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

УДК 004.4:004.7
№ госрегистрации 01201356240
Инв. №

УТВЕРЖДАЮ
Проректор МГУ
имени М.В. Ломоносова

_____ В.Е. Подольский
«___» _____ 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

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ ПОСТАВЛЕННЫХ ПЕРЕД НИР ЗАДАЧ. ОБОБЩЕНИЕ И ОЦЕНКА РЕЗУЛЬТАТОВ ИССЛЕДОВАНИЙ

(заключительный)

2013-1.4-14-514-0093-040

Руководитель темы
профессор, чл.-корр. РАН

_____ Р.Л. Смелянский
подпись, дата

Москва 2013

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы	подпись, дата	Р.Л. Смелянский (введение, заключение)
Исполнители темы		
Старший научный сотрудник, к.ф.-м.н.	подпись, дата	А.Г. Бахмуров (разделы 6, 7, 8)
Научный сотрудник, к.ф.-м.н.	подпись, дата	В.В. Балашов (раздел 6)
Младший научный сотрудник, к.ф.-м.н.	подпись, дата	А.П. Капитонова (раздел 7, 8)
Ассистент	подпись, дата	Д.Ю. Волканов (разделы 4, 8)
Младший научный сотрудник	подпись, дата	М.В. Чистолинов (раздел 2, 5)
Программист	подпись, дата	В.А. Алтухов (разделы 1,3,4)
Инженер	подпись, дата	А.В. Сапожников (разделы 2, 5)
Программист	подпись, дата	М.А. Булгаков (раздел 2)
Программист	подпись, дата	А.А.Сковорода (раздел 1)
Программист	подпись, дата	М.Н. Самойлов (раздел 1)
Программист	подпись, дата	Е.В. Чемерицкий (разделы 1,3,4)
Нормоконтролёр, инженер	подпись, дата	С.А. Косачева

РЕФЕРАТ

Страниц 145. Рисунков 19. Таблиц 3. Источников 26.

Ключевые слова: ПРОГРАММНО-КОНФИГУРИРУЕМЫЕ СЕТИ (ПКС), СТАНДАРТ OPENFLOW, СЕТЕВАЯ ОПЕРАЦИОННАЯ СИСТЕМА, УПРАВЛЕНИЕ СЕТЕВЫМИ РЕСУРСАМИ, ПРОГРАММНАЯ МОДЕЛЬ ПЛАТФОРМЫ УПРАВЛЕНИЯ

Объектом исследования являются средства управления сетевыми ресурсами и потоками в программно- конфигурируемых сетях.

Цели исследования:

– проведение исследований и сравнительного анализа существующих сетевых операционных систем для ПКС, выявление перспективных направлений их развития, сравнительный анализ перспектив последующего развития существующих СОС на основе критериев производительности, масштабируемости, надежности, безопасности;

– разработка программной модели платформы управления сетевыми ресурсами ПКС с открытым программным кодом.

Основные результаты исследования:

– разработана программная модель платформы управления сетевой инфраструктурой ПКС в соответствии со сформированным набором требований;

– проведено экспериментальное исследование сетевых ОС и разработанных алгоритмов, проведен их анализ;

– проведена технико-экономическая оценка рыночного потенциала полученных результатов;

– сформированы рекомендации и предложения по использованию результатов НИР в реальном секторе экономики, а также дальнейших исследованиях и разработках;

– разработаны методические материалы по построению и настройке ПКС в российских университетах и научно-исследовательских организациях;

– разработан проект технического задания на проведение ОКР по теме: «Разработка отечественной распределенной платформы управления с открытым программным кодом для ПКС».

Разработки, созданные на основе результатов НИР, могут быть использованы в системах управления сетями национального масштаба, корпоративными сетями, сетями ЦОД, сетями сервис-провайдеров, а также в домашних сетях.

Результаты НИР могут быть использованы как основа для проведения НИР и ОКР по направлениям:

– разработка распределенной платформы управления в соответствии с разработанным проектом ТЗ и разработанной программной моделью платформы управления;

– доказательство корректности разработанных алгоритмов для контроллера;

– совершенствование программной модели и архитектуры платформы управления сетевой инфраструктурой ПКС;

– разработка новых модулей сетевой ОС;

– совершенствование алгоритмов функционирования модулей;

– развитие методик для экспериментальной оценки показателей производительности, масштабируемости, надежности и безопасности сетевых ОС.

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)
ПМИ	Программа и методики испытаний	
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	
ICMP	Internet Control Message Protocol	

Обозначения	Расшифровка	Аналог (русск.)
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	
NMC	Network Management Center	

Обозначения	Расшифровка	Аналог (русск.)
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	
SOM	Security Operations Manager (модуль IBM Tivoli)	

Обозначения	Расшифровка	Аналог (русск.)
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 на общий 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 актуальность проводимой НИР.

Настоящий документ представляет собой научно-технический отчет по заключительному этапу НИР «Создание и развитие отечественной платформы с открытым программным кодом для управления программно-конфигурируемыми сетями (ПКС)». Документ содержит отчет по пунктам 2.1, 2.2, 2.4, 2.7-2.9 календарного плана (результаты по пп. 2.3, 2.5-2.6, 2.10 приведены в отдельных документах) 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 научно-технического задела в области создания программных средств с открытым кодом для управления сетевыми ресурсами и потоками данных на основе подхода программно-конфигурируемых сетей с оценкой возможности применения ПКС для повышения эффективности управления компьютерными сетями и потоками данных; проведение исследований и сравнительного анализа существующих сетевых операционных систем для ПКС, выявление перспективных направлений их развития, сравнительный анализ перспектив последующего развития существующих СОС на основе критериев производительности, масштабируемости, надежности, безопасности; разработка программной модели платформы управления сетевыми ресурсами ПКС с открытым программным кодом.

В ходе НИР был подготовлен промежуточный отчет за первый этап «Выбор направления исследований. Теоретические исследования поставленных перед НИР задач».

На первом этапе были проведены следующие работы в соответствии с календарным планом, результаты которых были отражены в промежуточном отчете за первый этап НИР:

– Проведен аналитический обзор современной научно-технической, нормативной, методической литературы и публикаций по принципам построения, управления сетевыми ресурсами и архитектур ПКС в сравнении с традиционными сетями.

– Проведен 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.

– Сформирован набор критериев эффективности управления инфраструктурой КС и потоками данных. Были выделены 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.

– Проведен сравнительный анализ методов и средств управления ПКС с традиционными методами и средствами управления сетевыми ресурсами и потоками данных в КС.

– Проведен анализ традиционных сервисов/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.

– Проведен анализ средств формирования OpenFlow таблиц для OpenFlow сетевых коммутаторов, возможности формирования сетевых срезов с помощью таких коммутаторов.

– Проанализированы существующие подходы к виртуализации сетей.

– Проведены исследование, обоснование и выбор методов обеспечения QoS для ПКС.

– Разработаны методы и алгоритмы управления сетевыми ресурсами и потоками с помощью сетевой операционной системы.

– Разработана программа и методики испытаний для экспериментального исследования сетевых ОС и разработанных алгоритмов.

– Сформирован набор требований для разработки отечественной платформы управления ПКС.

Целью заключительного этапа НИР является проведение экспериментальных исследований и разработка программных реализаций алгоритмов управления сетевыми ресурсами сегмента ПКС на основе теоретических исследований, проведенных на первом этапе НИР, а также обобщение и оценка результатов НИР.

Задачи, решаемые на заключительном этапе НИР:

– Программная реализация функциональных алгоритмов сегмента ПКС путем выборочной проверки функциональных алгоритмов сегмента ПКС.

– Разработка программной модели платформы управления ПКС на основе выбранной сетевой ОС и алгоритмов управления сегментами ПКС.

– Создание стенда для проведения экспериментальных исследований.

– Проведение экспериментальных исследований характеристик программной модели платформы управления и существующих сетевых ОС для ПКС в соответствии с Программой и методиками экспериментальных исследований.

– Разработка методических материалов по построению и настройке ПКС в российских университетах и научно-исследовательских организациях.

Разработка технического задания на ОКР по разработке отечественной распределенной платформы управления для ПКС.

Проведение технико-экономической оценки рыночного потенциала полученных результатов.

Разработка рекомендаций и предложений по использованию результатов НИР в

реальном секторе экономики, а также в дальнейших исследованиях и разработках.

В первом разделе данного отчета заключительного этапа НИР в соответствии с пунктом 2.1 календарного плана проводится разработка функциональных алгоритмов для сегмента ПКС: алгоритма автоматического построения топологии ПКС; алгоритма маршрутизации в рамках сегмента ПКС; алгоритма кодирования/декодирования пакетов; алгоритма для поддержки механизма подписки на события для модулей сетевой ОС.

Во втором разделе отчета описана разработка программной модели платформы управления на основе выбранной сетевой ОС и алгоритмов управления сегментами ПКС.

В третьем разделе приводится перечень программной документации для разработанных программных реализаций модулей СОС.

В четвертом разделе отчета приводятся результаты экспериментального исследования существующих сетевых ОС и программной модели платформы управления.

Пятый раздел содержит методику развёртывания экспериментальных сегментов ПКС в учебных и научно-исследовательских организациях.

В шестом разделе проводятся обобщение и оценка результатов НИР.

В седьмом разделе отчета дана технико-экономическая оценка рыночного потенциала полученных результатов.

В восьмом разделе отчета формулируются рекомендации и предложения по использованию результатов НИР в реальном секторе экономики, а также в дальнейших исследованиях и разработках.

В заключении изложены основные результаты заключительного этапа НИР.

1 Программная реализация функциональных алгоритмов сегмента ПКС путем выборочной проверки следующих функциональных алгоритмов сегмента ПКС Разработка методов и алгоритмов управления сетевой инфраструктурой

1.1 Алгоритм автоматического построения топологии ПКС

Реализован алгоритм, предложенный в п. 9.2.3 и 9.2.4 промежуточного отчета о НИР [1]. Процедура построения топологии ПКС состоит из двух этапов: подготовка и рассылка на коммутаторы служебных LLDP/BDDP кадров и построение топологии на основе анализа полученных от коммутаторов сообщений Packet-In.

На первом этапе, как показано на рисунке 1.1, на коммутаторах устанавливаются правила, согласно которым LLDP/BDDP кадры должны быть отосланы на контроллер. Также формируются сообщения Packet-Out, содержащие служебные LLDP/BDDP кадры, которые рассылаются на все порты всех коммутаторов.

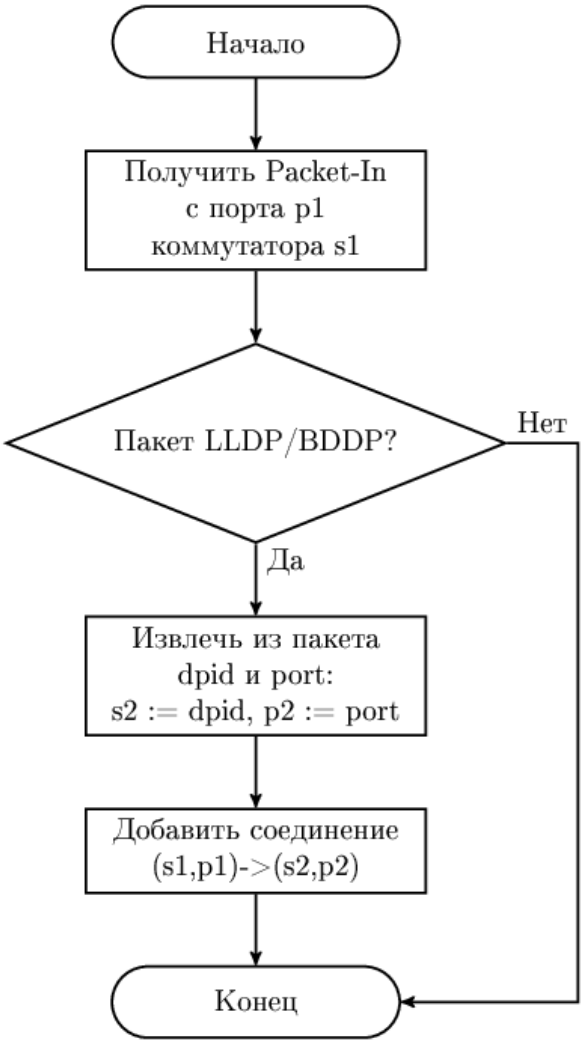

Рисунок 1.1 – Блок-схема алгоритма построения топологии ПКС: установка правил и рассылка Packet-Out

На втором этапе, в соответствии с рисунком 1.2, при получении от коммутатора сообщения Packet-In с кадром LLDP/BDDP содержимое кадра анализируется, и в топологию добавляется соединение между соответствующими коммутаторами.

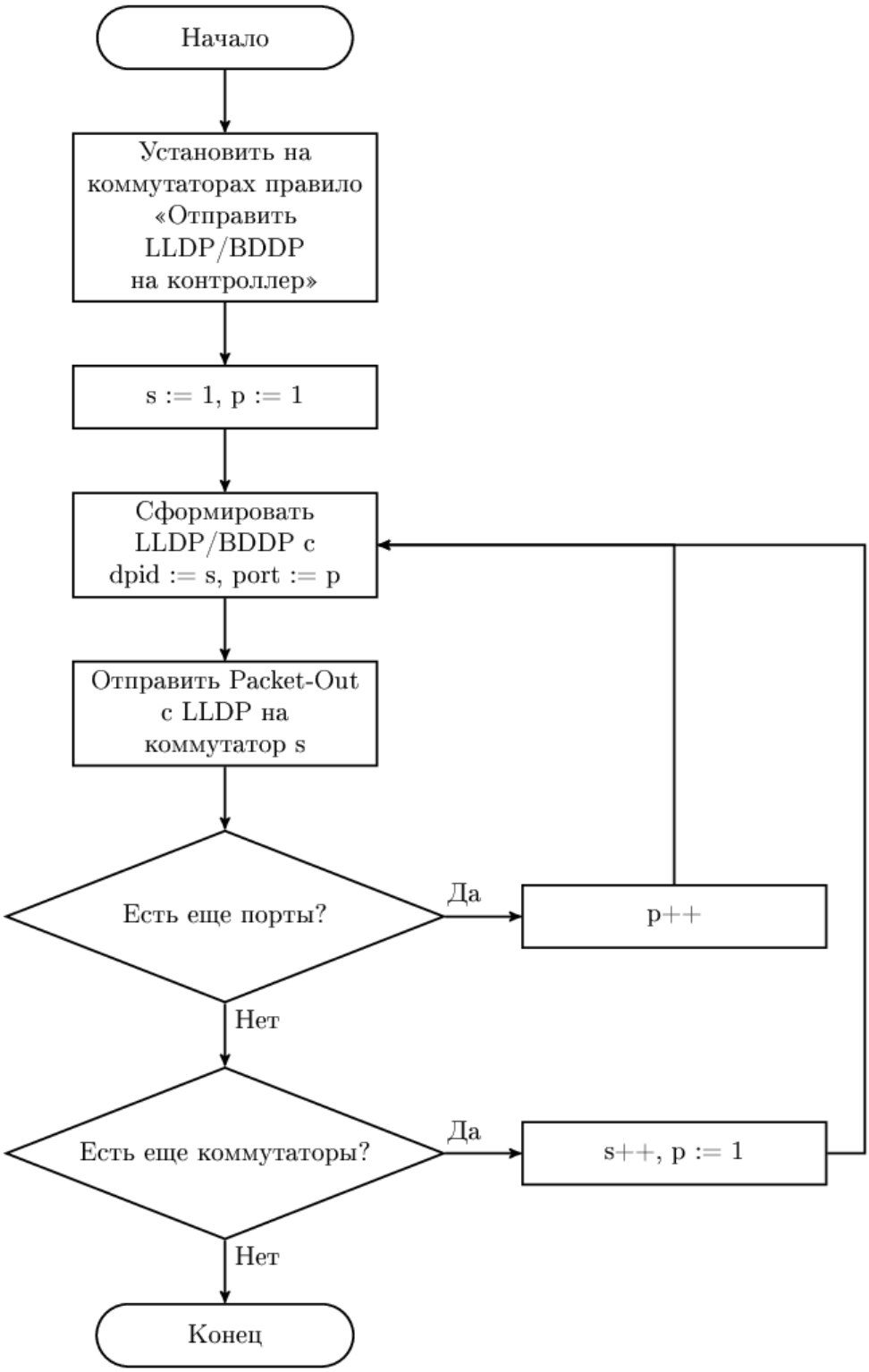

Рисунок 1.2 – Блок-схема алгоритма построения топологии ПКС: добавление соединений на основе анализа Packet-In

1.2 Алгоритм маршрутизации в рамках сегмента ПКС

Реализован алгоритм Деметрецу и Италиано, предложенный в п. 9.1.3.2 промежуточного отчета о НИР [1].

