ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В.ЛОМОНОСОВА»

УДК 004.4:004.7
№ госрегистрации 01201356240
Инв. №

УТВЕРЖДАЮ
Проректор МГУ имени
М.В.Ломоносова
_____ В.Е.Подольский
«10» июня 2013г.

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Создание прототипа отечественной ПКС платформы управления сетевыми ресурсами и потоками с помощью сетевой операционной системы (СОС) на основе анализа и оценки существующих сетевых операционных систем для ПКС сетей и выбора одной из них для последующего развития по критериям производительности, масштабируемости, надежности, безопасности

ВЫБОР НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ ПОСТАВЛЕННЫХ ПЕРЕД НИР ЗАДАЧ

(промежуточный)

2013-1.4-14-514-0093-040

Руководитель НИР, д.ф.-м.н.,
профессор _____ Р.Л.Смелянский
подпись, дата

Москва 2013

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы	подпись, дата	Р.Л. Смелянский (введение, заключение)
Исполнители темы	подпись, дата	А.Г. Бахмуров (разделы 1,4)
	подпись, дата	В.В. Балашов (разделы 2,5)
	подпись, дата	Д.Ю. Волканов (разделы 3,11)
	подпись, дата	М.В. Чистолинов (разделы 6,7)
	подпись, дата	Д.А. Зорин (раздел 7)
	подпись, дата	П.Е. Шестов (раздел 8)
	подпись, дата	А.В. Сапожников (раздел 9)
	подпись, дата	А.П. Капитонова (раздел 10,11)
Нормоконтролёр	подпись, дата	С.А. Косачева

РЕФЕРАТ

Страниц 252. Рисунков 54. Таблиц 13. Источников 62.

Ключевые слова: ПРОГРАММНО-КОНФИГУРИРУЕМЫЕ СЕТИ (ПКС), СТАНДАРТ OPENFLOW, СЕТЕВАЯ ОПЕРАЦИОННАЯ СИСТЕМА, КАЧЕСТВО СЕРВИСА (QOS), УПРАВЛЕНИЕ СЕТЕВЫМИ РЕСУРСАМИ

Объект исследований: Средства управления программно-конфигурируемыми сетями.

Цель исследования на этапе 1 НИР: Разработка научно-технического задела в области создания программных средств с открытым кодом для управления сетевыми ресурсами и потоками данных на основе подхода программно-конфигурируемых сетей (далее – ПКС) с оценкой возможности применения ПКС для повышения эффективности управления компьютерными сетями и потоками данных.

Основные результаты исследования на этапе 1:

1) проведен анализ средств формирования OpenFlow таблиц для OpenFlow сетевых коммутаторов, возможности формирования сетевых срезов с помощью таких коммутаторов;

2) проанализированы существующие подходы к виртуализации сетей;

3) проведены исследование, обоснование и выбор методов обеспечения QoS для ПКС;

4) разработаны методы и алгоритмы управления сетевой отечественной ПКС платформы управления сетевыми ресурсами и потоками с помощью сетевой операционной системы;

5) сформирован набор требований для разработки прототипа отечественной платформы управления ПКС на втором этапе НИР.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В работе используются следующие русскоязычные обозначения и сокращения:

Обозначения	Расшифровка	Аналог (англ.)
ГПС	Глобальное представление сети	
ЗУ	Задачи управления	
КЗУ	Косвенные задачи управления	
КС	Компьютерная сеть	Network
ОС	Операционная система	Operating System (OS)
ПКС	Программно-конфигурируемые сети	Software-defined network (SDN)
ПО	Программное обеспечение	ПО
СОС	Сетевая операционная система	Network Operating System (NOS)
УПД	Уровень управления данными	Data plane
УУ	Уровень управления	Control plane
ЦОД	Центр обработки данных	Data center

В работе используются следующие англоязычные обозначения и сокращения:

Обозначения	Расшифровка	Аналог (русск.)
ACL	Access Control List	

Обозначения	Расшифровка	Аналог (русск.)
API	Application Programming Interface	
ATM	Asynchronous Transfer Mode	
ARP	Address Resolution Protocol	
BGP	Border Gateway Protocol	
CLI	Command-Line Interface	
CPU	Central Processing Unit	ЦПУ – Центральное процессорное устройство
DHCP	Dynamic Host Configuration Protocol	
DNS	Domain Name System	
DOVE	Distributed Overlay Virtual Ethernet	
EIGRP	Enhanced Interior Gateway Routing Protocol	
ERP	Enterprise Resource Planning	
FTP	File Transfer Protocol	
FR	Frame Relay	
GPL	GNU Public License	
GRE	Generic Route Encapsulation	
HTTP	HyperText Transfer Protocol	
HP NMC	Hewlett-Packard Network Management Center	

Обозначения	Расшифровка	Аналог (русск.)
ICMP	Internet Control Message Protocol	
IDS	Intrusion Detection System	
IP	Internet Protocol	
IPFIX	IP Flow Information Export	
IPS	Intrusion Prevention System	
IS-IS	Intermediate System to Intermediate System	
ITIL	Information Technology Infrastructure Library	
LDAP	Lightweight Directory Access Protocol	
LLDP	Link Layer Discovery Protocol	
LSP	Layered Service Provider	
MAC	Medium Access Control	
MPLS	MultiProtocol Label Switching	
MPLS TE	Multiprotocol Label Switching Traffic Engineering	
NA	Network Automation	
NAT	Network Address Translation	
NCM	Netcool Configuration Manager (модуль IBM Tivoli)	
NFS	Network File System	

Обозначения	Расшифровка	Аналог (русск.)
NMC	Network Management Center	
NNMi	Network Node Manager I (модуль HP NMC)	
N/O	Netcool/OMNIbus (модуль IBM Tivoli)	
NPFA	Netcool Performance Flow Analyzer (модуль IBM Tivoli)	
NSLP	Net Ware Link Services Protocol	
NTP	Network Time Protocol	
NUMA	Non-Uniform Memory Architecture	
ONF	OpenNetworkingFoundation	
OSPF	Open Shortest Path First	
PI	Performance Insight	
PVLAN	Private Virtual Local Area Network	
QoS	Quality of Service	
RAM	Route Analytics Management	
RSVP	Resource ReSer Vation Protocol	
SLA	Service Level Agreement	
SMSA	Security Management System Appliance (средство HP NMC)	
SNMP	Simple Network Management Protocol	

Обозначения	Расшифровка	Аналог (русск.)
SOM	Security Operations Manager (модуль IBM Tivoli)	
SPI	Smart Plug-in	
SSH	Secure Shell	
TCP	Transmission Control Protocol	
TE	Traffic Engineering	
ToS	Type of Service	
UDP	User Datagram Protocol	
UUID	Universally Unique Identifier	
VAN	Virtual Application Network	
VCN	Virtual Cloud Network	
VLAN	Virtual Local Area Network	
VoIP	Voice over IP	
VN	Virtual Networking	
VTN	Virtual Tenant Network	
VXLAN	Virtual Extensible Local Area Network	

ВВЕДЕНИЕ

Современное состояние и тенденции развития компьютерных сетей показали, что потенциал роста производительности, пропускной способности сетей на основе традиционных технологий практически исчерпан. Это связано с ростом затрат времен на маршрутизацию, с трудностями в конфигурации сети и управления потоками в ней, особенно с учётом новых потребностей в политиках качества сервиса для высокоскоростных глобальных сетей и сетей центров обработки данных, с ростом потребности виртуализации сетей, т.е. отображения нескольких логически изолированных сетей с независимыми политиками качества обслуживания наобщий набор сетевых ресурсов.

В ответ на указанные выше проблемы в 2006 году возникла (и с тех пор интенсивно развивается) концепция программно-конфигурируемых сетей (ПКС). Следование этой концепции позволит ускорить маршрутизацию в сетях, повысить удобство конфигурирования, виртуализации, настройки качества обслуживания, но требует дополнительных исследований и разработок, в частности, в области организации аппаратуры сетевых коммутаторов, программных приложений для управления сетью и платформ для их выполнения.

Предварительный анализ предметной области и патентный поиск показали, что несмотря на наличие промышленного производства оборудования и программных средств для ПКС, а также большого количества научно-исследовательских проектов:

технологию ПКС на текущий момент нельзя признать зрелой, рынок только развивается, прогнозируется его резкий рост к 2015-2016 годам;

программные средства в исследовательских проектах не доведены до достаточного уровня «отлаженности», некоторые пока находятся на стадии «альфа-версии»;

в данной области многие экспериментальные разработки распространяются с открытым исходным кодом, с возможностью использования и доработки третьими сторонами без нарушения прав;

коммерческие программные и аппаратные средства способны взаимодействовать по открытым стандартам из данной предметной области.

Сказанное выше, в совокупности с открывающейся возможностью для отечественных разработчиков обеспечить технологическую независимость в области сетевых технологий, а также войти на новый рынок, обосновывает актуальности проводимой НИР.

Настоящий документ представляет собой научно-технический отчет по первому этапу НИР «Создание и развитие отечественной платформы с открытым программным кодом для управления программно-конфигурируемыми сетями (ПКС)». Документ содержит отчет по пунктам 1.1, 1.2, 1.4-1.12, календарного плана (результаты по пп. 1.3 приведены в отчёте по патентным исследованиям) в соответствии с техническим заданием (ТЗ) по государственному контракту № 14.514.11.4047 от 01 марта 2013 г. между Московским государственным университетом имени М.В. Ломоносова и Министерством образования и науки Российской Федерации.

Основной целью НИР является разработка прототипа отечественной распределенной платформы управления программно-конфигурируемыми сетями, который будет обеспечивать управление сегментами ПКС и потоками трафика в них с показателями производительности, сравнимыми с показателями производительности традиционных (не относящихся к ПКС) сетей при меньших требованиях к ресурсам в сравнении с существующими решениями в области ПКС.

Работы первого этапа НИР направлены, главным образом, на обобщение и анализ накопленного мирового опыта, сопоставление ПКС и традиционных сетей, на отбор имеющихся программных средств для более

детального исследования, на разработку отдельных алгоритмов управления ресурсами в ПКС.

На втором этапе основными задачами будут: программная реализация упомянутых алгоритмов, экспериментальное исследование этой реализации и отобранных программных средств (сетевых операционных систем), разработка методики применения ПКС (в сетях образовательных учреждений), подготовка проекта ТЗ на ОКР.

1 АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ, МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ, ПУБЛИКАЦИЙ ПО ПРИНЦИПАМ ПОСТРОЕНИЯ, УПРАВЛЕНИЯ СЕТЕВЫМИ РЕСУРСАМИ И АРХИТЕКТУР ПКС В СРАВНЕНИИ С ТРАДИЦИОННЫМИ СЕТЯМИ

1.1 Проблемы традиционных компьютерных сетей

Архитектура традиционных компьютерных сетей и Интернет закладывалась в конце 60-х – 70-е годы при создании первой сети ARPANET [1]. Однако с тех пор произошли серьезные качественные и количественные изменения в области использования КС. В настоящий момент архитектура КС устарела и не всегда способна быстро и эффективно реагировать на новые тенденции в развитии информационных технологий и потребности современного общества.

Согласно ежегодному исследованию компании Cisco на 2009-2014 годы [2–5] к 2014 году объем глобального интернет-трафика вырастет более чем в четыре раза и достигнет 767 эксабайт (1эксабайт = 10^{18}байт). Диаграмма исследования приведена на Рисунке 1.1. Прогнозируемый на 2014 год ежемесячный объем трафика (почти 64 эксабайта) равен 16 миллиардам дисков DVD, или 21 триллиону файлов MP3.

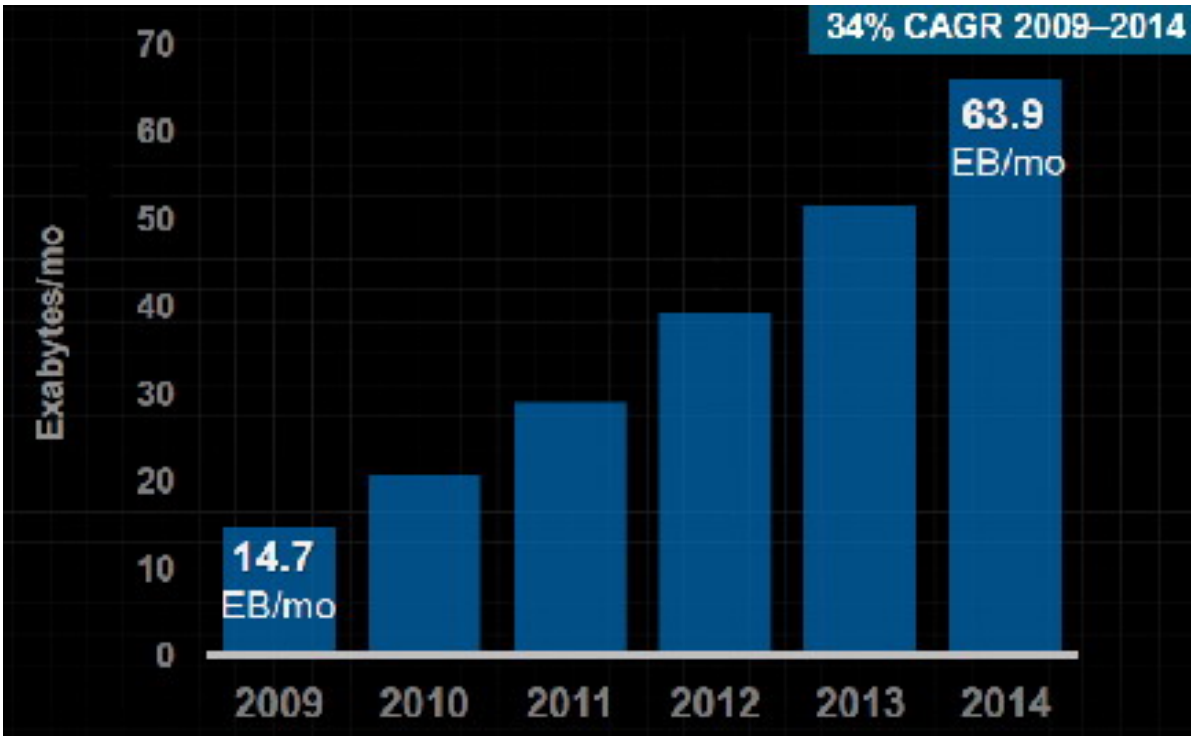

Рисунок 1.1 – Прогнозируемый рост глобального трафика Интернет (эксабайт в месяц)

При этом изменяется структура и динамика трафика. Прогнозируемая структура трафика приведена на Рисунке 1.2. Основным фактором роста станет видео-трафик. К 2014 году его доля должна превысить 91% в глобальном пользовательском интернет-трафике. Расширение сетевой полосы пропускания и скорости передачи данных в Интернете, а также рост популярности телевидения высокой четкости (HDTV) и объемного телевидения (3DTV) станут важнейшими факторами четырехкратного роста IP-трафика, который должен произойти в период с 2009 по 2014 гг. К 2014 году совокупный объем видеотрафика 3D и HD составит 42% от общего объема пользовательского видеотрафика в Интернете. Быстрее всего будет развиваться сегмент веб-конференций: в этой области объем трафика с 2009-го по 2014-й год увеличится в 180 раз.

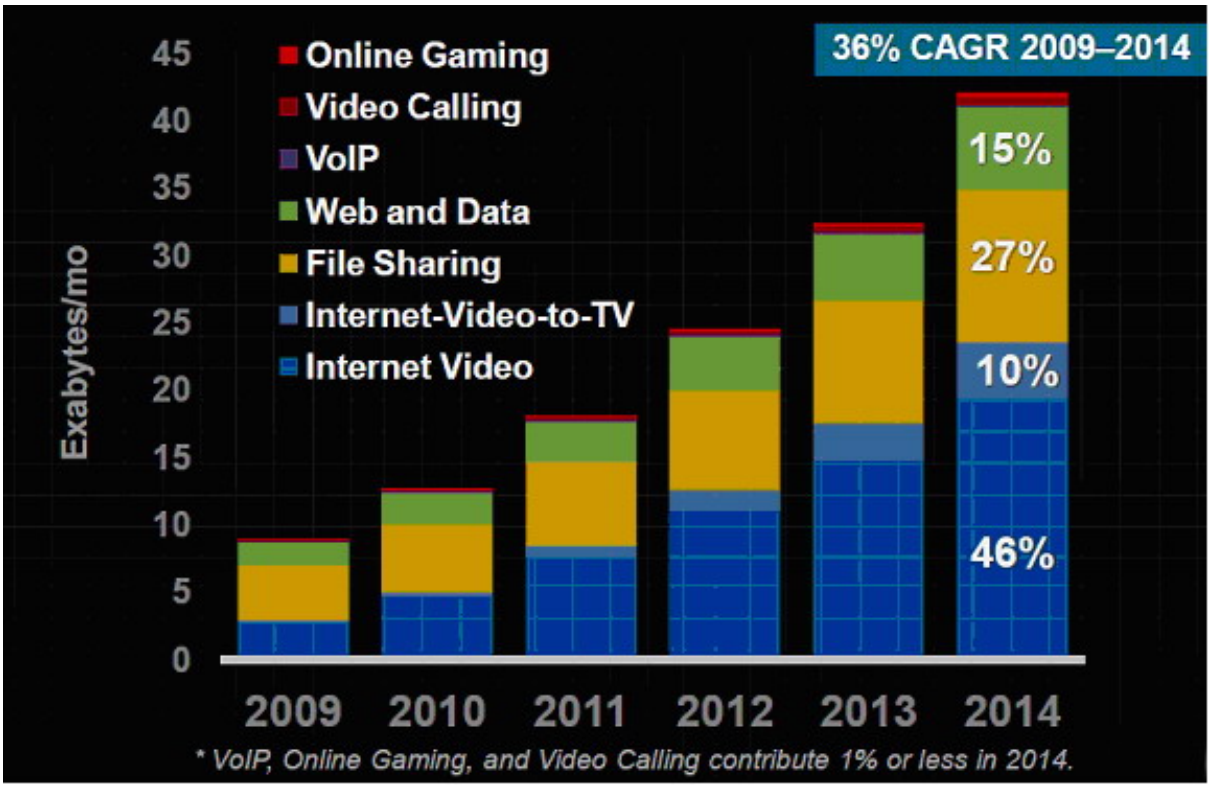

Рисунок 1.2 – Прогнозируемая структура трафика (эксабайт в месяц)

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 количество 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.

По данным исследований Cisco с 2009-го по 2014-й год объем глобального мобильного трафика, связанного с передачей данных, увеличится в 39 раз и к 2014 году составит 3,5 эксабайта в месяц или более 42 эксабайт в год. Тенденции роста мобильного трафика представлены на Рисунке 1.3.

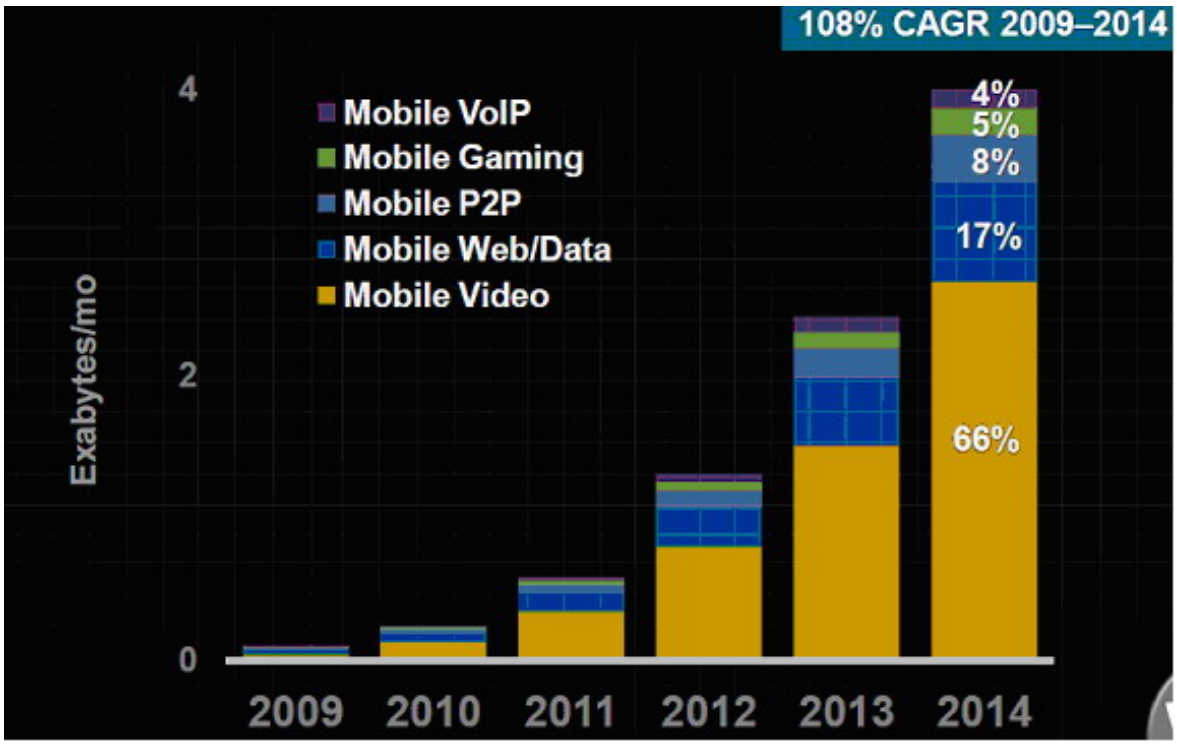

Рисунок 1.3 – Рост мобильного трафика и его распределение по типам

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.

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.

В 70-е-80-е годы в СССР велись работы по разработке своих стандартов и средств построения Сетей в рассматриваемой области. Чаще всего принимаемые стандарты были не совместимы со стандартами, принимавшимися тем, что позднее стало Интернет Сообществом (ISOC). Это привело к тому, что в 90-е годы в России не возникло своего производства программных и аппаратных средств построения Сетей. Сегодня Сети в России строятся на основе зарубежных средств. Это означает, что управление инфокоммуникационными сетями в России возможно настолько, насколько позволяют зарубежные производители соответствующих средств.

Таким образом, в традиционных сетях накопился целый ряд проблем, решение которых требует внесение изменений в существующую архитектуру:

1) Научно-технические проблемы - сегодня невозможно контролировать и надежно предвидеть поведение таких сложных объектов, как компьютерные сети;
2) Социальные - в повседневной жизни мы все больше и больше полагаемся на Интернет. Однако, безопасность данных, включая персональные данные, не гарантирована, Интернет не устойчив к внешним атакам;
3) Экономические – сети дороги и сложны в обслуживании, требуют высококвалифицированных специалистов.
4) Проблемы развития - в архитектуре современных сетей есть существенные барьеры для введения инноваций, экспериментирования, создания новых сервисов.
5) Архитектура компьютерных сетей требует пересмотра принципов ее построения.

1.2 Архитектура программно-конфигурируемых сетей

Программно-конфигурируемая сеть (ПКС) – это новый подход к построению архитектуры компьютерных сетей, при котором уровень управления (УУ) сетью (состоянием сетевой инфраструктуры и потоками данных в сети) и уровень передачи данных (УПД) разделяются за счет переноса функций управления (выполняемых в традиционной сети маршрутизаторами и коммутаторами) на отдельное центральное устройство, называемое контроллером. За счет такого разделения контроль состояния сети и управление сетью логически централизовано на контроллере. Кроме того такой подход позволяет уровню управления абстрагироваться от физической сетевой инфраструктуры уровня передачи данных, используя некоторое логическое представление сети. Взаимодействие между УУ и УПД осуществляется посредством единого унифицированного открытого интерфейса.

Основные идеи подхода ПКС были сформулированы специалистами университетов Стэнфорда (Ник Маккеон) и Беркли (Скотт Шенкер) в 2006 году [1, 2] и нашли широкую поддержку в академической среде, сообществе ведущих производителей сетевого оборудования и крупными ИТ-компаниями.

Заинтересованность ведущих ИТ-компаний вызвана тем, что, как показали первые практические эксперименты, ПКС подход позволяет повысить эффективность сетевого оборудования на 25%-30%, снизить затраты на эксплуатацию сетей более чем на 30%, повысить гибкость управления сетью за счет написания программ, существенно повысить безопасность, программно создавать новые сервисы и оперативно загружать их в сетевое оборудование. Внедрение этого подхода, прежде всего, должно оказать значительное влияние на сети центров обработки данных (дата-центров, ЦОД), корпоративные сети, WAN, сотовые и домашние сети.

Основные идеи, которые закладывались в ПКС, заключаются в следующем:

1) Разделение уровня передачи и уровня управления данными.
2) Единый, унифицированный, независящий от поставщика интерфейс между уровнем управления и уровнем передачи данных.
3) Логически централизованный уровень управления данными.
4) Виртуализация физических ресурсов сети.
5) Архитектура ПКС согласно [6] имеет три уровня согласно Рисунку 1.4.:
6) Уровень инфраструктуры сети включает в себя набор сетевых устройств (коммутаторов, маршрутизаторов) и каналов передачи данных.
7) Уровень управления, на котором отслеживается и поддерживается глобальное представление сети (ГПС). Под глобальным представлением сети понимается топология сети и состояние сетевых устройств. Уровень управления предоставляет программный интерфейс (API) для сетевых приложений.
8) Уровень сетевых приложений, в которых реализуются различные функции управления сетью: управление потоками данных в сети, управление безопасностью, мониторинг трафика, управление качеством сервиса, управления политиками и так далее.

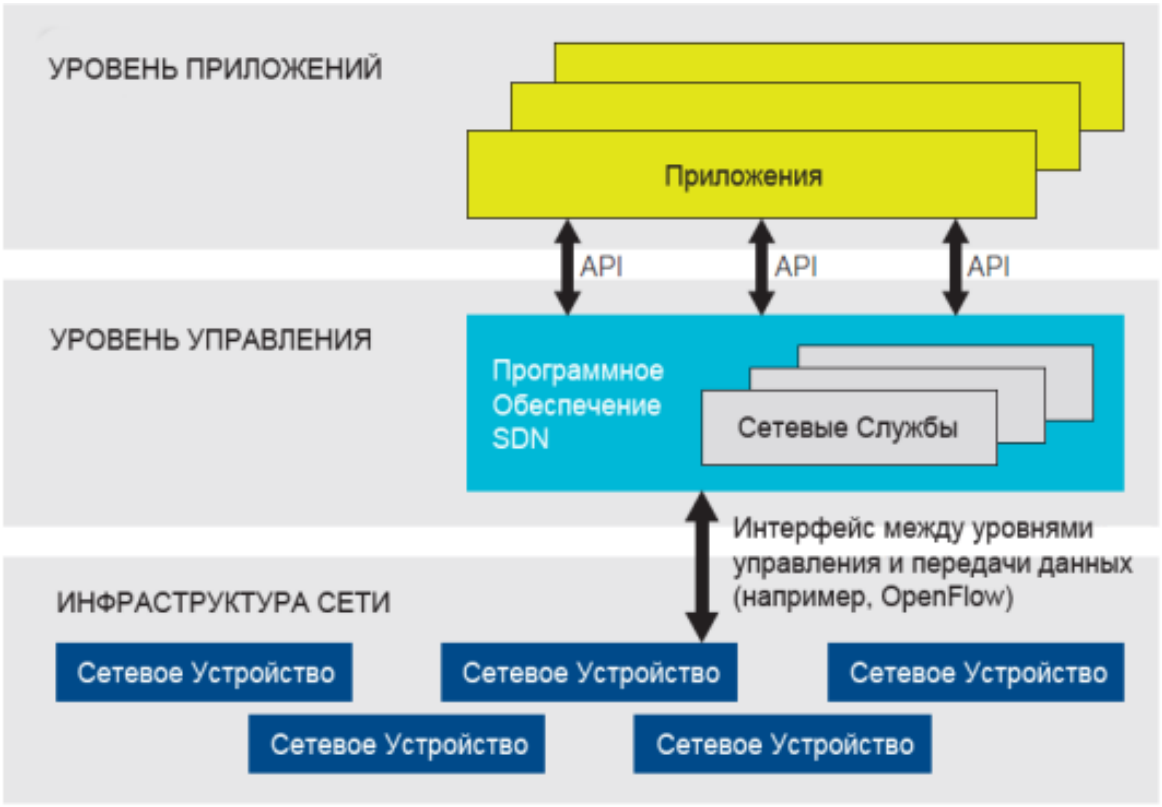

Рисунок 1.4 – Архитектура программно-конфигурируемых сетей

Современные маршрутизаторы решают две основные задачи: передача данных (forwarding) – продвижение пакета от входного порта на определенный выходной порт, и управление данными – обработка пакета и принятие решения о том, куда его маршрутизировать, на основе текущего состояния маршрутизатора. Таким образом, в рамках всей сети можно выделить уровень передачи данных (УПД), состоящий из средств передачи данных (линий связи, каналообразующего оборудования, маршрутизаторов и коммутаторов), и уровень управления (УУ) состояниями средств передачи данных

Развитие маршрутизаторов шло по пути сближения и «сращивания» этих двух уровней, аппаратного ускорения, совершенствования ПО и внедрения новых функциональных возможностей для увеличения скорости принятия решения по маршрутизации каждого пакета. Но при этом уровень управления остался достаточно примитивным, опираясь на сложные

распределенные алгоритмы маршрутизации и замысловатые инструкции по конфигурированию и настройке сети. Нужно отметить, что программное обеспечение маршрутизаторов, реализующее уровень управления, оставалось проприетарным и закрытым для разработчиков, исследователей и сетевых операторов.

В подходе ПКС было предложено разделить УУ и УПД. Схема разделения уровней в ПКС представлена на Рисунке 1.5.

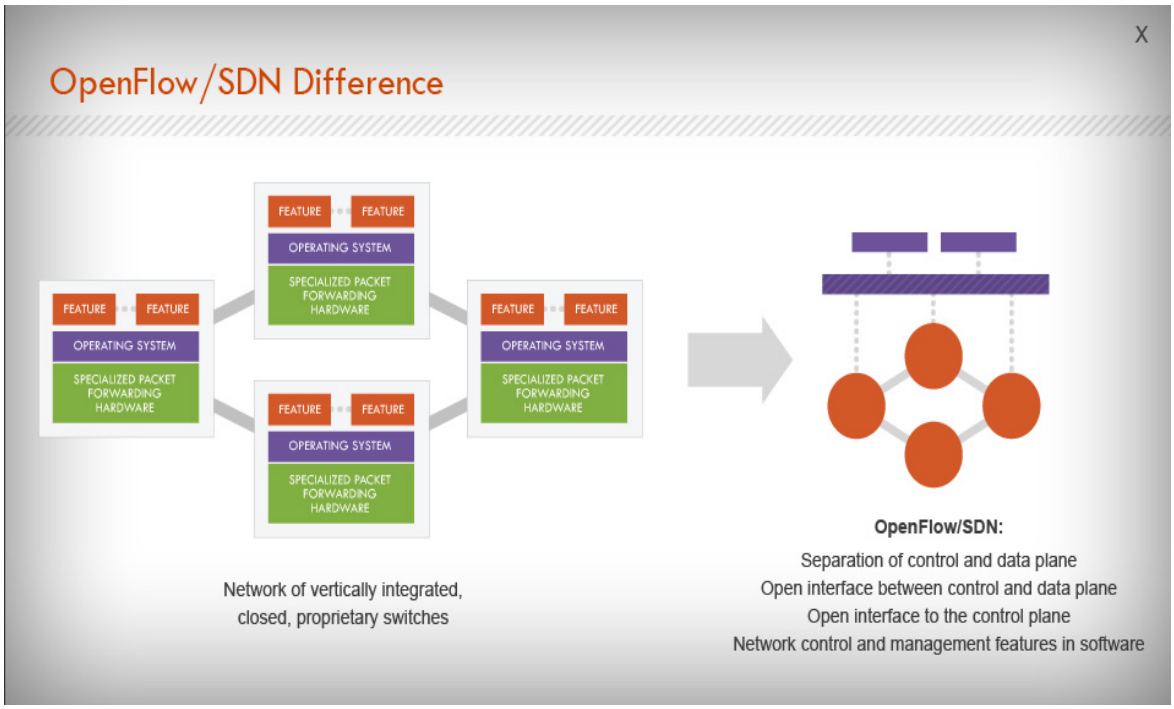

Рисунок 1.5 – Сравнение архитектуры ПКС и традиционной архитектуры КС

Таким образом, архитектура ПКС и предлагаемый централизованный подход дает следующие преимущества по сравнению с традиционными сетями с распределенным управлением передачей данных:

— *Программируемость и гибкость* управления сетью, значительное упрощение возможности модификации управления сетью за счет создания новых приложений или модификации существующих, автоматизация управления и администрирования сетями.

— *Адаптивность управления сетью*, то есть возможность изменять

поведение и состояние сети в режиме реального времени с учетом изменяющихся условий функционирования и адаптироваться к ним, адаптироваться к меняющимся потребностям пользователей сетей за счет создания новых сетевых приложений и сервисов. На разработку сетевых приложений требуется значительно меньше времени по сравнению с ручным переконфигурированием всей сети.

— Независимость от оборудования и проприетарного программного обеспечения производителей сетевого оборудования.

— Возможность независимого развертывания УУ и УПД.

— Возможность независимого масштабирования УУ и УПД.

— *Повышение надежности* за счет снижения объема распределенного состояния для управления. Вместо имеющихся распределенных протоколов, которые работают на каждом узле сети, каждый из них поддерживает базу данных распределенных копий состояний каналов в каждом узле, однако такая информация может быть собрана централизованно в одном месте – на контроллере. Таким образом, такая централизованная база данных будет содержать значительно меньше несогласованной информации, и такой подход позволит уменьшить вероятность циклов в сети.

— *Упрощение структуры и логики сетевых устройств*, поскольку теперь им не требуется обрабатывать огромное количество стандартов и протоколов, а достаточно выполнять только инструкции, полученные от контроллера.

— *Снижение стоимости коммутаторов и сетевой инфраструктуры* в целом за счет вынесения «мозгов роутеров» в контроллер.

Таким образом, подход ПКС позволяет значительно автоматизировать и упростить управление сетями за счет возможности их «программирования», позволяя строить гибкие масштабируемые сети, которые могут легко адаптироваться к изменяющимся условиям

функционирования и потребностям пользователей. Внедрение этого подхода, прежде всего, должно оказать значительное влияние на управление сетевой инфраструктурой в центрах обработки данных (ЦОД), корпоративными сетями, WAN, сотовыми и домашними сетями.

Однако в архитектуре ПКС можно отметить и определенные недостатки:

— *Проблема надежности:* Контроллер является потенциальной точкой отказа работы сети.

1) Количество управляющего трафика, предназначенного для централизованного контроллера, растет пропорционально количеству коммутаторов в сети.

2) Время установления новых потоков может расти значительно с ростом размеров сети.

— *Проблема производительности.* Производительность сети напрямую зависит от производительности контроллера и его физических ограничений.

1.3 Технология OpenFlow

Одной из наиболее перспективных и развивающихся реализаций подхода программно-конфигурируемых сетей является технология OpenFlow. Основным её документом является спецификация OpenFlow [7–11], в которой описываются основные компоненты OpenFlow-сети, принципы работы и взаимодействия компонентов (протокол OpenFlow). Стандартизирующей организацией для спецификации является ONF – Open Networking Foundation (https://www.opennetworking.org/).

На Рисунке 1.6 представлены основные компоненты OpenFlow сети.

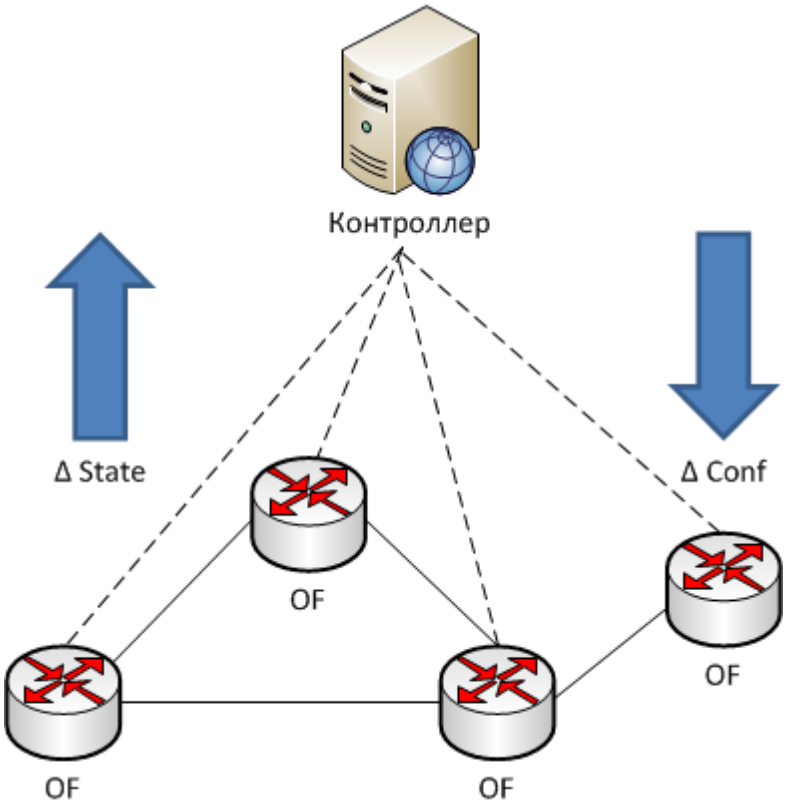

Рисунок 1.6 – Основные компоненты OpenFlow сети

Согласно спецификации основными компонентами OpenFlow сети являются:

– контроллер, содержащий:

1) сетевую операционную систему;

2) сетевые приложения.

– OpenFlow коммутатор;

– защищенный канал между контроллером и коммутатором;

– протокол OpenFlow.

Общий принцип функционирования OpenFlow-сети заключается в следующем: каждый OpenFlow коммутатор устанавливает защищенный канал с контроллером, посредством которого контроллер управляет им. Взаимодействие между коммутаторами и контроллером осуществляется посредством сообщений протокола OpenFlow. Контроллер получает

информацию об изменении состояний элементов в сети, на основе которой он конфигурирует сетевое оборудование, управляет сетевой инфраструктурой и потоками данных в сети.

1.4 Основные компоненты OpenFlow сети

1.4.1 OpenFlow коммутатор

1.4.1.1 Структура коммутатора. Согласно спецификации OpenFlowверсии 1.3 каждый коммутатор содержит одну или более таблиц потоков (flow table), групповую таблицу (group table) и поддерживает защищенный канал (Secure channel) для связи с удаленным контроллером. Понятие контроллера будет разъяснено позже. Пока будем считать, что это выделенный физический сервер с установленным специальным программным обеспечением.

1.4.1.2 Каждая таблица потоков в коммутаторе содержит набор записей (flow entries), которые мы будем называть записями о потоках или *правилами*. Каждая запись о потоке (правило) состоит из полей признаков (match fields), счетчиков (counters) и набора инструкций (instructions) согласно Рисунку 1.7.

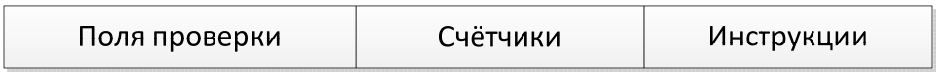

Рисунок 1.7 – Структура правила в таблице потоков OpenFlow коммутатора

1.4.1.3 Механизм работы OpenFlow коммутатора достаточно простой. Каждый пришедший пакет попадает на конвейер коммутатора согласно Рисунку 1.8. У каждого пакета «вырезается» *заголовок* (битовая строка определенной длины). Для этой битовой строки в таблицах потоков, начиная с первой, ищется правило, у которого поле признаков больше всего соответствует (совпадает) с заголовком пакета. При наличии совпадения, над

пакетом и его заголовком выполняют преобразования, определяемые набором инструкций, указанных в найденном правиле. Инструкции, ассоциированные с каждой записью таблицы, описывают действия, связанные с пересылкой пакета, модификацией заголовка пакета, обработкой в таблице групп, обработкой в конвейере, пересылкой пакета на определенный порт коммутатора. Инструкции конвейера обработки позволяют пересылать пакеты в последующие таблицы для дальнейшей обработки и позволяют передавать информацию (в виде метаданных) между таблицами. Инструкции так же определяют правила модификации счетчиков, которые могут быть использованы для сбора разнообразной статистики.

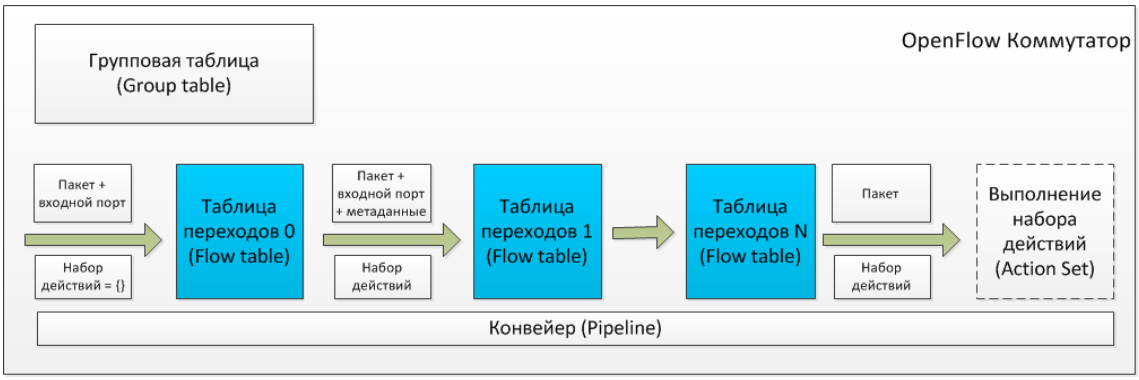

Рисунок 1.8 – Пакет потока, проходящий конвейер обработки

В том случае, если нужного правила в первой таблице не обнаружено, пакет инкапсулируется и отправляется контроллеру, который формирует соответствующее правило для пакетов данного типа и устанавливает его на коммутаторе (или на наборе управляемых им коммутаторов), либо пакет может быть сброшен (в зависимости от конфигурации коммутатора).

1.4.1.4 Как уже было отмечено, запись о потоке может предписывать переслать пакет в определенный порт. Это обычно физический порт, но это может быть и виртуальный порт, определенный коммутатором или зарезервированный виртуальный порт, определенный спецификацией протокола. Зарезервированные виртуальные порты могут определять общие действия пересылки, такие как отправка контроллеру, широковещательная

(лавинная) рассылка, или пересылка, не использующая OpenFlow-методов, то есть «обычная» обработка коммутатором. Виртуальные порты, определенные коммутатором, могут точно определять группы агрегирования каналов, туннели или интерфейсы с обратной связью.

Записи о потоках могут также указывать на группы, в которых определяется дополнительная обработка. Группы представляют собой наборы действий для широковещательной рассылки, а также наборы действий пересылки с более сложной семантикой (например, multipath, быстрое изменение маршрута, агрегирование каналов). Механизм групп позволяет эффективно применять общие выходные действия для потоков.

1.4.1.5 Таблица групп содержит записи о группах; каждая групповая запись содержит список контейнеров действий со специальной семантикой в зависимости от типа группы. Действия в одном или нескольких контейнерах действий применяются к пакетам, отправляемым в группу.

Установка, обновление и удаление правил в таблицах потоков коммутатора осуществляется контроллером. Правила могут устанавливаться реактивно (в ответ на пришедшие пакеты) или проактивно (заранее, до прихода пакетов).

1.4.1.6 Управление данными осуществляется не на уровне отдельных пакетов, а на уровне потоков пакетов. Правило в OpenFlow коммутаторе устанавливается с участием контроллера только для первого пакета, затем все остальные пакеты потока его используют.

1.4.1.7 Типы коммутаторов. По функциональности коммутаторы с поддержкой протокола OpenFlow можно разделить на два типа: OpenFlow-only и OpenFlow-hybrid.

1.4.1.8 OpenFlow-only - это коммутаторы, которые поддерживают только OpenFlow операции. В таких коммутаторах все пакеты

обрабатываются OpenFlow конвейером и не могут быть обработаны иным способом.

1.4.1.9 OpenFlow-hybrid – это коммутаторы, которые поддерживают одновременно как OpenFlow операции, так и обычные операции Ethernet-коммутации, то есть традиционную L2 Ethernet-коммутацию, VLAN, L3 маршрутизацию, ACL и QoS обработку. Эти коммутаторы должны обеспечивать механизм классификации трафика, маршрутизируя его либо в конвейер OpenFlow, либо в обычный конвейер обработки. Например, коммутатор может использовать VLAN тег или входной порт пакета, чтобы принять решение о том, каким образом обрабатывать пакет, или следует направить все пакеты в OpenFlow конвейер. Этот механизм классификации находится за рамками спецификации OpenFlow. OpenFlow-hybrid коммутаторы могут также разрешать передавать пакеты из OpenFlow конвейера в обычный конвейер через виртуальные порты *NORMAL* и *FLOOD*.

1.4.1.10 По реализации коммутаторы можно разделить на программные и аппаратные. В Таблице 1.1 приведены существующие программные реализации коммутаторов, поддерживающих работу по протоколу OpenFlow. На сегодняшний день существует множество гибридных коммутаторов с поддержкой протокола OpenFlow от ведущих производителей сетевого оборудования (NEC, HP и другие).

Таблица 1.1 – Существующие программные реализации OpenFlow коммутаторов

Название	of11 softswitch	LINC switch	of13 softswitch	Openv Switch	ofss
Разработчик	CPqD	Infoblox	CPqD	Nicira	TrafficLab, Ericsson Research, Hungary
Лицензия		Apache 2 license	BSD license	Apache 2 license	
Ссылка	https://github.com/CPqD/of11softswitch.git	https://github.com/FlowForwarding/LINC-Switch	https://github.com/CPqD/ofsoftswitch13	http://openvswitch.org/	https://github.com/TrafficLab/ofss
Протокол	OF 1.1	OF 1.2, OF 1.3, OF-config 1.1.1	OF 1.3	OF 1.2	OF 1.0
Язык програм-мирования	C/C++	Erlang	C++		C++
ОС	Linux	Linux	Linux	Linux	Linux

1.4.1.11 Существующие коммутаторы можно также разделить по версии поддерживаемого протокола OpenFlow. Однако на сегодняшний день все аппаратные коммутаторы поддерживают только версию протокола 1.0 и содержат только одну таблицу потоков. Программные коммутаторы поддерживают протокол до версии 1.3 включительно с множеством таблиц.

1.4.1.12 Можно выделить следующие основные функции OpenFlow-коммутатора:

1) Инициирование, установление и поддержка защищенного канала с контроллером.
2) Информирование контроллера об изменении состояния портов.
3) Информирование контроллера об удалении записи потока.
4) Уведомление об изменении статуса.
5) Уведомление об ошибке.
6) Запрос на установление правила для нового потока данных.

1.4.2 Защищенный канал связи

1.4.2.1 Защищенный канал используется для обмена сообщениями между коммутатором и контроллером. Типовой контроллер управляет множеством OpenFlow каналов, один на каждый OpenFlow коммутатор. OpenFlow коммутатор может иметь один OpenFlow канал к одному контроллеру или множество каналов для повышения надежности к разным контроллерам (в версии спецификации 1.3).

OpenFlow контроллер обычно управляет OpenFlow коммутаторами удаленно, одной или несколькими сетями. OpenFlow канал устанавливается между коммутатором и контроллером, используя TLS или обычный TCP. OpenFlow канал может состоять из нескольких сетевых соединений для использования параллелизма (с использованием дополнительных соединений

- в версии 1.3). OpenFlow коммутатор обычно инициирует соединение к OpenFlow контроллеру следующим образом.

1.4.2.2 Для установления соединения коммутатора с контроллером должны быть определены IP адрес и порт контроллера. Если эти данные известны коммутатору, он инициирует стандартное TLS или TCP соединение к контроллеру.

Когда OpenFlow соединение впервые установлено, каждая сторона должна немедленно послать сообщение OFPT_HELLOc максимальным номером протокола, которое поддерживается отправителем (коммутатором). После этого сообщения контроллер вычисляет наименьшую версию. Если эта версия поддерживается контроллером, соединение продолжается. В противном случае посылает сообщение OFPT_ERROR.

1.4.2.3 Прерывание соединения. В том случае если коммутатор теряет контакт со всеми контроллерами в результате таймаута echo запросов, таймаутов TLS сессий и т.п., коммутатор должен немедленно войти либо в режим «failsecuremode» или «failstandalonemode», в зависимости от реализации и конфигурации коммутатора. В режиме «failsecuremode» только изменяется поведение коммутатора, и все пакеты и сообщения, направляемые на контроллер, сбрасываются. Правила в таблице должны оставаться до истечения соответствующего таймаута в «failsecuremode». В режиме «failstandalonemode», коммутатор обрабатывает все пакеты, используя зарезервированный портOFPP_NORMAL. Другими словами коммутатор действует как обычный Ethernet коммутатор или маршрутизатор. Такой режим обычно доступен на гибридных коммутаторах.

