлена следующим рекурсивным уравнением:

$$D_{i,j}^{proactive} = p_j \cdot D_{i,j}^{reactive} + (1-p_j) \cdot \left(D_j^{bypass} + x_{j-1,j} + D_{i,j-1}^{proactive}\right) \tag{3.17}$$

3.5 Задержки, возникающие в программно-конфигурируемых сетях при обработке ARP-пакетов

В традиционных локальных сетях в процессе передачи пакетов между узлами важную роль играет процесс определения адресов, осуществляемый в соответствии с протоколом определения адреса ARP. Исходный узел посылает ARP-запрос коммутатору для получения MAC-адреса узла получателя. Однако механизм обработки ARP-запросов и ARP-ответов в программно-конфигурируемых сетях отличается от традиционных локальных сетей. В

SDN механизм определения адресов выполнен на контроллере и может требовать больших временных затрат, зависящих от текущей нагрузки и производительности контроллера.

Выделим две величины, которые можно применить для характеристики задержки механизма определения адреса [21]:

1) Address Resolution Delay No Forwarding Flow Registrations (ARDNFFR) – задержка определения адреса без установки потока для передачи пакета на коммутаторе.

2) Address Resolution Delay Forwarding Flow Registrations (ARDFFR) – задержка определения адреса с установкой потока для передачи пакета на коммутаторе.

Дадим определения этим величинам, основываясь на топологии сети, представленной на рисунке 3.4, где C – контроллер, S – коммутатор, H_1 и H_2 – узлы в сети.

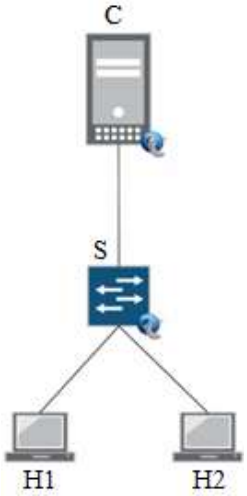

Рисунок 3.4 – Схема сети

Определение 1: Рассмотрим задержку ARDNFFR. Пусть узел H_1 инициализирует ARP-запрос узлу H_2 в момент времени T_{H1}, тогда H_1 принимает ARP-ответ от H_2 в момент времени $T_{H1}+d\tau_{NF}$ при отсутствии требуемой записи в таблице потоков коммутатора. Получаем, что задержка ARDNFFR есть величина $d\tau_{NF}$.

С учетом определения 1 и топологии сети, представленной на рисунке 3.4, задержка ARDNFFR складывается из задержки передачи по каналам связи, задержки обработки пакета на контроллере и задержки обработки на коммутаторе:

$$d\tau_{NF} = L + C_{NF} + S_{NF} \tag{3.18}$$

где L – задержка в каналах связи;

C_{NF} – задержка обработки на контроллере;

S_{NF} – задержка обработки на коммутаторе без установленных правил.

Определение 2: Рассмотрим задержку ARDFFR. Пусть узел H_1 инициализирует ARP-запрос узлу H_2 в момент времени T_{H1}, тогда H_1 принимает ARP-ответ от H_2 в момент времени $T_{H1}+d\tau$ при наличии требуемой записи в таблице потоков коммутатора. Получаем, что задержка ARDFFR есть величина $d\tau$.

С учетом определения 2 и топологии сети, представленной на рисунке 3.4, задержка ARDFFR складывается из задержки передачи по каналам связи и задержки обработки на коммутаторе:

$$d\tau = L + S \tag{3.19}$$

где L – задержка в каналах связи;

S – задержка обработки на коммутаторе с установленными правилами.

Исходя из определений 1 и 2, а также формул (3.18), (3.19) можно сделать вывод, что задержка механизма определения адресов в SDN будет превышать значения задержки в традиционной сети только в том случае, когда на коммутаторе отсутствует запись в таблице потоков.

Применение ARDFFR вместо ARDNFFR обусловлено следующими факторами:

– определение адреса с установленными потоками на коммутаторах является основной формой механизма определения адреса в SDN;

– когда время жизни ARP-таблиц на коммутаторе заканчивается, узел должен вновь инициировать механизм определения адреса, а значит и установку потока на коммутаторах.

Высокое значение параметров ARDFFR и ARDNFFR может свидетельствовать о проблемах на одном или более сегментах сети, а значит, данный параметр может быть использован для диагностики программно-конфигурируемой сети в целом.

Рассмотрим возможность использования двух этик параметров для оценки программно-конфигурируемых сетей.

Введем следующие обозначения:

– H_1, H_2 – сетевые узлы;

– T_i – временной интервал, в течении которого на коммутаторах существуют записи о потоках, с;

– T_0,T_f – временные границы рассматриваемого интервала, с;

– λ – интенсивность потока, 1/с.

В [21] указано, что поток ARP-пакетов можно рассматривать в качестве Пуассоновского потока. Имея значения T_0,T_f и λ можно описать

псевдослучайный Пуассоновский процесс на временном интервале T_0,T_f с интенсивностью поступления пакетов λ. В каждый момент времени T_i значение задержки для первого процесса механизма определения адреса является параметром ARDNFFR, а для последующих процессов механизма определения адреса ARDFFR. Далее можно получить значение параметров ARDNFFR и ARDFFR на промежутке от T_0 до T_f с интервалом T_i.