Построение кратчайших маршрутов и их динамическое перестроение в случае изменения топологии происходит путем поддержания множеств кратчайших путей, локальных кратчайших путей (любой подпуть которых является кратчайшим) и их правых и левых расширений.

Перестроение маршрутов в сети происходит при получении сообщения о включении и отключении портов, а также при появлении новых узлов в сети и удалении существующих узлов. При изменении топологии инициируется вызов функции fully_update, инициирующей обновления вершины *u,* как показано на рисунке 1.3.

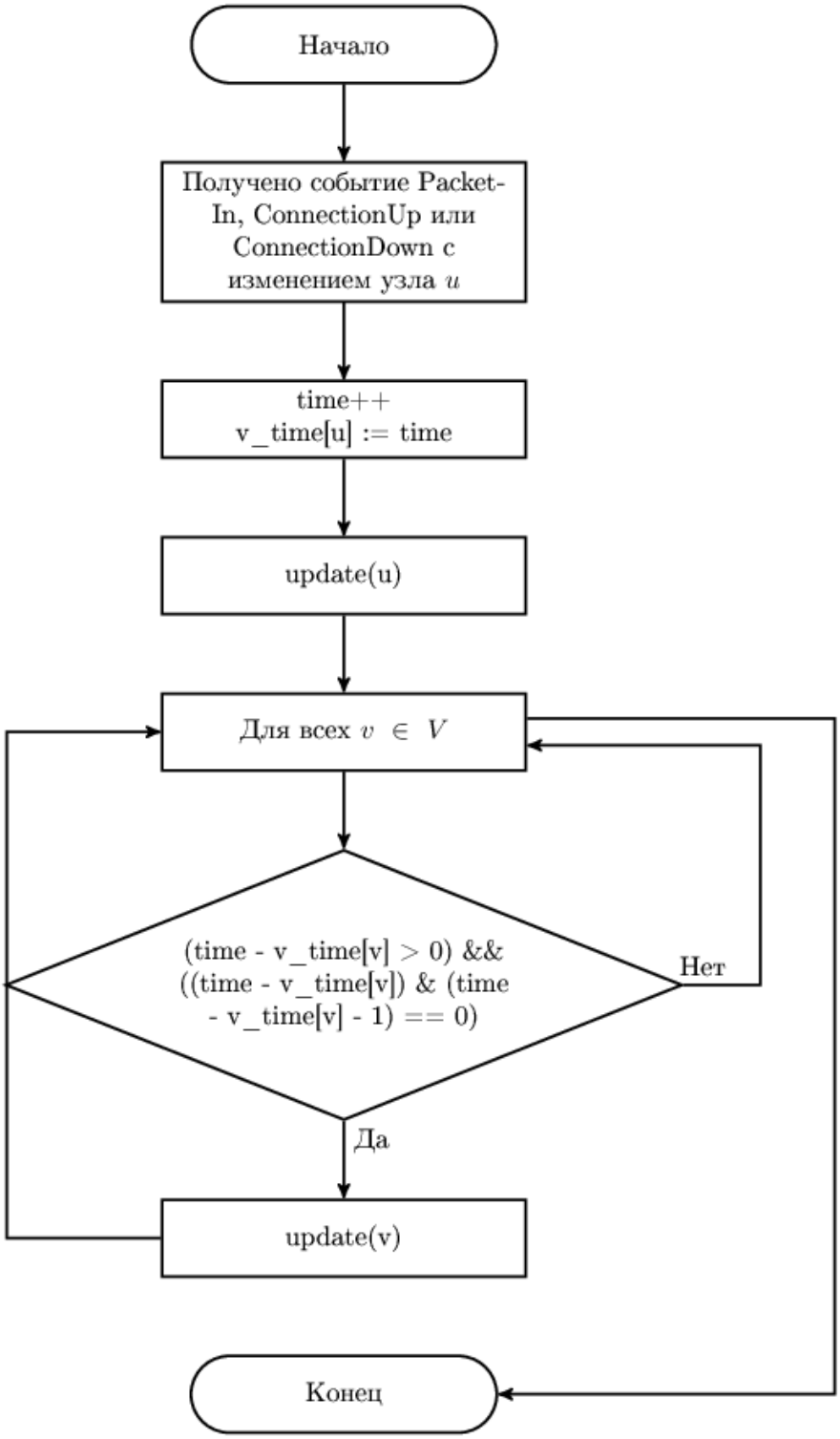

Рисунок 1.3 – Алгоритм построения кратчайших путей: функция fully_update

Функция fully_update, является интерфейсом функции update, которая последовательно вызывает функции fixup и cleanup, как показано на рисунке 1.4.

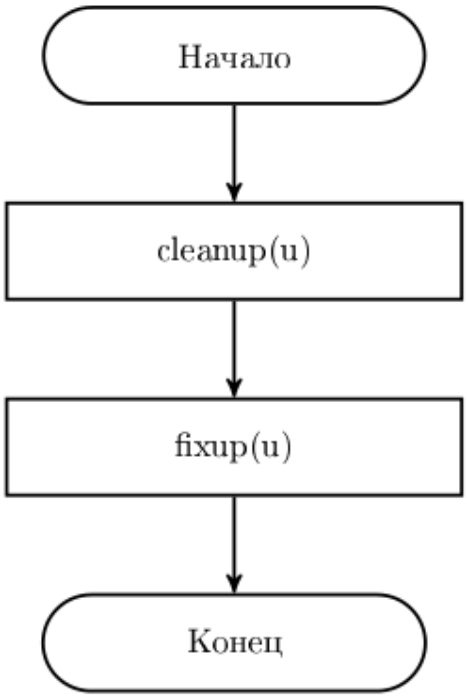

Рисунок 1.4 – Алгоритм построения кратчайших путей: функция update

Описание работы каждого из блоков дано ниже:

Cleanup. На вход функция получает вершину *u*, для которой происходит обновление. Цель функции – удалить из построенных ранее множеств кратчайших путей все маршруты, проходящие через вершину *u*. Удаление происходит итерационно. Для этого вводится множество Q, изначально состоящее из вершины *u*. Далее из построенных ранее множеств удаляются маршруты из Q, а в Q добавляются маршруты из расширений множеств.

Fixup. На вход функция получает вершину *u*, для которой происходит обновление. Цель вызова функции – добавить в измененные на предыдущем шаге множества кратчайшие маршруты, проходящие через вершину *u*. Функция работает в 3 этапа:

– В множества добавляются тривиальные маршруты – ребра, входящие в/исходящие из вершины u.

– Инициализируется множество H – множество, состоящее из всех кратчайших путей для каждой пары вершин.

– Каждый кратчайший маршрут каждой пары вершин из H добавляется в множество кратчайших маршрутов (если его не было). Далее этот маршрут комбинируется с его правыми и левыми расширениями. Полученные маршруты добавляются в множество локальных кратчайших маршрутов.

1.3 Алгоритмы кодирования/декодирования пакетов

Реализован алгоритм, предложенный в п. 9.4.2 и 9.4.3 промежуточного отчета о НИР [1].

Алгоритм декодирования пакетов представлен на рисунке 1.5. Из очереди полученных сообщений последовательно считываются сообщения OpenFlow. После анализа заголовка сообщения принимается решение о сбросе сообщения, если поля заголовка некорректны, или о преобразовании сообщения во внутреннее представление и рассылке его подписчикам.

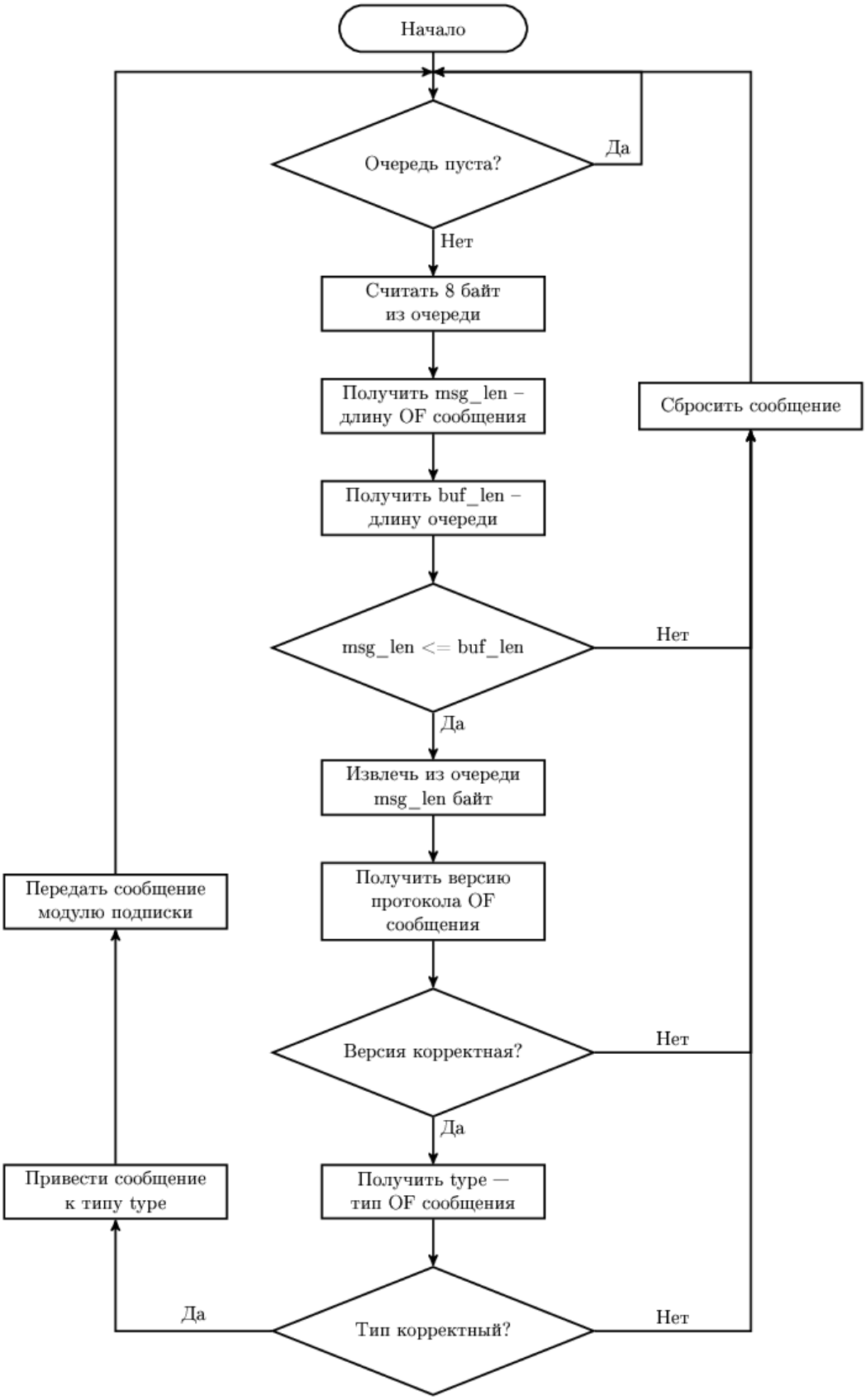

Рисунок 1.5 – Блок-схема алгоритма декодирования пакетов

Алгоритм кодирования пакетов представлен на рисунке 1.6. Для сообщения вычисляются значения полей заголовка OpenFlow, заголовок добавляется к сообщению. Далее полученное сообщение преобразуется в битовую строку и отправляется.

