ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)**

А.Г. Владыко, А.С. Мутханна, Р.В. Киричек, А.Н. Волков

ПРОГРАММИРУЕМЫЕ СЕТИ SDN

Учебное пособие

**Санкт-Петербург
2017**

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Прогресс есть претворение Утопий в жизнь.
(Оскар Уайльд)

Мир инфокоммуникационных технологий одной ногой стоит уже в сетях следующего (пятого) поколения, инфраструктура, предлагаемые услуги и приложения которой, порой реализуют утопические идеи, предложенные фантастами буквально последних 10-15 лет, не говоря уже о более ранних идеях. Уровень развития технологий на данный момент времени, как результат достигнутого прогресса в науке, дает как раз возможность начать реализацию тех утопических технологических решений, которые еще несколько лет назад мы могли наблюдать только в фильмах типа фэнтези.

Концепция 5G/IMT-2020 или иначе сети пятого поколения включает в себя целый комплекс концепцией и технологий, а не только описывает принципы организации сети мобильного доступа. В рекомендации ITU-R M.2083-0 Международного Союза Электросвязи (МСЭ) сети пятого поколения позиционируются как средство коммуникации для людей и машин, которые помогают в развитии других отраслей промышленности, таких как: медицина, транспорт (логистика), образование, индустрия и друге. Ожидается, что IMT-2020 на 2020 год и последующий этап развития обеспечит гораздо более высокие возможности. Ключевыми принципами проектирования сети IMT-2020 является гибкость, масштабируемость и разнообразие услуг, при этом МСЭ также выделяет следующие ключевые возможности сетей IMT-2020: пиковая скорость для получателя (измеряется в Гбит/с), задержка (например, для Тактильного интернета — не более 1 мс.), мобильность (должно быть обеспечено выполнение требований QoS при высокой скорости передвижения объекта/пользователя), энергоэффективность, высокая плотность подключенных устройств (IMT-2020 предполагает реализацию Интернета Вещей, что способствует увеличение количества подключенных устройств на единице пространства и соответственно нагрузка на аппаратуру сетевой инфраструктуры будет увеличена в несколько раз). Помимо вышеперечисленных требований, относительно традиционных сетей существует проблема удорожания оборудования и его поддержка (обновление программного обеспечения и т.д.), которое, в свою очередь не предусматривает способ быстрого внедрения самим оператором собственных приложений для предоставления новых сервисов/услуг. В результате тех требований, которые поставлены перед сетями пятого поколения, с целью достижения выполнения жесточайших критериев качества в новым услугам, например таким как: Тактильный интернет и Интернет Вещей, требуется проработать вопрос миграции сетевых технологий на те технологические решения (концепции), которые смогут обеспечивать стабильную работу сетевой инфраструктуры, ее масштабируемость, модульность, высокую абстракцию уровня управления, что

в конечном итоге позволит создать единую гибкую систему контроля и управления через стандартные программные интерфейсы.

В той же рекомендации Международного Союза Электросвязи (ITU-R M.2083-0) в качестве новых сетевых технологий, которые должны быть использованы при построении сетевой инфраструктуры являются — SDN/NFV.

Концепция программно-конфигурируемых или программируемых сетей (Software-Defined Networking, SDN) предполагает новый подход к организации сетевого взаимодействия, при котором уровни управления сетью и устройств передачи данных разделены, при этом функции уровня управления реализуются в отдельном узле, взаимодействующим с сетевыми устройствами.

Управляющее устройство в такой сети (контроллер) – концептуально централизованный «мозг», который знает топологию и состояние сети и принимает решения о правилах пересылки пакетов и доставки их до переадресованных объектов. Эта концепция расходится с классической моделью организации работы коммутаторов и маршрутизаторов в действующих сетях связи, в которой функции управления и передачи трафика объединены в рамках одного устройства(АПК решение). Этот подход позволяет значительно автоматизировать и облегчить управление сетями за счет возможности их программирования. Стандарты концепции SDN разрабатываются несколькими международными стандартизирующими организациями, форумами и консорциумами.

В книге рассматривается структура сетей SDN, а также взаимодействие с другими технологиями, контроллеры, тестирование и перспективы концепции SDN в Российской Федерации.

1. ОБЗОР КОНЦЕПЦИИ ПРОГРАММИРУЕМЫХ СЕТЕЙ SDN

Парадигма SDN была предложена специалистами и сотрудниками Стэнфордского и Калифорнийского университета в Беркли в 2002 году. Она сразу же была поддержана и привлекла внимание IT-компаний и инвесторов и стала активно развиваться.

Основной организацией занимающейся развитием и продвижением концепции SDN является ONF, основанная совместно крупнейшими мировыми компаниями Deutsche Telekom, Yahoo, Google, Verizon, Microsoft и Facebook. ONF (Open Networking Foundation) — некоммерческая организация, цели которой сосредоточены на развитии и разработке открытых стандартов. Помимо открытого сообщества разработчиков исследованиями в области SDN занимаются и другие сообщества IEEE, IETF, ITU-T, ONRC, IRTF. Ведущие телекоммуникационные вендоры оборудования и программного обеспечения, такие как: Brocade, Cisco, HP, IBM, Juniper и Nuage, также заинтересованы в развитии ПКС и активно участвуют в разработке новых спецификаций и стандартов. Но их продукты и решения в большинстве случаев являются проприетарными, в то время, как SDN изначально позиционировался как Open Source.

Стоит также отметить, что во многих литературных источниках существуют несколько вариантов перевода на русский язык аббревиатуры SDN, это программно-конфигурируемые сети, программно-определяемые сети, программируемые сети. По мнению специалистов, наиболее точным и правильным является - программируемые сети.

1.1. Прогноз увеличения трафика

Архитектура традиционных компьютерных сетей и Интернет закладывалась в конце 60-70-е годы, при создании первой сети ARPANET. Однако с тех пор произошли серьезные качественные и количественные изменения в области использования компьютерных сетей. В настоящий момент архитектура сетей устарела и не всегда способна быстро и эффективно реагировать на новые тенденции в развитии информационных технологий и потребности современного общества.

Одновременно с ростом количественных показателей нагрузки на компьютерные сети повысилась сложность управления сетью, значительно расширился перечень решаемых задач, их значимость и критичность, повысились требования к безопасности и надежности сетей.

На сегодняшний день определяющее значение для деятельности любой организации или компании играет скорость, с которой она способна адаптироваться в современных быстроменяющихся условиях. Важное влияние на способность к быстрой адаптации бизнес-процессов, взаимодействия с партнерами и клиентами играют сетевые технологии. Компьютерную сеть уже давно никто не рассматривает как совокупность соединенных между собой

компьютеров, на сегодняшний день она является совокупностью сервисов для предоставления услуг, платформой, которая начинена соответствующими сервисами. Прежде чем перейти к рассмотрению новых подходов к организации и управлению компьютерными сетями, мы должны рассмотреть в каких условиях работают сети, отметить появление целого ряда новых трендов, существенно повлиявших на вычислительную инфраструктуру. Среди них следует отметить: изменение модели вычислений (outsourcing & robosourcing); взрывной рост мобильности; быстрый рост трафика: к 2016 г. его объем увеличился в 6 раз; изменение структуры трафика: к 2016 г. 90 % – видеотрафик; несоответствие темпов роста трафика и доходов операторов.

По словам Эрика Шмидта, председателя совета директоров компании Google: «Пять экзабайт информации создано человечеством с момента зарождения цивилизации до 2003 г., но столько же сейчас создается каждые два дня, и скорость увеличивается. Люди не готовы к технологической революции, которая должна произойти».

Рост количества и разнородности контента, развитие сервисов и масштабов их охвата привели к изменению парадигмы организации вычислений – на место клиент-серверной архитектуры пришли ЦОД и облака, а файловые системы и базы данных трансформировались в сети хранения данных.

На июль 2013 г. каждые 60 секунд онлайн в Google посылается около 2 млн. поисковых запросов, на YouTube загружается 72 ч видео, на Facebook публикуется 41 тыс. постов, в Skype происходит 1,4 млн. соединений, отправляется свыше 200 млн. электронных писем. И трафик будет только увеличиваться.

По статистике Cisco Systems, глобальный IP-трафик за последние 5 лет возрос более чем в четыре раза и в течение ближайших 5 лет увеличится еще не менее чем в три раза. На Content Delivery Networks (CDNs) будет приходиться более половины интернет-трафика в 2018 г.

Во всем мире объем потребительского интернет-трафика составит не менее 70 % всего его объема в 2018 г. (для сравнения 2012 г. – 57 %). Этот показатель не включает peer-to-peer (P2P) трафик. Суммарно все виды видеотрафика (ТВ, video on demand (VoD), Интернет и P2P) будут составлять от 80 до 90 % глобального трафика к 2018 г.

Во всем мире объем потребительского интернет-трафика составит не менее 70 % всего его объема в 2018 г. (для сравнения 2012 г. – 57 %). Этот показатель не включает peer-to-peer (P2P) трафик. Суммарно все виды видеотрафика (ТВ, video on demand (VoD), Интернет и P2P) будут составлять от 80 до 90 % глобального трафика к 2018 г.

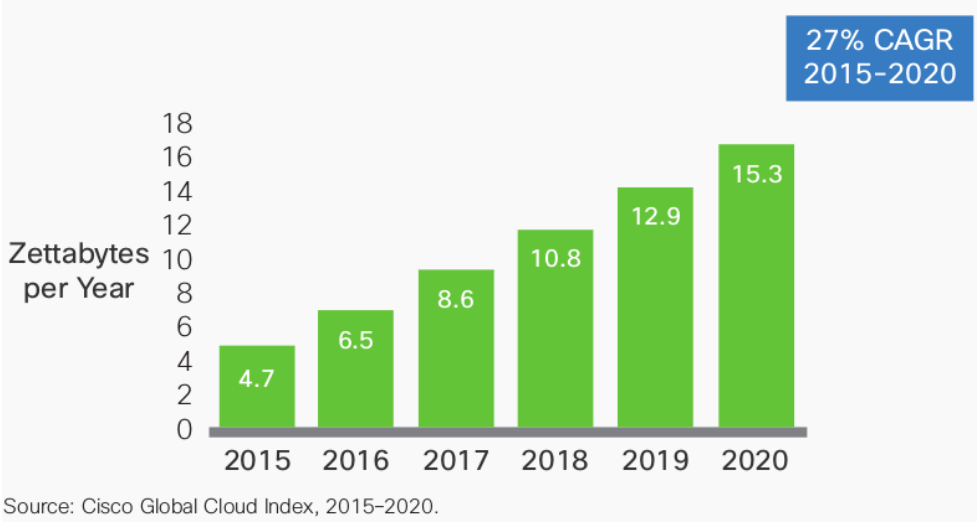


Рисунок 1.0 — Рост трафика IP

Согласно официальной статистике Cisco Systems прогнозируется, что объем глобального трафика сети Интернет и IP-сети WAN достигнут 2,3 ЗБ в год к 2020 году, количество уже в 2015 году глобального трафика центров обработки данных по оценкам, составляет 4,7 ЗБ и к 2020 году ожидается рост 15,3 ЗБ в год. Это увеличение представляет собой 27% CAGR.

Согласно исследованиям аналитической фирмы IHS Technology, мировой рынок средств программируемых сетей за период 2015-2020 гг. будет расти со среднегодовыми темпами около 98%.
На телекоммуникационном рынке за последние несколько лет наблюдается явная тенденция перехода к двум основным технологиям: программируемой сети (software-defined networking, SDN) и виртуализации сетевых функций (network functions virtualization, NFV), затрагивающих как основные сетевые, так и эксплуатационные характеристики сети и преследующих две главные цели – автоматизацию и скорость предоставления сервисов. Для обеих технологий – SDN и NFV – характерен сдвиг от традиционного аппаратного подхода к построению сети в сторону программного решения для организации ее работы.

Хотя операторы, внедряющие технологии SDN и NFV, говорят об успехе данных проектов, и число коммерческих внедрений растет, аналитики IHS полагают, что полное освоение этих технологий произойдет не раньше, чем через 10–15 лет. В Японии внедрение SDN начали компании NEC с NTT, но, несмотря на активность, коммерческое применение этой технологии пока невелико. Активно приступили к внедрению SDN и в Китае, где в 2015–2016 гг. отмечается множество крупных коммерческих проектов. Лидерами по объемам выручки от внедрения SDN стали провайдеры Северной Америки и Европы, и в 2015–2020 гг. их доля мирового рынка SDN может составить около

13%. Рынок SDN в целом в плоскости затрат сервис-провайдеров подразделяется на аппаратное оборудование, программное обеспечение и сервисы. IHS прогнозирует, что его объем вырастет с 289 млн долл., зафиксированных в 2015 г., до 8,7 млрд долл. в 2020 г. Сервисы, реализуемые программно и предоставляемые через аутсорсинг, в 2020 г. составят, по оценкам аналитиков, 46% от общего объема рынка SDN. А объем сегмента SDN-средств и программного обеспечения контроллеров может в 2020 г. достичь 1,8 млрд долл. Однако к 2020 г. вклад SDN-приложений в общий объем рынка окажется больше, чем от сегмента SDN-средств и программного обеспечения контроллеров, и цены будут умеренные по причине сильной конкуренции на этом рынке.

Более 70% опрошенных операторов в качестве одного из главных драйверов развития SDN/NFV называют сокращение сроков модернизации сети за счет упрощения сетевой инфраструктуры. Почти 2/3 опрошенных операторов в тройку основных драйверов развития SDN и NFV отнесли снижение OPEX.

1.2. Архитектура SDN

Концепция SDN создавалась с целью отделить уровень транспорта данных (data plane) от уровня управления (control plane), посредством вынесения логики управления сетью в централизованный узел – контроллер. Это позволило сделать устройства сети проще и, следовательно, дешевле. Программируемая сеть предусматривает в своей архитектуре три уровня: уровень приложений, уровень управления, уровень инфраструктуры(передачи данных). Концептуально данные уровни отображены на рисунке 1.1.

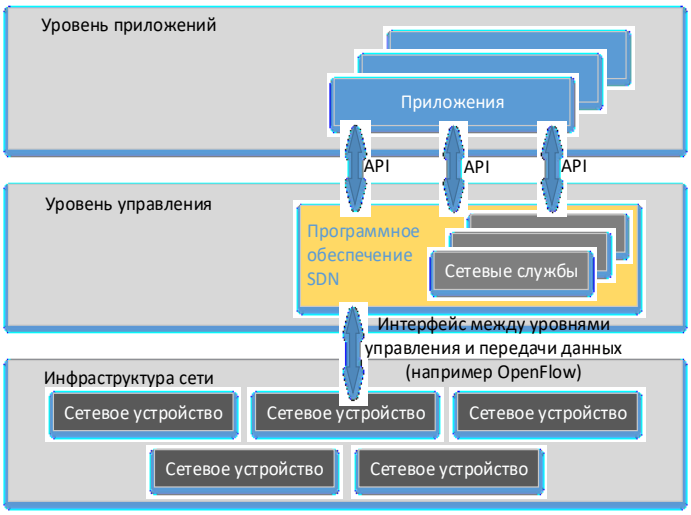


Рис. 1.1 Общая архитектура программно-конфигурируемой сети

Уровень инфраструктуры.

Уровень инфраструктуры (передачи данных) представляет собой устройства программно-конфигурируемой сети, функционирующие по протоколу OpenFlow и находящиеся под управлением контроллера сети SDN. Такими устройствами на данный момент являются OpenFlow коммутаторы.

Уровень управления.

На данном уровне сосредоточена вся логика управления сетью. Управление осуществляется контроллером (сетевой операционной системой), связанной с уровнем приложений по API интерфейсу, называемому «северный мост» (northbound interface). При этом контроллер может быть представлен в виде АПК решения или ПО.

Уровень приложений. На уровне приложений реализуются те сервисы и услуги, которые необходимы для управления сетью, конфигурирования сети, обеспечения заданного качества обслуживания (QoS), мониторинга и оптимизации сети, обеспечения безопасности (аутентификация, межсетевые экраны).

Приложение SDN представляет собой программное обеспечение (ПО), работающее через прогрммный интерфейс контроллера для автоматической подстройки сети под конкретное бизнес приложение, к примеру, Microsoft Lync. Например, в данном случае может быть произведено изменение QoS сети между двумя телефонными абонентами для создания между ними сеанса видеосвязи высокого качества (HD) в реальном времени без задержек или создание VPN туннеля между двумя абонентами.

1.3 Международная стандартизация SDN

Разработкой стандартов SDN и изысканиями в этом направлении в настоящее время занимается большое число отраслевых объединений и международных организаций, в которых участвуют заинтересованные коммерческие компании – вендоры сетевого оборудования, операторы сетей связи, разработчики программного обеспечения (ПО).

Концепция SDN активно продвигается, однако ее понимание в полной мере еще не устоялось, а ключевые стандарты находятся на различных стадиях разработки и апробирования, поэтому сегодня общепринятое определение понятия SDN отсутствует. Отраслевые стандартизирующие организации: Фонд открытых сетевых технологий (Open Networking Foundation, ONF), Рабочая группа по инженерным задачам Интернет (Internet Engineering Task Force, IETF) и МСЭ-Т (Сектор стандартизации Международного союза электросвязи) – предлагают следующие определения:

МСЭ-Т. SDN – технология построения сетей, которая позволяет реализовать централизованный, программируемый уровень управления и изоляцию (абстракцию) уровня данных; при этом уровни управления и данных разделены, благодаря чему операторы сетей связи могут напрямую управлять своими виртуальными ресурсами и сетями. Программируемый уровень управления – уровень управления, который должен быть программируемым и

управляемым централизованным образом; изоляция (абстракция) уровня данных – модели уровня данных должны быть абстрактными и упрощенными, а не специализированным аппаратным обеспечением.

IETF. SDN – подход к построению сетей, обеспечивающий прямое управление ресурсами и сетями, а также их распределение за счет добавления собственных средств обработки, администрирования и программного управления посредством открытых сетевых интерфейсов и абстракции (абстрагирования, изоляции) уровня сети.

ONF. SDN – динамичная, управляемая и адаптируемая сетевая архитектура, в которой разделены уровни управления сетью и передачи данных, что обеспечивает программное управление сетью и абстрагирование/изоляцию (уровня) сетевой инфраструктуры от (уровня) приложений и сетевых услуг/сервисов.

Исследованием общих вопросов и стандартизацией SDN занимаются ONF, IETF, Исследовательская группа интернет-технологий (Internet Research Task Force, IRTF), Европейский институт по стандартизации в области телекоммуникаций (European Telecommunications Standards Institute, ETSI) и МСЭ-Т. Представители Форума широкополосных сетей (Broadband Forum, BBF) изучают некоторые частные вопросы построения и эксплуатации сетей SDN.

Стандарты ONF.

В 2011 г. компании Facebook, Deutsche Telekom, Microsoft, Verizon и Yahoo! организовали консорциум ONF с целью развития технологий SDN в целом и протокола OpenFlow в частности. Сегодня членами ONF являются практически все основные поставщики сетевого оборудования, включая Alcatel-Lucent, Brocade, Ciena, Cisco, Dell, Ericsson, Extreme Networks, HP, Huawei, IBM, Infinera, Intel, Juniper Networks, Mellanox, Netgear, Nokia Solutions and Networks, ZTE, а также лидеры рынка систем виртуализации VMware и Citrix.

Основной задачей ONF является представление стандарта OpenFlow, который позволяет осуществлять удаленное управление уровнем передачи данных. Стандарт OpenFlow, первый стандарт SDN, является существенным элементом в открытой архитектуре SDN. В настоящее время в ONF продолжается работа по анализу требований к SDN, развитию стандарта OpenFlow в соответствии с запросами, возникающими при коммерческом развертывании SDN, а кроме того, создаются новые стандарты в целях расширения возможностей SDN.

Исследования в ONF по концептуальным и архитектурным вопросам, по разработке стандарта OpenFlow и стандартов совместно с мировыми экспертами в области SDN проводятся в рамках рабочих групп (РГ):

- Architecture and Framework (архитектура): результатом работы РГ станут документы по архитектуре и концептуальным положениям SDN, в которых должны быть определены терминология SDN и требования к

архитектуре, разработана сама архитектура, исследованы вопросы взаимодействия с существующими сетями, включая протоколы маршрутизации и OAM, разработаны принципы защиты и восстановления в сети и др.;

- Forwarding Abstractions (изоляция уровня передачи данных): РГ занимается разработкой средств унификации доступа к функциям аппаратных составляющих коммутатора, обеспечивающих передачу данных. Создание абстракций для работы сетевого оборудования позволит избежать излишней детализации передаваемых инструкций и необходимости связывать их с технической реализацией отдельных функций в сетевых устройствах различных производителей;
- Optical Transport (оптический транспорт): исследования РГ включают в себя разработку вариантов применения SDN и OpenFlow в оптических транспортных сетях, определение целевой эталонной архитектуры управления ONT, в которых применяется протокол OpenFlow, разработку расширений протокола OpenFlow, а также участие в разработке эталонной архитектуры SDN и терминологии, моделировании оптических коммутаторов и сетей, виртуализации сети OTN. РГ планирует разработать информационную модель обобщенных оптических транспортных сетей и оптических коммутаторов, соответствующие модели данных, которые могут использоваться протоколами ONF, а также предложить решения и расширения к протоколам ONF, определяющие требования к SDN и стандарту OpenFlow для управления ONT;
- Configuration and Management (конфигурирование и управление): РГ занимается вопросами OAM, разработкой рекомендаций по использованию механизмов мониторинга физических и логических сетей на базе коммутаторов OpenFlow, каналов и путей для определения и локализации неисправностей, мониторинга качества функционирования. Исследуемые механизмы включают определение топологии физических и логических сетей, мониторинг физических и логических каналов между коммутаторами, мониторинг путей в логических сетях OpenFlow. РГ разрабатывает принципы передачи сообщений о состоянии (например, о неисправности канала, изменении конфигурации, потере соединений с контроллером и др.) от коммутаторов OpenFlow к контроллеру OpenFlow и другим элементам сети, а также разрабатывает спецификации для состояний коммутатора OpenFlow, в которых передаются сообщения, семантика и содержание сообщений;
- Market Education Committee (исследование рынка): РГ осуществляет образовательную деятельность в сообществе SDN в целях продвижения сетей SDN на базе стандарта OpenFlow и самих стандартов ONF;
- Testing and Certification (тестирование и сертификация): РГ разрабатывает методологию и тесты (включая проверку качества функционирования) в целях тестирования и сертификации оборудования SDN, а также требования к сертификационным лабораториям и средствам (программным и аппаратным) для проведения тестирования;

- Extensibility (расширяемость): РГ занимается развитием протокола OpenFlow, включая разработку прототипов (опытных образцов) для каждой новой функции, добавляемой в протокол;
- Migration (миграция): РГ разрабатывает методы перевода (миграции) сетевых услуг из традиционных сетей связи на сеть SDN, построенную на базе протокола OpenFlow;
- Wireless and Mobile (беспроводные и подвижные радиотелефонные сети): РГ исследует вопросы применения концепции SDN и протокола OpenFlow в беспроводных и подвижных радиотелефонных сетях, включая разработку архитектуры, требований к семейству протоколов OpenFlow, требований по безопасности и др., а также вариантов применения концепции SDN;
- Northbound Interface: РГ занимается разработкой API для «северного» интерфейса;
- Discussion Groups: «дискуссионные группы» ONF организуют форумы по тематике SDN.

В состав РГ и групп по переписке входят только сотрудники компаний-членов ONF.

В настоящее время ONF уже разработаны следующие документы:

- спецификация протокола и коммутатора OpenFlow, текущая версия 1.4.0 (08/2013), предыдущие версии 1.0.x, 1.3.x и расширения к ней; версия 1.5.0.
- спецификация протокола конфигурации коммутаторов OpenFlow и построения сетевой среды OF-Config, версия 1.2 (2014);
- структура оповещений о событиях OpenFlow, версия 1.0;
- спецификация тестов на соответствие стандарту OpenFlow Switch, версия 1.0.1.

Стандарты IETF.

Работа по вопросам SDN в IETF, как и в других организациях распределена между рабочими группами (РГ). Сообщества ученых, операторов сетей связи и производителей сетевого оборудования, участвуя в данных рабочих группах занимаются такими тематиками как: маршрутизация, транспорт, безопасность и прочие. Стоит отметить, что разработка аналитических и стандартизирующих документов IETF, которые относятся к Программируемым сетям , стартовала еще в конце 2012 года. Рассмотрим следующие рабочие группы: I2RS, PCE.

Рабочая группа I2RS (Interface to the Routing System, интерфейс к системе маршрутизации), в свою очередь была создана в конце 2012 года с целью разработки протокола I2RS, который обеспечивает взаимодействие с системой маршрутизации сети посредством протоколов и интерфейсов управления (администрирования). Данный протокол реализует возможность управления действующими сетевыми устройствами при условии сохранения за

ними уровня управления в части маршрутизации передаваемых пакетов данных. В данный момент целью РГ является разработка высокоуровневой архитектуры I2RS и ее основных составляющих, что в итоге позволит разработать информационную модель и сформулировать требования по протоколам и форматам передачи данных для I2RS.

Выделяют следующие направления работы РГ I2RS:

- архитектура I2RS, с учетом вопросов политик управления и безопасности;
- информационные модели, соответствующие частным сценариям применения;
- анализ существующих протоколов и форматов представления данных IETF и других организаций на соответствие требованиям;
  сценарии работы I2RS в частных сценариях применения, в том числе взаимодействие с данными маршрутизации, управления и анализ работы протокола BGP, контроль и оптимизация трафика, распределенное реагирование на сетевые атаки, маршрутизация на уровне услуг, получение информации о топологии сети (однако, формирование сетевой топологии на данный момент не рассматривается);

В настоящее время действующими являются проекты документов РГ по постановке проблемы I2RS, архитектуре I2RS и базовой информационной модели данных маршрутизации. К тематике РГ I2RS также относятся документы IETF, касающиеся работы сетевых служб (балансировка нагрузки, предотвращение вторжений, функциональность сетевых экранов и пр.) в I2RS, модели публикации-подписки для событий в I2RS, сценариев работы наложенной сети применительно к I2RS и перехода I2RS на IPv6, информационной модели сетевых топологий, протоколо-независимых сценариев работы I2RS.

Рабочая группа PCE (Path Computation Element, Элемент Вычисления Пути), была уже относительно давно создана, еще в 2005 году, с целью разработки концепции PCE и протокола PCEP (PCE Communication Protocol), обеспечивающего взаимодействие с PCE. В свою очередь, сама концепция PCE предполагает вынесение в отдельную логическую составляющую алгоритма вычисления путей в сети. Рабочая группа разрабатывает PCEP и необходимые для взаимодействия клиенты вычисления пути (Path Computation Clients, PCC), а также механизмы взаимодействия PCE между собой, включая средства аутентификации и защиты данных (конфиденциальности).

Выделяют следующие направления работы РГ I2RS:

- расширение протокола PCEP для моделей расчета LSP, в том числе расчета основных, резервных и защищенных путей, оптимизация на локальном или глобальном уровне и балансировки нагрузки;
- разработка расширений для сигнализации RSVPTE с целью поддержки PCE (совместно с другими рабочими группами);

- разработка спецификации расширений протокола PCEP для взаимодействия в сетях GMPLS, включая сети WSON (Wavelength Switched Optical Network);
- определение расширений PCEP для вычисления путей в многоуровневых сетях;
- определение расширений PCEP, предлагающих новые пути для существующего или нового LSP элементом PCE, хранящим состояния (stateful).

К тематике рабочей группы PCE также относятся документы IETF, рассматривающие расширения протоколов IS-IS, OSPF с целью обнаружения элементов PCE.

Рекомендации МСЭ-Т.

МСЭ-Т занимается исследованием технических, эксплуатационных, тарифных и других вопросов, разрабатывает рекомендации с целью стандартизации электросвязи на международном уровне. Как уже было сказано в начале данной книги, Международный Союз Электросвязи Программируемые сети (SDN) в совокупности с виртуализацией сетевых функций (NFV) видит, как основные (базовые) технологии построения сетевой инфраструктуры сетей связи пятого поколения 5G/IMT-2020. На данный момент, в МСЭ-Т (сектор стандартизации) существует13 SG исследовательская комиссия, целью работы которой является — выработка основных рекомендаций, утверждающих определения, термины, архитектуры и используемые технологические решения, при построения сетей пятого поколения. Также 11 Исследовательская группа является ведущей исследовательской группой сектора стандартизации МСЭ по сигнализации и протоколам. Изучение требований к сигнализации и протоколам управления сетевыми ресурсами в 11 Исследовательской группе расширился до новых областей исследования и стандартизации ITU-T, таких как: SDN (Software-Defined Networking, Программируемые сети) и NFV (Network Function Virtualization, Виртуализация сетевых функций), так как считается, что именно решения SDN и NFV смогут удовлетворить поставленные жесточайшие требования к сетям пятого поколения 5G/IMT-2020 и их новых услуг, таких как: Тактильный Интернет, e-Health и др. Согласно тех требований, которые поставлены перед сетями пятого поколения, с целью достижения выполнения жесточайших критериев качества к новых услугам, например, таким как: Тактильный Интернет и Интернет вещей, требуется проработать вопрос миграции сетевых технологий на те технологические решения(концепции), которые смогут обеспечивать стабильную работу сетевой инфраструктуры, ее масштабируемость, модульность, высокую абстракцию уровня управления, что в конечном итоге позволить создать единую гибкую систему контроля и управления через стандартные программные интерфейсы. Также одним из принципов, согласно которым разрабатывается инфраструктура сетей пятого поколения - «Network Slicing», где «Network Slice», иначе говоря «Сетевой

срез» - это логическая сеть, которая обеспечивает определенные сетевые возможности и характеристики (ITU-T Y.3100), то есть по-сути является «программируемой единицей», которая позволяет логически изолировать предоставляемые сетевые функции, например: «slice» для организации услуги VoIP, E-health и т. п.

Международный Союз электросвязи (МСЭ-Т) определяет программируемую сеть, как технологию построения сетей, которая позволяет реализовать централизованный, программируемый уровень управления и изоляцию уровня данных. В настоящее время исследованиями в области SDN в МСЭ-Т занимаются ИК 13 (архитектуры и функциональные требования к SDN) и ИК 11 (эталонные архитектуры сигнализации SDN, требования к сигнализации и протоколы SDN, включая протоколы взаимодействия, а также тестирование на соответствие и взаимодействие). Интерес к концепции Программируемых сетей проявляют также ИК 15 (транспорт в SDN) и ИК 17 (безопасность в SDN). В ИК 13 SDN занимается РГ 3/13 «SDN и сети будущего» в рамках исследовательского вопроса (ИВ) 11 «Развитие сетевых технологий и услуг,ориентированных на пользователя, и взаимодействие с перспективными сетями, включая SDN» и ИВ 14 «Сети SDN и функционирование перспективных сетей с учетом оказываемых услуг», а также РГ 2/13 «Облачные вычисления и основные возможности» в части прикладных вопросов — QoS, безопасность, мобильность.