Поток событий – последовательность событий, происходящих одно за другим в случайный момент времени [22].

Пуассоновский поток – это поток, обладающий двумя свойствами – ординарностью и отсутствием последействия.

Поток называется ординарным, если для малого интервала Δt выполняется условие:

$$P_1(t,\Delta t) >> P_{>1}(t,\Delta t) \tag{3.20}$$

где P_1(t, Δt) – вероятность того, что за Δt произойдет одно событие;

$P_{>1}$(t, Δt) – вероятность того, что за Δt произойдет более одного события.

Таким образом, поток можно считать ординарным, если за малый промежуток времени может произойти не более одного события (или ни одного события, вероятность чего обозначим P_0(t, Δt)). Для любого Δt справедливо:

$$P_0(t,\Delta t)+P_1(t,\Delta t)+P_{>1}(t,\Delta t)=1 \tag{3.21}$$

Так как составляющие формулы (3.21) определяют полную группу несовместных событий.

Для ординарного потока:

$$P_0(t,\Delta t)+P_1(t,\Delta t)\approx 1 \tag{3.22}$$

Потому что $P_{>1}$(t, Δt) = 0(Δt), где 0(Δt) – величина, порядок малости, который выше чем Δt [22], т.е.:

$$\lim_{\Delta t\to 0}\frac{0(\Delta t)}{\Delta t}=0 \tag{3.23}$$

Рассмотрим интервал [T_0,T_f] = mT_i, представленный на рисунке 3.5.

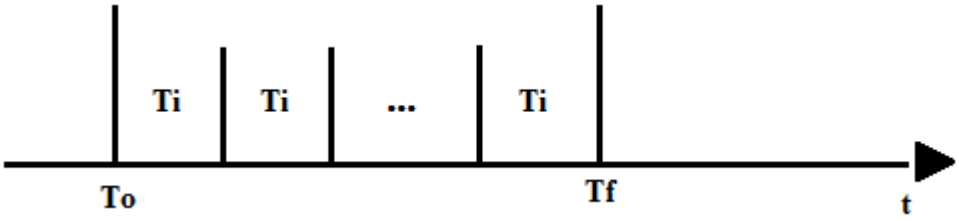

Рисунок 3.5 – Интервал $[T_0,T_f]$

Отсутствие последействия – свойство, когда для двух неперекрывающихся интервалов времени число событий, попадающих в один интервал, не зависит от того, сколько событий попало в другой.

Воспользуемся формулой распределения Пуассона [22] для оценки вероятности возникновения k-запросов от коммутатора к контроллеру на установление правил на временном интервале $[T_0,T_f] = mT_i$. Получим:

$$P_{ARDNFFR} = \frac{(\lambda m T_i)}{k!} e^{-\lambda m T_i} \tag{3.24}$$

Тогда вероятность возникновения n-событий в режиме функционирования с установленными правилами на коммутаторе будет определятся формулой:

$$P_{ARDFFR} = 1 - P_{ARDNFFR} \tag{3.25}$$

Таким образом, общая задержка D, возникающая при функционировании механизма определения адреса может быть вычислена как:

$$D = l \cdot ARDNFFR + n \cdot ARDFFR \tag{3.26}$$

где l и n – число временных интервалов, к которым применимы параметры задержки ARDNFFR и ARDFFR соответственно.

Из полученных результатов исследования можно сделать следующие выводы:

1) Если величина T_i стремиться к бесконечности, то параметр ARDFFR будет преобладать над ARDNFFR. В противном случае может практически не быть ARDFFR.

2) Если величина λ стремиться к бесконечности, то параметр ARDNFFR ничтожно мал на фоне ARDFFR, поскольку только одно из всего множества событий на интервале T_i будет соответствовать режиму функционирования сети без установленных потоков на коммутаторах. В противном случае, если λ слишком мала, вероятность возникновения ARP-запросов на интервале $[T_0,T_f] = mT_i$ может стремиться к нулю.

Полученные формулы оценки задержки в программно-конфигурируемых сетях имеют практическую ценность, поскольку позволяют сетевому администратору получить конкретные значения времени обработки пакетов на каждом из устройств сети. В случае если величина задержки на контроллере весьма велика, системный администратор может сделать вывод о необходимости оптимизации работы контроллера либо путем перенастройки установленных правил, либо путем модернизации аппаратной части.

3.5.1 Экспериментальное выявление зависимости задержки передачи служебного трафика от задержки контроллера. Для выявления зависимости между задержкой передачи служебного трафика и задержкой контроллера в сети, функционирующей с использованием протокола OpenFlow в среде Mininet была смоделирована древовидная топология, показанная на рисунке 3.6. Число коммутаторов последовательно задавалось равным 3, 7, 15 и 31. Выполнялась команда ping с узла H1 до H2, после чего вычислялась задержка $d\tau_{NF}$, задержка коммутатора S_{NF} и задержка контроллера C_{NF}, представленные в формуле (3.18). При этом задержка коммутатора S_{NF} считалась как среднее арифметическое по всем OpenFlow-коммутаторам.