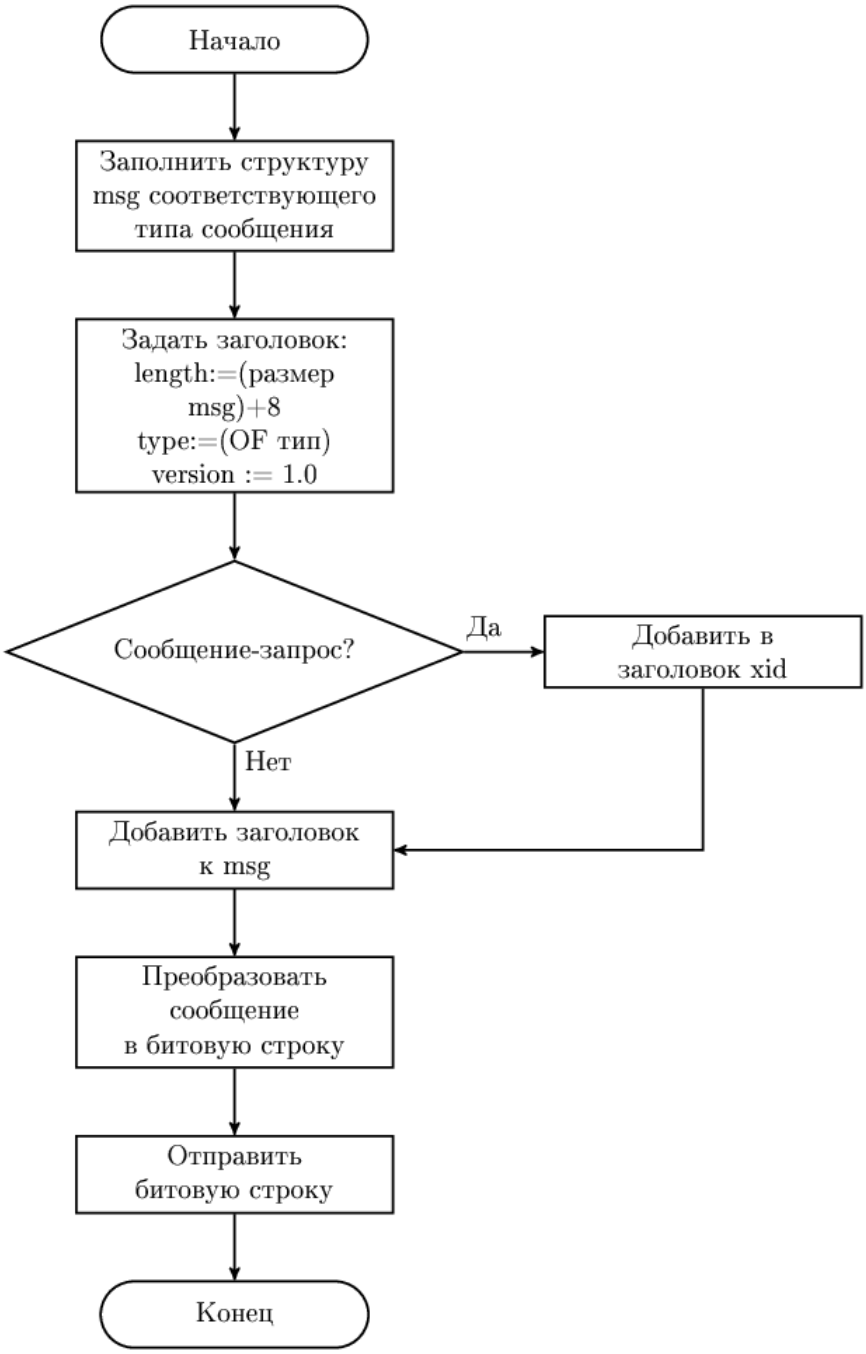

Рисунок 1.6 – Блок-схема алгоритма кодирования пакетов

1.4 Алгоритм для поддержки механизма подписки на события для модулей сетевой ОС

Реализован алгоритм, предложенный в п. 9.3.2 и 9.3.3.2 промежуточного отчета о НИР [1]. Алгоритм реализует три основные функции, которые описаны ниже.

Регистрация нового типа события отправителем отображена на рисунке 1.7. Для каждого отправителя создается своя очередь сообщений. Отправитель объявляет типы событий, на которые могут подписаться получатели.

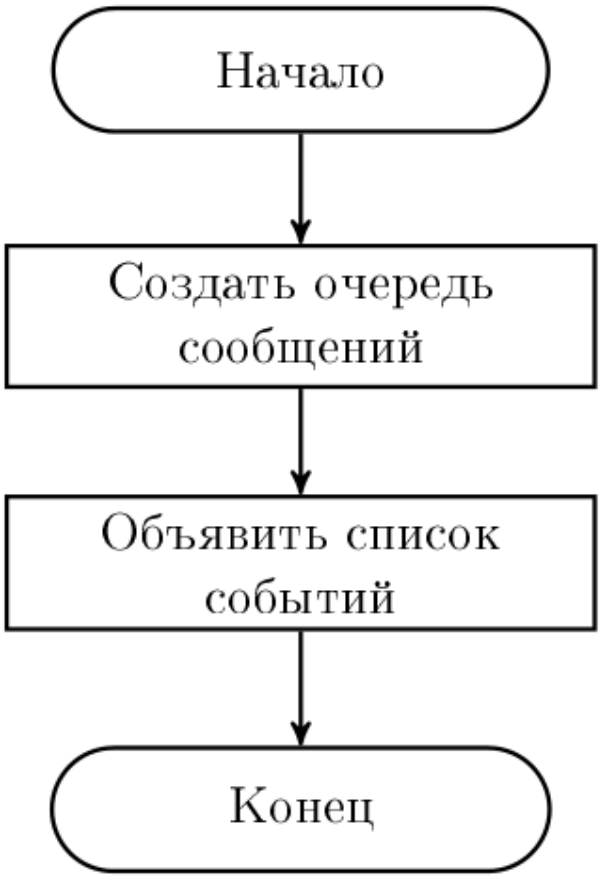

Рисунок 1.7 – Блок-схема алгоритма регистрации событий

Подписка на событие получателем изображена на рисунке 1.8. Для подписки на события получатель добавляет себя в список подписчиков соответствующего отправителя и реализует функции — обработчики событий, в которых он заинтересован.

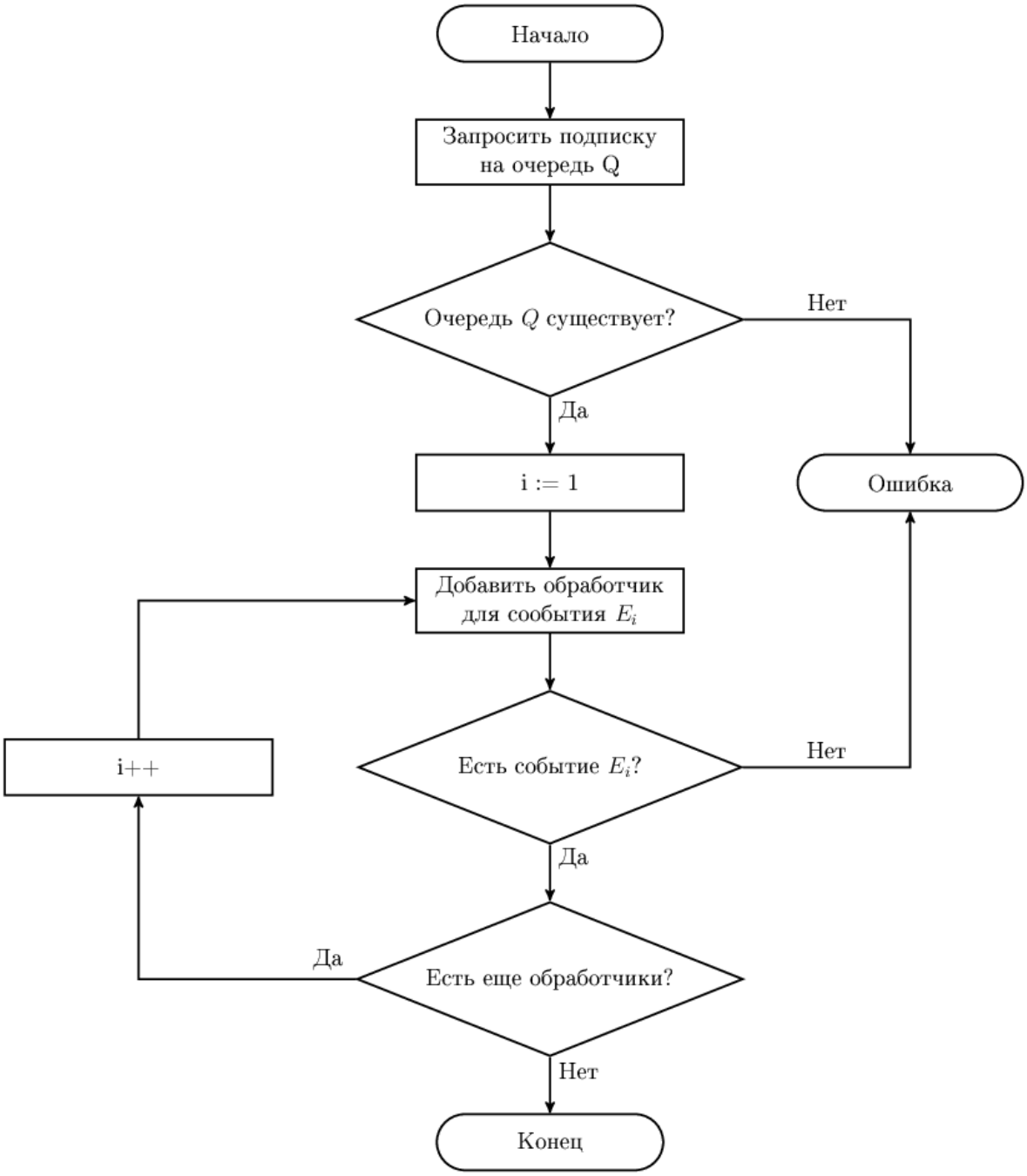

Рисунок 1.8 – Блок-схема алгоритма подписки на события

Алгоритм отправки уведомления о событии представлен на рисунке 1.9. Отправитель посылает уведомление о событии в ассоциированную с ним очередь сообщений. Сообщения с уведомлением рассылаются последовательно всем подписчикам данной очереди, которые реализовали обработчик для данного

события. В случае, если некоторый подписчик запретил дальнейшую рассылку сообщений, рассылка прекращается.

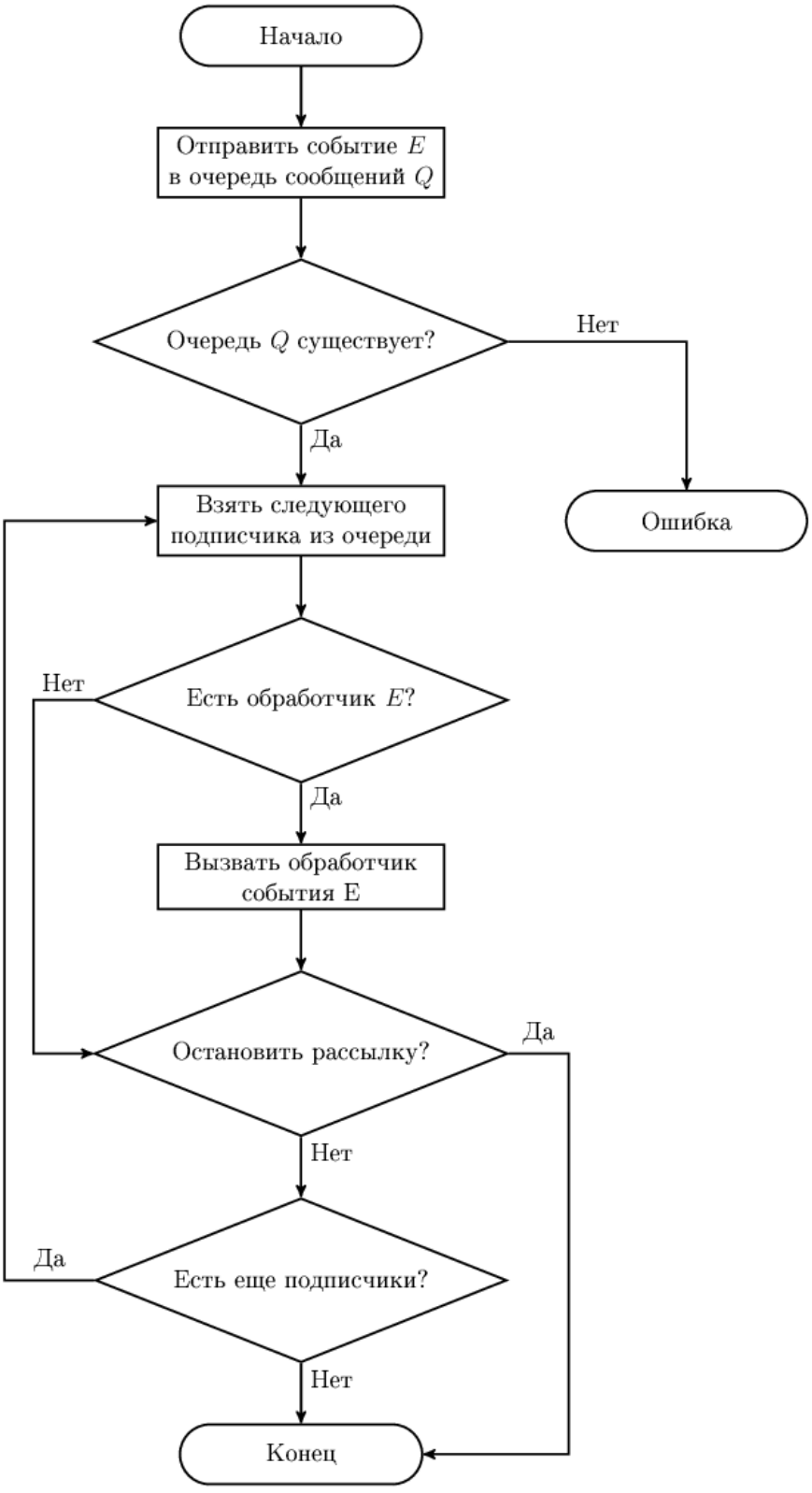

Рисунок 1.9 – Блок-схема алгоритма рассылки уведомлений о событии.

2 Разработка программной модели платформы управления ПКС на основе выбранной сетевой ОС и алгоритмов управления сегментами ПКС

2.1 Общие положения

Целью данного мероприятия является разработка программной модели платформы управления ПКС, отражающей следующие основные аспекты функционирования сетевой операционной системы (контроллера) для ПКС:

– построение топологии сети;

– обработка контроллером Openflow сообщений от коммутаторов ПКС сети;

– взаимодействие компонентов и приложений контроллера с помощью механизма подписки на события;

– маршрутизация трафика в сети за счет построения маршрутов на основе топологии сети и формирование соответствующих правил для коммутаторов;

– распределение обработки запросов от коммутаторов по ядрам сервера, на котором установлен контроллер.

2.2 Модель платформы управления ПКС

2.2.1 Основные функции платформы управления ПКС

Двумя основными функциями платформы управления ПКС, включающей в себя сетевую ОС и набор сетевых приложений, являются:

– управление и мониторинг состояния сетевой инфраструктуры ПКС – реализуется сетевой ОС (контроллером);

– управление потоками данных в сети ПКС – реализуется сетевыми приложениями, работающими поверх ОС .

2.2.2 Основные функции сетевой ОС

Рассмотрим подробнее основные функции сетевой ОС (контроллера):

– Управление устройствами сети (конфигурацией и состоянием): управление топологией (построение топологии сети, обработка добавления/удаления новых элементов сети).