На данный момент, основные рекомендации Исследовательских групп, как принятые, так и находящиеся на стадии предложенных и рассматривающихся:

- Y.3300 «Framework of software-defined networking» - Структура Программируемых сетей;
- Y.3301 «Functional requirements of software-defined networking»;
- Y.3302 «Functional architecture of software-defined networking»;
- Y.3320 «Requirements for applying formal methods to software-defined networking»;
- Y.3321 «Requirements and capability framework for NICE implementation making use of software-defined networking technologies»;
- Y.3011 «Framework of network virtualization for future networks» - структура виртуализации сетевых функций для сетей будущего;
- Y.3100 «Terms and definitions for IMT-2020 network» - Термины и определения для сетей 5G/IMT-2020;
- Y.3111 «IMT-2020 Network Management Framework» - Структура управления сетью IMT-2020;
- Y.3110 «IMT-2020 Network Management Requirements» - Требования к управлению сетью IMT-2020;
- Y.3100-series Supplement 44, «Standardization and open source activities related to network softwarization of IMT-2020»;
- Q.IN-DS «Identification of network elements using digital signature»;

- Q.IN-DS «Identification of network elements using digital signature»;
- Q.ETN-DS «Signalling architecture of the moveable telecommunication network to be used in a natural disaster»;
- Q.PR-MSA «Protocol for managing services and applications with requested network parameters in IMT-2020 networks»;
- Y.SDN-CT «Framework of SDN controller testing»;
- Q.HEE_D2D_CP «Hybrid energy efficient D2D communication protocol for IMT 2020»;
- Q.IEC-REQ: Proposed enhancement for «Signalling requirement of intelligent edge computing»;
- Y.IMT2020-frame, «Framework of IMT-2020 network»;
- Y.IMT2020-reqts, «Requirements of IMT-2020 network»;
- Y.NSOM, «Network slicing orchestration and management»;
- Y.IMT2020-MultiSL, «Framework for the support of Multiple Network Slicing»;
- Y.IMT2020-BM, «Business models of IMT-2020»;
- Y.IMT2020-arch, «Architecture of IMT-2020 network»;
- и другие.

Можно сделать вывод, что в данный момент ведется активная работа по стандартизации в области сетевой инфраструктуры сетей пятого поколения 5G/IMT-2020, в этой работе принимают несколько Исследовательских комиссий. В основу сетевой инфраструктуры, как уже выше было отображено, кладутся технологии программируемых сетей и виртуализации сетевых функций, которые в свою очередь благодаря своим архитектурным решениям позволят реализовать соответствующие поставленные требования качества услуг и сервисов сетей пятого поколения.

Стандарты ETSI.

Институт ETSI — это некоммерческая организация по разработке стандартов в области телекоммуникаций, официально признанная Европейским союзом. Институт разрабатывает стандарты фиксированной, подвижной, радио, конвергентной связи, а также телевещания и интернет-технологий.

В свою очередь, в составе ETSI была выделена группа отраслевой спецификации (Industry Specification Group, ISG) по разработке концепции виртуализации сетевых функций (служб) (Network Functions Virtualization, NFV). Концепция NFV предполагает замещение разнообразных сетевых устройств стандартизированными высокопроизводительными серверами, коммутаторами и системами хранения данных с реализацией сетевых функций (служб) программным обеспечением.

Виртуализируемые сетевые функции могут включать:

- Коммутирующее оборудование BRAS, маршрутизаторы, функции NAT;

- узлы сети подвижной радиотелефонной связи MME, SCSF, HLR/HSS, SGSN, GGSN, PDN-GW, Node B, eNode B;
- функции пользовательских устройств, включая маршрутизаторы и STB;
- шлюзы IPSec/SSL VPN;
- функции анализа трафика DPI, оценки QoE;
- функции обеспечения качества услуг, включая мониторинг SLA, тестирование и диагностику;
- функции сигнализации NGN в SBC, IMS;
- общесетевые функции сервера AAA, контроля политик, тарификации;
- функции прикладного уровня в сети распределения контента CDN и серверах кэширования, балансировки нагрузки;
- функции безопасности, включая межсетевые экраны, антивирусные средства, системы обнаружения вторжений, средства защиты от спама.

Реализация концепции NFV также должна затронуть системы OSS/BSS.

В настоящее время в число участников ISG NFV входят операторы сетей связи, производители сетевого оборудования, компании-разработчики ПО и вычислительного оборудования: AT&T, BT Group, China Mobile, Deutsche Telekom, KDDI, NTT DoCoMo, Orange, Telefonica, Verizon UK, Amdocs Software Systems, Alcatel-Lucent, Cisco Systems, Citrix Systems, France Telecom, Hewlett-Packard, IBM Europe, Intel, Juniper Networks, Nokia Solutions and Networks, Vodafone Group Services, ZTE Corporation и др.

Предполагается, что ISG NFV разработает требования и архитектуру NFV, рассмотрит вопросы управления, оркестровки служб, архитектуры ПО, производительности и переносимости, надежности и устойчивости, безопасности и миграции к NFV. Концепция NFV сходна с концепцией SDN, однако на текущем, начальном, этапе работы степень их взаимного соотношения не является точно определенной.

Согласно ETSI, концепция NFV в большой степени дополняет SDN, но концепции независимы, каждая из них может быть реализована отдельно. Сегодня стандарты группы ISG NFV находятся на начальной стадии разработки. В октябре 2013 г. группой были разработаны первые спецификации.

На семинаре ETSI по вопросам сетей будущего было объявлено о рассмотрении в рамках РГ ISG AFI (Autonomic network engineering for the self-managing Future Internet) вопросов использования возможностей протокола IPv6 при автономном администрировании и управлении, в том числе применительно к сетям SDN. Также РГ планирует разработку требований к использованию специфических возможностей IPv6 в автономных сетях эталонной архитектуры SDN и с возможным учетом концепции SDN в эталонных архитектурах подвижной радиотелефонной связи 3GPP и не-3GPP,

NGN/IMS, BBF, TISPAN CDN, беспроводных самоорганизующихся/ячеистых топологиях и сенсорных сетях.

1.4 Протоколы SDN

Консорциум Open Networking Foundation был создан рядом компаний, таких как: Facebook, Deutsche Telekom, Microsoft, Verizon и Yahoo, с целью развития технологий SDN в целом и протокола OpenFlow в частности. Сегодня членами ONF являются практически все основные поставщики сетевого оборудования, включая Alcatel-Lucent, Brocade, Ciena, Cisco, Dell, Ericsson, Extreme Networks, HP, Huawei, IBM, Infinera, Intel, Juniper Networks, Mellanox, Netgear, Nokia Solutions and Networks, ZTE, а также лидеры рынка систем виртуализации VMware и Citrix.

Основной задачей консорциума является разработка стандарта OpenFlow, который является протоколом управления коммутаторами (уровня передачи данных) контроллером SDN. Стандарт OpenFlow – первый стандарт в области Программируемых сетей. В настоящий момент в ONF продолжается работа по анализу требований к Программируемым сетям, развитию стандарта протокола SDN в соответствии с запросами, возникающими при коммерческом развертывании SDN, а кроме того, разрабатываются и представляются новые стандарты протокола, которые по-сути являются результатом его эволюционного развития.

Консорциум Open Networking Foundation (ONF), являясь разработчиком открытых стандартов SDN, выделяет следующие протоколы взаимодействия различных уровней архитектуры:

- OpenFlow – открытый расширяемый протокол установления соединения между контроллером и коммутатором, непосредственного управления потоками и формирования правил их обработки коммутатором;
- OF-CONFIG – открытый расширяемый протокол конфигурирования сетевой среды и управления ею (operational context), включая виртуальные и физические устройства;
- NB-API (NorthBound API) – API «северного» интерфейса, предоставляемый контроллером сетевым приложениям.

ONF предполагает, что инфраструктурный уровень представлен специализированными коммутаторами OpenFlow (OpenFlow-only), которые работают только в рамках идеологии OpenFlow или гибридными коммутаторами (OopenFlow-hybrid). Поддерживающими одновременную работу с классической и OpenFlow–управляемой сетью, включая коммутацию пакетов между ними. Стоит отметить, что на данный момент развития и внедрения концепции Программируемых сетей на практику, на телекоммуникационном рынке коммутаторы SDN представлены в большей своей части в качестве гибридной реализации и OpenFlow, как протокол является одним из программных модулей коммутатора, как и любой другой поддерживаемый коммутатором протокол. Данный подход обусловлен

экономической составляющей и позволяет реализовать «мягкий» переход сетей на новую концепцию. Эволюционный подход к реализации на практике программируемых сетей позволяет проводить более активную доработку стандартов SDN большинством международных организаций и проработку множества вопросов, которые возникают в процессе эксплуатации, тем самым усовершенствуя и развивая концепцию. Не малую роль здесь также играют и открытые сообщества разработчиков, благодаря которым происходит активная работа по разработке (реализации) SDN, например, уже существует ряд открытых (opensource) реализаций контроллеров SDN, а также существует открытая реализация программного коммутатора, при этом производится активная поддержка данных проектов и частый выпуск обновлений. Как уже было отмечено ранее, на данный момент существует уже ряд реализаций стандарта протокола OpenFlow, при этом, на рынке чаще представлены коммутаторы (в программном или аппаратно-программном исполнении) с поддержкой не самых последних версий протокола OpenFlow. Чаще встречаемыми версиями являются: OpenFLow 1.0, 1.1, 1.3. С точки зрения применения Программируемых сетей на практике, версия протокола OpenFlow 1.3 – является более актуальной, так как в данной версии реализован в большей мере необходимый функционал для обеспечения гибкости управления потоками на уровне передачи данных, по сравнению с версиями 1.0 и 1.1.

Если рассматривать коммутатор SDN, абстрагировавшись от его физической реализации, то в коммутаторе с поддержкой протокола версии 1.3 можно выделить следующие функциональные блоки: одна или несколько таблиц потоков, таблица групп и канал OpenFlow для взаимодействия с контроллером.

Записи в таблицах потоков и групп определяют порядок обработки поступающих на входы коммутатора пакетов. Взаимодействие по каналу OpenFlow (канал взаимодействия с контроллером Программируемой сети) осуществляется с использованием протокола OpenFlow, его сообщения таблицами потоков и групп не обрабатываются (рис 1.3).

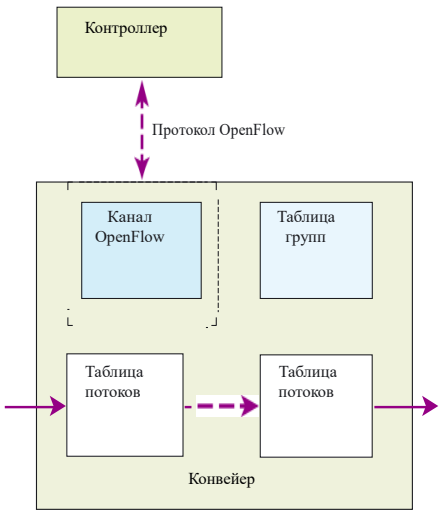


Рис.1.3. Абстрактная модель коммутатора SDN с поддержкой протокола OpenFlow.

Примечание [A.N.V1]: Требуется ли обновление (или переисовка данной картинки 1.3) ?!

1.4.1 Протокол OpenFlow

Как уже было отмечено выше, протокол OpenFlow является открытым стандартом, разрабатываемый консорциумом ONF. Следуя эволюционному пути развития сете связи, протокол OpenFlow, на данном этапе развития и реализации концепции SDN на практике, внедряется в виде программного модуля в аппаратные реализации коммерческих Ethernet коммутаторов, маршрутизаторов и беспроводных узлов доступов, в качестве расширения возможностей и их применения в качестве SDN устройств.

Стандарт OpenFlow в настоящее время де-факто принят большинством производителей сетевого оборудования. На телекоммуникационном рынке, в настоящее время доступны коммутаторы с поддержкой OpenFlow таких производителей как Cisco, Juniper, Brocade, Huawei, Zelax, B4N и др.

Сообщения протокола OpenFlow принято делить на три типа:

Сообщения *контроллер-коммутатор* – инициализируются на контроллере, служат для управления коммутатором и контролем за событиями, происходящими на нем.

К данному типу сообщений относятся следующие сообщения:

– *Features*: данное сообщение служит для запроса контроллером возможностей коммутатора; коммутатор в свою очередь отвечает на такой запрос ответом features, в котором обозначает свои возможности. Такой процесс происходит при открытии OpenFlow канала.

– *Configuration*: настоящим сообщением контроллер запрашивает и устанавливает параметры настройки коммутатора.

– *Modify-State*: эти сообщения отсылаются сетевой ОС для управления состоянием коммутаторов. Задача сообщений добавление, удаление правил и изменение OpenFlow таблиц и настройка портов коммутатора.

– *Read-State*: данные сообщения отвечают за сбор статистики коммутаторов.

– *Packet-out*: сообщения Packet-out используются контроллером для отправки пакетов из определенного порта на коммутаторе и пересылке пакетов, полученных с помощью сообщения Packet-in. Содержат целый пакет или идентификатор ID буфера, ссылающегося на пакет, загруженный в коммутатор. Сообщение должно содержать список действий, которые применяются в указанном порядке: если список действий пуст, то пакет сбрасывается.

– *Barrier*: сообщения Barrier обеспечивают установление зависимостей между сообщениями или оповещения о завершенных операциях. Используются при необходимости обработки сообщений в определенном порядке.

– *Role-Request*: данный запрос служит для изменения приоритета контроллера на коммутаторе (для повышения роли с Slave до Master).

– *Asynchronous-Configuration*: с помощью данного сообщения контроллер устанавливает фильтр на асинхронные сообщения от коммутаторов.

Вторым типом сообщений являются *Асинхронные сообщения*, которые инициализируются OpenFlow-коммутаторами, предназначены для извещения контроллера о событиях на сети, например, сбои, ошибки, изменения состояния.

К данному типу сообщений относятся следующие сообщения:

– *Packet-in*: данное сообщение коммутатор инициирует и отсылает контроллеру в случае если пришедший пакет не имеет нужного правила в таблице коммутации. Для всех пакетов, пересылаемых в виртуальный порт CONTROLLER, сообщение Packet-in отправляется в контроллер.

– *Flow-Removed*: сообщение для удаления правил, которые не используются и неактивны.

– *Port-status*: генерируются коммутатором на контроллер в случае изменения настроек порта.

– *Error*: этим сообщением контроллер извещает контроллер о произошедших на нем ошибках или сбоях.

Третьим типом сообщений являются *Симметричные сообщения*, которые рассылаются как коммутаторами, так и контроллером.

К данному типу сообщений относятся следующие сообщения:

– *Hello*: сообщения, обмен которыми между коммутатором и контроллером происходит при установлении соединения.

– Echo: сообщения вида запрос/ответ могут инициироваться и контроллером, и коммутатором, при условии, что обязательно будет получен ответ. Также могут служить для измерения задержек или пропускной способности соединения контроллер-коммутатор, а также проверки эффективности соединения.

Experimenter: сообщения Experimenter предназначены для обеспечения дополнительной функциональности, при проведении экспериментов в пространстве типов сообщений OpenFlow.

1.4.3 Протоколы NB-API

До середины 2013 г. ONF (Open Networking Foundation) целенаправленно не разрабатывал и не накладывал требований на NB-API (Northbound Application programming Interface, Северный программный интерфейс), реализуемые контроллерами для организации взаимодействия с сетевыми приложениями. Таким образом, ONF опирался на практику, в большей степени характерную для отрасли разработки ПО, нежели для сетевого оборудования. Решения, создаваемые для реализации функций

контроллеров, предлагают собственные API, конкурирующие в рыночных условиях. Предполагалось, что наиболее популярный, сформированный при прямом взаимодействии различных участников рынка (разработчиков контроллеров и разработчиков сетевых приложений), API станет де-факто стандартом.

Сложившаяся тенденция в практических решениях SDN предлагает два типа реализации NB-API интерфейса:

- REST (или RESTful) - Representational State Transfer – архитектурное решение взаимодействия компонент распределенного приложения в сети, на основе модели «клиент-сервер», в основе обычно используется протокол HTTP и форматы передачи данных - *.xml, *.json.
- Программный интерфейс взаимодействия ПО (например, Java API платформы Java EE или Java OSGI API), который непосредственно зависит от реализации контроллера Программируемой сети.

Обычно, как в первом (REST API), так и во втором случае, существует документация на предоставляемый программный интерфейс, а также возможные ответы системы (в данном случае контроллера) на соответствующие формируемые запросы. Тем самым, контроллер Программируемой сети, с точки зрения администратора или стороннего разработчика приложений под контроллер, выглядит как «черный ящик» с известным набором функций и реакций на них.

В настоящее время в ONF создана рабочая группа Northbound Interface («Северный интерфейс»), в планы которой входят разработка функциональных требований к базовым API контроллера и реализация базового эталонного API.

Ключевые слова

Концепция Программируемых сетей, SDN, прогноз увеличения трафика, архитектура, международная стандартизация, протоколы SDN.

1.4.2 Протокол OF-CONFIG

Данный протокол разработан для решения задач более высокого, по сравнению с протоколом OpenFlow, уровня. В основном это задачи по построению сетевой среды в целом, конфигурации коммутаторов и принятию решений, например, о закрытии или открытии отдельных портов. Протоколом OF-CONFIG предусмотрены следующие абстракции (рис. 1.4):

- логический коммутатор OpenFlow — абстракция узла передачи данных OpenFlow. Протокол OF-CONFIG позволяет осуществить конфигурацию логического коммутатора OpenFlow так, чтобы контроллер OpenFlow мог взаимодействовать с ним и управлять им по протоколу OpenFlow;
- OpenFlow-совместимый коммутатор — физический или логический сетевой элемент, ресурсы которого (порты, очереди и пр.) выделены одному

или нескольким логическим коммутаторам OpenFlow. Протокол OF-CONFIG позволяет динамически назначать ресурсы OpenFlow-совместимого коммутатора размещенным на нем логическим коммутаторам OpenFlow;

- точка конфигурации OpenFlow — источник сообщений OF-CONFIG для OpenFlow-совместимых коммутаторов. Сущность точки конфигурации OpenFlow и взаимодействие точек конфигурации с контроллерами OpenFlow в настоящий момент спецификациями ONF не регламентируются.

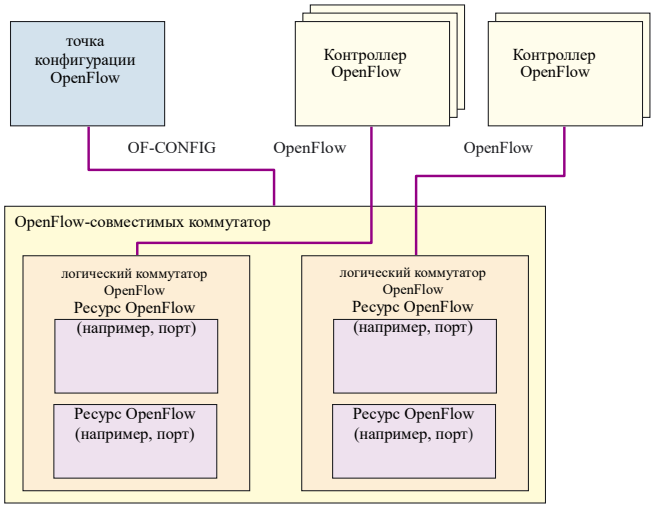


Рис. 1.4. Основные абстракции протокола OF-CONFIG

Примечание [A.N.V2]: Требуется ли перерисовка даной картинки (1.4) ?!

В качестве базового протокола для OF-CONFIG спецификацией 1.1.1 определен протокол NETCONF.

Протокол OF-CONFIG используется для решения следующих задач:

- назначение одного или более контроллеров OpenFlow коммутатору;
- конфигурирование портов и очередей;
- удаленное изменение свойств портов;
- конфигурирование сертификатов для безопасного взаимодействия логических коммутаторов OpenFlow и контроллеров OpenFlow;
- запрос возможностей логических коммутаторов OpenFlow;
- конфигурирование ограниченного набора туннелей (IP-in-GRE, NV-GRE, VxLAN);
- инициализация логических коммутаторов OpenFlow;
- назначение ресурсов OpenFlow-совместимого коммутатора одному и более логическому коммутатору OpenFlow;
- поддержка согласованных моделей участка передачи (Negotiable Datapath Model, NDM).

Предполагается, что следующие версии OF-CONFIG будут предусматривать также такие возможности, как обнаружение коммутаторов и

топологии, конфигурирование характеристик, обработка триггеров, связанных с событиями, инициализация сети OpenFlow, поддержка большего числа конфигурируемых туннелей.

Спецификацией протокола OF-CONFIG определена модель передаваемых протоколом данных, которая имеет существенное значение, так как стандартизированная модель данных сети связи является необходимым условием успешного практического применения SDN. Регламентированная модель описания сети обеспечивает согласованное представление о состоянии сети SDN, всех ее составляющих и возможность независимой разработки программных средств для работы с сетью SDN посредством протоколов консорциума ONF при сохранении совместимости.

Контрольные вопросы

1. В каком году и кем была предложена парадигма SDN?
2. Какой тип трафика в ближайшие годы будет преобладающим, как он повлияет на существующие сети?
3. Какова была цель создания концепции SDN?
4. Опишите общую архитектуру программируемой сети. Какие задачи решает каждый уровень?
5. Перечислите основные рабочие группы (РГ) консорциума ONF, задачи каждой из них и ожидаемые результаты работы.
6. Назовите цель и основные задачи РГ I2RS.
7. Перечислите ИК в МСЭ-Т, занимающиеся исследованиями в области SDN и их основные задачи.
8. Что предполагает концепция NFV
9. Перечислите протоколы взаимодействия различных уровней архитектуры SDN, выделяемые консорциумом ONF, опишите их основные функции.
10. Назовите типы сообщений протокола OpenFlow и примеры сообщений каждого типа.
11. Какие задачи возложены на протокол OF-CONFIG? Перечислите абстракции, предусмотренные протоколом OF-CONFIG.

Литература к главе 1

1. Recommendation ITU-R M.2083-0 “IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond”
2. Recommendation ITU-T Y.3300 “Framework of Software-defined networking”
3. Recommendation ITU-T Y.3011 “Framework of network virtualization for future network”

4. S. Dotcenko, A. Vladyko, and I. Letenko, “A fuzzy logic-based information security management for software-defined networks,” in Advanced Communication Technology (ICACT), 2014 16th International Conference on, Feb 2014

5. ONF. Sof tware-Def ined Networking (SDN) Def inition [Электронный ресурс] — Режим доступа: https://www.open-networking.org/sdn-resources/sdn-definition (дата обращения: 07.03.14)

6. IETF Draft Formal Specification: Framework for Software-Defined Networks (SDN). — 02/2013. [Электронный ресурс] — Режим доступа: https://tools.ietf.org/html/draft-shin-sdn-formal-specification-03 (дата обращения: 07.03.14)

7. Framework of Telecom SDN (Software-Defined Networking): ITU-T Draft Recommendation Y.FNsdn, 02/2013.

8. Volkov, A., Khakimov, A., Muthanna, A., Kirichek, R., Vladyko, A., Koucheryavy, A.: Interaction of the IoT traffic generated by a smart city segment with SDN core network. In: Koucheryavy, Y., Mamatas, L., Matta, I., Ometov, A., Papadimitriou, P. (eds.) WWIC 2017. LNCS, vol. 10372, pp. 115–126. Springer,Cham (2017). doi:10.1007/978-3-319-61382-6 10

2. ВИРТУАЛИЗАЦИЯ СЕТЕВЫХ ФУНКЦИЙ (NFV)

Согласно рекомендации Международного Союза Электросвязи, *виртуализация сети* (network virtualization function) – технология, которая позволяет создавать логически изолированные участки сети в рамках совместно используемых физических сетей таким образом, что в этой совместно используемой сети одновременно могут сосуществовать многие виртуальные сети (т.н. LINP's – Logically Isolated Network Partition, логически изолированный участок сети). При этом, говоря про виртуализацию сети стоит отметить такое понятие, как *виртуальный ресурс* (Virtual resource) – это абстракция физического или логического ресурса, которая может иметь характеристики, отличающиеся от характеристики этого физического или логического ресурса, и ее функциональные возможности могут быть не связаны с функциональными возможностями самого физического или логического ресурса.

Таким образом, виртуализация сетевых функций (NFV, Network Function Virtualization) – технология виртуализации физических сетевых элементов (физических ресурсов) телекоммуникационной сети путем исполнения сетевых функций программными модулями, работающими на стандартных серверах и виртуальных машинах (VM) в них. При этом, данные программные модули могут взаимодействовать между собой для предоставления услуг связи, чем ранее занимались аппаратные решения.

С развитием телекоммуникационных и информационных технологий, произошла их конвергенция, все больше и больше информационные технологии стали проникать в мир Телекома, тем самым на данный момент развития технологий, мы уже говорим об инфокоммуникационных технологиях. Подход виртуализации сетевых функций в мир телекоммуникаций пришел из мира информационных технологий, перевернув представление о том, как могут быть построены сети. При этом, в этот же момент времени назрела проблема масштабирования сетей, тенденция увеличения трафика и потребность абонентов в новых типах услуг (информационных) заставила телеком-операторов увеличивать парк дорогостоящего оборудования. Чтобы интегрировать в систему новое оборудование часто требуется много капитальных вложений (CAPEX), времени, а также требуется увеличивать штат высококвалифицированных инженеров для обслуживания возросшей в масштабах сети. Кроме же стоимости оборудования, оператору необходимо вкладываться в необходимые решения по обеспечению работы оборудования (серверные стойки, климатические установки, возможно новое помещение и так далее). Стоит также отметить, что при масштабировании сетевой инфраструктуры у оператора сети увеличиваются также операционные расходы (энергопотребление, кондиционирование, зарплаты новым сотрудникам и т.д.). Таким образом, эволюция сетей связи в сторону их виртуализации позволит

снизить темп роста показателей CAPEX и OPEX, при этом также в некоторых случаях меняется модели бизнеса оператора, например, телеком-оператор может стать также и сервис-оператором для небольших предприятий, предоставив им соответствующие сетевые функции по-запросу, как услугу (виртуальные VoIP станции, сетевые экраны и так далее), что несомненно принесет прибыль оператору.

Различные типы сетевого оборудования могут быть расположены на стандартных промышленных, больших вычислительных мощностей серверах, коммутаторов и систем хранения, которые могут быть расположены в Центре обработки данных (ЦОД), сетевых узлах и в помещениях конечных пользователей. «Виртуализация» включает в себя реализацию сетевых функций в программном обеспечении, которые могут работать в масштабе промышленного сервера и которые могут быть перемещены в различные места в сети по мере необходимости, без необходимости установки нового оборудования (АПК решения).

Основные задачи виртуализации.

Существует целый ряд проблем для реализации виртуализации сетевых функций, которые должны решается сообществом заинтересованных в ускорении процесса его внедрения. Данный ряд проблем можно интерпретировать как качества, которыми должен обладать новых подход. Рассмотрим некоторые из них.

Необходимые качества NFV:

- *Переносимость (Interoperability).*

Возможность загрузки и выполнения виртуальных устройств в различных стандартизованных средах центров обработки данных, предоставляемые различными поставщиками для различных операторов. Задача состоит в том, чтобы определить единый интерфейс. Переносимость и функциональная совместимость очень важна, поскольку поставщики разных программ используют свои уникальные интерфейсы. Переносимость также позволяет оператору свободно оптимизировать расположение и необходимые ресурсы виртуальных приборов.

- *Миграция.*

Виртуальные сетевые функции должны работать в гибридной сети, состоящей из классической физической сети и с виртуальными сетевыми элементами. Поэтому виртуальные устройства должны использовать существующие интерфейсы различных сервисов (для управления и контроля), а также взаимодействовать с физическими устройствами тех же сетевых функций (например, поддержка виртуальным маршрутизатором всех тех же функций, что и физическим, при этом данные маршрутизаторы могут быть подключены между собой и для обеспечения динамической маршрутизации использовать один из протоколов: RIP, OSPF, IS-IS или другие. При этом стоит отметить, что физический маршрутизатор «не знает», что подключенный к нему другой

маршрутизатор не является физической реализацией, а всего лишь является виртуальной абстракцией).

- *Автоматизация.*

Для обеспечения широкого распространения данного подхода (NFV) к организации сетевых инфраструктур, требуется обеспечить автоматизацию всех функций. Что является как раз одним из преимуществ данной технологии.

- *Безопасность и устойчивость*

Как и любая другая инфокоммуникационная структура, решение NFV должно обеспечивать необходимый уровень безопасности и устойчивости к внешним воздействиям, с целью обеспечения требуемого уровня качества. Операторы (телеком и сервис), которые постепенно будут внедрять на свои сети решения NFV должны быть уверены в безопасности и устойчивости интегрированных на сеть решений. Так как малейшая нештатная ситуация, при которой определенное количество клиентов могут не получить услугу, может нанести ущерб организации не только финансово, но более того, оператор может потерять клиентов, что в условиях современной высокой конкуренции считается наиболее большая потеря. Виртуальная сущность (виртуальная сетевая функция) должна также обеспечить требуемый уровень безопасности, как и ее АПК реализация. Особенно, в условиях данной структуры, такими качествами, как безопасность и устойчивость должен обладать гипервизор, при выходе из строя которого, оператор может потерять контроль над всеми сетевыми функциями, которые использовали данный гипервизор. В тоже время, для уверенности внедряемых решений на сеть, операторы связи проводят тестирование устройств по методикам, которые обычно разрабатываются специальным отделом оператора связи под собственную сеть (с учетом архитектурной особенности, профиля трафика и спектра предоставляемых услуг), не говоря уже про то, что внедряемые решения должны быть сертифицированы и отвечать установленным стандартам и регламентам качества.

- *Простота.*

Обеспечение простоты работы сетевых платформ виртуализации, по сравнению с теми, которые существуют на рынке в данный момент. То есть, например, для виртуализации сетевых функций возможно иметь платформу виртуализации, имеющую более простую реализацию, чем решения для виртуализации сервисов (IaaS, PaaS, SaaS). В некоторых случаях, компании-разработчики создают специализированное решение для NFV, например, компания VMware разработало решение виртуализации специально для NFV – VMware NSX, которое больше предназначено для сложно-управляемых сетей центров обработки данных (ЦОД).

- *Модульность и совместимость решений.*

Совместимость решений в NFV или иначе – прозрачная интеграция решений, подразумевает возможность развертывания любых виртуальных сетевых функций на высокопроизводительных серверах и гипервизорах, без

ограничения их функционала, стабильности и безопасности. Модульность в NFV подразумевает независимость решений от производителя серверов, гипервизоров и их архитектурных особенностей, в том числе должна быть обеспечена их совместимость без ограничения их функционала.

Иными словами, идея виртуализации заключается в том, чтобы на одном физическом устройстве находились несколько виртуальных устройств, реализующих соответствующие сетевые функции, с динамическим распределением ресурсов. Так, например, маршрутизатор, DPI, Firewall, BRAS и т.д. объединяются в одно физическое устройство и программно разделяют ресурсы вычислительной мощности между этими службами или функциями.

Преимущество NFV:

- Упрощение развертывания и обновления как софта, так и железа;
- Уменьшение стоимости за счет использования стандартных серверов;
- Объединение сервисов в группы;
- Быстрое масштабирование решений (программная и «физическая» составляющая);
- Возможность миграции виртуальных машин, реализующих те или иные сетевые функции по сети;
- Удобный мониторинг и контроль за сетью.