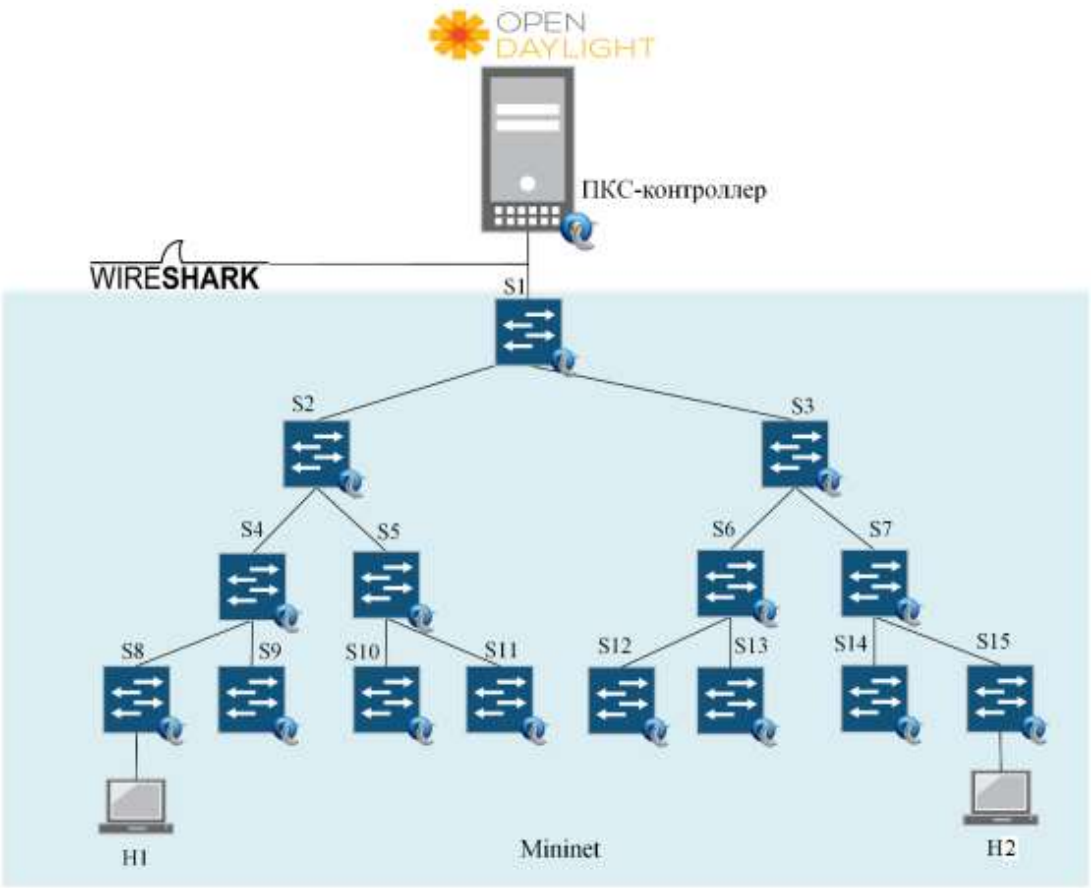

Рисунок 3.6 – Топология сети

Таблица 3.2 – Результаты измерения, полученные в ходе эксперимента

Количество коммутаторов в сети, шт.	Задержка $d\tau_{NF}$, мс	Задержка S_{NF}, мс	Задержка C_{NF}, мс
3	9,868	0,184	4,688
7	12,236	0,240	7,433
15	16,252	0,431	6,847
31	23,881	1,870	11,441

По результатам проведенного исследования был построен график, приведенный на рисунке 3.7.