При подключении к контроллеру снова, существующие записи потоков остаются. Затем контроллер имеет возможность удалить все записи, если это необходимо.

При первом запуске коммутатора он функционирует в режиме «failsecuremode» или «failstandalonemode» до тех пор, пока успешно не присоединится к контроллеру.

1.4.3 Протокол OpenFlow

1.4.3.1 Протокол поддерживает три типа сообщений:

1) Controller-to-switch – инициируются контроллером и используются для непосредственного контроля и управления состоянием коммутатора.
2) Асинхронные сообщения инициируются коммутатором и используются для уведомления контроллера о событиях в сети (ошибках, отказах) и изменениях состояния коммутатора.
3) Симметричные сообщения могут инициироваться как коммутатором, так и контроллером.

1.4.3.2 Сообщения контроллер-коммутатор. Контроллером инициируются следующие сообщения, которые могут требовать или не требовать ответа от коммутатора:

– Features: контроллер может запросить возможности коммутатора с помощью запроса features; коммутатор должен отвечать с помощью features ответа, в котором указываются возможности коммутатора. Это обычно выполняется при создании OpenFlow канала.

– Configuration: контроллер устанавливает и запрашивает параметры конфигурации коммутатора. Коммутатор только отвечает на запросы контроллера.

– Modify-State: сообщения Modify-State отправляются контроллером для управления состоянием коммутаторов. Их главная цель заключается в добавлении/удалении и модификации правит в OpenFlow таблицах и установке характеристик (параметров) порта коммутатора.

– Read-State: сообщения Read-State используется контроллером для сбора

статистических данных от коммутатора.

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

— Barrier: сообщения Barrier запрос/ответ используются контроллером для обеспечения того, чтобы были установлены зависимости между сообщениями или для получения уведомлений о завершенных операциях. Используются, когда необходима обработка сообщений в определенном порядке.

— Role-Request: запрос используется для изменения текущей роли контроллера на коммутаторе (для повышения роли с Slave до Master) (для версии протокола 1.3).

— Asynchronous-Configuration: это сообщение может быть использовано контроллером для установления фильтра на асинхронные сообщения, получаемые от коммутаторов (для версии протокола 1.3).

1.4.3.3 Асинхронные сообщения. Коммутаторы посылают асинхронные сообщения контроллеру для уведомления о прибытии пакета, изменении состояния коммутатора или ошибке. Ассинхронные сообщения бывают следующих типов:

— Packet-in. Для всех пакетов, которые не имеют соответствующих правил в таблице коммутатора, коммутатор генерирует сообщение Packet-inи отправляет его контроллеру. Для всех пакетов, пересылаемых в виртуальный порт CONTROLLER, сообщение Packet-in отправляется в контроллер. Если коммутатор имеет достаточно памяти в буфере пакетов, отправляемых

контроллеру, packet-in сообщение содержит некоторую часть заголовка пакета (по умолчанию 128 байт) и идентификатор ID буфера, который будет использоваться контроллером, когда коммутатор будет готов к пересылке пакета. Коммутаторы, которые не поддерживают внутреннюю буферизацию (или исчерпали внутренний буфер) должны посылать пакет целиком контроллеру как часть сообщения. Буферизованные пакеты, как правило, обрабатываются с помощью сообщений Packet-out от контроллера, или автоматически удаляются через некоторое время.

— Flow-Removed. Когда правило для нового потока добавляется в коммутатор с помощью сообщения flow-mod, устанавливается значение тайм-аута для него (время простоя правила). Это правило должно быть удалено через этот промежуток времени из-за недостатка активности или неиспользуемости правила. Также может быть указан жесткий тайм-аут, когда запись должна быть удалена независимо от активности потока. Сообщение flow-mod также определяет, должен ли коммутатор отправлять сообщение об удалении потока контроллеру, когда поток закончится.

— Port-status. Коммутатор имеет возможность отправлять сообщения Port-status контроллеру при изменениях состояний конфигурации порта.

— Error. Коммутатор имеет возможность уведомить контроллер о проблемах с помощью сообщений об ошибках.

1.4.3.4 Симметричные сообщения отправляются без запроса в любом направлении. Ассинхронные сообщения бывают следующих типов:

— Hello: сообщениями Hello коммутатор и контроллер обмениваются при установлении соединения.

— Echo: сообщения Echo типа запрос/ответ может отправлять любой коммутатор или контроллер, и в этом случае обязательно должен быть получен ответ. Они могут использоваться для измерения задержек или пропускной способности соединения контроллер-коммутатор, а также

проверки живучести соединения.

— Experimenter: сообщения Experimenter предназначены для обеспечения дополнительной функциональности для проведения экспериментов в пространстве типов сообщений OpenFlow.

1.4.3.5 Доставка сообщений:

— Протокол OpenFlow обеспечивает надежную доставку сообщений и их обработку, но не обеспечивает автоматические подтверждения о доставке или упорядоченную обработку сообщений. Обработка сообщений обеспечивается для основного соединения и дополнительных соединений, использующих надежную передачу данных, но не поддерживается на дополнительных соединениях, использующих ненадежную передачу данных (версия 1.3).

— Доставка сообщений гарантируется до тех пор, пока полностью не откажет OpenFlow канал, в этом случае контроллер не может делать каких-либо предположений о состоянии коммутатора.

1.4.3.6 Обработка сообщений. Коммутаторы должны обрабатывать каждое сообщение, полученное от контроллера полностью с возможной генерацией ответа, если это необходимо. Если коммутатор не может полностью обработать сообщение, полученное от контроллера, он должен послать обратно сообщение об ошибке. Для packet_out сообщений полная обработка сообщения не гарантирует, что содержащийся пакет действительно выйдет из коммутатора. Содержащийся пакет может быть сброшен после обработки из-за перегрузки коммутатора, QoS политики или если он отправлен на заблокированный или неисправный порт.

Дополнительно коммутаторы могут посылать контроллеру все асинхронные сообщения, сгенерированные вследствие изменения состояния коммутатора, такие как flow_removed, port_status или packet_in сообщения. Эти сообщения могут быть отфильтрованы на основе сообщения

Asynchronous Configuration. Например, пакеты, полученные на портах коммутатора и которые должны быть отправлены в контроллер, могут быть сброшены из-за перегрузки коммутатора или политики QoS внутри коммутатора и не генерировать packet_in сообщения. Такие сбросы пакетов могут случаться для пакетов с точно определенным выходным действием на контроллер. Такие сбросы пакетов могут также случаться, когда пакет не совпадает ни с одной записью в таблице и действие таблицы по умолчанию стоит «отправить на контроллер». Контроллеры могут игнорировать сообщения, которые они получают, но они должны отвечать на echo сообщения для предотвращения потери соединения с коммутатором.

1.4.3.7 Порядок сообщений. Порядок сообщений может быть обеспечен за счет использования сообщений типа barrier. При отсутствии таких сообщений, коммутаторы могут произвольно переупорядочивать сообщения для увеличения производительности; следовательно, контроллеры не должны зависеть от специфического порядка обработки сообщений. В частности, правила могут быть вставлены в таблицы в порядке, отличном от того, в котором flow-mod сообщения получены коммутатором. Сообщения не могут быть переупорядочены через сообщение barrier, и сообщение barrier должно быть обработано только когда все более приоритетные сообщения были обработаны.

Если два сообщения от контроллера зависят друг от друга (например, flow_mod добавляется с последующим packet_out в OFPP_TABLE), они должны быть отделены сообщением barrier, которые позволяют обрабатывать команды в определенном порядке.

1.4.4 Контроллер, сетевая ОС и сетевые приложения

1.4.4.1 Ключевым элементом в программно-конфигурируемых сетях (ПКС) является контроллер. Под *контроллером* понимают специальное ПО, установленное на физическом сервере, осуществляющее контроль и

управление состоянием сети и ее элементами и управление потоками данных в сети. Контроллер состоит из сетевой операционной системы и набора сетевых приложений, функционирующих поверх нее. *Сетевая ОС* осуществляет непосредственное взаимодействие с элементами сети (коммутаторами), контролирует 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. *Сетевые приложения*, написанные администратором сети, на основе информации о состоянии сети осуществляют непосредственное управление трафиком (на основе 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.

1.4.4.2 Общая архитектура контроллера представлена на Рисунке 1.9.

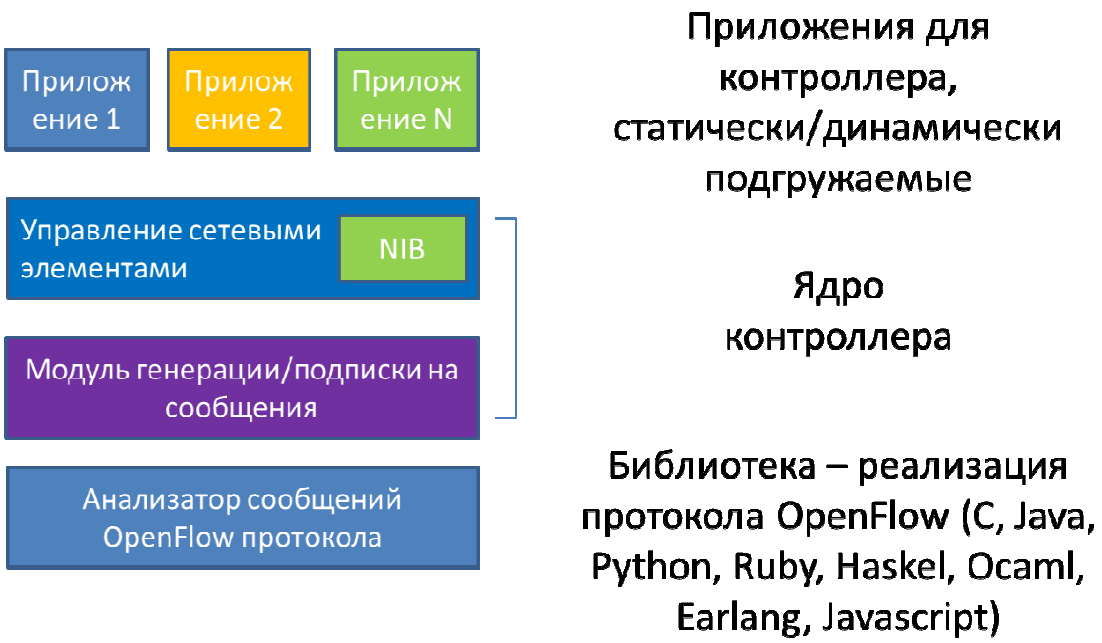

Рисунок 1.9 – Общая архитектура контроллера

Можно выделить следующие основные функции сетевой ОС:

— Управление устройствами сети.

— Управление топологией (построение топологии сети, обработка добавления/удаления новых элементов сети).

— Управление приложениями:

1) Одновременный запуск нескольких приложений.
2) Регистрация приложений.
3) Обработка ошибок приложений.
4) Обработка ошибок взаимодействия приложений.
5) Подписка на события от приложений.
6) Приоритизация приложений.
7) Обеспечение взаимодействия между приложениями посредством соответствующей инфраструктуры.
8) Балансировка нагрузки приложений (по потокам).
9) Разграничение прав доступа приложений к элементам сети.

— Управление доступными ресурсами сервера (потоками, ядрами):

1) Поддержка многопоточности (многопоточная обработка событий, приложений).
2) Мониторинг загрузки потоков.

— Управление событиями и прерываниями:

1) Обработка событий от коммутаторов.
2) Механизмы инициализации новых событий.
3) Распределение событий между приложениями.

1.4.4.3 Возможны *два режима установления правил* контроллером для новых потоков:

— Реактивный – на запрос установления нового потока, контроллер

формирует одно правило и устанавливает его на коммутатор, инициировавший запрос.

— Проактивный – на запрос установления нового потока, контроллер вычисляет маршрут (или его часть) для потока и устанавливает соответствующие правила на все коммутаторы этого маршрута.

1.4.4.4 Спецификация никаким образом не описывает требования к контроллеру, его архитектуру и API для приложений.

1.5 Подходы к реализации сети управления ПКС

1.5.1.1 Для корректной работы OpenFlow коммутатора необходимо, чтобы он мог устанавливать и поддерживать соединение с соответствующим контроллером. Существует два основных подхода к организации такого соединения:

— Передавать сообщения OpenFlow по физически независимому каналу, изолированному от сети передачи данных.

— Для передачи управляющей информации можно использовать непосредственно сеть передачи данных, которой управляет контроллер.

1.5.1.2 Первый подход получил название out-of-band управления, а второй — in-band управления.

В случае использования out-of-band подхода в OpenFlow коммутаторе выделяется отдельный порт. Вся управляющая информация между контроллером и коммутатором передается через этот порт, для передачи пакетов в сети передачи данных используются другие порты коммутатора. Таким образом, для управления коммутаторами создается отдельная сеть согласно Рисунку 1.10.

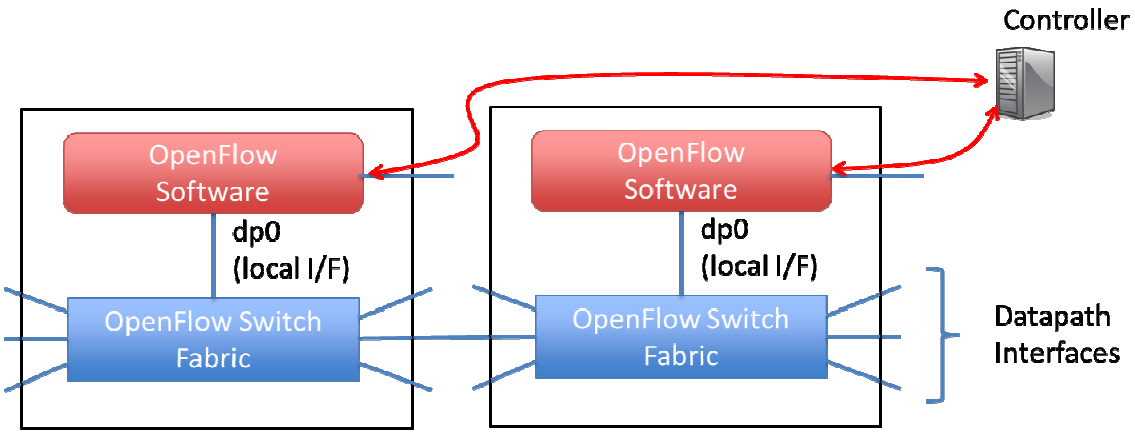

Рисунок 1.10 – Out-of-band подход управления коммутаторами в ПКС сети

Преимуществами out-of-band подхода являются простота (упрощается реализация OpenFlow коммутатора), надежность (трафик в сети передачи данных никак не влияет на передачу управляющей информации), безопасность (машины, не входящие в сеть управления, не имеют физической возможности получить доступ к управлению настройками коммутатора).

1.5.1.3 В случае in-band подхода (второй подход) для передачи управляющей информации используется та же сеть, что и для передачи пользовательских данных согласно Рисунку 1.11.

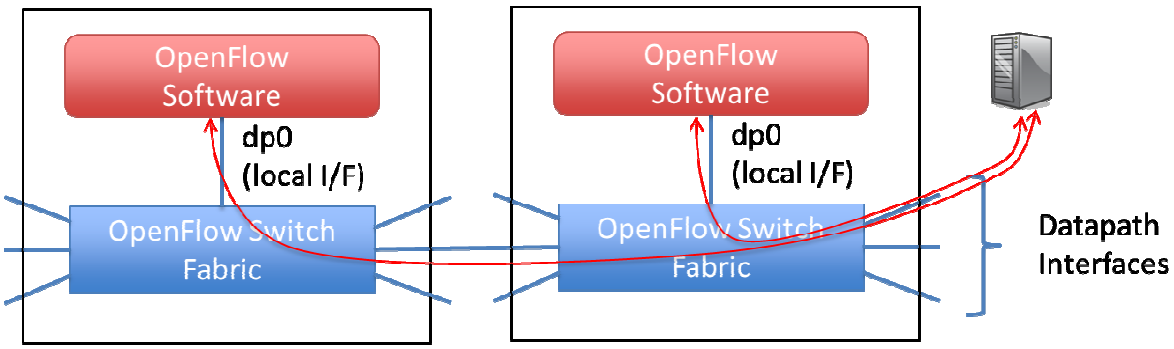

Рисунок 1.11– In-band подход управления коммутаторами в ПКС сети

К преимуществам in-bandподхода можно отнести экономичность (не требуется выделения отдельных портов и физических каналов под сеть управления, сокращение количества сетевого оборудования), упрощается

проектирование и поддержка сети. Однако могут возникать риски связанные с надежностью (перегрузки в сети передачи данных, отказы каналов) и с безопасностью (возможности DDoS-атак на контроллер).

Во многих случаях перечисленные выше достоинства in-band управления делают его использование более предпочтительным, чем out-of band управление.

1.6 Выводы

В данном разделе отчета:

— Приведено описание архитектуры ПКС, проведен анализ преимуществ и недостатков нового подхода и проблемы традиционной архитектуры КС.

— Рассмотрены основные компоненты ПКС на основе технологии OpenFlow: OpenFlow-коммутатор, принцип функционирования, типы, защищенный канал, контроллер и его основные функции.

— Подробно рассмотрен протокол OpenFlow, основные типы сообщений.

— Рассмотрены подходы к организации управления в ПКС сети.

2 АНАЛИЗ СУЩЕСТВУЮЩИХ СОС ДЛЯ ПКС И ВЫБОР ОДНОЙ ИЗ НИХ ДЛЯ ПОСЛЕДУЮЩЕГО РАЗВИТИЯ

2.1 Введение

Целью раздела является анализ существующих сетевых ОС для управления сетевой инфраструктурой и потоками данных в программно-конфигурируемых сетях и выбор одной из них для последующего развития.

В рамках НИР для выбранной сетевой ОС необходимо разработать алгоритмы и методы для управления соединениями, построения топологии сети, управления событиями, приложениями, управления потоками данных, поэтому отдельное внимание уделяется архитектуре сетевых ОС и реализациям перечисленных аспектов их функционирования.

Таким образом, к выбираемой сетевой ОС можно сформулировать следующие *общие требования*:

1) открытые исходные коды;
2) проект активно развивается;
3) поддерживаемый OpenFlow протокол – версия не ниже 1.0;
4) платформа развертывания – Linux;
5) поддержка многопоточности.

В данном разделе проводится анализ существующих сетевых ОС по следующим параметрам:

– Общие характеристики:
    1) разработчик;
    2) лицензия;
    3) репозиторий;
    4) дата начала разработки;
    5) дата последней версии;

6) поддержка и развитие проекта.

– Цель создания сетевой ОС.
– Особенности реализации:
  1) язык программирования;
  2) версия протокола OpenFlow;
  3) платформа, кросс-платформенность;
  4) базовая сетевая ОС.
– Архитектура и функциональные возможности:
  5) архитектура;
  6) основные компоненты;
  7) функциональные возможности.

2.2 Существующие сетевые ОС для ПКС

В настоящее время известны следующие сетевые ОС для программно-конфигурируемых сетей:

– NOX-Classic
– NOX
– POX
– SNAC
– Beacon
– Maestro
–
–
–
–
– Floodlight
– Trema
– MUL
– ONIX
– Kandoo

2.3 Сравнение сетевых ОС по цели создания

В Таблице 2.1 приведнео сравнение сетевых ОС по цели создания для ПКС.

Таблица 2.1– Цели создания сетевых ОС

Название	ЯП	Цель создания
NOX Classic	C++, Python	Первый контроллер для опытной аппробации подхода ПКС. В настоящее время данная версия не развивается. Проект NOX-Classic был разделен на два самостоятельных проекта POX – контроллер на Python и NOX – контроллер на C++ с с целью повышения производительности последнего.
NOX	C++	NOX был создан с целью повышения производительности контроллера для ПКС сети за счет его написания на C++ и использования многопоточности. Появился из NOX Classic.
POX	Python	POX создавался для исследований и обучения с возможностью быстрой разработки и прототипирования приложений для управления сетями на основе технологии OpenFlow. Появился из NOX Classic.
SNAC	C++	SNAC - Simple Network Access Control – OpenFlow контроллер, разработанный для обеспечения гибкого простого способа определения политик (контроля доступа) в корпоративных сетях. SNAC позволяет настраивать сеть с помощью формального языка моделирования - formal modelling language (FML).
Beacon	Java	Первая кросс-платформенная многопоточная сетевая ОС для ПКС на языке Java.

Продолжение таблицы 2.1

Название	ЯП	Цель создания
Maestro	Java	Maestro – кросс-платформенный многопоточный контроллер на языке Java.
FloodLight	Java	Контроллер с открытыми исходными кодами, поддерживаемый открытым сообществом разработчиков. Написан на основе контроллера Beacon языке Java. Разрабатывается по лицензии Apache, т.е. может использоваться для любых целей.
Trema	C, Rybu	Trema разрабатывалась как платформа для разработки OpenFlow контроллеров на языках C/Ruby. Trema разрабатывалась для исследовательских и учебных учреждений, поэтому включает в себя интегрированную среду тестирования и отладки приложений.
Mul	C	– Контроллер MūL разрабатывался для обеспечения производительности и надежности, которые необходимы для развертывания сетей, выполняющих критически-важные задачи (с повышенными требованиями к надежности).
ONIX	C++, Python, Java	– Обеспечить общий API по сравнению с подобными системами для реализации функций уровня управления (т.е. для приложений), позволяя им самостоятельно находить компромиссы между согласованностью, устойчивостью и масштабируемостью. – Ориентированность на использование ONIX в разных средах: WAN, облака, корпоративные ЦОД. – Обеспечить гибкие примитивы распространения, которые позволяют администраторам сети разрабатывать

Продолжение таблицы 2.1

Название	ЯП	Цель создания
		приложения, не заботясь о механизмах распространения событий.
Kandoo	C, C++, Python	– Kandoo должен быть совместим с OpenFlow, то есть не должна вноситься какая-либо новая функциональность на уровне передачи данных в коммутаторы. – Kandoo автоматически распределяет управляющие сетевые приложения, то есть приложения не осведомлены о том, как они размещены в сети и разработчики приложений могут предполагать, что их приложения будут запущены на централизованном OpenFlow контроллере.

2.4 Сравнение сетевых ОС по общим характеристикам

Сравнительный анализ сетевых ОС для ПКС по общим характеристикам приведен ниже в Таблицах 2.2, 2.3 и 2.4. На основе сравнительного анализа общих характеристик сетевых ОС необходимо выделить активно развивающиеся и наиболее перспективные и жизнеспособные открытые проекты.

Таблица 2.2 – Сравнение общих характеристик NOX-Classic, NOX, SNAC, POX

Характеристика	NOX-Classic	NOX	SNAC	POX
Разработчик	2008-2009 - Nicira Networks, 2009-2010 - Stanford University, 2009-2010 - ICSI UC Berkeley	2009-2012 - ICSI UC Berkeley (Murphy McCauley, Amin Tootoonchian)	Ed Swierk, Rob Vaterlaus (BigSwitch Networks)	James McCauley
Лицензия	GPL v3	GPL v3	GPL v2	GPL v3
Репозиторий	https://github.com/noxrepo/nox-classic.git	https://github.com/noxrepo/nox.git	https://github.com/bigswitch/snac-nox	https://github.com/noxrepo/pox.git
Дата начала разработки	14 марта 2009	11 мая 2012	3 февраля 2010	2011
Дата последней версии	15 сентября 2010	19 декабря 2012	6 мая 2011	24 ноября 2012
Поддержка и развитие	Не развивается	Разработка приостановилась в мае 2012.	Не развивается	Развивается

Таблица 2.3 – Сравнение общих характеристик Beacon, Maestro, FloodLight, Trema

Характе-ристика	Beacon	Maestro	FloodLight	Trema
Разработ-чик	Stanford University, David Erickson	Rice University	Floodlight community	NEC, Yasuhito Takamiya
Лицензия	GPL v2 и Stanford University FOSS License Exception v1.0	LGPL v2.1	Apache License v.2.0	GPL v2
Репози-торий	git://gitosis.stanford.edu/beacon.git	https://code.google.com/p/maestro-platform/downloads/list	https://github.com/floodlight/floodlight.git	https://github.com/trema/trema.git
Дата начала разработки	Нет данных	Декабрь, 2010	конец 2011	Нет данных
Дата последней версии	Октябрь, 2012	Май, 2011	Март, 2013	Март, 2013
Поддержка и развитие	Развивается	Не развивается	Активно развивается	Активно развивается

Таблица 2.4 – Сравнение общих характеристик Mul, ONIX, Kandoo

Характе-ристика	MUL	ONIX	Kandoo
Разработ-чик	Kulcloud Networks (Dipjyoti Saikia)	Teemu Koponen , Martin Casado , Natasha Gude , Jeremy Stribling , Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, Scott Shenker	Soheil Hassas Yeganeh, Yashar Ganjali, University of Toronto
Лицензия	GPL v2	Закрытая разработка	Закрытая разработка
Репози-торий	https://git.code.sf.net/p/mul/code mul-code	Закрытая разработка	Закрытая разработка
Дата начала разработки	15 августа, 2012	Нет данных	Нет данных
Дата последней версии	Август, 2013	4-й квартал 2010	3-й квартал 2012
Поддержка	Активно развивается	Нет данных	Нет данных

На основе информации, изложенной выше в Таблицах 2..2, 2.3 и 2.4, все проекты по разработке контроллеров для ПКС можно разделить по текущему состоянию поддержки проекта. Сравнение проектов по текущему состоянию поддержки приведено в Таблице 2.5.

Таблица 2.5 – Сравнение проектов по их текущему состоянию

Текущее состояние проекта	Сетевые ОС
Нет данных	Onix, Kandoo (закрытые разработки)
Не развивается	NOX Classic, SNAC, Maestro
Развивается	NOX, POX, Beacon
Активно развивается	MUL, Trema, FloodLight

Исходя из Таблицы 2.5, в данной работе будут рассматриваться проекты с открытым исходным кодом, которые развиваются и активно развиваются.

2.5 Сравнение особенностей реализации

Сравнение особенностей реализации рассматриваемых контроллеров приведено ниже в Таблицах 2.6, 2.7 и 2.8.

Таблица 2.6 – Сравнение особенностей реализации NOX-Classic, NOX, SNAC, POX

Характе-ристика	NOX-Classic	NOX	SNAC	POX
Язык программирования	C++, Python	C++	C++, Python	Python (2.7)
Версия OpenFlow протокола	OF 1.0	OF 1.0, 1.1, 1.2, 1.3	v0.8.9, v1.0	OF 1.0
Платформа	Linux-платформы	последнии версии Ubuntu 11.10 и 12.04, Debian и RHEL 6.	Linux-платформы	Linux, MacOS и Windows Android, BeOS-подобную систему Haiku OS
Многопо-точность	нет	да	нет	нет
Базовая сетевая ОС		На основе NOX Classic	на основе NOX Classic	На основе NOX Classic

Таблица 2.7 – Сравнение особенностей реализации Beacon, Maestro, FloodLight, Trema

Характе-ристика	Beacon	Maestro	FloodLight	Trema
Язык программирования	Java	Java	Java	C
Версия OpenFlow протокола	OF 1.0	OF 1.0	OF 1.0	OF 1.0 (OF 1.3 в разработке)
Платформа	Windows, Linux, Mac Android (64-х битные)	Linux	Windows, Linux, Mac Android	Ubuntu 12.10, 12.04, 11.10, and 10.04 (i386/amd64, Desktop Edition) Debian GNU/Linux 6.0 (i386/amd64) Fedora 16 (i386/x86_64)
Многопо-точность	да	да	да	да
Базовая сетевая ОС			На основе Beacon	

Таблица 2.8 – Сравнение особенностей реализации Mul, ONIX, Kandoo

Характе-ристика	MUL	ONIX	Kandoo
Язык программирования	C	C++, Python, Java	C, C++, Python
Версия OpenFlow протокола	OF 1.0 (OF 1.2 планируется)	1.0	1.0 (разработка версии для 1.1, 1.2)
Платформа	Ubuntu 10.04 LTS	Linux	Linux
Многопоточность	да	да	да
Базовая сетевая ОС		NOX-Classic	ONIX как root-контроллер

На основе вышеприведенных Таблиц 2.6, 2.7 и 2.8. все сетевые операционные системы можно классифицировать по следующим критериям:

– по языку программирования – результаты сравнения приведены в Таблице 2.9;

– по версии поддерживаемого протокола OpenFlow – результаты сравнения приведены в Таблице 2.10.

Таблица 2.9 – Классификация сетевых ОС по языку программирования

Язык программирования	Сетевые ОС
C	MUL
Python	POX
C++	NOX
Java	Beacon, Maestro, FloodLight
JavaScript	NodeFlow
C, Ruby	Trema
C++, Python	NOX Classic, SNAC

Таблица 2.10 – Классификация сетевых ОС по версии поддерживаемого протокола OpenFlow

Версия протокола OpenFlow	Сетевые ОС
OF 1.0	NOX Classic, NOX, SNAC,POX, Beacon, Maestro, Floodlight, Trema, Mul
OF 1.1	NOX
OF 1.2	NOX
OF 1.3	NOX (в разработке), Trema (в разработке)

2.6 Сравнение архитектур и функциональных возможностей сетевых ОС

2.6.1 Сетевая ОС NOX

2.6.1.1 NOX-Classic - первый контроллер для ПКС сети на основе технологии OpenFlow, написанный на языках C++ и Python. В настоящее время данная версия не развивается и не поддерживается. Проект NOX-Classic был разделен на два самостоятельных проекта POX – контроллер на Python и NOX – контроллер на C++ с целью повышения производительности последнего. В NOX построен на основе асинхронной событийно-ориентированной парадигамы программирования, в которой функционирование контроллера определяется событиями – сообщениями от коммутаторов, сообщениями от приложений, сообщениями от модулей сетевой ОС. Данный подход подразумевает наличием в коде сетевой ОС главного цикла, в котором осуществляется выборка событий и их обработка. Однако при использовании данного подхода часто оказывается недопустимым длительное выполнение обработчика событий, поскольку в это время программа не может реагировать на другие события. Данный подход оправдан для использования и в контроллерах ПКС, которые запускаются на серверах, и часто применяется в серверных приложениях для решения проблемы масштабируемости на поддержку контроллером более 10 тысяч одновременных соединений.

2.6.1.2 Архитектура и основные компоненты. Основные модули (в NOX - компоненты) NOX находятся в папке builtin:

— *Модуль Kernel* – ядро контроллера.

— *Модуль Component* – реализует базовую функциональность для создания компонентов NOX (конфигурирование, установка компонента, подписка на события, отправка события, регистрация обработчиков). Компонент инкапсулирует функциональность, предоставляемую NOX. Сетевое приложение представляет собой совокупность взаимодействующих компонентов, которые обеспечивают требуемую функциональность. Каждый компонент может находиться в разных

состояниях (в том числе: не установлен, установлен, описан, загружен, сконфигурирован). Компоненты могут взаимодействовать между собой через статические переменные. Взаимодействие ядра и компонентов осуществляется через контексты компонентов.

— *Модуль Connection manager* - для соединения с коммутаторами используются сокеты Беркли для Unix (sys/socket.h).

— *Модуль Deployer* используется для создания разделяемых объектов.

— *Модуль DSO Deployer (A dynamic shared object deployer)* – используется для создания динамически разделяемых объектов (которые могут использоваться несколькими компонентами) и для каждого компонента формирует контекст. DSO – это объектный файл, который предназначен для разделяемого общего использования множеством компонентов NOX во время их выполнения. NOX поддерживает множество Deployer-ов для динамически и статически связанных компонентов. Deployer конструирует контекст и передает его ядру.

— *Модуль Static Deployer* – используется для статических разделяемых объектов.

— *Модуль Event Dispatcher* отвечает за управление событиями:

1) Ассоциирование событий с обработчиками и отправку событий соответствующим обработчикам. В компоненте реализованы: регистрация событий, регистрация обработчиков на события, для каждого событию поставлен в соответствие перечень компонентов с приоритетом обработки ими этого события. Для каждого события поставлена в соответствие цепочка вызовов обработчиков.

2) Поддержка очередей ожидающих событий.

2.6.2 Сетевая ОС POX

2.6.2.1 POX создавался для исследований и обучения с возможностью быстрой разработки и прототипирования приложений для ПКС на основе технологии OpenFlow.

2.6.2.2 Архитектура и основные компоненты. POX содержит следующие компоненты:

1) *L2/L3 forwarding* обеспечивают маршрутизацию на уровне L2/L3.
2) *Topology discovery* – модуль, обеспечивающий обнаружение основных элементов сети и построение топологии сети.
3) *Topology view* – модуль, который является корневым объектной модели из сущностей таких как коммутаторы, хосты и линки. Эта объектная модель заполняется другими модулями. Так openflow.topology заполняет объект топологии OpenFlow коммутаторами.
4) *Messenger* – компонент, обеспечивающий интерфейс для POX с внешними процессами через
5) *GUI* (свой и web-интерфейс) (POX Desk, DjangoFlow web UI для POX, JSON-RPC-over-HTTP)

POX API включает в себя следующие основные библиотеки:

1) *pox.lib.addresses*, с помощью которой осуществляется работа с IP и Ethernet адресами.
2) *pox.lib.revent*, с помощью которой осуществляется поддержка работы с событиями (events). Обработка событий построена по принципу публикация/подписка. Каждый объект может подписаться на события определенного типа. Поддерживается возможность создания собственных сбытий.
3) *pox.lib.packet* позволяет осуществлять анализ пакетов приходящих в контроллер пакетов ethernet, ARP, IPv4, ICMP, TCP, UDP, DHCP, DNS, EAP, LLDP, VLAN.
4) *pox.lib.recoco* осуществляет поддержку работы с потоками, задачами и таймерами.
5) *pox.lib.ioworker* содержит высокоуровневый API для работы с асинхронными

сокетами в POX.

6) pox.lib.graph.
7) *pxpcap* находится в папке \pox\lib\pxpcap и по-видимому предназначена для создания программ анализа сетевых данных, поступающих непосредственно на сетевую карту сервера.

2.6.3 Сетевая ОС SNAC

2.6.3.1 SNAC – Simple Network Access Control – OpenFlow контроллер, ориентированный на создание корпоративных сетей. Он основан на NOX версии 0.4 и отличается гибким языком определения политик, дружественным пользовательским интерфейсом для настройки устройств и мониторинга событий Пользовательский интерфейс SNAC представлен на Рисунке 2.1. SNAC позволяет настраивать сеть с помощью формального языка моделирования - formal modelling language (FML).

2.6.3.2 Архитектура и основные компоненты. Поскольку SNAC разработан на основе NOX, то архитектура и основные компоненты совпадают (см. п.2.6.1). Стоит выделить некоторые новые компоненты и особенности реализации:

1) *Модуль Packet-Classifier* – классифицирует поступающие packet_in сообщения в соответствии с их приоритетом.
2) SNAC в отличие от NOX действительно поддерживает защищенное соединение коммутатора с контроллером на основе SSL.

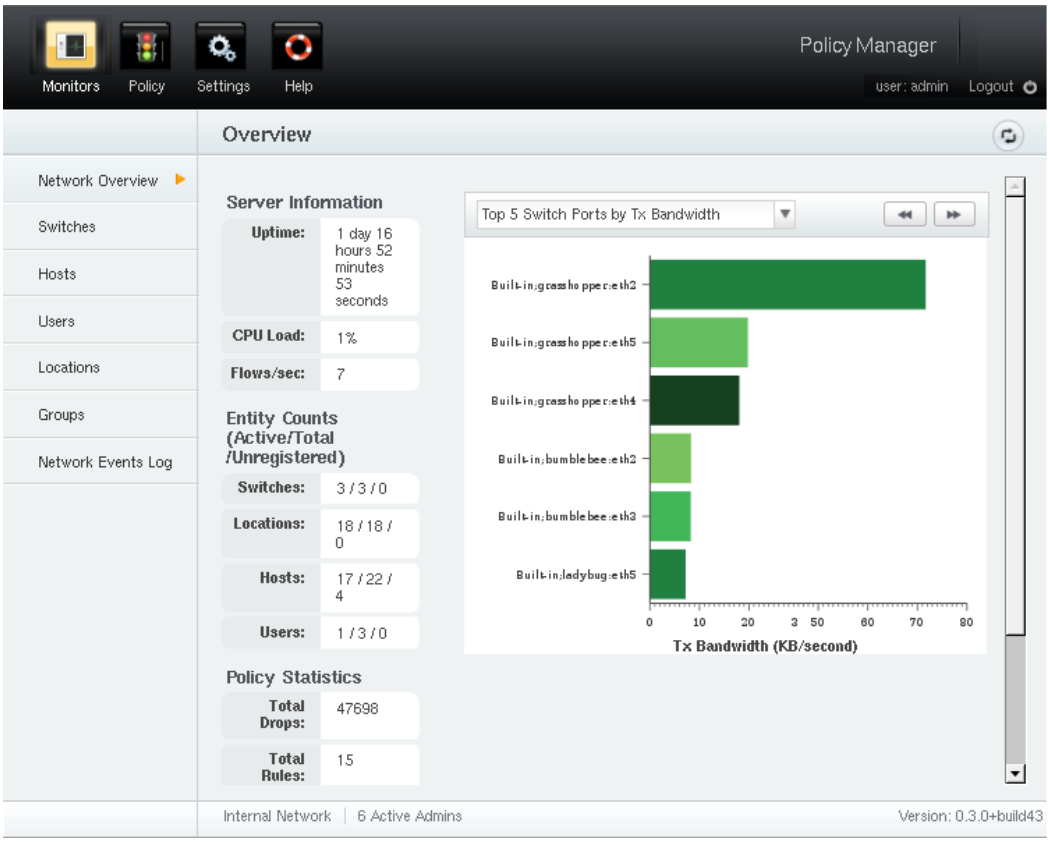

Рисунок 2.1 – Интерфейс SNAC

2.6.4 Сетевая ОС Beacon

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

2.6.4.2 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:

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

2.6.4.3 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 2.2 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.

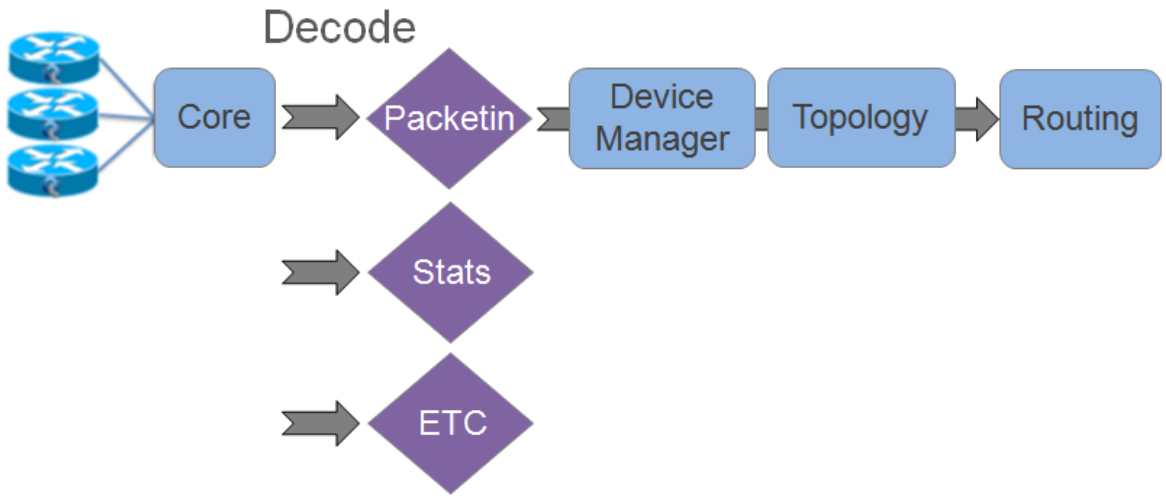

Рисунок 2.2 – Конвейер по обработке пакетов

Beacon также поддерживает многопоточный конвейер для обработки сообщений представленный на Рисунке 2.3.

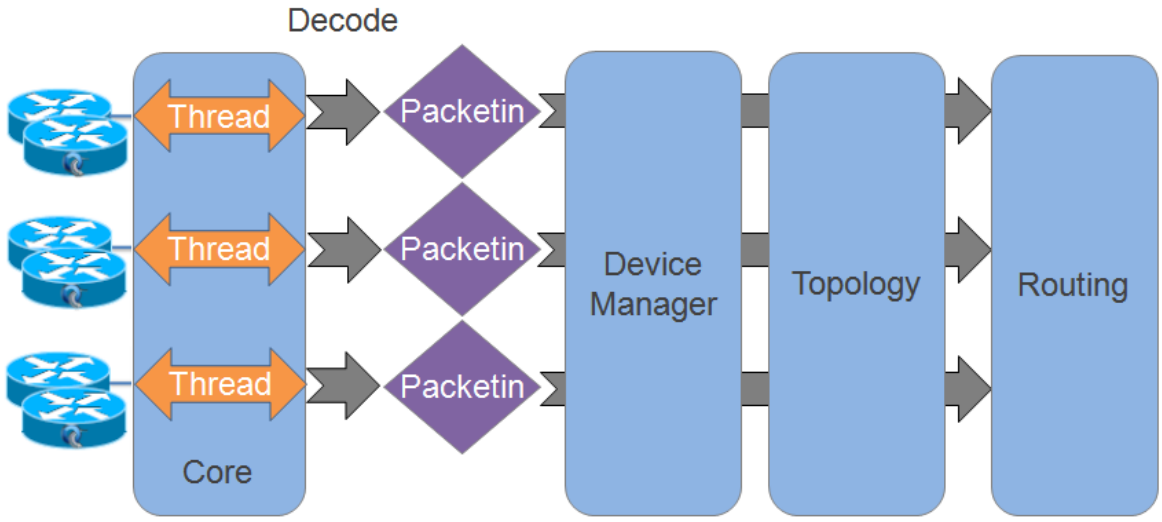

Рисунок 2.3 – Многопоточный конвейер по обработке пакетов.

2.6.4.4 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.

2.6.4.5 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.

2.6.5 Сетевая ОС Maestro

2.6.5.1 Maestro – расширяемый на языка Java OpenFlow контроллер, разработанный в университете Rice University, предоставляет интерфейсы для реализации модульных приложений управления доступом и состоянием сети. Maestro представляет собой платформу для обеспечения автоматического и программного управления сетью с помощью функций этих модульных приложений.

2.6.5.2 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. Общая структура Maestro приведена на Рисунке 2.4.

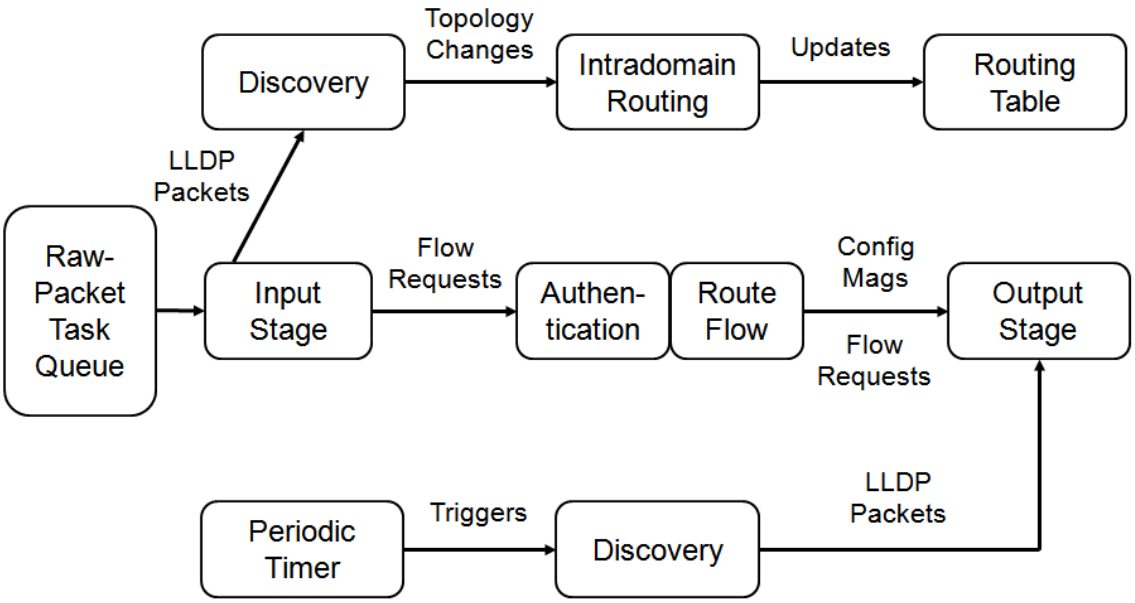

Рисунок 2.4 – Общая структура Maestro

2.6.5.3 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.

2.6.6 Сетевая ОС Floodlight

2.6.6.1 FloodLight – OpenFlow Java-контроллер для компаний и предприятий (класса enterprise) с открытыми исходными кодами. FloodLight появился из

исходных кодов Beacon. Имеет лицензию Apache – т.е. может использоваться для любых целей.

Использует чистый Java(не требуется OSGI, поддерживается Eclipse но не требуется). Очень просто собрать и запустить.

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

2.6.6.2 Архитектура 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.

2.6.6.3 Общая архитектура FloodLight представлена на Рисунке 2.5.

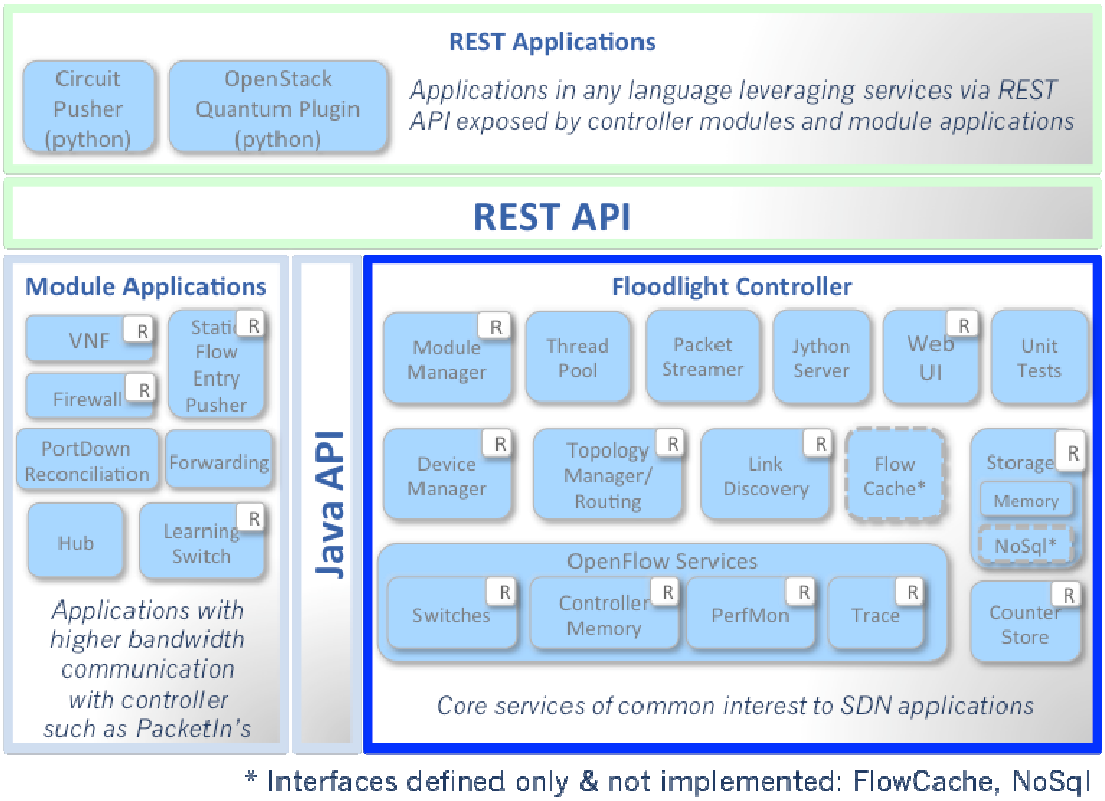

Рисунок 2.5 – Архитектура и основные модули FloodLight

2.6.6.4 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.

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

1) Обнаружение, выявление и предоставление информации о состоянии элементов сети и событий от них (топология, устройства, потоки).
2) Обеспечение взаимодействия контроллера с коммутаторами сети (через OpenFlow протокол)
3) Управление модулями FloodLight и распределение ресурсов таких как память, потоки, тесты.
4) Web-интерфейс и сервер для отладки (Jython).
5) Контроллер FloodLight содержит следующие модули:
6) *Module Manager* – система управления модулями FloodLight.
7) Thread Pool.
8) *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:
9) *REST*-интерфейс для того, чтобы определить характеристики OpenFlow сообщений, которые представляют интерес – фильтр
10) *Thrift*- интерфейс для потоковой передачи отфильтрованных пакетов.
11) Jython Server.
12) Web UI.
13) Unit Tests.
14) *Device Manager* – отслеживает устройства в сети (MACs, IPs), хосты, отображения MAC-(switch,port), MAC-IP, IP-MAC.
15) *Topology Manager/Routing* – отслеживает линки между хостами и коммутаторами, вычисляет кратчайший путь с помощью алгоритма Дейкстры.
16) *Link Discovery* – хранит состояние линков в сети, рассылает LLDP

пакеты.