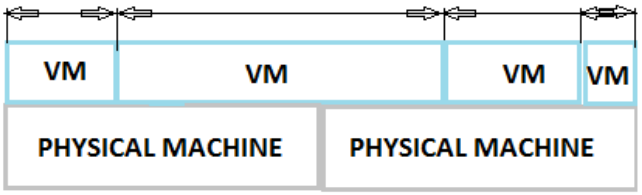


Рис. 2.1. Принципиальная структура NFV решения

2.1. Типовая структура NFV

Текущие сети связи состоят из большого количества разнообразных устройств (сетевых функций), которые обычно реализованы в виде АПК-решения. Структура NFV и ее описание отображается в спецификации ETSI GS NFV 002 от 2013 года. Виртуализация сетевых функций предусматривает реализацию сетевых функций в качестве только программного обеспечения (сущности), которые работают поверх инфраструктуры NFV (NFVI, Infrastructure). В спецификации ETSI GS NFV 002 выделяют три главные подсистемы:

- Виртуальная сетевая функция(-ии) (Virtualized Network Function), в качестве программной реализации сетевой функции, которая может работать поверх NFVI.
- NFV инфраструктура (NFVI, Infrastructure), включающая разнообразные физические устройства, а также необходимое программное обеспечение их объединяющее и реализующее функции виртуализации (гипервизор) для функционирования NF's.
- Управление и оркестрация NFV. На данную подсистему возложены функции оркестрации и управления жизненным циклом физических и/или программных ресурсов, которые в свою очередь реализуют NFVI, а также жизненным циклом VNFs.

Абстрактная структура NFV отображена на рисунке 2.2.

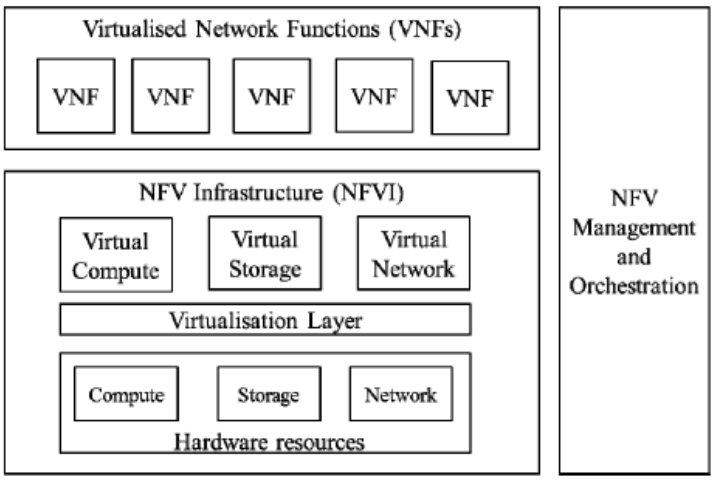


Рисунок 2.2 Абстрактарная структура NFV

Как уже отмечалось выше, NFV позволяет динамически управлять сетевыми функциями (VNF) и отношениями между ними, то есть по сути менять их конфигурацию и связи. Данный уровень автоматизации достигается путем унифицированности интерфейсов взаимодействия систем, что позволяет как раз использовать единую систему управления и мониторинга.

2.2. Функциональная архитектура NFV

Как и любая сложная система, состоящая из ряда подсистем, NFV имеет свою функциональную архитектуру, отображающую основные функциональные блоки (подсистемы) и их взаимосвязи через реперные точки. На рисунке 2.2 была приведена типовая структура NFV. Стоит отметить, что в некоторых ситуациях, для реализации NFV, требуется всего лишь добавить несколько функциональных модулей для обеспечения виртуализации и управления. Функциональная архитектура NFV, определенная в спецификации ETSI GS NFV 002 от 2013 года отображена на рисунке 2.3.

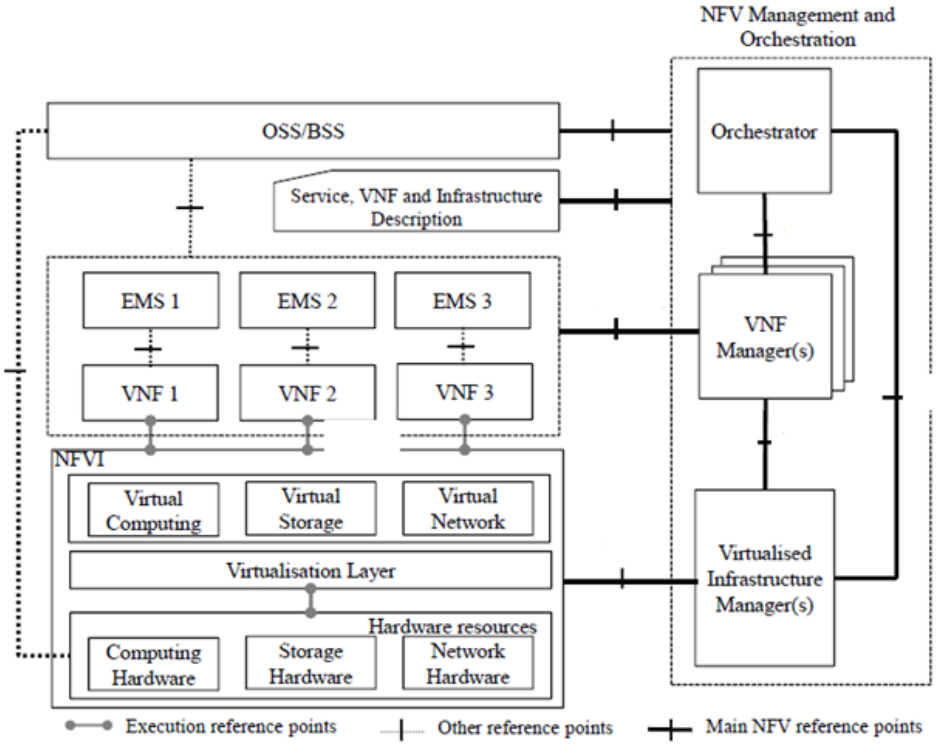


Рисунок 2.3 Функциональная архитектура NFV

В спецификации определены следующие функциональные модули:

- Virtualized Network Function (VNF) – Виртуальная сетевая функция;
- Element Management System (EMS) – Система управления элементами;

NFV Infrastructure (NFVI), включающая следующие элементы:

- Hardware resources – аппаратные ресурсы;
- Virtual resources – виртуальные ресурсы;
- Virtualization Layer – уровень виртуализации;
- Virtualized Infrastructure Manager (s) – подсистема управления инфраструктурой;
- Orchestrator – Оркестратор;
- VNF Manager(s) – Подсистема управления виртуальными сетевыми функциями;
- Service – VNF and Infrastructure Description, Услуги

Рассмотрим функциональные блоки NFV немного поподробнее.

Virtualized Network Function (VNF)

VNF, как уже отмечалось выше, является программной реализацией сетевой функции, которая обычно реализована в виде АПК-решения. Например, такие сетевые функции, как: Mobility Management Etitty (MME), Serving Gateway (SGW), Packet Data Network Gateway (PGW), Dynamic Host Configuration Protocol (DHCP-сервер), firewalls, DPI и другие, могут быть реализованы в качестве программной реализации, которые работают на сервере(-ах). Стоит также заметить, что виртуальная сетевая функция может быть развернута как централизовано на одной VM (Virtual Machine, виртуальной машине), так распределено на нескольких виртуальных машинах VM's.

Element Management System (EMS)

Работа данной подсистемы заключается в непосредственном функциональном управлении единицей или несколькими сетевыми функциями (VNFs).

NFV Infrastructure

Hardware Resources, HR (аппаратные ресурсы)

Аппаратные ресурсы в NFV архитектуре включают в себя комплекс подсистем, таких как: вычислительная часть, базы данных (обычно имеющие распределенную архитектуру), сетевая часть (кабельная инфраструктура, коммутаторы, сетевые экраны и т.д.). Аппаратные ресурсы по сути являются «уровнем обеспечения» или иначе, можно сказать «площадкой» на которой

взаимодействуют через прослойку гипервизора виртуальные сетевые функции. С точки зрения непосредственно самих виртуальных сетевых функций, аппаратная часть архитектуры NFV представляется как единая вычислительная платформа, имеющая способность масштабироваться. Под масштабированием понимается возможность изменения ее параметров (вычислительная способность, максимальная скорость передачи данных и др.). При этом масштабирование возможно за счет качественного или количественного изменения. При качественном изменении происходит замена составляющих элементов серверов (замена на более производительные процессоры или увеличение их количества в этом же сервере, соответственно добавление карт оперативной памяти). Такого типа масштабирование рано или поздно упрется в особенности архитектуры сервера (определенная архитектура процессора, их количество и соответствие определенных карт оперативной памяти и т.д.). При количественном масштабировании происходит непосредственное увеличение кластера серверов, возможно расширение сети передачи данных, которая обеспечивает связность серверов.

Стоит также отметить, в NFV различают два типа сетей передач данных:

- NFVI-PoP network. Данная сеть передачи данных обеспечивает взаимодействие вычислительной подсистемы с подсистемой хранения данных (базы данных), объединенные в рамках одной NFVI-PoP. Сюда также относят сетевые устройства (реализующие функции коммутации, маршрутизации), обеспечивающие взаимодействие с внешними сетями или другими NFVI-PoP.
- Transport Network. В контексте NFV – это сеть передачи данных, которая обеспечивает взаимодействие сетей NFVI-PoPs с другими сетями, принадлежащими различным операторам связи, а также с другими сетевыми устройствами или терминалами, не принадлежащие NFVI-PoPs.

Virtualization Layer and Virtualized Resources (Уровень виртуализации)

Уровень виртуализации представляет собой «прослойку» между физическими (вычислительными) ресурсами и программным обеспечением виртуальных сетевых функций. Таким образом обеспечивается независимый от оборудования на уровне вычислительных ресурсов жизненный цикл

виртуальных сетевых функций. Уровень виртуализации выполняет следующие ряд задач:

- Логическое разделение физических ресурсов и абстрагирование от них;
- Включение в работу программного обеспечения VNF и предоставление ей доступа к виртуальной инфраструктуре;
- Предоставление ресурсов для работы VNF;

С точки зрения практической реализации, уровень виртуализации представлен в качестве программного обеспечения – «гипервизор».

(VIM) Virtualization Infrastructure Manager(s) (подсистема управления инфраструктурой)

Подсистема VIM обеспечивает необходимые функциональные возможности для контроля и управления взаимодействием виртуальных сетевых функций с вычислительными, запоминающими и сетевыми ресурсами. Согласно отображенным в виде подсистем аппаратных ресурсов на функциональной архитектуре (рисунок 2.3, подсистема управления инфраструктурой выполняет следующие задачи:

- Учет развернутого программного обеспечения (например, используемых гипервизоров), вычислительных, запоминающих и сетевых ресурсов, в совокупности реализующих инфраструктуру NFV;
- Распределение ресурсов между виртуальными машинами;
- Управление ресурсами инфраструктуры и их распределение, например, при необходимости увеличить объем выделяемых ресурсов для определенной виртуальной машины, а также повышение энергоэффективности инфраструктуры, восстановление ресурсов;
- Сбор информации и анализ произошедших сбоев в работе той или иной подсистемы инфраструктуры;
- Сбор информации для мониторинга, планирования масштабирования ресурсов инфраструктуры.

Orchestrator (Оркестратор)

Оркестратор является подсистемой управления и ответственен за организацию и управление инфраструктурой NFV, программными ресурсами. При этом, одной из главных его функций является – организация и управление услугами на NFVI.

VNF Manager(s) (подсистема управления виртуальными сетевыми функциями)

Данная подсистема реализует необходимый и требуемый функционал по управлению жизненным циклом виртуальной сетевой функции, а именно: создание экземпляра VNF, обновление, масштабирование, завершение работы, удаление экземпляра VNF. Стоит отметить, что рассматривают также применение нескольких подсистем управления сетевыми функциями, например, возможны следующие случаи: одна подсистема на каждую VNF или одна подсистема на ряд VNF.

OSS / BSS (Системы поддержки операций и системы поддержки бизнеса оператора)

С точки зрения функциональной архитектуры, отображенной на рисунке 2.3, данные подсистемы являются сторонними системами поддержки операционной деятельности и поддержки бизнеса оператора, напрямую не относящиеся к NFV.

2.3. vCPE, как пример использования NFV на практике.

Концепция NFV давно не ноу-хау в телекоммуникациях и уже нашла свое применение операторами услуг связи в качестве vCPE. Однако, для более детального рассмотрения vCPE, как примера внедрения концепции NFV на практике, разберемся в том, почему данный вид услуги нашел своего потребителя.

Развертывание сетевых функций предприятиями в своей корпоративной сети требует больших денежных вложений, поскольку для организации какой-нибудь сетевой или вычислительной функции потребуется специальное оборудование. А именно, с точки зрения экономической составляющей предприятия – увеличение таких параметров, как: CAPEX (капитальные затраты) и OPEX (операционные затраты), особенно если мы говорим о проектировании и постройке собственного ЦОД с нуля.

Эти затраты можно уменьшить, если требуемые функции осуществлялись бы на стороне оператора связи. На этом основана одна из моделей применения NFV на практике, заключающаяся в предоставлении виртуальной сетевой функции в качестве услуги коммерческому клиенту

(VNFaaS – Virtual Network Function as a Service). Для предоставления виртуальной сетевой функции оператор использует вычислительное облако (один или кластер серверов). Можно считать, что VNFaaS сопоставимо по функциональности с моделью облачных структур SaaS или программное обеспечение как услуга. И при VNFaaS, и при SaaS потребитель услуг не управляет базовой конфигурацией.

При предоставлении VNF в качестве услуги поставщиком является сервис-провайдер, предоставляющий то или иное приложение в качестве виртуальной функции, а потребителем – компания-заказчик услуги, по сути корпоративный клиент. Потребитель не контролирует работу инфраструктуры NVF (NVFI), или саму виртуальную сетевую функцию. Также от потребителя услуг не требуется дополнительных затрат на развитие VNF. Необходимо заметить, что оператор может увеличить инфраструктурные ресурсы NVFI, при увеличении у заказчика услуги потребности в использовании других VNF.

В общем случае сеть для поставщика услуги NFV состоит из двух устройств (рисунок 2.4): граничного маршрутизатора оператора связи (PE) и конечного оборудования пользователя (CPE).

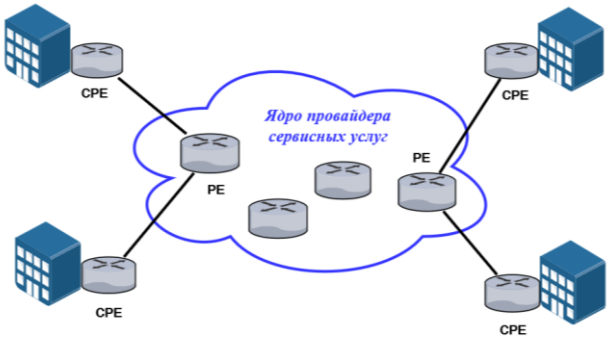


Рисунок 2.4 Устройства сети провайдера услуг NFV

На основе отображенной на рисунке 2.4 архитектуры сети, поставщики услуг NFV могут организовать следующие модели:

- vCPE или vE-CPE, устройство CPE также так же является виртуальной функцией и реализована в облаке провайдера;
- vPE, граничный маршрутизатор оператора связи также может быть виртуализован.

Стоит также отметить, что устройства vCPE и vPE могут управляться центральным контроллером, если сеть построена на основе концепции SDN, с использованием протокола OpenFlow. При этом, маршрутизаторы vCPE и vPE ориентированы на предоставление услуг различному количеству пользователей, то есть маршрутизатор vPE могут использовать несколько

заказчиков, в то время как маршрутизатор vCPE может использовать только один заказчик.

Блок vCPE располагается либо в центре обработке данных оператора, либо в пользовательской сети на границе сети оператора. Первый способ характеризует подход к реализации vCPE, называемый «Cloud-модель», а второй – «Edge-модель». В «Cloud-модели» предоставление ресурсов виртуальных функций, которые включены в vCPE, осуществляется на стороне сервис-провайдера (рисунок 2.5).

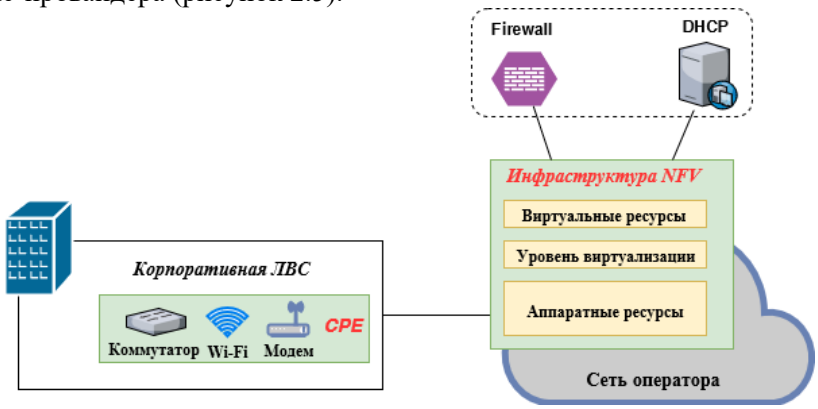


Рисунок 2.5 Cloud-модель реализации vCPE

В подходе «Edge-модель» функции vCPE расположены в сети пользователя на недорогом оборудовании, представляющем собой инфраструктуру локальной NFV. Принципиальная архитектура Edge-модели vCPE отображена на рисунке 2.6.

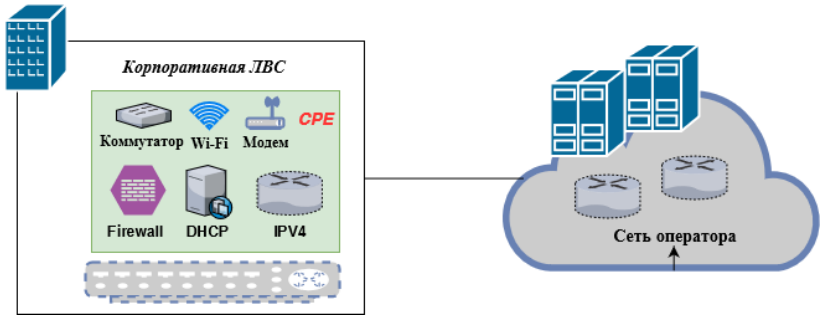


Рисунок 2.6. Edge-модель реализации vCPE

Стоит отметить, что на практике также существует смешанный подход, в котором одни функции vCPE размещаются на стороне оператора, а другие – в сети пользователя.

Подходы к виртуализации сетевых функций.

Можно выделить два способа организации виртуальной функции:

- Многофункциональная vCPE, развертываемая на одном виртуальном устройстве. Данный подход может называться «вертикально-масштабируемым».
- Одно функциональные VNF, помещаемые в контейнера. Данный подход называется «контейнерный»

Вертикально-масштабируемом подход.

В вертикально-масштабируемом подходе сетевая функция представлена в неделимом виде и сохраняет требования к архитектуре программного обеспечения и возможностей операционной системы, которые были на исходном аппаратном оборудовании. В данной ситуации, виртуализация сетевых функций представлена в неделимом виде и представляет собой некоторый «образ диска», помещённый в автономный программный экземпляр (рисунок 2.7.). Из-за того, что такой образ сохраняет требования исходного аппаратного устройства, копирование «образов дисков» затруднительно.

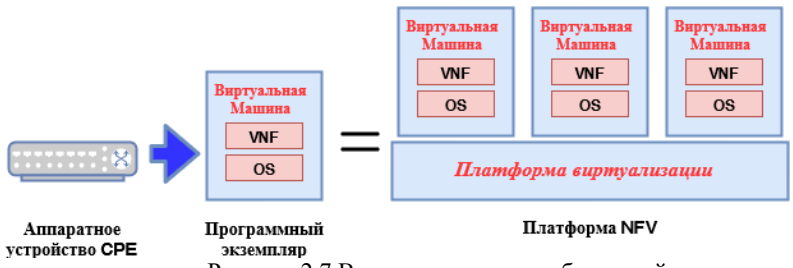


Рисунок 2.7 Вертикально-масштабируемый подход.

Данный подход достаточно неэффективен, поскольку обладал рядом отрицательных черт:

1. «Вертикально масштабируемый» подход экономически выгоден только для крупных предприятий, которые могли развернуть большое количество виртуальных машин на меньшем количестве серверов.
2. Затраты на развертывание VNF составляли 80-90 процентов по сравнению с аппаратной реализацией. Если же поставщики аппаратных устройств предоставляли скидки на свое оборудование, то физические аналоги виртуальных функций обладали меньшей стоимостью.
3. Операторы услуг снимали плату за полнофункциональную VNF, даже если заказчику не нужно было использовать весь набор функций.

Контейнерный подход.

Контейнерная технология стала следующим шагом в развитии виртуализации, после «вертикально-масштабируемого» подхода, позволяющем

быстро и недорого как создавать необходимые для потребителя функции, так и быстро развертывать их на стороне заказчика, или поставщика виртуальных услуг.

При контейнерной технологии ядро ОС помещает приложения в отдельные контейнеры. В результате, виртуальная машина может быть разделена на несколько слоёв – «контейнеров».

Современная платформа vCPE.

В настоящее время платформы vCPE отличаются:

1. Построением виртуальных функций на основе технологии контейнеров
2. Применением микро-сервисов.
3. Развитием концепции «нано-сервисов».

Концепция «микро-сервиса» основана на отдельной VNF в виде программного компонента, выполняющего одну функцию. При этом, программные компоненты могут объединяться для создания одной более сложно организованной сетевой функции. Особое значение в данной концепции отводится виртуальным функциям с открытым кодом (OpenSource VNF), обладающим возможностями быстрого их создания, а также быстрого разделения на отдельные функции. Применение виртуальных функций с открытым кодом позволяет повысить степень контроля и гибкость предоставляемых клиентам услуг сети. Операторы могут фильтровать не пользующиеся спросом VNF. Контейнерная технология позволяет за доли секунд разворачивать новые однофункциональные VNF. Поэтому концепция «микро-сервисов» тесно связана с виртуализацией в виде контейнеров.

В концепции «нано-сервиса» несколько контейнеров объединяется в законченную функцию сети, а сами нано-сервисы представляют собой услугу, состоящую из цепочки VNF’s, каждая из которых выполняет только одну функцию. Контейнеры могут располагаться и на стороне оператора, и на стороне заказчика (рисунок 2.8). Отдельные виртуальные функции при необходимости во время работы могут свободно устанавливаться, изменяться и удаляться из нано-сервиса.

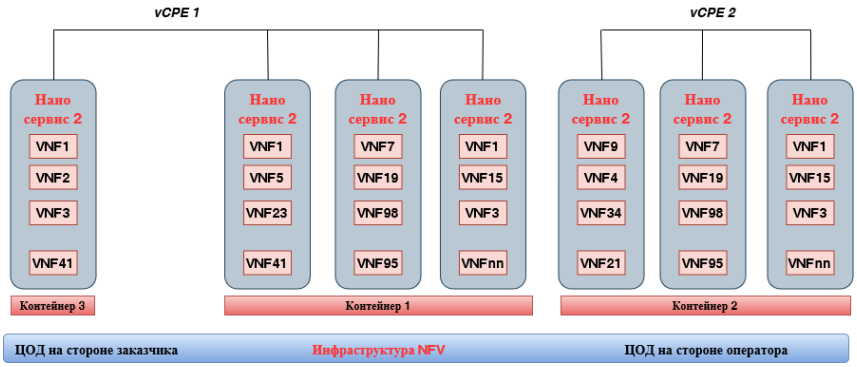


Рис. 5 Применение концепции нано сервиса и контейнерной технологии

Таким образом, применение подхода с использованием нано-сервисов одно-функциональных VNF позволяет:

1. Организовывать масштабируемые экземпляры только в случае их необходимости, что снижает первичные затраты и риски операторов. Кроме того, потребитель услуг виртуализации будет платить только за те функции, в которых он реально будет нуждаться.
2. Быстро создавать новые виртуальные функции (сложные) при их необходимости
3. Располагать VNF как в сети оператора услуг, так и в сети компании-заказчика.

Контейнерная технология позволяет также производить живую миграцию контейнеров между частями (локациями) vCPE. Под живой миграцией подразумевается процесс перемещения приложения (контейнера) между различными машинами или облаками без прерывания работы приложения и разрыва установленного соединения с пользователем.

Стоит сказать, что все вышеописанные технологии и подходы давно уже имеют апробацию на практике. При этом первопроходцами в применении данных технологий на «боевых» решениях являются ЦОДы и частные корпоративные сети. Однако на данный момент, крупные операторы связи уже имеют опыт внедрения данных технологий и замечают достигнутый положительный эффект, в том числе и в Российской Федерации.

Ключевые слова

Сетевая функция, автоматизация, виртуализация, переносимость, устойчивость, стабильная сеть, преимущество, управление.

Контрольные вопросы

1. В чем заключается идея виртуализации сетевых функций (NFV)?
2. Назовите необходимые качества NVF и соответствующие задачи для их обеспечения, которые требуется решить.

3. В чем заключается преимущества NVF?

4. Назовите основные функциональные блоки NFV, согласно стандарту ETSI?

5. Что такое vCPE?

6. В чем заключается контейнерная технология в vCPE?

7. В чем заключается концепция «нано-сервисов»?

8. В чем заключается «живая миграция» и как она может быть применена в NFV реализации?

Литература к главе 2

1. Network Functions Virtualization (NFV); Terminology for Main Concepts in NFV (ETSI GS NFV 003 V1.2.1 (2014-12))

2. Network Functions Virtualization (NFV); Architectural Framework (ETSI GS NFV 002 V1.2.1 (2014- 12))

3. Network Functions Virtualization (NFV); Infrastructure Overview (ETSI GS NFV-INF 001 V1.1.1 (2015-01))

4. Network Functions Virtualization (NFV); Infrastructure; Compute Domain (ETSI GS NFV-INF 003 V1.1.1 (2014-12))

5. Network Functions Virtualization (NFV); Infrastructure; Hypervisor Domain (ETSI GS NFV-INF 004 V1.1.1 (2015-01))

3. ТИПЫ КОНТРОЛЛЕРОВ

Контроллер программируемой сети может быть представлен как АПК-решение (аппаратно-программное решение), так и в качестве программной реализации. На данный момент, контроллеры на рынке чаще представлены в программной реализации. Контроллер, который представлен в качестве ПО, устанавливается на выделенный физический (или виртуальный) сервер. Контроллер логически состоит из сетевой операционной системы и сетевых приложений. Архитектура контроллера представлена на рис. 3.1.

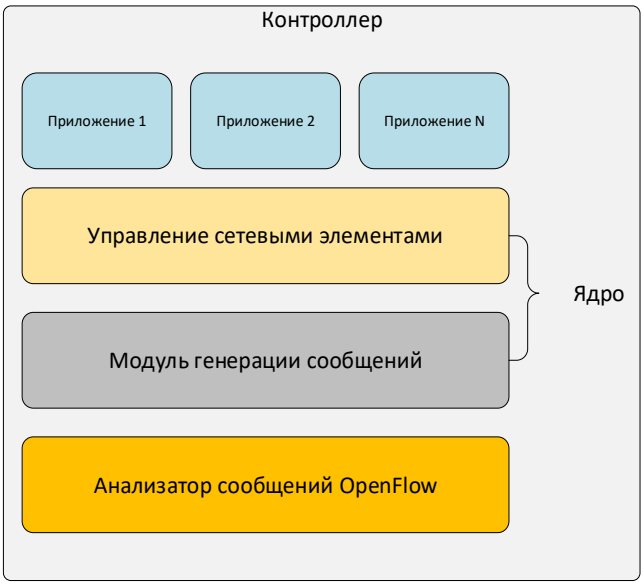


Рисунок 3.1. Архитектура контроллера программируемой сети

Сетевая ОС является ядром контроллера, она взаимодействует с элементами сети (коммутаторами), осуществляет контроль и формирует состояние сети на основе сообщений от элементов сети, предоставляет возможности взаимодействия приложений между собой, распространяет управляющие воздействия на элементы сети, то есть осуществляет управление сетевыми ресурсами сети. Также в функции сетевой ОС входит установление и поддержание соединений с коммутаторами, распределение обработки сообщений между потоками исполнения (для многопоточных контроллеров), работа с очередью полученных сообщений и очередью сообщений на отправку. Также ядро преобразует полученные сообщения во внутреннее представление, осуществляет сортировку сообщений, отправляемых приложениями, для передачи их по сети и предоставляет приложениям интерфейс для получения и отправки OpenFlow сообщений.

Сетевые приложения и сервисы, написанные администратором сети, на основе информации о состоянии сети осуществляют непосредственное

управление трафиком (на основе сервисных политик, текущей загрузки сети и т.п.). Таким образом, на контроллере должно быть установлено хотя бы одно приложение. В настоящее время таким приложением является learningSwitch, написанное для всех контроллеров как базовое. Приложение контроллера представляет функционал, необходимый для решения задач пользователя. Приложения могут реализоваться как программный модуль контроллера, так и как самостоятельный процесс, взаимодействующий с контроллером через некоторый интерфейс (например, APIrest). Реализация приложения в виде программного модуля контроллера обеспечивает высокую скорость обмена сообщениями между ядром контроллера и приложением и между различными приложениями. Также использование стандартного программного интерфейса для взаимодействия между контроллером и приложением позволяет создавать приложения на любом языке программирования. Некоторые приложения могут входить в состав контроллера, набор таких приложений может сильно различаться для различных контроллеров.

Сервисы контроллера позволяют вынести функциональность, часто используемую различными приложениями, в отдельные модули. Данные сервисы не являются обязательными, однако упрощают создание новых приложений для контроллера. Примеры таких сервисов: загрузка выбранных приложений, обеспечение взаимодействия между ними, отслеживание топологии сети. Наличие некоторых сервисов обязательно для оптимального функционирования приложений (например, загрузка приложений и интерфейс для взаимодействия между ними). Другие сервисы могут предоставлять необходимую информацию для некоторых приложений (например, об изменениях в топологии сети), однако они не требуются для работы всех приложений. В случае, если сервис представляет собой отдельный модуль, то в зависимости от архитектуры контроллера этот модуль может как запускаться сразу вместе с контроллером, так и динамически загружаться во время работы контроллера.

Сервисы контроллера.

Сервисы контроллера разделяют на четыре группы.

1. Сервисы, обеспечивающие взаимодействие с коммутаторами и сбор статистики по коммутаторам

Данная группа включает в себя следующие сервисы:

– OpenFlow Protocol. Модуль, обеспечивающий взаимодействие с коммутаторами по протоколу OpenFlow заданной версии. Преобразует сообщения OpenFlow во внутреннее представление на используемом языке программирования (обычно для этого используются библиотеки OpenFlow для выбранного языка), сериализует сообщения OpenFlow, сгенерированные приложениями.

– Network Protocols. Библиотеки для обработки и формирования заголовков пакетов различных уровней сетевого стека (используется,

например, для обработки сообщений PacketIn и формирования сообщений PacketOut).

– Monitoring. Модуль периодически запрашивает у коммутаторов статистику, предусмотренную протоколом OpenFlow: текущие значения счетчиков, связанных с портами и очередями портов (количество полученных и отправленных пакетов, количество сброшенных пакетов), таблицами правил (количество правил в таблице, количество сопоставлений заголовков пакетов с правилами), отдельными правилами (в том числе: количество полученных пакетов, время жизни).

– Echo Requiests. Модуль периодически рассылает коммутаторам сообщения echo. Данный сервис нужен в случае, если идет взаимодействие с коммутаторами, которые закрывают соединение с контроллером, если в течение некоторого времени от контроллера не поступало сообщений. Также информация о времени ответа на запрос echo дает возможность оценить, сколько времени понадобится контроллеру для того, чтобы определить, что было потеряно соединение с коммутатором: на основании этой информации контроллер может установить время ожидания ответа коммутатора на запрос echo, по истечении которого можно считать, что соединение с коммутатором потеряно.