- Управление соединениями с сетевыми устройствами:
    1) балансировка соединений по ядрам процессора;
    2) установление, разрыв, поддержка и возобновление соединения.
- Управление сервисами сетевой ОС:
    1) предоставление API сетевых сервисов для сетевых приложений;
    2) обеспечение взаимодействия между сервисами;
    3) динамическая загрузка сервисов.
    4) обработка ошибок, связанных с работой сервисов и их взаимодействием
- Управление сетевыми приложениями:
    1) одновременный запуск нескольких приложений;
    2) загрузка приложений;
    3) регистрация приложений;
    4) обработка ошибок приложений и их взаимодействия;
    5) регистрация на события от приложений;
    6) приоритизация приложений;
    7) обеспечение взаимодействия между приложениями;
    8) балансировка приложений по ядрам процессора;
    9) разграничение прав доступа приложений к элементам сети.
- Управление доступными ресурсами сервера:
    1) поддержка многопоточности;
    2) мониторинг загрузки потоков.
- Управление событиями:

1) обработка событий от сетевых устройств;

2) механизмы инициализации новых событий;

3) распространение событий между сервисами и сетевыми приложениями.

Диаграмма функционирования сетевой ОС представлена на рисунке 2.1.

Рисунок 2.10 – Диаграмма функционирования сетевой ОС

2.2.3 Основные функции сетевых приложений

Основными функциями сетевых приложений являются:

– управление потоками данных в сети;

– фильтрация потоков;

– анализ потоков;

– маршрутизация потоков;

– сбор статистики о потоках данных в сети.

2.3 Архитектура платформы управления

На основе построенной модели функционирования платформы управления ПКС можно построить общую архитектуру контроллера, выделить основные уровни, ключевые модули и компоненты, которые должны покрывать перечисленные функциональные возможности.

Общая программная модель платформы управления, представленная на рисунке 2.2, является многоуровневой и содержит шесть базовых уровней:

- уровень взаимодействия с сетью;
- уровень обработки OpenFlow сообщений;
- уровень обработки событий;
- уровень сетевых сервисов и внутренних приложений;
- интерфейс для сетевых приложений контроллера;
- уровень сетевых приложений.

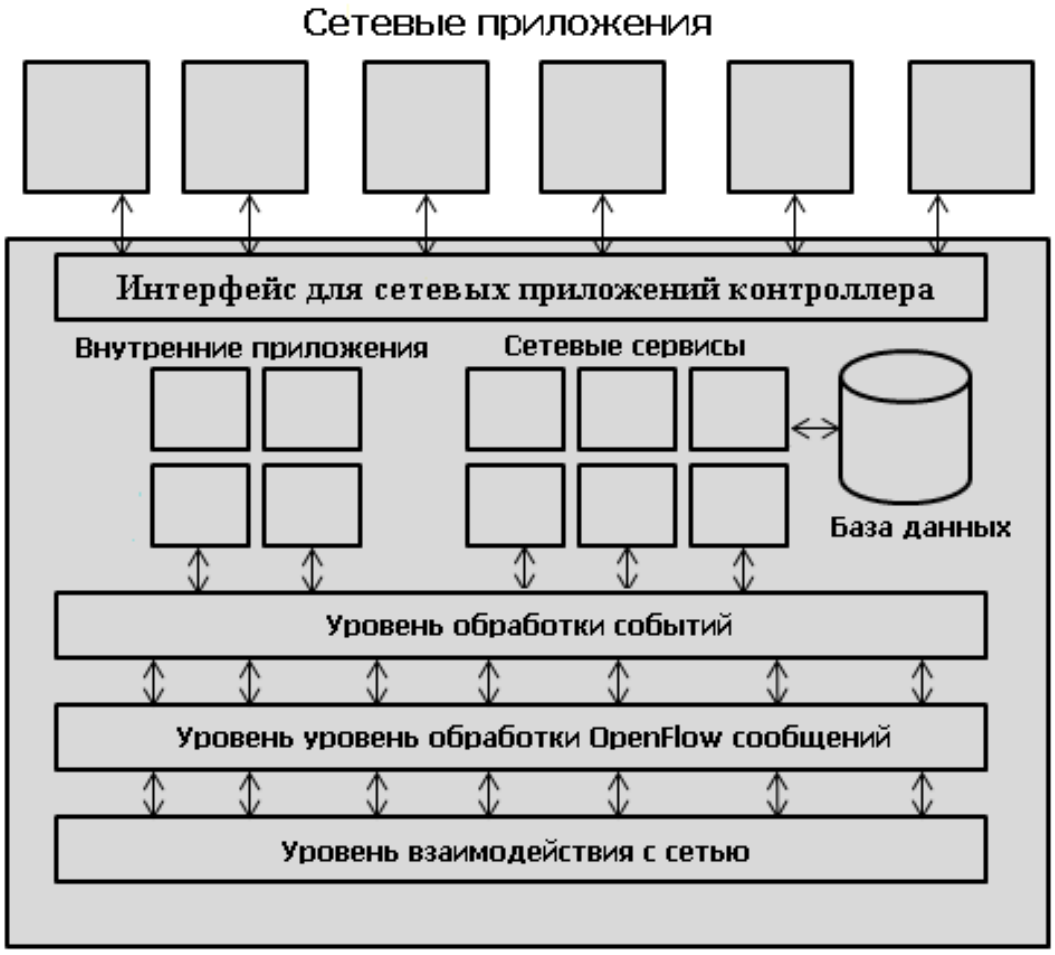

Рисунок 2.11 – Архитектура программной модели платформы управления

Уровень взаимодействия с сетью, уровень обработки Openflow сообщений и уровень обработки событий образуют ядро контроллера.

Платформа управления также включает в себя базу данных (NIB – Network Information Base) для хранения всей необходимой информации о топологии сети, состоянии сетевых элементов, журнализации сетевых событий, статистики.

Для управления программной платформой пользователю могут быть предоставлены следующие возможности: GUI, WEB-интерфейс, CLI.

Рассмотрим подробнее каждый из уровней программной модели платформы управления.

2.4 Описание базовых уровней

2.4.1 Уровень взаимодействия с сетью

Уровень взаимодействия с сетью предназначен для реализации функций управления соединениями с сетевыми устройствами, балансировкой соединений между ядрами процессора.

На этом уровне решаются две основные задачи, связанные с управлением соединениями:

– Чтение входящих OpenFlow пакетов из защищенных соединений контроллер-коммутатор. Для повышения скорости получения и обработки пакетов могут использоваться такие технологии, как netmap и Intel DPDK.

– Обработка входящих Openflow сообщений на основе использования многопоточности (multi-threading). Поддержка соединений и обработка пакетов распределяются по ядрам сервера, на котором работает контроллер.

2.4.2 Уровень обработки OpenFlow сообщений

Уровень обработки OpenFlow сообщений предназначен для кодирования и декодирования входящих и исходящих Openflow сообщений, проверки их корректности и порождения соответствующих событий об открытии нового потока (packet-in event), изменении конфигурации коммутатора (port-status event) и других. На этом уровне могут поддерживаться библиотеки обработки OpenFlow сообщений протоколов различных версий (1.0, 1.1, 1.2, 1.3) они могут быть реализованы на разных языках программирования (так в контроллере Trema поддерживаются библиотеки OpenFlow на языках C и Rybu, в контроллере SOX поддерживаются библиотеки OpenFlow с версиями протоколов 1.0 и 1.2). Соответственно, реализация данного уровня должна обладать расширяемостью – возможностью внедрения поддержки новых протоколов (не обязательно OpenFlow), соответствующих концепции ПКС.

2.4.3 Уровень обработки событий

Уровень обработки OpenFlow сообщений предназначен для реализации функций управления событиями: генерацией, тиражированием, распространением, фильтрацией, управлением приоритетами.

На данном уровне осуществляется обработка событий трех типов (по источнику их порождения):

– OpenFlow-событий, т.е. событий, порождаемых входящими OpenFlow сообщениями о событиях в сети;

– событий, порождаемых внутренними приложениями и сетевыми сервисами сетевой ОС;

– событий, порождаемых приложениями.

Приложение или сервис при инициализации имеет возможность подписаться на события заданного типа. На этом уровне устанавливается порядок распространения событий между приложениями. Например, возможна ситуация, когда событие распространяется только до некоторого установленного приложения, затем распространение события прекращается.

Производительность платформы управления достаточно сильно зависит от алгоритмов, реализованных на этом уровне.

2.4.4 Уровень сетевых сервисов и внутренних приложений

Уровень сетевых сервисов и внутренних приложений предназначен для обеспечения функциональных примитивов для работы с сетью. Уровень имеет модульную структуру, за счет которой облегчается процесс расширения и внесения изменений в сетевую ОС путем встраивания новых сервисов и приложений.

Под сервисом сетевой ОС понимается интерфейс, предоставляемый некоторой библиотекой, позволяющий экспортировать состояние некоторого объекта (например, объект – топология сети) и подписываться на события об изменении этого объекта. Потребители сервиса, могут запрашивать текущее состояние или

устанавливать новое состояние объекта, подписываться/отписываться на события, порождаемые этим сервисом. Допускается множество реализаций одного и того же сервиса.

Под внутренним приложением будем понимать приложение, реализующее некоторую функциональность для работы с сетью (маршрутизация, мониторинг состояния сети и другие) на основе использования некоторого набора сетевых сервисов. Таким образом, внутренние приложения экспортируют сервисы. При этом все приложения имеют минимальное количество зависимостей между собой, что упрощает разработку приложений.

К основным сервисам, реализованным в большинстве существующих сетевых ОС, например, относятся:

- сервис управления конкретным сетевым устройством;
- сервис определения топологии сети.

Внутренние приложения и сервисы запускаются вместе с ядром контроллера в его адресном пространстве, что позволяет существенно повысить их скорость работы, по сравнению с сетевыми приложениями, работающими на контроллере.

Внутренние приложения и сервисы не предполагают их изменений со стороны администратора сети и поставляются вместе с контроллером.

2.4.5 Интерфейс для сетевых приложений контроллера

Уровень интерфейса для сетевых приложений предназначен для обеспечения доступа к функциональности сервисов и внутренних приложений. Интерфейс (API) определяет многообразие сетевых приложений, которые могут быть написаны поверх него, а также позволяет минимизировать влияние исполняемых приложений друг на друга. В данной программной модели платформы управления предполагается реализация RestFull API, который позволяет реализовывать сетевые приложения на любом языке программирования.

2.4.6 Уровень сетевых приложений

Уровень сетевых приложений предназначен для запуска и исполнения пользовательских сетевых приложений, реализующих функции, необходимые администратору сети. Таким образом, набор сетевых приложений варьируется, в отличие от внутренних приложений и сервисов сетевой ОС, в зависимости от желаний и потребностей администратора сети.

По выполняемым функциям можно выделить следующие основные группы сетевых приложений:

– приложения маршрутизации трафика, поддерживающие механизмы реактивной и проактивной установки правил;

– приложения реализации политик;

– приложения поддержки работы с некоторыми сетевыми протоколами (ARP, DNS, DHCP);

– приложения мониторинга (состояния сети, сервера, приложений, статистики);

– приложения анализа и перераспределения трафика в сети;

– приложения поддержки виртуализации сетей.

К основным сетевым приложениям, реализованным в большинстве существующих сетевых ОС, относятся:

– Hub;

– L2 Learning Switch;

– L3 Learning Switch.

Сетевые приложения запускаются как отдельные процессы по отношению к ядру контроллера и могут также балансироваться по ядрам сервера. Сетевые приложения выполняются медленнее внутренних приложений и сервисов сетевой ОС.

2.5 Выводы

Таким образом, на основе исследования существующих сетевых ОС для ПКС, проведенном на первом этапе НИР, в данном разделе разработана модель платформы управления ПКС, приведены основные функции, которые должны быть реализованы в платформе, предложена многоуровневая архитектура платформы и сформирован базовый набор сервисов и приложений для сетевой ОС, которые должны входить в состав платформы управления ПКС.

3 Разработка программной документации в соответствии с п.7.3 ТЗ

3.1 Перечень, разработанной программной документации

В ходе выполнения НИР в соответствии с п.7.3 ТЗ была разработана в виде отдельных документов следующая программная документация отражающая экспериментальную разработанных программно-технических решений:

– «Программная реализация экспериментального сегмента ПКС. Описание применения».

– Документация по программным модулям экспериментальной программной реализации функциональных алгоритмов сегмента ПКС в составе:

1) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль обработки полученных пакетов. Описание программы».

2) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль обработки полученных пакетов. Текст программы»

3) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль построения топологии ПКС. Описание программы».

4) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль построения топологии ПКС. Текст программы».

5) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль построения маршрутов в сети. Описание программы».

6) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль построения маршрутов в сети. Текст программы»

7) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль формирования правил для сетевых элементов ПКС. Описание программы».

8) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль формирования правил для сетевых элементов ПКС. Текст программы»

9) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль, реализующий механизм подписки на события для других модулей СОС. Описание программы».

10) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль, реализующий механизм подписки на события для других модулей СОС. Текст программы»

11) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС Модуль распараллеливания обработки запросов по ядрам процессора. Описание программы».

12) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль распараллеливания обработки запросов по ядрам процессора. Текст программы»

13) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль контроля состояния сетевых ресурсов. Описание программы».

14) «Экспериментальная программная реализация функциональных алгоритмов сегмента ПКС. Модуль контроля состояния сетевых ресурсов. Текст программы»

3.2 Описание применения программной реализации

Данный документ содержит следующие сведения:

- описание назначения экспериментальной программной реализации;
- описание условий применения программной реализации, включающее: требования к техническим средствам, требования к программному обеспечению, требования к квалификации пользователей, а так же описание условий по обеспечению функционирования программы;
- описание всех основных задач, решаемых программной реализацией на данном этапе НИР;
- общий алгоритм работы с данной программной реализацией;
- перечень и краткие характеристики входных данных;
- перечень и краткие характеристики выходных данных.

Полное описание применения программной реализации экспериментального сегмента ПКС приведено в отдельном документе «Программная реализация экспериментального сегмента ПКС. Описание применения».

3.3 Документация на программные модули реализации

3.3.1 Перечень программных модулей

Программная реализация функциональных алгоритмов сегмента ПКС состоит из семи основных модулей:

- модуль обработки полученных пакетов;
- модуль построения топологии ПКС;
- модуль построения маршрутов в сети;
- модуль формирования правил для сетевых элементов ПКС;

– модуль, реализующий механизм подписки на события для других модулей СОС;

– модуль распараллеливания обработки запросов по ядрам процессора;

– модуль контроля состояния сетевых ресурсов.

Для каждого программного модуля были разработаны по два документа – описание программы и текст программы.

3.3.2 Документация с описанием программных модулей

Документы с описаниями программных модулей содержат:

– общие сведения о программном модуле, включая: обозначение и наименование программы; программное обеспечение, необходимое для выполнения программы; используемые языки программирования;

– описание функционального назначения модуля;

– описание логической структуры модуля, включая: алгоритмы, используемые методы, структуру программы, а так же связи с другими программными модулями;

– требования к техническим средствам, на которых разворачивается программный модуль;

– описание процедуры вызова модуля;

– описание процедуры загрузки модуля;

– описание входных и выходных данных.

3.3.3 Документация с текстом программных модулей

Документы с текстом программных модулей содержат непосредственно программный код соответствующих модулей.