17) Flow Cache.

18) *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.

19) *Counter Store* – модуль, отвечающий за сбор и обработку статистики OpenFlow и статистики сетевой ОС FloodLight.

20) *RestServer* – реализуется через Restlets (restlet.org), модули должны реализовывать RestletRoutable.

Сервисы FloodLight:

1) Switches;
2) Controller Memory;
3) PerfMon;
4) Trace.

2.6.6.6 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 МСЭ, 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. Приложение для реактивной пересылки пакетов.

— *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/frowarding приложения.

— *Learning Switch* – пример приложения Learning Switch Может заменить routing/frowarding приложения.

2.6.6.7 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.

— *OpuenStack*. 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.

2.6.6.8 Система загрузки модулей FloodLight. FloodLight использует свою собственную систему загрузки модулей, которая была разработана для следующих целей:

1) Определять какие модули должны быть загружены посредством изменения конфигурационного файла.
2) Заменять реализации модулей без изменения модулей от которых они зависят.
3) Создать ясную платформу и API для расширения FloodLight.
4) Обеспечить модульность кода.

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

2.6.6.9 Архитектура сети с контроллером FloodLight представлена на Рисунке 2.6.

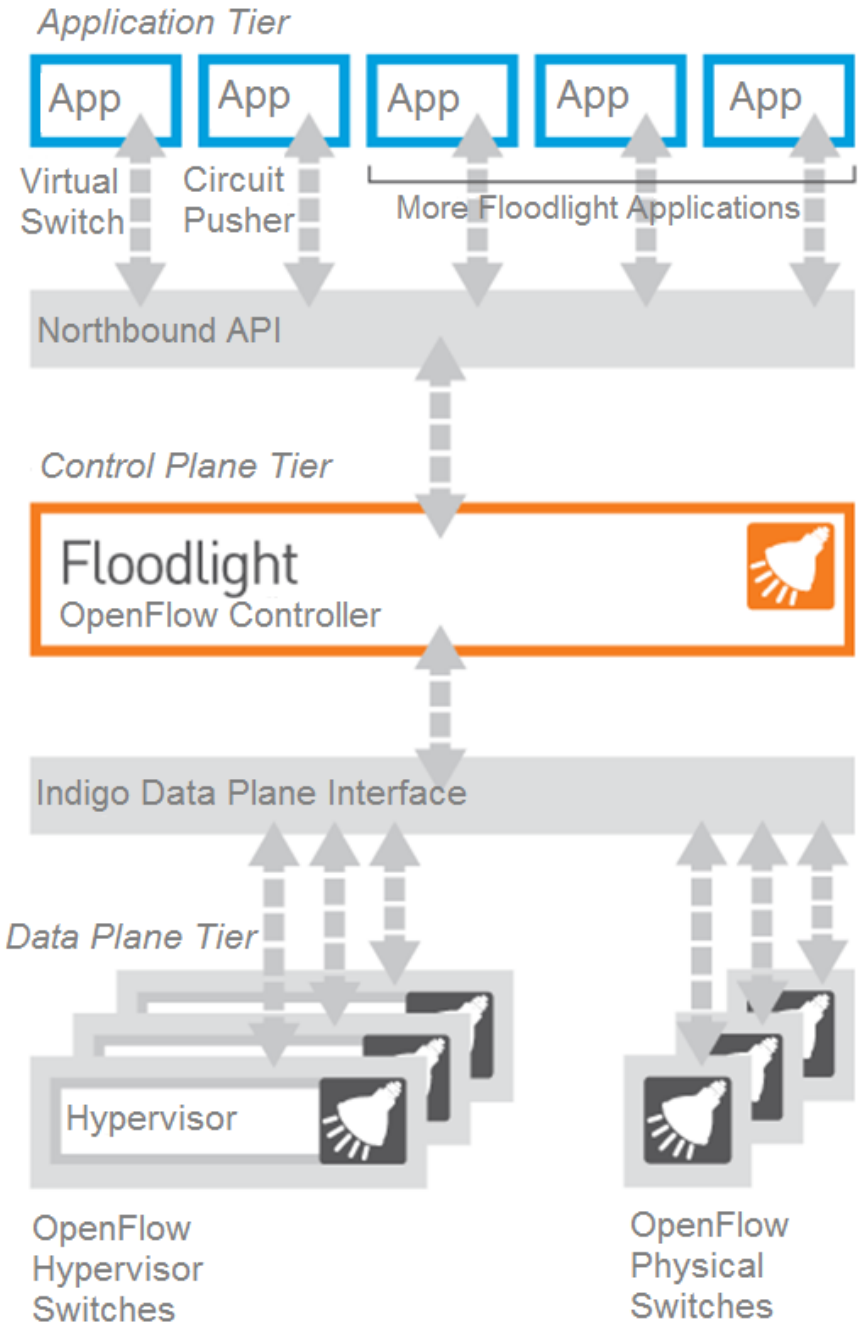

Рисунок 2.6 – Архитектура сети с контроллером FloodLight

2.6.6.10 Топологии OpenFlow-островов, поддерживаемые FloodLight. FloodLight может управлять OpenFlow островами, однако эти острова должны быть взаимодействовать с не OpenFlow сегментами посредством единственного канала. Вариант топологи OpenFlow-островов, поддерживаемых FloodLight, представлен на Рисунке 2.7.

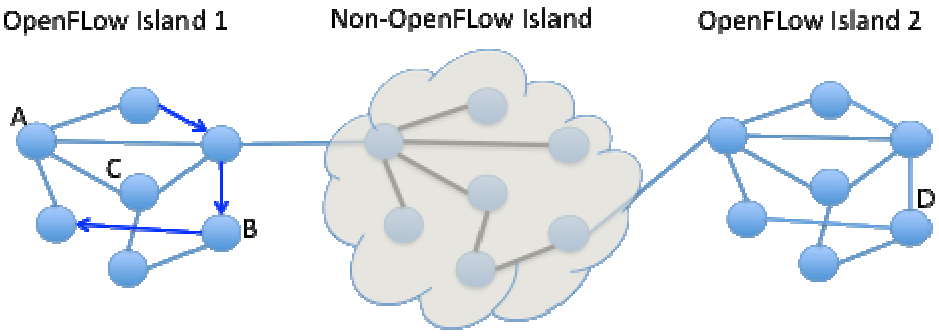

Рисунок 2.7 – Топология OpenFlow-островов, поддерживаемая FloodLight

На Рисунках 2.8 и 2.9 представлены варианты топологий OpenFlow-островов, неподдерживаемых FloodLight.

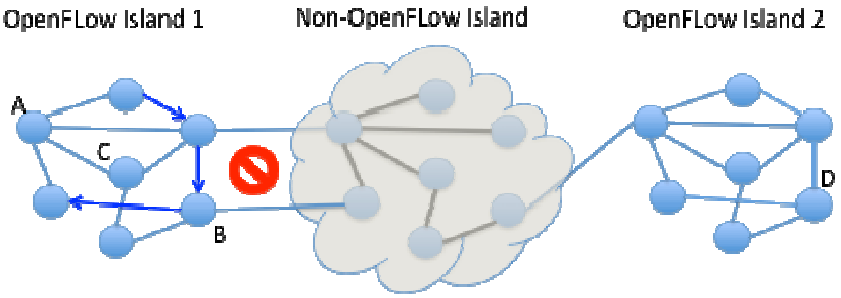

Рисунок 2.8 – Топология 2 OpenFlow-островов, не поддерживаемая FloodLight

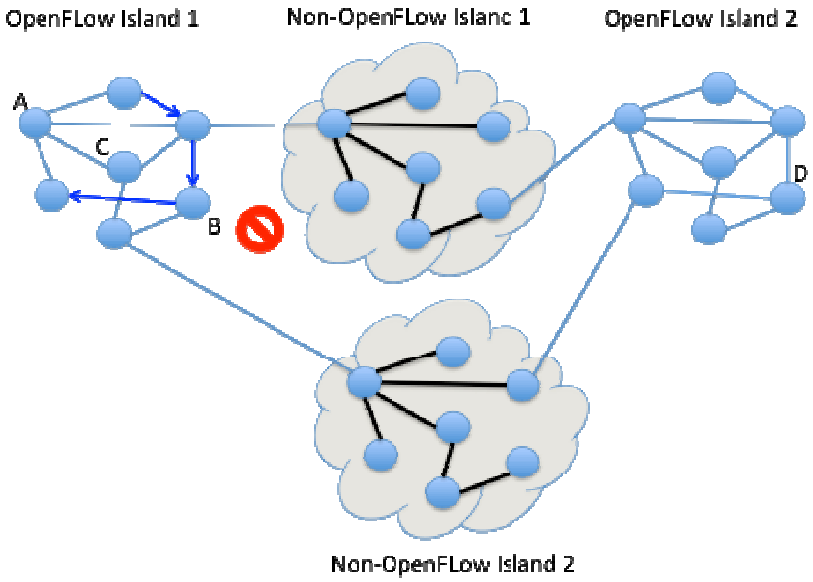

Рисунок 2.9 – Топология 3 OpenFlow-островов, не поддерживаемая FloodLight

2.6.6.11 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).

2.6.6.12 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).

2.6.6.13 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 формируется кольцевая топология или острова в форме кольцевой топологии.

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

2.6.6.14 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.

2.6.6.15 FloodLight поддерживает большинство современных OpenFlow виртуальных и аппаратных коммутаторов.

— Виртуальные: Open vSwitch (OVS).

— Аппаратные:

1) Arista 7050;
2) Brocade MLXe;
3) Brocade CER;
4) Brocade CES;
5) Dell S4810;
6) Dell Z9000;
7) Extreme Summit x440, x460, x670;
8) HP 3500, 3500yl, 5400zl, 6200yl, 6600, and 8200zl (the old-style L3 hardware match platform);
9) HP V2 line cards in the 5400zl and 8200zl (the newer L2 hardware match platform);

10) Huawei openflow-capable router platforms;

11) IBM 8264;

12) Juniper (MX, EX);

13) NEC IP8800;

14) NEC PF5240;

15) NEC PF5820;

16) NetGear 7328SO;

17) NetGear 7352SO;

18) Pronto (3290, 3295, 3780) - runs the shipping pica8 software.

Диаграмма текущей реализации FloodLight (от 12.09.2012) и взаимосвязи модулей представлена на Рисунке 2.10.

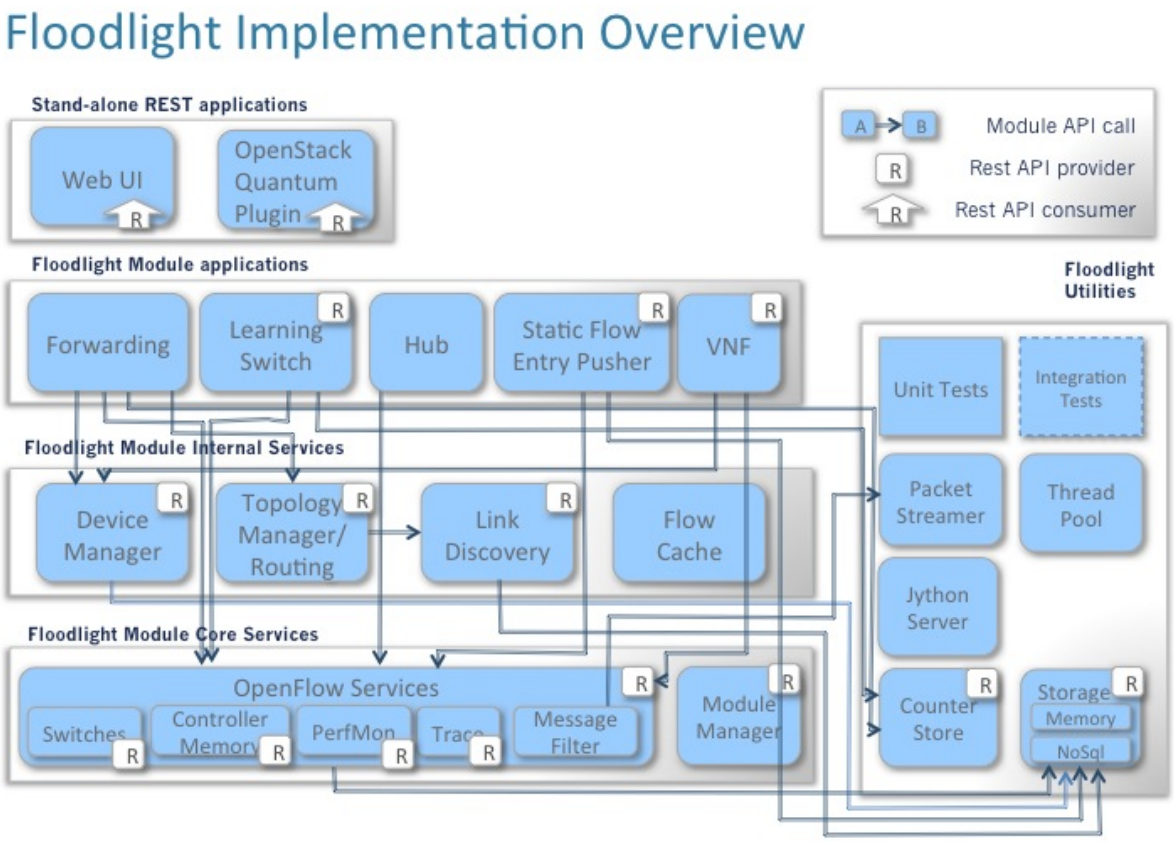

Рисунок 2.10 – Диаграмма текущей реализации FloodLight

2.6.6.16 Многопоточность в FloodLight. Многопоточность в FloodLight реализована через Netty библиотеку (поэтому все модули (и при разработке) должны быть с ориентацией на многопоточное исполнение (thread-safe)).

Netty (An asynchronous event-driven network application framework) – асинхронная утилита для сетевых приложений на основе событий. Структура данной утилиты представлена на Рисунке 2.11.

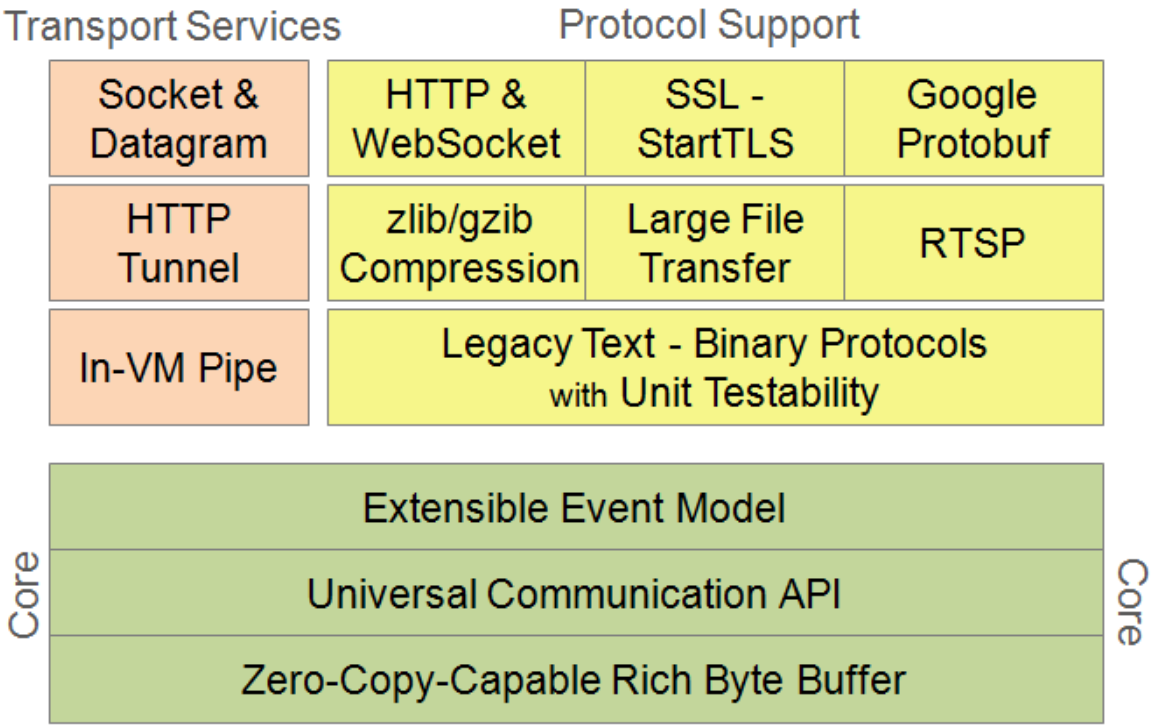

Рисунок 2.11 – Структура Netty

2.6.6.17 Используемая модель потоков в FloodLight:

1) Все межмодульное взаимодействие осуществляется через сервисы (меж-сервисные вызовы должны быть с ориентацией на многопоточное исполнение).
2) Обработка событий осуществляется в контексте потока публикующего объекта (не блокируются).
3) Существует сервис управления потоками (разрешает модулям разделять потоки).

4) Количество структур разделяемых данных защищается за счет блокировок.

5) Любой Java объект может быть событием

6) Применяются стандартные блокировки – синхронные.

2.6.7 Сетевая ОС Trema

2.6.7.1 Trema – это платформа для разработки OpenFlow контроллеров на языках C/Ruby, берущая своё начало в исследовательской лаборатории NEC, с целью использовать скорость C и продуктивность Ruby. Платформа Trema разрабатывалась для исследовательских и учебных учреждений, поэтому включает в себя интегрированную среду тестирования и отладки приложений. Принцип создания OpenFlow контроллера: платформа Trema с набором разработанных пользовательских модулей (приложений).

2.6.7.2 Архитектура и основные модули. Структура Trema является модульной и представлена на Рисунке 2.12.

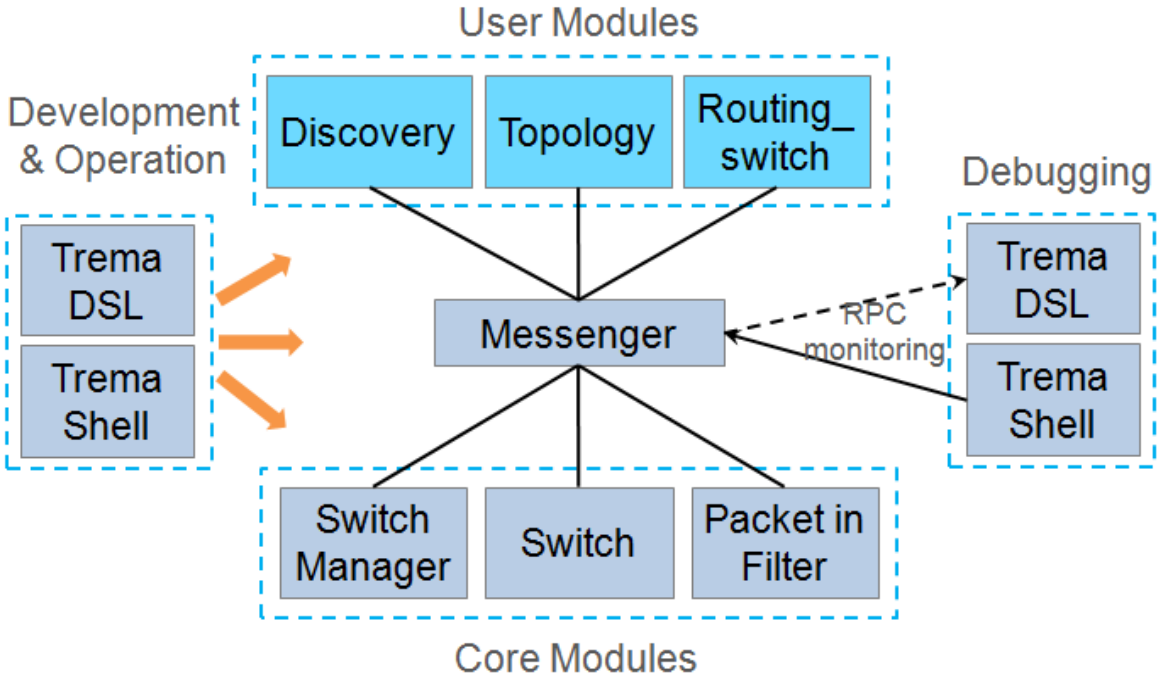

Рисунок 2.12 – Структура Trema

Trema – это платформа для создания OpenFlow контроллеров, которая включает в себя:

– *Компоненты конфигурации*, которые позволяют формировать конфигурационный файл для автоматического запуска приложений и модулей, фильтрации и роутинга сообщений, конфигурирований виртуальных сетей. Конфигурации описываются с помощью Domain Specific Language (DSL).

– Модули ядра Trema:

1) Компонент Switch Manager;
2) Компонент Switch;
3) Компонент Packet in filter;
4) Messenger – осуществляет взаимодействие между модулями Trema. В настоящее время реализовано point-to-point обмен сообщениями между модулями Trema на основе Unix-сокетов в рамках одного хоста. Планируется внедрение механизма публикация/подписка сообщений между приложениями среди множества хостов (как в HyperFlow).

– Дополнительные модули.

– Интегрированную среду для тестирования и отладки, представленную на Рисунке 2.13:

1) Эмулятор сети.
2) Инструменты отладки: Tremashark, Wireshark Plugin и средства логирования.

Таким образом, OpenFlow контроллер – это Trema и набор пользовательских модулей (приложений).

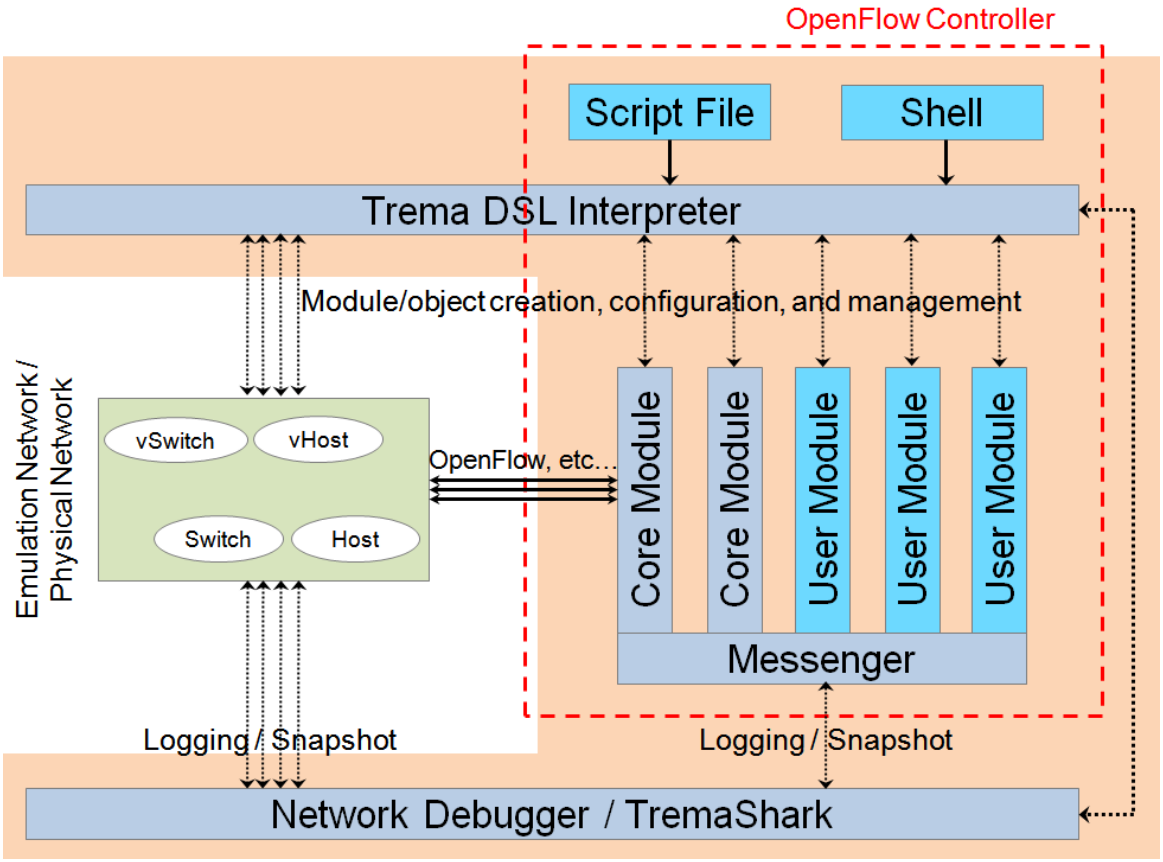

Рисунок 2.13 – OpenFlow контроллер и средства отладки Trema

Trema предлагает приложениям набор API на C и на Ruby: OpenFlow 1.0, Packet Parser, Logging, Timers, Messenger.

2.6.7.3 Приложения для Trema. Поскольку Trema имеет модульную структуру для обеспечения расширяемости (Multi-Process Model). Каждое приложение является процессом, поэтому приложения взаимодействуют между собой через IPC на основе сообщений. Для разработки приложений и модулей Trema могут использоваться разные языки: Ruby для обеспечения функциональности с простой и маленькой реализацией, либо C для обеспечения производительности, т.е. прототипировать и тестировать приложение на Ruby, а затем переписывать его на C.

Приложения для Trema.

– broadcast_helper;

- flow_dumper;
- flow_manager;
- learning_switch_memcached;
- load_balance_switch;
- multi_learning_switch_memcached;
- packetin_dispatcher;
- path_manager;
- redirectable_routing_switch;
- routing_switch;
- show_description;
- show_switch_features;
- simple_load_balancer;
- simple_multicast;
- sliceable_switch;
- topology;
- traffic_monitor_memcached.

2.6.7.4 Надежность: Trema является стабильной и динамической платформой, поддерживается защита контроллера от нестабильной работы модулей/приложений: базовая функциональность работает даже в случае сбоев в других модулях и нестабильные модули могут быть перезапущены после сбоев.

2.6.8 Сетевая ОС MUL

2.6.8.1 Контроллер MūL [12] разрабатывался для обеспечения *производительности* и *надежности* сетей, выполняющих критически важные задачи (с повышенными требованиями к надежности). MūL 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*.

2.6.8.2 Архитектура и основные компоненты. Основные компоненты Mul контроллера представлены на Рисунке 2.14.

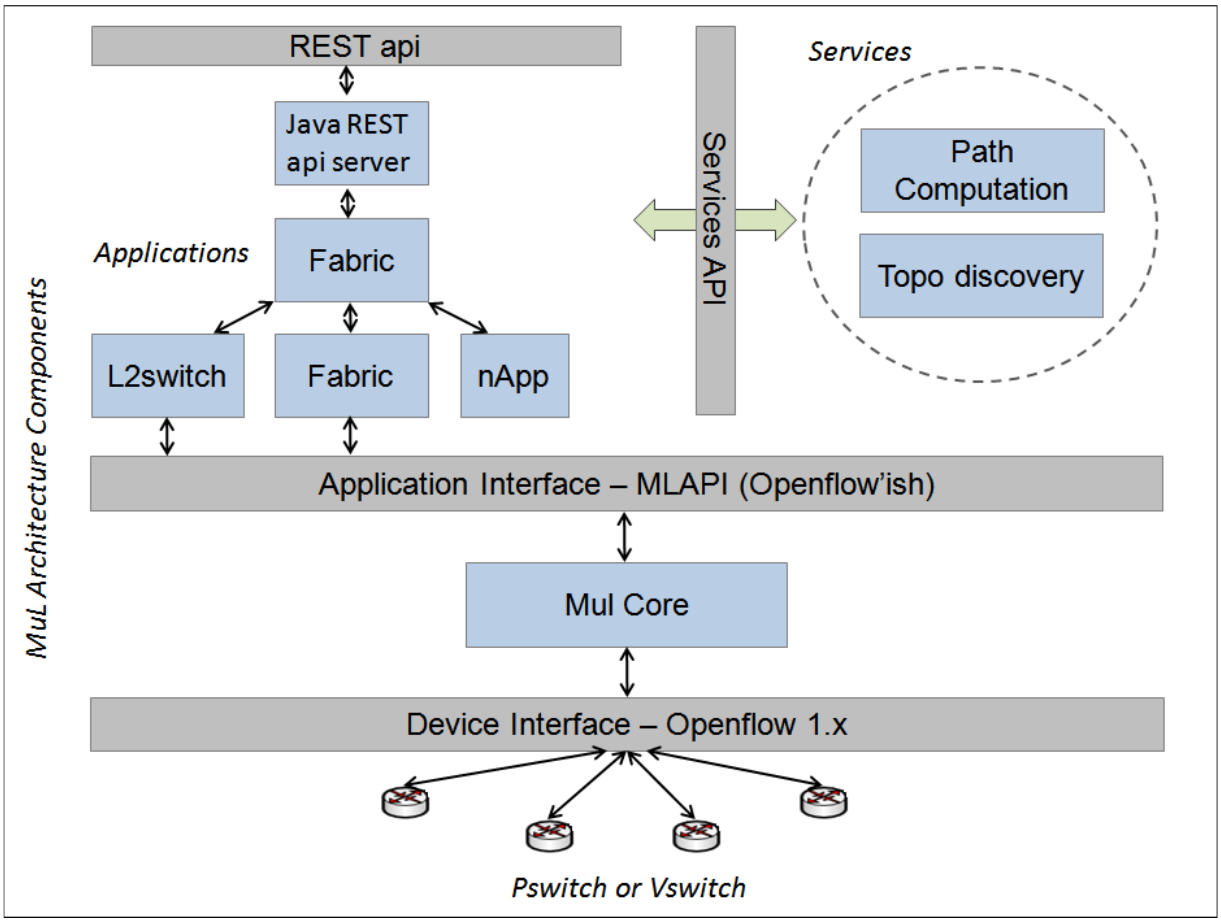

Рисунок 2.14 – Архитектура сетевой ОС Mul

2.6.8.3 Два режима установки Mul контроллера:

a) обычный;

b) с повышенной производительностью.

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

2.6.9 Сетевая ОС ONIX

2.6.9.1 Onix – первая распределенная сетевая ОС для управления ПКС сетью. Onix является закрытой разработкой. Onix создан на основе NOX Classic и реализован на языках C++, python и Java. Благодаря распределенной архитектуре Onix превосходит все остальные сетевые ОС по надежности и масштабируемости. Onix разрабатывался как универсальная платформа для использования в различных средах: корпоративных сетях, сетях ЦОД и облаках. Однако из-за большой сложности и громоздкости, Onix имеет трудности в развертывании. В настоящее время данный проект активно не развивается.

2.6.9.2 Архитектура сети под управлением Onix. В архитектуре выделяют четыре основных компонента в сети, управляемой Ониксом, согласно Рисунку 2.15:

— Физическая инфраструктура: Инфраструктура включает в себя сетевые коммутаторы и маршрутизаторы, а также любые другие сетевые элементы (например, балансировщики нагрузки), которые поддерживают интерфейс, позволяющий Ониксу читать и писать состояние, контролируя поведение элементов сети (например, таблица записей форвардинга). Этим элементам сети не нужно запускать какое-либо программное обеспечение, кроме того, что необходимо для поддержки этого интерфейса и (как описано в следующем абзаце) для обеспечения основных возможностей подключения.

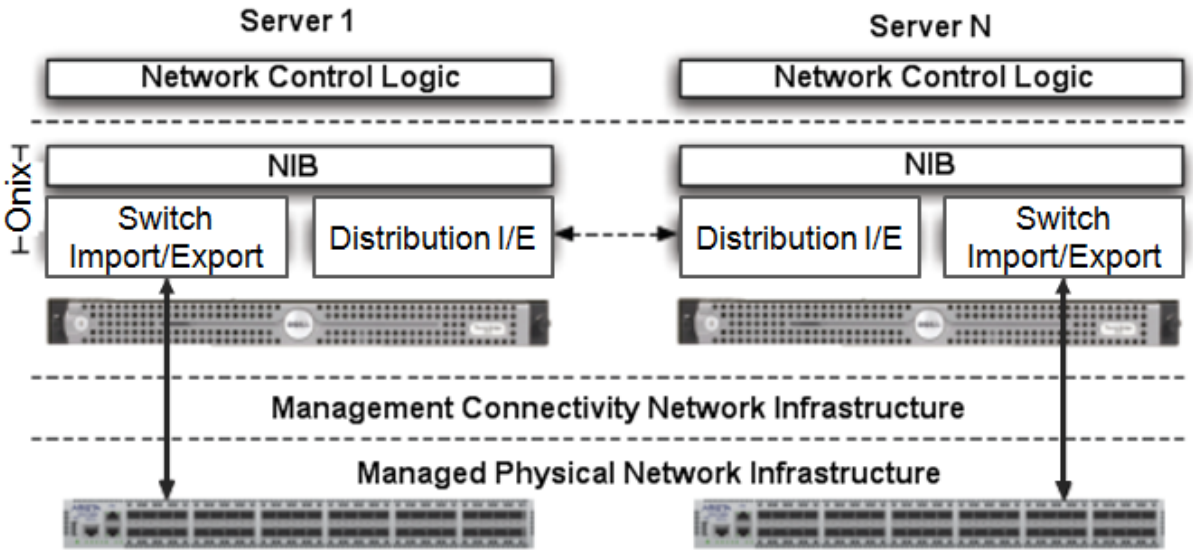

Рисунок 2.15 – Архитектура ONIX

– Инфраструктура подключения: Связь между физическим сетевым оборудованием и Ониксом («управляющий трафик» - control traffic) передается инфраструктурой подключения. Этот канал управления может быть реализован либо совместно с передачей данных (в которой управляющий трафик использует те же сетевые элементы форвардинга (коммутаторы), которые используются для трафика данных в сети), либо отдельно (отдельная физическая сеть используется для обработки управляющего трафика). Инфраструктура подключения должна поддерживать двустороннюю связь между экземплярами Оникс и коммутаторами, и, возможно, способствовать сходимости в случае отказа канала связи. Стандартные протоколы маршрутизации (например, IS-IS или OSPF) предназначены для создания и поддержания состояние пересылки (forwarding state) в инфраструктуре подключения.

– Оникс: Оникс является распределенной системой, которая работает на кластере из одного или нескольких физических серверов, каждый из которых может работать с несколькими экземплярами Оникс. Как платформа управления Оникс несет ответственность за программный доступ логики управления к сети (чтения и записи состояния сети). В целях расширения до очень больших сетей (млн. портов), а также обеспечения необходимой устойчивости/живучести (resilience) для

развертывания производственных сетей, экземпляр Оникс также несет ответственность за распространение состояние сети в другие экземпляры в пределах кластера.

— Управляющая логика: Логика управления сетью реализуется поверх API Оникс. Это логика управления определяет желаемое поведение сети; Оникс только предоставляет примитивы, необходимые для доступа к соответствующим состояниям сети.

Это четыре основных компонента SDN сети под управлением Onix.

2.6.10 Сетевая ОС Kandoo

2.6.10.1 Для обеспечения масштабируемости контроллера помимо наращивания производительности серверов контроллера можно попытаться снизить нагрузку на контроллер за счет уменьшения количества обрабатываемых OpenFlow сообщений. Один из для реализации этой идеи – обрабатывать часто возникающие события на месте их возниконовения, то есть уровне передачи данных в коммутаторах. Однако это требует внесения изменений в коммутаторы. В работе [13] предлагается использовать двухуровневую иерархическую архитектуру, в которой обработка локальных наиболее частых событий происходит на нижнем уровне (на уровне коммутаторов).

При разработке платформы управления Kandoo преследовались две основные цели:

— Kandoo должен быть совместим со стандартом OpenFlow, т.е. не должна вноситься какая-либо новая функциональность в коммутаторы и протокол OpenFlow.

— Kandoo автоматически распределяет сетевые приложения без участия администратора, т.е. сетевые приложения Kandoo не осведомлены о том, как они размещены в сети, и разработчики приложений могут предполагать, что их сетевое приложение будут запущены на централизованном контроллере.

2.6.10.2 Архитектура Kandoo. Kandoo имеет двухуровневую иерархию контроллеров согласно Рисунку 2.16:

– *Нижний уровень* – множество локальных контроллеров (local controllers), которые не взаимодействуют между собой и не имеют представления обо всей сети и ее состоянии.

– *Верхний уровень* – единый логически централизованный контроллер (root controller), который имеет представление о состоянии всей сети.

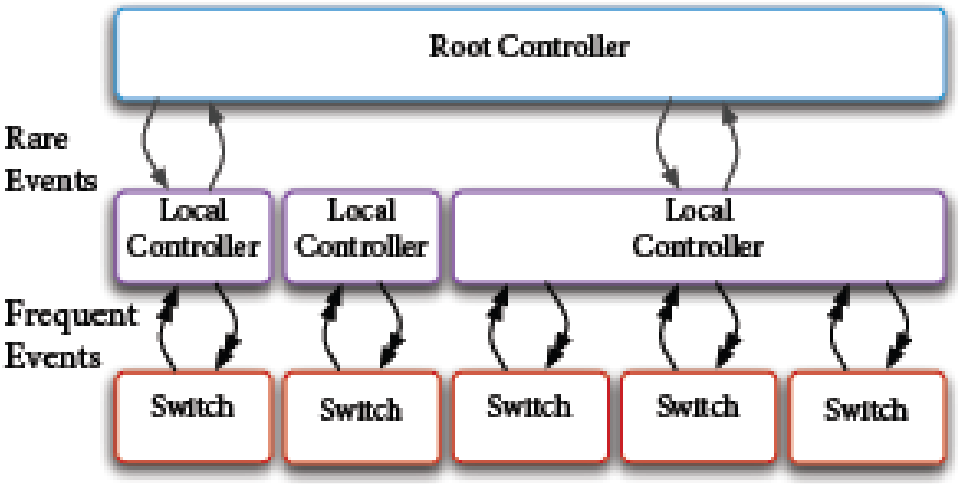

Рисунок 2.16 – Архитектура Kandoo

Все приложения разделяются на две группы, соответственно Рисунку 2.17:

– Локальные сетевые приложения (local control application) (которые например, могут функционировать, используя состояние одного коммутатора). Они обрабатывают наиболее частые события локально, и, тем самым, защищают root контроллер от перегрузки. Выполняются на локальных контроллерах.

– Нелокальные сетевые приложения (non-local control application) – т.е. приложения, которым требуется доступ к представлению сети.

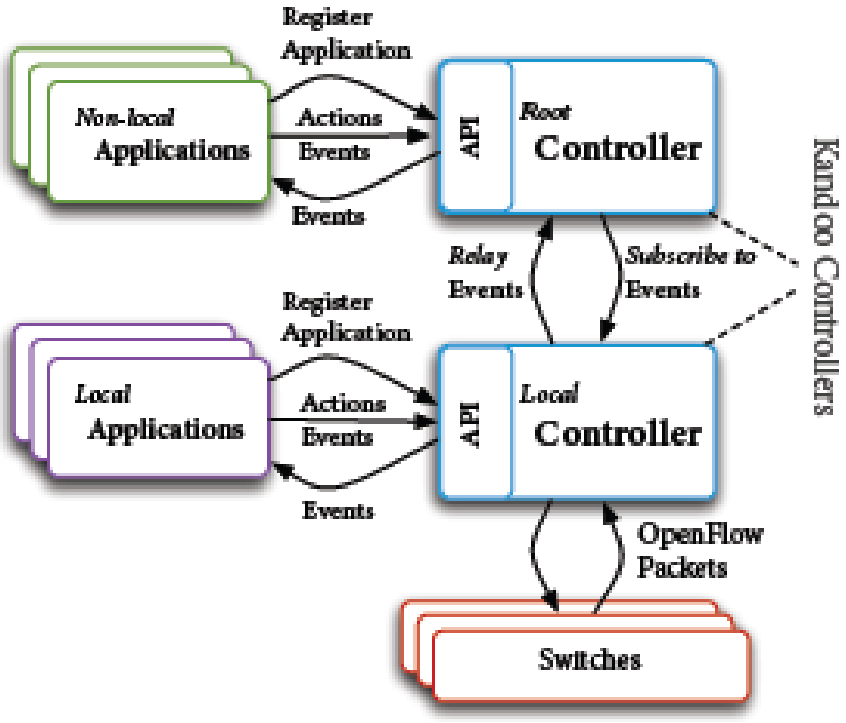

Рисунок 2.17 – Взаимодействие контроллеров и приложений в Kandoo

2.6.10.3 Производительность. Более 90% событий обрабатывается на локальных контроллерах Kandoo. Один локальный контроллер может обрабатывать пропускную способность более чем 1M pkt-in в секунду от 512 коммутаторов, используя один поток Intel Xeon (R) E7-4807 с 6 ядрами.

2.6.10.4 Надежность. Для повышения надежности root-контроллер регистрирует себя на каждом коммутаторе как slave-контроллер согласно протоколу OpenFlow 1.2. Таким образом, за счет двухуровневой архитектуры в случае обнаружения отказа локального контроллера, root-контроллер перехватывает управление коммутатором отказавшего локального контроллера.

2.6.10.5 Безопасность. Нет данных.

2.6.10.6 Масштабируемость. Эксперименты с Kandoo подтвердили, что Kandoo масштабируется на порядок лучше по сравнению с обычными OpenFlow сетями. Однако разработчики рекомендуют использовать проактивную установку network state.

2.6.10.7 Другие особенности реализации. Kandoo реализован на C, C++ и Python. Реализация позволяет динамически подгружать плагины, реализованные на C, Python и Java. Поддерживает RPC API. Реализация модульная. Kandoo поддерживает протокол OpenFlow 1.0 (1.1 и 1.2 в разработке). Для приложений поддерживается центральный репозиторий приложений и простая система управления пакетами, которая поддерживает зависимости между приложениями. Исходные тексты не доступны.

2.7 Выводы

По результатам проведенного анализа существляющих сетевых ОС можно сделать следующие выводы:

— Большинство существующих сетевых ОС написаны на языках высокого уровня: C, C++, Python, Java, Rybu.

— Кроссплатформенными являются только Java-контроллера FloodLight, Beacon и Maestro, остальные работают на Linux-платформах.

— Все рассмотренные сетевые ОС являются многопоточными, кроме POX.

— Все сетевые ОС поддерживают протокол OpenFlow версии 1.0 и, следовательно, работают со всеми существующими программными и аппаратными коммутаторами.

— Все контроллеры разрабатывались с совершенно разными целями, поэтому нет универсального контроллера, который позволяет решать все задачи в разных средах (WAN, центры обработки данных). Каждый контроллер имеет свои достоинства и недостатки. Например, контроллер POX позволяет быстро разрабатывать приложения для него по сравнению с другими. Однако C, C++, Java – контроллеры, требующие более трудоемкого написания приложений, существенно выигрывают по производительности. Распределенные контроллеры имеют более сложную архитектуру и требуют более сложных алгоритмов, однако позволяют повысить производительность, масштабируемость и надежность уровня управления ПКС.

— С точки зрения архитектуры все контроллеры используют модульный принцип, позволяющий наращивать сетевые сервисы для приложений внутри сетевой ОС, расширяя функциональные возможности контроллера. Все контроллеры имеют очень схожий базовый набор сервисов: сервисы для обработки OpenFlow сообщений, поддержки соединений с коммутаторами, механизмы подписки/генерации событий для модулей и приложений, механизмы управления сетевыми устройствами и построения топологии сети.

— Для всех контроллеров на текущий момент созданы репозитории самых разнообразных приложений, большинство из которых носит исследовательский или учебный характер. Одним из самых развивающихся и самых разнообразных является репозиторий контроллера FloodLight.

— С точки зрения надежности, существующие контроллеры не способны противостоять угрозам, связанным с отказами контроллеров и их приложений, поскольку такие механизмы в них не заложены. Исключением являются распределенные контроллеры: Onix, которые способен передавать управление с отказавшего узла контроллера на исправный, и Kandoo, в котором в случае отказа локального контроллера управление передается на верхний уровень – корневому контроллеру.

— О производительности и масштабируемости контроллеров по доступным материалам и интернет ресурсам судить крайне сложно. Это связано со следующими проблемами: устаревание исследований и их воспроизводимость – проекты развиваются, и характеристики контроллеров изменяются, различия в методиках исследований, необъективность результатов.

На основе сформулированных требований к контроллерам и проведенного анализа получены результаты, представленные в Таблице 2.11. В данной таблице курсивом выделены контроллеры, представляющие наибольший интерес по результатм сравнения.

Таблица 2.11 – Сравнение контроллеров с открытым исходным кодом

	Тип контроллера	Открытость	Развитие	OF >= 1.0	Плафторма Linux	Многопоточность
1	NOX Classic	+	–	+	+	–
2	NOX	+	–	+	+	+
3	POX	+	+	+	+	–
4	SNAC	+	–	+	+	–
5	*Beacon*	+	+	+	+	+
6	Maestro	+	–	+	+	+
7	*Floodlight*	+	+	+	+	+
8	*Trema*	+	+	+	+	+
9	*MUL*	+	+	+	+	+
10	ONIX	–	–	+	+	+
11	Kandoo	–	н/д	+	+	+

По итогам исследования существующих сетевых ОС с открытым исходным кодом, наиболее перспективными и активно развивающимися разработками являются: контроллеры *Beacon*, *FloodLight* (на языке Java), *Trema* и *Mul* (на языке C).

3 ОБОСНОВАНИЕ И ВЫБОР КРИТЕРИЕВ ЭФФЕКТИВНОСТИ УПРАВЛЕНИЯ СЕТЕВОЙ ИНФРАСТРУКТУРОЙ КС И ПОТОКАМИ ДАННЫХ

3.1 Общие положения

Основное требование, предъявляемое к компьютерной сети, заключается в выполнении тех функций и предоставления тех услуг, для которых она предназначена. Например, предоставление доступа к ресурсам Интернет, ERP-системе предприятия, обмен электронной почтой, обеспечение IP-телефонии и так далее. Таким образом, можно говорить об эффективности управления сетевой инфраструктурой КС и потоками данных в ней только в том случае, если она успешно выполняет задачи, для которых эта сеть создавалась.

Дополнительными важными требованиями, предъявляемыми к КС являются производительность, надежность, совместимость, управляемость, защищенность, расширяемость и масштабируемость, связанные с качеством выполнения задач и предоставления услуг сетью [14]. Каждое из этих требований определяется конкретными количественными характеристиками. Однако, эффективность управления сетевой инфраструктурой и потоками данных в сети характеризуются, главным образом, показателями производительности и надежности функционирования сети.

Средства управления сетями представляют собой системы, осуществляющие наблюдение, контроль и управление каждым элементом сети — от простейших до самых сложных устройств, при этом такая система рассматривает сеть как единое целое, а не как разрозненный набор отдельных устройств.

Управляемость сети подразумевает возможность централизованно контролировать состояние основных элементов сети, выявлять и решать проблемы, возникающие при работе сети, выполнять анализ производительности и планировать развитие сети. Хорошая система управления наблюдает за сетью и, обнаружив проблему, активизирует определенное действие, исправляет ситуацию и уведомляет администратора о том, что произошло и какие шаги предприняты.

Одновременно с этим система управления должна накапливать данные, на основании которых можно планировать развитие сети. Наконец, система управления должна быть независимой от производителя и обладать удобным интерфейсом, позволяющим выполнять все действия с одной консоли.

Решая тактические задачи, администраторы и технический персонал сталкиваются с ежедневными проблемами обеспечения работоспособности сети. Эти задачи требуют быстрого решения, персонал, обслуживающий сеть, должен оперативно реагировать на сообщения о неисправностях, поступающих от пользователей или автоматических средств управления сетью.

По мере роста сети, становятся заметны общие проблемы производительности, конфигурирования сети, обработки сбоев и безопасности данных, требующие стратегического подхода, то есть планирования сети. Планирование, кроме этого, включает прогноз изменений требований пользователей к сети, вопросы применения новых приложений, новых сетевых технологий и т. п.

Необходимость в системе управления особенно ярко проявляется в больших сетях: корпоративных или глобальных. Без системы управления в таких сетях требуется присутствие квалифицированных специалистов по эксплуатации в каждом здании каждого города, где установлено оборудование сети, что в итоге приводит к необходимости содержания огромного штата обслуживающего персонала.

В настоящее время в области систем управления сетями много нерешенных проблем. Явно недостаточно действительно удобных, компактных и многопротокольных средств управления сетью. Большинство существующих средств вовсе не управляют сетью, а всего лишь осуществляют наблюдение за ее работой. Они следят за сетью, но не выполняют активных действий, если с сетью что-то произошло или может произойти. Мало масштабируемых систем, способных обслуживать как сети масштаба отдела, так и сети масштаба предприятия, — очень многие системы управляют только отдельными элементами сети и не анализируют

способность сети выполнять качественную передачу данных между конечными пользователями.

Благодаря системам управления становятся возможными перераспределение нагрузки при сбоях оборудования, оптимизация трафика для повышения отдачи от существующих сетей и планомерное развитие сети за счет анализа трафика и моделирования потребностей пользователей и информационных систем в будущем. Средства управления и мониторинга состояния инфраструктуры позволяют избежать потерь связи или падения производительности сетевых сервисов за счет постоянного отслеживания и управления движением трафика.