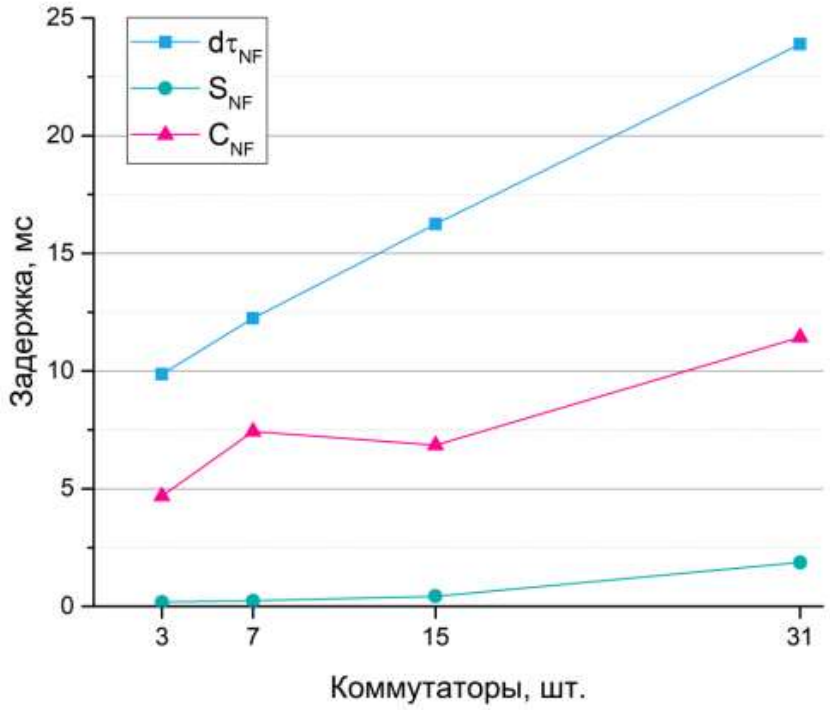

Рисунок 3.7 – Зависимость величины задержки от количества коммутаторов в сети

Исходя из графика, выдвинем гипотезу о наличии корреляции между исследуемыми параметрами. Для проверки этого осуществим расчет коэффициентов корреляции попарно для экспериментальных значений $d\tau_{NF}$ и C_{NF}, $d\tau_{NF}$ и S_{NF} согласно следующему уравнению:

$$r = \frac{\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{\left(n\sum x^2 - \left(\sum x\right)^2\right)\cdot\left(n\sum y^2 - \left(\sum y\right)^2\right)}} \qquad (3.27)$$

Полученные значения $r(d\tau_{NF}, C_{NF}) = 0.935$, $r(d\tau_{NF}, S_{NF}) = 0.951$ указывают на наличие положительной линейной корреляции между исследуемыми

параметрами, т.е. с ростом/уменьшением значения одного из параметра будет расти/уменьшаться и другой. Таким образом, снижение величины задержки, вносимой контроллером, приведет к снижению задержки передачи служебного трафика в сети SDN.

3.6 Экспериментальное выявление задержки трафика в сети SDN

Чтобы оценить влияние поступающих пакетов на задержку и влияние обновления TCAM на задержку был создан экспериментальный стенд в программе GNS3. Платформа измерения показана на рисунке 3.8.

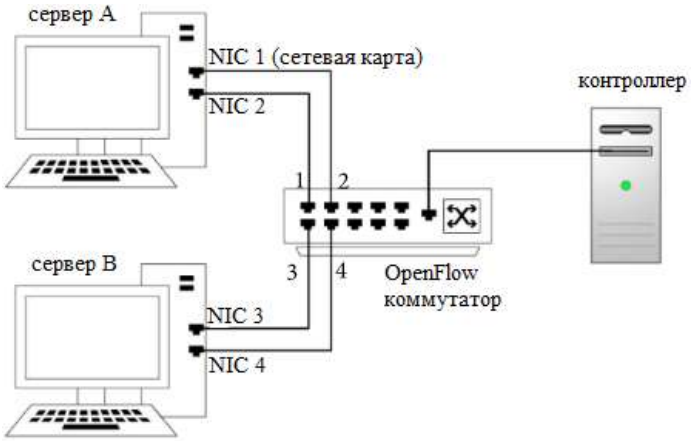

Рисунок 3.8 – платформа измерения

В экспериментальном стенде используется OpenDaylight (версия Carbon) в качестве работающего пакета в нашем контроллере, коммутатор Open vSwitch поддерживающий протокол OpenFlow. Сервер А используется для генерации пакетов для новых потоков в соответствии с различными параметрами. Эти новые потоки будут вызывать настройку потока в тестируемом коммутаторе. Сервер В используется для генерации базового трафика. Базовый трафик относится к пакетам, которые соответствуют предустановленным правилам в TCAM. И сервер А, с сервер В установлены с двумя сетевыми адаптерами 10 Гбит/с. Контроллер подключен к порту управления через порт 1 Гбит/с. Часы на разных серверах не синхронизируются. Чтобы избежать отклонения, вызванного синхронизацией времени, используется одно сетевая карта для отправки пакетов и другая сетевая карта на том же компьютере для приема пакетов. Используется Ostinato для создания определенных типов пакетов и потоков трафика. Развернут Wireshark для захвата пакетов.

3.6.1 Влияние поступления пакетов на задержку. В этой части проведены измерение задержки пакетов, когда генерируются пакеты packet_in на контроллер. Таблица потоков в коммутаторе изначально пуста. NIC 1 на сервере А отправляет новые потоки с разными скоростями на коммутатор через порт 1, и эти новые потоки инициируют соответствующие входящие пакеты packet_in на контроллер. Коммутатор принимает и анализирует

сообщения flow_mod от контроллера и вставляет соответствующие записи потока в TCAM. Установлены одинаковые приоритеты для всех этих потоковых записей, чтобы изолировать влияние реорганизации TCAM. Затем коммутатор пересылает эти пакета на NIC 2. Начальная точка задержки пакета – это момент, когда NIC 1 начинает отправку, а конечная точка – это момент, когда NIC 2 принимает пакеты.