2. Сервисы для загрузки модулей и обеспечения взаимодействия между ними

Для загрузки модулей и обеспечения взаимодействия между ними предназначены следующие сервисы:

– Module Manager. Модуль контроллера, осуществляющий загрузку всех остальных модулей и определяющий, какие сервисы предоставляет то или иное приложение и на какие сервисы других приложений оно подписано. Для загрузки модулей могут использоваться существующие архитектурные каркасы (framework), например, контроллер Beacon использует OSGI и Spring.

– Messenger. Модуль, обеспечивающий обмен сообщениями между приложениями контроллера или уведомление приложений о тех или иных событиях. Могут использоваться библиотеки, например, libevent для C.

– Web server. Веб-сервер, позволяющий приложениям контроллера, реализованным в виде внешних приложений, обмениваться сообщениями и предоставлять доступ к своим сервисам через стандартный API, например, RESTful.

– Packet Streamer. Модуль предоставляет приложениям, работающим через интерфейс RESTful, возможность подписаться на получение определенных типов сообщений OpenFlow.

– Memory Storage. Модуль предоставляет приложениям разделяемую базу данных с возможностью подписки на уведомления об изменении состояния базы.

– Flow Cache. Модуль собирает информацию о всех правилах, действующих на данный момент в сети, которые были установлены разными приложениями. Сервисы Module Manager и Messenger являются обязательными для обеспечения взаимодействия нескольких модулей, запущенных на одном контроллере, например, для осуществления подписки приложений на пришедшие сообщения OpenFlow.

3. Сервисы для работы с топологией сети Topology/Discovery.

Данный модуль предназначен для сбора, хранения и обновления информации о топологии сети. Сбор сведений о топологии сети осуществляется при помощи рассылки пакетов протокола LLDP. Контролер отправляет на каждый коммутатор PacketOut сообщение, содержащее LLDP пакет с требованием разослать его по всем портам коммутатора. Когда соседний коммутатор получает этот LLDP пакет, он отправляет контроллеру сообщение PacketIn с полученным пакетом, на другие порты коммутатора данный пакет не пересылается. Таким образом, контроллер может узнать о существовании соединения между двумя коммутаторами. В контроллере Floodlight для того, чтобы обнаружить непрямые соединения между OpenFlow-коммутаторами, т.е. такие OpenFlow-коммутаторы, которые соединены через обычные коммутаторы, используются пакеты протокола BDDP, который является расширением протокола LLDP. BDDP отличается от LLDP тем, что в качестве MAC-адреса получателя в нем указан широковещательный адрес, т.е. коммутатор при получении данного пакета разошлет его на все порты. Также данный модуль оповещает подписанные приложения об отключении соединений на портах коммутатора или о возникновении новых соединений.

– Device Manager. Отслеживает точки подключения конечных узлов в сети. Обнаружение нового узла происходит при получении сообщения PacketIn, каждый конечный узел идентифицируется своими MAC-адресом и идентификатором VLAN. Модуль проверяет, было ли это устройство подключено ранее, и удаляет старую информацию о его точке подключения. Информация о точке подключения устройства удаляется по истечении установленного времени, в течение которого не было получено пакетов от данного устройства.

– Forwarding. Модуль вычисляет маршруты между конечными узлами сети. Маршрут вычисляется на основе информации о топологии сети при помощи алгоритмов поиска пути в графе. Далее при помощи FlowMod сообщений на всех коммутаторах, через которые проходит маршрут, контроллер устанавливает соответствующие правила.

– Spanning Tree. Модуль получает от соответствующего сервиса информацию о топологии сети и строит на ее основе остовное дерево. На всех коммутаторах запрещается широковещательная рассылка пакетов (flooding) на порты, не входящие в остовное дерево. Таким образом, в сети ликвидируются существующие петли и предотвращаются широковещательные шторма (зацикливание широковещательных пакетов в сети). Данный модуль может

использоваться для реализации функциональности протокола STP на коммутаторах, на которых данный протокол не поддерживается.

4. Прочие служебные сервисы

Данные сервисы, хотя не являются обязательными для нормальной работы контроллера и приложений, но упрощают конфигурирование контроллера и отладку приложений.

– CLI. Интерфейс командной строки для управления контроллером. Он предоставляет возможность загрузки дополнительных модулей с заданными настройками во время работы контроллера. Также модули могут поддерживать свои команды для управления через интерфейс командной строки (например, добавление новых ACL-правил в приложении Firewall или задание настроек DHCP-сервера в приложении DHCP).

– Web GUI. Веб-интерфейс для управления контроллером.

– Log. Различные сервисы для записи логов: дамп инкапсулированных в PacketIn Ethernet кадров, вывод отладочных сообщений контроллера и т. п.

– PCAP. Модуль создает трассу в формате pcap из полученных OpenFlow сообщений для экспорта, например, в Wireshark.

– Network Emulation. Позволяет разработчикам приложений и администраторам моделировать сеть, в которой работает контроллер, для тестирования сетевых приложений. Функциональность, близкая к ПО mininet.

Приложения контроллера

С точки зрения функциональности, приложения контроллеров можно разделить на 4 группы:

1. Приложения, реализующие простейшие правила коммутации и маршрутизации пакетов в сети

К данной функциональной группе относятся следующие приложения:

– Hub. Приложение настраивает коммутаторы таким образом, чтобы они работали как сетевые концентраторы: устанавливает на коммутаторах правило, согласно которому любой пришедший пакет рассылается на все порты, кроме входного.

– L2 Learning Switch. Приложение осуществляет L2 коммутацию, т.е. правила на коммутаторах устанавливаются на основе MAC адресов. Для каждого коммутатора приложение создает таблицу, в которую заносится соответствие между MAC-адресом конечного узла и портом коммутатора, на который нужно передавать кадры, адресованные этому узлу. Таблица заполняется на основе адреса отправителя кадра, который был получен на некотором порту коммутатора. Когда приложение получает от коммутатора сообщение PacketIn с кадром, для которого нет правила на коммутаторе, оно находит в таблице MAC-адресов адрес получателя и соответствующий ему порт и устанавливает на коммутаторе правило, согласно которому надо отправлять все кадры с таким MAC-адресом на указанный порт.

Возможные модификации данного приложения:

- установка новых правил с заданием только соответствующих MAC адресов, независимо для каждого коммутатора (традиционный L2 learning, описанный выше).
- использование сопоставления по максимальному числу полей заголовка (например, для различных TCP соединений между двумя узлами будут установлены отдельные правила, несмотря на то, что MAC адреса в заголовках одинаковые).
- использование модуля построения топологии для получения информации о соединениях: как только один коммутатор обнаруживает устройство с новым MAC адресом, правило коммутации для данного потока добавляется не только на этот коммутатор, но и на другие коммутаторы, используя информацию о существующих соединениях.

– L3 Learning Switch. Модификация Learning Switch, осуществляющая L3 коммутацию (т. е. коммутацию на основе таблицы IP адресов). Однако под управлением данного приложения коммутаторы не работают как сетевые маршрутизаторы. Например, не учитывается принадлежность конечных узлов к разным подсетям, приложение просто определяет местоположение IP адреса в сети. Если узлы принадлежат к разным подсетям, взаимодействие между ними должно осуществляться через определенный шлюз. В этом случае приложение позволяет сымитировать использование шлюза с заданным IP адресом.

– Static Flow Pusher. Коммутация каналов. Позволяет вручную установить статический маршрут между двумя заданными узлами. Все существующие на сегодняшний день контроллеры поставляются с приложением L2 Learning Switch. Также большинство контроллеров содержит приложение Hub.

2. Приложения для работы с распространенными сетевыми протоколами.

К данной функциональной группе относятся следующие приложения:

– ARP. Приложение, которое отслеживает передаваемые в сети ARP-сообщения и заполняет на контроллере ARP таблицу (соответствие IP адреса MAC адресу), а также отвечает на ARP-запросы конечных узлов. Имеет интерфейс командной строки для внесения записей в ARP-таблицу.

– DNS. Приложение отслеживает в сети ответы DNS сервера и сохраняет их в таблицу соответствий доменного имени и IP адреса.

– DHCP. Приложение выполняет роль DHCP сервера в сети. Приложение отслеживает запросы DHCP клиентов в сети, сообщает им заданный IP в качестве адреса DHCP и выдает им IP адреса из указанного диапазона. Также может сообщать приложениям заданные IP адреса, которые должны использоваться в качестве адреса шлюза и адреса DNS сервера.

Большая часть существующих контроллеров (кроме NOX и POX) не поставляется с подобными приложениями.

3. Приложения для анализа и перераспределения трафика в сети

К данной функциональной группе относятся следующие приложения:

– Firewall. Приложение позволяет задавать ACL-правила для фильтрации трафика в сети. Далее приложение преобразует эти правила в правила OpenFlow и устанавливает их на коммутаторах.

– Load Balancer. Балансировка трафика для ICMP, TCP и UDP. Позволяет задать пул серверов, между которыми будет распределяться обработка запросов от клиентов с заданными IP адресами. Распределение запросов происходит не на основании интенсивности трафика, а только на основании активных в данный момент соединений с серверами. Интерфейс, близкий к OpenStack Quantum LBaaS.

4. Приложения для поддержки виртуализации и сетей ЦОД

К данной функциональной группе относятся следующие приложения:

– Virtualization. Приложение для создания виртуальных сетей на основе ПКС. В простейшем случае — виртуализация на канальном уровне (MAC), т. е. приложение предоставляет возможность добавить узел с заданным MAC адресом в некоторую виртуальную сеть, таким образом, возможна передача сообщений только между узлами одной подсети. Возможна поддержка VLAN и GRE, т. е. разделение на виртуальные сети на основе значений полей VLAN ID или Tunnel ID (идентификатор логической сети для протоколов туннелирования, например, GRE).

– Data Center Backend. Набор надстроек (англ. plug-ins) для систем управления сетями ЦОД (например, для OpenStack Quantum). Данный набор необходим для поддержки интерфейса управления сетью системы управления ЦОД. Таким образом, контроллер программируемой сети может применяться в качестве основного инструмента управления сетями ЦОД, предоставляя широкие возможности для автоматизации настройки сетевого оборудования в зависимости от изменения конфигурации сети (например, в случае миграции виртуальных машин между серверами).

Основные характеристики контроллера

Основными характеристиками контроллера программируемой сети выделяют следующие:

1. Производительность – число потоков, обрабатываемых контроллером в единицу времени [потоки/с];
2. Время обработки – количество времени, затрачиваемое контроллером на обработку запроса от коммутатора [с];
3. Надежность – число отказов при заданном профиле нагрузки;
4. Ресурсоемкость – утилизация контроллером оперативной памяти физического сервера, и нагрузка на ядра процессора;

5. Масштабируемость – поддержка контроллером многопоточности.

3.1 Сетевая ОС OpenDaylight

Контроллер OpenDaylight является открытым программным обеспечением для программируемых сетей, который использует открытые протоколы для обеспечения централизованного программного управления сетью и мониторинга сетевых устройств. OpenDaylight (ODL) реализован на языке Java и работает в собственной виртуальной машине (JVM, Java Virtual Machine), таким образом данный контроллер может быть развернут на любой операционной системе, поддерживающей Java. Контроллер поддерживает и обеспечивает совместимость с широким спектром оборудования от разных производителей. Открытое сообщество разработчиков активно развивает и обновляет платформу ODL каждые шесть месяцев и адаптирует ее для широкого применения в отрасли SDN и NFV.

Как и многие другие контроллеры SDN, OpenDaylight поддерживает OpenFlow, а также предлагает собственные сетевые решения для SDN, в рамках своей платформы.

OpenDaylight позволяет легко устанавливать дополнительные функциональные возможности (модули) для конкретных требований сети, с помощью Karaf. Karaf это – OSGI контейнер.

OpenDaylight выполняет следующие функции:

– Логически централизует программное управление физическими и виртуальными устройствами в сети.

– Управление устройствами с помощью стандартных, открытых протоколов.

– Обеспечивает абстракции более высокого уровня своих возможностей, что позволяет сетевым инженерам и разработчикам создавать новые приложения для настройки, установки и администрирования сети.

Основные модули OpenDaylight

– Model-Driven Service Abstraction Layer (MD-SAL) – MD-SAL позволяет разработчикам создавать новые возможности Karaf в виде сервисов/услуг и драйверов протоколов и компоновать их друг с другом.

MD-SAL состоит из двух компонентов:

1. Общее хранилище данных, состоящее из:

- Config Datastore – поддерживает требуемое состояния сети.
- Operational Datastore – представляет фактическое состояние сети на основе данных, получаемых из управляемых сетевых элементов.

2. Шина сообщения обеспечивает возможность различным службам и драйверам протоколов уведомлять и взаимодействовать друг с другом.

- Application-Layer Traffic Optimization (ALTO) прикладной протокол оптимизации трафика, предоставляет приложениям информацию о сети. Он предоставляет возможность абстрактного представления сети и сетевых служб,

что позволяет сервисам и приложениям прикладного уровня гибко управлять сетевыми ресурсами. ALTO включает в себя пять сервисов Network Map Service; Cost Map Service; Filtered Map Service; Endpoint Property Service и Endpoint Cost Service.

- Border Gateway Protocol (BGP) - который обеспечивает поддержку для протокола BGP (в том числе состояния канала распространения).
- Border Gateway Monitoring Protocol (BMP) - плагин, обеспечивающий поддержку протокола мониторинга BGP, служит для мониторинга сети.
- CAPWAP - Control and Provisioning of Wireless Access Points (CAPWAP) протокол управления и инициализации беспроводных точек доступа позволяет OpenDaylight управлять CAPWAP-совместимыми беспроводными точками доступа.
- Controller Shield - создает репозиторий, который называется «Единый-плагин безопасности» (USecPlugin) необходимый для обеспечения информационной безопасности контроллера. Данный плагин может быть использован для настройки брандмауэров и создания черных списков. Работает на стороне приложений «северного моста» (northbound), таких как:

1. Выявление источников атак, зарегистрированных в направлении южного интерфейса (southbound)
2. Сбор информации о возможных угрозах и проверка доверенных устройств сети.

- Device Identification and Driver Management (DIDM) – компонент идентификации устройств и управления драйверами.
- DLUX – пользовательский веб-интерфейс OpenDaylight, включает в себя:

1. Визуализатор сетевой топологии;
2. Функцию слежения за MD-SAL потоками;
3. Набор инструментов и модель YANG.

- Fabric as a Service (FaaS) - создает единый уровень абстракции поверх физической сети, таким образом сервисы и приложения могут легко взаимодействовать с абстрактными объектами устройств физической сети для их конфигурации.
- Group Based Policy (GBP) - определяет модель политики приложений, ориентированных на OpenDaylight, отделяющую информацию о требованиях к подключению приложений от информации о базовых сведениях сетевой инфраструктуры. Обеспечивает поддержку:

1. OpenStack Neutron;
2. Service Function Chaining;
3. OFOverlay, NAT.

- Internet of Things Data Management (IoTDM) – управление данными Интернета Вещей (IoT, Internet of Things). Программное обеспечение для

работы с данными, поступающими от IoT-устройств. Как выше было отмечено, при разработке OpenDaylight использовалась технология Java Karaf, которая позволяет конфигурировать OPenDaylight для различных задач, расширяя его функционал, с помощью установки соответствующих модулей. В данном случае, при установке модуля IoTDM возможно реализовать сервер для управления данными Интернета Вещей, который в свою очередь разработан на основе международных спецификаций oneM2M.

- Link Aggregation Control Protocol (LACP) – может автоматически находить и агрегировать множественные связи между OpenDaylight контролируемой сетью и LACP с поддержкой конечных точек или коммутаторов.
- Location Identifier Separation Protocol (LISP) – данный плагин обеспечивает взаимодействие устройств сети по протоколу LISP.
- NEMO – предметно-ориентированный язык программирования, служит для создания абстракций сетевых моделей и выявления закономерностей функционирования. NEMO позволяет пользователям сети/приложений, описать свои требования к сетевым ресурсам, услуги и логические операции интуитивно понятным способом.
- NETCONF – плагин обеспечивающий поддержку протокола NETCONF.
- NetIDE – обеспечивает портативность и взаимодействие приложений внутри одной сети, посредством клиент/серверной архитектуры, с несколькими контроллерами. Плагин создает плоскость, позволяющую приложениям, написанным для других контроллеров, запускаться на OpenDaylight.
- OVSDB – based Network Virtualization Services – плагин обеспечивает упрощенную интеграцию ODL с фреймворком OpenStack Neutron.
- OpenFlow Configuration Protocol (OF-CONFIG) – реализация протокола OF-CONFIG, позволяет дистанционно конфигурировать и устанавливать наборы инструкций на коммутаторы OF.
- OpenFlow plugin – данный плагин обеспечивает поддержку протокола OpenFlow
- Path Computation Element Protocol (PCEP) – данный плагин обеспечивает поддержку протокола PCEP.

Абстрактная модель контроллера программируемой сети OpenDaylight версии SR4 Beryllium представлена на рисунке 3.2, на котором отображены вышеописанные модули, сервисы и интерфейсы.

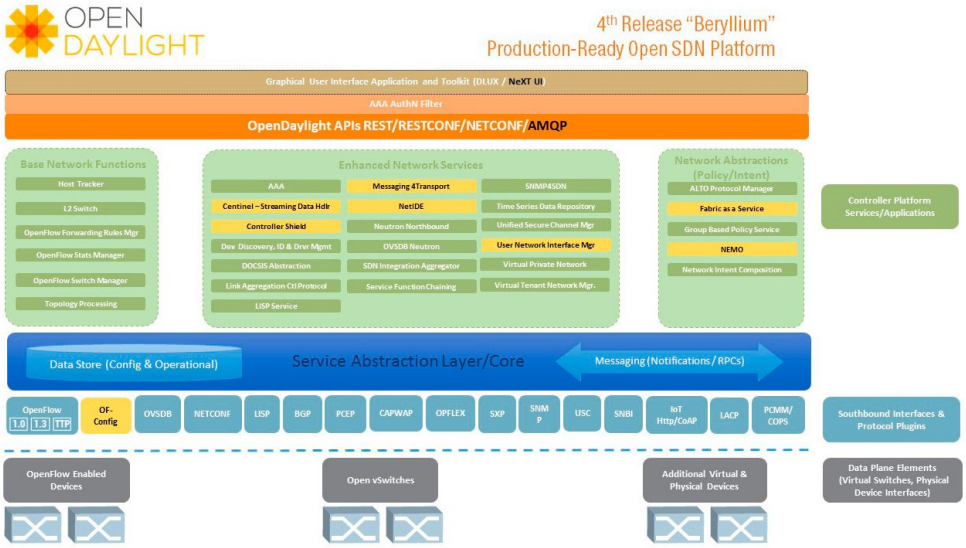


Рисунок 3.2. Абстрактная модель контроллера SDN OpenDaylight Beryllium SR4.

3.2 Сетевая ОС Floodlight

FloodLight – OpenFlow Java-контроллер для компаний и предприятий (класса enterprise) с открытым исходным кодам. FloodLight основан на контроллере Beacon. Имеет лицензию Apache – то есть может использоваться для любых целей.

Использует чистый Java (не требуется OSGI, поддерживается Eclipse). Прост в сборке и запуске.

Является ядром Big Network Controller от Big Switch. Обеспечивается переносимость между приложениями для FloodLight и Big Network Controller.

Архитектура Floodlight и ее особенности, основные компоненты.

FloodLight имеет модульную архитектуру, за счет которой облегчается процесс расширения и внесения изменений. При описании архитектуры используются два основных понятия: сервис и модуль. Сервис – это интерфейс, который экспортирует состояние и генерирует события. Потребители сервиса могут получать/устанавливать состояние и подписывать или отписываться на события. Допускается множество реализаций одного и того же сервиса. Каждый модуль в свою очередь может использовать некоторый набор сервисов (зависимостей) для реализации некоторой функциональности. Модуль может предоставлять соответственно ноль или более сервисов. То есть модули экспортируют сервисы. Все модули FloodLight написаны на языке Java. Все модули имеют минимальное количество зависимостей между собой, что упрощает разработку приложений.

Особенности:

1. Модульная система загрузки, что дает возможности для расширения и наращивания функциональности контроллера.
2. Легкая установка с минимальным набором зависимостей.
3. Поддерживает широкий диапазон физических и виртуальных OpenFlow коммутаторов.
4. Поддерживает интеграцию с не-OpenFlow сетями, т.е. он может управлять множеством «островов» физических OpenFlow коммутаторов.
5. Одна из основных целей разработки – высокая производительность.
6. Поддерживает платформу OpenStack cloud orchestration.

Модули контроллера реализуют общие функции для использования в большинстве приложений, такие как:

- Обнаружение, выявление и предоставление информации о состоянии элементов сети и событий от них (топология, устройства, потоки).
- Обеспечение взаимодействия контроллера с коммутаторами сети (через OpenFlow протокол).
- Управление модулями FloodLight и распределение ресурсов таких как память, потоки, тесты.
- Web-интерфейс и сервер для отладки (Jython).

Контроллер FloodLight содержит следующие модули:

- Module Manager – система управления модулями FloodLight.
- Thread Pool.
- Packet Streamer – сервис для потоковой передачи пакетов, который может выборочно формировать поток OpenFlow пакетов, которыми обмениваются коммутатор и контроллер для наблюдателя. Он содержит два функциональных интерфейса:

1. REST-интерфейс для того, чтобы определить характеристики OpenFlow сообщений, которые представляют интерес – фильтр.
2. Thrift- интерфейс для потоковой передачи отфильтрованных пакетов.

- Jython Server.
- Web UI.
- Unit Tests.
- Device Manager – отслеживает устройства в сети (MACs, IPs), хосты, отображения MAC-(switch,port), MAC-IP, IP-MAC.
- Topology Manager/Routing – отслеживает соединения между хостами и коммутаторами, вычисляет кратчайший путь с помощью алгоритма Дейкстры.
- Link Discovery – хранит состояние линков в сети, рассылает LLDP пакеты.
- Flow Cache.
- Storage (Memory + NoSql) – уровень абстракции для хранения памяти контроллера. Сейчас используется память (Memory). Реализовано в

стиле базы данных (есть механизм запросов). Модули могут получать доступ ко всем данным и подписываться на их изменение.

- Counter Store – модуль, отвечающий за сбор и обработку статистики OpenFlow и статистики сетевой ОС FloodLight.
- RestServer – реализуется через Restlets (restlet.org), модули должны реализовывать RestletRoutable.

Сервисы FloodLight:

Switches;

1. Controller Memory;
2. PerfMon;
3. Trace.

Базовый набор приложений FloodLight (приложения собраны как Java-модули и скомпилированы с FloodLight):

– VNF (Virtual Network Filter) - приложение для изоляции сетей на основе MAC адресов. Приложение использует REST API для добавления, удаления, получения информации о виртуальных сетях. Не включено по умолчанию. На данный момент работа приложения совместима с релизом OpenStack Essex, ведутся работы по реализации совместимости с релизом OpenStack Folsom.

– Компания Big Switch предоставляет OpenSource фреймворк Floodlight Test для разработки и проведения тестов по интеграции приложений для контроллера.

– Firewall - приложение реализующее поддержку stateless ACL. Приложение использует REST API для включения/отключения, добавления, удаления, просмотра списка правил.

– Static Flow Entry Pusher.

– Port Down Reconciliation - приложение для согласования потоков в случае неисправности в работе порта или канала передачи данных.

– Forwarding – ядро для хранения, вычисления путей и установки потоков (установки правил для них), обрабатывает передачу данных между островами.

– Hub – пример hub приложения. Приложение для реализации функциональности концентратора, которое рассылает входящий пакет по всем активным портам, кроме того порта, откуда пакет пришел. Может заменить routing/forwarding приложения.

– Learning Switch – пример приложения Learning Switch заменяет routing/forwarding приложения.

REST-приложения.

REST-приложения – приложения, написанные на любом языке программирования и построенные поверх REST API интерфейса,

предоставляемого модулями FloodLight и базовыми приложениями. В настоящий момент в FloodLight включены два приложения:

– Circuit Pusher - приложение, которое использует Floodlight REST API для установления двунаправленной связи между двумя IP хостами, т. е. постоянной записи о потоке на всех коммутаторах, входящих в путь между ними.

– OpenStack. Контроллер Floodlight может работать как подключаемый бэкенд к Openstack Quantum. Quantum реализует REST API, которые поддерживаются Floodlight. Данное приложение построено на основе двух основных компонентов Floodlight: модуль VirtualNetworkFilter, который реализует Quantum API, и Quantum RestProxy Plugin, который соединяет Floodlight с Quantum. VirtualNetworkFilter реализует изоляцию OpenFlow сетей уровня доступа к сети, основанную на MAC адресах. Этот модуль входит в Floodlight по умолчанию, и не зависит от работы Quantum. RestProxy плагин работает как часть сервиса Quantum. Приложение транслирует функциональные вызовы Quantum в аутентифицированные REST запросы к набору внешних сетевых контроллеров.

Система загрузки модулей FloodLight. FloodLight использует свою собственную систему загрузки модулей, которая была разработана для следующих целей: определять какие модули должны быть загружены посредством изменения конфигурационного файла, заменять реализации модулей без изменения модулей, от которых они зависят, создать ясную платформу и API для расширения FloodLight и обеспечить модульность кода.

Система содержит несколько основных частей: загрузчик модулей, модули, сервисы, конфигурационный файл и файл, который содержит список модулей, доступных в jar.

Реактивная и проактивная установка правил в FloodLight. В настоящее время FloodLight поддерживает два приложения с реактивной передачей пакетов (Forwarding, Learning Switch), которые имеют различное поведение и работают с разными топологиями и два приложения для проактивной установки правил (Static Flow Entry Pusher, Circuit Pusher).

Приложение Forwarding. Передача пакета между любыми двумя устройствами осуществляется для следующих сетевых топологий:

– Внутри OpenFlow-острова устройство A посылает пакет устройству B в этом же острове. Приложение Forwarding вычисляет единственный кратчайший путь между A и B.

– OpenFlow острова с не-OpenFlow островами между или среди них. Каждый OpenFlow остров может иметь ровно один канал связи с не-OpenFlow островом. Также OpenFlow и не-OpenFlow острова вместе не могут формировать циклы. Приложение Forwarding вычисляет кратчайший путь в каждом OpenFlow острове и ожидает пакеты, переданные через не-OpenFlow острова (предполагая, что каждый не-OpenFlow остров – это единый L2 широковещательный домен). В приложении устанавливается тайм-аут, когда

трафик не передается по пути более указанного времени тайм-аута (по умолчанию 5 секунд).

Приложение Learning Switch – простой L2 learning switch. Приложение для реализации обучения коммутатора уровня доступа к сети. Приложение реализует REST API для получения таблицы коммутатора в виде <известные хосты: vlan: порт>. Приложение может использоваться для следующих сетевых топологий:

– Для использования в любом количестве OpenFlow островов, даже с не-OpenFlow L2 островами между ними.

– Не может работать, если коммутаторы в острове формируются в виде кольцевой топологии или острова в форме кольцевой топологии.

Приложение гораздо менее эффективное по производительности по сравнению с другими подходами.

FloodLight обеспечивает два приложения для проактивной установки правил:

– Приложение Static Flow Entry Pusher – позволяет устанавливать правила для потоков от коммутатора к коммутатору на основе явного выбора портов коммутаторов пользователем. Поддерживает вставку и удаление статических потоков.

– Приложение Circuit Pusher построено поверх Static Flow Entry Pusher, Device Manager и Routing сервисов на основе их REST API для построения единственного кратчайшего пути в пределах одного OpenFlow острова.

FloodLight поддерживает большинство современных OpenFlow виртуальных и аппаратных коммутаторов.

Виртуальными представляют собой Open vSwitch (OVS), а аппаратными как Arista 7050, Brocade MLXe, Brocade CER, Brocade CES, Dell S4810, Dell Z9000, Extreme Summit x440, x460, x670, HP 3500, 3500yl, 5400zl, 6200yl, 6600, and 8200zl (the old-style L3 hardware match platform), HP V2 line cards in the 5400zl and 8200zl (the newer L2 hardware match platform), Huawei openflow-capable router platforms, IBM 8264, Juniper (MX, EX), NEC IP8800, NEC PF5240, NEC PF5820, NetGear 7328SO, NetGear 7352SO и Pronto (3290, 3295, 3780).

3.3 Сетевая ОС Mul

Контроллер Mul разрабатывался для обеспечения производительности и надежности сетей, выполняющих критически важные задачи (с повышенными требованиями к надежности). MuL является важным компонентом сетевой платформы KulOS (коммерческого контроллера) для использования в облаках, и проект поддерживается его компанией-основателем Kulcloud. Также одна из целей, которую преследовали разработчики - поддержка возможности модульного размещения приложений.

Два режима установки Mul контроллера обычный и с повышенной производительностью.

Параметры настройки запуска Mul:

1. Daemon режим работы. Mul запускается как процесс, выполняющийся автономно и работающий в фоновом режиме. Демоны обычно запускаются одновременно с системой и завершаются вместе с ней.
2. Количество потоков для обработки коммутаторов.
3. Количество потоков для обработки приложений.

Количество потоков для обработки сообщений коммутаторов от 0 до 16. Количество потоков для обработки приложений от 0 до 8. По умолчанию количество потоков для обработки сообщений коммутаторов составляет – 4, для обработки приложений – 2.

3.4 Сетевая ОС Maestro

Maestro – масштабируемый, написанный на языке Java OpenFlow контроллер, разработанный в университете Rice University, предоставляет интерфейсы для реализации модульных приложений управления доступом и состоянием сети.

Maestro является платформой, для обеспечения автоматического и программного управления сетью с помощью функций этих модульных приложений.

Архитектура и основные компоненты. Maestro активно использует параллелизм в рамках одной машины, чтобы повысить пропускную способность системы. Принципиальной особенностью программно-конфигурируемых сетей является то, что OpenFlow контроллер отвечает за первоначальную инициализацию потока, поэтому производительность контроллера может быть узким местом сети.

Достоинством Maestro является то, что в контроллере минимизируются усилия программиста, связанные с организацией распараллеливания. Maestro выполняет большую часть утомительной сложной работы по управлению задачами распределения нагрузки и планирования рабочих потоков.

Производительность. В работе была показана эффективность использования параллелизма в пределах одной серверной машины совместно с методами оптимизации пропускной способности, такими как минимизация межъядерных накладных расходов и группирование, которые позволили достичь Maestro почти линейной масштабируемости пропускной способности при обработке потоков на восьми ядерной серверной машине.

Хотя максимальная пропускная способность, которую смог достичь Maestro (600000 запросов в секунду) она все еще далека от требований, предъявляемых к большим центрам обработки данных (более 20 млн. запросов в секунду), ожидается, что Maestro может быть распределен, как сделано в NOX, до масштабов 10 млн. потоков в секунду. Кроме того, масштабируемость

Maestro в пределах одного многоядерного сервера может помочь сократить по крайней мере в 10 раз количество контроллеров, требуемых в NOX.

3.5 Сетевая ОС Beacon

Beacon – быстрый, кросс-платформенный, модульный контроллер на языке Java, который поддерживает как дискретно-событийные, так и потоковые операции.

Архитектура и ее особенности. Основным элементом и единицей сетевой ОС Beacon является пакет (Bundle). Пакет может содержать: метаданные (META-INF/MANIFEST.MA), Java-классы, ресурсы (xml) и другие JAR файлы.

Пакеты можно запускать, останавливать, обновлять, устанавливать во время работы контроллера, не прерывая выполнения других независимых пакетов. Также одновременно может существовать несколько версий одного пакета. Таким образом, контроллер Beacon представляет собой множество пакетов, работающих вместе.