Таким образом, ключевыми задачами систем управления сетями являются – обеспечение требуемого уровня производительности и надежности функционирования сети, а эффективность их управления определяется показателями производительности и надежности сети.

Также в качестве показателя эффективности управления могут выступать экономические показатели, например, стоимость обслуживания сетевой инфраструктуры, стоимость предоставления тех или иных услуг. При использовании наиболее продвинутых систем управления стоимость обслуживания и стоимость услуг должны снижаться.

Рассмотрим подробнее характеристики производительности и надежности компьютерных сетей.

3.2 Показатели производительности сети

3.2.1 Основные показатели производительности

Потенциально высокая производительность — это одно из основных преимуществ распределенных систем, к которым относятся компьютерные сети. Это свойство обеспечивается принципиальной, но, к сожалению, не всегда практически реализуемой возможностью распределения работ между несколькими компьютерами сети.

Производительность сети измеряется с помощью показателей двух типов - временных, оценивающих задержку, вносимую сетью при выполнении обмена данными, и показателей пропускной способности, отражающих количество информации, переданной сетью в единицу времени [15]. Эти два типа показателей являются взаимно обратными, и, зная один из них, можно вычислить другой для канала связи.

Таким образом, основными характеристиками производительности сети являются:

1) время реакции;
2) скорость передачи данных;
3) пропускная способность;
4) задержка передачи и вариация задержки передачи.

3.2.2 Время реакции

Обычно в качестве временной характеристики производительности сети используется такой показатель как время реакции. Термин "время реакции" может использоваться в очень широком смысле, поэтому в каждом конкретном случае необходимо уточнить, что понимается под этим термином.

Время реакции сети является интегральной характеристикой производительности сети с точки зрения пользователя. Именно эту характеристику имеет в виду пользователь, когда говорит: "Сегодня сеть работает медленно".

В общем случае, время реакции определяется как интервал времени между возникновением запроса пользователя к кому-либо сетевому сервису и получением ответа на этот запрос соответственно Рисунку 3.1. Очевидно, что смысл и значение этого показателя зависят от типа сервиса, к которому обращается пользователь, от того, какой пользователь и к какому серверу обращается, а также от текущего состояния других элементов сети – загруженности сегментов, через которые

проходит запрос, загруженности сервера и прочих показателей (каких именно – в данном контексте несущественно).

Поэтому имеет смысл использовать также и средневзвешенную оценку времени реакции сети, усредняя этот показатель по пользователям, серверам и времени дня (от которого в значительной степени зависит загрузка сети).

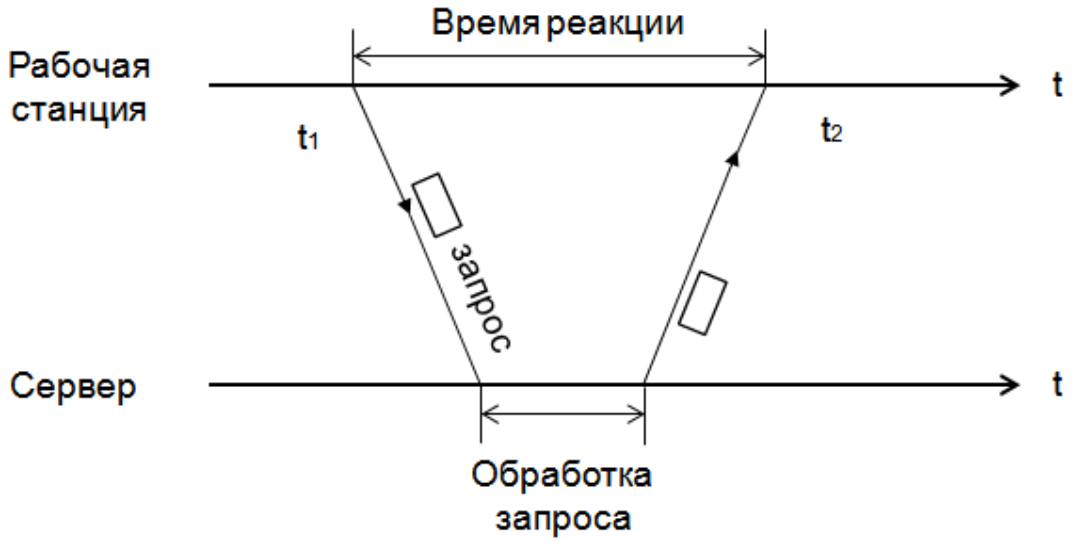

Рисунок 3.1 – Время реакции сети

Время реакции сети обычно складывается из нескольких составляющих. В общем случае в него входит:

1) время подготовки запросов на клиентском компьютере;
2) время передачи запросов между клиентом и сервером через сегменты сети и промежуточное коммуникационное оборудование;
3) время обработки запросов на сервере;
4) время передачи ответов от сервера клиенту и время обработки получаемых от сервера ответов на клиентском компьютере.

Очевидно, что разложение времени реакции на составляющие пользователя не интересует — ему важен конечный результат. Однако для сетевого специалиста очень важно выделить из общего времени реакции составляющие, соответствующие

этапам собственно сетевой обработки данных, — передачу данных от клиента к серверу через сегменты сети и коммуникационное оборудование. Знание сетевых составляющих времени реакции позволяет оценить производительность отдельных элементов сети, выявить узкие места и при необходимости выполнить модернизацию сети для повышения ее общей производительности.

Было рассмотрено несколько примеров на основе схемы, представленной на Рисунке 3.2.

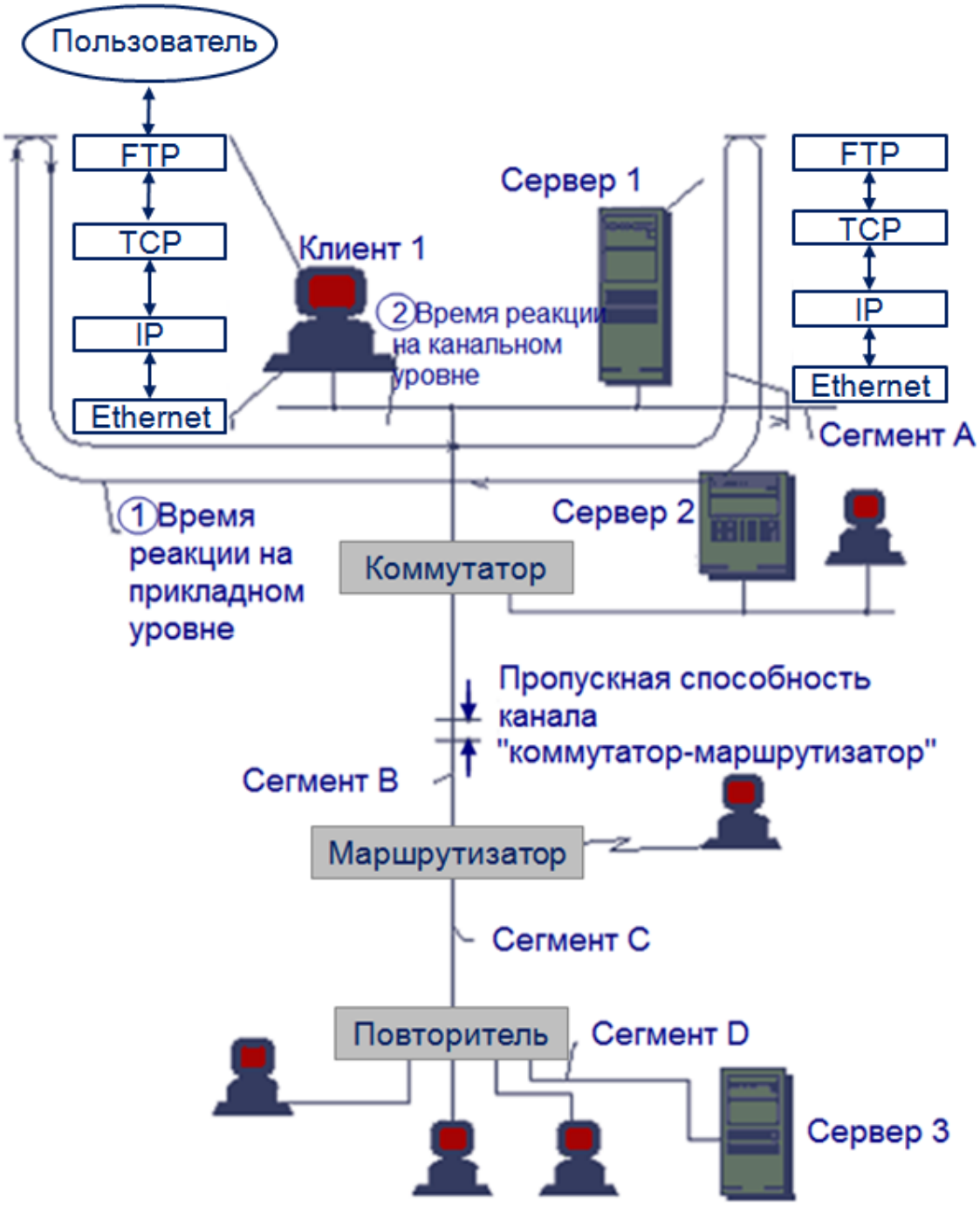

Рисунок 3.2 – Показатели производительности сети

В примере 1 под временем реакции понимается время, которое проходит с момента обращения пользователя к сервису FTP для передачи файла с сервера 1 на клиентский компьютер 1 до момента завершения этой передачи. Очевидно, что это время имеет несколько составляющих. Наиболее существенный вклад вносят такие

составляющие времени реакции как: время обработки запросов на передачу файла на сервере, время обработки получаемых в пакетах IP частей файла на клиентском компьютере, время передачи пакетов между сервером и клиентским компьютером по протоколу Ethernet в пределах одного коаксиального сегмента. Можно было бы выделить еще более мелкие этапы выполнения запроса, например, время обработки запроса каждым из протоколов стека TCP/IP на сервере и клиенте.

Для конечного пользователя, таким образом, определенное время реакции является понятным и наиболее естественным показателем производительности сети (размер файла, который вносит некоторую неопределенность в этот показатель, можно зафиксировать, оценивая время реакции при передаче, например, одного мегабайта данных). Однако сетевого специалиста интересует в первую очередь производительность собственно сети, поэтому для более точной ее оценки целесообразно вычленить из времени реакции составляющие, соответствующие этапам несетевой обработки данных – например, поиску нужной информации на диске, записи ее на диск. Полученное в результате таких сокращений время можно считать другим определением времени реакции сети на прикладном уровне.

Вариантами этого критерия могут служить времена реакции, измеренные при различных, но фиксированных состояниях сети:

А) Полностью ненагруженная сеть. Время реакции измеряется в условиях, когда к серверу 1 обращается только клиент 1, то есть на сегменте сети, объединяющем сервер 1 с клиентом 1, нет никакой другой активности - на нем присутствуют только кадры сессии FTP, производительность которой измеряется. В других сегментах сети трафик может циркулировать, главное - чтобы его кадры не попадали в сегмент, в котором проводятся измерения. Так как ненагруженный сегмент в реальной сети - явление экзотическое, то данный вариант показателя производительности имеет ограниченную применимость - его хорошие значения говорят только о том, что программное обеспечение и аппаратура данных двух узлов и сегмента обладают необходимой производительностью для работы в облегченных условиях. Для работы в реальных условиях, когда будет иметь место

борьба за разделяемые ресурсы сегмента с другими узлами сети, производительность тестируемых элементов сети может оказаться недостаточной.

В) Нагруженная сеть. Это более интересный случай проверки производительности сервиса FTP для конкретных сервера и клиента. Однако при измерении критерия производительности в условиях, когда в сети работают и другие узлы и сервисы, возникают свои сложности - в сети может существовать слишком большое количество вариантов нагрузки, поэтому главное при определении критериев такого сорта - проведение измерений при некоторых типовых условиях работы сети. Так как трафик в сети носит пульсирующий характер и характеристики трафика существенно изменяются в зависимости от времени дня и дня недели, то определение типовой нагрузки - процедура сложная, требующая длительных измерений на сети. Если же сеть только проектируется, то определение типовой нагрузки еще больше усложняется.

В примере 2 критерием производительности сети является время задержки между передачей кадра Ethernet в сеть сетевым адаптером клиентского компьютера 1 и поступлением его на сетевой адаптер сервера 3. Этот критерий также относится к критериям типа "время реакции", но соответствует сервису нижнего - канального уровня. Так как протокол Ethernet - протокол дейтаграммного типа, то есть без установления соединений, для которого понятие "ответ" не определено, то под временем реакции в данном случае понимается время прохождения кадра от узла-источника до узла-получателя. Задержка передачи кадра включает в данном случае время распространения кадра по исходному сегменту, время передачи кадра коммутатором из сегмента A в сегмент B, время передачи кадра маршрутизатором из сегмента B в сегмент C и время передачи кадра из сегмента C в сегмент D повторителем. Критерии, относящиеся к нижнему уровню сети, хорошо характеризуют качества транспортного сервиса сети и являются более информативными для сетевых интеграторов, так как не содержат избыточную для них информацию о работе протоколов верхних уровней.

При оценке производительности сети не по отношению к отдельным парам узлов, а ко всем узлам в целом используются критерии двух типов: средневзвешенные и пороговые.

Средневзвешенный критерий представляет собой сумму времен реакции всех или некоторых узлов при взаимодействии со всеми или некоторыми серверами сети по определенному сервису. Если усреднение производится и по сервисам, то добавляется еще одно суммирование - по количеству учитываемых сервисов. Оптимизация сети по данному критерию заключается в нахождении значений параметров, при которых критерий имеет минимальное значение или, по крайней мере, не превышает некоторое заданное число.

Пороговый критерий отражает наихудшее время реакции по всем возможным сочетаниям клиентов, серверов и сервисов. Оптимизация также может выполняться с целью минимизации критерия, или же с целью достижения им некоторой заданной величины, признаваемой разумной с практической точки зрения.

Чаще применяются пороговые критерии оптимизации, так как они гарантируют всем пользователям некоторый удовлетворительный уровень реакции сети на их запросы. Средневзвешенные критерии могут дискриминировать некоторых пользователей, для которых время реакции слишком велико, при том, что при усреднении получен вполне приемлемый результат.

Можно применять и более дифференцированные по категориям пользователей и ситуациям критерии. Например, можно поставить перед собой цель гарантировать любому пользователю доступ к серверу, находящемуся в его сегменте, за время, не превышающее 5 секунд, к серверам, находящимся в его сети, но в сегментах, отделенных от его сегмента коммутаторами, за время, не превышающее 10 секунд, а к серверам других сетей - за время до 1 минуты.

3.2.3 Скорость передачи данных

Производительность сети может характеризоваться также скоростью передачи данных. Скорость передачи данных может быть мгновенной, максимальной и средней.

– Средняя скорость вычисляется путем деления общего объема переданных данных на время их передачи, причем выбирается достаточно длительный промежуток времени — час, день или неделя;

– Мгновенная скорость отличается от средней тем, что для усреднения выбирается очень маленький промежуток времени — например, 10 мс или 1 с;

– Максимальная скорость — это наибольшая скорость, зафиксированная в течение периода наблюдения.

Чаще всего при проектировании, настройке и оптимизации сети используются такие показатели, как средняя и максимальная скорость. Средняя скорость, с которой обрабатывает трафик отдельный элемент или сеть в целом, позволяет оценить работу сети на протяжении длительного времени, в течение которого в силу закона больших чисел пики и спады интенсивности трафика компенсируют друг друга. Максимальная скорость позволяет оценить, как сеть будет справляться с пиковыми нагрузками, характерными для особых периодов работы, например в утренние часы, когда сотрудники предприятия почти одновременно регистрируются в сети и обращаются к разделяемым файлам и базам данных. Обычно при определении скоростных характеристик некоторого сегмента или устройства в передаваемых данных не выделяется трафик какого-то определенного пользователя, приложения или компьютера — подсчитывается общий объем передаваемой информации. Тем не менее, для более точной оценки качества обслуживания такая детализация желательна, и в последнее время системы управления сетями все чаще позволяют ее выполнять.

3.2.4 Пропускная способность

3.2.4.1 Основная задача, для решения которой строится любая сеть - быстрая передача информации между компьютерами.

Пропускная способность уже не является, подобно времени реакции или скорости прохождения данных по сети, пользовательской характеристикой, так как она говорит о скорости выполнения внутренних операций сети — передачи пакетов данных между узлами сети через различные коммуникационные устройства. Зато она непосредственно характеризует качество выполнения основной функции сети — транспортировки сообщений — и поэтому чаще используется при анализе производительности сети, чем время реакции или скорость. Поэтому критерии, связанные с пропускной способностью сети или части сети, хорошо отражают качество функционирования сети.

Существует большое количество вариантов определения критериев этого вида, точно также, как и в случае критериев класса "время реакции". Эти варианты могут отличаться друг от друга: выбранной единицей измерения количества передаваемой информации, характером учитываемых данных - только пользовательские или же пользовательские вместе со служебными, количеством точек измерения передаваемого трафика, способом усреднения результатов на сеть в целом. Рассмотрим различные способы построения критерия пропускной способности более подробно.

3.2.4.2 Критерии, отличающиеся единицей измерения передаваемой информации. В качестве единицы измерения передаваемой информации обычно используются пакеты (или кадры, далее эти термины будут использоваться как синонимы) или биты. Соответственно, пропускная способность измеряется в пакетах в секунду или же в битах в секунду.

Так как вычислительные сети работают по принципу коммутации пакетов (или кадров), то измерение количества переданной информации в пакетах имеет смысл, тем более что пропускная способность коммуникационного оборудования,

работающего на канальном уровне и выше, также чаще всего измеряется в пакетах в секунду. Однако, из-за переменного размера пакета (это характерно для всех протоколов за исключением ATM, имеющего фиксированный размер пакета в 53 байта), измерение пропускной способности в пакетах в секунду связано с некоторой неопределенностью - пакеты какого протокола и какого размера имеются в виду? Чаще всего подразумевают пакеты протокола Ethernet, как самого распространенного, имеющие минимальный для протокола размер в 64 байта (без преамбулы). Пакеты минимальной длины выбраны в качестве эталонных из-за того, что они создают для коммуникационного оборудования наиболее тяжелый режим работы - вычислительные операции, производимые с каждым пришедшим пакетом, в очень слабой степени зависят от его размера, поэтому на единицу переносимой информации обработка пакета минимальной длины требует выполнения гораздо больше операций, чем для пакета максимальной длины.

Измерение пропускной способности в битах в секунду (для локальных сетей более характерны скорости, измеряемые в миллионах бит в секунду - Мб/с) дает более точную оценку скорости передаваемой информации, чем при использовании пакетов.

3.2.4.3 Критерии, отличающиеся учетом служебной информации. В любом протоколе имеется заголовок, переносящий служебную информацию, и поле данных, в котором переносится информация, считающаяся для данного протокола пользовательской. Например, в кадре протокола Ethernet минимального размера 46 байт (из 64) представляют собой поле данных, а оставшиеся 18 являются служебной информацией. При измерении пропускной способности в пакетах в секунду отделить пользовательскую информацию от служебной невозможно, а при побитовом измерении - можно.

Если пропускная способность измеряется без деления информации на пользовательскую и служебную, то в этом случае нельзя ставить задачу выбора протокола или стека протоколов для данной сети. Это объясняется тем, что даже если при замене одного протокола на другой мы получим более высокую

пропускную способность сети, то это не означает, что для конечных пользователей сеть будет работать быстрее - если доля служебной информации, приходящаяся на единицу пользовательских данных, у этих протоколов различная (а в общем случае это так), то можно в качестве оптимального выбрать более медленный вариант сети. Если же тип протокола не меняется при настройке сети, то можно использовать и критерии, не выделяющие пользовательские данные из общего потока.

При тестировании пропускной способности сети на прикладном уровне легче всего измерять как раз пропускную способность по пользовательским данным. Для этого достаточно измерить время передачи файла определенного размера между сервером и клиентом и разделить размер файла на полученное время. Для измерения общей пропускной способности необходимы специальные инструменты измерения - анализаторы протоколов или SNMP или RMON агенты, встроенные в операционные системы, сетевые адаптеры или коммуникационное оборудование.

3.2.4.4 Критерии, отличающиеся количеством и расположением точек измерения. Пропускную способность можно измерять между любыми двумя узлами или точками сети, например, между клиентским компьютером 1 и сервером 3 предстваленным выше на Рисунке 3.2. При этом получаемые значения пропускной способности будут изменяться при одних и тех же условиях работы сети в зависимости от того, между какими двумя точками производятся измерения. Так как в сети одновременно работает большое число пользовательских компьютеров и серверов, то полную характеристику пропускной способности сети дает набор пропускных способностей, измеренных для различных сочетаний взаимодействующих компьютеров - так называемая матрица трафика узлов сети. Существуют специальные средства измерения, которые фиксируют матрицу трафика для каждого узла сети.

Так как в сетях данные на пути до узла назначения обычно проходят через несколько транзитных промежуточных этапов обработки, то в качестве критерия эффективности может рассматриваться пропускная способность отдельного

промежуточного элемента сети - отдельного канала, сегмента или коммуникационного устройства.

Знание общей пропускной способности между двумя узлами не может дать полной информации о возможных путях ее повышения, так как из общей цифры нельзя понять, какой из промежуточных этапов обработки пакетов в наибольшей степени тормозит работу сети. Поэтому данные о пропускной способности отдельных элементов сети могут быть полезны для принятия решения о способах ее оптимизации.

3.2.4.5 В рассматриваемом выше примере пакеты на пути от клиентского компьютера 1 до сервера 3 проходят через следующие промежуточные элементы сети:

Сегмент A -> Коммутатор -> Сегмент B -> Маршрутизатор -> Сегмент C -> Повторитель -> Сегмент D.

Каждый из этих элементов обладает определенной пропускной способностью, поэтому общая пропускная способность сети между компьютером 1 и сервером 3 будет равна минимальной из пропускных способностей составляющих маршрута, а задержка передачи одного пакета (один из вариантов определения времени реакции) будет равна сумме задержек, вносимых каждым элементом. Для повышения пропускной способности составного пути необходимо в первую очередь обратить внимание на самые медленные элементы - в данном случае таким элементом, скорее всего, будет маршрутизатор.

Имеет смысл определить общую пропускную способность сети как среднее количество информации, переданной между всеми узлами сети в единицу времени. Общая пропускная способность сети может измеряться как в пакетах в секунду, так и в битах в секунду. При делении сети на сегменты или подсети общая пропускная способность сети равна сумме пропускных способностей подсетей плюс пропускная способность межсегментных или межсетевых связей.

Пропускная способность сети зависит как от характеристик физической среды передачи (медный кабель, оптическое волокно, витая пара) так и от принятого способа передачи данных (технология Ethernet, FastEthernet, ATM). Пропускная способность часто используется в качестве характеристики не столько сети, сколько собственно технологии, на которой построена сеть. Важность этой характеристики для сетевой технологии показывает, в частности, и то, что ее значение иногда становится частью названия, например, 10 Мбит/с Ethernet, 100 Мбит/с Ethernet.

В отличие от времени реакции или скорости передачи трафика пропускная способность не зависит от загруженности сети и имеет постоянное значение, определяемое используемыми в сети технологиями.

3.2.4.6 На разных участках гетерогенной сети, где используется несколько разных технологий, пропускная способность может быть различной. Для анализа и настройки сети очень полезно знать данные о пропускной способности отдельных ее элементов. Важно отметить, что из-за последовательного характера передачи данных различными элементами сети общая пропускная способность любого составного пути в сети будет равна минимальной из пропускных способностей составляющих элементов маршрута. Для повышения пропускной способности составного пути необходимо в первую очередь обратить внимание на самые медленные элементы. Иногда полезно оперировать общей пропускной способностью сети, которая определяется как среднее количество информации, переданной между всеми узлами сети за единицу времени. Этот показатель характеризует качество сети в целом, не дифференцируя его по отдельным сегментам или устройствам.

3.2.5 Задержка передачи и вариация задержки передачи

Задержка передачи определяется как задержка между моментом поступления данных на вход какого-либо сетевого устройства или части сети и моментом появления их на выходе этого устройства.

Этот параметр производительности по смыслу близок ко времени реакции сети, но отличается тем, что всегда характеризует только сетевые этапы обработки данных, без задержек обработки конечными узлами сети.

Обычно качество сети характеризуют величинами максимальной задержки передачи и вариацией задержки. Не все типы трафика чувствительны к задержкам передачи, во всяком случае, к тем величинам задержек, которые характерны для компьютерных сетей, — обычно задержки не превышают сотен миллисекунд, реже — нескольких секунд. Такого порядка задержки пакетов, порождаемых файловой службой, службой электронной почты или службой печати, мало влияют на качество этих служб с точки зрения пользователя сети. С другой стороны, такие же задержки пакетов, переносящих голосовые или видеоданные, могут приводить к значительному снижению качества предоставляемой пользователю информации — возникновению эффекта "эха", невозможности разобрать некоторые слова, вибрации изображения и т. п.

3.2.6 Сопоставление характеристик производительности

Все указанные характеристики производительности сети достаточно независимы. В то время как пропускная способность сети является постоянной величиной, скорость передачи трафика может варьироваться в зависимости от загрузки сети, не превышая, конечно, предела, устанавливаемого пропускной способностью. Так в односегментной сети 10 Мбит/с Ethernet компьютеры могут обмениваться данными со скоростями 2 Мбит/с и 4 Мбит/с, но никогда — 12 Мбит/с.

Пропускная способность и задержки передачи также являются независимыми параметрами, так что сеть может обладать, например, высокой пропускной способностью, но вносить значительные задержки при передаче каждого пакета.

Пропускная способность глобальной сети обычно в 10–20 раз ниже, чем локальной, а задержки в 100–1000 раз выше. При увеличении загрузки канала WAN растет частота потери пакетов, что ведет к ухудшению качества работы и

увеличению времени отклика приложений. В результате пользователи испытывают неудобства и их продуктивность снижается. Кроме того, не исключены простои системы, что может привести к большим убыткам. Увеличение пропускной способности каналов (собственных или арендуемых) нередко обходится дорого и не всегда помогает из-за фактора задержки. Иногда проблему частично или полностью удается решить за счет применения правил приоритетного обслуживания (CoS/QoS), изменения настроек бизнес-приложений или пересмотра архитектуры решения. Большинство сетей используются для передачи трафика разного типа и важности, и многие организации стараются его регулировать, чтобы сократить время отклика важных приложений и уменьшить затраты.

3.3 Показатели надежности сети

3.3.1.1 Одна из первоначальных целей создания распределенных систем, к которым относятся и вычислительные сети, состояла в достижении большей надежности по сравнению с отдельными вычислительными машинами. Таким образом, важнейшей характеристикой вычислительной сети является надежность - способность правильно функционировать в течение продолжительного периода времени. Это свойство имеет три составляющих: собственно надежность, готовность, безопасность и отказоустойчивость.

3.3.1.2 Повышение *надежности* заключается в предотвращении неисправностей, отказов и сбоев за счет применения электронных схем и компонентов с высокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечения тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры. Надежность сетей как распределенных систем во многом определяется надежностью кабельных систем и коммутационной аппаратуры - разъемов, кроссовых панелей, коммутационных шкафов, обеспечивающих собственно электрическую или оптическую связность отдельных узлов между собой.

Важно различать несколько аспектов надежности. Для сравнительно простых технических устройств используются такие показатели надежности, как:

1) среднее время наработки на отказ;
2) вероятность отказа;
3) интенсивность отказов.

Однако эти показатели пригодны для оценки надежности простых элементов и устройств, которые могут находиться только в двух состояниях — работоспособном или неработоспособном. Сложные системы, состоящие из многих элементов, кроме состояний работоспособности и неработоспособности, могут иметь и другие промежуточные состояния, которые эти характеристики не учитывают.

Для оценки надежности сетей, как сложных систем применяется другой набор характеристик:

1) готовность или коэффициент готовности;
2) сохранность данных;
3) согласованность (непротиворечивость) данных;
4) вероятность доставки данных;
5) безопасность;
6) отказоустойчивость.

3.3.1.3 *Готовность* или коэффициент готовности (availability) означает период времени, в течение которого система может использоваться. Готовность может быть повышена путем введения избыточности в структуру системы: ключевые элементы системы должны существовать в нескольких экземплярах, чтобы при отказе одного из них функционирование системы обеспечивали другие.

Так как сети обслуживают одновременно большое количество пользователей, то при расчете коэффициента готовности необходимо учитывать это обстоятельство. Коэффициент готовности сети должен соответствовать доле времени, в течение которого сеть выполняла с должным качеством свои функции для всех

пользователей. Очевидно, что в больших сетях очень трудно обеспечить значения коэффициента готовности, близкие к единице.

Чтобы компьютерную систему можно было считать высоконадежной, она должна как минимум обладать высокой готовностью, но этого недостаточно. Необходимо обеспечить сохранность данных и защиту их от искажений. Кроме того, должна поддерживаться согласованность (непротиворечивость) данных, например если для повышения надежности на нескольких файловых серверах хранится несколько копий данных, то нужно постоянно обеспечивать их идентичность.

Так как сеть работает на основе механизма передачи пакетов между конечными узлами, одной из характеристик надежности является вероятность доставки пакета узлу назначения без искажений. Наряду с этой характеристикой могут использоваться и другие показатели: вероятность потери пакета (по любой из причин — из-за переполнения буфера маршрутизатора, несовпадения контрольной суммы, отсутствия работоспособного пути к узлу назначения и т. д.), вероятность искажения отдельного бита передаваемых данных, соотношение количества потерянных и доставленных пакетов.

Повышение готовности предполагает подавление в определенных пределах влияния отказов и сбоев на работу системы с помощью средств контроля и коррекции ошибок, а также средств автоматического восстановления циркуляции информации в сети после обнаружения неисправности. Повышение готовности представляет собой борьбу за снижение времени простоя системы.

Критерием оценки готовности является коэффициент готовности, который равен доле времени пребывания системы в работоспособном состоянии и может интерпретироваться как вероятность нахождения системы в работоспособном состоянии. Коэффициент готовности вычисляется как отношение среднего времени наработки на отказ к сумме этой же величины и среднего времени восстановления. Системы с высокой готовностью называют также отказоустойчивыми.

Основным способом повышения готовности является избыточность, на основе которой реализуются различные варианты отказоустойчивых архитектур. Вычислительные сети включают большое количество элементов различных типов, и для обеспечения отказоустойчивости необходима избыточность по каждому из ключевых элементов сети. Если рассматривать сеть только как транспортную систему, то избыточность должна существовать для всех магистральных маршрутов сети, то есть маршрутов, являющихся общими для большого количества клиентов сети. Такими маршрутами обычно являются маршруты к корпоративным серверам – например, серверам баз данных, Web-серверам, почтовым серверам. Поэтому для организации отказоустойчивой работы все элементы сети, через которые проходят такие маршруты, должны быть зарезервированы: должны иметься резервные кабельные связи, которыми можно воспользоваться при отказе одного из основных кабелей, все коммуникационные устройства на магистральных путях должны либо сами быть реализованы по отказоустойчивой схеме с резевированием всех основных своих компонентов, либо для каждого коммуникационного устройства должно иметься резервное аналогичное устройство.

Переход с основной связи на резервную или с основного устройства на резервное может происходить как в автоматическом режиме, так и вручную, при участии администратора. Очевидно, что автоматический переход повышает коэффициент готовности системы, так как время простоя сети в этом случае будет существенно меньше, чем при вмешательстве человека. Для выполнения автоматических процедур реконфигурации необходимо иметь в сети интеллектуальные коммуникационные устройства, а также централизованную систему управления, помогающую устройствам распознавать отказы в сети и адекватно на них реагировать.

Высокую степень готовности сети можно обеспечить в том случае, когда процедуры тестирования работоспособности элементов сети и перехода на резервные элементы встроены в коммуникационные протоколы. Примером такого типа протоколов может служить протокол FDDI, в котором постоянно тестируются физические связи между узлами и концентраторами сети, а в случае их отказа

выполняется автоматическая реконфигурация связей за счет вторичного резервного кольца. Существуют и специальные протоколы, поддерживающие отказоустойчивость сети, например, протокол SpanningTree, выполняющий автоматический переход на резервные связи в сети, построенной на основе мостов и коммутаторов.

3.3.1.4 Другим аспектом общей надежности является *безопасность* (security), то есть способность системы защитить данные от несанкционированного доступа. В распределенной системе это сделать гораздо сложнее, чем в централизованной. В сетях сообщения передаются по линиям связи, часто проходящим через общедоступные помещения, в которых могут быть установлены средства прослушивания линий. Другим уязвимым местом могут стать оставленные без присмотра персональные компьютеры. Кроме того, всегда имеется потенциальная угроза взлома защиты сети от неавторизованных пользователей, если сеть имеет выходы в глобальные общедоступные сети.

3.3.1.5 Еще одной характеристикой надежности является *отказоустойчивость* (fault tolerance). В сетях под отказоустойчивостью понимается способность системы скрыть от пользователя отказ отдельных ее элементов. Например, если копии таблицы базы данных хранятся одновременно на нескольких файловых серверах, пользователи могут просто не заметить отказа одного из них. В отказоустойчивой системе выход из строя одного из ее элементов приводит к некоторому снижению качества ее работы (деградации), а не к полному останову. Так, при отказе одного из файловых серверов в предыдущем примере увеличивается только время доступа к базе данных из-за уменьшения степени распараллеливания запросов, но в целом система будет продолжать выполнять свои функции.

Существуют различные градации отказоустойчивых компьютерных систем, к которым относятся и вычислительные сети. Приведем несколько общепринятых определений:

— *высокая готовность* (high availability) - характеризует системы, выполненные по обычной компьютерной технологии, использующие избыточные аппаратные и

программные средства и допускающие время восстановления в интервале от 2 до 20 минут;

– *устойчивость к отказам* (fault tolerance) - характеристика таких систем, которые имеют в горячем резерве избыточную аппаратуру для всех функциональных блоков, включая процессоры, источники питания, подсистемы ввода/вывода, подсистемы дисковой памяти, причем время восстановления при отказе не превышает одной секунды;

– *непрерывная готовность* (continuous availability) - это свойство систем, которые также обеспечивают время восстановления в пределах одной секунды, но в отличие от систем устойчивых к отказам, системы непрерывной готовности устраняют не только простои, возникшие в результате отказов, но и плановые простои, связанные с модернизацией или обслуживанием системы. Все эти работы проводятся в режиме online. Дополнительным требованием к системам непрерывной готовности является отсутствие деградации, то есть система должна поддерживать постоянный уровень функциональных возможностей и производительности независимо от возникновения отказов.

3.4 Выводы

Обоснование и выбор критериев эффективности управления сетевой инфраструктурой КС зависят от целей управления, особенностей среды (WAN, сеть ЦОД, корпоративная сети), характеристиками пользователей и трафика в ней и еще целого ряда факторов.

Основной целью функционирования КС является передача информации, поэтому ключевыми показателями являются скорость передачи, объемы и качество её передачи. Соответственно, чем эффективнее используемые алгоритмы управления сетевой инфраструктурой и потоками данных, тем выше эти показатели.

Таким образом, можно выделить следующие ключевые критерии и показатели эффективности управления сетевой инфраструктурой компьютерных сетей и потоками данных в них:

— Производительность КС и ее показатели:

1) Время реакции - интервал времени между возникновением запроса пользователя к кому-либо сетевому сервису и получением ответа на этот запрос.

2) Пропускная способность – скорость передачи пакетов данных между узлами сети через различные коммуникационные устройства.

3) Задержка передачи - задержка между моментом поступления данных на вход какого-либо сетевого устройства или части сети и моментом появления их на выходе этого устройства.

— Надежность КС и ее показатели

1) среднее время наработки на отказ;

2) вероятность отказа;

3) интенсивность отказов.

Между показателями производительности и надежности сети существует прямая связь. Ненадежная работа сети очень часто приводит к существенному снижению ее производительности. Это объясняется тем, что сбои и отказы каналов связи и коммуникационного оборудования приводят к потере или искажению некоторой части пакетов, в результате чего коммуникационные протоколы вынуждены организовывать повторную передачу утерянных данных. Так как в локальных сетях восстановлением утерянных данных занимаются, как правило, протоколы транспортного или прикладного уровня, работающие с тайм-аутами в несколько десятков секунд, то потери производительности из-за низкой надежности сети могут составлять сотни процентов. Поэтому необходимо учитывать надежность КС и ее показатели.

4 ПРОВЕДЕНИЕ СРАВНЕНИЯ МЕТОДОВ И СРЕДСТВ УПРАВЛЕНИЯ ПКС С ТРАДИЦИОННЫМИ МЕТОДАМИ И СРЕДСТВАМИ УПРАВЛЕНИЯ СЕТЕВЫМИ РЕСУРСАМИ И ПОТОКАМИ ДАННЫХ В КС

4.1 Сравнение подходов к управлению в ПКС

Программно-конфигурируемая сеть (ПКС) – это новый подход к построению архитектуры компьютерных сетей, при котором уровень управления (УУ) сетью (состоянием сетевой инфраструктуры и потоками данных в сети) и уровень передачи данных (УПД) разделяются за счет переноса функций управления (выполняемых в традиционной сети маршрутизаторами и коммутаторами) на отдельное центральное устройство, называемое контроллером. За счет такого разделения контроль состояния сети и управление сетью логически централизовано на контроллере. Кроме того такой подход позволяет уровню управления абстрагироваться от физической сетевой инфраструктуры уровня передачи данных, используя некоторое логическое представление сети. Взаимодействие между УУ и УПД осуществляется посредством единого унифицированного открытого интерфейса.

Основные идеи, которые закладывались в ПКС, заключаются в следующем:

– Разделение уровня передачи и уровня управления данными.

– Единый, унифицированный, не зависящий от поставщика интерфейс между уровнем управления и уровнем передачи данных.

– Логически централизованный уровень управления данными.

– Виртуализация физических ресурсов сети.

Архитектура ПКС согласно [6] имеет три уровня, согласно Рисунку 4.1. :

– Уровень инфраструктуры сети включает в себя набор сетевых устройств (коммутаторов, маршрутизаторов) и каналов передачи данных.

– Уровень управления, на котором отслеживается и поддерживается глобальное представление сети (ГПС). Под *глобальным представлением сети* понимается

топология сети и состояние сетевых устройств. Уровень управления предоставляет программный интерфейс (API) для сетевых приложений.

– Уровень сетевых приложений, в которых реализуются различные функции управления сетью: управление потоками данных в сети, управление безопасностью, мониторинг трафика, управление качеством сервиса, управления политиками и так далее.

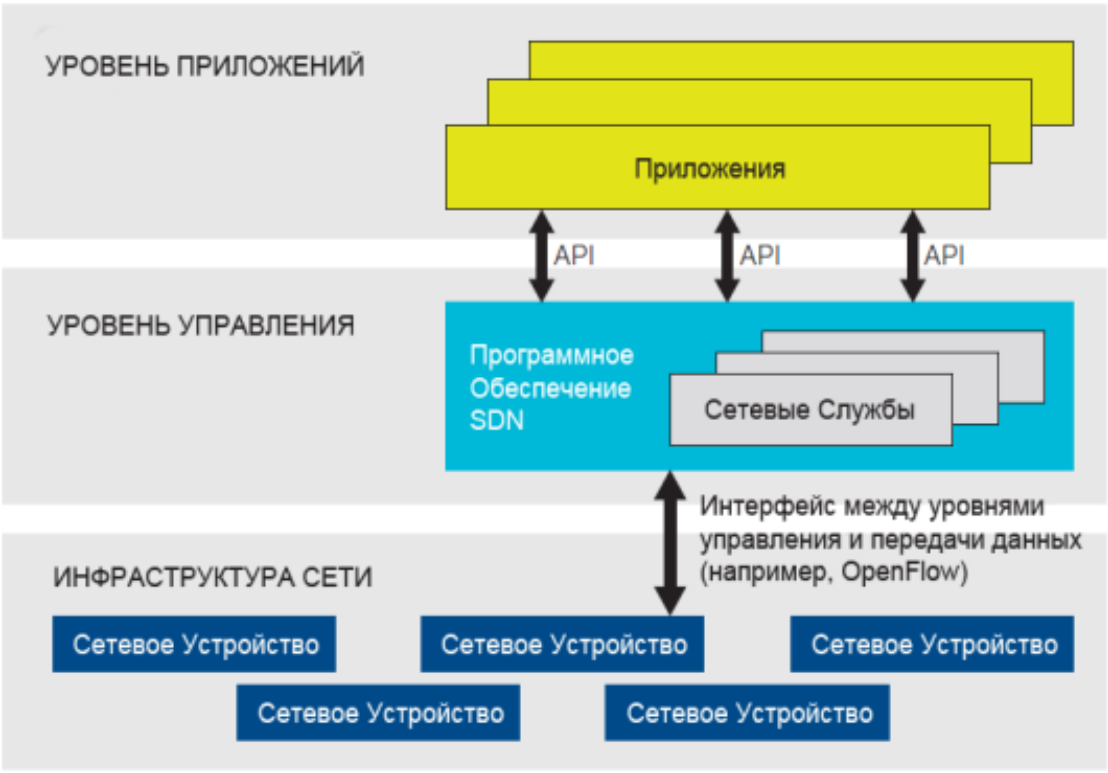

Рисунок 4.1 – Архитектура программно-конфигурируемых сетей

Современные маршрутизаторы решают две основные задачи: передача данных (forwarding) – продвижение пакета от входного порта на определенный выходной порт, и управление данными – обработка пакета и принятие решения о том, куда его маршрутизировать, на основе текущего состояния маршрутизатора. Таким образом, в рамках всей сети можно выделить уровень передачи данных (УПД), состоящий из средств передачи данных (линий связи, каналообразующего оборудования, маршрутизаторов и коммутаторов), и уровень управления (УУ) состояниями средств передачи данных

Развитие маршрутизаторов шло по пути сближения и «сращивания» этих двух уровней, аппаратного ускорения, совершенствования ПО и внедрения новых функциональных возможностей для увеличения скорости принятия решения по маршрутизации каждого пакета. Но при этом уровень управления остался достаточно примитивным, опираясь на сложные распределенные алгоритмы маршрутизации и замысловатые инструкции по конфигурированию и настройке сети. Нужно отметить, что программное обеспечение маршрутизаторов, реализующее уровень управления, оставалось проприетарным и закрытым для разработчиков, исследователей и сетевых операторов.

В подходе ПКС было предложено разделить УУ и УПД. Таким образом, архитектура ПКС и предлагаемый централизованный подход дает следующие преимущества по сравнению с традиционными сетями с распределенным управлением передачей данных:

— *Программируемость и гибкость* управления сетью, значительное упрощение возможности модификации управления сетью за счет создания новых приложений или модификации существующих, автоматизация управления и администрирования сетями.

— *Адаптивность управления сетью*, то есть возможность изменять поведение и состояние сети в режиме реального времени с учетом изменяющихся условий функционирования и адаптироваться к ним, адаптироваться к меняющимся потребностям пользователей сетей за счет создания новых сетевых приложений и сервисов. На разработку сетевых приложений требуется значительно меньше времени по сравнению с ручным переконфигурированием всей сети.

— Независимость от оборудования и несвободного (англ. proprietary) программного обеспечения производителей сетевого оборудования.

— Возможность независимого развертывания УУ и УПД.

— Возможность независимого масштабирования УУ и УПД.

— *Повышение надежности* за счет снижения объема распределенного состояния

для управления. Вместо имеющихся распределенных протоколов, которые работают на каждом узле сети, каждый из них поддерживает базу данных распределенных копий состояний каналов в каждом узле, однако такая информация может быть собрана централизованно в одном месте – на контроллере. Таким образом, такая централизованная база данных будет содержать значительно меньше несогласованной информации, и такой подход позволит уменьшить вероятность циклов в сети.

— *Упрощение структуры и логики сетевых устройств*, поскольку теперь им не требуется обрабатывать огромное количество стандартов и протоколов, а достаточно выполнять только инструкции, полученные от контроллера.

— *Снижение стоимости коммутаторов и сетевой инфраструктуры* в целом за счет вынесения «интеллекта маршрутизаторов» в контроллер.

Таким образом, подход ПКС позволяет в значительной степени автоматизировать и упростить управление сетями за счет возможности их «программирования», позволяя строить гибкие масштабируемые сети, которые могут легко адаптироваться к изменяющимся условиям функционирования и потребностям пользователей. Внедрение этого подхода, прежде всего, должно оказать значительное влияние на управление сетевой инфраструктурой в центрах обработки данных (ЦОД), корпоративными сетями, WAN, сотовыми и домашними сетями.

Однако в архитектуре ПКС можно отметить и определенные недостатки:

— *Проблема надежности:* контроллер является потенциальной точкой отказа работы сети.

— Проблема масштабируемости:

1) Количество управляющего трафика, предназначенного для централизованного контроллера, растет пропорционально количеству коммутаторов в сети.

2) Время установления новых потоков может расти значительно с ростом размеров сети.

– *Проблема производительности.* Производительность сети напрямую зависит от производительности контроллера и его физических ограничений.

4.2 Сравнение методов управления потоками данных

4.2.1 Методы управления данными в традиционных КС

4.2.1.1 Классификация алгоритмов маршрутизации. Протоколы маршрутизации в традиционных сетях могут быть реализованы с помощью алгоритмов маршрутизации, которые классифицируются по следующим признакам:

1) по способу выбора наилучшего маршрута;
2) по способу построения таблиц маршрутизации;
3) по месту выбора маршрутов (маршрутного решения);
4) по виду информации, которой обмениваются маршрутизаторы

Ниже приведено более подробное описание каждой группы алгоритмов.

4.2.1.2 Алгоритмы маршрутизации по способу выбора наилучшего маршрута

Алгоритмы маршрутизации по способу выбора наилучшего маршрута делятся на 2 группы:

– *Одношаговые алгоритмы маршрутизации*. Каждый маршрутизатор при выборе маршрута определяет только одно звено этого маршрута и несет ответственность только за один шаг этого маршрута. Одношаговые алгоритмы лучше работают на этапе инициализации сети.

– *Многошаговые алгоритмы маршрутизации* (алгоритм маршрутизации от источника). Весь маршрут задается в уже отправленном пакете узлом источника. Многошаговые алгоритмы считаются более перспективными.

4.2.1.3 Алгоритмы маршрутизации по способу построения таблиц маршрутизации делятся на 3 группы: алгоритмы простой маршрутизации, алгоритмы фиксированной и статической маршрутизации и адаптивные алгоритмы.

4.2.1.4 *Алгоритмы простой маршрутизации*. Таблиц маршрутизации как правило нет, или таблицы являются очень примитивными:

1) Алгоритмы случайной маршрутизации – пакет посылается в случайном направлении.
2) Лавинная маршрутизация (алгоритмы заполнения) – пакеты посылаются во все выходные направления, во все порты.
3) Алгоритмы скорейшей передачи (алгоритм горячей картошки) – как только маршрутизатор получает пакет – он старается скорее его отослать.
4) Алгоритмы кратчайшей очереди (наименьшей загрузки) – информация идет на порт, который наименее загружен.
5) Алгоритм по предыдущему опыту – таблица маршрутизации очень примитивна, есть запись, описывающая передачу предыдущего пакета.

В настоящее время используют лавинные алгоритмы. Они самые быстрые по доставке информации, а также могут информировать об экстренной информации, возникающей в сети передачи данных.

4.2.1.5 *Алгоритмы фиксированной и статической маршрутизации*. Это алгоритмы, в которых таблицы маршрутизации заполняются администратором сети, поэтому все записи являются статическими или редко изменяются во время функционирования сети. Используются для сетей с простой топологией и чаще всего применяются в теоретических приложениях.

- Однопутевые (одномаршрутные, безальтернативные).
- Многопутевые (многомаршрутные, допускающие альтернативу).

4.2.1.6 *Адаптивные алгоритмы, они же алгоритмы динамической маршрутизации*. Они наиболее распространены. Автоматическое построение таблиц

маршрутизации, эти алгоритмы адаптированы к изменениям в сети. К адаптивным алгоритмам предъявляются следующие требования:

1) простота – как можно меньшая вычислительная сложность алгоритма;
2) адаптивные алгоритмы маршрутизации должны обеспечивать если не оптимальные, то хотя бы близкие к таковым маршрутные решения;
3) сходимость алгоритма –алгоритм после некоторого (конечного) времени работы приводит к какому-либо определённому маршруту.

4.2.1.7 Алгоритмы маршрутизации по месту выбора маршрутов (маршрутного решения) делятся на 3 группы:

– *Изолированные алгоритмы (локальные).* Нет никакого обмена маршрутной информацией и каждый маршрутизатор принимает решение на основании той информации, которую он сам собрал.

– *Централизованные.* Вся маршрутная информация со всех маршрутизаторов стекается в сетевой маршрутный центр – он отвечает за определение оптимальных маршрутов и сбор маршрутной информации. Возможны два подхода:

1) Подход виртуального канала – маршрут определяется на основе информации, посылаемой во все промежуточные маршрутизаторы. Недостатком является уязвимость маршрутного центра.
2) Подход формирования по таблице для каждого маршрутизатора.