Проводятся три эксперимента, чтобы показать задержку пакетов с/без обращения на контроллер при новой скорости потока 50, 500 и 5 тысяч пакетов/с. Результаты представлены на рисунках 3.9, 3.10, 3.11. В каждом эксперименте общее количество потоков составляет 300, а ось x относится к идентификатору потока. Для определенной скорости поступления нового потока задержка пакетов, с обращением на контроллер значительно больше, чем у пакетов без обращения на контроллер. Например, когда скорость поступления потока составляет 5 тысяч пакетов/с, задержка пакета без обращения на контроллер является переменной со средним значением 0.0859 мс и стандартным отклонением 0.0452, в то время как среднее значение задержки пакета с обращением на контроллер составляет 0.7415 с, а стандартное отклонение 0.5954.

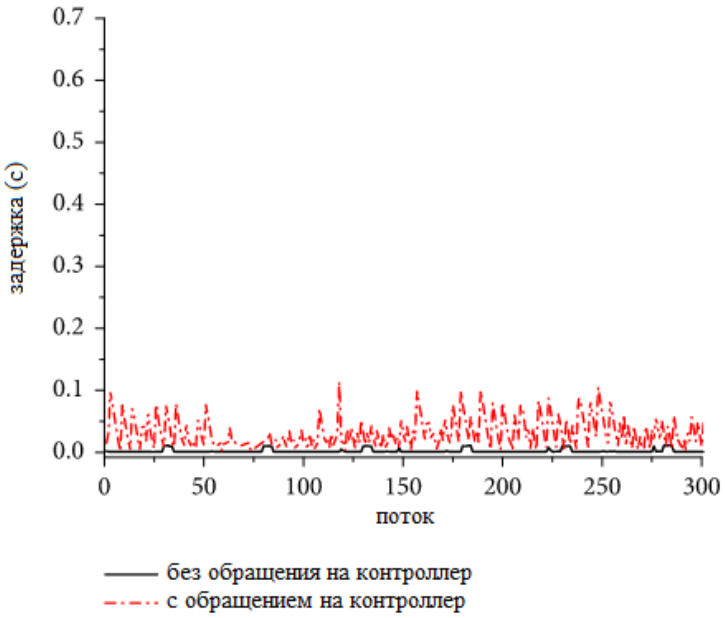

Рисунок 3.9 – Задержка пакетов с / без обращения на контроллер, когда новая скорость потока составляет 50 пакетов/с

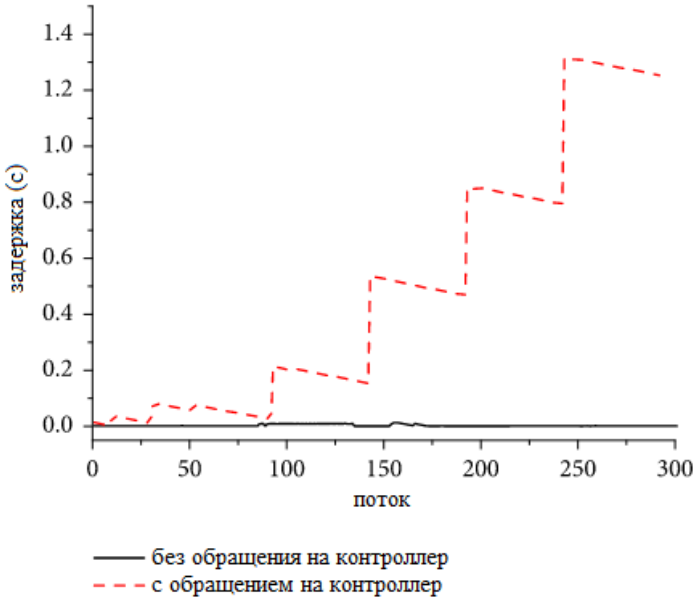

Рисунок 3.10 – Задержка пакетов с / без обращения на контроллер, когда новая скорость потока составляет 500 пакетов/с

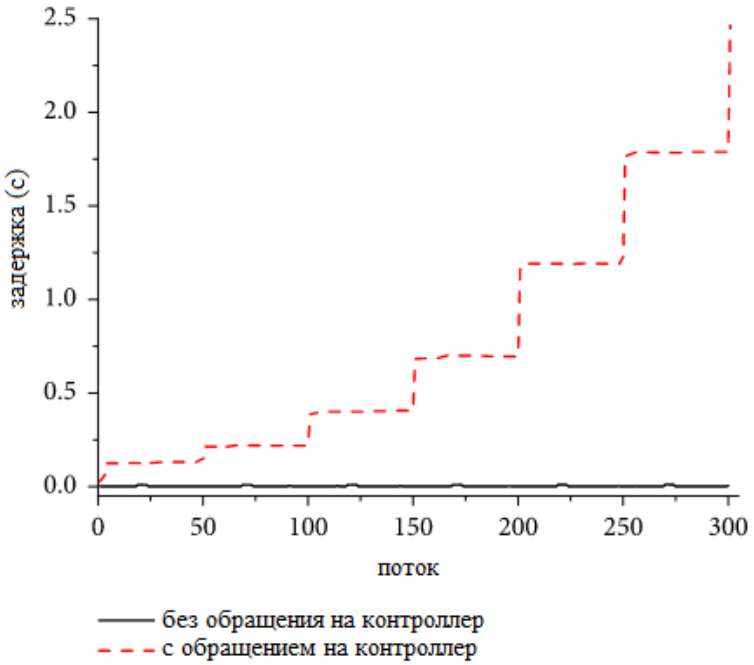

Рисунок 3.11 – Задержка пакетов с / без обращения на контроллер, когда новая скорость потока составляет 5 тысяч пакетов/с