4 Проведение экспериментальных исследований характеристик программной модели платформы управления и существующих сетевых ОС для ПКС в соответствии с Программой и методиками экспериментальных исследований

4.1 Возможность достижения показателей производительности, сравнимыми с показателями производительности традиционных (не относящихся к ПКС) сетей при меньших требованиях к ресурсам в сравнении с существующими решениями в области ПКС

4.1.1 Общие положения

Согласно пункту 1.2.4 промежуточного отчета о НИР [1] производительность ПКС сети напрямую зависит от производительности контроллера, который управляет сетью. Коммутаторы, используемые в сегменте ПКС, имеют характеристики, аналогичные характеристикам коммутаторов, использующихся в традиционных сетях. Таким образом, на производительность сети оказывает влияние только передача управляющей информации между контроллером и коммутаторами, которая происходит только при установлении новых потоков в сети. В дальнейшем коммутация пакетов одного потока производится без участия контроллера.

Поэтому тестирование на производительность ПКС сети заключается в тестировании производительности непосредственно контроллера ПКС.

4.1.2 Тестирование характеристик производительности

4.1.2.1 Краткое описание проведения тестирования

Тестирование текущей реализации алгоритмов (представленных в разделе 1 данного документа) проводилось при помощи средства Cbench согласно пункту 6.4 Программы и методики испытаний. Пропускная способность экспериментального сегмента ПКС сети оценивалась с учетом использования модуля распараллеливания обработки сообщений по ядрам процессора. Данный модуль использует

программное средство для балансировки нагрузки HAProxy для распределения обработки соединений с коммутаторами между процессами контроллера. Описание модуля распараллеливания обработки сообщений по ядрам процессора содержится в документе RU.00044977.00001-01 13 05.

Тестирование проводилось с задействованием 12 физических ядер процессора, каждое из которых предоставляет два логических ядра с использованием технологии Hyper-Threading. Таким образом, в ходе тестирования запускалось 24 процесса контроллера, между которыми при помощи системы балансировки нагрузки распределялась обработка запросов от коммутаторов.

4.1.2.2 Результаты тестирования на производительность

Тестирование с приведенными выше параметрами показало, что предложенная программная реализация ПКС сети имеет следующие характеристики производительности:

- количество запросов в секунду не менее 100000, график зависимости пропускной способности от числа логических ядер процессора представлен на рисунке 4.1. Количество процессов контроллера запускается равным количеству ядер процессора, таким образом, обеспечивается близкий к линейному рост пропускной способности контроллера;

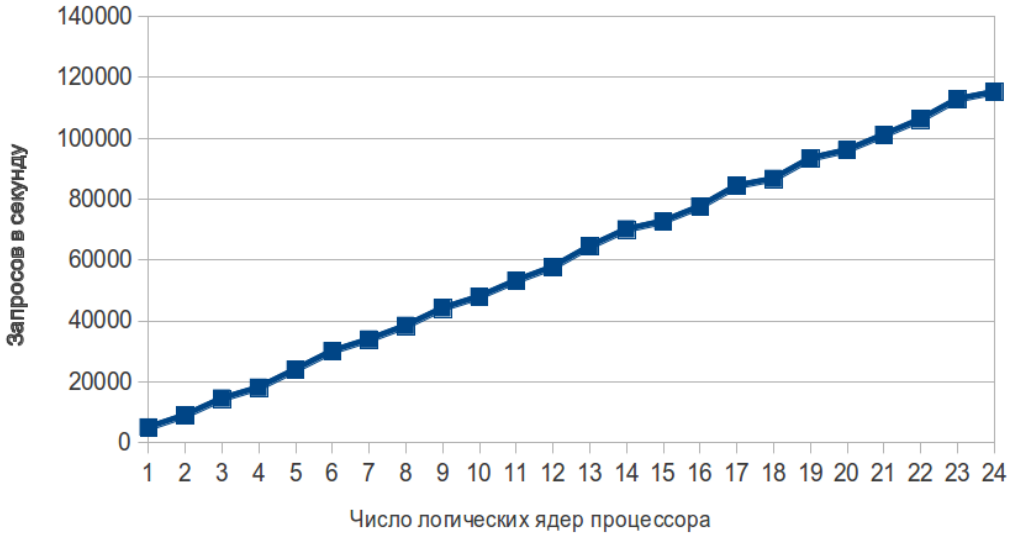

Рисунок 4.1. - Зависимость пропускной способности от числа логических ядер процессора (32 коммутатора, 10000 конечных узлов на коммутатор)

– минимальная задержка отклика не более 150 миллисекунд (в среднем 110-120 миллисекунд);

– поддержка не менее 64 коммутаторов, график зависимости пропускной способности от числа коммутаторов в сети представлен на рисунке 4.2. График демонстрирует рост пропускной способности до момента, пока количество коммутаторов не превысит количество доступных ядер процессора и, соответственно, количество запущенных процессов контроллера. Это обусловлено тем, что каждое TCP-соединение с коммутатором обрабатывается независимо одним процессом контроллера, следовательно, при числе коммутаторов, меньшим, чем число процессов, некоторые из ядер процессора остаются не загруженными. Увеличение числа коммутаторов позволяет равномерно распределить нагрузку между всеми ядрами процессора;

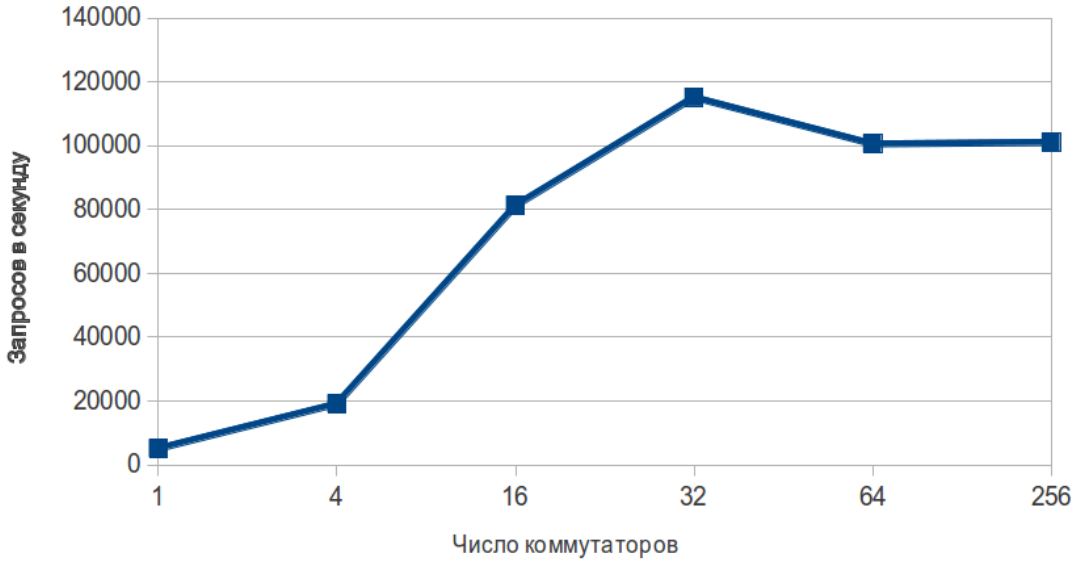

Рисунок 4.2 - Зависимость пропускной способности от числа коммутаторов (24 логических ядра процессора, 10000 конечных узлов на коммутатор)

– поддержка не менее 100000 конечных узлов, график зависимости пропускной способности от числа конечных узлов на один коммутатор представлен на рисунке 4.3. Тестирование показало слабую зависимость пропускной способности контроллера от числа конечных узлов в сети.

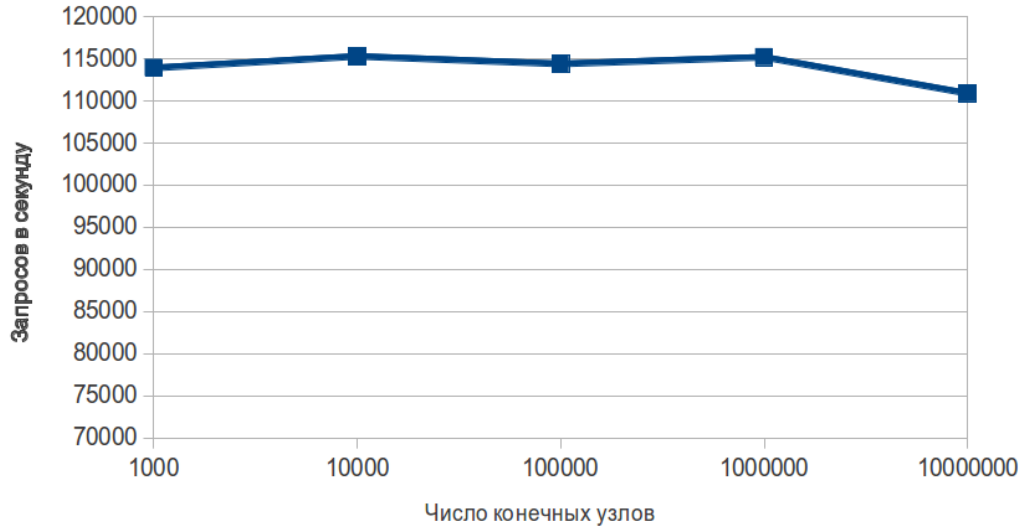

Рисунок 4.3 - Зависимость пропускной способности от числа конечных узлов на один коммутатор (24 логических ядра процессора, 32 коммутатора)

4.1.3 Вывод по результатам тестирования производительности

При выполнении требований к контроллеру, указанных в разделе 6.2 ТЗ, характеристики производительности сегмента ПКС, управляемого данным контроллером, сравнимы с показателями производительности традиционных (не относящихся к ПКС) сетей.

Но при этом, при использовании разработанных алгоритмов в проведении экспериментов требовалось меньше вычислительных ресурсов сервера, на котором работал контроллер, так как предложенные в ходе выполнения НИР реализации алгоритмов управления сетевыми ресурсами ориентированы на минимизацию и оптимизацию вычислений (см. раздел 1 данного документа).

Например, при тестировании сегмента ПКС сети совместно с разработанным модулем построения маршрутов в сети, использовался алгоритм динамического построения маршрутов, который позволил при изменениях в топологии сети перестроить лишь часть маршрутов, а не обновлять все существующие маршруты в сети.

Тестирование сегмента ПКС сети с использованием разработанного модуля кодирования/декодирования пакетов показало более устойчивую к ошибкам работоспособность, чем алгоритмы существующих на сегодняшний день контроллеров. Применение данного модуля позволило корректно обработать ошибки без закрытия соединения с коммутатором и установки повторного соединения.Возможность обеспечения совместимости с версией стандарта OpenFlow 1.0, поддерживаемой имеющимся оборудованием для ПКС

4.2.1 Общие положения

Предложенный в подразделе 1.3 алгоритм кодирования/декодирования пакетов реализует кодирование и декодирование всех сообщений протокола OpenFlow версии 1.0. Данный алгоритм реализован в программном модуле обработки полученных пакетов.

4.2.2 Проведение тестирования

Тестирование экспериментального сегмента ПКС сети на совместимость с протоколом OpenFlow 1.0 проводилось параллельно с тестированием работоспособности алгоритма кодирования/декодирования пакетов, реализованного в модуле обработки полученных пакетов.

Подробно алгоритм тестирования приведен ниже в п.4.5.3 данного документа. Тестирование проводилось с использованием среды mininet и виртуального коммутатора Open vSwitch, а также средств Cbench и HCProbe, взаимодействующих с контроллером по протоколу OpenFlow 1.0.

4.2.3 Выводы по результатам тестирования возможности совместимости с версией стандарта OpenFlow 1.0

Результаты тестирования демонстрируют, что все сообщения протокола OpenFlow 1.0 были обработаны контроллером правильно, некорректные сообщения были отброшены без нарушения функционирования контроллера. Таким образом, полученные результаты экспериментов позволяют сделать вывод о совместимости оборудования, использующегося в экспериментальном сегменте ПКС сети, с версией стандарта OpenFlow 1.0.

4.3 Выполнение технических характеристик, указанных в разделе 6.2 ТЗ

Приведенные в пп. 4.1.2.2 данного документа результаты тестирования характеристик производительности показывают, что разработанные алгоритмы при реализации с использованием компилируемого языка программирования вместо интерпретируемого языка программирования будут соответствовать заявленным в п. 6.2. ТЗ техническим характеристикам.

Текущая реализация алгоритмов использует контроллер POX и интерпретируемый язык программирования Python, который не поддерживает многопоточность (для реализации параллельной обработки пакетов требуется использование независимых процессов, что влечет за собой накладные расходы на

организацию межпроцессного взаимодействия). Разработанные алгоритмы могут быть с незначительными изменениями реализованы с использованием компилируемого языка программирования (например, C++), что с учетом поддержки этими языками многопоточности позволит получить пропускную способность более 500000 запросов в секунду.

4.4 Проверка работоспособности разработанных алгоритмов

4.4.1 Проверка работы алгоритма маршрутизации в рамках сегмента ПКС

4.4.1.1 Общий сценарий тестирования

Тестирование проводилось по следующему сценарию:

1) Запустить среду mininet с базовыми топологиями (линейная топология, дерево).

2) Запустить контроллер с модулем маршрутизации с различными стратегиями установки маршрута.

3) С помощью команды pingall сформировать потоки трафика с указанием узлов генерации и узлов назначения.

4) С помощью команды dpctl проверить содержимое таблиц правил коммутаторов сети на наличие там правил для соответствующего потока.

4.4.1.2 Тестирование алгоритмов на линейной топологии

Тестирование алгоритма с использованием стратегии установки правил с барьерами проводилось по описанному ниже сценарию:

1) Запуск контроллера:

```
./pox.py  openflow.discovery proute.dynamic_routing --
barrier=True
```

2) Запуск mininet:

```
sudo mn --topo=linear,5 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3 s4-eth3:s5-eth2
s5 lo:  s5-eth1:h5-eth0 s5-eth2:s4-eth3
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
```

4) Вывод команды dpctl dump-flows:

```
*** s1 ------------------------------------------------
------------------
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:1
*** s2 ------------------------------------------------
------------------
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:04), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
```

```
in_port(3),eth(src=ae:ad:56:3d:dd:f4,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:0, bytes:0, used:never,
actions:userspace(pid=4294952742,controller,length=65535)
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=5e:13:57:be:96:54,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:2, bytes:82, used:3.328s,
actions:userspace(pid=4294952743,controller,length=65535)
in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
```

```
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
*** s3 ----------------------------------------------------
------------------
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:04), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.2,op=1,sha=00:0
```

```
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
*** s4 ----------------------------------------------------
------------------
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:02), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
```

```
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:04), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(2),eth(src=f6:66:5a:4c:53:34,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:1, bytes:41, used:2.540s,
actions:userspace(pid=4294952709,controller,length=65535)
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=42:52:6d:a7:a7:46,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:2, bytes:82, used:0.028s,
actions:userspace(pid=4294952708,controller,length=65535)
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.4,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:04), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
```

```
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.3,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:03), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.4,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:04), packets:0, bytes:0,
used:never, actions:1
*** s5 ------------------------------------------------
------------------
in_port(2),eth(src=9a:1c:b3:74:9d:32,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:2, bytes:82, used:1.280s,
actions:userspace(pid=4294952692,controller,length=65535)
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
```

```
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.1,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:01), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.5,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:05), packets:0, bytes:0,
used:never, actions:1
```

Тестирование алгоритма с использованием стратегии установки правил для следующего коммутатора без барьеров проводилось по описанному ниже сценарию.