– *Распределенные.* Это самые распространенные алгоритмы, где все маршрутизаторы участвуют в сборе и распространении маршрутной информации. Работа по выбору наилучшего маршрута распределена между всеми маршрутизаторами.

4.2.1.8 Алгоритмы маршрутизации по виду информации, которой обмениваются маршрутизаторы, делятся на 2 класса: Дистанционно-векторные алгоритмы и алгоритмы маршрутизации по состоянию связи

4.2.1.9 *Дистанционно-векторные алгоритмы* – RIP протокол (протокол маршрутной информации). Каждый маршрутизатор периодически всем своим соседям передает вектор сообщений, где указывается адреса известных ему подсетей и расстояния до них от данного маршрутизатора.

Недостатки:

1) Плохая адаптация к отказам маршрутизаторов, интерфейсов, подсетей.
2) Возможность возникновения маршрутных петель.
3) Данный алгоритм используется для небольших сетей (количество скачков (hop) не больше 15).

Для устранения первых двух недостатков каждому маршруту присваивается время жизни (TTL=180 сек): если за это время информация не обновляется – то маршрут «умирает», прекращает действовать. Если маршрутизатор вышел из строя – то в качестве расстояния до него указывается 16, то есть «бесконечность».

Главная причина всех недостатков – это получение информации через соседние маршрутизаторы, а не напрямую (это называется отсутствием полной нужной информации).

4.2.1.10 Алгоритмы маршрутизации по состоянию связи:

1) OSPF протокол – Open Shortest Path First – открытый протокол кратчайшего маршрута – в стеке протоколов TCP/IP.
2) NLSP протокол – Netware Link Services Protocol – используется для сетевых служб, указанных в различных подсетях, с целью управления – принадлежит стеку протоколов IPX/SPX.
3) IS-IS – Intermediate System to Intermediate System – подмножество модели OSI.

Каждый маршрутизатор обеспечивается необходимой и точной (достаточной) информацией для построения адекватной (точной) топологии сети. Для точного построения графа-связи сети в данной топологии вершинами выступают маршрутизаторы и подсети, согласно Рисунку 4.2.

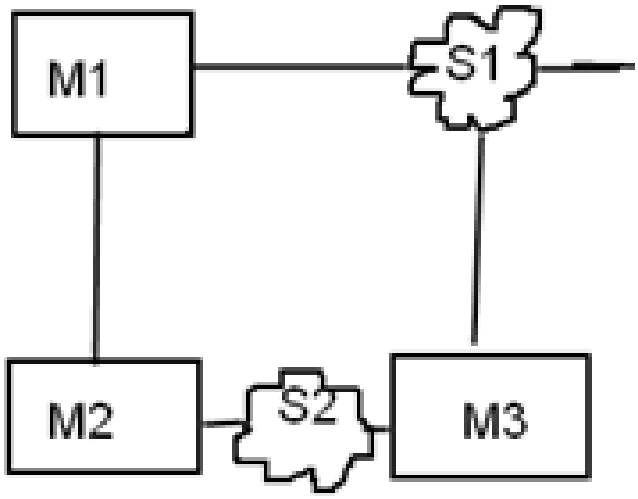

Рисунок 4.2 – Маршрутизация по состоянию связи

Принцип работы упомянутых выше алгоритмов таков:

– Каждый маршрутизатор распространяет соседям обо всех своих ближайших соседях следующую информацию:

1) Адрес соседней подсети
2) Тип интерфейса (M-M (маршрутизатор – маршрутизатор) и M-S (маршрутизатор – сеть))
3) Метрика интерфейса (пропускные способности каждого из путей, время задержки и метрика надежности)

– Соседний маршрутизатор получает информацию без коррекции, и через некоторое время все маршрутизаторы будут иметь полную информацию обо всех подсетях и маршрутизаторах. Вся информация записывается в базу данных, после чего каждый маршрутизатор знает топологию, определяет кратчайшие маршруты для всех подсетей. Для этого используется алгоритм Дейкстра – алгоритм определения кратчайшего маршрута, где каждое звено этого маршрута записывается в таблицу маршрутизации. Вычисления происходят по всей метрике.

– Для проверки в каждый маршрутизатор пересылается сообщение каждые 10 сек и если ответа нет, то таблица корректируется без учёта вышедшего из строя

маршрутизатора (поэтому, в том числе, алгоритм называется адаптивным).

Даже если топология сети длительное время не меняется, вполне возможны изменения информационных потоков, поэтому на практике предпочтительнее использовать адаптивные алгоритмы.

4.2.2 Методы управления потоками в ПКС

В отличие от распределенных алгоритмов, используемых в традиционных сетях, в ПКС могут использоваться централизованные алгоритмы управления данными и инфраструктурой, которые реализуются не в виде протоколов, а в виде сетевых приложений для контроллера.

Построение маршрута в ПКС осуществляется на основе глобального представления сети (состояния всех сетевых элементов), которое должно поддерживаться в актуальном состоянии. Необходимо отметить, что такой централизованный подход позволяет более гибко, быстро и эффективно управлять потоками данных в сети.

В ПКС выделяют два основных подхода к установке правил в коммутаторах, реализующих некоторый маршрут для потока данных:

— *Реактивный подход* – на запрос установления нового потока, контроллер формирует одно правило и устанавливает его на коммутатор, инициировавший запрос.

— *Проактивный подход* – на запрос установления нового потока, контроллер вычисляет маршрут (или его часть) для потока и устанавливает соответствующие правила на все коммутаторы этого маршрута.

Очевидны достоинства и недостатки подходов. Реактивный подход требует значительных вычислительных ресурсов контроллера, поскольку на каждом шаге маршрута запрашивает новое правило у контроллера, однако не требует информации о состоянии всей сети. Проактивный подход позволяет вычислить весь маршрут на основании информации о состоянии всей сети и быстро пропустить поток в сети, тем самым значительно увеличивая пропускную способность сети.

Допустимо предположить, что для разного типа трафика нужно использовать один из наиболее приемлемых подходов, т.е. в сети использовать одновременно оба подхода. При использовании проактивного подхода маршрут можно прокладывать не целиком, а например лишь его часть.

Показательным примером является контроллер FloodLight, для которого реализованы оба подхода. Приложения Forwarding и Learning Switch позволяют устанавливать потоки реактивно. Приложения Static Flow Entry Pusher и Circuit Pusher – проактивно.

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 (т.е., фрагмента ПКС сети между маршрутизаторами, не поддерживающими протокол OpenFlow).

4.3 Выводы

По итогам проведенного сравнения методов и средств управления ПКС с традиционными методами и средствами управления сетевыми ресурсами и потоками данных в КС нужно отметить, что подход ПКС позволяет выполнять функцию маршрутизации и доставки данных не хуже чем в традиционных сетях. Но подход ПКС предоставляет более гибкие возможности по управлению трафиком, перегрузками, управлению сетевой инфраструктурой с помощью сетевых приложений, запущенных на логически централизованном контроллере. В распределенной среде традиционных КС эти функции также реализованы с помощью различных протоколов, но в гибкости управления традиционные сети существенно проигрывают.

5 ПРОВЕДЕНИЕ АНАЛИЗА ТРАДИЦИОННЫХ СЕРВИСОВ/ПРИЛОЖЕНИЙ, ПРИМЕНЯЕМЫХ ДЛЯ УПРАВЛЕНИЯ СЕТЕВОЙ ИНФРАСТРУКТУРОЙ ПКС, И СПЕЦИФИКАЦИИ ТРЕБОВАНИЙ К УПРАВЛЕНИЮ КОМПЬЮТЕРНЫМИ СЕТЯМИ И ПОТОКАМИ ДАННЫХ В ПКС

5.1 Краткое описание контроллеров ПКС

Контроллер ПКС состоит из ядра, реализующего основную функциональность, а так же различных пользовательских приложений для управления сетью, работающих на контроллере.

Основная функциональность, необходимая для работы с коммутаторами и приложениями и реализованная в ядре контроллера — это 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 внутренне представление, сериализует сообщения, отправляемые приложениями, для передачи их по сети и предоставляет приложениям интерфейс для получения и отправки OpenFlow сообщений.

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 нормального 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. Сервисы контроллера могут быть реализованы как в виде отдельных загружаемых модулей, так и непосредственно в ядре контроллера. 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.

Приложение контроллера реализует произвольную функциональность, необходимую пользователю. Приложения могут представлять собой как модуль контроллера, так и самостоятельный процесс, взаимодействующий с контроллером через некоторый интерфейс (например, RESTful). Реализация приложения в виде модуля контроллера обеспечивает высокую скорость обмена сообщениями между ядром контроллера и приложением и между различными приложениями. Однако использование стандартного 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.

Таким образом, при разработке контроллера необходимо определить, какую функциональность следует реализовать в ядре контроллера, какую — вынести в загружаемый модуль, а какую — оформить в виде отдельного приложения.

Дальнейший обзор сервисов и приложений контроллеров ПКС основан на документации к контроллерам POX [16], NOX Classic [17], Floodlight [18], Beacon [19] и Ryu [20], эти контроллеры выбраны для обзора как наиболее представительные в части имеющихся сервисов и приложений.

5.2 Сервисы контроллера

5.2.1 Функциональные группы сервисов контроллера

С точки зрения функциональности сервисы контроллера можно разделить на четыре группы. Ниже приведено описание сервисов по группам.

5.2.2 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.

5.2.3 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 [21] и Spring [22].

– 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 [23].

– 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.

5.2.4 Сервисы для работы с топологией сети

5.2.4.1 *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 [1] [24]. 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.

5.2.4.2 *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.

5.2.4.3 *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

[1] **Link Layer Discovery Protocol** (**LLDP**) — протокол канального уровня, позволяющий сетевому оборудованию оповещать локальную сеть о своем существовании и характеристиках, а также собирать такие же оповещения, поступающие от соседнего оборудования.

коммутаторах, через которые проходит маршрут, контроллер устанавливает соответствующие правила.

5.2.4.4 *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 (зацикливаание широковещательных пакетов в сети). Данный модуль может использоваться для реализации функциональности протокола STP на коммутаторах, на которых данный протокол не поддерживается.

5.2.5 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 mininet.

5.3 Приложения

5.3.1 Функциональные группы приложений контроллера

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

1) Приложения, реализующие простейшие правила коммутации и маршрутизации пакетов в сети.
2) Приложения для работы с распространенными сетевыми протоколам.
3) Приложения для анализа и перераспределения трафика в сети.
4) Приложения для поддержки виртуализации и сетей ЦОД.

5.3.2 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:

1) установка новых правил с заданием только соответствующих MAC адресов, независимо для каждого коммутатора (традиционный L2 learning, описанный выше);
2) 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);
3) 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.

5.3.3 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.

5.3.4 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

[25].

На данный момент эти приложения входят только в контроллер Floodlight.

5.3.5 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 ПКС 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).

На сегодняшний день приложения для работы с виртуальными сетями и системами управления ЦОД поставляются с контроллерами NOX, Floodlight и Ryu.

5.4 Выводы

На основе проведенного анализа сервисов можно сделать вывод, что для нормальной работы контроллер должен предоставлять следующий минимальный набор сервисов: OpenFlow Protocol, Network Protocols, Module Manager, Messenger. Перечисленные сервисы, необходимые для работы приложений и взаимодействия с коммутаторами, должны быть реализованы в ядре контроллера. Остальные сервисы контроллера целесообразно реализовать в виде загружаемых модулей, таким

образом, при запуске контроллера пользователь сможет выбирать загрузку только необходимых ему сервисов.

Желательно предусмотреть пользовательский интерфейс для управления контроллером, например, при помощи командной строки (CLI) или через веб-интерфейс (Web GUI).

Приложения, для которых важна высокая скорость обмена сообщениями с ядром контроллера или с другими приложениями, целесообразно реализовать как модуль контроллера. К таким приложениям можно отнести Learning Switch, Firewall и т. п., которые должны получать все сообщения PacketIn от коммутаторов.

Использование таких API, как RESTful, позволяет писать приложения на любом языке программирования, поддерживающем работу с выбранным API, а также запускать приложения на удаленном сервере, хотя и снижает скорость обмена сообщениями между приложениями и ядром контроллера. Запуск приложений удаленно может понадобиться, например, для интеграции контроллера с пользовательскими интерфейсами систем управления ЦОД. На сегодняшний день такая схема взаимодействия с контроллером реализована, например, в приложении OpenStack Quantum Plugin для Floodlight.

6 ПРОВЕДЕНИЕ АНАЛИЗА СРЕДСТВ ФОРМИРОВАНИЯ OPENFLOW ТАБЛИЦ ДЛЯ OPENFLOW СЕТЕВЫХ КОММУТАТОРОВ, ВОЗМОЖНОСТИ ФОРМИРОВАНИЯ СЕТЕВЫХ СРЕЗОВ С ПОМОЩЬЮ ТАКИХ КОММУТАТОРОВ

6.1 Протокол OpenFlow. Проблемы реализации

Протокол OpenFlow определяет интерфейс между контроллером и коммутаторами, посредством которого контроллер устанавливает правила коммутации на них. OpenFlow позволяет разрабатывать широкий спектр новых сервисов управления сетью, однако это вызывает ряд трудностей. Разработчики приложений вынуждены бороться с решениями одних и тех же проблем.

Как правило, в сети осуществляется выполнение нескольких задач одновременно, например, задачи маршрутизации, контроля доступа и мониторинга трафика. К сожалению, разделенение этих задач друг от друга и их независимая реализация фактически невозможны, так как правила обработки пакетов, установленные одним модулем, могут перекрываться правилами, установленными другим модулем.

Например, приложение A занимается маршрутизацией и устанавливает правило, что все пакеты, приходящие с первого порта, надо пересылать на второй порт: <in_port=1:{output(2)}>. Так же в сети работает приложение B, которое отвечает за подсчет http-трафика (количество пакетов на порт 80 в TCP), проходящего через коммутатор. Самый простой и в то же время неправильный способ - это добавить правило без действия <tcp_port=80:{}>. Несмотря на то, что это означает сброс всех пакетов заданного вида, значения счетчиков правила будут обновляться и поставленная задача будет решена. В протоколе Openflow 1.1 нельзя решить обе эти задачи независимо друг от друга и требуется объединение обоих правил.

OpenFlow предоставляет очень низкий уровень абстракции по управлению сетью.

Например, набор правил, которые можно установить на коммутатор, сильно зависит от его функциональных возможностей. Сложные программы требуют

добавление большого числа правил с различными приоритетами и с использованием групповых символов. Но количество таких правил сильно ограничено используемым аппаратным обеспечением. И за всем этим должен следить программист.

Контроллер получает события только о пакетах, с которыми коммутаторы не знают что делать. Разработка сетевых программ становится более сложным за счет такой двух уровневой модели управления, в которой приходится учитывать, как пакеты, обрабатываемые на коммутаторе, так и пакеты, обрабатываемые на контроллере.

Сеть из коммутаторов представляет собой распределенную систему, разработка приложений для которой вызывает ряд трудностей, связанных с особенностями параллельного программирования: «гонками», блокировками, синхронизацией.

Например, в парадигме Openflow первый пакет нового потока перенаправляется на контроллер для принятия решения о его маршрутизации. Но на принятия такого решения требуется время, в течение которого на коммутатор уже могут придти следующие пакеты из этого потока. Разработчики сетевых приложений должны самостоятельно обрабатывать такие пакеты.

6.2 Языки программирования OpenFlow сетей

Для решения указанных проблем разрабатываются специальные языки программирования сетей, которые с одной стороны представляют высокоуровневые способы управления потоками в сети, а с другой стороны скрывают все детали по правильной и корректной установке низкоуровневых правил на коммутаторы.

Примерами таких языков могут служить Frenetic, NetCore и Nettle, которые будут подробнее рассмотрены ниже.

Все существующие языки программирования OpenFlow сетей предоставляют разработчику сетевых приложения такие конструкции языка, в терминах которых удобно описывать правила управления потоками, абстрагируясь тем самым от уровня команд коммутатора OpenFlow. Например, языки обычно позволяют обращаться к отдельным полям заголовка по их имени, предоставляют встроенные

механизмы для сбора статистики и анализа активности проходящих через сеть потоков данных. Некоторые языки предоставляют абстракции более высокого уровня, такие как состояние контроллера сети через историю всех пакетов, которые когда либо проходили через коммутаторы сети. Однако, чем больше выразительная мощность языка, и чем более высокоуровневые абстракции используются, тем сложнее преобразовать выражения в правила для коммутаторов OpenFlow.

Так же семантика языков программирования сетей позволяет исключить побочные эффекты, вызванные взаимодействием разнородных сетевых приложений, решая тем самым проблему композиции устанавливаемых приложениями правил. Возможность такой композиции требует дополнительной прослойки между приложением и реальным коммутатором, которая бы приводила конкурирующие политики маршрутизации к согласованному виду. Среды выполнения языков программирования сетей позволяют эффективно бороться с двойственностью в процессе обработки пакетов, и избавляют разработчиков от необходимости обдумывания сразу нескольких сценариев обработки пакетов. Например, некоторые среды выполнения автоматически обрабатывают «вторые» пакеты, пересланные коммутатором, уже после того, как приложение произвело обработку соответствующего потока по его «первому» пакету.

6.3 Описание языков

6.3.1 Frenetic

6.3.1.1 Состав языка. Frenetic – это один из первых языков программирования сетей [26]. Встроен в язык общего программирования Python. Состоит из двух частей:

— ограниченный, но высокоуровневый декларативный язык запросов к сети – *network query language*;

— функциональный язык общего назначения для реактивного управления политиками маршрутизации в сетях – *network policy management library*.

Оба эти языка обладают рядом полезных свойств, таких как: видимость каждого пакета в сети (как если бы все пакеты шли через контроллер), отсутствие гонок и ручного управления синхронизацией, а так же модель затрат, которая определяет какие операции являются тяжелыми для вычисления, а какие нет.

6.3.1.2 Язык запросов к сети. Язык запросов позволяет Frenetic программам получать различные состояния сети (например, количество HTTP трафика, проходящего через этот коммутатор). Это обеспечивается с помощью добавления вспомогательных служебных OpenFlow правил, которые не видны программисту. Большую сложность при разработке этого языка представлял выбор между его выразительностью, простой и низкой вычислительной сложностью. Сложность запроса определяется как число пакетов, которые требуется отправлять на контроллер для обработки. Возможность управлять сложностью необходимо для контроля задержки на работу всей сети целиком, так как невозможно обрабатывать все пакеты сети на контроллере.

Запросы позволяют задавать высокоуровневые фильтры над множеством всех пакетов сети. Запросы поддерживают группировку по заданным полям, разделение пакетов на множества в зависимости от времени прибытия на коммутатор или если поменялось какое-то значение заголовка, ограничение на размер результирующего множества, агрегацию по числу или размеру пакетов. В качестве результата запроса возвращается поток событий, который представляет собой бесконечный поток индексированных по времени значений.

6.3.1.3 Синтаксис языка запросов представлен на рисунке 6.1.

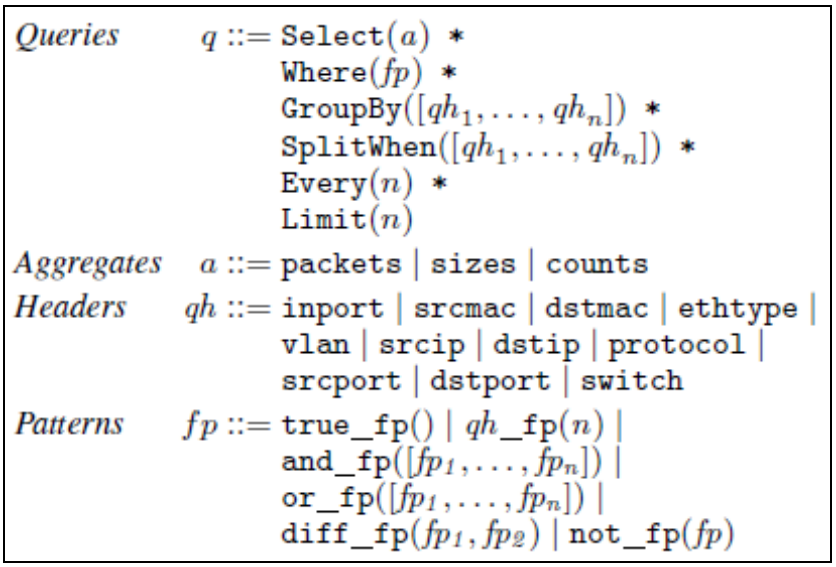

Рисунок 6.1 – Синтксис языка запросов Frenetic

– Оператор Select(a). Агрегирует результаты, возвращаемые последующими операторами в запросе, с помощью вспомогательной функции *a*, которая может быть:

1) *packets* – возвращать пакеты сами по себе;
2) *counts* – возвращает число пакетов;
3) *bytes* – возвращает сумму размеров всех пакетов.

– Оператор *Where(fp)*. Задает критерии отбора результирующих значений, оставляя только те, которые удовлетворяют условию-фильтру *fp*. В качестве условий могут выступать, например: коммутатор (*switch*), входящий порт (*inport*), MAC адрес отравителя (*srcmac*), IP адрес получателя (*destip*). Более сложные условия можно конструировать с помощью операций пересечения (*and_fp*), объединения (*or_fp*), вычитания (*diff_fp*), дополнения (*not_fp*).

– Оператор *GroupBy([qh1,...,qhN])*. Разделяет полученные результаты на подмножества согласно заголовкам пакетов от *qh1* до *qhN*. Например, можно сгруппировать в одно подмножество все пакеты с отправителя с IP адресом 10.0.0.1 и TCP портом 80, а в другое подмножество все пакеты от отправителя с IP адресом 10.0.0.2 и TCP портом 21.

– Оператор *SplitWhen([qh1,…,qhN])*. Так же, как и *GroupBy*, разбивает результаты на подмножества согласно заголовках пакетов от *qh1* до *qhN*. Однако эти условия задают условия, когда нужно создать новое подмножество при анализе полученных результатов. Например, пусть получены три пакета с IP адресами отправителей 10.0.0.1, 10.0.0.2, 10.0.0.1. В случае *SplitWhen* будет создано три подмножества, а в случае *GroupBy* только два.

– Оператор *Every(n)*. Объединяет пакеты за *n* секунд в одно подмножество.

– Оператор *Limit(n).* Ограничивает число пакетов в каждом подмножестве до *n*. По умолчанию *n* равно 1.

Пример запроса по мониторингу web трафика представлен на Рисунке 6.2. Он выводит каждые 30 секунд объем web трафика (80 порт), приходящего на второй порт коммутатора.

```
def web_query():
  return \
    (Select(sizes) *
     Where(inport_fp(2) & srcport_fp(80))) *
     Every(30))
```

Рисунок 6.2 – Пример запроса Frenetic по мониторингу веб трафика

6.3.1.4 Язык управления политиками маршрутизации в сетях. Язык управления политиками маршрутизации основан на использовании библиотек функционального и реактивного программирования (FRP). Такие библиотеки, как правило, используются для управления роботами, например, Yampa [27].

При этом стоит отметить, что язык управления политиками маршрутизации тесно интегрирован с языком запросов, описанным ранее (см. п.6.3.1.2).

Одной из основных операций во Frenetic программах является конструирование и установка правил пересылки пакетов (*rules*). Правила создаются с помощью конструктора, принимающего на вход шаблон (*pattern*) и список

действий (*actions*). Шаблон очень похож на условия фильтрации из языка запросов за одним исключением, что здесь нельзя использовать параметр switch. Действия включают в себя отправку пакета на указанный порт *p* (*forward(p)*), широковещательную рассылку на все порты (*flood()*), отправку пакета на контроллер (*controller()*) и изменение поля заголовка пакета *f* на значение *v* (*modify(f,v)*). Отсутствие какого-либо правила означает сброс пакета.

Под политикой маршрутизации понимается список правил маршрутизации, заданных в каждом коммутаторе.

Frenetic предоставляет возможность управления политиками через механизм событий, информирующих программу об изменении состояния сети. Например, события *SwitchJoin* (появление нового коммутатора в сети), *SwitchExit* (отключение коммутатора от сети), *PortChange* (появляение нового порта на заданном коммутаторе). Frenetic программа дальше анализирует эти события, используя специальную функцию событий (*event functions*), и управляет политиками, используя заранее заданные функции, такие как создание и добавление новых правил (*MakeForwardRules* и *AddRules*). Список выбранных операторов представлен на Рисунке 6.3.

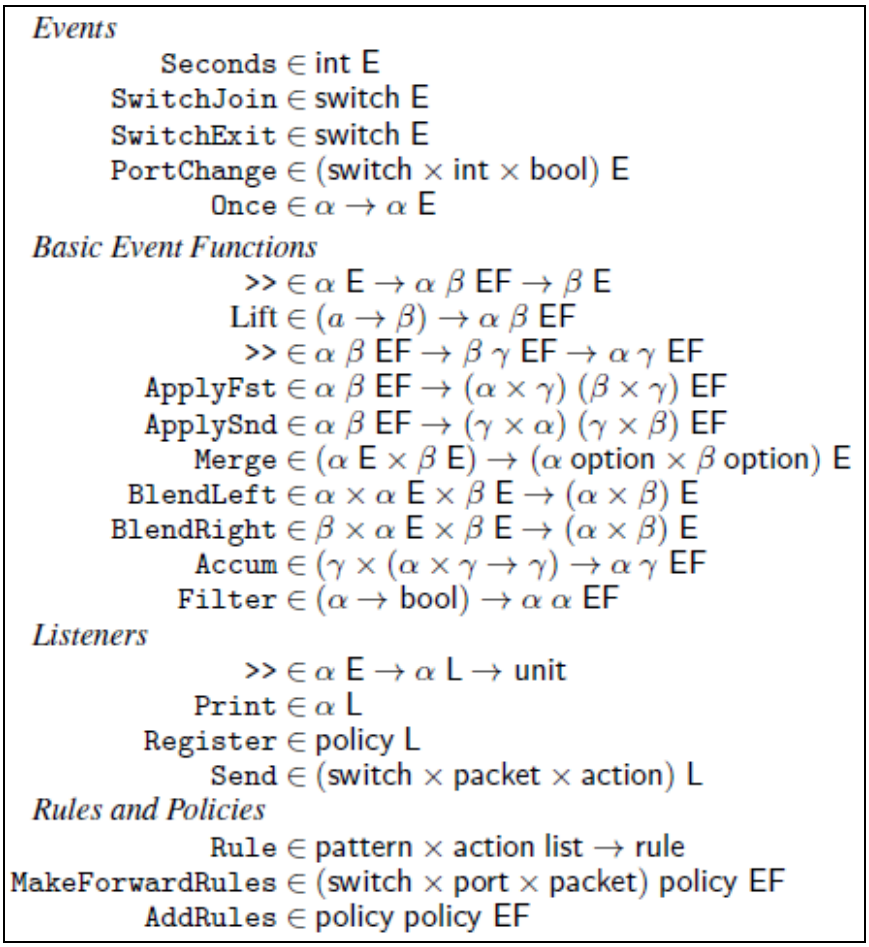

Рисунок 6.3 – Примеры операторов Frenetic

На Рисунке 6.4. представлен простой пример по созданию политики маршрутизации. Пример состоит из добавления двух правил. Первое правило – это пересылка всех пакетов с первого порта на второй порт. Второе правило – это обратная пересылка всех пакетов со второго порта на первый порт.

```
rules = [Rule(inport_fp(1),[forward(2)]),
         Rule(inport_fp(2),[forward(1)])]
def repeater():
  (SwitchJoin() >>
   Lift(lambda switch:{switch:rules}) >>
   Register())
```

Рисунок 6.4 – Пример задания политики маршрутизации в языке Frenetic

Далее представлен более сложный пример – балансировщик нагрузки (см. Рисунок 6.5). Алгоритм балансировки разбивает приходящие запросы по IP адресам

и каждая группа IP адресов направляется на заданый для неё порт. Соответствие может строиться динамически на основе языка запросов к сети. При этом Frenetic будет сам объединять правила по мониторингу и маршрутизации сети.

```
# query returning one packet per source IP
def src_ips() =
  return (Select(packets) *
          Where(inport_fp(1)) *
          GroupBy([srcip]) *
          Limit(1))

# helper to add switch to a port-packet pair
def add_switch(port,packet):
  return (switch(header(packet)),port,packet)

# parameterized load balancer
def balance(balancer):
  return \
    (src_ips()          >># (IP*packet) E
     ApplyFst(balancer) >># (port*packet) E
     Lift(add_switch)   >># (switch*port*packet) E
     MakeForwardRules() >># policy E
     AddRules())          # policy E
```

Рисунок 6.5 – Балансировка нагрузки на языке Frenetic

6.3.2 NetCore

NetCore (Network Core Programming) [28] является развитием языка Frenetic. NetCore предлагает более высокоуровневый язык по заданию политик маршрутизаци:

1) Можно использовать произвольные функции по обработке пакетов, а не только заранее заданные.
2) Можно использовать групповые символы (wildcards), а не только правила с точным сопоставлением заголовка пакетов.
3) Можно задавать правила проактивно (до того момента, как они понадобятся).

Всё это является первым шагом к возможности задания динамических правил маршрутизации, которые будут учитывать текущее состояния сети. Кроме того, NetCore дополнительно предоставляет возможность анализа истории трафика.

Простейший вариант NetCore программы представляет собой статическую политику маршрутизации, определенную с помощью предиката *e* и множества портов *S* с номера портов, на которые нужно отправить полученные пакеты. Простейшая версия предиката – это указание битовой маски. Например, SrcAddr:10.0.0.0/8 определяет, что первые восемь бит адреса отправителя должны равняться 10. Более сложные предикаты строятся с помощью операция объединения, пересечения, отрицания и дополнения. Полный синтаксис языка представлен на Рисунке 6.6.

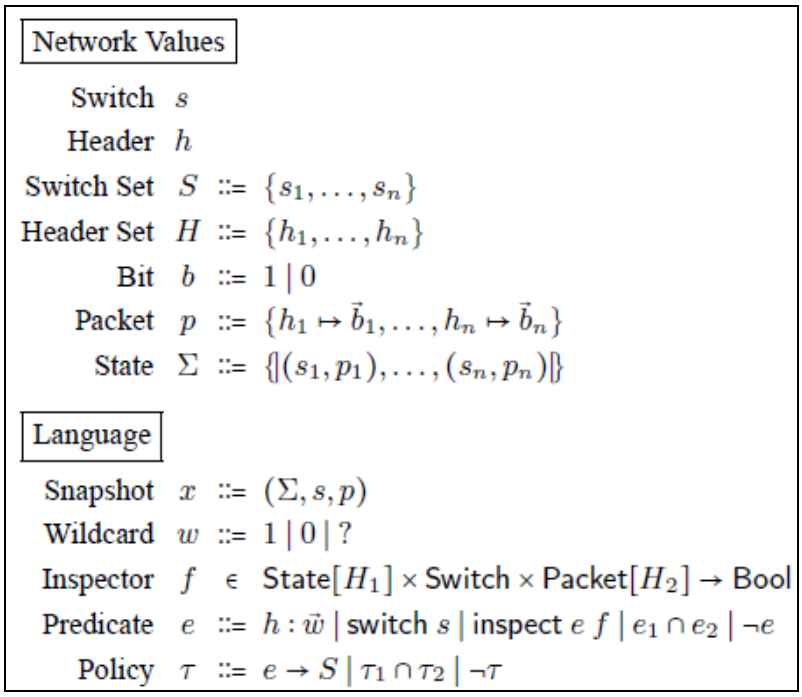

Рисунок 6.6 – Синтаксис языка NetCore

На рисунке 6.7. приведен пример политики, которая задает, что все пакеты отправители из подсети 10.0.0.0/8 должны быть перенаправлены на порт номер 1, за исключением пакетов идущих с адресов 10.0.0.1 или идущих на порт 80.

```
SrcAddr:10.0.0.0/8 \ (SrcAddr:10.0.0.1 ∪ DstPort:80)
→ {Switch 1}
```

Рисунок 6.7 – Пример статической политики маршрутизации в языке NetCore

Далее рассмотрим пример задания динамических политик маршрутизации на примере создания приложения, реализующего внутри сетевую аутентификацию, который приведен на Рисунке 6.8.

Пусть задана внутренняя сеть Network 1 и внешняя сеть Network 2. Сервер A представляет сервер аутентификации конечных узлов из первой сети на возможность доступа во вторую сеть. Тогда политика будет заключатется в следующем: пересылка всех не аутентифицированных пакетов из внутренней сети на сервер аутентификации A, для разрешенных пакетов прямая пересылка во внешнию сеть и пересылка пакетов с сервера A и с внешней сети обратно во внетреннюю сеть.

```
   (InPort:Network 1 ∩ inspect ps auth → {Network 2})
 ∪ (InPort:Network 1 ∩ ¬(inspect ps auth) → {Server A})
 ∪ (InPort:Server  A ∪ InPort:Network 2 → {Network 1})

where

 ps               = InPort:Server A
 auth (Σ,s,p)     = any (isAddr p) Σ
 isAddr p (_,p')  = p.SrcAddr == p'.DstAddr
```

Рисунок 6.8 – Пример динамической политики маршрутизации в языке NetCore

Заметим, что указанная политика использует предикат-инспектор (inspector predicate) для определения разрфешенного трафика из внутренней сети. Такой предикат *inspect e f* имеет два аргумента: фильтр *e* по всей истории трафика и решающая булева функция *f*. Фильтр формирует состояние контроллера, которое представлено множеством всех отобранный пакетов. Решающая функция *f* принимает решение о возможности пересылки пакета.

6.3.3 Nettle

Nettle [29] является предшественником Frenetic. Это был первый подход, который использовал функциональный реактивный язык (FRP) для программирования OpenFlow коммутаторов. Nettle принимает на вход поток OpenFlow событий (например, *switch_join*, *port_change*, *packet_in*) и выдает поток OpenFlow сообщений (*install*, *uninstall*, query_stats). Nettle представляет более низкий уровень в разработке сетевых приложений. С другой стороны Frenetic – это надстройка над контроллером NOX, в то время как Nettle представляет собой полноценное решение для разработки не только отдельных приложений, но и контроллера в целом.

На Рисунке 6.9 изображна архитектура Nettle.

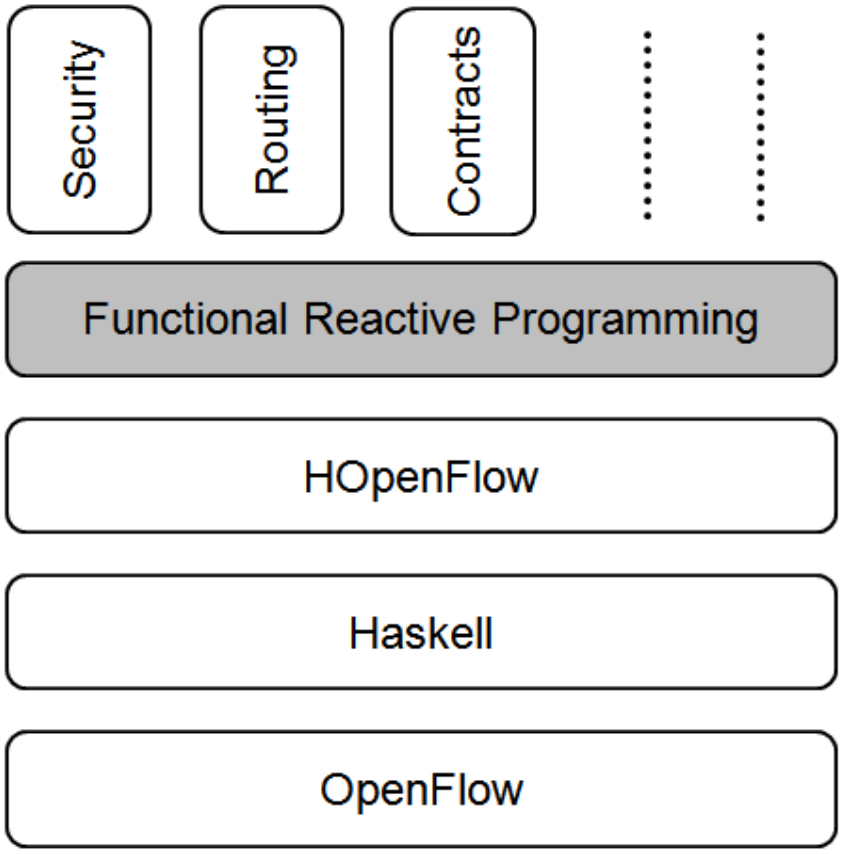

Рисунок 6.9 – Архитектура Nettle

Самый нижний уровень представляет OpenFlow коммутаторы. Уровень выше – это язык высокоуровненого функционального программирования Haskell. Дальше библиотека HOpenFlow для работы по протоколу OpenFlow. Следующий уровень представляет собой подход к функциональному реактивному программированию, который используется для программирования интерактивных систем в декларативном стиле. На последнем уровне находятся разрабатываемые сетевые приложения: безопасность, мониторинг, маршрутизации и д.р. Nettle гарантирует и проверяет, что правила установленные разными приложениями не будут влиять друг на друга.

Nettle основан на понятии сигнальной функции *SF* (signal function), которая преобразует входные переменный или сигналы в выходные. Такие функции так же могут иметь некоторое состояние. Например, сигнальная функция, которая переводит поток событий типа *M1* в поток событий типа *M2* будет иметь определение вида *SF (Event M1) (Event M2)*. Nettle сама по себе представляет сигнальную функцию, которая переводит сообщения от коммутатора в набор команд *SF (Event SwitchMessage) (Event SwitchCommand)*. Кроме того Nettle имеет ряд предопределенных функций для работы с событиями, например, *hold::a -> SF (Event a) a*, которая постоянно выдает одно и тоже событие *a*.

В качестве примера рассмотрим реализацию L2 коммутации для Nettle. L2 маршрутизатор поддерживает таблицу соответствия MAC адресов и портов, на которых появлялись сообщения с указанного адреса. При получении нового пакета, проверяется наличие адреса получателя в указанной таблице. В случае наличия данного адреса в таблице происходит отправка пакета на соответствующий порт. Если же соответствия не было найдено, то происходит широковещательная рассылка на все порты.

На рисунке 6.10 показана базовая функция L2 коммутации для Nettle.

```
packetToCmd hostDirTable (swid, pktInfo) =
  do ps ← lookup (swid, s) hostDirTable
     pr ← lookup (swid, r) hostDirTable
     return (makeCommand swid s ps r pr)
  where ethFrame = packetInFrame pktInfo
        (s, r)   = (sourceAddress ethFrame, destAddress ethFrame)
```

Рисунок 6.10 – L2 коммутация для Nettle

При получении нового пакета проверяется наличии адресов в таблице. Если адреса есть, то строится правило пересылки заданного пакета на полученный порт согласно Рисунку 6.11, если же нет – то *Nothing*, что означает, что нужно заполнить таблицу соответствия.

```
makeCommand swid s ps r pr =
  deleteRules (ethSourceDestAre s r ∨ ethSourceDestAre r s)) swid ⊕
  insertRule (flowFromTo s ps r pr) swid ⊕
  insertRule (flowFromTo r pr s ps) swid
  where flowFromTo s ps r pr =
          ((inPortIs ps ∧ ethSourceDestAre s r) ⟹ sendOnPort pr)
          `expireAfterInactiveFor` 30
```

Рисунок 6.11 – Пример создания нового правила для Nettle

6.4 Выводы

Существующие языки программирования политик маршрутизации облегчают создание новых приложений. Каждый из языков обладает своими преимуществами.

Язык NetCore вводит понятие состояния контроллера, как историю всех пакетов, которые когда-либо проходили через коммутаторы сети. Язык NetCore разрешает динамические политики, требуя при этом явного указания параметров заголовка, от которых зависит политика. Последние ограничения хоть и не являются критически важными – компилятор языка сможет синтезировать необходимые правила OpenFlow и без них, однако отказ от использования таких указаний приводит к серьёзному ухудшению показателей производительности системы.

Описания множества политик на языках Frenetic и NetCore нуждаются в предварительной компиляции, в результате которой получается управляющее приложение, соответствующее всем выбранным политикам.

Система Nettle представляет собой совокупность интерпретатора соответствующего языка и контроллера, которая самостоятельно разрешает возможные конфликты между управляющими приложениями непосредственно в процессе их работы.

Среда выполнения Frenetic автоматически обрабатывает «вторые» пакеты, пересланные коммутатором уже после того, как приложение произвело обработку соответствующего потока по его первому пакету.

Однако, чтобы построить по-настоящему надёжную сеть передачи данных, необходимо иметь возможность проверять синтезируемые приложением правила на противоречивость предписанным спецификациям. При этом существует несколько альтернативных вариантов реализации такой проверки. Один из них – непосредственная проверка синтезируемых приложением правил OpenFlow через средства динамической проверки спецификации.

Перспективным же является способ проверки спецификаций через формальную верификацию программ. Осуществить формальную верификацию управляющего приложения, написанного на универсальном языке программирования, не представляется возможным в силу чрезмерной вычислительной сложности. Однако, верификация приложения написанного на специальном ограниченном подмножестве такого языка представляется вполне возможной. Поэтому актуальной становится задача создания такого языка, выразительной мощности которого хватает для описания несложных управляющих приложений, а свойства позволяют проводить верификацию простейших свойств за приемлемое время.

7 ПРОВЕДЕНИЕ АНАЛИЗА СУЩЕСТВУЮЩИХ ПОДХОДОВ К ВИРТУАЛИЗАЦИИ СЕТЕЙ, ВКЛЮЧАЯ ПОДХОД НА ОСНОВЕ ТЕХНОЛОГИИ ПКС

7.1 Принципы виртуализации сетей. Применение виртуализации

Виртуализация сетей — это технология, предполагающая абстракцию логического представления сети от реального физического устройства сети передачи данных.

Различают два вида виртуализации сетей:

— *Внешняя виртуализация* (external) — технологии разделения одной физической сети несколькими логическими сетями или, наоборот, объединения нескольких физических сетей в одну логическую.

— *Внутренная виртуализация* (internal) — виртуализация сетей внутри одного сервера, подразумевающая создание логических сетей между виртуальными машинами без задействования физической сети передачи данных.

Виртуализация сетей традиционно используется для объединения нескольких физически распределенных локальных сетей в единую логическую сеть и для предоставления удаленного доступа к локальной сети (технология VPN), а также для разделения физической сети на логические сети (технология VLAN).

Сравнительно новой областью применения виртуализации сетей является использование виртуальных сетей в центрах обработки данных (ЦОД). Для ЦОД, особенно предоставляющих услугу Infrastructure-as-a-Service, необходимо обеспечивать разделение трафика большого количества пользователей, прозрачность взаимодействия внутри логической сети одного пользователя, а также иметь возможность централизованного и автоматизированного управления сетевой инфраструктурой. Также возникают задачи внутренней виртуализации сетей, т. е. создание сетей виртуальных машин внутри одного физического сервера.

Дополнительные сложности создает возможность миграции виртуальных машин между серверами.

Особенности виртуализации сетей ЦОД привели к созданию новых технологий организации логических сетей, таких как VXLAN, NVGRE и STT, а также к развитию комплексных систем управления виртуальными сетями (в том числе на основе программно-конфигурируемых сетей).

7.2 Основные традиционные технологии виртуализации сетей

7.2.1 Технология VLAN

7.2.1.1 VLAN — традиционное средство разделения физической сети Layer 2 на логические сети. Узлы, относящиеся к одной логической сети (VLAN), могут взаимодействовать друг с другом так, как если бы они были подключены к одному широковещательному домену, независимо от их физического расположения. VLAN изолированы друг от друга, трафик между ними может передаваться только через маршрутизатор.

Для обозначения членства во VLAN существуют разные подходы, описанные ниже.

7.2.1.2 Идентификация принадлежности по порту (port-based). Данный подход является наиболее распространенным и поддерживается большей частью существующего сетевого оборудования.

Способ разделения физической сети на логические подсети зависит от используемого сетевого оборудования. В простейшем случае коммутатор определяет принадлежность трафика к той или иной VLAN по тому, на каком порту он был получен, таким образом, каждая VLAN требует отдельного соединения с коммутатором. Если же требуется передавать данные из разных VLAN через один порт, то данный порт должен быть магистральным (магистральный порт, или транк — канал типа «точка-точка» между коммутатором и другим сетевым

оборудованием, по которому может передаваться трафик из различных VLAN). При передаче кадра через транк коммутатор тегирует его в соответствии с тем, из какой VLAN он был получен, отдельная VLAN выделена для всего нетегированного трафика.

Для передачи информации о принадлежности к VLAN применяется протокол стандарта IEEE 802.1Q [30], согласно которому в заголовок Ethernet кадра помещается тэг. Тэг имеет размер 4 байта и содержит поле VLAN ID, которое определяет принадлежность к VLAN. Таким образом, протокол поддерживает создание не более 4096 логических сетей на основе одной физической сети.

7.2.1.3 Идентификация принадлежности по MAC адресу (MAC-based). Если сетевое оборудование поддерживает данную технологию, то членство узлов в логической сети может быть задано при помощи таблицы соответствия между MAC адресом и VLAN ID.

7.2.1.4 Идентификация принадлежности по протоколу (protocol-based). Принадлежность к логической сети определяется по тому, данные какого протокола 3-4 уровня содержатся в Ethernet кадре. Например, одна VLAN может объединять весь IP трафик. Недостаток данного подхода заключается в том, что нарушается независимость уровней модели ISO/OSI.

7.2.1.5 Идентификация принадлежности по аутентификации (authentication-based). Данный подход предполагает использование протокола контроля доступа и аутентификации стандарта IEEE 802.1X: при подключении к сети каждый узел проходит процедуру аутентификации и на основании данных аутентификации помещается в ту или иную VLAN.

7.2.1.6 Преимущества технологии VLAN:

— Возможность объединять узлы в логические сети независимо от их физического расположения и топологии физической сети.

— Т.к. каждая VLAN представляет собой один широковещательный домен, в

случае широковещательных рассылок трафик передается только между узлами одной VLAN. За счет этого оптимальнее используется полоса пропускания каналов и вычислительные ресурсы сетевого оборудования.

— Технология VLAN поддерживается большинством существующего на сегодняшний день сетевого оборудования.

7.2.1.7 Недостатки технологии VLAN:

— Логические сети разделяют общее физическое оборудование, таким образом, трафик одной логической сети может оказывать влияние на трафик другой логической сети. Для обеспечения справедливого разделения ресурсов сети могут понадобится дополнительные механизмы QoS. Также для повышения пропускной способности транков может потребоваться агрегация каналов.

— Количество логических сетей не может превышать 4096, чего может быть недостаточно, например, для центров обработки данных.

7.2.2 Технология VPN

Virtual Private Network (VPN) — общее название технологий, позволяющих создавать виртуальные частные сети поверх публичных сетей.

Технология VPN может использоваться для связи нескольких физически распределенных локальных сетей, например, сетей филиалов компании, для предоставления удаленного соединения с локальной сетью отдельному узлу или для связи типа «точка-точка» поверх незащищенной сети. Уровень доверия к создаваемой логической сети не зависит от уровня доверия к публичной сети, что достигается за счет использования различных средств криптографии и туннелирования.

Туннелирование — процесс, в ходе которого создается логическое соединение между двумя конечными точками посредством инкапсуляции различных протоколов. Инкапсулируемый протокол относится к тому же или более низкому уровню сетевой модели, чем протокол, используемый в качестве туннеля. Чаще

всего в качестве протокола, на основе которого осуществляется туннелирование в VPN, используется протокол IP.

С точки зрения требуемого уровня доверия сети VPN можно разделить на 2 группы:

– *Защищенные VPN* — наиболее распространенный вариант виртуальных сетей. Позволяют создавать защищенную сеть поверх ненадежной сети (например, Интернет). Технология опирается на использование средств шифрования, что приводит к снижению скорости передачи данных в таких сетях. Примеры: IPSec, OpenVPN, PPTP.

– *Доверительные VPN* – используются в том случае, когда уровень доверия к сети передачи данных достаточно высок, и виртуальные сети используются преимущественно для того, чтобы разделить трафик. Примеры: MPLS и L2TP.

Туннелирование и шифрование в сетях VPN может быть реализовано как аппаратно, что позволяет достичь большей скорости передачи данных, так и при помощи только программных средств.

Примеры наиболее распространенных на сегодняшний день технологий VPN:

– *IPSec* [31] представляет собой набор протоколов (надстройки над протоколом IP), которые используются для обеспечения надежного соединения поверх IP. В IPSec входят протоколы обеспечения целостности виртуального соединения и аутентификации, шифрования и обмена секретными ключами. IPSec поддерживает два режима: канальный и туннельный. В канальном режиме шифруется только содержимое IP пакета, все заголовки остаются неизменными. В туннельном режиме пакет шифруется целиком вместе с заголовком IP, и в зашифрованном виде инкапсулируется в новый IP пакет. С точки зрения безопасности туннельный режим является более предпочтительным.