Beacon содержит следующие основные компоненты: OpenFlowJ (протокол OpenFlow v1.0) – реализация протокола на Java, шифратор/дешифратор пакетов (Ethernet, ARP, IPv4, LLDP, TCP, UDP), модули-пакеты: Core, Learning Switch, Hub, Device Manager, модули-пакеты: Topology, Layer 2 Shortest Path Routing, ARP Proxy, DHCP Proxy, Multicast eliminator, модуль декларативной маршрутизации (с помощью загрузки текстового файла) и веб-интерфейс пользователя.

Функционирование контроллера Beacon построено следующим образом: пакет ядра (Core bundle) соединяется с коммутаторами, создается объект класса IBeaconProvider, осуществляющий прием сообщений от коммутаторов и с которым работают все остальные пакеты, создается конвейер для обработки входящих пакетов. На Рисунке 3.3 приведен пример однопоточного конвейера, осуществляющего обработку пришедшего от коммутатора сообщения packet-in о создании нового потока. Сообщение поступает в модуль Core, дешифруется, передается в модуль Device manager, управляющий элементами сети для поиска получателя, затем в модуль Topology для определения местоположения получателя в сети и модуль Routing для создания маршрута для нового потока данных.

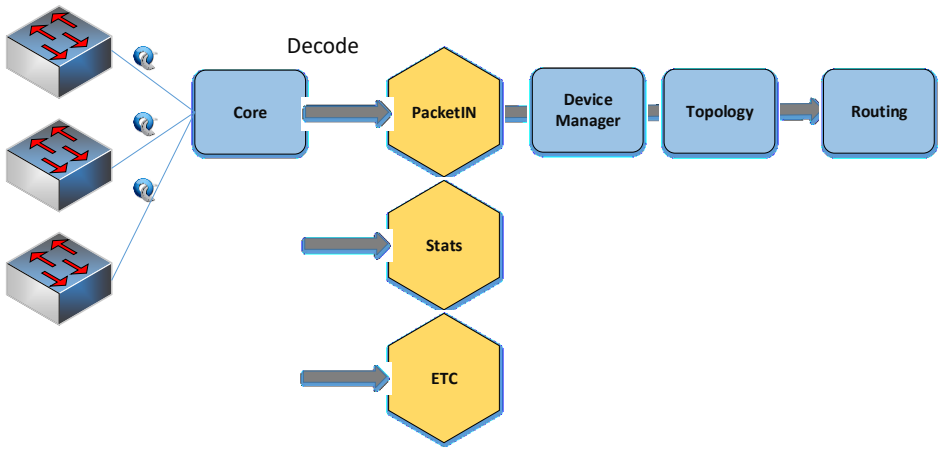


Рисунок 3.3. Конвейер по обработке пакетов.

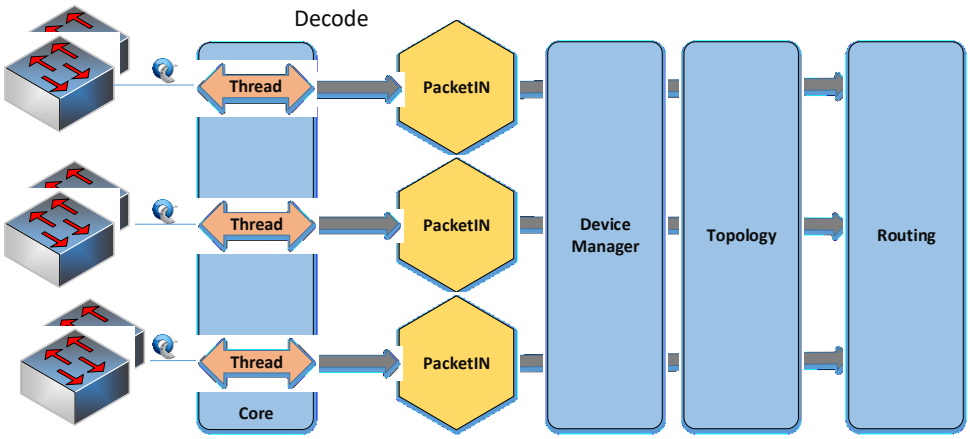


Рисунок 3.4. Многопоточный конвейер по обработке пакетов

Beacon также поддерживает многопоточный конвейер для обработки сообщений представленный на Рисунке 3.4.

Надежность. Beacon разрабатывался в течение полутора лет и использовался в нескольких научно-исследовательских проектах, сетевых классах и пробных опытных сегментах. Beacon в настоящее время поддерживает работу со 100 виртуальными коммутаторами и 20-ю физическими коммутаторами экспериментального центра обработки данных (DNRC) и имеет возможность обеспечить его непрерывное функционирование без простоев в течение нескольких месяцев.

Особенности Beacon:

1. Возможность быстрой разработки приложений – Beacon легко настраивается и запускается. Java и Eclipse позволяют упростить разработку и отладку сетевых приложений.

2. Веб-интерфейс – Beacon опционально встраивается в веб-сервер предприятия Jetty и имеет расширяемый пользовательский интерфейс.

3. Фреймворки – Beacon строится на основе Java фреймворков, таких как Spring and Equinox (OSGI).

Для достижения наибольшей производительности лучше использовать 64 битную Java машину и 64 битную операционную систему.

3.6 Сетевая ОС Ryu

Ryu - это фреймворк, который предоставляет библиотеки и инструменты, которые необходимы для разработки приложений SDN. Платформа обеспечивает базовые функции для управления плоскости данных и функции, которые являются общими для приложений SDN.

Функция OpenFlow контроллера. Ryu совместим с широким набором версий протокола OpenFlow, а также поддерживает большое количество программных и аппаратных реализаций коммутаторов OpenFlow.

Пакетная функция (Bundle function): эта функция предназначена для рассылки и установки одновременно нескольких правил и команд на OpenFlow коммутаторы. Ранее была необходимость подтверждения установки каждой команды индивидуально на стороне приложения. Эта функция облегчает управление сетью, делая возможным задание нескольких правил для отправки на коммутатор в одном пакете.

1. Функция приоритизации (Eviction function): Эта функция позволяет контролировать переполнение буфера коммутаторов. Объем буфера переключатель варьирует в зависимости от возможностей оборудования и степени его загрузки. До создания данной функции существовала необходимость конфигурирования каждого приложения контроллера под определенный тип коммутатора, чтобы предотвратить переполнение буфера памяти. Эта функция облегчает разработку приложений с помощью использования автоматически устанавливаемых, специальных приоритетов.

2. Функция получения информации по оптическому порту (Optical port information collection function): позволяет OpenFlow коммутаторам использовать 10-Гбит/с или 40-Гбит/с оптические порты так же, как и порты Ethernet. В предыдущих версиях, не было возможности для получения параметров оптического сигнала через оптический порт, необходимо было использовать специальные функции вне спецификации openflow для конкретных типов устройств, для получения такой информации.

3. Функция взаимодействия с сетевыми устройствами

Контроллер OpenFlow может передавать различные типы данных, таким же образом, как и традиционные сетевые устройства, следовательно, это дает

возможность построения сети, в которой устройства традиционных сетей будут «сосуществовать» с устройствами OpenFlow. Например, при расширении существующей сети или работы с устройствами, которые имеют сложные алгоритмы управления, с помощью специальных приложений можно частично ввести функции контроллера с обеспечением необходимой совместимости. Ryu предоставляет наборы инструментов для таких введений.

1. Функция сбора информации о сети: чтобы избежать разделения системы управления на две подсистемы, когда конфигурация сети включает в себя работу и с OpenFlow коммутаторами, и с традиционными устройствами, Ryu содержит функцию для сбора информации как от обычных сетевых устройств, так и от OpenFlow устройств. Ryu позволяет осуществлять общее комплексное управление всеми устройствами сети независимо от технологии.
2. Функция конфигурирования: Ryu совмещает в себе два протокола для конфигурации сети OF-CONFIG и NETCONF. Первый позволяет конфигурировать узлы сети, работающие по технологии OpenFlow, а второй – устройства сети на основе традиционных технологий.
3. Функция обмена маршрутной информацией: контроллер Ryu соответствует спецификациям BGP, соответственно между устройствами существующих сетей и коммутаторами OpenFlow возможен обмен маршрутной информацией. Кроме того, изменения, внесенные в таблицы маршрутизации на стороне коммутатора OpenFlow могут быть отправлены на традиционные сетевые устройства с помощью Ryu. Таким образом, обеспечивается взаимодействие между ныне существующими сетями и ПКС, по ранее разработанной технологии BGP между ними. Данная функция облегчает интеграцию SDN и традиционных сетей.

Ключевые слова

сетевая операционная система, ODL, floodlight, MUL, Maestro, Beacon, Ryu

Контрольные вопросы

1. Дайте определение контроллеру SDN сети.
2. Опишите типовую архитектуру контроллера ПКС, кратко сформулировав основные задачи каждого из уровня.
3. Приведите пример базового приложения, написанного для всех контроллеров.
4. В каких «моделях взаимодействия» могут находиться приложение и контроллер? В каком случае реализация приложения обеспечивает высокую скорость обмена сообщениями между ними?
5. Перечислите группы сервисов контроллера и примеры каждого их них.
6. Какой протокол заложен в основу сервиса работы с топологией сети?

7. В чем заключается различие работы протокола BDDP от LLDP?

8. В чем заключается работа сервиса «Spanning Tree»?

9. Перечислите 4 группы приложений контроллеров и их примеры, с точки зрения их функциональности.

10. Назовите основные характеристики контроллера SDN.

11. На каком языке реализована платформа ODL (OpenDaylight)? Какие преимущества и недостатки данного метода реализации? Что такое Karaf?

12. Какую функцию выполняет модуль MD-SAL OpenDaylight?

13. Какой модуль ODL решает вопрос информационной безопасности контроллера?

14. Какой модуль ODL предоставляет графический интерфейс приложений и с сервисов? Из чего он состоит?

15. Назовите модуль, обеспечивающий работу с данными IoT.

16. Какой модуль обеспечивает упрощенную интеграцию с фреймворком OpenStack Neutron?

17. На каком языке реализован контроллер Floodlight?

18. Какая архитектура заложена в сонове Floodlight? Какие два основных понятия используются при описании архитектуры, какая связь между ними?

19. Какой API используется для взаимодействия контроллера Floodlight и приложениями?

20. Какие типы установки правил реализованы в Floodlight, приведите примеры соответствующих приложений?

21. Какие главные функции лежали в основе разработки контроллера Mul?

22. Какие режимы установки Mul контроллера существуют?

23. Сколько потоков требуется по умолчанию, для обработки сообщений от коммутаторов и обработки приложений для контроллера Mul?

24. На каком языке разработан контроллер Maestro?

25. В чем заключается достоинство Maestro?

26. Каковы требования к максимальной пропускной способности больших центров обработки данных? Какова максимальная пропускная способность, которую смог достичь Maestro? Каким способом можно достичь требуемых характеристик больших центров обработки данных?

27. На каком языке разработан контроллер Beacon? Основной элемент (единица) сетевой ОС Beacon?

28. Какой способ функционирования лежит в основе работы контроллера Beacon? Опишите стадии обработки сообщения от OF коммутатора в условиях стандартной модели обработки.

29. Назовите особенности контроллера Beacon.

30. Дайте краткое определение сетевой ОС Ryu.

31. Как реализованы функции сбора информации о сети, конфигурирования и обмена маршрутной информацией в контроллере Ryu?

Литература к главе 3

1. Software-Defined Networking: A Perspective from within a Service Provider Environment [Электронный ресурс] – Режим доступа: https://tools.ietf.org/html/rfc7149 (дата обращения: 07.05.2016).
2. What are SDN Controllers (or SDN Controller Platforms)? [Электронный ресурс] – Режим доступа: https://www.sdxcentral.com/sdn/resources/sdn-controllers/ (дата обращения: 14.05.2016).
3. List of OpenFlow Software Projects Электронный ресурс] – Режим доступа: yuba.stanford.edu/~casado/of-sw.html (дата обращения: 09.05.2016).
4. Прогнозы компании Cisco [Электронный ресурс] – Режим доступа: http://www.cisco.com/web/RU/news/releases/txt/2012/021512.html (дата обращения: 08.05.2016).
5. SDN Testing and Validation Theme & Demo [Электронный ресурс] – Режим доступа: https://www.opennetworking.org/onf-sdn-solutions-showcase/testing (дата обращения: 02.05.2016).
6. Web-страница проекта OpenDaylight [Электронный ресурс] – Режим доступа: https://www.opendaylight.org/ (дата обращения: 02.05.2016).
7. Web-страница проекта Floodlight [Электронный ресурс] – Режим доступа: www.projectfloodlight.org/floodlight/ (дата обращения: 18.05.2016).
8. Web-страница проекта Mul [Электронный ресурс] – Режим доступа: http://sourceforge.net/p/mul/wiki/Home/ (дата обращения: 06.05.2016).
9. Web-страница проекта Beacon [Электронный ресурс] – Режим доступа: https://openflow.stanford.edu/display/Beacon/ (дата обращения: 06.05.2016).
10. Web-страница проекта Ryu [Электронный ресурс] – Режим доступа: https://osrg.github.io/ryu/ (дата обращения: 06.05.2016).

4. ПРЕИМУЩЕСТВА КОНЦЕПЦИИ SDN ДЛЯ РАЗЛИЧНЫХ ТЕХНОЛОГИЙ

4.1 Преимущества концепции SDN для Интернета Вещей

Управление сетью SDN осуществляется сетевым контроллером с установленной на нем сетевой операционной системой, который управляет узлами сети, обрабатывающими пакеты с помощью определенных механизмов, например, OpenFlow. В свою очередь контроллер управляется приложением или скриптом, т.е. управление сетью сводится к написанию нового приложения (скрипта). SDN позволяет следить за поведением сети с помощью инструментов программного обеспечения. В настоящее время функции сетевого оборудования определяются программным обеспечением, однако часто бывают трудности в том, как изменить метод наблюдения и требует специальные знания оборудования.

В соответствии с SDN, приложение контроллера ИВ также разделяет логику контроля от функций физических устройств, через логически централизованный контроллер, который управляет устройствами через стандартный интерфейс. В частности, приложение контроллера ИВ расширяет возможности подхода SDN, начиная от сетевых устройств до датчиков (сенсоры) и облачных платформ, потом объединяет их, для обеспечения поддержки приложений Интернета Вещей, предоставляя четко определенный сервис APIs с точки зрения получения и накопления данных, их передачи и обработки.

На рисунке 4.1, заметим то, что физическая инфраструктура состоит из платформ сенсоров, передающих устройств и серверов. Верхний уровень этой инфраструктуры представляет собой разнообразные приложения ИВ, и каждое приложение через сервис APIs осуществляет настройку, получение и накопление данных, их передачу и обработку. Стандартный сервис API уменьшает сложность и разработку циклов для развертывания нового приложения, в то время как разделение физической инфраструктуры значительно уменьшает капитальные и ЭАО (эксплуатация, администрирование и обслуживание) расходы. Эти особенности, расширяют возможности приложение контроллера ИВ, и позволяют эффективно удовлетворять различным требованиям приложений ИВ.

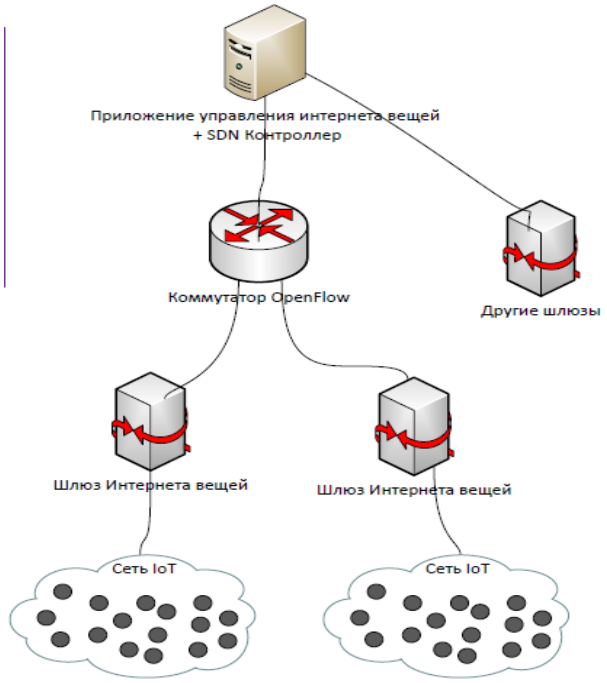


Рисунок 4.1. Структура Интернета Вещей и сетей SDN

На уровне приложения можно написать программу для управления потоками ИВ, то есть программно-конфигурируемую архитектуру ИВ. Как показано на рисунке 4.1, приложение управления ИВ состоит из трех уровней: уровень физической инфраструктуры, уровень управления, и уровень приложений.

Уровень физической инфраструктуры: этот уровень состоит из различных видов физических устройств, в том числе платформ сенсоров, шлюзов, базовых станций, коммутаторов/маршрутизаторов и серверов. Эти устройства содержат основные функции и ресурсы, для управления сенсорных узлов, передачи данных от одного узла к другому, для их обработки и извлечения необходимой информации. Однако, достаточно просто можно добавить механизмы управления. Устройства взаимодействуют с уровнем управления через стандартные интерфейсы, т.е. южный интерфейс, а уровень управления в свою очередь осуществляет контроль всех устройств.

Уровень управления: уровень управления выступает в качестве посредника между физическим уровнем и уровнем приложений. С одной стороны, уровень управления контролирует физические устройства с различными характеристиками и функциями с помощью различных

интерфейсов (южный интерфейс). С другой стороны, уровень управления предоставляет сервисы на уровне приложений через API, известных как северные интерфейсы. Для приложений ИВ, уровень управления обеспечивает сбор и передачу данных и сервисы обработки.

Уровень приложений: на данном уровне, разработчики могут создавать приложения городского мониторинга используя API. В частности, они могут настроить сбор данных, передачу и обработку, не беспокоясь о необходимом изменении конфигурации физических устройств, что значительно упрощает процедуру разработки новых приложений. Кроме того, физическая инфраструктура может использоваться несколькими приложениями, следовательно, расходы на техническое обслуживание снижаются.

В совокупности, SDN/NFV виртуализирует компоненты и службы в сети, снижая задержку и уменьшая потребность в оборудовании. Для устройства и пользователей в ИВ сеть SDN может стать входным шлюзом, а также надежным защитным механизмом.

Приложения SDN могут также управлять различными устройствами в пределах Интернета Вещей. Использование автоматизированного управления на основе политик обеспечивает прозрачный доступ через любое устройство или платформу с постоянной проверкой подлинности устройств и пользователей и сканированием на наличие потенциальных угроз. Виртуализованная среда, предлагаемая SDN/NFV, означает, что сеть доступна для любых устройств, любого типа интерфейса и всех типов пользователей.

SDN может обеспечить безопасность в Интернете Вещей, SDN автоматически настраивает сетевые устройства, собирает данные, необходимые для аналитики, не предоставляя при этом доступ к сведениям о базовой инфраструктуре устройств.

Также Сеть SDN предназначена для проверки подлинности устройств и пользователей и применения заданных правил доступа.

SDN значительно упрощает подготовку сети и способна устранять входящие угрозы безопасности по мере их возникновения, что обеспечивает эффективную работу пользователя на любом устройстве, на любой платформе, в любой операционной системе и браузере.

SDN может обеспечить гибкость, необходимую для управления услугами Интернета Вещей. Одна из самых больших проблем, с которой сталкиваются сетевые администраторы в IoT — это необходимость собирать данные и проводить анализ для мгновенного улучшения предоставляемых услуг пользователям. SDN делает это гораздо быстрее и лучше, чем аппаратные сетевые компоненты. При необходимости SDN может автоматически перенаправлять трафик, согласно полученным аналитическим данным, значительно ускоряя этот процесс.

При использовании SDN устранение неисправностей в сети осуществляется главным образом автоматически. Виртуализация, то есть в данном контексте симбеоз технологий SDN/NFV, также позволяет

организациям предоставлять возможности Интернета Вещей гораздо дешевле, чем в сети на базе оборудования.

4.1.1 Требования к протоколам взаимодействия IoT и SDN

Рассмотрим структуру взаимодействия SDN и IoT, например, как рисунке 4.2 показано то, что в каждом сетевом сценарии имеется только один узел, выступающий в качестве контроллера, а оставшиеся узлы находятся под управлением. В будущем планируется и дальнейшее развитие данного решения, путем расширения имитированной среды, учитывая возможности реального транспортного сценария, обслуживаемого несколькими контроллерами.

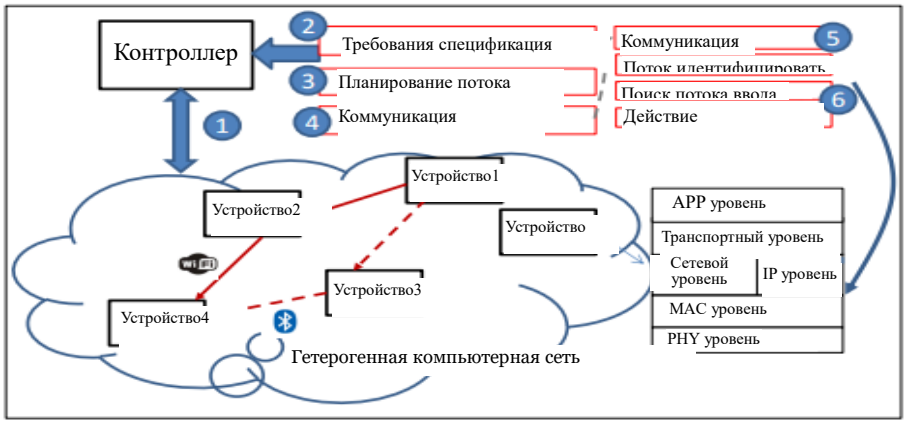


Рисунок 4.2. Схема протокола взаимодействия IoT и SDN

На рисунке 4.2 показан принцип работы протокола в программно-определяемом стиле организации сети:

1. Требования к службам или приложениям, сетевая топология и свойства устройств заносятся в контроллер и сохраняются в базе данных;
2. Контроллер переводит требования к службам в требования к качеству сетевых служб. При необходимости выполняются предварительная обработка и анализ;
3. Контроллер применяет алгоритм для составления графика потоков, чтобы удовлетворить требования к качеству услуг;
4. Контроллер отправляет записи о потоках на управляемые устройства, отвечающие за маршрутизацию потоков. Запись о потоке содержит такую информацию, как исходный/конечный IP-адрес/порт, IP-адрес следующего сегмента и новый конечный IP-адрес;
5. Управляемые устройства получают записи о потоках от контроллера;
6. Управляемые устройства идентифицируют каждый проходящий сквозь них поток (по исходному/конечному IP-адресу/порту) и проверяют наличие

записи для данного потока, затем выполняют действия, определенные IP-адресом следующего сегмента и новым конечным IP-адресом.

Обратите внимание, что одно из важных различий между SDN в средах IoT и в DCN (Data Communication Network) состоит в том, что конечные устройства в IoT обычно обладают несколькими сетевыми интерфейсами, и часто происходит горизонтальная/вертикальная передача. Тем не менее, как только промежуточное устройство перенаправляет поток, оно должно изменить не только следующий сегмент, но и конечный IP-адрес. Конечно, эта процедура требует более защищенного механизма на промежуточном устройстве.

4.1.2 Анализ нового подхода для взаимодействия SDN и Интернета Вещей IoTDM

Модуль «Управление данными Интернета Вещей» (IoTDM) на основе платформы OpenDaylight (ODL), который является развитием связующего программного обеспечения (middleware) центра данных на основе совместимого брокера данных Интернета Вещей oneM2M и позволяющий авторизировать приложения для извлечения данных IoT, загруженные с помощью любого устройства.

Платформа ODL (Рис.4.3) используется для реализации хранения данных на основе спецификаций oneM2M. IoTDM реализует так называемое виртуальное дерево ресурсов, где каждый узел в дереве представляет собой oneM2M ресурс (данные/подконтейнеры/контейнеры). Как правило, устройства IoT и приложения взаимодействуют с деревом ресурсов на основе стандартных протоколов Интернета Вещей таких, как CoAP, MQTT и HTTP.

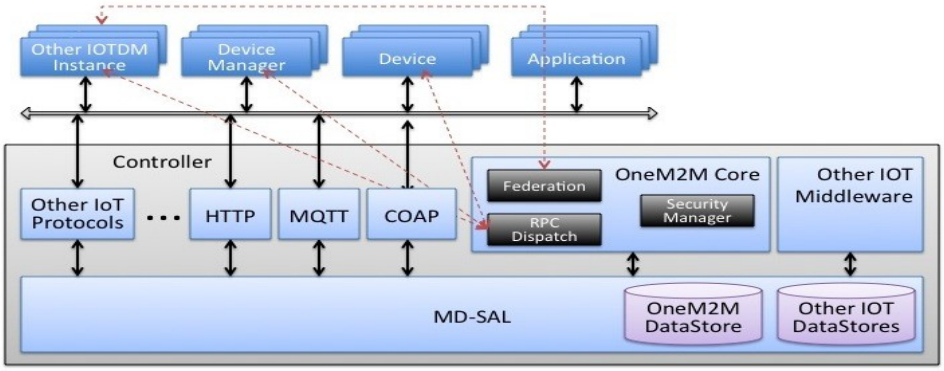


Рисунок 4.3. Архитиктура OpenDaylight

В 2012 году был создан Глобальный партнерский проект oneM2M, способствующий формированию общедоступных и общепризнанных технических спецификаций, и технических отчетов, относящихся, прежде всего, к уровню услуг M2M (M2M Service Layer). В рамках проекта oneM2M

созданы четыре рабочих группы по следующим направлениям разработки: технические требования; архитектура; безопасность; управление, общее описания объектов и их семантика. Результаты работы данных групп пока носят предварительный характер и разрабатываемые документы находятся на стадиях согласования. Инициатива oneM2M предусматривает в идеале формирование единого стандарта услуг M2M.

Применение системы IoTDM возможно, построенной на основе спецификаций oneM2M, в качестве системы управления/инвентаризации устройств или аналитической системой для обработки больших данных. Но иногда приложениям необходимо настроить устройства.

На данный момент, платформа ODL, реализующая широкие возможности управления и контроля в области SDN/NFV, реализует также сервис IoTDM для управления данными Интернета Вещей и предоставляет возможность разработки приложений поверх, через программный интерфейс платформы REST API. В зависимости от необходимых функций, платформа ODL может быть сконфигурирована только с возможностями сбора данных IoT, и развернута рядом с устройствами IoT, или платформа ODL также может быть сконфигурирована для запуска в виде высоконагруженного, масштабируемого и распределенного кластера с функциями IoT, SDN/NFV, развернутого в центре обработки данных.

4.1.3 Архитектура программного обеспечения

Платформа ODL была выбрана для реализации подсистемы управления данными IoT oneM2M из-за ее многофункциональности и принципа модульности архитектуры. Вот некоторые из ее характеристик:

- MDSAL и транзакционные хранилище данных ODL является естественным для хранения дерева ресурсов oneM2M;
- Гибкая поддержка протоколов;
- Поддержка кластеризации для масштабирования и повышения производительности;
- Гибкие возможности распределения/развертывания через OSGI и karaf;
- Потенциал для эффективно использования AAA;
- Функция взаимодействия SDN и NFV;
- Мультиплатформенность решения.

4.1.4 Основы oneM2M

Цель консорциума oneM2M заключается в разработке технических спецификаций для общего связующего программного обеспечения (middleware) IoT. На рисунке.4.4 отображены сценарии операции oneM2M Middleware.

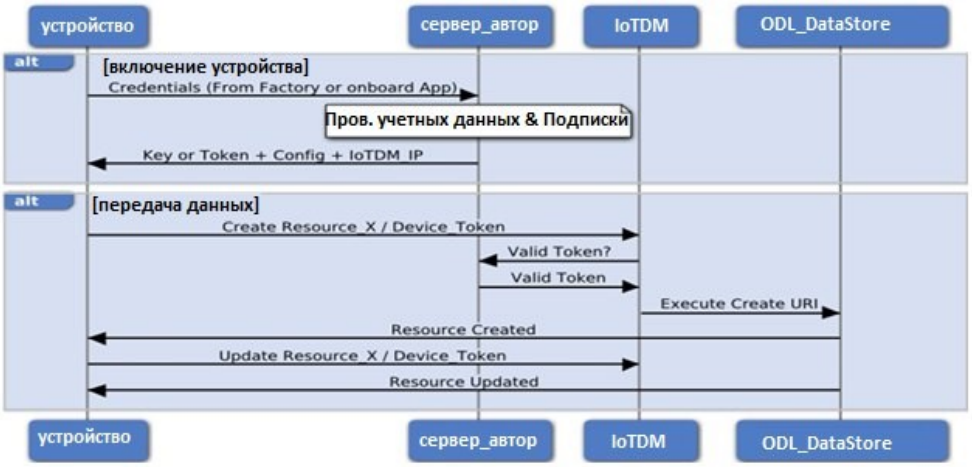


Рисунок.4.4. Операции oneM2M Middleware

Принцип взаимодействия клиента IoTDM и сервера отображено на рисунке 4.5 где указаны основные блоки, которые должны быть на клиенте и контроллере, также указаны на каких языках программирования можно написать скрипты и возможные протоколы уровня приложений.

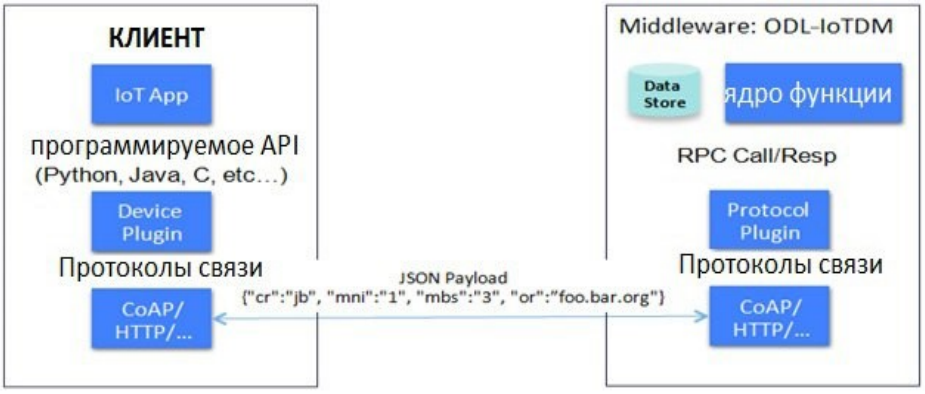


Рисунок.4.5. Взаимодействие клиента (устройства ИВ) с сервером IoTDM.

4.1.5 Особенности протоколов взаимодействия Интернета Вещей и программно-конфигурируемых сетей

OpenDaylight является проектом с открытым исходным кодом и одновременно универсальной платформой для программно-определяемых сетей (SDN). Ее последний релиз, как ожидается, будет внедрен в более чем в 20 коммерческих продуктах, и это при условии взаимодействия с другими

проектами с открытым исходным кодом, включая открытую платформу для сети с функциями виртуализации (NFV) и облака.

Ядро OpenDaylight – программное обеспечение, позволяющее получить доступ к настройкам аппаратных сетевых элементов. Аналогичным образом, IoTDM обеспечивает работу служб, которые выступают в качестве брокера данных IoT и позволяют приложениям размещать и извлекать данные IoT загруженные любым устройством.