1) Запуск контроллера:

```
./pox.py openflow.discovery proute.dynamic_routing –
barrier=False –nex_hop=True
```

2) Запуск mininet:

```
sudo mn --topo=linear,5 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3 s4-eth3:s5-eth2
s5 lo:  s5-eth1:h5-eth0 s5-eth2:s4-eth3
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
```

4) Вывод команды dpctl dump-flows: совпадает с выводом для стратегии без барьеров.

Тестирование алгоритма с использованием стратегии установки правил без барьеров проводилось по описанному ниже сценарию.

1) Запуск контроллера:

```
./pox.py  openflow.discovery proute.dynamic_routing –
barrier=False --nex_hop=False
```

2) Запуск mininet:

```
sudo mn --topo=linear,5 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3 s4-eth3:s5-eth2
s5 lo:  s5-eth1:h5-eth0 s5-eth2:s4-eth3
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
```

4) Вывод команды dpctl dump-flows: совпадает с выводом для стратегии без барьеров.

4.4.1.3 Тестирование алгоритмов на топологии дерево

Тестирование алгоритма с использованием стратегии установки правил с барьерами проводилось по описанному ниже сценарию.

1) Запуск контроллера:

```
./pox.py  openflow.discovery proute.dynamic_routing --
barrier=True
```

2) Запуск mininet:

```
sudo mn --topo=tree,3 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
```

```
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
```

4) Вывод команды dpctl dump-flows:

```
*** s1 ------------------------------------------------
------------------
in_port(1),eth(src=ba:23:b2:38:fb:47,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:1, bytes:41, used:1.096s,
actions:userspace(pid=4294952466,controller,length=65535)
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=46:b6:69:27:7e:a9,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:3, bytes:123, used:2.972s,
actions:userspace(pid=4294952465,controller,length=65535)
in_port(1),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.7,op=2,sha=00:0
```

```
0:00:00:00:04,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
```

```
in_port(2),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:2
*** s2 ------------------------------------------------
------------------
```

```
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.7,op=2,sha=00:0
```

```
0:00:00:00:03,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=de:34:2a:5f:fe:df,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:1, bytes:41, used:1.368s,
actions:userspace(pid=4294952448,controller,length=65535)
in_port(2),eth(src=32:27:e4:d5:1b:75,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:0, bytes:0, used:never,
actions:userspace(pid=4294952447,controller,length=65535)
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.6,op=2,sha=00:0
```

```
0:00:00:00:01,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
*** s3 ------------------------------------------------
------------------
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=32:80:64:5b:2f:93,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:0, bytes:0, used:never,
actions:userspace(pid=4294952429,controller,length=65535)
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.2,op=1,sha=00:0
```

```
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(1),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
*** s4 ------------------------------------------------
------------------
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.7,op=2,sha=00:0
```

```
0:00:00:00:03,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=66:e0:11:4a:e9:de,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:2, bytes:82, used:0.840s,
actions:userspace(pid=4294952412,controller,length=65535)
in_port(1),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
*** s5 ------------------------------------------------
------------------
in_port(1),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.7,op=2,sha=00:0
```

```
0:00:00:00:06,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:01),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.1,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.6,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:08),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.8,op=2,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:08), packets:0, bytes:0,
used:never, actions:2
```

```
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.5,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:05,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:03),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.3,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=06:81:88:27:46:c2,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:1, bytes:41, used:1.904s,
actions:userspace(pid=4294952397,controller,length=65535)
in_port(2),eth(src=00:00:00:00:00:08,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.8,tip=10.0.0.6,op=1,sha=00:0
0:00:00:00:08,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.7,op=2,sha=00:0
```

```
0:00:00:00:04,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.6,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:06), packets:0, bytes:0,
used:never, actions:1
in_port(2),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:04),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.4,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(1),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:02),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.2,op=1,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:3
in_port(2),eth(src=d6:ce:dc:b5:55:74,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:0, bytes:0, used:never,
actions:userspace(pid=4294952396,controller,length=65535)
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:2
*** s6 ------------------------------------------------
------------------
in_port(3),eth(src=42:36:0b:03:cd:dc,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:0, bytes:0, used:never,
actions:userspace(pid=4294952378,controller,length=65535)
```

```
in_port(3),eth(src=00:00:00:00:00:08,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.8,tip=10.0.0.6,op=1,sha=00:0
0:00:00:00:08,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:06),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.6,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:2
in_port(2),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:08),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.8,op=2,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:08), packets:0, bytes:0,
used:never, actions:3
in_port(3),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:05),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.5,op=1,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:00), packets:0, bytes:0,
used:never, actions:1
in_port(1),eth(src=00:00:00:00:00:05,dst=00:00:00:00:00:0
in_port(3),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.2,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:02,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.1,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:01,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:04,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.4,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:04,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:1
```

```
in_port(1),eth(src=00:00:00:00:00:07,dst=00:00:00:00:00:08),
eth_type(0x0806),arp(sip=10.0.0.7,tip=10.0.0.8,op=2,sha=00:0
0:00:00:00:07,tha=00:00:00:00:00:08), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:1
in_port(3),eth(src=00:00:00:00:00:06,dst=00:00:00:00:00:08),
eth_type(0x0806),arp(sip=10.0.0.6,tip=10.0.0.8,op=2,sha=00:0
0:00:00:00:06,tha=00:00:00:00:00:08), packets:0, bytes:0,
used:never, actions:2
in_port(3),eth(src=5a:24:2b:ac:6f:47,dst=01:23:20:00:00:01),
eth_type(0x88cc), packets:1, bytes:41, used:2.172s,
actions:userspace(pid=4294952357,controller,length=65535)
in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:07),
eth_type(0x0806),arp(sip=10.0.0.3,tip=10.0.0.7,op=2,sha=00:0
0:00:00:00:03,tha=00:00:00:00:00:07), packets:0, bytes:0,
used:never, actions:1
```

Тестирование алгоритма с использованием стратегии установки правил без барьеров до следующего коммутатора проводилось по описанному ниже сценарию.

1) Запуск контроллера:

```
./pox.py  openflow.discovery proute.dynamic_routing --
barrier=False --next_hop=True
```

2) Запуск mininet:

```
sudo mn --topo=tree,3 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
```

4) Вывод команды dpctl dump-flows: совпадает с выводом для стратегии без барьеров.

Тестирование алгоритма с использованием стратегии установки правил без барьеров проводилось по описанному ниже сценарию.

1) Запуск контроллера:

```
./pox.py  openflow.discovery proute.dynamic_routing --
barrier=False --next_hop=False
```

2) Запуск mininet:

```
sudo mn --topo=tree,3 --mac --switch ovsk --controller
remote
```

3) Вывод команды net:

```
c0
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
```

4) Вывод команды dpctl dump-flows: совпадает с выводом для стратегии без барьеров.

4.4.1.4 Выводы по проверке алгоритма маршрутизации

На основании сравнения выводов команд net и dpctl средства mininet можно сделать вывод, что для всех рассмотренных топологий маршруты были построены правильно, так как правила, установленные контроллером на коммутаторах, обеспечивают маршрутизацию трафика в сети с заданной топологией.

Например, для линейной топологии для конечных узлов h1 и h2 был построен следующий маршрут (последний байт в MAC адресе совпадает с идентификатором узла):

h1 → s1-eth1 → s1-eth2 → s2-eth2 → s2-eth1 → h2

Этот маршрут соответствует топологии сети в mininet, в которой узел h1 соединен с интерфейсом eth1 коммутатора s1, интерфейс eth2 коммутатора s1 соединен с интерфейсом eth2 коммутатора s2, а интерфейс eth1 коммутатора s2 соединен с узлом h2 (это следует из вывода команды net средства mininet).

Следовательно, алгоритм маршрутизации работает корректно.

4.4.2 Проверка работы алгоритма построения топологии сети

4.4.2.1 Общие положения

Тестирование проводилось согласно сценарию, приведенному в пп. 6.5.1.3 ПМИ. Также проводилось тестирование работы модуля на топологии, содержащей коммутаторы, не поддерживающие OpenFlow.

Для моделирования топологий при помощи mininet использовались как стандартные топологии (линейная топология, топология «дерево»), так и дополнительно разработанные топологии («звезда», «толстое дерево», «полносвязный граф»).

Сравнивался вывод команды net из интерфейса mininet, содержащий информацию о созданной топологии, и вывода модуля discovery, содержащий информацию об обнаруженных соединениях между коммутаторами.

Запуск модуля топологии производился следующей командой:

```
./pox.py --no-cli openflow.discovery
```

4.4.2.2 Тестирование на топологии дерево

Тестирование алгоритма построения топологии проводилось по следующему алгоритму:

1) Запуск mininet:

```
sudo mn --controller=remote --ip=<controller_ip> --
topo=tree,depth=2,fanout=2
```

2) Вывод команды net:

```
s5 <-> s6-eth3 s7-eth3
s6 <-> h1-eth0 h2-eth0 s5-eth1
s7 <-> h3-eth0 h4-eth0 s5-eth2
```

3) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-05.2 -
> 00-00-00-00-00-07.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-05.1 -
> 00-00-00-00-00-06.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-06.3 -
> 00-00-00-00-00-05.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-07.3 -
> 00-00-00-00-00-05.2
```

4.4.2.3 Тестирование на топологии «толстое дерево»

Тестирование алгоритма построения топологии проводилось по следующему алгоритму:

1) Запуск mininet:

```
sudo mn --custom ~/mininet/custom/fat_tree.py --
topo=fat_tree --controller=remote --ip=192.168.56.1
```

2) Вывод команды net:

```
s1 <-> s2-eth1 s3-eth1 s4-eth1 s5-eth1
s2 <-> s1-eth1 s6-eth1 h7-eth0
s3 <-> s1-eth2 s6-eth2 h8-eth0
s4 <-> s1-eth3 s6-eth3 h9-eth0
s5 <-> s1-eth4 s6-eth4 h10-eth0
s6 <-> s2-eth2 s3-eth2 s4-eth2 s5-eth2
```

3) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.4 -
> 00-00-00-00-00-05.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.1 -
> 00-00-00-00-00-02.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -
> 00-00-00-00-00-06.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.1 -
> 00-00-00-00-00-01.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -
> 00-00-00-00-00-04.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-06.3 -
> 00-00-00-00-00-04.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-06.2 -
> 00-00-00-00-00-03.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-06.4 -
> 00-00-00-00-00-05.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-06.1 -
> 00-00-00-00-00-02.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.2 -
> 00-00-00-00-00-06.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.1 -
> 00-00-00-00-00-01.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-05.2 -
> 00-00-00-00-00-06.4
INFO:openflow.discovery:link detected: 00-00-00-00-00-05.1 -
> 00-00-00-00-00-01.4
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -
> 00-00-00-00-00-06.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.1 -
> 00-00-00-00-00-01.2
```

4.4.2.4 Тестирование на линейной топологии

Тестирование алгоритма построения топологии проводилось по следующему алгоритму:

1) Запуск mininet:

```
sudo mn --controller=remote --ip=192.168.56.1 --topo=linear
```

2) Вывод команды net:

```
s1 <-> h3-eth0 s2-eth2
s2 <-> h4-eth0 s1-eth2
```

3) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -
> 00-00-00-00-00-02.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -
> 00-00-00-00-00-01.2
```

4.4.2.5 Тестирование на топологии «звезда»

Тестирование алгоритма построения топологии проводилось по следующему алгоритму:

1) Запуск mininet:

```
sudo mn --custom ~/mininet/custom/star.py --topo star --
controller=remote —ip=192.168.56.1
```

2) Вывод команды net:

```
s1 <-> s2-eth1 s3-eth1 s4-eth1
s2 <-> s1-eth1 h5-eth0
s3 <-> s1-eth2 h6-eth0
s4 <-> s1-eth3 h7-eth0
```

3) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.1 -
> 00-00-00-00-00-01.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -
> 00-00-00-00-00-04.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -
> 00-00-00-00-00-03.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.1 -
> 00-00-00-00-00-02.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.1 -
> 00-00-00-00-00-01.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.1 -
> 00-00-00-00-00-01.2
```

4.4.2.6 Тестирование на топологии «полносвязный граф»

Тестирование алгоритма построения топологии проводилось по следующему алгоритму:

1) Запуск mininet:

```
sudo mn --custom ~/mininet/custom/mesh.py --topo mesh --
controller=remote --ip=192.168.56.1
```

2) Вывод команды net:

```
s1 <-> s2-eth1 s3-eth1 s4-eth1 h5-eth0
s2 <-> s1-eth1 s3-eth2 s4-eth2 h6-eth0
s3 <-> s1-eth2 s2-eth2 s4-eth3 h7-eth0
s4 <-> s1-eth3 s2-eth3 s3-eth3 h8-eth0
```

3) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -
> 00-00-00-00-00-04.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -
> 00-00-00-00-00-03.1
```

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.1 -
> 00-00-00-00-00-02.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.3 -
> 00-00-00-00-00-04.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -
> 00-00-00-00-00-03.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.1 -
> 00-00-00-00-00-01.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.3 -
> 00-00-00-00-00-04.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -
> 00-00-00-00-00-02.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.1 -
> 00-00-00-00-00-01.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.3 -
> 00-00-00-00-00-03.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.2 -
> 00-00-00-00-00-02.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-04.1 -
> 00-00-00-00-00-01.3
```

4.4.2.7 Тестирование модуля построения топологии на топологии, содержащей коммутаторы, не поддерживающие OpenFlow

Использование BDDP пакетов помимо стандартных LLDP для обнаружения соединений между коммутаторами позволяет обнаружить наличие соединения между двумя OpenFlow-коммутаторами, если они соединены через коммутаторы, не поддерживающие OpenFlow.

Топология для тестирования моделируется при помощи двух виртуальных машин (ВМ) VirtualBox, на которых запущено средство mininet. На каждой виртуальной машине создается два сетевых интерфейса, один из которых

используется для связи с контроллером, работающим на основной системе, а второй — для связи коммутаторов, запущенных на разных ВМ. ВМ соединены через виртуальный коммутатор системы VirtualBox.