– *PPTP* [32] используется для организации туннелей типа «точка-точка» между двумя узлами или двумя локальными сетями. Используется инкапсуляция кадров PPP (канальный протокол для соединений «точка-точка») в IP пакеты с

использованием протокола инкапсуляции GRE (протокол GRE инкапсулирует пакеты сетевого уровня в IP-пакеты с добавлением специального заголовка GRE). Может применяться шифрование с помощью MPPE и различные методы аутентификации, однако PPTP считается менее надежным, чем IPSec (был обнаружен ряд уязвимостей в используемых механизмах шифрования и аутентификации).

– *L2TP* [33] — протокол туннелирования канального уровня. Управляющие и информационные пакеты L2TP пересылаются в пакетах UDP. Для передачи информации в информационных пакетах используется протокол PPP. L2TP сам по себе не осуществляет шифрование, для обеспечения безопасности соединения используются средства туннелирующих протоколов, например, IPSec.

– *PPoE* [34] — сетевой протокол канального уровня, использующий передачу кадров PPP через Ethernet. Предоставляет дополнительные возможности, такие как аутентификация, шифрование и сжатие данных.

– *OpenVPN* [35] позволяет создавать как туннели третьего уровня для передачи IP пакетов, так и туннель второго уровня для передачи Ethernet кадров. Соединение устанавливается поверх TCP или UDP. Для шифрования используется SSL/TLS, поддерживается аутентификация на основе ключей, сертификатов или пары логин-пароль.

– *MPLS* [36] — протокол канального уровня, осуществляющий коммутацию в сети на основе меток, присвоенных каждому пакету, без изучения содержимого пакета и его заголовков. К каждому пакету может быть добавлена одна или несколько меток, идентифицирующих виртуальный канал. При получении пакета маршрутизатор ищет метку в таблице коммутации (вместо поиска IP в таблице маршрутизации), и на основе метки принимает решение о направлении пакета на тот или иной порт. MPLS позволяет инкапсулировать протоколы любого уровня и создавать виртуальные каналы, независящие от среды передачи данных. MPLS может быть использован для создания VPN сетей с использованием инскапсуляции протоколов второго или третьего уровня.

— *PBB* [37] использует технологию инкапсулирования mac-in-mac, т. е. инкапсулирует Ethetnet-кадры пользователя в Ethernet-кадры провайдера, для чего вводится новый заголовок кадра Ethernet. PBB позволяет изолировать домены сетей пользователей и провайдера.

7.3 Технологии организации крупномасштабных логических сетей L2

7.3.1 Перечень основных технологий по преодолению ограничений

Стандартная технология VLAN позволяет создавать не более 4096 виртуальных сетей, чего может быть недостаточно для ЦОД (например, если они предоставляют услугу Infrastructure-as-a-Service — IaaS). Для преодоления этого ограничения были предложены несколько технологий, позволяющих организовывать логические сети L2 поверх сетей уровня L3:

— Virtual eXtensible Local Area Netrwork, VXLAN[38] (VMware, Cisco и др.).

— Network Virtualization using Generic Route Encapsulation, NVGRE[39] (Microsoft и др.).

— Stateless Transport Tunneling, STT [40] (Nicira и др.).

Данные технологии являются конкурирующими разработками различных компаний, специализирующихся на управлении ЦОД, и имеют похожую логику работы, различаясь деталями реализации.

Общим для всех этих технологий является использование технологии туннелирования для изоляции трафика различных логических сетей, для увеличения количества поддерживаемых сетей и для упрощения процедуры миграции узлов внутри физической сети без внесения изменений в логические сети.

Идея использования туннелирования для организации виртуальных сетей заключается в следующем. Каждой виртуальной сети присваивается идентификатор. Кадр, отправленный одним конечным узлом проходит через ближайшую конечную

точку туннеля, где он инкапсулируется во внешний пакет протокола, который используется в сети в качестве транспортного. При этом к каждому кадру добавляется заголовок протокола туннелирования, который содержит идентификатор логической сети. Далее данная конечная точка туннеля определяет, к какой из конечных точек подключен узел-получатель, и пересылает на нее пакет, содержащий исходный кадр и внешние заголовки протоколов. Для маршрутизации пакета между двумя конечными точками туннелей используется внешний заголовок инкапсулированного пакета. При поступлении пакета на вторую конечную точку туннеля производится извлечение инкапсулированного кадра и на основании идентификатора логической сети и внутреннего заголовка кадра определяется его получатель.

7.3.2 Параметры сравнения технологий

Все три технологии — VXLAN, NVGRE и STT — придерживаются приведенной схемы передачи данных, однако различаются деталями реализации.

Для сравнения данных технологий будем их рассматривать со следующих точек зрения:

– Процедура инкапсуляции. Сюда входит используемый протокол туннелирования, транспортный протокол (т. е. протокол, в который инкапсулируется пакет логической сети и который используется для маршрутизации между конечными точками туннеля), способ идентификации логических сетей, а также реализация процедуры инкапсуляции на аппаратном или программном уровне.

– Уровень управления. Сюда входит описание процедуры принятия решения о направлении пакета на ту или иную конечную точку туннеля, а также механизмы, которые должны поддерживаться сетью для маршрутизации пакетов между конечными точками туннелей.

– Уровень представления логических сетей. Сюда входят особенности реализации логических сетей с точки зрения узлов данной сети: прозрачность

взаимодействия, уровень сетевой модели, используемый в сети, поддержка разбиения на логические подсети внутри одной логической сети, а также поддержка независимости адресных пространств различных логических сетей.

7.3.3 Сравнение по реализации процедуры инкапсуляции

7.3.3.1 На сегодняшний день все три технологии передают L2 кадры поверх L3 сетей.

7.3.3.2 Описание инкапсуляции в VXLAN. Для идентификации логической сети используется 24-битное поле VNI. При инкапсуляции к кадру добавляется заголовок VXLAN, далее он упаковывается в UDP-пакет. Формат Ethernet-кадра, передаваемого между конечными точками туннелей (в технологии VXLAN они называются VXLAN Tunnel End Point — VTEP) показан на Рисунке 7.1.

Рисунок 7.1 – Формат Ethernet-кадра VXLAN

Внешние заголовки Ethernet и IP определяются адресами конечных точек туннеля. Заголовки UDP и VXLAN имеют следующий вид:

- UDP-заголовок:
    1) Source Port — значение устанавливается VTEP отправителя.
    2) VXLAN Port — порт, обозначающий использование VXLAN.
    3) UDP Checksum — контрольная сумма пакета.
- VXLAN Header:
    1) VXLAN Flags — различные флаги.

2) VNI (VXLAN ID) — идентификатор VXLAN.

3) Reserved — зарезервированные поля.

7.3.3.3 Описание инкапсуляции в NVGRE. Для идентификации логической сети используется Virtual Subnet Identifier (VSID) (24 бита). Для передачи Ethernet-кадров между узлами одной логической сети перед кадром добавляется стандартный заголовок протокола GRE, поле Key которого используется для указания VSID. Далее кадр с GRE-заголовком инкапсулируется в IP пакет, который и передается в транспортную сеть. Формат Ethernet-кадра, передаваемого между конечными точками туннелей, показан на Рисунке 7.2.

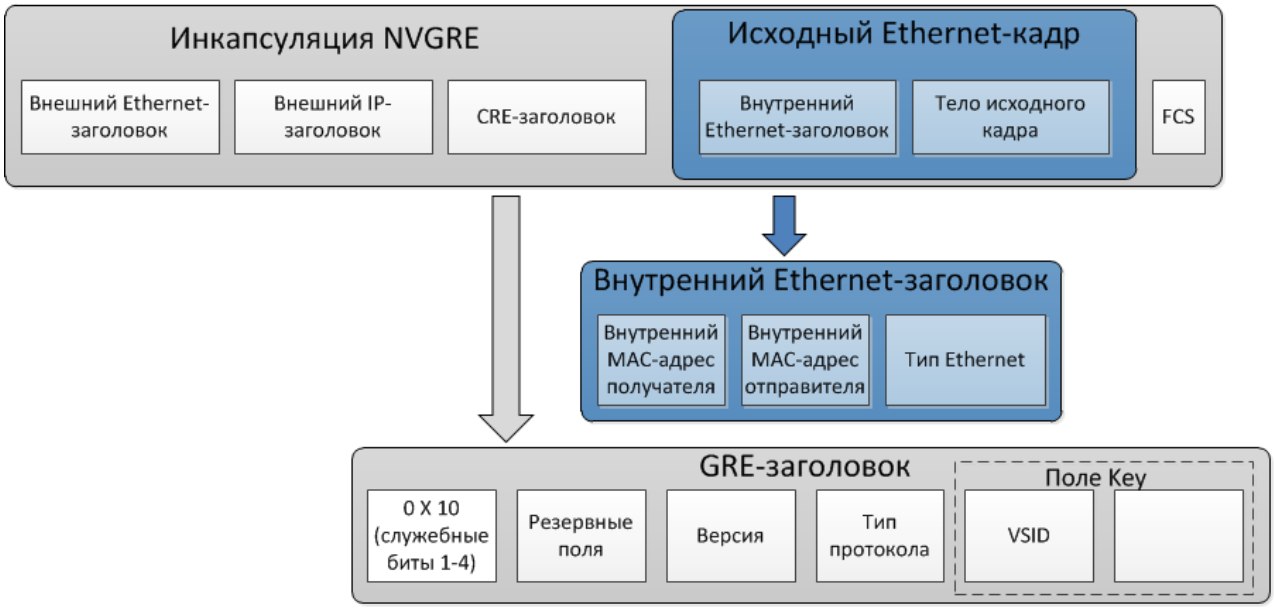

Рисунок 7.2 – Формат Ethernet-кадра NVGRE

Внешние заголовки Ethernet и IP определяются адресами конечных точек туннеля. Заголовок GRE имеет следующую структуру:

1) служебные биты (первые 4 бита заголовка) доджны иметь следующие значения 0-X-1-0 (т. е. поле Checksum в заголовке отсутствует, наличие поля Routing неважно, поле Key присутствует, поле Sequence Number отсутствует);

2) резервные поля;

3) версия протокола;

4) тип протокола;

5) поле Key (32 бита), 24 бита из которых используются для указания VSID.

7.3.3.4 Описание инкапсуляции в STT. В качестве протокола инкапсуляции используется Stateless Transport Tunneling. При инкапсуляции перед кадром добавляется заголовок STT, поле Context ID (64 бита) которого может быть использовано для указания идентификатора логической сети. Далее кадр вместе с заголовком STT инкапсулируется в TCP-пакет, при этом значения полей TCP имеют специальные значения. Особенностью протокола STT является возможность разбиения исходных кадров на несколько фрагментов и их передача в нескольких пакетах транспортного уровня. При этом заголовок STT включается только в кадр, содержащий первый сегмент исходного кадра. Общая структура Ethernet-кадров, содержащих сегменты STT, представлена на Рисунке 7.3.

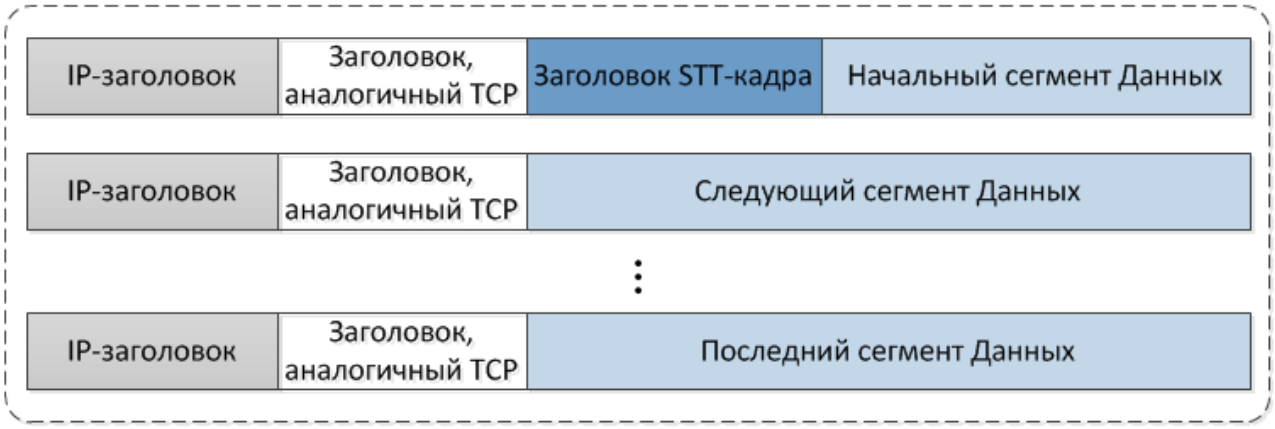

Рисунок 7.3 – Формат Ethernet-кадров STT

Как и для первых двух технологий, заголовки Ethernet и IP полностью определяются адресами конечных точек туннеля. Заголовки TCP и STT имеют следующий вид:

– TCP заголовок:

1) *Destination Port* должен находиться в диапазоне 1024-49151.

2) *Source port* для всех фрагментов одного кадра должен иметь одинаковое значение (рекомендуется использовать значение в диапазоне

4912-65536). Это требуется для корректной сборки кадра, разбитых на несколько фрагментов, на стороне получателя.

3) *Sequence number*: старшие 16 бит задают длину кадра STT в байтах, младшие 16 бит указывают смещение данного фрагмента в исходном кадре.

4) *Acknowledgment number* используется в качестве идентификатора исходного кадра, разбитого на сегменты, и должно иметь одинаковое значение для всех фрагментов одного кадра.

5) Поля *Window* и *Urgent pointer* не используются протоколом, рекомендуется выставлять их равными 0.

6) Значения других полей не специфицировано.

– STT заголовок. Состав STT заголовока приведен на Рисунке 7.4:

1) *Версия* — версия протокола (текущая — 0).

2) *Флаги* — флаги, описывающие параметры инкапсулированного пакета: расчет контрольной суммы, использование TSO отправителем, использование IPv4 или IPv6, использование TCP.

3) *Смещение L4* — сдвиг в байтах от конца заголовка кадра STT до инкапсулированного заголовка TCP/UDP.

4) *MSS* — максимальный размер TCP-сегмента, используемый конечной точкой туннеля, которая передает кадр в другую сеть.

5) *Priority Code Point*, указывается, если конечная точка туннеля передает пакет в другую сеть.

6) *V* — флаг, который указывает на наличие параметра VLAN ID в следующем поле и параметра PCP в предыдущем поле;

7) *VLAN ID* — VLAN tag, который должен быть добавлен в заголовок инкапсулированного пакета при передаче в другую сеть.

8) *Context ID* — поле, на основе которого определяется получатель пакета. Например, может быть использовано для указания идентификатора сети.

Рисунок 7.4 – Формат заголовка STT

7.3.4 Сравнение технологий

За инкапсуляцию исходных кадров во всех технологиях отвечают конечные точки туннеля. При этом VXLAN предполагает использования отдельно шлюза VXLAN для передачи данных из внутренней сети в сеть, не поддерживающую VXLAN. Другие технологии возлагают эту функцию также на конечные точки туннеля. При передаче кадра из внутренней сети во внешнюю шлюз (или конечная точка) должен извлечь кадр из пакета транспортного протокола и на основе идентификатора логической сети и внутреннего Ethernet-заголовка определить порт, через который передать его во внешнюю сеть (например, передать его в некоторую VLAN).

Шлюзы и конечные точки туннеля могут быть реализованы как аппаратно, так и программно, например, являться частью гипервизора. При этом стоит заметить, что VXLAN и NVGRE ориентированы, в основном на использование физического сетевого оборудования в качестве конечных точек туннелей, что позволяет ускорить процесс инкапсуляци и извлечения кадра, который реализован непосредственно в сетевом оборудовании. С другой стороны, данные технологии зависят от поддержки протокола туннелирования оборудованием.

Технология STT, напротив, ориентирована, в первую очередь, на использование с виртуальными коммутаторами и аппаратным ускорением в сетевых картах (технологий TSO и LRO). С этим связан выбор формата заголовка, совпадающего с заголовком TCP, так как на сегодняшний день в сетевые карты не включается поддержка протоколов туннелирования. Использование TCP позволяет

использовать технологии TSO и LRO, реализованные в сетевых картах. Данные технологии применяются для сегментации блоков данных протокола TCP больших размеров на несколько более мелких и их обратной сборки аппаратными средствами сетевой карты, без участия центрального процессора. Таким образом, STT не требует использования специального сетевого оборудования, конечные точки туннелей реализованы программно на сервере.

7.3.5 Сравнение с точки зрения уровня управления

Концептуальным отличием STT от VXLAN и NVGRE является то, что данная технология специфицирует только технологию туннелирования, в то время, как VXLAN и NVGRE предъявляют дополнительные требования к организации управления сетью.

NVGRE и ранние версии VXLAN требуют поддержки групповых рассылок (multicast) сетевым оборудованием. Это требование необходимо для выполнения шроковещательных рассылок внутри одной логической сети, что реализуется при помощи групповых рассылок в сети передачи данных. Также эти технологии предполагают использование широковещательных рассылок для определения адреса конечной точки туннеля, к которой подключен узел-получатель.

Другой подход, реализованный в STT и частично в последних спецификациях VXLAN — отказ от привязки протокола туннелирования к конкретной логике управления сетью. STT позволяет реализовать любой механизм управления, например, использовать идеи ПКС для организации централизованного управления.

Также на реализацию управления сетью влияет используемый технологией транспортный протокол.

VXLAN использует UDP, что позволяет дополнительно использовать UDP заголовки для идентификации потоков в сети (например, для разделения пропускной способности между несколькими приложениями или для балансировки нагрузки).

Достоинством NVGRE является использование стандартного протокола GRE, который поддерживается многим сетевым оборудованием.

STT использует формат TCP заголовка, что позволяет не зависеть от поддержки протокола туннелирования оборудованием. Однако такой подход имеет и свои недостатки: трафик STT не всегда возможно отличить от обычного TCP трафика, что может приводить к некорректной обработке пакетов некоторым сетевым оборудованием (таким, как middlebox).

7.3.6 Сравнение с точки зрения уровня представления логических сетей

Все три технологии на сегодняшний день предполагают развертывание L2 логических сетей. Допускается пересечение адресных пространств, используемых в разных логических сетях, так как их трафик полностью изолирован. Взаимодействие между узлами одной логической сети является полностью прозрачным для самих узлов (стоит отметить, что это может усложнить диагностику сетевых неполадок).

VXLAN и NVGRE не поддерживают использования VLAN внутри логических сетей, так как при инкапсуляции кадра конечной точкой туннеля поле VLAN ID удаляется из Ethernet-заголовка. Если же конечная точка туннеля получит пакет с инкапсулированным кадром, содержащим VLAN ID, то такой кадр может быть сброшен. Это сделано для обеспечения совместимости между сетями VXLAN и внешней сетью, не поддерживающей VXLAN, так как при отправке кадра внешнюю сеть ему может быть присвоен другой VLAN ID, определяемый идентификатором логической сети. STT не специфицирует обработку кадров с полем VLAN ID в конечных точках туннеля, однако предоставляет поле в заголовке STT, в котором может быть указан VLAN ID для передачи во внешнюю сеть.

Другой особенностью STT является использование 64-битного поля Context ID, в котором может быть указана произвольная мета-информация (идентификатор логической сети и другая дополнительная информация), которое может интерпретироваться произвольным образом, в зависимости от механизмов управления сетью. В отличие от STT, VXLAN и NVGRE ограничивают поле идентификатора 24 битами и не предполагают использование его для иных целей.

В заключение приведем данные о поддержке технологий VXLAN, NVGRE и STT различными производителями сетевого оборудования и платформами для управления сетевой инфраструктурой ЦОД, которые представлены в Таблице 7.1.

Таблица 7.1– Поддержка VXLAN, NVGRE и STT

VXLAN	NVGRE	STT
— VMware vSphere — Cisco Nexus 1000V — Citrix XenServer — Citrix NetScaler Cloud Bridge — Open vSwitch — Arista Networks — Brocade	— Microsoft Hyper-V — Intel Ethernet Switch — Broadcom — Open vSwitch	— Nicira Network Virtualization Platform — Open vSwitch

7.4 Виртуализация сетей с использованием ПКС

7.4.1.1 Описанные выше подходы к виртуализации сетей позволяют объединить отдельные группы конечных узлов логические сети и организовать взаимодействие внутри логической сети таким образом, чтобы входящие в сеть узлы имели единое представление сети, не зависящее от их физического расположения. Однако при использовании описанных выше технологий возникает две проблемы.

7.4.1.2 Первая проблема - при использовании виртуализации в ЦОД необходимо постоянно изменять настройки сетевого оборудования и правила определения конечных точек туннелей, к которым подключена та или иная виртуальная машина, в соответствии с текущим состоянием сети. Например,

внесение изменений в конфигурацию оборудования может потребоваться при создании новых логических сетей, создании новых виртуальных машин или при миграции виртуальных машин между серверами. Таким образом, требуется некоторый интерфейс для автоматического и централизованного управления виртуальными сетями. Также данный интерфейс, по возможности, должен быть открытым и поддерживаться различными производителями сетевого оборудования.

7.4.1.3 Вторая проблема – в описанных технологиях логические сети разделяют общее физическое оборудование и, следовательно, принципы управления сетью (например, алгоритмы маршрутизации или балансировки нагрузки). То есть разделяется только уровень передачи данных, но не уровень управления. Для использования различной логики управления в нескольких логических сетях необходимо использовать дополнительные механизмы разграничения уровня управления.

7.4.1.4 Данные проблемы могут быть решены с использованием технологии программно-конфигурируемых сетей.

7.4.2 Организация управления виртуальными сетями ЦОД на основе ПКС

7.4.2.1 Основной идеей ПКС является разделение уровня управления сетью и уровня передачи данных. Применительно к управлению виртуальными сетями в ЦОД технология ПКС позволяет вынести всю логику управления сетевым оборудованием (например, конечными точками туннелей) в контроллер и таким образом централизовать и автоматизировать управление сетью.

Можно выделить две задачи, связанные с управлением сетевым оборудованием в ЦОД: автоматизация изменения настроек коммутаторов и управление передачей данных в сети.

7.4.2.2 Задачап автоматизации изменения настроек коммутаторов. Для виртуальных коммутаторов, используемых для соединения виртуальных машин

внутри одного сервера и передачи данных между несколькими серверами необходимо вносить следующие изменения в настройки:

1) Создавать, удалять виртуальные порты коммутатора, а также менять их конфигурацию при создании, удалении и миграции виртуальных машин.
2) Создавать, удалять и менять конфигурацию очередей, ассоциированных с тем или иным портом. Очереди служат для обеспечения разделения сетевых ресурсов между логическими сетями, и изменение их конфигурации может потребоваться, например, при создании или удалении логической сети.
3) Управлять объединением портов в группы. Группы портов позволяют задавать единые настройки для указанного множества портов и могут быть связаны с некоторой логической сетью.
4) Задавать настройки соединения между контроллером и коммутатором, например, IP адрес и TCP порт контроллера, использование SSL/TLS и сертификатов для аутентификации.

Для решения этих задач могут быть использованы открытые протоколы OpenFlow Management and Configuration Protocol (OF-Config)[41] или OVSDB[42]. OF-Config является открытым, независимым от оборудования протоколом для управления OpenFlow коммутаторами. OVSDB — это протокол, используемый виртуальным коммутатором Open vSwitch[43] для управления конфигурацией, однако он также является открытым и может быть реализован другими производителями сетевого оборудования.

7.4.2.3 Задача управления передачей данных в сети. Для управления таблицами коммутации виртуальных и физических коммутаторов может быть использован протокол OpenFlow[10], позволяющий задавать различные правила обработки и маршрутизации пакетов для разных классов трафика. Последние спецификации протокола (1.3 и старше) предполагают использование логических портов, пересылка пакетов на которые может включать в себя инкапсуляцию, и поля Tunnel ID, которое может содержать в себе идентификатор логической сети,

согласно используемому протоколу туннелирования (VNI для VXLAN, поле Key для GRE или Context ID для STT). Это открывает возможности для использования ПКС совместно с технологиями вирутализации сетей на основе туннелирования.

В качестве примера создания виртуальных сетей в ЦОД на основе технологии ПКС можно привести пример совместного использования платформы OpenStack Quantum[25], виртуального коммутатора Open vSwitch и контроллера Ryu[20]. Заметим, что все перечисленные компоненты являются программным обеспечением с открытым исходным кодом.

7.4.3 Разделение уровня управления — FlowVisor

Использование технологии ПКС позволяет разбивать трафик сети на различные классы (потоки) и реализовывать свою логику управления для каждого потока. При использовании в ПКС протокола OpenFlow выделение потоков основано на определении некоторого подмножества значений заголовков протоколов различных уровней и сравнении заголовков пакета с этими значениями. Данный подход открывает широкие возможности для вирутализации сетей, в том числе, для реализации различных механизмов управления для каждой логической сети.

В качестве примера использования ПКС для разделения уровня управления в виртуальных сетях можно привести средство FlowVisor [44]. FlowVisor выделяет заданные множества потоков в отдельные срезы сети (slices), каждый из которых имеет свое логическое представление и логику управления. Таким образом, срез сети определяется множеством потоков, передаваемых в данном срезе, и логическим представлением топологии сети (коммутаторы, порты коммутаторов, соединения).

При разработке FlowVisor ставилась задача обеспечить разделение различными срезами следующих ресурсов сети:

1) *Пропускная способность*: необходимо реализовать механизмы предоставления каждому срезу сети гарантированной доли пропускной способности каналов.

2) *Топология*: каждый срез сети должен иметь свое логическое представление сети, т. е. входящих в нее узлов и соединений между ними.

3) *Вычислительные ресурсы сетевого оборудования*: в некоторых случаях для обработки пакета требуется участие центрального процессора сетевого устройства (slow forwarding), поэтому необходимо обеспечить гарантированную долю процессорного времени для каждого среза.

4) *Таблицы коммутации*: необходимо изолировать правила таблицы коммутации сетевых устройств, относящиеся к управлению одним срезом, от правил других срезов. Срез сети не должен иметь возможности вносить изменения в правила других срезов.

С точки зрения архитектуры ПКС, FlowVisor является прозрачным прокси-сервером между коммутаторами и контроллером ПКС. К одному FlowVisor может быть подключено несколько контроллеров, реализующих различную логику управления, каждый из которых управляет своим срезом сети. FlowVisor определяет, какие множества потоков относятся к той или иной сети и, следовательно, могут управляться соответствующим контроллером, предоставляет каждому контроллеру собственное видение топологии сети и обеспечивает разделение сетевых ресурсов. С точки зрения каждого контроллера, он монопольно управляет всей сетью и ничего не знает о существовании других срезов.

FlowVisor предоставляет возможность однозначно ассоциировать некоторый класс трафика (т. е. некоторое множество потоков) с одним или несколькими срезами. В качестве класса трафика может выступать, например, весь трафик между определенными конечными узлами или весь http-трафик.

7.4.4 Сценарий управления срезом

Общий сценарий управления срезом через FlowVisor следующий:

— Передача сообщений от контроллера коммутаторам. При прохождении сообщения через FlowVisor проверяется, входят ли коммутаторы, которым адресовано сообщение, в срез данного контроллера. Сообщение пересылается

только коммутаторам из соответствующего среза. Если сообщение содержит инструкции по изменению правил на коммутаторе, оформленные в виде некоторого множества полей заголовков, то FlowVisor выполняет следующее:

1) Проверяется, может ли контроллер устанавливать правила такого типа и вносить изменения в данные поля заголовков.
2) В правило добавляются поля заголовков, определяющие данный срез. Это делается для того, чтобы на коммутаторе это правило применялось только к трафику данного среза.
3) Действия, содержащиеся в правиле, заменяются таким образом, чтобы затрагивать только те порты коммутатора, которые входят в данный срез.
4) После этого измененное сообщение перенаправляется в соответствующие коммутаторы.

– Передача сообщения от коммутатора контроллеру. При прохождении сообщения через FlowVisor оно может быть переслано только тем контроллерам, в чей срез входит коммутатор, отправивший сообщение.

1) Если сообщение содержит информацию об изменении состояния некоторых портов, то каждый контроллер получит информацию только о тех портах, которые входят в его срез.
2) Если сообщение содержит инкапсулированный пакет (сообщение типа PacketIn), то FlowVisor сопоставляет заголовок пакета с шаблонами, описывающими множество потоков каждого среза, и перенаправляет сообщение только тем контроллерам, которые могут управлять данным классом трафика.

Таким образом, взаимодействие полностью прозрачно для обоих сторон: для коммутаторов FlowVisor представляет собой единственный контроллер в сети, а со стороны контроллера передача сообщения через FlowVisor выглядит точно так же, как передача сообщения напрямую коммутатору.

Разделение сети на срезы может иметь иерархическую структуру с использованием нескольких FlowVisor. На Рисунке 7.5 показана схема разделения сети на срезы, в которой FlowVisor 1 выделяет в сети три среза, в одном из которых, в свою очередь, FlowVisor 2 выделяет еще два среза.

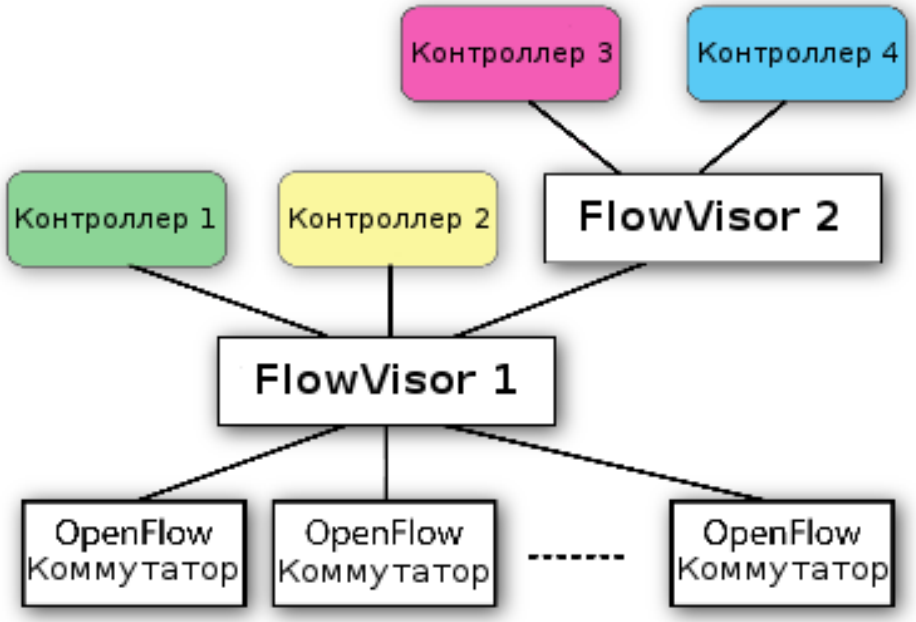

Рисунок 7.5 – Иерархическое разделение сети на сегменты при помощи FlowVisor

7.4.5 Разделение сетевых ресурсов

Разделение сетевых ресурсов реализовано в FlowVisor следующим образом:

— *Разделение пропускной способности*. Для разделения пропускной способности используется механизм очередей, предусмотренный протоколом OpenFlow. С каждым портом коммутатора может быть ассоциировано несколько выходных очередей пакетов. Если контроллер отправляет коммутатору сообщение «перенаправить пакет на порт X», то FlowVizor заменяет его на сообщение «перенаправить пакет на порт X, очередь Y», где Y — очередь, ассоциированная с данным срезом сети. Механизм выборки пакетов из очередей настраивается таким образом, чтобы каждый срез получал не менее запрошенной доли пропускной способности канала.

— *Разделение топологии*. При установлении нового TCP соединения с коммутатором FlowVisor создает соответствующее соединение только для тех контроллеров, в чьи срезы входит данный коммутатор. При передаче сообщений с информацией о состоянии портов и конфигурации коммутатора каждый контроллер получает оповещение только о коммутаторах и портах своего среза. Если контроллер желает разослать LLDP-пакеты для сбора информации о топологии, то FlowVizor помечает их меткой соответствующего среза, а при получении ответных LLDP-пакетов сравнивает их метки с меткой среза.

— Разделение вычислительных ресурсов процессора. Можно выделить три ситуации, в которых требуется участие центрального процессора коммутатора: установление нового потока, обработка сообщения от контроллера и непосредственно обработка пакета, требующая использования ЦПУ (slow-path forwarding). Во всех трех ситуациях FlowVisor использует различные механизмы обеспечения справедливого разделения ресурсов процессора:

1) Установление нового потока. При получении пакета, заголовку которого не соответствует ни одно правило, коммутатор отправляет контроллеру сообщение PacketIn. FlowVisor следит за частотой сообщений PacketIn для каждого среза и, если она превышает некоторый предел, устанавливает на коммутаторе правило, согласно которому требуется сбрасывать все пакеты данного среза в течение короткого промежутка времени.

2) Обработка сообщений от контроллера. FlowVisor следит за тем, чтобы частота сообщений от контроллера не превышала некоторого установленного предела. В случае превышения этого предела FlowVisor на некоторое время прекращает передавать сообщения от этого контроллера.

3) Slow-path forwarding. FlowVisor не позволяет контроллеру устанавливать на коммутаторе правила, согласно которым требуется обработка пакета с участием ЦПУ. Вместо установки такого правила

FlowVisor отправляет аналогичное сообщение PacketOut, предполагающее однократное выполнение данного действия без добавления новых правил в таблицу. Следовательно, разделение ресурсов обеспечивается использованием первых двух механизмов. Следует заметить, что коммутатору также требуются ресурсы ЦПУ для поддержания своего внутреннего состояния (например, для обработки счетчиков и статистики). Поэтому FlowVisor следит за тем, чтобы установленные максимальные частоты для передачи сообщений между контроллером и коммутатором гарантировали, что коммутатор получит необходимое процессорное время для своих внутренних нужд.

— *Разделение таблицы правил коммутатора*. FlowVisor устанавливает для каждого среза максимальное количество записей в таблице правил коммутатора и следит за тем, чтобы в любой момент времени количество правил среза не превышало установленный для него лимит. Если контроллер пытается установить новое правило в тот момент, когда для данного среза уже присутствует максимально допустимое число правил в таблице, FlowVisor возвращает ему ошибку с сообщением, что таблица правил полностью заполнена. Также, как было отмечено выше, множества заголовков правил разных потоков не пересекаются, поэтому контроллер не может изменить правило, установленное контроллером другого среза.

Таким образом, FlowVisor позволяет создавать виртуальные сети, разделяющие как уровень передачи данных, так и уровень управления. Каждый срез сети управляется своим контроллером, реализующим произвольную логику управления сетью. Это может быть полезно, например, для проведения экспериментов или при внедрении новых технологий в сети: для проведения эксперимента выделяется отдельный срез, поэтому данный эксперимент никак не влияет на работу остальной сети.

Что использование FlowVisor на сегодняшний день не является заменой описанных выше технологий виртуализации сети (VXLAN, NVGRE или STT). FlowVisor нацелен, в большей степени, на разделение сети с точки зрения ПКС

контроллера, чем на создание логических сетей с точки зрения конечных узлов. Использование пространства заголовков пакетов согласно протоколу OpenFlow для разделения срезов не позволяет, например, использовать одинаковую адресацию в различных логических сетях.

7.5 Nicira Network Virtualization Platform

Nicira Network Virtualization Platform (NVP) [45] является примером комплексной платформы для виртуализации сетей (преимущественно для ЦОД).

NVP предоставляет средства для создания логических сетей поверх существующих физических сетей с внесением минимальных изменений в структуру сети. Также в NVP входят средства для управления логическими сетями и набор сетевых сервисов, которые могут использоваться в логических сетях (например, балансировщик нагрузки, firewall). NVP имеет программный интерфейс для реализации собственных сетевых приложений и поддерживает интеграцию с существующими платформами управления ЦОД (например, OpenStack) и гипервизорами (Xen, Xen Server KVM, VMware, HyperV, Debian).

NVP основана на архитектуре Distributed Virtual Network Infrastructure (DVNI), общая схема которой представлена на Рисунке 7.6.

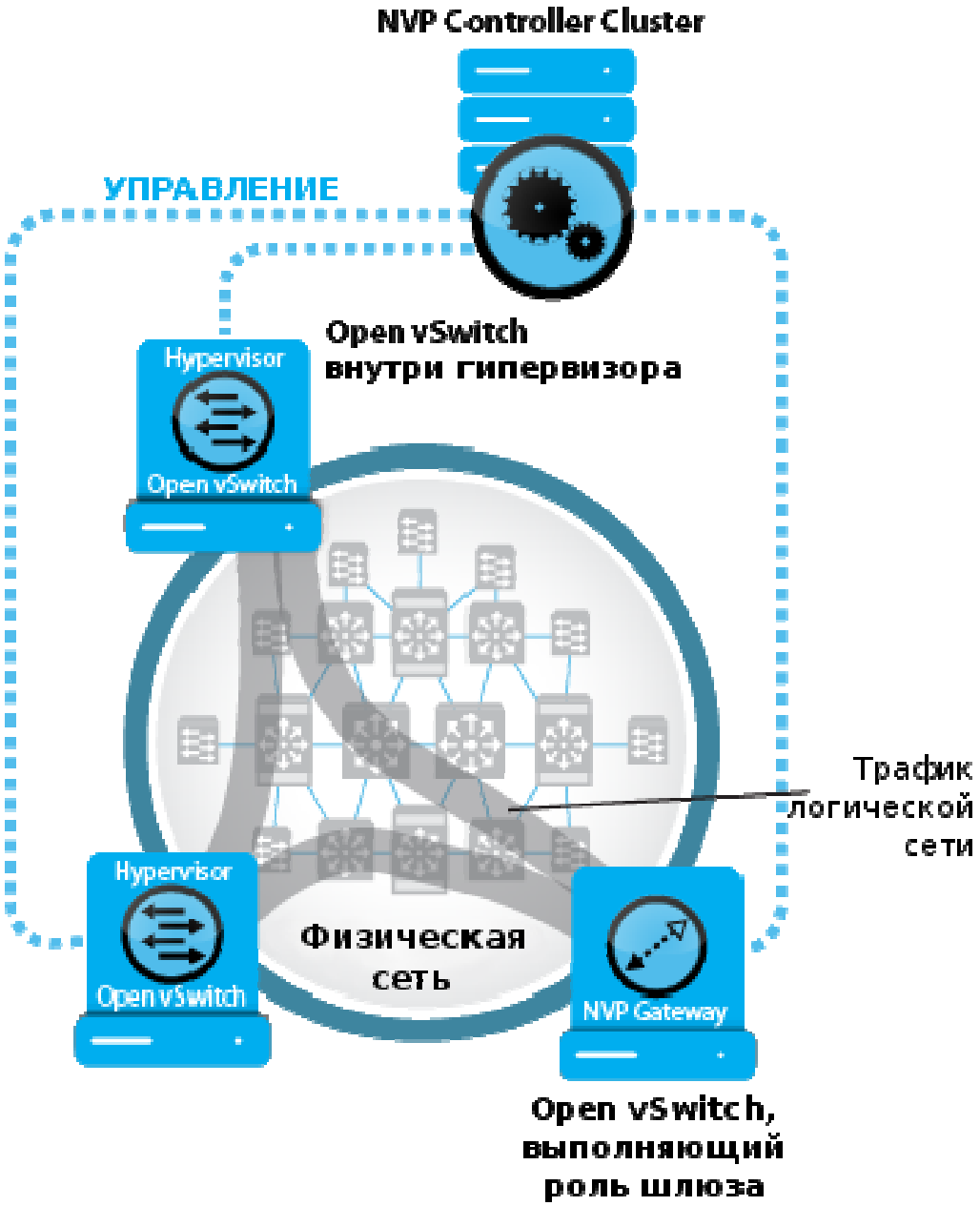

Рисунок 7.6 – Архитектура DVNI Nicira Network Virtualization Platform

Архитектура DVNI предполагает создание логических сетей на основе технологии STT и размещение конечных точек туннеля в виртуальных коммутаторах, которые также служат для связи виртуальных машин внутри одного сервера. Управление сетью логически централизовано и организовано на основе ПКС.

Основным элементом NVP является виртуальный коммутатор Open vSwitch (OVS). OVS располагается внутри гипервизора, между физической сетью и виртуальными машинами.

Возможны два режима использования OVS: когда каждый работающий на отдельном сервере виртуальный коммутатор представлен как отдельная сущность с точки зрения управляющего кластера и виртуальных машин, или использование распределенного коммутатора (Distributed vSwitch, vDS). В случае использования vDS все OVS выглядят как единый коммутатор со стороны виртуальных машин. В случае миграции виртуальной машины между серверами она остается подсоединенной к тому же порту vDS, следовательно, новые сетевые настройки будут заданы автоматически. Также OVS поддерживает объединение портов в группы, которым могут быть заданы единые настройки, что может быть использовано для объединения виртуальных машин, входящих в одну логическую сеть.

Основная функция Open vSwitch — обеспечение прозрачного взаимодействия между виртуальными машинами одной логической сети. Для развертывания логических сетей используется описанная выше технология STT. Выбор этой технологии обусловлен тем, что STT оптимизирована для работы в первую очередь именно для использования с виртуальными коммутаторами и позволяет применять аппаратное ускорение для обработки пакетов на сетевых картах.

Другая функция OVS в NVP — шлюз между внутренней сетью, реализованной с использованием технологии STT, и внешней сетью, например, Интернет.

Таким образом, OVS является конечной точкой туннелей STT. Использование технологии виртуализации сетей, основанной на туннелировании, позволяет полностью изолировать трафик различных логических сетей, разделить их адресные пространства и предоставить всем узлам логической сети свое видение сети, сделав взаимодействие между ними полностью прозрачным. Виртуальные машины будут иметь единое представление своей логической сети независимо от их физического

расположения, таким образом, миграция виртуальных машин между серверами также является прозрачной для пользователя.

Также NVP является примером использования концепции ПКС для управления виртуальными сетями. Так как STT не специфицирует никакую конкретную управляющую логику, для управления конечными точками туннелей, OVS, используются протоколы OpenFlow и OVSDB.

Основным преимуществом применения ПКС для управления виртуальными сетями ЦОД является логическая централизация управления в контроллере (в NVP используется распределенный контроллер). Централизация управления позволяет автоматизировать настройку сетевого оборудования (OVS), например, автоматически назначать сетевые настройки при появлении новых узлов в сети или при миграции существующих узлов.

Модуль NVP, отвечающий за управление виртуальными сетями и содержащий в себе контроллер ПКС — NVP Contoller Cluster. NVP Controller Cluster предоставляет различные сетевые сервисы, которые могут быть использованы в логических сетях: QoS, балансировщик нагрузки, мониторинг трафика, firewall и др., а также предоставляет программный интерфейс (веб-интерфейс RESTful API) для реализации пользовательских сетевых приложений и интеграции с системами управления ЦОД.

Итак, основные технологии, используемые в NVP для управления виртуальными сетями:

— *Технология STT* для развертывания логических сетей поверх существующей физической сети.

— *Open vSwitch*, реализующий функции конечной точки туннеля STT и шлюза между виртуальными сетями на основе STT и внешней сетью и управляемый посредством протоколов OpenFlow и OVSDB.

— *NVP Controller Cluster* — модуль в котором логически централизовано управление всеми виртуальными сетями, согласно концепции ПКС. Модуль

обеспечивает управление OVS при помощи протоколов OpenFlow и OVSDB, а также предоставляет интерфейс для использования совместно с сетевыми сервисами и платформами управления ЦОД.

Таким образом, за счет использования виртуального коммутатора внутри гипервизора в качестве конечной точки туннеля, NVP позволяет развертывать логические сети с внесением минимальных изменений в существующую структуру физической сети. Использование технологии туннелирования в STT позволяет существующему физическому оборудованию передавать трафик виртуальных сетей как обычный трафик в физической сети. А использование централизованного управления упрощает контроль за большим количеством постоянно изменяющихся сетей.

7.6 Выводы

Таким образом, развитие ЦОД и облачных сервисов требует разработки новых подходов к виртуализации сетевой инфраструктуры, так как существующие технологии виртуализации сетей не всегда удовлетворяют поставленным требованиям. Это привело к появлению таких технологий создания логических сетей поверх физических сетей, как VXLAN, NVGRE и STT. Также наблюдается тенденция к созданию комплексных решений для виртуализации сетей, таких как Nicira NVP, и их интеграции с системами управления ЦОД. Отдельно стоит отметить перспективы использования ПКС для виртуализации сетей, во-первых, для автоматизации и централизации управления сетевой инфраструктурой, а во-вторых, для обеспечения разделения логическими сетями не только уровня передачи данных, но и уровня управления.

8 ИССЛЕДОВАНИЕ И ОБОСНОВАНИЕ МЕТОДОВ ОБЕСПЕЧЕНИЯ QOS ДЛЯ ПКС

8.1 Понятие о качестве сервиса

8.1.1 Требования качества сервиса

В системотехнике и программной инженерии требования к свойствам разрабатываемой системы принято разделять на *функциональные* и *нефункциональные* [46]. Первые рассматривают систему как абстрактный автомат, описывая зависимость между исходными данными, поданными ему на вход, и результатами, полученными от него на выходе. Они определяют, что должна делать система, какие *функции* она должна выполнять [47]. Для некоторых требований, таких как удобство использования, детальность документации и конечная стоимость системы, автоматная модель неудобна. Их принято называть нефункциональными. Часть нефункциональных требований, описывающих как именно система должна работать с точки зрения взаимодействующих с ней агентов, обычно называют *качественными* требованиями или требованиями *качества сервиса* (Quality of Service, QoS). Данный класс требований описывает такие свойства системы как её отзывчивость, надёжность, живучесть, безопасность и энергетическая эффективность [48].

Приведённая классификация требований применяется и в области компьютерных сетей. В качестве описываемой системы здесь выступает сетевая инфраструктура, а в качестве агентов – её пользователи, администраторы и владельцы. Их требования качества сервиса, в свою очередь, принято разделять ни *пользовательские*, *административные* и *инфраструктурные* [49].

Первая категория соответствует требованиям приложений к качеству предоставляемых им end-to-end соединений, например, их пропускной способности или времени передачи пакета [50]. При этом каждое приложение может предъявлять свой собственный набор требований качества сервиса независимо от других приложений. Для видеотрансляций наиболее важным требованием является

пропускная способность: отставание допустимо, но дрожание картинки неприемлемо. Для биржевых роботов первостепенную роль играет время передачи данных. Обычно такие приложения оперируют небольшими объёмами сетевого трафика, но должны быстро реагировать на изменения котировок. Для интерактивных телеконференций существенны оба перечисленных параметра.

Административные требования определяют правила использования сетевой инфраструктуры для пользовательских приложений [49]. Обычно это *требования безопасности*, выраженные в виде набора запрещённых и, наоборот, обязательных к выполнению сценариев поведения. Например, корпоративная политика безопасности может запрещать доступ к почтовым серверам из внешней сети и предписывать передавать трафик между офисами компании только через зашифрованные туннели. К административным требованиям также можно отнести изоляцию трафика различных пользователей ЦОД и эксклюзивность их доступа к той части инфраструктуры, которую они арендовали.

Инфраструктурные требования качества затрагивают вопросы эффективности организации передачи данных с точки зрения сети в целом. Критериями эффективности могут служить сразу несколько различных, зачастую противоречащих друг другу, оценочных суждений. Естественным, например, представляется желание уменьшить энергопотребление сети. В случае небольшой загруженности сети, часть её элементов сети может быть обесточена, а проходящие через эти элементы маршруты – направлены обходными путями [51].

Другим примером инфраструктурного требования качества является устойчивость к скачкам нагрузки. Если в процессе построения маршрутов усреднить утилизацию сетевых устройств и каналов передачи данных, то вероятность перегрузки отдельного сегмента сети из-за резкого всплеска активности меньше, чем при обычной маршрутизации. К сходной стратегии выбора маршрутов может привести требование повышения отказоустойчивости сети. Чем меньше потоков передаётся через один элемент сети, тем меньше приложений узнают о его отказе [52].

Как показывают приведённые примеры, инфраструктурные требования могут существенно влиять на выбор маршрутов передачи данных. Однако в отличие от других категорий, требования данного вида редко используются в качестве ограничения для множества допустимых маршрутов. Например, если для обработки трафика приложения с заданным качеством сервиса требуется задействовать обесточенный канал передачи данных, то он будет включен несмотря на общее увеличение энергопотребления. Вместо этого инфраструктурные требования как правило служат критериями оптимизации, на основании которых выбирается наиболее подходящий маршрут передачи данных. На практике такая задача обычно сводится к минимизации функционала, зависящего от абсолютного значения и распределения нагрузки по отдельным элементам сети.

8.1.2 Метрики качества сервиса

Сервис обмена данными, которым сеть обеспечивает функционирующие в ней приложения, можно оценивать в терминах различного рода *метрик* качества. Зачастую метрики соответствуют характеристикам производительности отдельных элементов сети. Например, время передачи пакета данных через предоставленное соединение может быть вычислено как сумма времён передачи пакета через каждый элемент соответствующего этому соединению маршрута. Таким образом, расчёт метрики для заданного соединения сводится к вычислению *функции композиции* для характеристик от нескольких элементов сети.