Как говорилось ранее, IoTDM совместим с проектом oneM2M, который обеспечивает ряд ограничений для подключения различных устройств через общие службы платформы. Службы позволяют клиентам и операторам осуществлять контроль за активностью Интернет Вещей, например, наблюдать как часто дистанционный датчик фиксирует данные или перенастроить устройства с необходимой частотой обновлений. Сам OneM2M проект опирается на более чем 200 технологии и решения различных компаний, органов по стандартизации и правительственных учреждений.

Как уже было отмечено, платформу IoTDM можно настроить для различных вариантов использования. Простота использования этой платформы было доказана, когда группа студентов из Бостонского университета использовала IoTDM совместно с мобильными и серверными инструментами разработки, чтобы разработать пару приложений для «умного города». Результаты были продемонстрированы в зоне DEVNET для IoT, организованной Cisco на его ежегодном мероприятии Cisco Live в San Diego в июне.

Эти приложения, использовались для визуализации местоположения мобильных устройств на интерактивных картах в закрытом зале. Они объединяли сигналы, полученные от разрозненных Вещей, включая Bluetooth маяков, Интернет маршрутизаторов и GPS сигналы. Для ускорения разработки, студенты использовали открытый исходный код программного обеспечения, включая iOS и Android, а также sigma.js и node.js для поддержки функций на основе браузера.

Одно приложение позволяло пользователям осуществить поиск других зарегистрированных пользователей на их мобильных устройствах и посмотреть карту покрытия, которая отображала в реальном времени трафик людей и историю данных. Другое приложение осуществляло визуализацию IoT узлов сети и данных, включая приложения, пользователей, устройств и агрегированные данные датчиков IoT. Пользователи могли изменять графическое представление сети с использованием курсора для выбора конкретных узлов на их отображение для редактирования, а также создание новых узлов.

4.2 Преимущества концепции SDN для магистральных сетей

Технология плотного спектрального уплотнения каналов (Dense Wavelength Division Multiplexing, DWDM) позволяет организовывать оптические магистральные сети нового поколения, скорости в таких сетях достигают значения в десятки Тбит/с. Такой качественный скачок производительности обеспечивает принципиально иной, нежели в синхронной цифровой иерархии (SDH), метод мультиплексирования - информация в одном оптическом волокне передается сразу на нескольких длинах волн, что позволяет организовать до 160 независимых каналов. Каждая волна несет собственную информацию, при этом для оборудования DWDM неважен способ кодирования и протоколы, используемые для передачи данных - устройства DWDM занимаются только объединением различных волн в один световой поток, а также выделением из общего сигнала.

Благодаря технологии DWDM можно многократно увеличить пропускную способность волоконно-оптических линий связи, не прокладывая новые кабели и не устанавливая на каждое волокно новое оборудования. Кроме громадного увеличения пропускной способности, обеспечиваемого при создании волоконно-оптических сетей, оптический уровень является для телекоммуникационных компаний единственным способом объединения различных технологий, применяемых в их уже существующих сетях, в одну реальную инфраструктуру. Технология плотного волнового мультиплексирования даст компаниям, предоставляющим услуги связи и передачи данных, возможность максимально использовать уже имеющуюся у них инфраструктуру и обеспечить скорость передачи информации, соответствующую потребностям современных сетей.

Транспортные сети стремительно развиваются. Они становятся все более автоматизированными, управляются с помощью программного обеспечения для минимизации эксплуатационных затрат и предоставления новых услуг и приложений более быстрым и эффективным путём. Внедрение концепции ПКС в магистральные транспортные сети считается наиболее перспективным подходом в этом направлении, оно поможет вывести использование сетевых ресурсов на новый уровень, обеспечит мониторинг трафика в оперативном режиме, позволит производить динамическую адаптацию ёмкости сети в зависимости от предъявляемых требований, повысит оперативность внедрения новых сервисов и услуг и их эффективность. Как упоминалось выше, принцип ПКС подразумевает отделение уровня управления от уровня передачи данных, что позволяет внедрить инфраструктуру приложений и сетевых услуг поверх контроллера ПКС. Однако, если внедрять концепцию ПКС в транспортные сети, необходимо делать акцент на физический уровень передачи данных, так как контроллер ПКС должен принимать во внимание аналоговую природу оптической среды передачи. При физических искажениях оптического волокна в DWDM сетях возможны разрывы соединений, а также появление потерь, задержек.

В настоящее время взаимодействие магистральных сетей и программно-конфигурируемых сетей является актуальной проблемой, которая нуждается во всестороннем изучении. Комплексное исследование взаимодействия таких сетей поможет выявить основные режимы совместного функционирования ПКС и магистральных сетей, а также ускорит процесс внедрения программно-конфигурируемых сетей в существующие магистральные сети.

До недавнего времени все разработки в области программно-конфигурируемых сетей были направлены на управление инфраструктурой data-центров. На данный момент, концепция программно-конфигурируемых сетей может быть применена и в транспортных сетях. Такой подход получил название транспортной программно-конфигурируемой сети (T-SDN), основной идеей которой также, как и в традиционной программно-конфигурируемой сети, является разделение плоскостей управления и передачи данных.

Перечислим основные положения, которые закладываются в концепцию транспортных программно-конфигурируемых сетей:

- Упрощение операций по работе с транспортной сетью. Предполагается, что запрос услуги у провайдера будет предельно простым. После того, как клиент заказывает определенный сервис, контроллер автоматически проводит реконфигурацию всех устройств и выделяет необходимые ресурсы, для предоставления пользователю услуги. Процесс получения сервисов таким способом займет считанные минуты;
- Оптимизация использования пропускной способности канала. Предполагается, что решение данной задачи в будущем возьмет на себя контроллер транспортной программно-конфигурируемой сети. Он сможет автоматизировать процесс оптимизации и перераспределения нагрузки транспортной сети таким образов, что клиент будет получать свои сервизы со стабильным уровнем качества обслуживания;
- Пропускная способность по запросу. Предполагается, что, если объем трафика превышает ёмкость канала, которая предоставляется клиенту за определенную плату, возможно автоматическое увеличение пропускной способности клиентского сервиса. Это позволит передавать трафик с большей скоростью в течение небольшого промежутка времени. Финансовые вложения при этом будут минимальны – дополнительная плата должна быть внесена только за тот промежуток времени, в течение которого полоса пропускания была увеличена;
- Построение оптимальных маршрутов. Предполагается, что централизованное управление сетью делает возможным выбор оптимального маршрута предоставления услуг с учетом разнообразных требований и метрик;
- Высокий уровень надежности и доступности предоставляемых услуг. Предполагается, что автоматическое управление сетью контроллером позволит моментально реагировать на сбои в работе сети, а также проводить защитное резервирование маршрута для быстрого восстановления сервисов при сбоях;

- Безопасное конфигурирование сети через приложения высокого уровня. Предполагается, что автоматизированная работа контроллера, обеспечиваемая приложениями, позволит повысить безопасность при конфигурировании сети, в том числе путём исключения «человеческого фактора»;
- Масштабируемость сети. Увеличение числа сетевых устройств и приложений для управления не требует дополнительного конфигурирования контроллера.

Рассмотрим наиболее интересные исследования в данной области.

Компания NetCracker Technology (дочерняя компания корпорации NEC) ведет разработку контроллера транспортной программно-конфигурируемой сети, который мог бы в одиночку контролировать сеть провайдера любого уровня. Ввиду высоких требований к производительности для управления такими сетями, контроллер имеет иерархическую структуру, которая основана на контроллерах двух типов:

- Контроллер нижнего уровня (доменный), который управляет только частью сети (доменом), и возлагает на себя функции непосредственного управления сетевыми устройствами транспортной сети;
- Контроллер верхнего уровня, который управляет доменными контроллерами, абстрагируясь от остальной сетевой инфраструктуры, что позволяет повысить производительность всей системы.

Контроллер верхнего уровня определяет группу устройств, принадлежащих одному доменному контроллеру, как одно устройство. Это позволяет производить мониторинг всей сети целиком. Такой подход даёт возможность предоставлять клиентский сервис по всей сети (от точки входа в сеть провайдера до точки выхода из этой сети), и управлять оптимизацией, доступностью и надежностью предоставляемых услуг.

Стоит отметить, что данный продукт в октябре 2015 года был развернут на сети одного из европейских операторов и позволил автоматизировать часть сетевых операций, тем самым сэкономив время на конфигурирование сети и предоставление сервиса.

Помимо компании NetCracker, активной разработкой продуктов для реализации транспортной программно-конфигурируемой сети занимаются инженеры Huawei Technologies. Компания определяет данную концепцию как ключевую технологию, влияющую на эволюцию и инновацию транспортных сетей. В отличии от традиционных транспортных сетей, транспортные программно-конфигурируемые сети характеризуются автоматизацией, виртуализацией, гибкостью, представляя огромный потенциал для развития сетей.

Хронология Huawei и транспортная программно-конфигурируемая сеть:

- В декабре 2013 года компании Huawei и Telefonica закончили первое полевое исследование транспортной программно-конфигурируемой сети, основанной на синергии IP и оптической сети.

- В августе 2014 года компания Huawei и оператор MTN (ЮАР) закончили первое в мире исследование транспортной программно-конфигурируемой сети и системы DWDM в магистральной сети. Данная инновация получила статус самого влиятельного телекоммуникационного события года в Африке, став лауреатом награды от Africa Com.
- В сентябре 2014 года компания Huawei прошла первый в истории тест взаимной совместимости транспортной программно-конфигурируемой сети. Тестирование было организовано Форумом оптического взаимодействия (OIF) и Фондом открытого сетевого взаимодействия (ONF).
- В октябре 2014 года компания Huawei заявила о партнёрстве с компанией China Telecom в развертывании первой коммерческой транспортной программно-конфигурируемой сети.

Дальнейшим продолжением развития данной технологии стала сенсационная новость, получившая огласку в апреле 2016 года: компании Huawei и Baidu реализуют в Китае оптическую транспортную программно-конфигурируемую сеть для центра обработки данных.

4.3 Преимущества концепции SDN для точек доступа AP

В последнее время развитие беспроводных локальных сетей (WLAN) резко возросли в связи с ростом спроса на беспроводной доступ. Для того, чтобы гарантировать пользователям WLAN всегда лучше качества обслуживания (QoS), типичные системы WLAN всегда состоят из сотен точек доступа (AP), контроллеров беспроводной локальной сети, системы управления и различных услуг в масштабируемой сети. Эти услуги включают в себя контроль доступа, управление мобильностью, балансировка нагрузки, динамической реконфигурации канала и так далее. Есть много WLAN решений, предлагаемых различными производителями. Но большинство этих решений с закрытым исходным кодом и дорогим оборудованием, что ограничивает гибкость и расширяемость сети. Предлагается новый подход к построению WLAN сетей - SWAN (Software-defined Wireless Access Network), открытая платформа WLAN, на основе подхода Software-Defined Networking (SDN), заключающегося в разделении плоскости данных и плоскости управления.

Архитектура SWAN содержит логический централизованный SWAN-контроллер, множество точек доступа, агентов, работающих на верхнем уровне точек доступа, а также набор приложений SWAN. SWAN контроллер имеет глобальный обзор всей сети (общую топологию), и позволяет производить связь операций с помощью API. Приложения, работающие на верхнем уровне SWAN-контроллера, используют API для реализации различных приложений сетевого управления. Пользовательский протокол, называемый протоколом SWAN, используется контроллером для вызова команд к агентам точек и сбор статистических данных из них.

Существует два частных случая построения сети SDN в плотных WLAN сетях: напрямую и через маршрутизаторы. В первом подходе каждая точка доступа должна поддерживать OpenFlow протокол, что позволяет управлять точками доступа и потоками трафика. При построении через маршрутизаторы поддержка протокола точками доступа не обязательна, но остается возможность управления трафиком

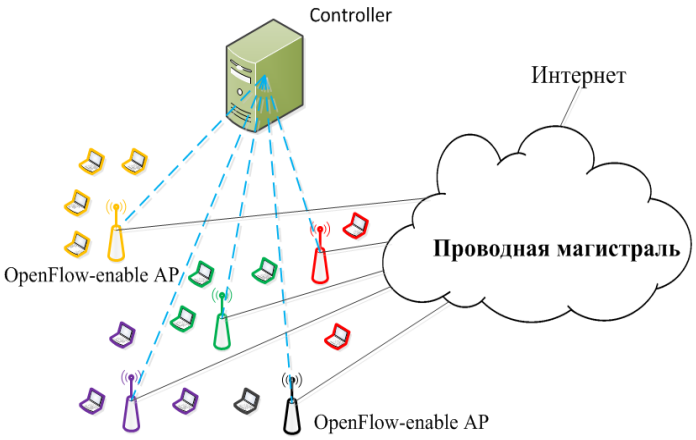


Рисунок 4.6. Управление сетью WLAN на прямую.

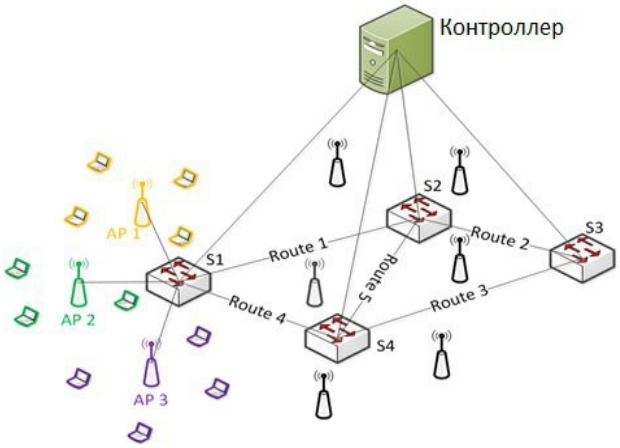


Рисунок 4.7. Управление сетью WLAN через маршрутизаторы

Схема решения для обеспечения *QoS* в беспроводных сетях с высокой плотностью устройств с использованием *SDN/OpenFlow.*

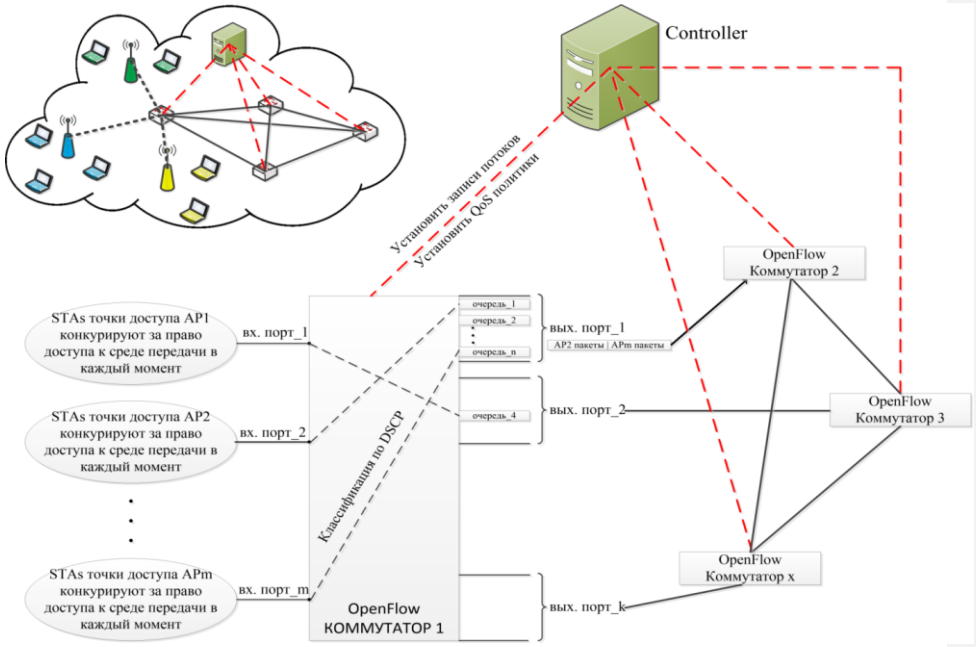


Рисунок. 4.8. Архитектура решения для обеспечения *QoS* в беспроводных сетях с высокой плотностью устройств с использованием *SDN*/*OpenFlow*

Основными преимуществами подхода SDN для WLAN являются: уменьшение количества одновременно работающих беспроводных устройств (снижение интерференции между AP путем гибкого подхода к их управлению), возможность повышения приоритета трафика приложений, чувствительных к задержкам, также возможность легкости применения QoS-политики с динамичным обеспечением QoS через программные приложения, установленные на SDN контроллере.

4.4 Преимущества концепции SDN для мобильных сетей 5G

В настоящее время пятое поколение мобильной системы связи (5G) обсуждается как в науке, так и в промышленности. Основное внимание сейчас уделяется вариантам использования и техническим требованиям к 5G, а не технической реализации, которая наступит на более позднем этапе. Требование к этому поколению связи состоит в том, что сети 5G должны поддерживать связь с низкой задержкой, для реализации услуги «Тактильный Интернет». В предыдущих поколениях, а именно 3G и 4G имеет большое значение мультимедийный контент в то время, как 5G предоставляет возможность управления и контроля в Интернете Вещей (IoT) и задержка становится значительно более важным параметром по сравнению с другими. Говоря про сети связи пятого поколения, говорится уже не только о модернизации сети,

обеспечивающей беспроводной доступ, а согласно документам Международного Союза Электросвязи новой архитектуре системы связи, которая, в целом включает в себя беспроводной интерфейс, ядро сети и облачные сервисы. В настоящее время базовая сеть осуществляет передачу информации в виде пакетов из одного места в другое. Основная концепция существующих пакетных коммутируемых сетей - это хранить и передавать, которая является слишком жесткой и негибкой с точки зрения контроля и управления, чтобы поддерживать требования 5G, однако текущие разработки в области программного обеспечения сетей SDN «открывают дверь» для решения данного вопроса.

5G - (Мобильные сети пятого поколения) наименование, употребляемое в редких научных исследованиях и разработках как средство идентификации следующих главных этапов мобильного поколения после 4G. При разработке и исследовании сетей 5G, первоочередным является обеспечение, как минимум, десятикратного роста скорости и объёмов потребляемого трафика, по отношению к сетям предыдущего поколения, но при этом не увеличивая энергетические затрат и стоимость эксплуатации. К тому же, они будут работать сразу в нескольких режимах, поддерживая промышленное оборудование и многие другие устройства.

Приведем небольшую статистику. В 2012 году число мобильных пользователей, достигло 1,2 млрд. К 2018-му это количество уже достигает 5 млрд. Объем передаваемых данных в промежутках 2012-го и 2013-го возрос вдвое, а к концу 2018-го увеличивается в 12 раз. Увеличение количества мобильных устройств и качественное изменение трафика послужит поводом для роста нагрузки на каналы передачи данных, что приведет к необходимости увеличения скорости. С учетом беспроводных датчиков, промышленного оборудования и множества потребительских гаджетов, через несколько лет на каждого человека будет приходиться около 10 мобильных соединений. Если к мобильному миру присоединятся 5 млрд. человек, то сетям 5G придется обслуживать порядка 50 млрд. соединений.

После развертывания и внедрения сетей 5G, операторы будут вынуждены уменьшить стоимость передачи мобильных данных. Большинство потребителей не замечают сегодня падения цен, потому что объемы их трафика продолжают расти. Благодаря повышению эффективности сетевых механизмов стоимость передачи 1 Мбайт данных ежегодно снижается примерно на 50%, и если в 2008 году она составляла около 46 центов, то сегодня – от 1 до 3 центов. Но при этом счета, приходящие к абонентам в конце месяца, не уменьшаются, поскольку уровень их потребления ежегодно увеличивается в два раза и даже больше.

Уже сегодня считается, что средний объем ежемесячного трафика, приходящийся на каждого человека в сетях 5G, будет составлять около 50 Гбайт. Обобщим все сказанное в рисунке 4.9.

Рис. 4.9. Факторы, обусловившие необходимость появление сети

Фундаментом 5G будут контекстно-ориентированные сети (context-aware radio networks). В общем смысле, «Умная» сеть, как еще называют 5G, сможет: переключаться между различными стандартами до того, пока качество связи не ухудшилось; например, обновлять медиа-контент устройства перед заездом в туннель; рекламировать нынешние промо-акции по пути в торговый центр. Специалисты давно начали работу над этими инновационными технологиями, которые придут к 2020 году.

Следующим новшеством сетей 5G будет виртуализация. Вся инфраструктура будет перестроена в «облака». В результате виртуализации операторы смогут уменьшить расходы БС на 35 - 50 %;

В результате разработок сетей 5G будет осуществляться предстоящее формирование M2M. Однако для полноценной организации взаимодействия M2M требуется создать мощную технологию, позволяющую оборудованию быть включенными в течение долгого времени. Так как приборы, подключаясь к сети и объединяясь между собой, обмениваются большим объемом информации, в результате чего происходит быстрая разрядка аккумулятора.

Ко всему прочему, в результате дефицита радиочастот, началась разработка технологии на участке маленьких, миллиметровых волн, с очень просторным спектром. Но существенным недостатком этих волн является работа на очень малые расстояния.

Далее приведем более подробное описание мобильной сети пятого поколения.

Стандартизация мобильной связи 5-го поколения будет завершена приблизительно в 2020 году. По сравнению с сетью 4G, сети 5G должны достичь в 1000 раз большей пропускной способности и в 10 раз большей спектральной эффективности. Предполагается, что пиковая скорость передачи данных возрастет до 10 Гбит/с для терминалов с низкой мобильностью и до 1 Гбит/с для терминалов с высокой мобильностью. При этом, в 25 раз в среднем увеличится пропускная способность ячейки. Цель создания систем пятого

поколения состоит в том, чтобы соединить весь мир людей и машин между собой: человек- человек, человек- машина, и машина - машина, где бы они не находились. Это означает, что сети 5G должны быть в состоянии поддерживать связь для некоторых специальных сценариев по сравнению с сетями 4G, например, для пользователей высокоскоростных поездов, которые могут развивать скорость до 500 км / час. Перспективные технологии, которые должны обеспечить новые возможности для сетей 5G включают в себе D2D соединения, M2M соединения, миллиметровый частотный диапазон mmwave *MIMO,* программно-управляемые сети *SDN*, сверхплотные сети и т.д. Большинство этих технологий основаны на идеях самоорганизующихся сетей.

Рисунок 4.10. Мобильная сеть 5G в 2020 г.

Ключевые слова

Преимущества концепции SDN, Интернет Вещей, магистральные сети, точки доступа AP, мобильные сети 5G.

Контрольные вопросы

1. Каким образом выглядит структура взаимодействия Интернета Вещей и SDN. Выделите уровни взаимодействия, кратко описав их функции.
2. Какие возможности предоставляет централизованное управление IoT? Перечислите положительные стороны SDN для IoT, также предложите на примере, возможно отрицательные стороны.
3. Какое важное различие можно выделить между SDN в средах IoT и в DCN?

4. На основе каких проектов и спецификаций был разработан проект IoTDM?

5. Каким образом контроллер IoTDM представляет сеть IoT? С помощью каких протоколов происходит взаимодействие устройств и формируемой структурой IoTDM?

6. Каким образом может быть развернута система IoTDM?

7. Какие функции платформы ODL явились основными для реализации oneM2M на ODL?

8. С помощью какой технологии магистральных сетей возможно достичь до 160 независимых каналов в одном физическом соединении? Каких скоростей позволяет достичь данная технология на магистральных сетях?

9. Что требуется учитывать (уровень OSI), при внедрении ПКС на магистральные сети?

10. Назовите основные положение, которые закладываются в концепцию T-SDN?

11. Каким образом компания NetCracker Technology предлагает решить задачу высоких требований к производительности контроллера T-SDN? Опишите принцип работы такой архитектуры.

12. Опишите два варианта построения сети SDN в плотных WLAN сетях. В чем заключается преимущества и недостатки одного варианта построения над другим?

13. Назовите основные преимущества подхода SDN для построения WLAN (SWLAN) сетей?

14. Какое требование накладывает «Тактильный Интернет» на сеть 5G?

15. Сформулируйте требуемое условие разработки и исследования сетей 5G, с точки зрения дальнейшего внедрения на коммерческие сети операторов связи.

16. Какие факторы обусловили необходимость появления мобильной сети 5G?

17. Каких показателей предполагается достичь в сетях 5G?

Литература к главе 4

1. А.Е.Кучерявый. Самоорганизующиеся сети и новые услуги. Электросвязь №1, 2009.

2. А.Е.Кучерявый,А.В.Прокопьев,Е.А.Кучерявый. самоорганизующиеся сети. Любавич, СПб, 2011, 312 с.

3. Кучерявый А.Е. Интернет Вещей. «Электросвязь». – №1, 2013.

Б.С.Гольдштейн, А.Е.Кучерявый. Сети связи пост-NGN. БХВ, С.Петербург, 2013.

5. ПАКЕТЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ SDN

5.1 Имитационное моделирование программно-конфигурируемой сети в пакете Mininet

Mininet является программой-эмулятором сетей на ограниченном ресурсами персональном компьютере или виртуальной машине. Симулятор создает сеть с виртуальными хостами, коммутаторами и контроллерами. Инструмент для исследований и быстрого моделирования программируемых сетей SDN, mininet представляет собой «полигон» для проведения экспериментов с SDN.

Симулятор является программным обеспечением с открытым исходным кодом, который симулирует OpenFlow устройства и SDN контроллеры. По умолчанию, симулятор содержит контроллер, но часто в экспериментах, которые требуют дополнительных функций определенного типа, существует возможность внешнего подключения контроллеров, таких как: POX, RYU, ONOS, и т.д.

Топологии сетей в mininet. Механизм Cgroups, предназначенный для ограничения и изоляции вычислительных ресурсов в ОС Linux, обеспечивает процессы с сетевыми интерфейсами, таблицами маршрутизации и ARP-таблицами в рамках одной системы, т. е. позволяет запустить множество процессов в изолированном и ограниченном по ресурсам окружении.

Mininet может создавать в пространстве ядра OpenFlow контроллеры и хосты, а также изменять конфигурации моделируемой сети.

Адаптированная реализация Open vSwitch позволяет созвать виртуальные коммутаторы. Mininet с поддержкой протокола OpenFlow предоставляет описывать топологию в симуляторе с помощью языка программирования Python.

С помощью команды **-mn** в виртуальной сети mininet реализуется изменение параметров хостов и коммутаторов развертывание сетевой топологии и т. п.

Созданная виртуальная сеть в mininet не является постоянной; так как она создается при вызове mn с конкретными параметрами или без них, и разрушается при выходе из оболочки. Даже большая сеть с сотнями хостов и десятки коммутаторов создается в считанные секунды. И все это на однопроцессорной виртуальной машине с одним гигабайтом оперативной памяти. Для выхода в оболочку ОС и закрытие виртуальной сети используется команда «quit». При некорректном завершения интерпретатора повисшие процессы можно закрыть командой «sudo mn –c».

Симулятор содержит по умолчанию четыре базовых топологии.

1. Minimal: самая простоя из всех топологий, состоит из одного OpenFlow коммутатора и двух хостов, подключенных к ним под управлением OpenFlow контроллера.

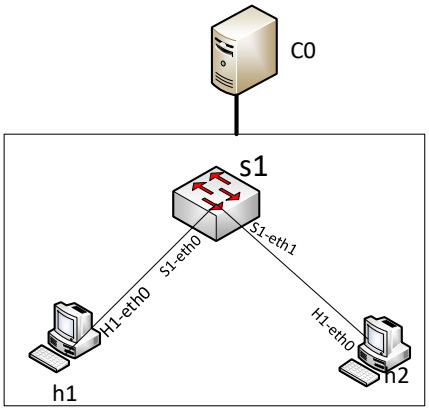


Рисунок.5.1. Single: простая топология с n-хостов подключенные к одному коммутатору.

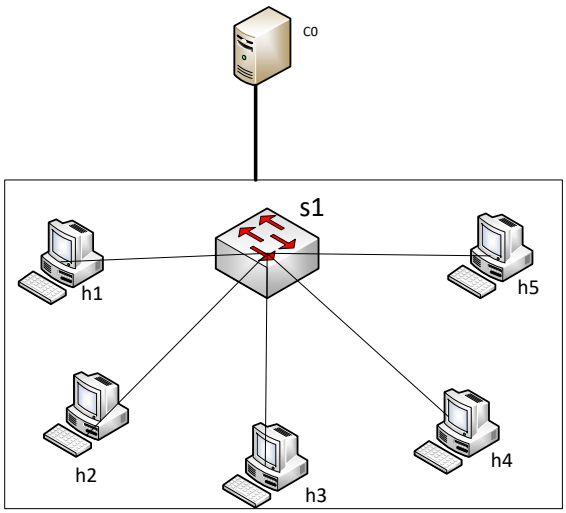


Рисунок.5.2. Linear: Топология linear создает n-хостов подключенные к собственным n-коммутаторам

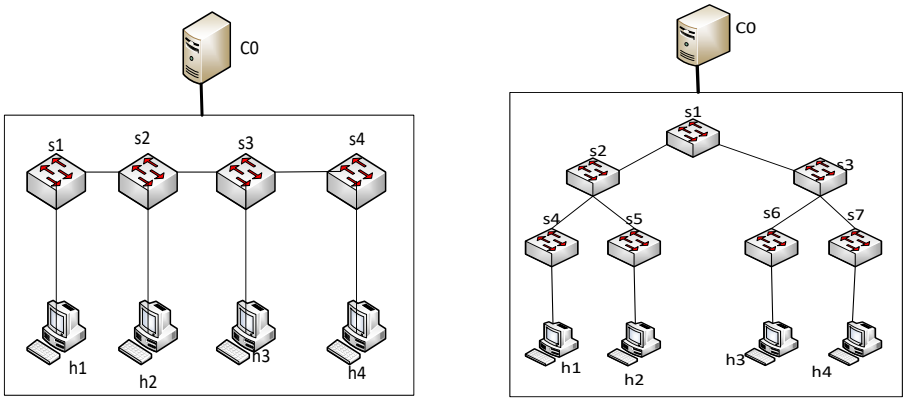


Рисунок.5.3. Tree: Сложная топология, состоящая из n-уровней. Топология позволяет указать глубину коммутаторов(depth) и число подключенных к ним хостов(fanout).

Кроме того, mininet позволяет также создать топологии по желанию. Для этого требуется писать небольшой программный код, используя интерфейс Python API. Такая топология может быть сохранена и использована для дальнейших исследований.

Для работы с коммутаторами OVS используются команды:

- «ovs-vsctl» - для создания и управления коммутаторами;
- «ovs-ofctl» - для создания и управления правилами передачи данных;
- *nodes, net, dump* - просмотр топологии информации и связь между узлами.

5.2 Имитационное моделирование программно-конфигурируемой сети в пакете NS3

Network Simulator 3 предоставляет исследователю набор классов, пользуясь которыми, наследуя и модифицируя, можно смоделировать широкий спектр протоколов и процессов, происходящих в сетях. Также симулятор позволяет моделировать процессы в реальном времени и интегрировать его с испытательным стендом (testbed), делать испытательный стенд частью моделируемой сети и т.д. Симулятор NS-3 содержит множество тестов для всех компонент, что гарантирует достоверность получаемых результатов. NS3 является очень гибким и в то же время мощным средством моделирования за счёт использования C++ в качестве встроенного языка описания моделей. Так же, помимо C++, может использоваться Python. Оба языка в симуляторе равноправны и принимаются для описания моделей телекоммуникационных систем.

Моделирование SDN в пакете имитационного моделирования NS3.

Модуль OpenFlow 1.3 для NS-3, также известный как OFSwitch13 модуль был разработан для улучшения NS-3 сетевого симулятора с поддержкой технологии Software-Defined Networking (SDN). Несмотря на то, что NS-3 уже имеет модуль, который поддерживает моделирование с OpenFlow коммутаторами, но имеющаяся реализация обеспечивает очень устаревший протокол OpenFlow (версия 0.8.9, с 2008).