Далее выполняется следующий сценарий:

1) На первой ВМ запускается mininet со стандартной топологией mininal (один коммутатор и два конечных узла):

```
sudo mn --controller=remote --ip=192.168.56.1 —topo=minimal
```

2) На второй ВМ запускается mininet с топологией, совпадающей со стандартной топологией minimal, однако с измененным значением dpid коммутатора, не равным 1 (так как контроллер идентифицирует различные коммутаторы по их dpid):

```
sudo mn --custom ~/mininet/custom/dpid.py --
controller=remote --ip=192.168.56.1 —topo=dpid
```

3) Один из сетевых интерфейсов каждой ВМ добавляется в список портов виртуального коммутатора mininet (используется коммутатор Open vSwitch):

```
sudo ovs-dpctl add-if dp0 <interface>
```

4) Вывод mininet> net для первой ВМ:

```
s1 <-> h2-eth0 h3-eth
```

5) Вывод mininet> net для второй ВМ:

```
s20 <-> h2-eth2 h3-eth
```

6) Вывод модуля discovery:

```
INFO:openflow.discovery:link detected: 00-00-00-00-00-14.3 -
> 00-00-00-00-00-01.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -
> 00-00-00-00-00-14.3
```

4.4.2.8 Выводы по тестированию алгоритмов построения топологии

Сравнение вывода модуля discovery с заданной в mininet топологией показывает, что алгоритмом были правильно построены топологии как для сети, целиком состоящей из коммутаторов с поддержкой OpenFlow, так и для сети, содержащей коммутаторы пез поддержки OpenFlow. Все соединения, присутствующие в топологии, были обнаружены алгоритмом.

Например, для топологии «звезда» алгоритмом были обнаружены соединения между коммутаторами s1 и s2, s1 и s3, s1 и s4 (последний байт в MAC адресе совпадает с идентификатором узла), что соответствует топологии «звезда» с коммутатором s1 в центре и совпадает с соединениями, отображенными командой net среды mininet.

Для двух коммутаторов s1 и s2, соединенных через коммутатор без поддуржки OpenFlow, было обнаружено единственное соединение, проходящее через этот коммутатор, что соответствует построенной топологии. Данное соединение не отображается командой net ни в одной из сред mininet, так как каждый коммутатор относится к своей среде, которая не имеет информации о втором коммутаторе.

Следовательно, алгоритм работает корректно.

4.4.3 Проверка работы алгоритма кодирования/декодирования пакетов

4.4.3.1 Общие положения

Тестирование проводилось согласно сценарию, приведенному в п. 6.5.3.2 ПМИ.

4.4.3.2 Программные средства, использующиеся для тестирования

В качестве генератора сообщений OpenFlow использовалось средство HCProbe. Для проверки правильности декодирования пакетов были разработаны следующие программные средства:

– Модуль контроллера dump.py, который подписывается на различные типы сообщений OpenFlow и выводит на экран структуры данных, полученные в результате декодирования пришедших сообщений.

– Файл packet_examples.hs с тестовым сценарием для HCProbe: отправка коммутатором сообщений OpenFlow различных типов.

– Файл version.hs с тестовым сценарием для HCProbe: отправка коммутатором сообщения OpenFlow с некорректной версией протокола в заголовке.

– Файл proto.hs с тестовым сценарием для HCProbe: отправка коммутатором Packet-In сообщения OpenFlow с некорректным заголовком инкапсулированного Ethernet-кадра.

4.4.3.3 Проведение тестирования

Контроллер запускался следующим образом:

```
./pox.py --no-cli samples.dump
```

Все корректно сформированные сообщения OpenFlow были корректно декодированы. Пример структуры данных, соответствующей полученному сообщению Flow-Removed:

```
header:
  version: 1
  type:    11 (OFPT_FLOW_REMOVED)
  length:  89
  xid:     0
match:
cookie: 0
priority: 0
reason: 0
duration_sec: 0
duration_nsec: 0
idle_timeout: 0
```

```
packet_count: 0
byte_count: 0
```

Сообщения с некорректной версией протокола сбрасываются контроллером.

Сообщения Packet-In с некорректно сформированным заголовком Ethernet-кадра не влияют на работу контроллера (обрабатываются согласно полученным после декодирования данным), однако выводится предупреждение о некорректных значениях полей заголовка Ethernet-кадра.

4.4.3.4 Выводы по тестированию работы алгоритма кодирования/ декодирования пакетов

На основе проведенного тестирования можно сделать вывод о корректности работы алгоритма кодирования/декодирования пакетов, так как все сообщения OpenFlow были правильно преобразованы во внутренне представление контроллера: все поля заголовков отправленных сообщений совпадают с полями заголовков сообщений во внутреннем представлении.

4.4.4 Проверка работы алгоритма для поддержки механизма подписки на события для модулей сетевой ОС

4.4.4.1 Общие положения

Тестирование проводилось согласно сценарию, приведенному в п. 6.5.4.2 ПМИ.

4.4.4.2 Программные средства, использующиеся для тестирования

Для проведения тестирования дополнительно к модулю dump.py были разработаны следующие вспомогательные модули контроллера:

– модуль dump_pi.py, который подписывается только на сообщения Packet-In от коммутатора с dpid равным 0 и выводит структуры, соответствующие полученным сообщениям, на экран;

– модуль dump_discovery.py, который подписывается на событие Link Event (изменение состояния соединения) модуля discovery.

В качестве генератора сообщений OpenFlow использовалось средство HCProbe с тестовым сценарием packet_examples.hs (отправка различных типов сообщений OpenFlow).

4.4.4.3 Тестирование на модели: нет подписчиков на события, вызванные получением OpenFlow сообщений

Для проведения тестирования была смоделирована ситуация, когда нет подписчиков на события, вызванные получением OpenFlow сообщений.

Для этого контроллер запускался без дополнительных модулей, подписанных на события:

```
./pox.py --no-cli
```

Результат тестирования – полученные контроллером сообщения не были обработаны.

4.4.4.4 Тестирование на модели: существует единственный подписчик-приложение, которое получает все события

Для проведения тестирования была смоделирована ситуация, когда существует единственный подписчик-приложение, которое получает все события.

Для этого контроллер запускался с модулем dump.py (подписка на все сообщения и вывод их на экран):

```
./pox.py --no-cli samples.dump
```

Результат тестирования – все сообщения были получены модулем dump.

4.4.4.5 Тестирование на модели: когда существует множество подписчиков на различные события

Для проведения тестирования была смоделирована ситуация, когда существует множество подписчиков на различные события.

Для этого контроллер запускался с модулями l2_learning.py и dump.py:

```
./pox.py --no-cli forwarding.l2_learning samples.dump
```

Результат тестирования – оба модуля подписаны на событие ConnectionUp и на сообщения Packet-In, модуль dump.py, помимо этого, подписан на другие типы сообщений OpenFlow. Событие ConnectionUp и сообщения PacketIn были обработаны обоими модулями, прочие сообщения OpenFlow — только модулем dump.py.

4.4.4.6 Тестирование на модели: когда подписчик получает только события определенного типа

Для проведения тестирования была смоделирована ситуация, когда подписчик получает только события определенного типа.

Для этого контроллер запускался с модулем dump_pi.py:

```
./pox --no-cli samples.dump_pi
```

Результат тестирования – данный модуль принимает только сообщения Packet-In от коммутатора с dpid равным 0. Все сообщения Packet-In, посланные коммутатором с dpid 0, были обработаны модулем. Сообщения других типов или от других коммутаторов проигнорированы.

4.4.4.7 Тестирование на модели: подписка приложения на события другого приложения

Для проведения тестирования была смоделирована ситуация, когда осуществляется подписка приложения на события другого приложения.

Для этого контроллер запускался с модулями discovery.py и dump_discovery.py:

```
./pox --no-cli openflow.discovery samples.dump_discovery
```

В модуле dump_discovery.py реализована подписка на событие LinkEvent модуля discovery.py и вывод отладочного сообщения при получении этого события.

Для генерации события LinkEvent контроллер тестировался со средством mininet, которое запускалось с топологией из двух коммутаторов и двух конечных узлов:

```
sudo mn --controller=remote --switch=user --topo=linear
```

Результат тестирования – при обнаружении модулем discovery.py соединений между коммутаторами модуль dump_discovery.py выводит сообщение о получении события LinkEvent.

4.4.4.8 Выводы по тестированию работы алгоритма для поддержки механизма подписки на события для модулей сетевой ОС

На основе проведенного тестирования можно сделать вывод о корректности работы алгоритма для поддержки механизма подписки на события для модулей сетевой ОС: все сообщения были получены только модулями, которые были подписаны на данный тип сообщений, при этом каждый модуль получил все сообщения требуемого типа.

4.5 Выводы по результатам экспериментальных исследований

Результаты НИР обеспечивают разработку программного обеспечения по модульному принципу с обеспечением:

– возможности масштабирования и эффективности поддержки топологии и контроля состояния сетевых ресурсов за счет использования динамического алгоритма построения маршрутов (см. результаты тестирования характеристик производительности в п. 4.1.2 данного документа);

– поддержки многопоточности на уровне не менее 20 потоков при использовании модуля распараллеливания обработки сообщений (см. результаты тестирования характеристик производительности в п. 4.1.2 данного документа);

– функций поддержки OpenFlow транзакций согласно версии протокола OpenFlow 1.0 (см. результаты тестирования алгоритмов кодирования/декодирования в пункте 4.4.3);

– поддержки совместимости с СОС NOX: разработанные алгоритмы могут быть реализованы как модули СОС NOX. Алгоритмы были реализованы в виде модулей СОС POX и продемонстрировали корректную работу совместно с этой СОС (см. п. 4.4.1 — 4.4.4). Так как интерфейс взаимодействия модулей СОС POX совместим с интерфейсом взаимодействия модулей СОС NOX, возможно использование модулей СОС NOX Classic совместно с разработанными модулями;

– поддержки совместимости с инструментальными средствами, разработанными в рамках Open Network Foundation, такими как среда mininet (см. результаты тестирования по п. 4.4.1 и 4.4.2, которое проводилось с использованием mininet) и средством тестирования контроллеров Cbench (см. результаты тестирования по п. 4.1.2 данного документа);

– функционирования в среде Linux и Windows:

1) Тестирование, которое проводилось по п. 3.2, а так же п. 4.4.1-4.4.4, проводилось с использованием серверов под управлением ОС Ubuntu 12.04 LTS.

2) Совместимость с Windows гарантируется за счет использования интерпретируемого языка Python, для которого существует интерпретатор python 2.7 под ОС Windows.

5 Разработка методических материалов по построению и настройке ПКС в российских университетах и научно-исследовательских организациях

5.1 Общие положения

Программно-конфигурируемые коммуникационные сети (ПКС, Software Defined Networks, SDN) — это новый тип коммуникационных сетей, в которых пространство управления отделено от пространства передачи данных. Логика их работы определяется программно, поверх сетевой операционной системы (СОС). Программа может иметь свой интерфейс взаимодействия (CLI, API) с внешними ресурсами, тем самым обеспечивая технические условия для согласованного состояния сетевой среды относительно меняющихся условий. Управление ПКС сетью может производиться как из одного центрального узла (контроллера), так и из множества таких узлов. При этом разделение ресурсов сети между узлами управления может производиться как на базе географического распределения, когда каждому коммутатору явно указывается управляющий им контроллер, так и на базе сетевых идентификаторов (номера портов, IP адреса, MAC-адреса, номер VLAN).

ПКС сеть является более согласованной по сравнению с "традиционной" компьютерной сетью в связи с тем, что контроллеры имеют полное видение сети. Применение принципов ПКС позволяет при работе с сетью перейти от примитивов, связанных с описанием сетевого взаимодействия оборудования, к примитивам, описывающим поведение сети с точки зрения пользовательских свойств, при этом приведение в соответствие этих точек зрения ложится на СОС.

Подобно традиционным телекоммуникационным компьютерным сетям, данные циркулируют по каналам связи между коммутаторами. Однако в ПКС данные управления передаются по специально выделенным каналам между специально выделенными устройствами управления - контроллерами и коммутаторами. Каналы управления (control plane) и каналы передачи данных (data plane) могут разделять одну физическую среду передачи. В этом случае говорят об In-Band управлении, для этого коммутаторы устанавливают специальные правила в

таблицы flow table. Но большинство типов современного оборудования не поддерживает в полной мере этот режим работы, в следствие чего сеть управления оказывается отделена от сети передачи данных физически. В этом случае говорят об Out-of-Band управлении.

Коммутаторы в ПКС находятся под управлением контроллеров, которые представляют из себя вычислительные устройства (серверы). На контроллерах выполняются прикладные программы, управляющие коммутацией и маршрутизацией пакетов в сети. Взаимодействие коммутаторов и контроллеров обеспечивает специальный сетевой протокол. В настоящее время существует стандартизированный протокол сетевого взаимодействия в сетях ПКС OpenFlow.

Протокол OpenFlow постоянно изменяется. На момент написания данного документа номер актуальной версии спецификации протокола OpenFlow 1.3.1. Учитывая достаточно большой лаг во времени между принятием спецификации протокола и выходом продуктов с его поддержкой, в современных условиях чаще всего можно встретить обордование с поддержкой версии OpenFlow 1.0. Из практических соображений при написании данных методических материалов авторы опираются на версию OpenFlow 1.0.

5.2 Возможные цели и задачи развёртывания сегмента ПКС

В настоящее время развёртывание сегмента ПКС выполняется:

– с образовательной целью (изучение сетевых технологий, знакомство с процессом конфигурирования сети, освоение методов наблюдения за работой сети и измерения пропускной способности;

– с целью исследования, разработки и отладки сетевых устройств, отработки методов конфигурирования сетей;

– с целью разработки приложений ПКС, оценки их работы, разработки и тестирования контроллеров для Openflow.

Данная методика не предполагает развертывание сети ПКС для начального знакомства и обучения базовым принципам ввиду наличия уже готового средства для этих целей – Mininet. Предполагается, что сегмент ПКС развертывается для тестирования оборудования, для применения в качестве составной части телекоммуникационной сети или для нагрузочного тестирования приложений.

5.3 Оборудование, необходимое для развертывания сегмента ПКС.

Для построения сегмента ПКС в университете или научно-исследовательской организации необходимо иметь хотя бы один коммутатор с поддержкой OpenFlow с характеристиками, покрывающими требования к сети, а также высокопроизводительный сервер для управления. Задачи, возлагаемые на сеть, могут быть самыми разными, зачастую бывает трудно рассчитать требуемую нагрузку заранее. Это в первую очередь связано с тем, что приложение, управляющее сетью может создавать неоправданно высокую нагрузку на сервер обрабатывая каждый пришедший пакет, а может лишь изредка проставлять статические правила.

5.4 Контроллеры ПКС

Рекомендуемое ПО контроллера ПКС представлено в таблице 5.1.