Задержка пакета увеличивается с увеличением скорости поступления нового потока. Например, средняя задержка новых потоков со скоростью поступления 50 пакетов/с составляет 32.291 мс, в то время как задержка новых потоков со скоростью поступления 5 тысяч пакетов/с составляет 0.7415 с. Максимальная задержка составляет 2.4654 с при 5 тысяч пакетов/с. Также можем обнаружить, что распределение задержки для новых потоков со

скоростью поступления 50 пакетов/с отличается от распределения задержки для новых потоков со скоростью поступления 500 пакетов/с и 5 тысяч пакетов/с. Значительно увеличивается задержка для каждых 50 пакетов.

Чтобы найти первопричину этого явления проведем следующие две серии экспериментов. С одной стороны, используем Cbench, инструмент тестирования производительности контроллера SDN, чтобы провести измерения на контроллере. Cbench настроен как режим пропускной способности. Контроллер в экспериментальной топологии управляет 1000 хостами через один коммутатор. Тест повторяется 10 раз. Экспериментальные результаты показывают, что минимальная пропускная способность составляет 451119.36 ответов/с, максимальная пропускная способность составляет 473804.17 ответов/с, а средняя пропускная способность составляет 460435.15 ответов/с. С другой стороны, используем Wireshark для захвата пакетов на сетевой карте контроллера. Обнаруживаем, что коммутатор отправляет пакеты контроллеру партиями и размер партии составляет 50 пакетов. Средняя задержка каждого нового потока, обрабатываемого контроллером составляет около 1-2 мс и существенно не увеличивается. Это означает, что скорость генерации входящих пакетов в вышеупомянутых экспериментах не достигает узкого места производительности контроллера. В таблице 3.3 перечислена загрузку ЦП с/без обращения на контроллер при новой скорости потока 50, 500 и 5 тысяч пакетов/с. Из таблицы видно, что загрузка ЦП значительно увеличивается с увеличением новой скорости потока, особенно когда инициируются входящие пакеты на контроллер.

Таблица 3.3 – Использование ЦП с/без обработки на контроллере при различных новых скоростях потока.

Вид	Скорость потока		
	50 пакетов/с	500 пакетов/с	5000 пакетов/с
Без обращения на контроллер	3% - 4.1%	12%	13.20%
С обращением на контроллер	24% - 30%	36% - 48%	35.2% - 67%

Основываясь на вышеупомянутых экспериментах, можем сделать вывод, что локальный ЦП в коммутаторе SDN вносит большую часть задержки. Генерация входящих пакетов на контроллер вместе с сообщением flow_mod потребляет слишком много вычислительных ресурсов ЦП. Когда скорость поступления нового потока увеличивается, пакеты должны буферизироваться и обрабатываться партиями.

3.6.2 Влияние обновления TCAM на задержку. В этом разделе исследуем задержку коммутатора во время обновления TCAM. Сначала в таблице потоков хранится только одно правило. Это правило имеет самый низкий приоритет и используется для пересылки базового трафика. NIC 1 отправляет на коммутатор 1600 новых потоков. Скорость поступления

пакетов установлена на уровне 5 пакетов/с. Новые потоки запускают установку правил, таким образом контроллер вставит соответствующее правило для каждого потока в TCAM. Затем коммутатор пересылает эти пакеты на NIC 2, руководствуясь таблицей потоков. NIC 3 используется для отправки базового трафика. Трафик будет соответствовать предустановленному правилу. NIC 4 на сервере B получает эти пакеты.

Сначала сосредоточимся на том, как приоритет правила влияет на задержку коммутатора. Проводим три эксперимента. Устанавливаем приоритет правил в одном порядке, порядке возрастания и порядке убывания. Последовательность загрузки правил может быть определена как r_1, r_2...r_n. Обозначим r_i как приоритет r_i. Для возрастающего порядка $r_i.prio>r_j.prio$, если i>j. Для одинакового порядка $r_i.prio=r_j.prio$, если i>j. В порядке убывания $r_i.prio<r_j.prio$, если i>j. Результат показан на рисунке 3.12. Из рисунка видно, когда приоритет правил в одном порядке и в порядке убывания задержка нового потока обычно меньше 100 мс. Однако в возрастающем порядке задержка нового потока показывает резкое увеличение, особенно после прибытия 1200-го нового потока.