Модуль OFSwitch13 обеспечивает поддержку протокола OpenFlow версии 1.3, в результате чего оба коммутирующих устройства и интерфейс приложения контроллера подключаются к симулятору NS-3, как показано на рисунке ниже. С помощью этого модуля можно соединить Ns-3 узлов для передачи и приема трафика с использованием существующих устройств CSMA. Для того, чтобы организовать сеть, интерфейс приложения контроллера может быть расширен для реализации любой нужной логической схемы управления. Связь между контроллером и коммутатором осуществляется через стандартный протокол NS-3 стека, устройств и каналов. Модуль также зависит от внешнего OpenFlow 1.3 программного обеспечения коммутатора для NS-3, собранным в качестве библиотеки, также известный как ofsoftswitch13 библиотека, которая обеспечивает реализацию коммутатора канала данных, код для преобразования сообщений OpenFlow и из формата проводной, а dpctl утилиты инструмент для конфигурирования коммутатор из командной строки.

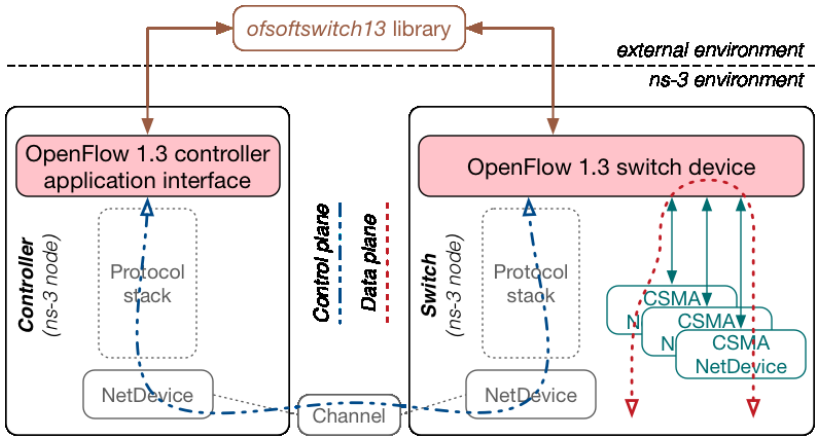


Рисунок.5.4 Архитектура модуля OFSwitch13.

Ключевые слова

Имитационное моделирование, программно-конфигурируемое сети, пакет Mininet, пакет NS3.

Контрольные вопросы

1. В чем заключается особенность симулятора Mininet?
2. Назовите варианты организации подключения контроллера к OF сети.
3. Какие методы существуют в Mininet для построения топологии сети.
4. Сохраняется ли созданная топология при выходе из программы-симулятора?
5. С помощью какой команды Mininet создается сеть?
6. Назовите и опишите основные четыре базовых топологии, которые содержит симулятор.
7. Какие возможности предоставляет пакет имитационного моделирования NS3?
8. Какой модуль NS3 обеспечивает поддержку протокола OpenFlow 1.3?
9. В чем заключается преимущества использования пакета имитационного моделирования NS3 надо другими?

Литература к главе 5

1. Rogério Leão Santos de Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, Ligia Rodrigues Prete, “Using Mininet for Emulation and Prototyping Software-Defined Networks”, Brazil, ©2014 IEEE.
2. Mininet. An Instant Virtual Network on your Laptop.2014, Accessed: Sept. 2014[Online]Available: http://mininet.org.
3. Network Simulator 3: Discrete-event network simulator. NSNAM [online]. Available from: www.nsnam.org
4. OMNeT++: Discrete Event Simulator. [online]. Available from: www.omnetpp.org

6. ТЕСТИРОВАНИЕ СЕТЕЙ SDN

6.1. Виды тестирования

Основными видами тестирования SDN контроллеров, позволяющими наиболее полно определить какие-либо дефекты, установить корректность функционирования и соответствие ключевых показателей предъявляемым требованиям являются:

- Тестирование производительности: измеряется пропускная способность (количество запросов от коммутаторов, обрабатываемых контроллером в секунду) и задержка (время, затрачиваемое контроллером на обработку одного запроса);
- Тестирование ресурсоемкости: оценивается загрузка ядер процессора и использование физической памяти в процессе работы контроллера;
- Тестирование масштабируемости: при исследовании масштабируемости контроллера рассматривается зависимости показателей производительности от количества ядер процессора, соединений между коммутаторами, конечных узлов сети;
- Тестирование функциональных возможностей: проверка корректности обработки контроллером сообщений протокола OpenFlow от коммутаторов;
- Тестирование надежности: измеряется количество отказов за время тестирования и время безотказной работы при заданном профиле нагрузок
- Безопасность: устойчивость контроллера ПКС к некорректно сформированным сообщениям протокола OpenFlow.

6.2. Существующие реализации программного обеспечения для тестирования SDN контроллеров

6.2.1. Hcprobe

Hcprobe – это фреймворк для тестирования SDN контроллеров. Он включает в себя библиотеку, написанную на языке программирования Haskell которая обеспечивает средства для работы с протоколом OpenFlow. Помимо этого, она включает в себя стандартную реализацию программного OpenFlow коммутатора и встроенный предметно-ориентированный язык (EDSL) для создания новых коммутаторов с требуемыми параметрами и написания программ для них.

Hcprobe предоставляет комплект библиотек, которые могут быть использованы для создания различных видов тестов:

- Производительности;
- Функциональности;
- Безопасности.

Ядром Hcprobe является FakeSwitch – модель OpenFlow коммутатора с изменяемыми параметрами и конфигурацией. Коммутатор производит инициализацию соединения с контроллером, определяет поведение коммутатора по умолчанию, при получении определенных типов сообщений OpenFlow и предоставляет функции для использования в собственных тестовых сценариях

Для реализации собственных тестовых сценариев для использования в Hcprobe придется использовать либо Haskell либо встроенным предметно-ориентированным языком, что удобнее. EDSL предоставляет средства для создания любого количества коммутаторов с требуемыми настройками и определения логики их работы, например, отправку сообщения любого типа из протокола OpenFlow на контроллер и сбора различной статистики.

Стандартный тест, написанный на EDSL включает в себя определение модулируемого коммутатора, сценарий отправки сообщений на SDN контроллер и установку обработчиков входящих и исходящих сообщений, которые собирают статистику, либо изменяют поведение коммутатора.

Hcprobe включает стандартную реализацию коммутатора с параметрами тестирования по умолчанию. Этот тест моделирует 16 OpenFlow коммутаторов которые пытаются подключиться к контроллеру по адресу 127.0.0.1, порт 6633. Можно изменить IP и TCP порты, количество моделируемых коммутаторов и уникальных MAC адресов для каждого.

Также в комплекте идут несколько тестовых сценариев на EDSL для тестирования контроллера:

- length.hs – отправка пакета формата PacketIn с некорректным полем длины в заголовке
- version.hs – отправка пакета формата PacketIn с некорректной версией протокола OpenFlow
- type.hs – отправка сообщения OpenFlow с некорректно указанным типом
- proto.hs – оправка сообщения формата PacketIn с некорректным сформированным кадром Ethernet
- ascii.hs - оправка сообщения формата PortStatus с некорректно сформированной ASCII строкой в поле port_name

6.2.2. Cbench

Cbench – программное обеспечение с открытым исходным кодом для тестирования пропускной способности и задержки SDN контроллеров, находящееся в свободном доступе. Он написан на языке программирования Си и входит в состав пакета эмулятора компьютерных сетей Mininet, который предельно близко эмулирует реальную физическую сеть. Cbench использует некоторые библиотеки Mininet для эмуляции SDN сети и генерации трафика. Cbench - это инструмент для тестирования OpenFlow контроллеров путем генерации и отправки пакетов формата PacketIn. Он эмулирует группу

коммутаторов, которые подключаются к контроллеру, отправляют PacketIn сообщения и считают ответные пакеты FlowMod.

Установка параметров тестирования, его запуск и вывод результатов осуществляется с помощью интерфейса командной строки, путем ввода команд определенного формата.

6.2.3. Ctltest

Ctltest - это собрание сценариев для тестирования производительности SDN контроллеров поддерживающих протокол OpenFlow. Сценарии используют Cbench для измерения пропускной способности и задержки. Это собрание включает сценарии для семи популярных OpenFlow контроллеров: NOX, POX, Floodlight, Beacon, Mul, Maestro и Ryu. Также предоставляется возможность для добавление других контроллеров для тестирования.

Сценарии устанавливают и запускают контроллеры на одном сервере и, затем, по SSH запускают Cbench на другом. Cbench отправляет сообщения формата PacketIn на контроллер и регистрирует ответы контроллера (сообщения: FlowMod или PacketOut).

Сценарии запускают несколько тестов, которые помогают анализировать взаимосвязь между производительностью контроллера и количеством доступных ядер ЦП, количества коммутаторов и хостов. Также предоставляется инструмент для построения графиков этой зависимости.

6.3. Разработка программного обеспечения для тестирования SDN контроллеров

6.3.1 Разработка пользовательского интерфейса

Для реализации удобства предоставления данных тестирования, требуется реализовать графический интерфейс пользователя. В данном разделе описываются основные этапы разработки GUI программного обеспечения по тестированию контроллеров SDN.

Рисунок. 6.1. Этапы разработки графического интерфейса

Проектирование.

Разработка интерфейса обычно начинается с определения задачи или набора задач, для которых продукт предназначен. Задачей этапа проектирования является анализ цели разработки, формирование функциональных требований и определения как именно будут реализовываться требования заказчика к системе. На данном этапе формируются основные принципы, которые впоследствии будут дополняться, уточняться и использоваться на протяжении всего срока жизни системы.

Функциональные требования:

Целью разработки графического интерфейса пользователя для разрабатываемого программного обеспечения является предоставление пользователю возможности быстрой и простой настройки конфигурации исходных данных для тестирования, получения исчерпывающего отчета о его результатах в понятном и максимально удобном пользователю виде.

Исходя из цели, сформулируем основные функциональные требования к графическому интерфейсу:

- Должны быть реализованы поля для ввода исходных данных, таких как адрес и порт контроллера, количество моделируемых коммутаторов, количество уникальных MAC адресов отправителей на коммутатор, количество и продолжительность тестов, режим тестирования (задержки/ пропускной способности), длина отправляемых пакетов, интервал между пакетами;
- Должен быть реализован вывод подробного отчета о ходе и результатах тестирования в текстовой форме;
- Должен быть реализован вывод результатов тестирования в виде графиков отображающих зависимость результатов от ключевых параметров тестирования. В зависимости от выбранного режима тестирования график должен отображать различные зависимости, характерные для этого режима;

- Должны быть реализованы элементы управления для запуска тестирования и построения графика;
- Для упрощения процесса изучения интерфейса необходима справка - подсказка, объясняющая значение того или иного элемента интерфейса. Полное руководство должно быть частью интерфейса, доступной в любой момент.

Основные принципы, которые должны быть использованы при разработке графического интерфейса:

- Интерфейс должен быть интуитивно понятным. Таким, чтобы пользователю не требовалось объяснять, как им пользоваться
- Интерфейс должен разрабатываться исходя из принципа наименьшего возможного количества действий со стороны пользователя

Концептуальное проектирование:

По результатам анализа сформированных целей и требований к интерфейсу была составлена концептуальная модель взаимодействия пользователя с продуктом через его интерфейс. После запуска программы пользователь видит поля для установки параметров тестирования, с кратким пояснением назначения конкретного поля и примерами значений полей, установленных по умолчанию. При недостатке информации пользователь может вызвать справку, описывающую элементы интерфейса программы, способы взаимодействия с ними и допустимые значения для ввода более подробно. Значения полей устанавливаются путем наведения курсора мыши на поле и клика левой кнопкой мыши, после чего производится ввод данных с клавиатуры. Для запуска тестирования пользователь нажимает соответствующую кнопку. В процессе тестирования пользователь видит информацию о ходе выполнения тестирования в виде подробного текстового отчета, выводимого поэтапно в соответствующий элемент интерфейса. По завершению тестирования пользователь может перейти к окну для построения графиков по результатам отчета кликнув по соответствующей вкладке на панели интерфейса. Для построения графика пользователь должен кликнуть соответствующую кнопку интерфейса, после чего в окне будет выведен результат в виде линейной диаграммы.

Реализация.

Следующим шагом после определения и анализа целей, требований и концепта проектируемого графического интерфейса является набор практических работ по воплощению этих требований и концептуальных моделей.

Прототипирование.

Целью прототипирования является предоставить макет интерфейса для оценки разработчиками, а также пользователями верности принятых решений по разработке. Этот этап особенно важен, так как на данном этапе корректировка каких-либо задач потребует гораздо меньших затрат, чем на последующих этапах. Прототипирование позволяет создавать макеты

интерфейсов разной степени достоверности: от набросков «на скорую руку» и бумажных прототипов до интерактивных макетов с использованием специальных программ. Польза от предварительного создания макета интерфейса выражается в таких преимуществах как:

- Предоставление возможности рассмотреть будущий интерфейс разрабатываемого ПО с реальным взаимодействием его элементов без программирования и реализации функционала, что существенно сокращает расходы по корректировке интерфейса (чем раньше разрабатываемый интерфейс будет соответствовать своему финальному виду, тем меньше средств будет потрачено на его реализацию);
- Наглядное демонстрирование будущих возможностей заказчику или конечному пользователю интерфейса;
- Тестирование «юзабилити» проектируемого графического пользовательского интерфейса;
- Тестирование новых, нестандартных решений;

Выбранный нами инструмент разработки Qt Creator предоставляет возможность довольно быстро разрабатывать прототипы интерфейса пользователя максимально близко похожие на конечную реализацию, что существенно облегчает процесс согласования интерфейса с заказчиком.

Интерфейс программы представляет собой окно с двумя вкладками:

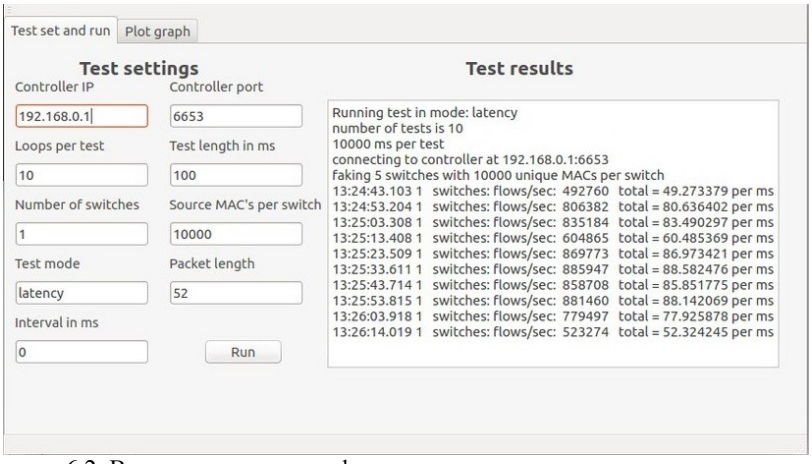
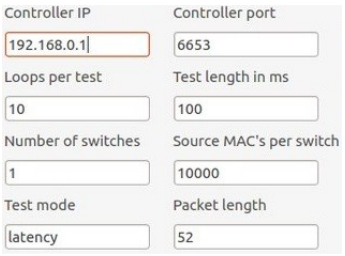
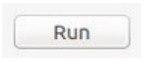


Рисунок.6.2. Вкладка с вводом конфигурации тестирования и результатами в виде текста

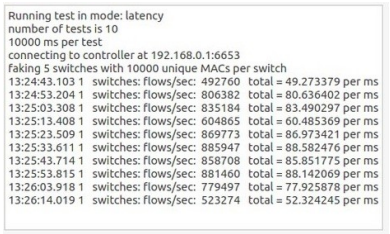
Элементы первой вкладки:



– поля для ввода параметров тестирования



– кнопка запуска тестирования



– вывод результатов тестирования

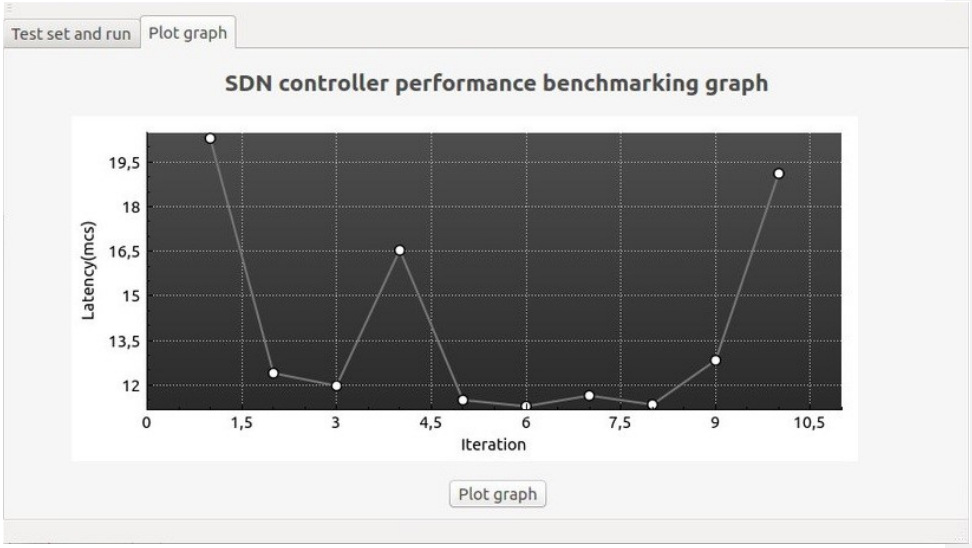
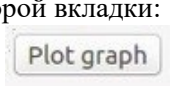
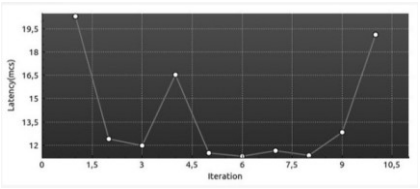


Рисунок.6.3. Вкладка с виджетом, отражающим результаты тестирования в виде графика

Элементы второй вкладки:



– кнопка запуска построения графика



– результат построения

«Юзабилити-тестирование».

Благодаря особенности среды разработки Qt Creator позволяющей нам быстро и с максимальной степенью достоверности составлять прототипы интерфейса, уже на ранней стадии проектирования мы можем провести юзабилити-тестирования с привлечением предполагаемых пользователей. На данном этапе было выявлено, что наиболее важным элементом интерфейса является виджет для построения графиков по результатам тестирования. Было принято решение убрать разделение на две вкладки и реализовать все элементы интерфейса в одном окне, перегруппировав их, уменьшив виджет для вывода текстового отчета, а основное пространство отвести под виджет для вывода линейных диаграмм. В связи с этим было принято решение расположить элементы интерфейса согласно правилу золотого сечения, по которому соотношение большей части к меньшей должно быть равным 1.62:

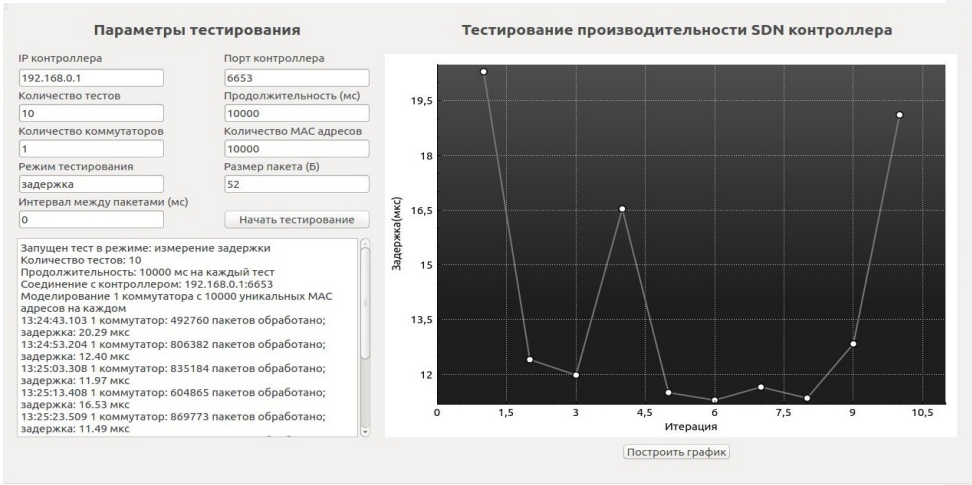


Рисунок.6.4. Корректировка интерфейса

Решение оказалось удачным, что и было подтверждено повторным тестированием с привлечением пользователей.

Конструирование.

Финальным шагом в разработке интерфейса является разработка классов на языке C++ для обработки сигналов, поступающих от пользователя и

управления взаимодействием графического интерфейса с основным кодом программы.

Для реализации формы главного окна был создан класс MainWindow, основанный на базовом классе QMainWindow:

```
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
  void setText(QString text);

private slots:
  void on_runTest_clicked();
  void on_plotButton_clicked();

private:
  Ui::MainWindow *ui;
};
```

Для удобства использования полей для ввода исходных данных была реализована функция автодополнения для текстовых виджетов QLineEdit проектируемого интерфейса. Для этого в конструкторе главной формы MainWindow были созданы несколько объектов класса QComleter, инициализированные списками QStringList с вариантами для автодополнения, и подключены к виджетам с помощью метода setCompleter:

```
QStringList wordList;
 QStringList wordList2;
 wordList << "throughput" << "latency";
 wordList2 << "6653" << "6633" << "6635";
 QCompleter *completer = new QCompleter(wordList, this);
 QCompleter *completer2 = new QCompleter(wordList2, this);
 completer->setCaseSensitivity(Qt::CaseInsensitive);
 ui->testMode->setCompleter(completer);
 ui->controllerPort->setCompleter(completer2);
```

Благодаря этой функции при попытке ввода исходных данных пользователю будут выданы подсказки по заполнению конкретной формы. Это позволит упростить работу с полями ввода и сделать использование интерфейса более интуитивно понятным, удобным и быстрым.

Также для обработки сигнала нажатия пользователем на кнопку запуска теста был реализован слот on_runTest_clicked():

```
    void MainWindow::on_runTest_clicked()
    {
      std::string fval = ui->controllerIP->text().toStdString();
      char *ctrlIP = new char [fval.size()+1];
      strcpy( ctrlIP, fval.c_str() );
     TestHandler* test = new TestHandler(this,ctrlIP,ui->controllerPort-
>text().toInt(),ui->switchNum->text().toInt(),ui->srcMacs-
>text().toInt(),ui->testLenght->text().toInt(),ui->testLoops-
>text().toInt(),MODE_THROUGHPUT,ui->pktLen->text().toInt(),           ui-
>msgInterval->text().toInt());
      test->manageTest();
    }
```

При поступлении связанного с ним сигнала, слот конвертирует значения полей, введенные пользователем в соответствующие типы данных. Так как выполнение тестирования будет производиться в отдельном классе, происходит инициализация объекта класса TestHandler, являющимся главным классом разрабатываемого программного обеспечения. После чего запускается функция manageTest этого класса, запускающая тестирование и управляющая ходом его выполнения. Подробнее эту функцию мы рассмотрим в соответствующих пунктах дипломной работы.

Для обработки сигнала нажатия пользователем на кнопку запуска построения линейной диаграммы отражающей результаты тестирования, был реализован слот void on_ plotButton_clicked(); При поступлении сигнала слоту, в зависимости от выбранного пользователем режима тестирования производится определение значений, на основании которых будет строится линейная диаграмма. Значения берутся из текстовых файлов, генерируемых в процессе выполнения тестирования. Рассчитывается количество и граничные значения для отображения на осях графика, а также заполняются два массива данных соответствующих координатам X и Y точек графика:

```
          if (ui->testMode->text()=="throughput"){
            nums.open("avg");
          } else { nums.open("latency");}
          for (int i=0;i<n;i++)
          {
            x[i] = i+1;
            nums>>y[i];
          }
          double max = 0;
          double min = y[0];
          for(int i=0; i<n; i++){
            if(y[i] > max)
```

```
            max = y[i];
          if(y[i] < min)
            min = y[i];
        }
```

Для построения линейных диаграмм использовалась свободно распространяемая библиотека для рисования графиков и визуализации данных QCustomPlot на QWidgets. Она предоставляет инструменты для построения различных типов диаграмм и их настройке. Для вывода графика требуется задать обязательные параметры, такие как значения осей координат, их подписи и массивы координат точек графика:

```
        ui->widget->clearGraphs();
        ui->widget->addGraph();
        ui->widget->graph(0)->setData(x, y);
        ui->widget->xAxis->setLabel("Iteration");
        if (ui->testMode->text()=="throughput"){
          ui->widget->yAxis->setLabel("Flows/sec");
        } else {ui->widget->yAxis->setLabel("Latency(mcs)");}
        ui->widget->xAxis->setRange(0, n+1);
        ui->widget->yAxis->setRange(min-(min/100), max+(max/100));
```

Аналогично, стандартными функциями библиотеки QCustomPlot были заданы стили отрисовки графика, осей координат, фона и надписей.

Для вывода отчета о ходе и результатах тестирования в интерфейс пользователя в виде текстовой информации был предусмотрен метод setText(QString text), он принимает значение типа QString и выводит его в соответствующий виджет интерфейса textBrowser.

```
    void MainWindow::setText(QString text)
    {
      ui->textBrowser->setPlainText(ui->textBrowser-
>toPlainText()+"\n"+text);
    }
```

6.3.2 Разработка классов, отвечающих за основную логику программы

Для реализации разработанного алгоритма работы программного обеспечения, было создано два класса: TestHandler и FakeSwitch. Первый класс является главным и управляет запуском тестирования с заданными параметрами и управляет ходом его выполнения, а также формирует отчеты по результатам выполнения и взаимодействует с интерфейсом пользователя. Класс FakeSwitch отвечает за моделирование OpenFlow коммутатора и его взаимодействие с контроллером программно-конфигурируемой сети.

Класс TestHandler

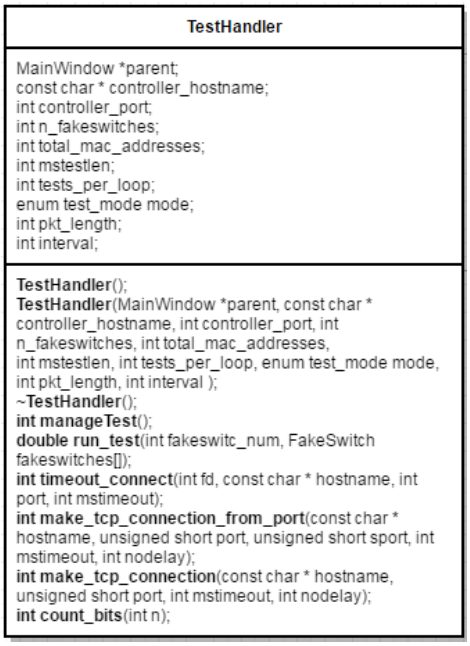


Рисунок 6.5. UML диаграмма класса TestHandler

Так как данный класс должен запускать тестирование с введенными пользователем параметрами и взаимодействовать с интерфейсом пользователя, выводя отчет о результатах, был реализован конструктор данного класса, принимающий значения, передаваемые классом интерфейса пользователя, а именно:

```
    TestHandler::TestHandler(MainWindow *parent, const char * controller_hostname, int controller_port, int n_fakeswitches, int total_mac_addresses,int mstestlen, int tests_per_loop, enum test_mode mode, int pkt_length, int interval)
    {
      this->parent = parent;
      this->controller_hostname = controller_hostname;
      this->controller_port = controller_port;
      this->n_fakeswitches = n_fakeswitches;
      this->total_mac_addresses = total_mac_addresses;
      this->mstestlen = mstestlen;
      this->tests_per_loop = tests_per_loop;
      this->mode = mode;
      this->pkt_length = pkt_length;
      this->interval = interval;
    }
```

MainWindow *parent – поле, используемое для хранение ссылки на экземпляр класса основной формы графического интерфейса, с помощью которого будет производиться вывод результатов тестирования.

const char* controller_hostname – массив символов для хранения адреса тестируемого OpenFlow контроллера, требуемого для установления соединения.

int controller_port – целочисленное поле для хранения порта получателя тестируемого контроллера, требуемого для установления соединения.

int n_fakeswitches – целочисленное поле для хранения требуемого количества моделируемых коммутаторов заданного пользователем.

int total_mac_adresses – целочисленное поле для хранения количества уникальных MAC адресов для каждого моделируемого коммутатора.

int mstestlen – целочисленное поле для хранения значения времени, отводимого на каждое отдельное испытание.

int tests_per_loop – целочисленное поле для хранения количества проводимых испытаний в процессе тестирования.

enum test_mode mode – поле для хранения выбранного пользователем режима тестирования контроллера.

int pkt_length – целочисленное поле для хранения размера пакетов PacketIn отправляемых в ходе тестирования моделируемым коммутатором.

int interval – целочисленное поле для хранения значения интервала между отправками пакетов.

Основным методом данного класса является функция int TestHandler::manageTest(), именно она обеспечивает запуск остальных функций и их взаимодействие с классами FakeSwitch и MainWindow. Сначала производится создание текстовых файлов для сохранения результатов тестирования. В зависимости от режима тестирования это будут файлы для хранения значений средней пропускной способности контроллера по проведенным тестам, либо значения средней задержки. Также производится инициализация массива объектов класса FakeSwitch для хранения моделируемых коммутаторов и последующего обращения к их функциям:

```
FakeSwitch fakeswitches[n_fakeswitches];
FILE *f;
if(mode==MODE_THROUGHPUT)
{
f = fopen("avgTrougtput","w");
fclose(f);
}else{
f = fopen("avgLatency","w");
fclose(f);
}
```

Затем происходит цикличная инициализация всех моделируемых коммутаторов, создание неблокирующих TCP-сокетов и их связывание для взаимодействия с контроллером OpenFlow. Это реализуется благодаря двум созданным функциям, которые будут подробнее рассмотрены далее:

```
    sock           =           make_tcp_connectio.n(controller_host.name,
controller_port,3000, mode!=MODE._THROUGHPUT );
        if(sock < 0 )
        {
          fprint.f(stderr, "make_tcp_conn.ection :: returned %d", sock);
          exit(1);
        }
        fflush(stderr);
        Fake.Switch        fake.switch_init(1+i,sock,BUFLEN,        mode,
total_mac_.addresses, pkt_.length);
        fakesw..itches[i] = fake.sw.itch_init;
```

Следующим шагом будет поочередный запуск в цикле заданного пользователем количества тестовых сценариев с участием заданного набора моделей коммутаторов, представленных объектами класса FakeSwitch. Возвращаемое значение функции на каждой итерации будет сохраняться в массив результатов тестирования, а также будет производиться вычисление минимального и максимального значения для последующего вывода в интерфейс пользователя:

```
    for( j = 0; j < tests_per_loop; j ++) {
          v = 1000.0 * run_test(i+1, fakeswitches);
          results[j] = v;
          if(j<warmup || j >= tests_per_loop-cooldown)
            continue;
          sum += v;
          if (v > max)
           max = v;
          if (v < min)
           min = v;
        }
```

Последним шагом в рассматриваемой функции выполняется вычисление средних значений, полученных данных в результате тестирования, их форматирование к соответствующему типу для удобного представления этих данных пользователю в виде тестовой информации в специальном окне графического интерфейса.