Для каждой метрики, вообще говоря, может применяться собственная функция композиции. Например, надёжность предоставленного соединения может быть вычислена как произведение вероятностей возникновения ошибки на каждом из элементом используемого им маршрута передачи данных, в то время как его задержка рассчитывается в виде суммы задержек отдельных элементов этого маршрута. Однако на практике обычно применяются лишь три типа метрик: *минимаксные*, *аддитивные* и *мультипликативные*. Для первых композиция определяется как минимум (или максимум) среди значений операндов. Аддитивные

метрики используют в качестве композиции сложение, мультипликативные – умножение [53].

Помимо метрик, соответствующих характеристикам элементов сети, иногда используются и более сложные *производные метрики*, которые вычисляются на основе базовых. В частности, прослеживается тенденция к заданию пользовательских требований в терминах *качества восприятия* (Quality of Experience), что позволяет конечным потребителям охарактеризовать сервис в наиболее естественных для них метриках. Например, пользователю услуги IPTV может быть проще пожаловаться на длительность переключения каналов телевидения, чем заметить, что предоставленное ему соединение имеет слишком высокую задержку.

8.1.3 Связь между обеспечением качества и управлением ресурсами

Один из базовых принципов организации компьютерных сетей – использование единой инфраструктуры для одновременного обслуживания сразу нескольких независимых приложений. При этом если их трафик проходит через одни и те же элементы сети, то, в силу их конечной производительности приложения вынуждены конкурировать за право использовать *ресурсы* этих элементов. Традиционно под критическими *сетевыми ресурсами* подразумевают пропускную способность каналов передачи данных, процессорное время и объём буферов коммутационных устройств. Однако список может расширяться. Например, в случае программно-конфигурируемых сетей, построенных на основе протокола OpenFlow, к списку критических ресурсов естественным образом добавляется также процессорное время контроллера, и размеры таблиц правил OpenFlow коммутаторов.

Качество предоставляемого конкретному приложению сервиса определяется количеством выделенных на его обслуживание ресурсов сети: чем больше ресурсов – тем выше качество. Задача построения end-to-end соединения с заданными метриками качества фактически эквивалентна задаче надлежащего распределения

ресурсов сети между работающими в ней приложениями. При этом перераспределять ресурсы можно не только внутри отдельных элементов сети, но между разными элементами. Например, перенаправление потока данных эквивалентно освобождению ресурсов прежнего маршрута, и захвату ресурсов нового маршрута.

Замена исходной задачи поиска соединения с заданными характеристиками качества сервиса эквивалентной задачей распределения ресурсов порождает фундаментальную проблему соответствующего преобразования требований. Например, непонятно, каким образом можно преобразовать время передачи пакета данных в процессорное время, которое должен уделить трафику приложения конкретный коммутатор. При этом проблема только усугубляется, если использовать более удобные для пользователя метрики качества восприятия.

Универсальные механизмы обеспечения заданного качества сервиса могут выполнять требования приложений лишь за счёт управления инфраструктурой сети. Поэтому они так или иначе должны решать задачу распределения ресурсов, и, следовательно, предоставлять способы преобразования требований приложений в инструкции по перераспределению ресурсов. На практике обозначенная проблема обычно решается с помощью последовательного пересчёта метрик качества сервиса для заданного соединения и последующей корректировки распределения ресурсов на основе полученных замеров.

8.2 Методы обеспечения качества сервиса

8.2.1 Адаптация приложения

Решение проблемы обеспечения заданного уровня качества сервиса в компьютерной сети может быть достигнуто различными способами и на разных этапах взаимодействия между сетью и функционирующими в ней приложениями. Наименее требовательным к сетевой инфраструктуре способом реализации соответствующего средства является *адаптация приложения*. Данный метод

обеспечения качества рассматривает возможности приложения для приспособления к тому уровню качества, который реально предоставляет ему сеть.

Адаптация позволяет подогнать требования качества приложения под требования качества восприятия, которыми руководствуется пользователь этого приложения. Например, единственным реальным требованием пользователя может быть возможность прослушивать радиопередачу в реальном времени. Тогда на ухудшение предоставленного сервиса целесообразно реагировать, соответствующим снижением качества звукопередачи, а не разрывом соединения и нарушением исходного пользовательского требования.

Достоинством описанного подхода является его полная инфраструктурная независимость. Если приложение умеет адаптироваться к предоставленному качеству, то оно может сделать это в любой сети, вне зависимости от принципов построения и характеристик её производительности. Все изменения происходят непосредственно внутри приложения. Однако данный метод не обладает свойством универсальности. Адаптации приложения неизбежно требует вмешательства в его логику и нетривиального изменения кода, которое невозможно проделать в автоматическом режиме. Кроме того, возможности для адаптации сильно ограничены и существенно меняются от приложения к приложению. Адаптация принципиально неприменима, например, к приложениям для передачи через сеть двоичных файлов.

8.2.2 Приоритезация трафика

Более универсальным подходом к обеспечению качества является *приоритезация трафика*. В отличие от адаптации, данный механизм зависит от инфраструктуры сети и работает в терминах управления ресурсами её коммутаторов. Приоритезация разделяет весь передаваемых трафик на несколько классов, сопоставляя каждому из них собственный *приоритет*. Далее ресурсы каждого коммутатора перераспределяются так, чтобы траффик с большим приоритетом получал большее количество его ресурсов. Маршруты передачи данных при этом не перестраиваются.

Примером использования приоритезации может служить перераспределение ресурсов сети между видеотрансляцией и резервным копированием файлов. Пусть эти приложения конкурируют за пропускную способность одного из каналов передачи. Тогда качество видео можно улучшить, повысив приоритет её собственного трафика или же понизив приоритет трафика приложения резервного копирования.

На практике желаемое перераспределение ресурсов производится на коммутаторах за счёт планирования очерёдности обработки пакетов из разных потоков. В случае, если пакеты одного потока обрабатываются чаще, то он получает большее количество сетевых ресурсов и, как следствие, лучшее качество сервиса. Существует несколько различных подходов к буферизации пакетов внутри коммутатора, а так же множество алгоритмов *активного управления очередью* (Active Queue Management), осуществляющих выбор пакетов для обработки из его буферов. Эффективность работы таких алгоритмов является одним из центральных факторов, определяющих параметры качества сервиса проходящих через сеть соединений.

Приоритезация трафика требует от коммутаторов сети способности перераспределение ресурсы в зависимости от приоритетности трафика. Кроме того, трафик приложений должен помечаться соответствующими значениями приоритета при попадании пакетов в сеть. Такова минимальная цена за независимость механизма управления качеством от логики конкретных приложений.

8.2.3 QoS-маршрутизация

Существенным ограничением приоритезации является *локальность* его действия. Данный метод позволяет управлять ресурсами каждого конкретного коммутатора независимо друг от друга, однако он не предоставляет механизмов для согласованного управления сразу несколькими коммутаторами. Например, если один из коммутаторов сети перегружен, но при этом существует альтернативный пути передачи, не проходящий через данный коммутатор, то одна лишь приоритезация не позволит перераспределить ресурсы сети так, чтобы часть потоков

данных воспользовалась обходными путями. Более мощным механизмом, обладающим необходимым глобальным видением сети и предоставляющим возможности для подобного перераспределения ресурсов, является *QoS-маршрутизация*.

Традиционные протоколы маршрутизации, такие как RIP, OSPF и EIGRP, также обладают механизмами для динамического детектирования и устранения заторов с помощью изменения части маршрутов и балансировки трафика по нескольким альтернативным путям (multipath routing). Однако, при балансировке они не учитывают требования качества сервиса отдельных потоков трафика. В результате, если затор образован несколькими нетребовательными приложениями, и одним приложением с жёсткими ограничениями качества, вполне вероятно, что часть пакетов последнего будет направлена альтернативным маршрутом с худшим качеством.

QoS-маршрутизация, напротив, осуществляет маршрутизацию каждого потока исходя из его требований качества и независимо от трафика других приложений. Маршрутизация на уровне потоков позволяет, например, направлять трафик приложений, запущенных на одних и тех же узлах сети по разным маршрутам и даёт средство для более детального контроля за распределением доступных сетевых ресурсов. Как и в случае с приоритезацией, увеличение точности контроля за распределением ресурсов требует усложнения коммутаторов. Реализация маршрутизации на уровне потоков предполагает, что коммутаторы способны запоминать правила маршрутизации для каждого из проходящих через него потоков и сопоставлять с ними поступающие пакеты. Описанную функциональность тяжело эффективно реализовать в сетях с традиционной архитектурой, и это всегда препятствовало широкому применению QoS-маршрутизации на практике. Маршрутизация потоков, однако, является одним из базовых принципов программно-конфигурируемых сетей на основе протокола OpenFlow, что делает эту платформу очень перспективной для развития данного метода обеспечения качества.

8.2.4 Резервирование ресурсов

Изменение нагрузки на отдельные элементы сети, вызванное непредсказуемым поведением пользователей, приводит к постоянному перераспределению ресурсов сети, и, как следствие, постоянному изменению метрик качества для существующих в сети соединений. В то же время многие приложения неспособны работать с использованием соединений, не соответствующих их требованиям качества. Например, соединение со слишком большим временем передачи пакета может сделать отзывчивость игры неприемлемо низкой. Для приложений данного типа нужны соединения с *гарантиями качества*. В тоже время ни один из рассмотренных выше способов обеспечения качества сервиса такие гарантии не предоставляет. Ни адаптация приложения, ни QoS-маршрутизация не приспособлены для сохранения качества соединения при перегрузке сети. Приоритезация позволяет изменить распределение ресурсов в условиях перегрузки, но так же не позволяет решить данную проблему. В самом деле, даже если назначить максимальный приоритет единственному потоку, а всем остальным потокам в сети назначить самые низкие приоритеты, то достаточно большое количество низкоприоритетных потоков всегда может вызвать нарушение требований качества высокоприоритетного потока.

Единственным механизмом управления качеством, способным дать приложению гарантии относительно построенного соединения, является *резервирование ресурсов*. При использовании резервирования для потока заранее прокладывается маршрут, часть ресурсов вдоль которого отдаются этому потоку в эксклюзивное пользование. Зарезервированные таким образом ресурсы не будут использоваться для обработки других потоков. Тем самым, приложение получает гарантию на сервис заданного качества.

Существует несколько протоколов резервирования.

Резервирование ресурсов, так же как и QoS-маршрутизация, работает на уровне отдельных потоков данных и предъявляет аналогичные требования к коммутаторам сети. Более того, для резервирования дополнительно требуется

возможность явного выделения ресурсов коммутатора, что ещё больше затрудняет его реализацию в традиционных компьютерных сетях. Тем не менее, возможность получения гарантий качества подтолкнуло сообщество на разработку нескольких протоколов резервирования. Наиболее известный из них, протокол RSVP был разработан ещё в начале 1990х годов [54]. Однако, ни он, ни его последователи, протоколы RSVP2 и NSLP [55], не получили широкого распространения главным образом в силу чрезмерно высокой нагрузки, которой они подвергали коммутационные устройства сети. Важно отметить, что протоколы резервирования обычно не занимаются самостоятельным прокладыванием маршрутов передачи данных, а полагаются на маршруты, построенные другими протоколами, поэтому им, вообще говоря, не свойственны возможности маршрутизации с учётом потоков и, тем более, требований качества сервиса.

8.2.5 Комплексная модель

Из приведенного выше описания существующих механизмов обеспечения качества видно, что все они имеют недостатки:

— Возможности адаптации ограничены, к тому же данный подход требует вмешательства в логику работы сетевого приложения.

— Приоритезация трафика позволяет лишь локальное управление ресурсами коммутатора, чего может быть недостаточно для действительно эффективного распределения ресурсов между сетевыми приложениями.

— QoS-маршрутизация использует недостающее приоритезации глобальное видение сети. Однако, ни приоритезация, ни QoS-маршрутизация не дают гарантий относительно построенных соединений.

— Резервирование, напротив, предоставляет такие гарантии, но, вообще говоря, не позволяет так же эффективно управлять ресурсами, как это позволяет QoS-маршрутизация.

Независимое использование описанных методов не способно дать того же результата, что их координированное применение.

Сложившаяся ситуация порождает необходимость в разработке комплексного механизма управления качеством, который позволял бы совместить сильные стороны каждого из описанных выше подходов. Стоит заметить, что два наиболее перспективных метода управления качеством QoS-маршрутизация и резервирование ресурсов предполагают независимую маршрутизацию потоков. Поэтому перспективной представляется идея адаптировать данные методы к сетям ПКС, которые строятся на том же базовом принципе. Более того, контроллер ПКС располагает глобальным видением сети, необходимым QoS-маршрутизации для эффективного построения путей передачи данных. К тому же глобальное видение позволяет контроллеру осуществлять неявное резервирование ресурсов сети, сознательно избегая построения маршрутов через участки сети, которые должны были быть зарезервированы за другими потоками. Данные особенности сетей ПКС также позволяют надеяться на дополнительное повышение эффективности этих методов, которое было недоступным в условиях сетей с традиционной архитектурой.

8.3 Требования к ПКС для реализации управления качеством сервиса

8.3.1 Гранулярный контроль над коммутаторами

Реализация механизмов управления качеством сервиса на уровне сетевой инфраструктуры требует надлежащего контроля за распределением сетевых ресурсов между приложениями, конкурирующими за их использование.

Во-первых, контроллер ПКС должен располагать средствами для сбора статистики о состоянии подконтрольных ему OpenFlow коммутаторов, что необходимо, например, для вычисления метрик качества для построенных маршрутов передачи данных и, возможно, последующей корректировки текущего распределения ресурсов сети.

Во-вторых, контроллер ПКС должен иметь возможность изменения текущего распределения сетевых ресурсов, согласно требованиям индивидуальных потоков

трафика. Данная функциональность позволит совместить между собой сразу несколько механизмов обеспечения качества: приоритезацию, QoS-маршрутизацию и резервирование.

8.3.2 Эффективная маршрутизация

Центральной проблемой, решение которой необходимо для эффективного распределения ресурсов на уровне всей сетевой инфраструктуры, является проблема генерации маршрута соединения с заданными параметрами качества. Обычно данная задача формализуется как задача многокритериальной оптимизации на графе, веса рёбер которого отражают характеристики элементов графа, с заданным набором ограничений на метрики, вычисляемые как композиции характеристик. Описанная задача является NP-трудной уже для случая, когда на маршрут накладывается хотя бы два ограничения. Поэтому исходную задачу обычно решают с помощью сведения многокритериальной задачи оптимизации к однокритериальной с помощью *функции свёртки*. Она переводит набор метрик качества (и соответствующих им характеристик элементов сети) в единственный весовой коэффициент. Например, протокол EIGRP использует в качестве свёртки функцию, зависящую сразу от четырёх параметров. Решением исходной задачи считается маршрут с наименьшей суммой весовых коэффициентов, эффективный поиск которого возможен за полиномиальное время.

Полученное таким образом решение не всегда удовлетворяет требованиям исходной задачи. Известны несколько способов расширения области поиска решения, которые способны увеличить вероятность корректного решения задачи. Например, существуют алгоритмы, способные вычислять сразу несколько кратчайших путей. При увеличении числа путей соответствующим образом увеличивается и вероятность того, что один из них будет корректен с точки зрения исходных требований. Вместе с тем, однако, растёт и вычислительная сложность алгоритма. Каждый из подобных способов имеет свои сильные и слабые стороны, и не ясно, какой из них будет лучше работать на практике.

Описанный подход к вычислению маршрутов с заданными свойствами слишком тяжел для реализации данной функциональности в рамках каждого управляющего приложения. Поэтому перспективным представляется вынести её в отдельный модуль контроллера. При этом вполне вероятно, что при решении различных практических задач разные стратегии поиска оптимальных маршрутов передачи данных будут иметь разную эффективность.

8.3.3 Взаимодействие с приложениями

Современные компьютерные сети обеспечивают приложения интерфейсом для отправки и получения данных от других приложений. Однако подобного взаимодействия часто недостаточно. Приложения всё чаще стремятся получить информацию о ресурсах используемой ими сети, в то время как сеть пытается обеспечить приложения честным распределением ресурсов, исходя из запросов этих приложений. Тем самым, естественной необходимостью становится разработка новых механизмов взаимодействия между сетью и работающим в ней приложением. При этом необходимость прямого взаимодействия приложения и сети не ограничивается вопросами качества сервиса. Разработчики распределённых приложений часто вынуждены реализовывать свои собственные кустарные протоколы для поиска одних компонентов программы другими её компонентами. Например, клиент игрового приложения может искать в локальной сети машину, у которой был бы открыт определённый порт, считая эту машину игровым сервером.

В то же время открытие новых способов для взаимодействия между приложением и сетью порождает новые проблемы безопасности. Раньше программно-конфигурируемые сети чётко разделяли между собой плоскости передачи данных и управления сетью, и злоумышленник был не в состоянии преодолеть этот барьер. Однако создание механизмов взаимодействия между этими плоскостями приведёт к потенциальным уязвимостям и необходимости более обстоятельного изучения соответствующих аспектов построения сетей данного вида.

9 РАЗРАБОТКА МЕТОДОВ И АЛГОРИТМОВ УПРАВЛЕНИЯ СЕТЕВОЙ ИНФРАСТРУКТУРОЙ ПКС

9.1 Алгоритмы построения кратчайших путей

9.1.1 Введение

Ряду приложений управления сетью, работающих на ПКС контроллере, требуется решать задачу прокладки в сети оптимальных маршрутов между конечными узлами. Целесообразно вынести данную функциональность в отдельный сервис контроллера, таким образом, чтобы вся логика построения маршрутов в сети была сосредоточена в одном модуле, который бы предоставлял другим приложениям интерфейс для установления виртуального канала между заданными конечными узлами.

Данный модуль должен на входе принимать от приложения набор узлов, между которыми требуется проложить виртуальные каналы. Используя данные о топологии сети, полученные от соответствующего сервиса, данный модуль вычисляет оптимальные маршруты между заданными узлами и устанавливает на коммутаторах соответствующие правила для коммутации пакетов между этими узлами.

Таким образом, требуется реализовать алгоритм построения кратчайших путей и поддержания актуальной информации о кратчайших путях в сети.

Так как сеть может быть представлена в виде графа, в котором вершины соответствуют узлам сети, а ребра — соединениям между узлами, то для решения поставленной задачи могут быть применены алгоритмы поиска кратчайших путей из теории графов.

В качестве весов ребер могут быть использованы различные характеристики канала, такие как среднее время передачи сообщения по данному каналу, пропускная способности или текущая загруженность канала.

Далее приводятся описания классического алгоритма Дейкстры для поиска кратчайших путей от одной вершины графа до всех остальных вершин и алгоритм

Деметрецу и Итальяно, который используется для динамического обновления информации о кратчайших путях.

9.1.2 Алгоритм Дейкстры

9.1.2.1 Для однократного построения кратчайших путей может быть использован традиционный алгоритм Дейкстры. Также данный алгоритм можно использовать для обновления информации о кратчайших путях путем полного перестроения кратчайших путей, в случае, если веса ребер меняются редко.

Алгоритм Дейкстры позволяет найти все кратчайшие пути от одной вершины графа до всех остальных. Данный алгоритм применим только для графов с ребрами неотрицательного веса.

Пусть дан граф $G=(V,E)$, требуется найти кратчайшие пути от вершины a до всех остальных вершин графа.

9.1.2.2 Обозначения:

- w(i,j) — вес ребра (i,j);
- a — вершина, расстояния от которой ищутся;
- d — массив, который по окончании работы алгоритма содержит длины кратчайших путей из a (d[u] — длина кратчайшего пути из a в u);
- p — массив, который по окончании работы алгоритма содержит кратчайшие пути из a (d[u] — кратчайший путь из a в u).

9.1.2.3 Алгоритм:

d[a]=0

p[a]=a

Для всех u, отличных от a:

d[u]=∞

Пока существуют не посещенные вершины v

Если v — вершина с минимальным d[v]

пометить v как посещенную

Для всех не посещенных вершин u таких, что существует ребро из v в u

Если d[u]>d[v]+w(v,u)

d[u]=d[v]+w(v,u)

p[u]=p[v],u

9.1.2.4 Временная сложность алгоритма:

– при простейшей реализации: $O(n^2)$;

– при реализации с использованием фибоначчиевой кучи для хранения не посещенных вершин: $O(n \log n + m)$, где:

1) n — число вершин в графе,

2) m — число ребер в графе.

9.1.3 Алгоритм Деметрецу и Италиано

9.1.3.1 Если веса ребер в графе обновляются часто, то использование алгоритма Дейкстры для полного пересчета кратчайших путей может быть нецелесообразным. Такая ситуация может возникнуть, если, например, в сети реализован алгоритм маршрутизации по состоянию канала и требуется отслеживать изменения пропускной способности на каналах. Другой пример — маршрутизация с

учетом требований QoS, предъявляемых различными сетевыми приложениями: при появлении нового приложения в сети может потребоваться пересчет маршрутов в связи с перераспределением сетевых ресурсов.

Для решения данной проблемы используются алгоритмы динамического построения кратчайших маршрутов в графе, один из них — алгоритм Деметрецу и Италиано [56], предложенный в 2004 году.

Алгоритм Деметрецу и Италиано осуществляет динамическое обновление кратчайших путей в графе при изменении весов ребер графа. Изначально кратчайшие пути могут быть построены при помощи алгоритма Дейкстры. Данный алгоритм может применяться для ориентированных графов с неотрицательными вещественными весами ребер и с петлями.

9.1.3.2 Введем следующие понятия, на которые опирается алгоритм Деметрецу и Италиано:

– *Однородный путь* — такой путь, что любой его подпуть является кратчайшим в данном графе.

– *Исторически кратчайший путь* – это путь из вершины a в вершину b, который был кратчайшим до некоторой последовательности изменений в графе, при этом данные изменения не затрагивали этот путь. Таким образом, длина данного пути осталась прежней, однако в графе могли возникнуть новые пути из a в b, имеющие меньшую длину.

– *Потенциально однородный путь* — такой путь, что любой его подпуть является исторически кратчайшим.

Алгоритм основан на поддержании динамического набора потенциально однородных путей. Данный набор, в частности, включает в себя однородные пути и кратчайшие пути.

Авторы статьи показывают, что если в графе существует z исторически кратчайших путей между каждой парой вершин, то число путей, которые станут

потенциально однородными после каждого обновления — $O(zn^2)$, где n — число вершин в графе.

Для того, чтобы снизить рост числа потенциально однородных путей в графе, используется стратегия сглаживания последовательности обновлений, позволяющая сократить количество исторических путей. Пусть в графе имеется последовательность обновлений S длины k. Стратегия сглаживания строит последовательность изменений $S1$ несколько большей длины (длины порядка $O(klogk)$), однако данная последовательность добавляет меньше исторически кратчайших путей между каждой парой узлов (порядка $O(logk)$). Из приведенного выше утверждения следует, что число путей, которые станут потенциально однородными в результате полученной последовательности обновлений, будет иметь порядок $O(n^2logk)$.

Обновление весов ребер происходит в две фазы:

– Из поддерживаемого набора удаляются все пути, которые содержат обновленные ребра.

– Параллельно на всех вершинах запускается динамическая модификация алгоритма Дейкстры. На каждом шаге кратчайший путь с минимальной длиной извлекается из очереди и комбинируется с существующими кратчайшими путями. При этом создаются новые потенциально однородные пути.

9.1.3.3 Сложность алгоритма:

– по времени обновления — $O(n^2(logn)^3)$,

– по памяти — $O(nnlogn)$.

9.2 Алгоритмы построения топологии сети

9.2.1 Введение

Модуль построения топологии присутствует в любом существующем на сегодняшний день контроллере ПКС. Данный модуль собирает информацию о

соединениях между коммутаторами, которые находятся под управлением данного контроллера, и формирует на основе этой информации некоторую структуру, отражающую топологию сети. Также данный модуль отвечает за периодическое обновление информации о топологии сети. Приложения, работающие на контроллере, могут запросить у модуля информацию о текущем состоянии сети или подписаться на получение сообщений об изменении топологии. Данный сервис необходим приложениям, реализующим механизмы маршрутизации пакетов в сети, например, на основе информации о топологии могут быть построены кратчайшие пути между конечными узлами.

Алгоритмы построения топологии сети основываются на использовании кадров протокола LLDP для обнаружения соединений между соседними узлами.

9.2.2 Построение топологии в традиционных сетях с использованием протокола LLDP

Link Layer Discovery Protocol (LLDP) [24] — протокол канального уровня, который позволяет сетевым устройствам отправлять соседним устройствам сообщения с информацией о себе, а также принимать такие сообщения от соседей. Каждое устройство, поддерживающее LLDP, отправляет информацию о себе соседям независимо от того, отправляет ли сосед информацию о себе, т. е. механизм запрос/ответ не используется.

LLDP-кадр содержит следующую информацию об устройстве: имя устройства, описание устройства, идентификатор порта (с которого был отправлен кадр), описание порта, возможности устройства и др.

Для кадров LLDP зарезервирован специальный multicast MAC-адрес: 01:23:20:00:00:01. Данный MAC-адрес предполагает, что оборудование, получившее кадр с таким адресом назначения, не будет пересылать его дальше.

При получении LLDP-кадра информация о соседнем устройстве и порте, на котором был получен данный кадр, сохраняется устройством в своей Management Information Base (MIB). Впоследствии информация из MIB может быть запрошена

управляющими узлами при помощи протокола Simple Network Management Protocol (SNMP). Таким образом, используя протокол SNMP, управляющий узел может запросить у каждого узла сети (сетевого оборудования, поддерживающего SNMP и LLDP) информацию о его соединениях с соседними узлами и на основе нее восстановить топологию всей сети.

9.2.3 Алгоритм построения топологии ПКС контроллером

9.2.3.1 В ПКС также используется алгоритм построения топологии на основе LLDP-кадров.

Пусть структура, содержащая информацию о топологии сети, содержит информацию о существующих соединениях в следующем виде: <dpid1,port1>-<dpid2,port2>, где

- dpid1 и dpid2 — идентификаторы коммутаторов;
- port1 и port2 — номера портов на соответствующих коммутаторах.

9.2.3.2 Алгоритм построения топологии:

1. При помощи сообщения FlowMod установить на всех коммутаторах правило, согласно которому все кадры протокола LLDP (MAC-адрес назначения равен 01:23:20:00:00:01, тип протокола в заголовке Ethernet равен 0x88cc) должны быть направлены на порт контроллера.

2. Для каждого порта каждого коммутатора сформировать PacketOut сообщения, содержащие LLDP-кадр следующего вида:

а) Chasis Subtype = Locally assigned, Id = dpid:<dpid>, где <dpid> — идентификатор данного коммутатора;

б) Port Subtype = Port Component, Id = <port_num>, где <port_num> - номер данного порта коммутатора;

в) Time To Live = 120 sec;

г) System Description = dpid:<dpid>

3. Отправить сформированные сообщения на коммутаторы, в качестве действия для каждого кадра указать отправку на указанный в кадре порт.

4. Каждый коммутатор рассылает данные LLDP-кадры соседним коммутаторам. При получении LLDP-кадра коммутатор отправляет его контроллеру в сообщении PacketIn.

5. Контроллер получает PacketIn сообщение с LLDP-кадром от коммутатора с идентификатором dpid1. В заголовке сообщения PacketIn указано, что данный кадр был получен на порте port1. Пусть в этом кадре указан идентификатор коммутатора dpid2 и порт коммутатора port2. Следовательно, контроллер заносит в структуру, содержащую информацию о топологии, информацию о наличии соединения между портом port1 коммутатора dpid1 и портом port2 коммутатора dpid2.

9.2.4 Обнаружение соединений, проходящих через коммутаторы, не управляемые контроллером ПКС

Описанный выше алгоритм предполагает, что все OpenFlow-коммутаторы, управляемые одним контроллером, соединены друг с другом напрямую. Однако на практике может возникнуть ситуация, когда один контроллер управляет несколькими OpenFlow-сегментами, которые соединены между собой сетью, состоящей из коммутаторов, не поддерживающих OpenFlow (соответственно, контроллер не может ими управлять). В данном случае алгоритм сформирует независимые топологии для каждого OpenFlow-сегмента, таким образом, с точки зрения контроллера отдельные сегменты не будут соединены друг с другом. Это не будет соответствовать действительности, так как на самом деле пакеты между OpenFlow-сегментами могут передаваться через обычные коммутаторы.

Для решения этой проблемы предложено дополнение алгоритма для обнаружения соединений между двумя OpenFlow-коммутаторами, соединенными через коммутаторы, не поддерживающие OpenFlow. Вместе с кадрами LLDP рассылаются кадры BDDP, которые отличаются от LLDP тем, что в них указан широковещательный MAC-адрес. Следовательно, при получении такого кадра

обычным коммутатором он будет разослан по всем портам, кроме входящего. Соответственно, при установке правила на коммутаторе требуется указать широковещательный адрес ff:ff:ff:ff:ff:ff и тип протокола 0x8999. В итоге BDDP-кадр будет доставлен во все OpenFlow-сегменты, с которыми имеются соединения. При получении этого кадра первым OpenFlow-коммутатором в каждом сегменте он будет отправлен на контроллер и обработан согласно описанному выше алгоритму, с учетом того, что данное соединение проходит через не управляемые контроллером коммутаторы.

9.3 Алгоритмы передачи сообщений

9.3.1 Особенности передачи сообщений OpenFlow

Передача сообщений OpenFlow от ядра контроллера приложениям, работающим на контроллере, а также механизм взаимодействия между различными приложениями — функциональность, которая должна быть реализована в контроллере в обязательном порядке, так как без нее управление сетью при помощи приложений становится невозможным.

Простейший способ взаимодействия между приложениями и ядром — взаимодействие через «общую шину». В этом случае передача сообщений между приложениями является широковещательной, т. е. все приложения получат сообщение, отправленное ядром или некоторым приложением. Задача фильтрации нужных сообщений возлагается на само приложение. Однако использование такой схемы приведет к тому, что каждое приложение будет получать большое количество сообщений OpenFlow от контроллера и сообщений от других приложений, которые ему не требуются.

Для решения этой проблемы применяются техники событийно-ориентированного программирования. В качестве механизма распространения сообщений реализуется схема «издатель-подписчик».

Другой способ взаимодействия — использование запросов, т. е. когда приложение само запрашивает необходимую ему информацию по мере необходимости.

Рассмотрим, как эти схемы могут быть реализованы для передачи сообщений OpenFlow от ядра приложениям и передачи сообщений между приложениями.

9.3.2 Передача сообщений OpenFlow от ядра приложениям

Приложение может быть заинтересовано в получении OpenFlow сообщений только определенных типов (например, PortStatus или PacketIn) или только от определенных коммутаторов. Также в некоторых ситуациях важен порядок получения сообщений различными приложениями. Например. некоторые приложения могут выполнять фильтрацию сообщений для других приложений. Таким образом, при получении сообщения данное приложение должно принимать решение, должно ли это сообщение быть сброшено или отправлено другим приложениям.

Для реализации данных требований может быть использован следующий механизм:

— В ядре должен быть реализован механизм подписки на сообщения. Для каждого типа сообщений OpenFlow и для различных коммутаторов должен храниться список приложений-подписчиков, которые заинтересованы в получении данных сообщений. При получении сообщения ядро отправляет его только тем приложениям, которые находятся в соответствующих списках подписчиков. Также ядро предоставляет интерфейс для подписки на сообщения.

— Используя интерфейс подписки, приложение может добавить себя в список подписчиков на сообщения. При подписке приложение указывает тип сообщений, которые оно желает получать, а также множество коммутаторов, сообщения от которых следует пересылать данному приложению. Таким образом, приложение получит сообщения только тех типов и только от тех коммутаторов, в которых оно заинтересовано.

— Для задания порядка получения сообщений приложениями список подписчиков организован в виде очереди. По умолчанию каждый новый подписчик добавляется в конец этой очереди. Однако приложение при подписке может указать, каким номером (до или после кого приложения) оно желает получать данные сообщения. При рассылке сообщения оно рассылается всем подписчикам в порядке очереди. Фактически, при этом вызывается обработчик сообщений данного приложения. При передачи управления обратно ядру приложение может явно указать, продолжать ли рассылку данного сообщения следующим в очереди приложениям, или остановить рассылку.

— Списки подписчиков могут изменяться в процессе работы контроллера. Таким образом, должны быть реализованы механизмы как подписки на сообщения, так и отписки.

9.3.3 Взаимодействие между приложениями

9.3.3.1 Для реализации взаимодействия между приложениями контроллера могут параллельно использоваться два механизма: подписка на события (в этом случае все подписчики получают уведомления от рассылающего приложения) и явные запросы (заинтересованное приложение само запрашивает у приложения необходимую ему информацию).

9.3.3.2 Механизм подписки на события (сообщения) – данный способ взаимодействия приложений аналогичен описанному выше механизму передачи сообщений OpenFlow между ядром и приложениями. Каждое приложение может опубликовать набор сервисов, которые оно предоставляет другим приложениям (например, приложение построения топологии может рассылать оповещения об изменении топологии сети). Другие приложения могут подписаться на этот сервис, т. е. добавить себя в список получателей соответствующих уведомлений. Дальнейшее взаимодействие происходит по описанной выше схеме для рассылки OpenFlow сообщений.

9.3.3.3 Механизм запросов. В данном случае приложение также публикует набор предоставляемых сервисов, однако вместо интерфейса для подписки на уведомления оно предоставляет интерфейс для осуществления запросов к приложению. Другие приложения по мере необходимости могут запрашивать информацию у данного приложения, используя интерфейс запросов. Данный механизм применим, если приложение осуществляет не рассылку уведомлений об изменениях, а выполняет некоторую задачу по требованию других приложений (например, прокладывает кратчайший маршрут между заданными узлами) или поддерживает некоторую базу данных (например, ARP-таблицу), к которой могу осуществляться запросы.

9.3.4 Реализация описанных механизмов

На сегодняшний день поддержка событийно-ориентированного программирования (СОП) включена во многие широко используемые языки и библиотеки, в частности имеются:

- Языки с поддержкой СОП: Java, C#, Delphi, Perl.
- Библиотеки с поддержкой СОП: glib, libevent, Gui4Cli, libsigc++, Boost, Qt, Cocoa, Simple Unix Events (SUI), QP, Twisted.
- И другие.

Механизм запросов может быть реализован как средствами языка программирования, на котором реализован контроллер и приложения, так и с помощью стандартных интерфейсов (например, RESTful). Использование стандартных интерфейсов для взаимодействия между приложениями позволяет создавать приложения для контроллеров на различных языках программирования, а также запускать приложения удаленно. Однако данный способ взаимодействия практически неприменим для реализации схемы «издатель-подписчик».

Также отметим, что реализация механизмов взаимодействия между приложениями средствами языка программирования, на котором реализован контроллер предпочтительна в том случае, если требуется высокая скорость

взаимодействия между приложениями, или если между приложениями передается большое количество сообщений, так как это позволяет сократить накладные расходы на передачу сообщений.

9.4 Алгоритмы преобразования сообщений

9.4.1 Особенности преобразования сообщений OpenFlow

Для преобразования сообщений OpenFlow во внутреннее представление и обратно необходимо создать библиотеку на используемом языке программирования (или выбрать уже существующую), в которой согласно спецификации протокола OpenFlow будут описаны типы и структуры данных, соответствующие типам сообщений OpenFlow.

9.4.2 Алгоритм преобразования полученного сообщения OpenFlow во внутреннее представление

Процесс преобразования сообщения OpenFlow во внутреннее представление зависит от используемого языка программирования, однако можно сформулировать следующий общий алгоритм (дальнейшее описание типов сообщений приводится для версии OpenFlow 1.0.0 (wire protocol 0x01)):

1. Пусть все полученные сообщения поступают в общую очередь сообщений и представляют собой единый массив данных. Для выделения отдельного сообщения из данного массива необходимо сначала считать первые 8 байт (размер OpenFlow-заголовка) и привести его к структуре со следующими полями:
   а) version — версия протокола OpenFlow (1 байт)
   б) type — тип сообщения OpenFlow (1 байт)
   в) length — длина всего сообщения, включая заголовок (2 байта)
   г) xid — номер транзакции (используется в сообщениях типа запрос/ответ) (4 байта)

2. Извлечь из очереди массив длиной (length-8) байт.

3. Если версия, указанная в version, не соответствует используемой, то сбросить данное сообщение без дальнейшей обработки.

4. Если версия поддерживается, то привести считанный массив к структуре, соответствующей типу, указанному в type, добавив полученный на шаге 1 заголовок. Возможные типы полученных сообщений и особенности их преобразования:

а) OFPT_PACKET_IN:

```
struct ofp_packet_in {
  struct ofp_header header;
  uint32_t buffer_id;     /* номер буфера */
  uint16_t total_len;     /* длина кадра */
  uint16_t in_port;       /* порт, на к */отором получен кадр */
  uint8_t reason;         /* причина отправки кадра контроллеру*/
  uint8_t pad;
  uint8_t data[0];        /* пересылаемый кадр */
}
```

Поле data — содержимое пакета, переданного в packet_in, размер рассчитывается как (length-18) байт.

б) OFPT_PORT_STATUS

```
struct ofp_port_status {
  struct ofp_header header;
  uint8_t reason;         /* причина отправки сообщения */
  uint8_t pad[7];
  struct ofp_phy_port desc;
```

```
}


struct ofp_phy_port {
   uint16_t port_no;
   uint8_t hw_addr[OFP_ETH_ALEN];
   char name[OFP_MAX_PORT_NAME_LEN];
   uint32_t config;       /* набор флагов */
   uint32_t state;        /* набор флагов */
   uint32_t curr;         /* текущие возможности порта */
   uint32_t advertised;   /* возможности, объявленные портом */
   uint32_t supported;    /* возможности, поддерживаемые портом */
   uint32_t peer;         /* возможности. поддерживаемые смежным портом */
}
```

в) OFPT_FEATURES_REPLY

```
struct ofp_switch_features {
   struct ofp_header header;
   uint64_t datapath_id;   /* идентификатор свича */
   uint32_t n_buffers;     /* количество буферов для пакетов */
   uint8_t n_tables;       /* количество таблиц */
   uint8_t pad[3];
   uint32_t capabilities;  /* поддерживаемые свойства */
   uint32_t actions;       /* поддерживаемые действия */
   struct ofp_phy_port ports[0];  /* описание портов */
}
```

Поле ports содержит массив описаний физических портов коммутатора, размер поля рассчитывается как (length-32).

г) OFPT_GET_CONFIG_REPLY

```
struct ofp_switch_config {
   struct ofp_header header;
   uint16_t flags;          /* обработка фрагментов кадров */
   uint16_t miss_send_len;    /* количество байт кадра,
                                посылаемых в packet_in */
}
```

д) OFPT_QUEUE_GET_CONFIG_REPLY

```
struct ofp_queue_get_config_reply {
   struct ofp_header header;
   uint16_t port;
   uint8_t pad[6];
   struct ofp_packet_queue queues[0]; /* список настроенных
                                         очередей*/
}
```

Поле ports содержит массив описаний настроенных на коммутаторе очередей, размер поля рассчитывается как (length-11).

е) OFPT_STATS_REPLY

```
struct ofp_stats_reply {
   struct ofp_header header;
   uint16_t type;          /* тип передаваемой статистики */
   uint16_t flags;
```

```
  uint8_t body[0];/
}
```

Размер поля body определяется, исходя из типа передаваемой статистики (поле type).

ж) OFPT_FLOW_REMOVED

```
struct ofp_flow_removed {
  struct ofp_header header;
  struct ofp_match match;   /* описание полей правила */
  uint64_t cookie;        /* дополнительая инофрмация о
                          правиле */
  uint16_t priority;      /* приоритет правила */
  uint8_t reason;         /* причина удаления правила */
  uint8_t pad[1];
  uint32_t duration_sec;    /* время действия правила
                            в секундах */
  uint32_t duration_nsec;   /* время дейстрия правила
                            в наносекундах */
  uint16_t idle_timeout;    /* период, после которого правило
                            удаляется, если не было к нему
                            обращений*/
  uint8_t pad2[2];
  uint64_t packet_count;  /* количество переданных в
                            соответствии с правилом пакетов */
  uint64_t byte_count;  /* количество переданных в
                            соответствии с правилом байт */
```

```
}
```

з) OFPT_ERROR_MSG

```
struct ofp_error_msg {
   struct ofp_header header;
   uint16_t type;          /* тип ошибки */
   uint16_t code;          /* код ошибки в соответствии с типом */
   uint8_t data[0];        /* описание ошибки */
}
```

Длина поля data рассчитывается как (length-12) байт и интерпретируется в зависимости от значений type и code

и) OFPT_BARRIER_REPLY, OFPT_HELLO

Содержит только полученный в п. 1 заголовок: все содержимое сверх этого заголовка игнорируется

к) OFPT_ECHO_REQUEST, OFPT_ECHO_REPLY

содержимое сообщение сверх заголовка, полученного в п. 1, может иметь произвольную длину и никак не интерпретируется.

Если длина, указанная в length, превышает длину сообщения соответствующего типа, и данный тип не предусматривает полей переменного размера (как, например, сообщения типа OFPT_PACKET_IN или OFPT_ERROR_MSG), то все лишние данные удаляются из очереди сообщений без интерпретации.

9.4.3 Алгоритм преобразования сообщения OpenFlow для передачи по сети

При отправке сообщения OpenFlow требуется сделать следующее:

1. Заполнить все поля структуры, соответствующей сообщению отправляемого типа.

2. Сформировать заголовок OpenFlow для данного сообщения, где в качестве версии протокола OpenFlow указать используемую версию, в качестве типа — тип пересылаемого сообщения, длину установить равной (длина передаваемого сообщения + 8) байт. Если данный тип сообщения представляет собой запрос коммутатору, на который предполагается получение ответа (OFPT_ECHO_REQUEST, OFPT_FEATURES_REQUEST, OFPT_GET_CONFIG_REQUEST, OFPT_STATS_REQUEST, OFPT_QUEUE_GET_CONFIG_REQUEST), то в поле xid необходимо записать номер, по которому можно будет однозначно идентифицировать ответ на данный запрос. Если сообщение является ответом на запрос коммутатора (OFPT_ECHO_REPLY), то поле xid должно совпадать с таким же полем в запросе (OFPT_ECHO_REQUEST). В остальных случаях значение поля может быть произвольным.

3. Объединить заголовок и сообщение в единую структуру и передать ее в виде байтовой строки.

Список сообщений, которые могут быть отправлены контроллером коммутатору:

– OFPT_ECHO_REQUEST, OFPT_ECHO_REPLY. Сообщения могут содержать произвольные данные после заголовка OpenFlow; OFPT_ECHO_REPLY должен содержать те же данные, что и запрос, на который это сообщение отвечает.

– OFPT_VENDOR

```
struct ofp_vendor_header {
   struct ofp_header header;
   uint32_t vendor;
}
```

– OFPT_SET_CONFIG

```
struct ofp_switch_config {
   struct ofp_header header;
   uint16_t flags;          /* обработка фрагментов кадров */
   uint16_t miss_send_len;    /* количество байт кадра,
                                посылаемых в packet_in */
}
```

– OFPT_PACKET_OUT

```
struct ofp_packet_out {
   struct ofp_header header;
   uint32_t buffer_id;         /* номер буфера */
   uint16_t in_port;          /* порт, на клтором был получен пакет
                              (если это ответ на packet_in) */
   uint16_t actions_len;       /* длина массива действий в байтах */
   struct ofp_action_header actions[0]; /* действия */
   uint8_t data[0];       /* содержимое пакета */
}
```

Поле actions_len содержит длину массива actions.

Поле data может не содержать данных, если сообщение является ответом на packet_in и предполагает отправку буферизованного на коммутаторе пакета.

– OFPT_FLOW_MOD

```
struct ofp_flow_mod {
   struct ofp_header header;
   struct ofp_match match;     /* поля заголовка правила */
```

```
  uint64_t cookie;          /* дополнительная информация
                            о правиле */
  uint16_t command;         /* действие над правилом */
  uint16_t idle_timeout;    /* время, в течение которого
                            не используется правило,
                            после котроого его требуется удалить */
  uint16_t hard_timeout;    /* максимальное время жизни правила */
  uint16_t priority;        /* приоритет правила */
  uint32_t buffer_id;       /* буфер, содержащий пакет, к которому
                            надо применить правило */
  uint16_t out_port;        /* номер выходного порта в праивле
                            в случае удаления праивла */
  uint16_t flags;           /* флаги */
  struct ofp_action_header actions[0]; /* действия */
}
```

– OFPT_PORT_MOD

```
struct ofp_port_mod {
  struct ofp_header header;
  uint16_t port_no;
  uint8_t hw_addr[OFP_ETH_ALEN]; /* MAC-алрес порта */
  uint32_t config;      /* новая конфигурация порта */
  uint32_t mask;        /* маска свойств. которые надо изменить */
  uint32_t advertise;   /* битовая маска "ofp_port_features" */
  uint8_t pad[4];
}
```

– OFPT_STATS_REQUEST

```
struct ofp_stats_request {
   struct ofp_header header;
   uint16_t type;          /* тип статистики */
   uint16_t flags;         /* флаги */
   uint8_t body[0];
}
```

Содержимое body зависит от типа статистики в поле type

– OFPT_QUEUE_GET_CONFIG_REQUEST

```
struct ofp_queue_get_config_request {
   struct ofp_header header;
   uint16_t port;      /* номер порта, по которому запрашивается
                          статистика очередей */
   uint8_t pad[2];/
}
```

– OFPT_HELLO, OFPT_FEATURES_REQUEST, OFPT_GET_CONFIG_REQUEST, OFPT_BARRIER_REQUEST

Данные сообщения не содержат никаких данных помимо заголовка OpenFlow.

10 ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ОТЕЧЕСТВЕННОЙ ПЛАТФОРМЕ УПРАВЛЕНИЯ ПКС

10.1 Цель работы

Цель работы заключается в разработке отечественной распределенной платформы управления программно-конфигурируемыми сетями, которая будет обеспечивать управление сегментами ПКС и потоками трафика в них с показателями производительности, сравнимыми с показателями производительности традиционных (не относящихся к ПКС) сетей при меньших требованиях к ресурсам в сравнении с существующими решениями в области ПКС.

10.2 Требования к процессу разработки платформы управления ПКС

Процесс разработки должен удовлетворять современным требованиям разработки ПО. В основе должны лежать все современные методики, технологии и средства объектно-ориентированного проектирования и разработки ПО [57]. Можно сформулировать следующие основные требования к процессу разработки отечественной платформы управления ПКС:

— Платформа должна иметь модульную архитектуру, позволяющую расширять её функциональность.

— Разработка платформы должны вестись, используя спиральную модель, где на каждом этапе разработке создается работающий прототип, реализующий часть конечной функциональности.

— Разработка платформы должна вестись с использование системы контроля версия, для обеспечения удобного контроля изменений, ревизий и наличия версий проекта. В качестве системы контроля версий должна быть использована система Git [58], как обладающая наибольшей производительностью и функциональностью.

— Текст программы отечественной платформы должен быть оформлен и откомментирован в соответствие с современными требованиями к программам. Код должен иметь комментарии в формате Doxygen, для возможности автоматической

генерации документации.

10.3 Требования к функциональности платформы управления ПКС

В пункте 1.4.4 (Контроллер, сетевая ОС и сетевые приложения) и в разделе 2 (Анализ существующих СОС для ПКС) были даны свойства и сформулированы основные требования к функциональности платформы управления ПКС, на основании чего выдвинуты следующие основные требования к функциональности отечественной платформы управления ПКС:

– Поддержка протокола OpenFlow версии не ниже 1.1.

– Поддержка автоматического построения топологии ПКС.

– Поддержка мониторинга состояния сети.

– Поддержка маршрутизации данных в рамках сегмента сети ПКС (в реактивном и проактивном режимах).

– Поддержка кодирования/декодирования пакетов.

– Поддержка механизмов генерации/подписки на события для модулей и приложений сетевой ОС.

– Поддержка многопоточной обработки запросов от элементов сети.

10.4 Требования к производительности

В работе [59] проведён анализ сетевого трафика в десяти ЦОД, расположенных на территории США. ЦОД средних размеров характеризуется наличием примерно 10000 конечных узлов, 256 коммутаторов и 10 миллионами запросов на установление нового потока в секунду.

Поэтому отечественная платформа управления ПКС должна обладать следующими характеристиками производительности:

– Пропускная способность платформы не менее 10 миллионов запросов в секунду.

— Задержка платформы не более 100-150 миллисекунд.

— Поддержка не менее 256 коммутаторов по 10000 хостов на каждый.

10.5 Требования к составу платформы

В разделе 5 (Проведение анализа традиционных сервисов/приложений) сформулирован требуемый состав платформы управления ПКС, включающий модули ядра платформы, предоставляемые сервисы и минимальный набор приложений.

По составу отечественная платформа управления ПКС должна удовлетворять следующим требованиям:

— Ядро платформы должно включать следующие основные *модули*:

1) Модуль, обеспечивающий взаимодействие с коммутаторами по протоколу OpenFlow заданной версии.