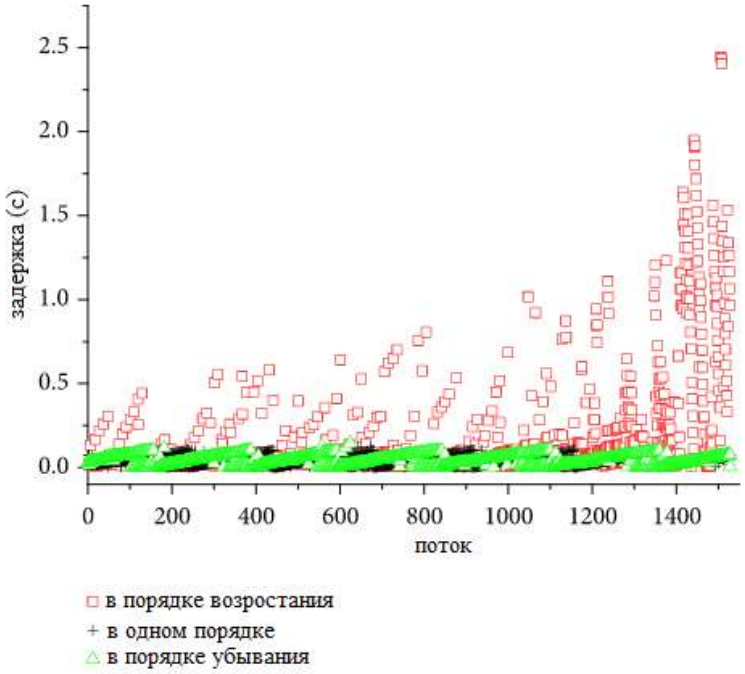

Рисунок 3.12 – Сравнение задержки пакетов, когда приоритет правил в одном порядке, порядке возрастания и порядке убывания

Основная причина этого явления – механизм организации правил в TCAM. Правило с наивысшим приоритетом всегда будет хранится по наименьшему адресу TCAM. Когда вставляется правило с более высоким приоритетом, его позиция уже занята правилом с относительно низким приоритетом. Затем правила, которые уже были в TCAM должны быть перемещены одно за другим, чтобы освободить позицию с меньшим адресом. Эти новые потоки должны буферизоваться до конца обновления TCAM. Если

последовательность правил загрузки находится в одном порядке или в порядке убывания, входного движения не будет или будет только несколько входных движений.

Также изучим влияние обновления TCAM на задержку базового трафика. Начальная точка задержки базового трафика – это момент, когда NIC 3 на сервере B начинает отправлять пакеты. Конечная точка – это момент, когда NIC 4 принимает пакеты. Правила вставляются в порядке возрастания. На рисунке 3.13 показано сравнение задержки потока с/без обновления TCAM. Из рисунка видно, что при обновлении TCAM задержка базового трафика значительно увеличивается. Средняя задержка базового трафика с обновлением TCAM в 2.42 раза больше, чем задержка базового трафика без обновления TCAM. Между тем, увеличивается загрузка локального ЦП и также увеличивается счетчик правила на плоскости данных. Делаем вывод, что во время обновления TCAM поиск таблицы в TCAM приостанавливается, а базовый трафик буферизуется и направляется на локальный ЦП для поиска в таблице. Можно сделать вывод, что обновление TCAM не только вызывает увеличение задержки нового потока, но также увеличивает задержку базового трафика. Вероятность потери пакетов значительно возрастает, когда необходимо переместить больше правил, а базовая скорость трафика высока. Это потенциально снизит стабильность всей сети.

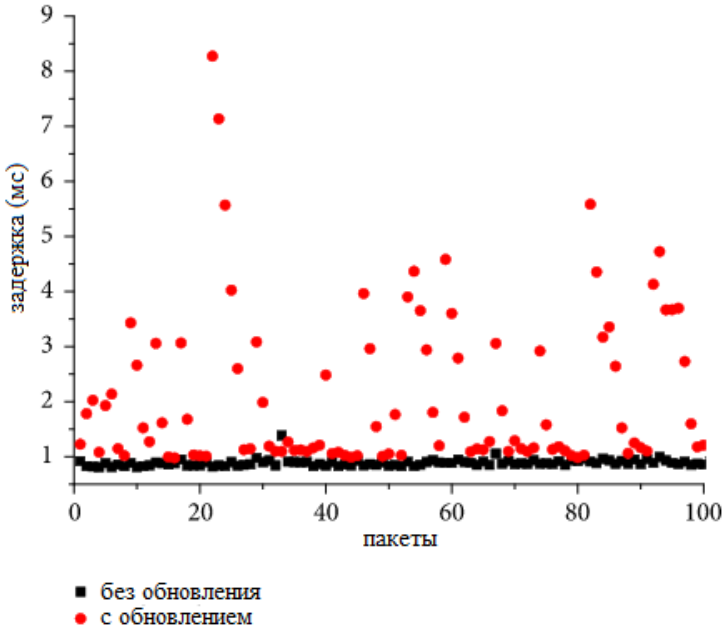

Рисунок 3.13 – Сравнение задержки трафика с/без обновления TCAM

Заключение

Преимущества, которые дает концепция SDN – это централизованное управление, мониторинг и сбор статистики в мультивендорной среде, независимость от технологий конкретного производителя, упрощение модернизации и обслуживания сети.

При этом на рынке услуг уже сейчас наблюдается широкое разнообразие сетевого оборудования, поддерживающего протокол OpenFlow, как для организации беспроводных сетей, так и проводных в зависимости от нужд потребителя.