Для создания TCP-сокетов их связывания и установления соединения были реализованы две функции make_tcp_connection и timeout_connect. В первой функции происходит создание сокета потокового типа с протоколом TCP. Этот тип обеспечивает последовательный, надежный, ориентированный

на установление двусторонней связи поток байтов. Для создания сокетов служит библиотечная функция socket, которая создает конечную точку для коммуникаций и возвращает файловый дескриптор, ссылающийся на сокет, или -1 в случае ошибки. Затем устанавливается параметр сокета на уровне протокола TCP, а именно отключения алгоритма буферизации, который препятствует мгновенной отправке небольших сообщений, объединяя их до достижения максимального размера TCP пакета. Затем происходит связывание сокета с адресом и номером прослушиваемого порта вызовом библиотечной функции bind, после чего вызывается функция timeout_connect.

```
      s = socket(AF_INET,SOCK_STREAM,0);
      if(nodelay  &&  (setsockopt(s,  IPPROTO_TCP,  TCP_NODELAY,  &zero,
sizeof(zero)) < 0))
      local.sin_family=PF_INET;
      local.sin_addr.s_addr=INADDR_ANY;
      local.sin_port=htons(sport);
      err=bind(s,(struct sockaddr *)&local, sizeof(local));
      if(err)
      {
        perror("make_tcp_connection_from_port::bind");
        return -4;
      }
      err = timeout_connect(s,hostname,port, mstimeout);
```

Функция timeout_connect предназначена для создания соединения сокета с удаленным хостом (в нашем случае - SDN контроллером). Для этого сначала полученный от пользователя адрес и порт контроллера преобразуется в структуру адресов сокета удаленного хоста, для дальнейшего использования в функции connect. Функция connect устанавливает соединение с удаленным хостом, используя полученные ранее файловый дескриптор, указатель на адрес сокета и длину адреса сокета. После данных операций сокет готов к приему и передаче данных. Помимо этого, так как для выполнения задачи требуется использование неблокирующих сокетов, для указания режима работы используется функция fcntl с параметром O_NONBLOCK, благодаря этому любой вызов функции read() для сокета будет возвращать управление сразу же.

```
      err = getaddrinfo(hostname,sport,&hints,&res);
      if(fcntl(fd, F_SETFL, flags | O_NONBLOCK) < 0) {
        fprintf(stderr, "timeout_connect: unable to put the socket in
non-blocking mode\n");
        freeaddrinfo(res);
        return -1;
      }
        if(::connect(fd, res->ai_addr, res->ai_addrlen) < 0)
        {
          if((errno != EWOULDBLOCK) && (errno != EINPROGRESS))
```

```
          {
            fprintf(stderr, "timeout_connect: error connecting: %d\n",
errno);
            freeaddrinfo(res);
            return -1;
          }
        }
```

Следующая рассматриваемая функция run_test осуществляет управление ходом тестирования. Она запускает тестирование путем цикличного вызова функций fakeswitch_set_pollfd, fakeswitch_handle_io и системного вызова poll. Благодаря первой функции формируется массив структур типа pollfd, в который вносятся файловые дескрипторы и битовые маски событий, важных для приложения. Этот массив используется системным вызовом poll который блокирует процесс, пока какой-либо из файловых дескрипторов не станет доступным для чтения или записи с таймаутом в 100мс. Затем, поочередно для каждого моделируемого коммутатора запускается функция fakeswitch_handle_io предназначенная для обработки событий чтения либо записи для файловых дескрипторов, связанных с сокетами. Также в цикле происходит отслеживание времени выполнения текущего теста. По прошествии отведенного времени, заданного пользователем, выполнение теста завершается. Производится запись полученных данных в ходе его выполнения в текстовый файл и вывод в графический интерфейс пользователя.

```
    while(1)
      {
        gettimeofday(&now, NULL);
        timersub(&now, &then, &diff);
        if( (1000* diff.tv_sec + (float)diff.tv_usec/1000)> total_wait)
          break;
        for(i = 0; i< fakeswitch_num; i++)
          fakeswitches[i].fakeswitch_set_pollfd(&pollfds[i]);

        poll(pollfds, fakeswitch_num, 1000);    // block until something
is ready or 100ms passes

        for(i = 0; i< fakeswitch_num; i++)

fakeswitches[i].fakeswitch_handle_io(&pollfds[i],pkt_length,interval,last
Write);
      }
```

Класс FakeSwitch.

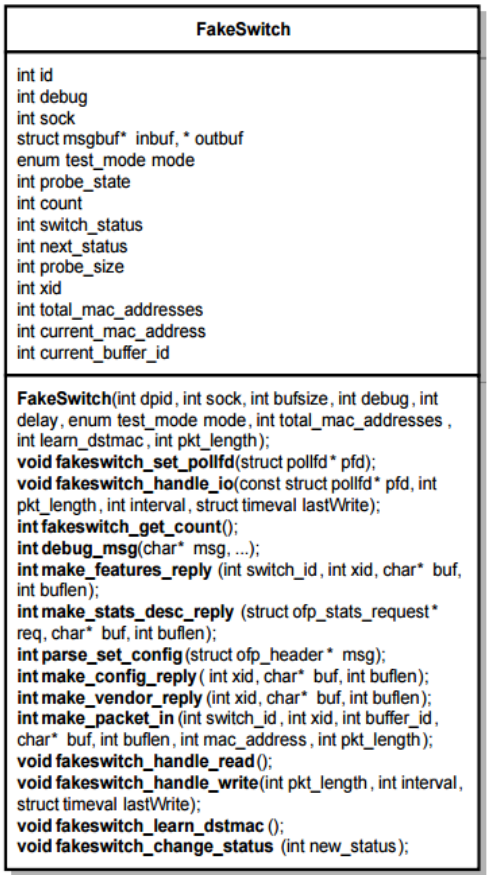


Рисунок.6.6. UML диаграмма класса FakeSwitch

Так как данный класс отвечает за моделирование коммутатора, поддерживающего протокол OpenFlow и его взаимодействие с контроллером программно-конфигурируемой сети, был реализован конструктор данного класса, который принимает значения, передаваемые функцией run_test основного класса TestHandler, заполняет ими поля экземпляра класса, производит инициализацию буферов для входящих и исходящих сообщений, и записывает в исходящий буфер сообщение формата OFPT_HELLO:

```
    FakeSwitch::FakeSwitch(int dpid, int sock, int bufsize, enum
test_mode mode, int total_mac_addresses, int pkt_length)
    {
      char buf[BUFLEN];
      struct ofp_header ofph;
      this->sock = sock;
      this->id = dpid;
      this->inbuf = msgbuf_new(bufsize);
```

```
      this->outbuf = msgbuf_new(bufsize);
      this->probe_state = 0;
      this->mode = mode;
      this->probe_size   =   make_packet_in(id,   0,   0,   buf,   BUFLEN,
current_mac_address++, pkt_length);
      this->count = 0;
      this->switch_status = START;
      this->total_mac_addresses = total_mac_addresses;
      this->current_mac_address = 0;
      this->xid = 1;
      this->learn_dstmac = 1;
      this->current_buffer_id = 1;
      ofph.version = OFP_VERSION;
      ofph.type = OFPT_HELLO;
      ofph.length = htons(sizeof(ofph));
      ofph.xid  = htonl(1);
      msgbuf_push(outbuf,(char * ) &ofph, sizeof(ofph));
      debug_msg(" sent hello");
    }
```

int id – поле, используемое для хранения номера идентификатора коммутатора.

int sock – поле, используемое для хранения файлового дескриптора, ссылающегося на сокет.

struct msgbuf * inbuf, * outbuf – поля, используемое для буферизации входящих и исходящих сообщений OpenFlow.

enum test_mode mode - поле, используемое для хранения выбранного режима тестирования.

int probe_state – поле, которое отображает, есть ли пакет, ожидающий отправки в режиме тестирования задержки или количество пакетов ожидающих отправки в режиме тестирования пропускной способности.

int probe_size – поле, используемое для хранения размера пакета.

int count – поле, используемое для хранения данных о количестве принятых ответов от SDN контроллера

int switch_status – поле, используемое для хранения текущего состояния коммутатора. Для состояний коммутатора был обьявлен набор целочисленных констант:

```
    enum handshake_status {
      START,
      LEARN_DSTMAC,
      READY_TO_SEND,
      WAITING
    };
```

int xid – поле для хранения идентификатора OpenFlow транзакции.

int total_mac_addresses, current_mac_address – поля для хранения данных о заданном количестве уникальных MAC адресов для каждого коммутатора и текущем MAC адресе, который будет использоваться в пакете PacketIn

int current_buffer_id – поле для хранения идентификатора пакета PacketIn отправляемого моделируемым коммутатором, используемого для его отслеживания.

Для реализации взаимодействия созданных моделей OpenFlow коммутаторов с удаленным контроллером был разработан ряд функций, отвечающих за прием и обработку запросов от контроллера, а также формирование ответных пакетов и их последующую отправку коммутатором.

Функция fakeswitch_set_pollfd устанавливает флаги ожидаемых событий и файловый дескриптор в массив структур типа pollfd для последующего отслеживания этих событий на сокете с помощью системного вызова poll():

```
    void FakeSwitch::fakeswitch_set_pollfd(struct pollfd *pfd)
    {
      pfd->events = POLLIN|POLLOUT;
      pfd->fd = sock;
    }
```

Функция fakeswitch_handle_io предназначена для обработки произошедших событий. В зависимости от установленного флага в поле revents, передаваемой ей структуры типа pollfd, происходит вызов функций для взаимодействия с коммутатором, а именно осуществления процедур чтения либо записи данных.

```
    void FakeSwitch::fakeswitch_handle_io(const struct pollfd *pfd, int
pkt_length, int interval, struct timeval lastWrite)
    {
      if(pfd->revents & POLLIN)
        fakeswitch_handle_read();
      if(pfd->revents & POLLOUT)
        fakeswitch_handle_write(pkt_length, interval, lastWrite);
    }
```

Функция fakeswitch_handle_write предназначена для осуществления операций по созданию и отправке сообщений на удаленный SDN контроллер. В зависимости от текущего состояния коммутатора, значение которого хранится в поле switch_status, происходит формирование пакетов и занесение их в буфер для отправки, после чего с помощью библиотечной функции msgbuf_write производится отправка сформированных пакетов SDN контроллеру Если коммутатор находится в состоянии готовности к отправке исходящих сообщений, то, в зависимости от выбранного режима тестирования счетчик количества требуемых сообщений для отправки устанавливается либо 1 (в режиме измерения задержки) либо его значение рассчитывается путем

вычисления количества сообщений требуемых для полного заполнения буфера отправки(в режиме измерения пропускной способности). После этого производится цикличное формирование пакетов формата PaketIn, с записанным в соответствующее поле уникальным MAC-адресом и идентификатором пакета, и занесение их в буфер отправки сообщений.

```
        for (i = 0; i < send_count; i++)
        {
          count = make_packet_in(id, xid++, current_buffer_id, buf,
BUFLEN, current_mac_address, pkt_length);
          current_mac_address = ( current_mac_address + 1 ) %
total_mac_addresses;
          current_buffer_id = ( current_buffer_id + 1 ) % NUM_BUFFER_IDS;
          msgbuf_push(outbuf, buf, count);
        }
```

Следующим шагом производится проверка наличия в буфере сообщений, ожидающих отправки. Если буфер не пуст, а тестирование запущено в режиме тестирования задержки SDN контроллера и пользователем при запуске было указано значение интервала между отправляемыми пакетами, то сначала производится обеспечение этого интервала, с помощью функции usleep, которая приостанавливает работу потока, в котором она была вызвана, на указанное в время в микросекундах.

По истечению таймера, либо в случае, если выбран режим тестирования пропускной способности, производится отправка буферизованных пакетов.

```
    if( msgbuf_count_buffered(outbuf) > 0)
        if((mode==MODE_LATENCY)&&(interval!=0))
        {
          struct timeval now;
          gettimeofday(&now, NULL);
          int timer=(1000000* now.tv_sec + now.tv_usec)-(1000000*
lastWrite.tv_sec + lastWrite.tv_usec+interval);
          if (timer>0)
            usleep(timer);
        }
        msgbuf_write(outbuf, sock, 0);
```

Для обработки сообщений, отправляемых контроллером моделируемому коммутатору, была реализована функция fakeswitch_handle_read. При вызове данной функции происходит цикличное извлечение пакетов из буфера входящих сообщений, и в зависимости от типа сообщений производится их обработка и формирование ответа.

В случае если было получено сообщение типа OFPT_HELLO оно игнорируется, так как сообщения данного типа используются только при установлении соединения коммутатор-контроллер для сверки совпадения

поддерживаемой версии протокола OpenFlow, а моделируемый нами коммутатор уже отправил сообщение данного типа при инициализации, так что повторная отправка не требуется.

После того, как TCP соединение между коммутатором и контроллером было установлено, первым делом идет определение характеристик коммутатора. По транспортному каналу контроллер отправляет пакет формата FeatureReq, на который коммутатор должен ответить пакетом FeatureRes содержащий в себе перечисление параметров коммутатора, таких как размер буфера для очереди пакетов PacketIn, количество таблиц, возможности и действия, поддерживаемые коммутатором и последовательность доступных портов. Для обработки данного события служит следующий участок кода:

```
        case OFPT_FEATURES_REQUEST:
          count = make_features_reply(id, ofph->xid, buf, BUFLEN);
          msgbuf_push(outbuf, buf, count);
          fakeswitch_change_status(READY_TO_SEND);
          break;
```

Для формирования ответных пакетов с параметрами коммутатора FeatureRes была создана отдельная функция make_features_reply. Данные для заполнения пакеты были взяты из пакета, отправленного OpenFlow коммутатором, и пойманного с помощью программы, анализирующей трафик – WireShark.

```
    struct ofp_switch_features * features;
    const char fake[] =
    {
   //тут 16чным кодом, записаны данные параметров компилятора
    };
    assert(buflen> sizeof(fake));
    memcpy(buf, fake, sizeof(fake));
    features = (struct ofp_switch_features *) buf;
    features->header.version = OFP_VERSION;
    features->header.xid = xid;
    features->datapath_id = htonll(id);
    return sizeof(fake);
```

При поступлении OpenFlow сообщений типа FlowMod или PacketOut происходит увеличение счетчика обработанных запросов, так как пакеты FlowMod и PacketOut являются результатами обработки контроллером пакетов PacketIn принятых от моделируемого с помощью разрабатываемого программного обеспечения SDN коммутатора. Они служат для сообщения коммутатору правил обработки пакета и обновления значений таблицы потоков.

```
          case OFPT_PACKET_OUT:
            po = (struct ofp_packet_out *) ofph;
            if    (    switch_status    ==    READY_TO_SEND    &&    !
packet_out_is_lldp(po)) {
              this->count++;
              probe_state--;
            }
            break;
```

На протяжении всего времени соединения контроллер и коммутатор время от времени обмениваются сообщениями EchoReq и EchoRes, для обмена информацией о задержках, пропускной способности и состоянии соединения. Отсутствие ответа EchoRes соответсвует разрыву соединения с контроллером. Для обработки данной ситуации был написан следующий участок кода:

```
       case OFPT_ECHO_REQUEST:
            echo.version= OFP_VERSION;
            echo.length = htons(sizeof(echo));
            echo.type  = OFPT_ECHO_REPLY;
            echo.xid = ofph->xid;
            msgbuf_push(outbuf,(char *) &echo, sizeof(echo));
            break;
```

OpenFlow контроллер отправляет пакет BarrierReq для запроса коммутатору, чтобы он обработал все сообщения, отправленные до BarrierReq, прежде чем обрабатывать запросы, пришедшие после BarrierReq. Это позволяет удостовериться в том, что коммутатор обрабатвает и отправляет все сообщения по завершенным операциям до обработки новых запросов. Когда OpenFlow коммутатор получает данный запрос, он ставит в очередь все последующие входящие сообщения, кроме Echo, пока обработка всех приоритетных сообщений не будет завершена. Когда обработка сообщений будет завершена, коммутатор отправляет контроллеру ответ BarrierRes и только после этого приступает к обработке буферизованных за это время сообщений. Для формирования ответа контроллеру типа BarrierRes, содержащего xid полученного запроса, был написан следующий участок кода:

```
   case OFPT_BARRIER_REQUEST:
            barrier.version = OFP_VERSION;
            barrier.length = htons(sizeof(barrier));
            barrier.type = OFPT_BARRIER_REPLY;
            barrier.xid = ofph->xid;
            msgbuf_push(outbuf,(char *) &barrier, sizeof(barrier));
            break;
```

Основной массой пакетов, которые приходится обрабатывать контроллеру во время его функционирования, являются пакеты типа PacketIn.

Помимо этого, важным свойством данных пакетов является то, что контроллер обязан после его обработки отправить ответ коммутатору, в виде пакета типа FlowMod или PacketOut. Именно поэтому для тестирования контроллера был выбран данный тип пакетов. Для формирования пакетов типа PacketIn была разработана функция make_packet_in, которая создает пакет OpenFlow, который инкапсулируется в TCP, IPv4и Ethernet. В зависимости от указанного пользователем размера пакета рассчитываются и подставляются соответствующее значение полей в заголовках пакета. Также в заголовке Ethernet указывается сгенерированный MAC адрес хоста отправителя

```
    assert(buflen> sizeof(fake));
    memcpy(buf, fake, sizeof(fake));
    pi = (struct ofp_packet_in *) buf;
    pi->header.version = OFP_VERSION;
    pi->header.xid = htonl(xid);
    pi->buffer_id = htonl(buffer_id);
    eth = (struct ether_header * ) pi->data;
    memcpy(&eth->ether_shost[1], &mac_address, sizeof(mac_address));
    eth->ether_dhost[5] = switch_id;
    eth->ether_shost[5] = switch_id;
    return sizeof(fake);
```

Для определения статуса контроллера в заголовке класса был объявлен набор именованных целых констант. Для смены статусов моделируемого контроллера предусмотрена отдельная функция fakeswitch_change_status, принимаемым значением которой является одна из объявленных констант. Если статус контроллера изменился READY_TO_SEND счетчики принятых и отправленных пакетов устанавливаются в ноль.

```
  void FakeSwitch::fakeswitch_change_status (int new_status) {
    switch_status = new_status;
    if(new_status == READY_TO_SEND) {
      count = 0;
      probe_state = 0;
    }
  }
```

По окончанию времени, отведенного на выполнение теста вызывается функция подсчета количества обработанных контроллером пакетов и возврат полученного значения в вызывающую функцию. Также в этой функции производится очищение буфера входящих сообщений и обнуление счётчиков.

```
  int FakeSwitch::fakeswitch_get_count()
  {
    int ret = this->count;
    int i;
```

```
    int msglen;
    struct ofp_header * ofph;
    this->count = 0;
    this->probe_state = 0;
    while( (i = msgbuf_read(inbuf,sock)) > 0) {
      while(i > 0) {
        ofph = (struct ofp_header*)msgbuf_peek(inbuf);
        msglen = ntohs(ofph->length);
        msgbuf_pull(inbuf, NULL, ntohs(ofph->length));
        i -= msglen;
      }
    }
    return ret;
  }
```

Ключевые слова

Виды тестирования, существующие реализации программного обеспечения, тестирование SDN, контроллеры, разработка программного обеспечения.

Контрольные вопросы

1. Назовите и опишите основные виды тестирования SDN-котроллеров.
2. Перечислите существующие реализации программного обеспечения для тестирования SDN-контроллеров
3. С помощью какого ЯП и соответствующей библиотеки на данном языке реализован фреймворк Hcprobe.
4. Какие виды тестирования могут быть проведены с помощью фреймворка Hcprobe?
5. Что включает в себя стандартный тест, написанный на EDSL для Hcprobe?
6. Какие тестовые сценарии имеет фреймворк Hcprobe по умолчанию?
7. С помощью какого ЯП был разработан такой инструмент тестирования, как Cbench?
8. Каким образом построена работа программы cbench?
9. Дайте определение Ctltest. Для тестирования каких контроллеров оно предназначено по умолчанию?
10. Каким образом построена работа с Ctltest?
11. Какие зависимости возможно получить, при тестировании с помощью Ctltest?
12. Перечислите основные этапы разработки графического интерфейса для программы тестирования. Сформулируйте задачу этапа проектирования.
13. Назовите основные принципы, которые должны быть использованы при разработке графического интерфейса.

14. В чем заключается цель прототипирования? Сформулируйте цель «юзабилити- тестирования»

15. Сформулируйте основные задачи, которые ожидается решить с помощью таких технологий как: SDN и NVF.

16. Что может способствовать внедрению технологий SDN и NVF в России?

17. Сформулируйте две крупные проблемы, которые стоят перед вендорами, операторами и дистрибьюторами оборудования в сфере SDN и NVF?

18. Как может повлиять на бизнес-процессы переход на технологии SDN и NVF?

Литература к главе 6

1. Ctltest [Электронный ресурс] – Режим доступа: http://arccn.github.io/ctltest/ (дата обращения: 06.05.2016).
2. Cbench (controller benchmarker) [Электронный ресурс] – Режим доступа: http://archive.openflow.org/wk/index.php/Oflops/ (дата обращения: 06.05.2016).
3. Описание утилиты HCprobe [Электронный ресурс] – Режим доступа: arccn.github.io/hcprobe/ (дата обращения: 06.05.2016).

7. Перспективы SDN в Российской Федерации

По состоянию на 2018 год, SDN, как в отдельности, так и в совокупности с NFV имеют как практические коммерческие реализации, так и пилотные проекты. Ряд центров обработки данных и коммерческих распределенных корпоративных сетей построены или осуществляют переход к технологиям SDN/NFV. В том числе, за время развития технологий сформировался международный рынок решений технологий SDN/NFV, существуют как проприетарные решения именитых вендоров, так и уже сформировавшие доверие со стороны пользователей - решения, основанные на открытом исходном коде. Однако, стоит отметить, что операторы связи не торопятся переходить к масштабному переходу на SDN/NFV технологии в связи с ожиданием устойчивых решений, выполняющих требования их реализации (устойчивость, безопасность и т.п.) для крупномасштабных и высоконагруженных архитектур. При этом наблюдается мягкая интеграция конкретных решений из SDN/NFV в существующие операторские сети. К примеру, на данный момент внедрение NFV в качестве vCPE, и формирование на основе данного решения нового ряда услуг. Однако, недостаточная информированность о технологиях, отсутствие комплексного понимания достигаемого синергетического эффекта при совместном внедрении SDN/NFV, являются главными барьерами для широкомасштабного внедрения этих технологий.

Как уже было отмечено в первой главе книги, существует множество причин, побуждающих задуматься о новых технологиях в строении сетей, одной из них является – неумолимый рост трафика в расчете на одного абонента сети и его качественное изменение. Основным фактором роста станет видео-трафик. К 2014 году его доля превысила 91% в глобальном пользовательском интернет трафике. К 2014 году совокупный объем видео трафика 3D и HD составил 42% от общего объема пользовательского видео трафика в Интернете. Согласно официальной статистике Cisco Systems прогнозируется, что объем глобального трафика сети Интернет и IP-сети WAN достигнут 2,3 ЗБ в год к 2020 году, количество уже в 2015 году глобального трафика центров обработки данных по оценкам, составляет 4,7 ЗБ и к 2020 году ожидается рост 15,3 ЗБ в год. Это увеличение представляет собой 27% CAGR.

Эти цифры говорят о том, что пропускная способность современных каналов связи при существующих методах и средствах управления трафиком в сетях близка к исчерпанию. Существующие темпы роста пропускной способности сети будут не в состоянии удовлетворять растущие потребности пользователей. Рост количества и разнообразия мобильных устройств, развитие различных технологий беспроводной связи (WiFi, 3G, WIMAX, LTE), рост количества мобильных сервисов привели к тому, что сегодня количество пользователей компьютерных сетей на основе беспроводных технологий

превышает число пользователей с фиксированной связью, а число мобильных терминалов, приходящихся на одного пользователя в развитых странах, достигает трех. Однако мобильные беспроводные сети сегодня сталкиваются с двумя противоречивыми тенденциями. Увеличение вычислительной мощности мобильных терминалов влечет за собой увеличение вычислительной емкости приложений, работающих на них. Это в свою очередь ведет к увеличению требований к пропускной способности каналов мобильной связи. На сегодня объем мобильного трафика растет в геометрической прогрессии, а виды трафика становятся все более разнообразным.

Одновременно с ростом количественных показателей нагрузки на компьютерные сети повысилась сложность управления сетью, значительно расширился перечень решаемых задач, их значимость и критичность, повысились требования к безопасности и надежности сетей. Наблюдается необходимость внедрения и развития новых, мультимедийных услуг. При этом, с учетом анонсированных возможностей сетей пятого поколения широкой публике, наблюдается активность как различных разработчиков решений, так и операторов с целью скорейшего внедрения инновационных решений.

Современные сети строятся с использованием коммутаторов, маршрутизаторов и других устройств, которые стали чрезвычайно сложными, поскольку они осуществляют все большее число сложных распределенных протоколов, стандартизированных IETF, (на сегодня число активно используемых протоколов и их версий превысило 600) и используют закрытые и проприетарные интерфейсы внутри. В таких условиях исследователи не могут проводить необходимые им эксперименты в работающей сети, операторы не могут быстро вводить новые сервисы для своих пользователей, производители сетевого оборудования не могут внедрять инновации настолько быстро, чтобы удовлетворить требования заказчиков. Поддержка и управление сложной сетевой инфраструктурой сегодня представляет больше искусство, чем инженерию. Рост сетевых атак, вирусов и других сетевых угроз говорит о том, что вопросы безопасности до сих пор не имеют надежных решений. Международным сообществом на сегодня осознано, что компьютерные и телекоммуникационные сети являются объектом национальной безопасности.

Рост количества и разнородности контента, развитие сервисов и масштабов их охвата привели к изменению парадигмы организации вычислений в Обществе: на место клиент-серверной организации вычислений пришли Центры Обработки Данных (ЦОД) и облачные вычисления, файловые системы и базы данных трансформировались в Сети Хранения Данных(СХД).

Развитие микропроцессорной техники и телекоммуникаций (закон Мура, закон Гилдера) способствовали росту среднего числа чипов на человека во встроенных блоках управления бытовыми приборами, индивидуальными транспортными средствами и так далее, и сейчас составляет около 40 чипов на человека. Развитие телекоммуникационного рынка способствовали появлению новых сетевых устройств.

По последним проведенным опросам, о перспективах внедрения технологий программно-конфигурируемых сетей SDN и технологий виртуализации NFV, проведенным среди российских телекоммуникационных операторов и провайдеров, вендоров и дистрибьюторов ожидается приход данных технологий на российский телекоммуникационный рынок в 2016 – 2017 году. Основным сдерживающем фактором выступает экономическая неопределенность и недостаточная информируемость о технологиях. Около 80% участников ожидают внедрение SDN и NFV в 2016–2017 гг., а 30% опрошенных операторов/провайдеров обещают внедрение не раньше ближайших 5 лет, то есть не раньше 2020 г.

Российские операторы, провайдеры, вендоры и дистрибьюторы хотят развивать технологии SDN и NFV с целью экономии средств при развертывании и эксплуатации сетей и ставят своим главным приоритетом «сокращение расходов». Такие направления, как инновации и новые возможности остаются менее приоритетными. Это может быть обусловлено поведением рынка телекоммуникаций и общей экономической ситуацией в Российской Федерации.

Стоит учесть, что SDN и NFV в России во многом зависит и от заинтересованности государственных структур, в том числе от запуска специальных проектов. В начале 2014 г. Правительство России внесло технологии SDN и NFV в перечень приоритетных научных задач, для решения которых требуется задействовать центры коллективного пользования научным оборудованием. Стоит отметить, что SDN и NFV стали единственными технологиями из блока ИКТ, вошедшими в данный перечень.

Что касается интереса государственных структур, то с 2013 г. Министерство образования и науки профинансировало проведение двух профильных НИОКР. Это создание и развитие отечественной платформы с открытым программным кодом для управления SDN, а также исследование и разработка средств управления ИТ-инфраструктурой в корпоративных и ведомственных компьютерных сетях на основе SDN и NFV. Помимо этого, Минобрнауки профинансировало проведение первой международной конференции по данной теме «Управление и виртуализация в современных сетях: SDN&NFV», которая прошла в октябре 2014 г. с участием экспертов из США, ЕС, Японии и России. В 2013 г. по инициативе ЦПИКС был так же создан Консорциум «SDN в научно-образовательной среде», в котором на сегодня состоит 14 российских вузов и исследовательских лабораторий.

Производители российского телекоммуникационного оборудования, операторы связи, дистрибьюторы и другие сталкиваются с рядом проблем при внедрении в свои сети технологий SDN и NFV. В две более крупные проблемы стоит внести:

- Отсутствие четкого понимания своих потребностей;
- Отсутствие единых комплексных стандартов в данной области.

В России мало компаний, готовых экспериментировать и желающих быть первопроходцами. «У российских операторов нет четкого видения и понимания своих потребностей. Причины в отсутствии сформированной службы R&D, подобной тем, какие есть у западных операторов связи. Как следствие, большинство отечественных операторов связи заточены на закупку уже готовых решений, а не осознание своих потребностей», – так комментируют ситуацию на российском рынке некоторые из экспертов в данной области.

Основное препятствие – это наличие большого парка оборудования, которое не отслужило свой срок, считает Илья Астахов. Иван Иашагашвили отмечает, что лишь немногие существующие инфраструктуры российских компаний готовы к внедрению SDN: «Для реализации технологии большинству операторов придется почти полностью заменить ИТ-инфраструктуру, что означает большие издержки». Еще одним из ключевых сдерживающих факторов, по мнению эксперта, является недостаточная информированность участников рынка об этих технологиях, а также отсутствие отраслевых стандартов и правил регулирования.

Технологии компьютерных сетей нового поколения могут поменять бизнес-модели тесно-связанных с ними предприятий. SDN и NFV смещают акцент по переходу в область программного обеспечения от аппаратной, так как разработка программных продуктов традиционно считается более сильной стороной, чем создание новых аппаратных платформ, что в последнее время так же развивается благодаря проведению политики импортозамещения.

Для российского рынка это начало создания нового сегмента – программного обеспечения для управления сетями и сетевыми сервисами. У отечественных сервис-провайдеров появится возможность покупать ПО у локальных производителей и продавать виртуализированные сервисы вместо услуг физической сети. В результате в ближайшие 3–5 лет в России в массовом порядке появятся новые операторы виртуальных сетей, уверен Руслан Смелянский.

Уже есть первые подвижки, к примеру, на рынок вышли несколько профильных отечественных стартапов. Компания Wimark Systems специализируется на централизованном управлении Wi-Fi-сетями, которое интегрировано с SDN. Компания NFWare является владельцем платформы для развертывания высокопроизводительных виртуальных сетевых сервисов и автоматического управления ими из единого центра, решение реализовано на базе NFV-технологий и другие. В конце 2014 г. было объявлено о разработке первого российского SDN-контроллера Runos, который был протестирован «Ростелекомом» и «Воентелекомом». Разработкой SDN-контроллера и SDN-коммутатора на базе импортных микросхем занимается еще одна российская компания – Zelax. О выпуске своей SDN/NFV-платформы также объявила компания Brain4Net.

Владыко Андрей Геннадьевич
Мутханна Аммар Салех Али
Киричек Руслан Валентинович
Волков Артем Николаевич

ПРОГРАММНО-КОНФИГУРИРУЕМЫЕ СЕТИ

Учебное пособие

Редактор …
Компьютерная верстка …

План издания 2018 г., п. 179

Подписано к печати 24.12.2018
Объем … усл.-печ. л. Тираж … экз. Заказ …
Редакционно-издательский отдел СПбГУТ
191186 СПб., наб. р. Мойки, 61
Отпечатано в СПбГУТ