2) Библиотеки для обработки и формирования заголовков пакетов различных уровней сетевого стека.

3) Модуль контроллера, осуществляющий загрузку всех остальных модулей и обеспечивающих их взаимодействие.

4) Модуль, обеспечивающий обмен сообщениями между приложениями контроллера или уведомление приложений о тех или иных событиях.

— Отечественная платформа должна поддерживать следующие *сетевые сервисы*:

1) Сервис для сбора, хранения и обновления информации о топологии сети.

2) Сервис для установления маршрутов между конечными узлами сети.

3) Сервис для создания виртуальных сетей.

4) Сервис для взаимодействия с системами управления сетями ЦОД (OpenStack).

5) Сервис для поддержки RESTfull интерфейса для внешних приложений.

– Отечественная платформа должна иметь следующий минимальный набор *приложений*: приложение для L2 коммутации (правила на коммутаторах устанавливаются на основе MAC адресов).

– Платформа должна поддерживать следующие возможности её настройки и конфигурирования:

1) Коммандная консоль (CLI).
2) Web-интерфейс.
3) Конфигурационный файл.

10.6 Требования к масштабируемости

В ходе анализа структуры существующих СОС (разделы 2 и 5) были определены архитектурные особенности их организации. При проектировании СОС необходимо учитывать возможность взаимодействия с не менее чем 256 коммутаторами. Также выявлено, что без использования многопоточности, не возможно обеспечить требуемую пропускную способность.

Требования к масштабируемости платформы:

– Поддержка работы с не менее 256 коммутаторами.

– Поддержки многопоточности на уровне не менее 8 потоков.

10.7 Требования к составу и параметрам технических средств

В ходе анализа существующих СОС для ПКС (раздел 1 и 2) и анализа методов управления традиционными сетями (раздел 4) выявлено, что необходима поддержка взаимодействия, как с аппаратными коммутаторами OpenFlow, так и с программными, использование которых характерно для ЦОД и сети провайдера услуг. Для обеспечения открытости платформы управления для ПКС должна развертываться на базе ОС Linux.

Отечественная платформа управления должна удовлетворять следующим техническим требованиям:

– Платформа должна развертываться на сервере с базовой ОС Linux.

– Платформа должна поддерживать работу с виртуальным коммутатором OpenvSwitch.

– Платформа должна поддерживать работу с доступными аппаратными OpenFlow коммутаторами.

10.8 Требования к надежности

В ходе анализа существующих СОС (раздел 2) и анализа механизмов управления традиционными сетями (раздел 4) выявлено, что платформа управления ПКС должна быть устойчива к сбоям, как контролируемых коммутаторов, так и функционирующих сетевых приложений.

Требования к надежности платформы:

– Платформа должна обрабатывать сбои в работе приложений.

– Платформа должна обрабатывать сбои в соединениях между узлами платформы.

– Платформа должна обрабатывать сбои в узлах сетевой ОС.

10.9 Требования к экспериментальному исследованию платформы управления

В процессе выбора критериев эффективности управления сетевой инфраструктурой ПКС (раздел 3) и анализа механизмов управления традиционными сетями (раздел 4) получено, что тестовый сегмент сети ПКС должен обеспечивать возможность демонстрации работы платформы управления несколькими коммутаторами, как физическими, так и виртуальными, соединяющих физические и виртуальные машины, и обеспечивать работу новых пользователей, подключаемых к сети через Wi-Fi соединение.

Экспериментальные исследования должны быть проведены на экспериментальном сегменте ПКС, который должен включать в себя:

– коммутаторы с поддержкой протокола OpenFlow:

1) количество коммутаторов не менее двух;
2) количество портов на коммутатор: не менее 48;
3) количество соединений верхнего уровня на коммутатор: не менее 2x10G;
4) поддержка протокола OpenFlow версии стандарта 1.1 или выше;
5) скорость не ниже 1 Гигабит/сек.

– коммутатор Gigabit Ethernet;

– серверы виртуализации контроллеров OpenFlow:

1) количество серверов: не менее четырех;
2) количество ядер ЦПУ на сервер: не менее восьми с частотой не менее 2Ггц;
3) количество ОЗУ на сервер: не менее 48 Гигабайт;
4) ОС серверов: Linux (Ubuntu 12.04 LTS или выше, RHEL 6.2 или выше);

– точку доступа Wi-Fi.

10.10 Требования к программной документации

Требование к программной документации – документация должна содержать:

– описание разработанной платформы управления ПКС (архитектуры, структуры, назначения основных модулей, функциональные возможности);

– описание программного кода.

10.11 Выводы

В данном разделе сформулированы основные требования к отечественной платформе управления ПКС, от требований к функциональному наполнению платформы и требуемым показателям производительности, до требований к процессу разработки и тестирования платформы на экспериментальном сегменте ПКС. Эти требования послужат основой для разработки (на втором этапе НИР) проекта технического задания на ОКР по созданию отечественной платформы управления ПКС.

11 РАЗРАБОТКА ПРОГРАММЫ И МЕТОДИК ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

11.1 Общие положения

Данный раздел посвящён разработке Программы и методик экспериментального исследования сетевых операционных систем (СОС) для программно конфигурируемых сетей (ПКС) и экспериментальной реализации алгоритмов управления ПКС (п. 1.11 Календарного плана). Само экспериментальное исследование должно быть выполнено на этапе 2 НИР. Рассмотрены объект исследований, цель исследований, методики исследований. Подробно описаны применяемые в исследованиях программные средства. В виде отдельного документа выпущены Программа и методики, оформленные по требованиям приложения 24.2 к Порядку приемки выполненных работ (этапов работ) по государственным контрактам, заключенным в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007—2013 годы».

11.2 Цели и задачи экспериментальных исследований

Одним из ключевых факторов, влияющих на производительность программно конфигурируемых сетей (ПКС), является производительность контроллера ПКС. Поэтому именно контроллер является основным *объектом экспериментального исследования*.

Целью экспериментального исследования является определение основных характеристик производительности, масштабируемости, надежности и безопасности существующих контроллеров.

В рамках данной работы экспериментальные исследования будут нацелены на контроллеры с открытым исходным кодом, выбранные для исследования в разделе 2 данной НИР.

11.3 Программа экспериментального исследования

Программа экспериментального исследования состоит из следующих основных этапов:

1) Измерение характеристик производительности контроллера:
2) пропускной способности контроллера (количество запросов от коммутаторов, обрабатываемых контроллером в секунду — потоков/секунда);
3) задержка контроллера (время, затрачиваемое контроллером на обработку одного запроса — миллисекунды).

Измерение характеристик масштабируемости:

1) изменение показателей производительности при увеличении числа ядер процессора;
2) изменение показателей производительности при увеличении числа соединений с коммутаторами;
3) изменение показателей производительности при увеличении числа конечных узлов в сети.

Измерение характеристик ресурсоемкости:

1) загрузка ядер процессора;
2) использование физической памяти.

Проверка функциональных возможностей – поддержка функций OpenFlow.

Исследование следующих характеристик надежности:

1) количество отказов за время тестирования;
2) время безотказной работы при заданном профиле нагрузок.

Оценка безопасности – проверка устойчивости к некорректно сформированным сообщениям протокола OpenFlow.

11.4 Методика экспериментального исследования

11.4.1 Измерение пропускной способности контроллера

11.4.1.1 В ходе тестирования необходимо исследовать зависимость пропускной способности контроллера от следующих параметров:

– Количество подключенных к контроллеру коммутаторов (1, 4, 8, 16, 64, 256) при фиксированном числе хостов с уникальными MAC адресами на один коммутатор (10^5) (тестовый сценарий 1).

– Количество хостов с уникальных MAC адресами на один коммутатор (10^3, 10^4, 10^5, 10^6, 10^7) при фиксированном числе коммутаторов (32) (тестовый сценарий 2).

Также для каждого тестового сценария необходимо исследовать масштабируемость контроллера по ядрам, т. е. зависимость его пропускной способности от количества используемых ядер процессора.

Для каждого теста нужно проводить три запуска Cbench (средство по высоконагрузочному тестированию контроллеров через посылку сгенерированных packet_in сообщений) [60], в качестве результата брать среднее значение по трем запускам.

11.4.1.2 *Тестовый сценарий 1.* Исследование зависимости пропускной способности контроллера от количества подключенных коммутаторов.

Cbench необходимо запустить три раза с имитацией различного количества коммутаторов – 1, 4, 8, 16, 64, 256.

Для каждого коммутатора имитировать 10^5 подключенных конечных узлов.

Каждый контроллер должен быть запущен на 1-12 ядрах процессора.

11.4.1.3 *Тестовый сценарий 2.* Исследование зависимости пропускной способности от количества конечных узлов в сети.

Cbench необходимо запустить три раза с имитацией 32 коммутаторов,

Для каждого коммутатора задавать различное количество подключенных конечных узлов – 10^3, 10^4, 10^5, 10^6, 10^7.

Каждый контроллер должен быть запущен на 1-12 ядрах процессора.

11.4.2 Измерение задержки

11.4.2.1 Величину задержки в контроллере нужно тестировать:

1) с имитацией одного коммутатора;
2) для контроллера, запущенного на одном ядре процессора.
3) измерять зависимость величины задержки от количества конечных узлов в сети.

11.4.2.2 Тестовый сценарий:

1) Cbench необходимо запускать три раза с имитацией одного коммутатора,
2) для коммутатора задавать различное количество подключенных конечных узлов (10^3, 10^4, 10^5, 10^6, 10^7).
3) Каждый контроллер должен быть запущен на одном ядре процессора.

11.4.3 Исследование надежности

11.4.3.1 В ходе исследования надежности контроллеров проверяется, как контроллер работает в течение длительного периода времени в условиях низкой нагрузки.

В качестве примера профиля трафика можно взять данные о трафике в сети университета Стэнфорд, представленные на Рисунке 11.1. Данный трафик необходимо сгенерировать при помощи средства hcprobe (аналог cbench, но который позволяет более тонкую настройку параметров тестирования), который может моделировать работу пяти коммутаторов, посылающих контроллеру сообщения PacketIn с заданной частотой.

11.4.3.2 Тестовый сценарий. Необходимо сгенерировать трафик, аналогичный трафику в сети университета Стенфорда с помощью средства hcprobe:

– Тестирование нужно проводить в течение 24 часов для каждого контроллера.

– По окончании этого времени зафиксировать следующие данные:

1) на какое количество сообщений, отправленных коммутаторами, не был получен ответ от контроллера;

2) какое количество сессий между контроллером и коммутатором было закрыто.

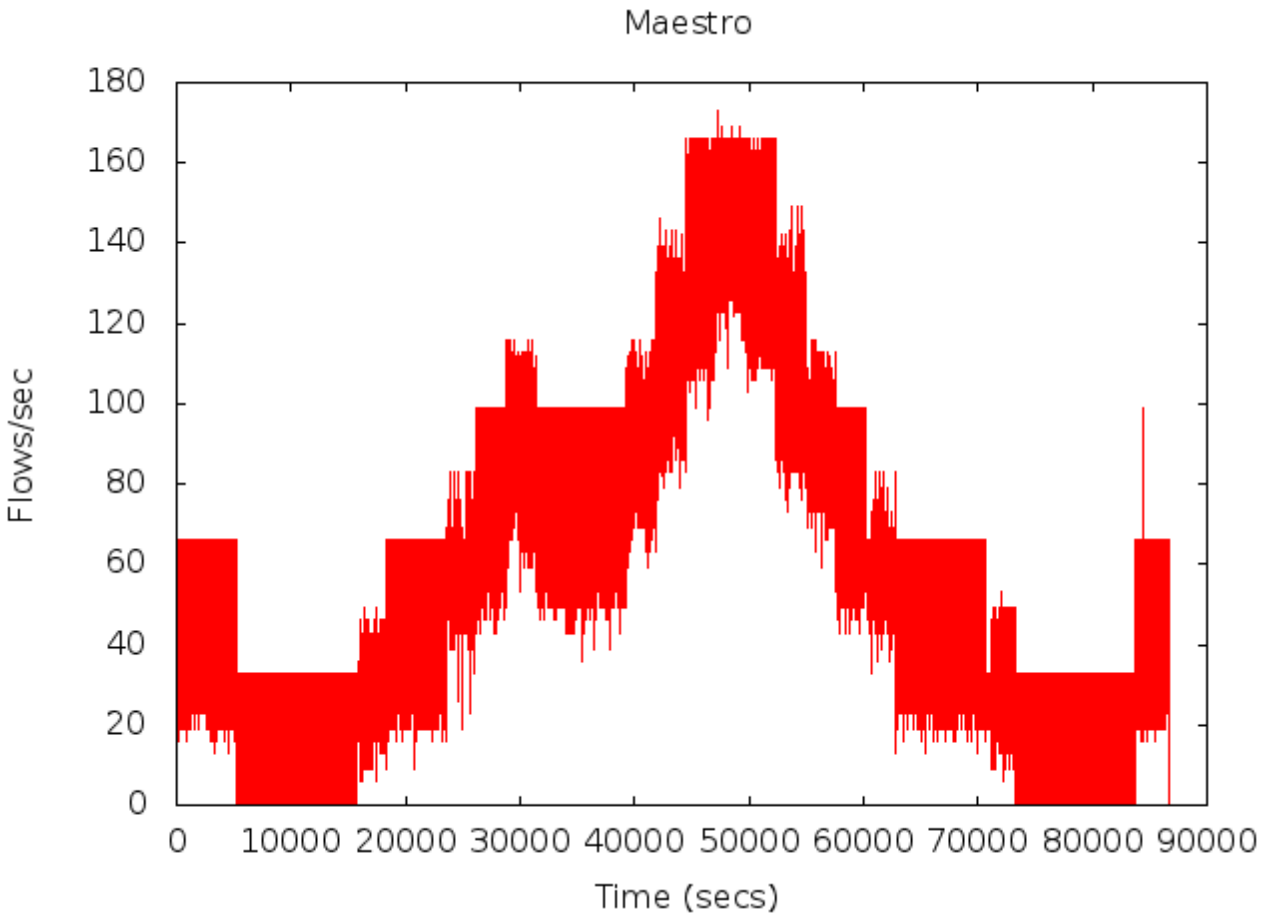

Рисунок 11.1 – Пример профиля трафика (тестирование Maestro)

11.4.4 Исследование безопасности

11.4.4.1 В ходе исследования безопасности контроллера проверяется, как контроллер обрабатывает OpenFlow сообщения с неправильно сформированными заголовками.

11.4.4.2 Тестовый сценарий. Нужно протестировать работу контроллера со следующими видами некорректных сообщений, сгенерированных при помощи hcprobe:

1. Для произвольного типа OpenFlow сообщений в поле «length» заголовка OpenFlow записывать значение, не соответствующее реальной длине сообщения. Неправильную длину сообщения выбирать следующими способами:

   а) длина, равная длине заголовка OpenFlow;

   б) длина, равная длине заголовка OpenFlow + A, где A — число, меньшее длины заголовка инкапсулированного OpenFlow сообщения (например, PacketIn или FlowMod);

   в) длина, превышающая реальную длину всего сообщения.

2. В поле «version» заголовка OpenFlow нужно записать неправильную версию протокола OpenFlow. В ходе инициализации соединения с контроллером коммутаторы анонсируют общение по протоколу OpenFlow v1.0, но в заголовок пакета OpenFlow, содержащего сообщение PacketIn, записывается несуществующая версия протокола.

3. В поле «type» заголовка OpenFlow записывать тип инкапсулированного сообщения (OFPT_PORT_STATUS), не соответствующий реальному типу сообщения (OFPT_PACKET_IN).

4. Для сообщения PacketIn в Ethernet заголовке инкапсулированного Ethernet кадра записывать код протокола (ARP) содержащегося в нем сообщения, не соответствующий реальному типу сообщения (IP пакет).

5. Для сообщения PortStatus в структуре, описывающей физический порт, в поле «port name» записать неправильную ASCII-строку, не содержащую в конце «\0».

11.5 Состав и описание экспериментального стенда

11.5.1 Состав стенда

Испытания контроллеров следует проводить на экспериментальном стенде, который состоит из следующих компонентов:

- Сервер 1. На сервере 1 разворачивается исследуемый контроллер и скрипт управления экспериментом.
- Сервер 2. На сервере 2 устанавливается OFlops, содержащий утилиту CBench для тестирования контроллеров согласно Рисунку 11.3.
- Сервер 1 и Сервер 2 соединены по каналу Ethernet 1Gb.

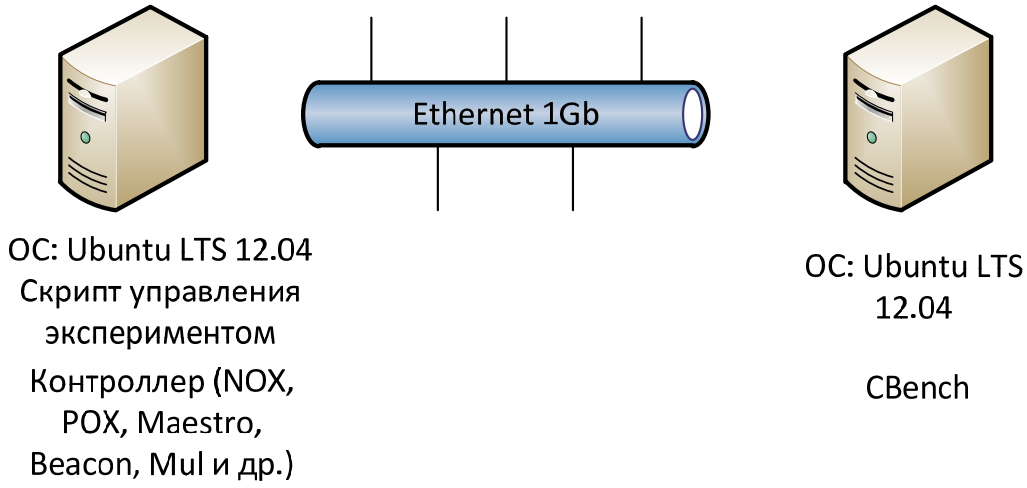

Рисунок 11.2 – Физическая схема экспериментального стенда

Скрипт управления последовательно запускает контроллеры на Сервере 1 и по протоколу ssh – Cbench на Сервере 2. Физическая и логическая схемы стенда представлены соответственно на рисунках 11.3 и 11.4.

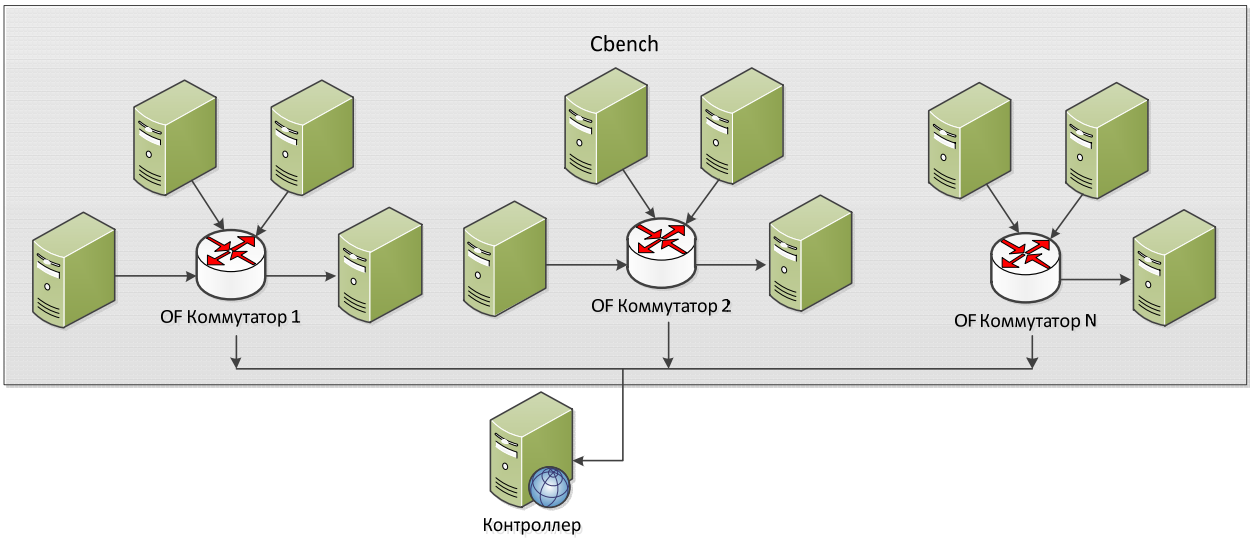

Рисунок 11.3 – Логическая схема экспериментального стенда

11.5.2 Инструментальное ПО стенда

На узлах экспериментального стенда устанавливается следующее ПО:

1) CBench – специальная утилита для тестирования производительности контроллеров.
2) Hcprobe – специальное средство для тестирования производительности контроллеров с помощью генерации сообщений (в том числе сообщений с измененными заголовками).
3) Скрипт управления экспериментом – данный скрипт содержит сценарии исследований.

11.6 Описание используемых программных средств

11.6.1 Программное средство Cbench

11.6.1.1 Общие сведения:

1) Разработчик: Rob Sherwood (BigSwitch)
2) Язык программирования: C
3) Дата начала разработки: Август, 2009.

4) Текущее состояние проекта: поддерживается

5) Репозиторий: git://gitosis.stanford.edu/oflops.git

Утилита CBench (controller benchmarker) [60] на сегодняшний день является единственным доступным инструментом для проведения тестирования производительности контроллеров. Cbench входит в состав OpenFlow платформы тестирования OFlops ([60], [61], [62]).

11.6.1.2 Измеряемые характеристики. Текущие показатели, которые измеряются с помощью Cbench согласно [60]:

1) Максимальное, среднее, минимальное количество ответов контроллера в секунду на packet_in сообщения.
2) Максимальная, средняя, минимальная задержка между прибытием пакета и соответствующим packet_in.
3) Показатели, вычисление которых планировалось внедрить в Cbench:
4) Максимальная скорость обработки flow-stats.
5) Максимальная совокупная скорость обработки статистики.

11.6.1.3 Cbench поддерживает два режима работы:

— Режим измерения пропускной способности (Throughput mode). В режиме пропускной способности каждый коммутатор отправляет сообщения контроллеру, не дожидаясь от него ответа, до тех пор, пока есть место в буфере отправки. Этот режим позволяет оценить количество запросов, которое контроллер может обрабатывать за единицу времени при максимальной нагрузке.

— Режим измерения задержки (Latency mode). В режиме задержки каждый коммутатор отправляет одно сообщение контроллеру и дожидается от него ответа, прежде чем отправить следующее, при этом отправление сообщений другими коммутаторами не блокируется.

11.6.1.4 Алгоритм работы Cbench:

1. Cbench эмулирует N коммутаторов.

2. Создает N OpenFlow сессий к контроллеру.

3. Если Cbench запущен в режиме Latency mode (по умолчанию), то:

   а) Посылает сообщение Packet_in контроллеру.

   б) Ожидает назад сообщения Flow_mod.

   в) Повторяет.

   г) Подсчитывает, сколько раз шаги с а) по с) происходят за секунду.

4. Если Cbench запущен в режиме Throughput mode, то для каждой сессии пока буфер не полон:

   а) Устанавливает packet_in в очередь.

   б) Подсчитывает сообщения Flow_mod и PacketOut по мере их возвращения.

5. По результатам работы каждого теста определяется суммарное количество потоков, установленных на всех коммутаторах, и среднее количество потоков, устанавливаемых контроллером в секунду (flow/sec).

Результат одного запуска Cbench: минимальное, максимальное и среднее значение flow/sec по всем тестам, а также среднеквадратическое отклонение.

11.6.1.5 Основные настраиваемые параметры Cbench

Основные настраиваемые параметры в Cbench представлены в Таблице 11.1.

Таблица 11.1 – Основные настраиваемые параметры Cbench

Продолжение Таблицы

Параметр	Значение	Описание	По умолчанию
-c/--controller	<str>	имя/IP адрес сервера, на котором запущен контроллер	Localhost
-l/--loops	<int>	количество тестов, проводимых за один запуск Cbench	16
-M/--mac-addresses	<int>	количество хостов с уникальными MAC адресами на один коммутатор	100000
-m/--ms-per-test	<int>	длительность одного теста в мс	1000
-p/--port	<int>	TCP порт, на котором работает контроллер	6633
-s/--switches	<int>	количество эмулируемых коммутаторов	16
-t/--throughput		запуск Cbench в режиме тестирования пропускной способности (по умолчанию без указания опции -t — в режиме тестирования задержки)	
-w/--warmup	<int>	количество тестов после запуска Cbench, результаты которых не учитываются в общей статистике (количество тестов для разминки)	1

Продолжение Таблицы

Параметр	Значение	Описание	По умолчанию
-C/--cooldown	<int>	количество тестов до завершения работы Cbench, результаты которых не учитываются в общей статистике	0
-r		Диапазон тестирования по количеству коммутаторов от 1 до N	
-i		Задержка между соединением групп коммутаторов к контроллеру (в миллисекундах)	
-I		Количество коммутаторов в присоединяемых группах коммутаторов	
-delay		Задержка начала тестирования после feature_reply	

11.6.1.6 Тестирование пропускной способности. Для тестирования пропускной способности контроллера необходимо запустить Cbench с помощью следующей команды:

```
cbench -c 192.168.11.56 -m 10000 -l 10 -M 200000 -s 32 -t
```

— Cbench запускается для тестирования пропускной способности контроллера по адресу 192.168.11.56, порт 6633.

— Длительность теста – 10000 миллисекунд.

— Количество тестов – 10.

— Эмулируется 32 коммутатора с 200000 уникальных хостов на каждый.

— Все коммутаторы «запускаются» одновременно.

11.6.1.7 Тестирование задержки контроллера. Для тестирования задержки контроллера необходимо запустить Cbench с помощью следующей команды:

```
cbench -c localhost -m 10000 -l 10 -M $50000 -s 1
```

— Cbench запускается для тестирования задержки контроллера, запущенного на локальном хосте на порте 6633.

— Длительность теста 10000 миллисекунд,

— Количество тестов – 10.

— Эмулируется 1 коммутатор с 50000 уникальных хостов.

11.6.2 Программное средство Hcprobe

11.6.2.1 Общие сведения:

1) Разработчик: Центр прикладных исследований компьютерных сетей (ЦПИКС).
2) Язык программирования: Haskell.
3) Дата начала разработки: Декабрь, 2012.
4) Текущее состояние проекта: активно развивается.
5) Репозиторий: https://github.com/voidlizard/hcprobe/commits/master

Традиционное средство тестирования контроллеров CBench [60] позволяет измерить базовые характеристики производительности существующих контроллеров – пропускную способность и задержку контроллера. Средство было разработано в 2009 году и по функциональности и набору измеряемых характеристик существенно не развивается с тех пор. CBench не позволяет проводить комплексный анализ характеристик производительности, надежности и масштабируемости контроллера, не позволяет гибко настраивать параметры

тестирования. Таким образом, по-прежнему актуальным является проблема разработки платформы или средства для комплексного анализа характеристик контроллеров.

В ЦПИКС было разработано средство HCProbe, которое направлено на восполнение пробела в области оценки надежности, безопасности контроллера и проверки его соответствия протоколу OpenFlow.

11.6.2.2 Можно выделить следующие особенности HCProbe:

1) HCProbe позволяет генерировать произвольные сообщения OpenFlow, изменять значение любого поля заголовка OpenFlow сообщения или его содержимое.
2) HCProbe реализован на языке программирования высокого уровня Haskell, что упрощает сопровождение и расширение функциональности данного средства.
3) HCProbe предоставляет гибкий API для разработки тестовых сценариев на языке Domain Specific Language.

11.6.2.3 Архитектура HCProbe – архитектура hcprobe согласно Рисунку 11.5 содержит следующие основные модули:

– *Модуль Network datagram*. Библиотека, предоставляющая интерфейс для генерации сообщений всех протоколов сетевого стека (кадры Ethernet, пакеты IP и ARP, пакеты TCP и UDP). Интерфейс предоставляет возможность задавать произвольные значения полей заголовков и генерировать произвольное содержимое пакета.

– *Модуль OpenFlow*. Библиотека для генерации и обработки OpenFlow сообщений.

– *Модуль Configuration*. Библиотека, предоставляющая интерфейс для задания параметров теста при помощи опций командной строки или конфигурационного файла.

– *Модуль FakeSwitch*. Модуль, эмулирующий работу коммутаторов, которые устанавливают соединения с контроллером. Данный модуль предоставляет интерфейс обработки и отправки сообщений контроллеру.

– *Модуль eDSL*. Модуль предоставляет упрощенный интерфейс для написания тестовых сценариев.

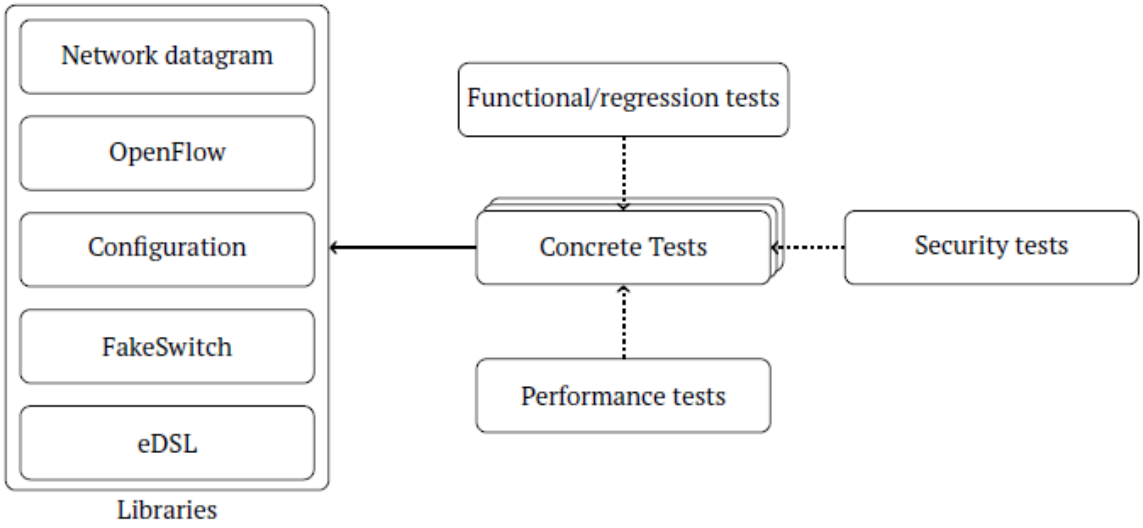

Рисунок 11.4 – Архитектура hcprobe

11.6.2.4 Написание тестовых сценариев для HCProbe. HCProbe предоставляет возможности для исследования поведения контроллера при получении различных типов OpenFlow сообщений. Таким образом, HCProbe позволяет проанализировать различные сценарии работы контроллера, в том числе и самые нестандартные, которые трудно воспроизвести с использованием реальных OpenFlow коммутаторов или решений для генерации трафика. Данное средство может применяться разработчиками для тестирования функциональности и безопасности контроллеров.

На Рисунке 11.5. Приведен пример использования eDSL для создания тестового сценария для HCProbe.

```
test1 = connectToController $ handshake $ do
    forever $ do

        let pktIn = makeOFPacket $ do
            makeOFHeader $ do
                putOFVersion (fromEnum OF_HDR_VERSION_1_0)
                putOFType    (fromEnum OFPT_FEATURES_REPLY)
                putOFLength  42 -- Bad length!

            makeOFData $ do
                putOFByteString randomByteString

        sendOFPacket pktIn
```

Рисунок 11.5 – Пример использования eDSL для создания теста

Сценарий заключается в отправке packet_in сообщений протокола OpenFlow с некорректно сформированным заголовком – неверно указана длина сообщения.

11.6.2.5 Основные настраиваемые параметры в HCProbe могут быть заданы при запуске при помощи опций командной строки или в конфигурационном файле. Поддерживаемые опции командной строки приведены в Таблице 11.3.

Таблица 11.2 – Опции командной строки HCProbe

Параметр	Описание	По умолчанию
`-c --config=PATH`	Путь к конфигурационному файлу	-
`--macspacedim=NUM`	Количество уникальных MAC адресов на один порт коммутатора	100
`--switchnum=NUM`	Количество моделируемых коммутаторов	64
`--portnum=NUM`	Количество физических портов на один коммутатор	48
`--maxtimeout=NUM`	Наибольшая задержка между отправкой двух сообщений одним коммутатором (мкс)	10
`--statsdelay=NUM`	Период, за который обновляется статистика (мкс)	300 000
`-t --testduration=NUM`	Продолжительность теста (мкс)	300 000 000
`-l --logfilename=PATH`	Путь к файлу, в который записывается лог статистики	
`-h --host=ADDRES`	IP адрес контроллера	127.0.0.1
`--port=NUM`	TCP порт контроллера	6633
`-? –help`	Показать информацию об опциях командной строки	
`-V --version`	Показать информацию о программе	

В конфигурационном файле могут быть заданы те же параметры, которые описываются опциями командной строки (кроме config, help и version). Формат файла приведен на Рисунке11.6.

```
[SECTION]
    option1=value
    option2=value
    option3=value(option1)
[END SECTION]
```

Рисунок 11.6 – Формат конфигурационного файла HCProbe

11.6.2.6 Установка HCProbe

Ниже на Рисунке 11.7 приведен скрипт для сборки и установки средства тестирования контроллеров HCProbe с помощью cabal-dev (локальная установка зависимостей).

```
apt-get install haskell-platform
apt-get install cabal
cabal update
cabal install cabal-dev

git clone git://github.com/voidlizard/hcprobe.git
cd hcprobe/src
cabal-dev install-deps
cabal-dev configure
cabal-dev build
```

Рисунок 11.7 – Скрипт для установки HCProbe

После сборки исполнимые файлы находятся в директории *hcprobe/src/dist/build/*. Исполнимый файл основного теста находится в директории *hcprobe/src/dist/build/hcprobe/*.

11.6.2.7 Пример строки для запуска HCProbe:

./hcprobe --switchnum=16 –macspacedim=1000 --host=10.0.0.3 --logfilename=log

– HCProbe запускается для тестирования контроллера по адресу 10.0.0.3 и портом 6633 (по умолчанию).

– Длительность теста 300 000 000 микросекунд.

— Моделируется 16 коммутаторов с 48 портами (по умолчанию) и с 1000 конечных узлов, подключенных к каждому порту.

— Лог статистики записывается в файл log.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работ по первому этапу НИР «Создание и развитие отечественной платформы с открытым программным кодом для управления программно-конфигурируемыми сетями (ПКС)» были получены следующие результаты.

Проведён аналитический обзор современной научно-технической, нормативной, методической литературы, публикаций по принципам построения, управления сетевыми ресурсами и архитектур ПКС в сравнении с традиционными сетями.

Выполнен сравнительный анализ существующих сетевых операционных систем для ПКС на основе доступных материалов: статей, документации, Интернет-ресурсов и исходных кодов. Рассмотрены следующие сетевые ОС: NOX, POX, SNAC, Beacon, Maestro, FloodLight, Trema, MUL, ONIX, Kandoo. Проведен сравнительный анализ архитектур, основных компонентов, особенностей реализации. Выявлены наиболее перспективные и активно развивающиеся проекты с открытыми исходными кодами для последующего исследования и развития: Beacon, FloodLight, MUL и Trema.

Сформирован набор критериев эффективности управления инфраструктурой КС и потоками данных. Были выделены и обоснованы следующие основные показатели производительности и надежности: время реакции, скорость передачи данных, пропускная способность, задержка передачи, среднее время наработки на отказ, вероятность отказа, интенсивность отказов, коэффициент готовности и другие.

Проведён сравнительный анализ методов и средств управления ПКС с традиционными методами и средствами управления сетевыми ресурсами и потоками данных в КС. По результатам анализа сделан вывод о том, что подход ПКС предоставляет более гибкие возможности по управлению потоками данных, перегрузками, управлению элементами сетевой инфраструктуры по сравнению с традиционным подходом.

Проведен анализ традиционных сервисов и приложений, применяемых для управления сетевой инфраструктурой ПКС, и спецификации требований к управлению компьютерными сетями и потоками данных в ПКС. Рассмотрена общая структура контроллера ПКС, выделены основные сервисы ядра контроллера и основные сетевые приложения для существующих сетевых ОС. Полученные результаты использованы при разработке требований к отечественной платформе управления ПКС.

Проведён анализ средств формирования OpenFlow таблиц для OpenFlow сетевых коммутаторов, возможности формирования сетевых срезов с помощью таких коммутаторов. Выполнено сравнение языков, используемых для описания правил заполнения OpenFlow-таблиц. Отмечена важность формальной (логической) проверки правильности создаваемых правил маршрутизации. Предложены два принципа такой проверки:

— динамическая (во время работы ПКС-контроллера) верификация построенных правил на соответствие спецификации;

— статическая (до начала работы ПКС-контроллера) верификация самого сетевого приложения, строящего правила; этот подход представляется более перспективным с точки зрения производительности и качества результата, но требует разработки языка описания сетевых приложений, достаточно выразительного и в то же время позволяющего формально проверять свойства приложений.

Проанализированы существующие подходы к виртуализации сетей. Отмечена перспективность использования подхода ПКС для автоматизации и централизации управления сетевой инфраструктурой, а также для обеспечения разделения логическими сетями не только уровня передачи данных, но и уровня управления (в частности, поддержки различных политик качества сервиса в разных логических сетях).

Проведены исследование, обоснование и выбор методов обеспечения QoS для ПКС. Показано, что ни один из традиционных подходов по отдельности не

покрывает все потребности пользователей сети. Сформулирована необходимость построения комплексного подхода к обеспечению качества сервиса и целесообразность его реализации в рамках технологии ПКС. Выдвинуты требования к элементам ПКС для реализации управления качеством сервиса.

Разработаны методы и алгоритмы управления сетевой отечественной ПКС платформы управления сетевыми ресурсами и потоками с помощью сетевой операционной системы. Полученные оценки сложности алгоритмов позволят добиться большей скорости вычисления маршрута по сравнению с традиционными подходами.

Сформирован набор требований для разработки прототипа отечественной платформы управления ПКС на втором этапе НИР. На втором этапе эти требования должны быть развёрнуты в проект ТЗ на ОКР по созданию указанной платформы.

Разработаны Программа и методики экспериментальных исследований сетевых ОС и экспериментальных реализаций алгоритмов управления ПКС. Исследование планируется в части измерения пропускной способности, задержки пакетов, надёжности, безопасности.

Результатом работы по первому этапу НИР являются: настоящий промежуточный отчёт о НИР за первый этап, программа и методики экспериментальных исследований, отчет о маркетинговых исследованиях и отчёт о патентных исследованиях.

Таким образом, задачи, поставленные в рамках первого этапа НИР, успешно выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Компьютерные сети: учебник для студ. высш. учеб. заведений [Текст]: в 2 т. Т. 2 / Р.Л. Смелянский. — М.: Издательский центр «Академия», 2011. — 240 с.

2 — Режим доступа: http://www.cisco.com/web/RU/news/releases/txt/2012/021512.html (дата обращения: 08.05.2013).

3 — Режим доступа: http://www.thg.ru/technews/20100604_152700.html (дата обращения: 08.05.2013).

4 — Режим доступа: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html (дата обращения: 10.05.2013).

5 — Режим доступа: http://www.cisco.com/web/RU/news/releases/txt/2010/060310.html (дата обращения: 07.05.2013).

6 Software-Defined Networking: The New Norm for Networks [Electronic resource] // Open Networking Foundation. — [2012]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf (accessed date: 28.04.2013).

7 OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01) [Electronic resource]. — [2009]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf (accessed date: 28.04.2013).

8 OpenFlow Switch Specification, Version 1.1.0 (Wire Protocol 0x02) [Electronic resource] // Open Networking Foundation. — [2011]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf (accessed date: 28.04.2013).

9 OpenFlow Switch Specification, Version 1.2.0 (Wire Protocol 0x03) [Electronic resource] // Open Networking Foundation. — [2011]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf (accessed date: 28.04.2013).

10 OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04) [Electronic resource] // Open Networking Foundation. — [2012]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed date: 28.04.2013).
11 OpenFlow Switch Specification, Version 1.3.1 (Wire Protocol 0x04) [Electronic resource] // Open Networking Foundation. — [2012]. — Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf (accessed date: 28.04.2013).
12 Web-страница проекта Mul [Электронный ресурс] — Режим доступа: http://sourceforge.net/p/mul/wiki/Home/ (дата обращения: 06.05.2013).
13 Kandoo: a framework for efficient and scalable offloading of control application [Текст] / Soheil Hassas Yeganeh, Yashar Ganjali
14 Компьютерные сети: учебник для студ. высш. учеб. заведений [Текст]: в 2 т. Т. 2 / Р.Л. Смелянский. — М.: Издательский центр «Академия», 2011. — 240 с.
15 Средства анализа и оптимизации локальных сетей [Текст] / Н.А. Олифер, В.Г. Олифер — 1998.
16 POX-Wiki [Электронный ресурс] — Режим доступа: https://openflow.stanford.edu/display/ONL/POX+Wiki
17 NOX Classic Wiki [Электронный ресурс] — Режим доступа: https://github.com/noxrepo/nox-classic/wiki
18 Floodlight Wiki [Электронный ресурс] — Режим доступа: http://www.projectfloodlight.org/documentation/
19 Beacon Guides [Электронный ресурс] — Режим доступа: https://openflow.stanford.edu/display/Beacon/Guides
20 Ryu OpenStack Design Summit 2012 presentation [Текст]
21 OSGI Framework [Electronic resource] — Mode of access: http://www.osgi.org/Technology/WhatIsOSGi
22 Spring Framework [Электронный ресурс] — Режим доступа: www.springsource.org

23 REST API Tutorial [Электронный ресурс] — Режим доступа: www.restapitutorial.org

24 LLDP: IEEE standard 802.1AB-2009 [Электронный ресурс] — Режим доступа: http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf

25 OpenStack [Электронный ресурс] — Режим доступа: www.openstack.org

26 Frenetic: a network programming language [Текст] / N. Foster, R. Harrison, M.J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker // Proceedings of the 16th ACM SIGPLAN international conference on Functional programming: ICFP '11. — New York, NY, USA: ACM, 2011. — С. 279–291.

27 The Yampa arcade [Текст] / A. Courtney, H. Nilsson, J. Peterson // Proceedings of the 2003 ACM SIGPLAN workshop on Haskell: Haskell '03. — New York, NY, USA: ACM, 2003. — С. 7–18.

28 A compiler and run-time system for network programming languages [Текст] / C. Monsanto, N. Foster, R. Harrison, D. Walker // Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages: POPL '12. — New York, NY, USA: ACM, 2012. — С. 217–230.

29 Nettle: Functional Reactive Programming of OpenFlow Networks [Текст] / A. Voellmy, P. Hudak // Practical Aspects of Declarative Languages. — 2011. — С. 235–249.

30 Virtual Local Area Network: IEEE 802.1Q-2005 Standard [Электронный ресурс] — Режим доступа: www.dcs.gla.ac.uk/~lewis/teaching/802.1Q-2005.pdf (дата обращения: 10.05.2013).

31 IPSec: RFC2401-2412 [Электронный ресурс] — Режим доступа: http://www.ietf.org/rfc/rfc2401.txt

32 Point-to-Point Tunneling Protocol: RFC 2637 [Электронный ресурс] — Режим доступа: http://www.ietf.org/rfc/rfc2637.txt

33 Layer Two Tunneling Protocol «L2TP»: RFC 2661 [Электронный ресурс] — Режим доступа: http://www.ietf.org/rfc/rfc2661.txt

34 A Method for Transmitting PPP Over Ethernet (PPPoE): RFC 2516 [Электронный ресурс] — Режим доступа: www.ietf.org/rfc/rfc2516.txt

35 OpenVPN: [Электронный ресурс] — Режим доступа: http://www.openvpn.net

36 Multiprotocol Label Switching Architecture RFC 3031 [Electronic resource] / E. Rosen, A. Viswanathan, R. Callon // Internet Engineering Task Force (IETF). — [2001]. — Mode of access: http://tools.ietf.org/html/rfc3031 (accessed date: 28.04.2013).

37 Provider Backbone Bridges: IEEE 802.1ah-2008 [Электронный ресурс] — Режим доступа: http://www.ieee802.org/1/pages/802.1ah.html

38 VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks: [Электронный ресурс] — Режим доступа: http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-03

39 NVGRE: Network Virtualization using Generic Routing Encapsulation [Электронный ресурс] — Режим доступа: http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-02

40 A Stateless Transport Tunneling Protocol for Network Virtualization (STT) [Электронный ресурс] — Режим доступа: http://tools.ietf.org/html/draft-davie-stt-03

41 OpenFlow Management and Configuration Protocol (OF-Config 1.1.1) [Электронный ресурс] — Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf

42 The Open vSwitch Database Management Protocol: [Электронный ресурс] — Режим доступа: http://tools.ietf.org/html/draft-pfaff-ovsdb-proto-02

43 Virtual Switching in an Era of Advanced Edges [Текст] / J. Pettit, J. Gross, B. Pfaff, S. Casado, S. Crosby // 2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES), ITC 222010.

44 OPENNETWORKINGLAB / flowvisor [Электронный ресурс] — Режим доступа: https://github.com/OPENNETWORKINGLAB/flowvisor/wiki

45 Nicira Network virtualization platform (NVP) [Electronic resource] // Nicira Products. — Mode of access: http://nicira.com/en/network-virtualization-platform/ (accessed date: 30.04.2013).

46 Requirements Analysis - A Management Perspective [Текст] / R. T. Yeh. — COMPSAC ’82: 1982. — С. 410–416.

47 Applying Extended Finite State Machines in Software Testing of Interactive Systems [Текст] / M. Fantinato, M. Jino // Interactive Systems Design, Specification, and Verification: Т. 2844: Lecture Notes in Computer Science.2003. — С. 34–45.

48 On Non-Functional Requirements in Software Engineering [Текст] / L. Chung, J.C.S. do P. Leite // Conceptual Modeling: Foundations and Applications2009. — С. 363–379.

49 Administrative Policies to Regulate Quality of Service Management in Distributed Multimedia Applications. [Текст] / M. Katchabaw, H. Lutfiyya, M.A. Bauer // MMNS: Т. 2839: Lecture Notes in Computer Science.Springer, 2003. — С. 341–354.

50 Shenker Integrated Services in the Internet Architecture: an Overview [Электронный ресурс] / R. Braden, D. Clark, S. Shenker // IETF Network Working Group, Request for Comments: 1633. — [1994]. — Режим доступа: http://tools.ietf.org/html/rfc1633

51 ElasticTree: saving energy in data center networks [Текст] / B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown // Proceedings of the 7th USENIX conference on Networked systems design and implementation: NSDI’10. — Berkeley, CA, USA: USENIX Association, 2010. — С. 17–17.

52 MARA: Maximum Alternative Routing Algorithm [Текст] / Y. Ohara, S. Imahori, R.V. Meter // INFOCOM’092009. — С. 298–306.

53 A survey on multi-constrained optimal path computation: Exact and approximate algorithms [Текст] / R.G. Garroppo, S. Giordano, L. Tavanti // Comput. Netw. — 2010. — Т. 54, № 17. — С. 3081–3107.

54 The Evolution of the RSVP Protocol [Электронный ресурс] / Robert Braden, [et. al.] // USC/lnformation Sciences Institute. — [2000]. — Режим доступа: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA374855 (дата обращения: 30.04.2013).

55 Next Steps in Signaling (NSIS): Framework, RFC 4080 [Электронный ресурс] / R. Hancock, G. Karagiannis, J. Loughney, S. Van den Bosch // Internet Engineering Task

Force (IETF). — [2005]. — Режим доступа: http://tools.ietf.org/html/rfc4080 (дата обращения: 30.04.2013).

56 A new approach to dynamic all pairs shortest paths [Текст] / C. Demetrescu, G.F. Italiano // J. ACM. — 2004. — Т. 51, № 6. — С. 968–992.

57 Объектно-ориентированный анализ и проектирование с примерами приложений на C++ [Текст] / Гради Буч. — Москва: Бином, 1998.

58 Pragmatic version control using Git [Текст]: Pragmatic starter kit / T. Swicegood. — Raleigh, N.C: Pragmatic, 2008. — 179 с.

59 Network traffic characteristics of data centers in the wild [Текст] / T. Benson, A. Akella, D.A. Maltz // Proceedings of the 10th ACM SIGCOMM conference on Internet measurement: IMC '10. — New York, NY, USA: ACM, 2010. — С. 267–280.

60 OpenFlow Benchmarking [Electronic resource] // OpenFlow Wiki. — Mode of access: http://www.openflow.org/wk/index.php/Oflops

61 OFlops User Manual [Electronic resource]. — [2011]. — Mode of access: http://www.openflow.org/wk/index.php/File:Manual.pdf

62 OFLOPS: An Open Framework for OpenFlow Switch Evaluation [Текст] / C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A.W. Moore // Passive and Active Measurement: Т. 7192. — Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. — С. 85–95.