Результаты исследования показали, что использование программно-конфигурируемых сетей с протоколом OpenFlow является современным, актуальным и перспективным решением вопроса об эффективности и производительности существующих сетей. Однако, как и каждая технология, SDN имеет свои недостатки:

– Во-первых, каждому коммутатору, несмотря на использование OpenFlow, приходится обрабатывать большие заголовки пакетов, в поисках соответствия их полей в записях своих таблиц потоков. Разумеется, плюсом здесь является то, что стандарт OpenFlow рассматривает потоки пакетов, а не отдельные пакеты, соответственно обрабатывается только заголовок первого пакета потока.

– Во-вторых, коммутаторам необходимо хранить большие объемы данных в таблицах потоков, что приводит к необходимости поддержания достаточно большого объема памяти и вынуждает использовать более мощные и соответственно достаточно дорогие платы.

– В-третьих, реализация всей функциональности OpenFlow на одном устройстве может подвергнуть сеть сбоям в результате перегрузки контроллера.

В ходе работы было рассмотрено устройство OpenFlow-коммутатора и алгоритм работы сети SDN. Представлена модель задержки передаваемого трафика по сети SDN. В результате исследования можно сделать вывод, что на задержку передачи трафика в сети SDN влияет задержка коммутатора и задержка контроллера. В случае высокого значения задержки трафика ее можно понизить путем снижения задержки коммутаторов либо контроллера. Задержка, вносимая коммутатором, определяется такими факторами как производительность коммутационной матрицы, размером буфера и наличием очередей пакетов на интерфейсах.

Задержка, вносимая контроллером, зависит как от производительности его аппаратной платформы, так и от эффективности оптимизации программного кода СОС контроллера и приложений. Для снижения задержки контроллера рекомендуется выбирать аппаратную платформу сервера с максимальной производительностью и оптимизировать работу приложений, выполняющих функции управления трафиком, что в свою очередь может привести к снижению нагрузки на систему.

Список литературы

1 Fei Hu. Network Innovation through OpenFlow and SDN. Principles and Design. CRC Press. 1st edition. February 2014. – 520 p.

2 Thomas D. Nadeau and Ken Gray. SDN Software Defined Networks. OReilly Media. September 7, 2013. – 384 p.

3 Siamak Azodolmolky. Software Defined Networking with OpenFlow. Packt Publishing. October 25, 2013. – 152 p.

4 Логинов С. С. Об уровнях управления в программно-конфигурируемой сети (SDN)/ T-Comm: Телекоммуникации и транспорт. 2017. Том 11. №3. С. 50-55

5 URL: http://flowgrammable.org/sdn/openflow/ (дата обращения 5.09.2020)

6 Wolfgang Braun and Michael Menth. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. Future Internet 2014,6, 302-336 p.

7 OpenFlow Switch Specification – March 26, 2015. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. p. 18-30 (дата обращения 1.09.2020)

8 Семенов Ю. А. Сетевая технология OpenFlow (SDN). URL: http://book.itep.ru/4/41/openflow.htm (дата обращения 1.09.2020)

9 Владыко А.Г, Матвиенко Н.А, Новиков М.И., Киричек Р.В. Тестирование контроллеров программно-конфигурируемой сети на базе модельной сети // Информационные технологии и телекоммуникации. 2016. Том 4. №1. С. 17-28.

10 Ю. Ю. Коляденко, Е. Э. Белоусова. Программно-конфигурируемые сети на базе протокола OpenFlow и их характеристики. Scientific Journal «ScienceRise» №3/2(20)2016 г.

11 В. А. Лихачев. Программно-конфигурируемые сети на основе протокола OpenFlow. Вестник ВГУ, серия: Системный Анализ и информационные технологии, 2014, №1

12 URL: https://www.virtualbox.org (дата обращения 10.09.2020)

13 Mininet документация. 13 декабрь 2019. URL: http://mininet.org (дата обращения 10.09.2020)

14 Сайт проекта OpenDaylight. URL: https://www.opendaylight.org (дата обращения 10.09.2020)

15 URL: https://wiki.wireshark.org/OpenFlow (дата обращения 10.09.2020)

16 M. Casado, M. J. Freedman, J. Pettit “Rethinking enterprise network control” IEEE/ACM Transactions on Networking, vol. 17, no. 4, pp. 1270-1283, 2009.

17 B. Stephens, A. Cox, W. Felter “Past: scalable ethernet for data centers” in Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, pp. 49-60, France, December 2012.

18 A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Sharma and S. Banerjee “DevoFlow: scaling flow management for high-performance networks” in Proceedings of the ACM SIGCOMM, pp. 254-265, August 2011.

19 M. Moshref, M. Yu, A. Sharma and R. Govinda “Scalable rule management for data center” in Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, pp. 157-170, USENIX Association, Berkeley, USA, 2013.

20 S. Ghorbani, C. Schlesinger, M. Monaco “Transparent, live migration of a software-defined network” in Proceedings of the ACM Symposium, ACM, Seattle, November 2014.

21 X. Pan, W. Sun. Address Resolution Delay Metric in Software-Defined Networking (SDN). April 22, 2015

22 Карташевский В. Г. Основы теории массового обслуживания: учебник для вузов. Горячая линия-Телеком, 2013 – 130 с.