Таблица 5.1 – Контроллеры ПКС

Контроллер	Язык реализации	Открытость исходного кода	Разработчик
POX [2]	Python	Да	Nicira
NOX [3]	Python /C++	да	Nicira
MUL [4]	C	да	Kulcloid
Maestro [5]	Java	да	Rice University
Trema [6]	Ruby/C	да	NEC
Beacon [7]	Java	да	Stanford University

Продолжение таблицы 5.1

Контроллер	Язык реализации	Открытость исходного кода	Разработчик
Jaxon [8]	Java	да	Independent Developers
Floodlight [9]	Java	да	BigSwitch
SNAC [10]	C++	нет	Nicira
Ryu [11]	Pythom	да	NTT, OSRG group
NodeFlow[12]	JavaScript	да	Независимые разработчики
Ovs-controller [13]	C	да	Независимые разработчики
Flowvisor [14]	C	да	Stanforg/Nicira
RouteFlow [15]	C++	да	CPQD

Контроллеры имеют разную производительность [16], поддерживают различный набор уже созданных приложений, имеют различную стабильность и удобство разработки приложений.

5.5 Сетевое оборудование с поддержкой протокола OpenFlow

На текущий момент поддержка протокола OpenFlow реализована во многих продуктах, представленных в таблице 5.2.

Таблица 5.2– Сетевые коммутаторы с поддержкой протокола OpenFlow

Производитель	Модель коммутатора	Поддерживаемая версия протокола
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240 PF5820	v1.0
Pronto	3290 and 3780	v1.0
Juniper	Junos MX- Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.0

Программная реализация протокола OpenFlow входит в состав ПО, представленного в таблице 5.3.

Таблица 5.3 – Программно реализованные OpenFlow-коммутаторы

Коммутатор	Язык реализации	Краткое описание	Поддерживаемая версия протокола
Open vSwitch [17]	C/Python	Open source software switch that aims to implent a switch platform in virtualized server environments. Support standart management interfaces and enabls programmatic extension and control of the forwarding function. Can be ported info ASIC switches.	v1.0
Pantou/OpenWRT [18]	C	Turns a commercial wireless router or Access Point into an OpenFlow-enabled switch.	v1.0
Ofsoftswitch13 [19]	C/C++	OpenFlow 1.3 compativle user-space software switch implementation.	v1.3
Indigo [20]	C	Open source OpenFlow implementation thet runs on physical switches and uses the hardware features of Ethernet switch ASICs to run OpenFlow.	v1.0

Работа протокола OpenFlow даже одной и той же версии в различных коммутаторах может серьезно отличаться. Во-первых, в протоколе существует множество опциональных возможностей, во-вторую его реализация может во-многом различаться, а следовательно и такие параметры скорость работы и допустимое количество правил могут оказаться неприемлемыми для конкретной сети.

Коммутаторы бывают гибридные (hybrid) – с поддержкой традиционных сетевых протоколов, а также целиком полагающиеся на управление по OpenFlow. В гибридном коммутаторе имеется возможность обработки пакетов традиционным способом, тем самым контроллер ПКС получает дополнительную опцию выбора, а

коммутатор лишь воспринимает OpenFlow как еще один сетевой протокол. Гибридные коммутаторы позволяют реализовывать ПКС постепенно, без революционных изменений в сетевом управлении. С другой стороны, гибридные коммутаторы стоят значительно дороже и часть ресурсов (процессор, память, TCAM) вынуждены разделять между традиционной и ПКС обработкой.

Высокая скорость работы коммутатора возможна если коммутация пакетов будет происходить минуя центральный процессор коммутатора. Это возможно если FlowTable реализуется через TCAM, но в качестве недостатка данной схемы можно указать относительную дороговизну памяти TCAM. В программной реализации вся нагрузка ложится на центральный процессор, а значит и возможности таких коммутаторов ограничены производительностью процессора. Например, в реализации для OpenWRT количество правил ограничено до 100 штук, но даже и в этом случае (проверено на маршрутизаторах TP-Link TL-WR1043ND) даже при небольшом трафике задержка коммутации будет достигать десятков миллисекунд.

В качестве примера будем рассматривать гибридный коммутатор NEC PF5240, который поддерживает одновременно до 160 тыс. правил коммутации OpenFlow, способен буферизировать пакеты, отправляя на контроллер лишь заголовки. Коммутатор имеет 48 портов на 1Gbit и 4 оптических порта на 10Gbit.

При реализации беспроводной сети ПКС не рекомендуется применение Pantou/OpenWRT на точках доступа в силу ограниченности программных ресурсов последних. Сеть будет надежнее и быстрее, если точки доступа будут каким-либо образом (например, через VLAN) туннелировать трафик до ПКС коммутаторов, а вся логика ПКС будет реализована на них.

5.6 Серверное оборудование для построения сегмента ПКС сети

В зависимости от частоты проводимых в сети изменений, логики приложений контроллера, сетевой нагрузки, таймеров устаревания правил и пр. факторов, а также выбора контроллера из списка, необходимо выбирать контроллер управления сетью. Во многом в данном вопросе следует полагаться на рекомендации

разработчиков контроллеров?. Что касается сетевой подсистемы сервера, то от ее производительности во многом будет зависеть время реакции сети. С точки зрения масштабируемости сети и необходимости соединения контроллера со всеми управляемыми коммутаторами, лучшим выбором будет подключение контроллера к каждому коммутатору индивидуально. То есть количество сетевых интерфейсов в идеале должно быть больше или равно числу управляемых коммутаторов.

Для исследований в сети ПКС дополнительно рекомендуется установить прозрачный proxy для ПКС сетей FlowVisor, который позволяет разделять одну физическую среду на несколько независимых виртуальных. Тем самым ПКС сеть может использоваться различными группами разработки независимо, надежно разделяя трафик между ними и управляющими контроллерами этих групп.

Рекомендуемые системные требования для FlowVisor: 4 процессорных ядра; минимум 4 гигабайта оперативной памяти.

Строго не рекомендуется установка FlowVisor на виртуальную машину ввиду высокой нагрузки на сетевой стек, а также на дисковый массив.

В зависимости от используемых приложений, возложенных на сегмент ПКС задач и сетевой нагрузки стоит выделить ресурсы для контроллеров. При распределении ресурсов следует руководствоваться документацией по контроллеру.

5.7 Дополнительное оборудование для построения сегмента ПКС сети

В случае, если невозможно подключить к контроллеру (или FlowVisor) необходимое количество коммутаторов напрямую, можно подключить их через выделенный коммутатор для Control Plane с достаточной пропускной способностью. При этом сеть ControlPlane строится на принципах традиционной сети. Однако при таком подключении необходимо проверить, что коммутаторы ПКС не допускают объединения ControlPlane и DataPlane ни в одном из режимов работы.

Для развертывания беспроводной сети ПКС понадобятся точки доступа и/или micro-PC с беспроводными модулями. В отличие от точек доступа, современные micro-PC ограниченно могут использоваться в качестве ПКС коммутаторов

(требуется установка OpenVSwitch), если у сети нет высоких требований на производительность беспроводного сегмента и количество правил на точках не велико (<1000).

5.8 Пример ПКС сети

5.8.1 Краткое описание сегмента ПКС сети

Для примера построим следующую ПКС сеть, представленную на рисунке 5.1, где жирными линиями показаны соединения data plane, а тонкими линиями – control plane. Сервер с FlowVisor соединяется напрямую с OpenFlow коммутаторами, которые в свою очередь объединены в единую сеть. К коммутаторам подключены WiFi точки доступа, к которым, в свою очередь, подключаются абоненты ПКС сети. В примере мы не будем углубляться в настройку точки доступа, в частности изоляцию на уровне WiFi и безопасность. Будем считать, что с точки зрения нашей сети точки доступа являются всего лишь прозрачным мостом между проводной и беспроводной сетью. К FlowVisor прямым проводом подключен контроллер сети, который управляет выделенным сегментом. Если контроллеров несколько, можно подключать их аналогично первому. Если ресурсов одного FlowVisor недостаточно, можно использовать сразу несколько, при необходимости строя иерархию из таких серверов.

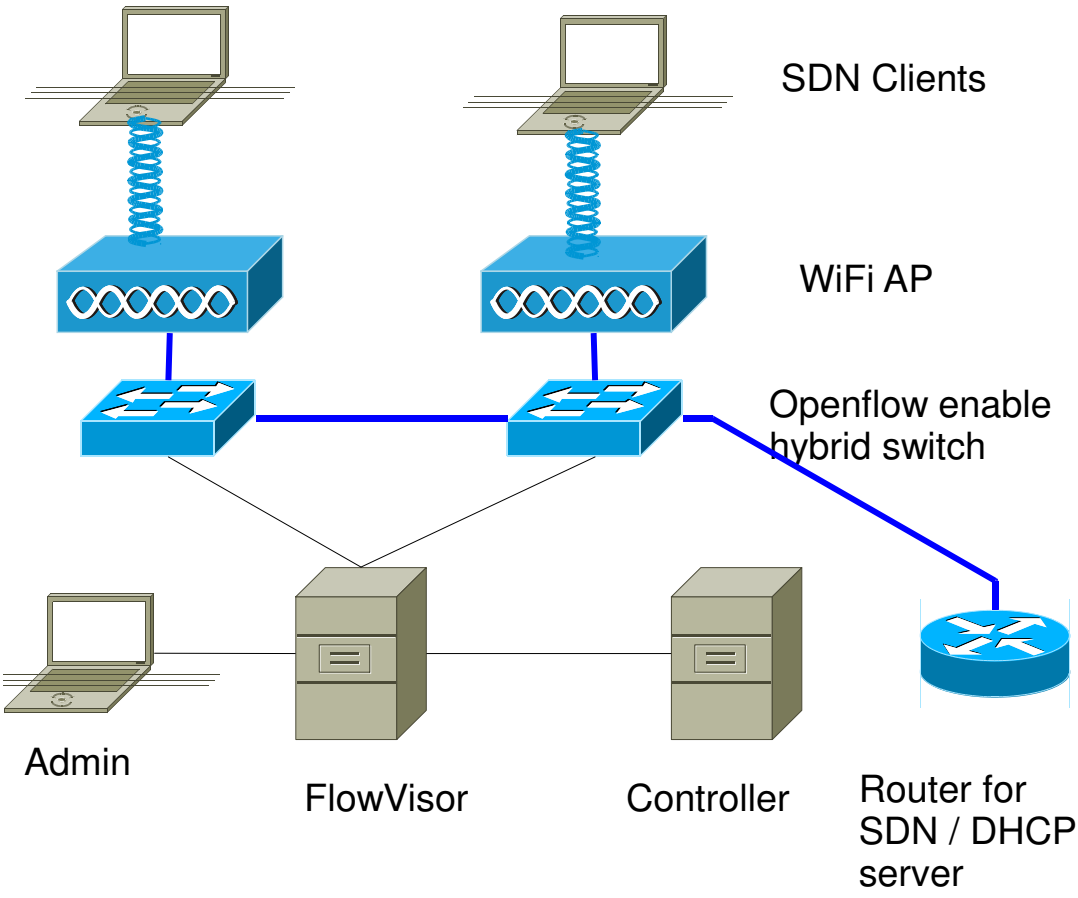

Рисунок 5.12 – Структура создаваемой ПКС-сети

Беспроводная сеть должна иметь DHCP сервер и желательно маршрутизатор (показан справа). В качестве такого устройства может быть любой традиционный маршрутизатор.

5.8.2 Установка и первичная настройка компонентов ПКС сети

Разрежем сеть 192.168.254.0/24 на сети управления:

- для 1-го коммутатора назначим адрес/маску –192.168.254.2/30;
- для 2-ого – 192.168.254.6/30;
- для контроллера – 192.168.254.10/30;
- для администрирования –192.168.254.14/30.

Для Flowvisor на 4 соответствующих интерфейса назначим:

- 192.168.254.1/30;

– 192.168.254.5/30;

– 192.168.254.9/30;

– 192.168.254.13/30.

5.8.3 Настройка коммутаторов

Для коммутатора в ControlPlane специальная настройка не нужна, - подойдет даже любой ненастраиваемый коммутатор достаточной производительности (порты 1Gbit или выше). Настройка коммутатора OpenFlow (в нашем примере это NEC PF5240) происходит по следующему алгоритму:

1) Войти в режим конфигурирования коммутатора:

```
SDN_switch#configure terminal
```

2) Для VLAN для ControlPlane:

```
SDN_switch(config)#vlan 1
SDN_switch(config-vlan)#name "Control plane"
SDN_switch(config-vlan)#exit
```

3) Задействовать VLAN для DataPlane:

```
SDN_switch(config)#vlan 2
SDN_switch(config-vlan)#name "Data plane"
SDN_switch(config-vlan)#exit
```

4) Настроить порты: 1-й для управления, 2-48-й для ПКС:

```
SDN_switch(config)#interface gigabitethernet 0/1
SDN_switch(config-if)#switchport mode access
SDN_switch(config-if)#switchport access vlan 1
SDN_switch(config-if)#exit
SDN_switch(config)#interface range gigabitethernet 0/2-48
SDN_switch(config-if)#switchport mode access
SDN_switch(config-if)#switchport access vlan 2
SDN_switch(config-if)#exit
```

5) Выключить STP:

```
SDN_switch(config)#spanning tree disable
```

6) Назначить параметры сети:

```
SDN_switch(config)#interface vlan 1
SDN_switch(config-if)#ip address 192.168.254.2
255.255.255.252
SDN_switch(config-if)#exit
```

7) Включить протокол OpenFlow и назначить IP адрес контроллера:

```
SDN_switch(config)#openflow openflow-id 1 virtual-switch
SDN_switch(config-of)#controller controller-name team1-
controller 100 192.168.254.1 port 6633 tcp
SDN_switch(config-of)#dpid 0000000000aaaa01
SDN_switch(config-of)#openflow-vlan 2
SDN_switch(config-of)#emergency-mode disable
SDN_switch(config-of)#mac-learning disable
SDN_switch(config-of)#enable
SDN_switch(config-of)#exit
```

8) Аналогично поступить и со вторым коммутатором:

```
SDN_switch(config)#interface vlan 1
SDN_switch(config-if)#ip address 192.168.254.6
255.255.255.252
SDN_switch(config-if)#exit
SDN_switch(config)#openflow openflow-id 1 virtual-switch
SDN_switch(config-of)#controller controller-name team1-
controller 100 192.168.254.5 port 6633 tcp
SDN_switch(config-of)#dpid 0000000000aaaa02
SDN_switch(config-of)#openflow-vlan 2
SDN_switch(config-of)#emergency-mode disable
SDN_switch(config-of)#mac-learning disable
```

```
SDN_switch(config-of)#enable
SDN_switch(config-of)#exit
```

Остальные параметры у коммутаторов в точности совпадают.

5.8.4 Настройка сервера управления ПКС

Когда есть необходимость в независимом управлении одной ПКС сетью с двух независимых контроллеров и при этом нет возможности физического разделения сети на два сегмента, лучше всего воспользоваться OpenFlow proxy FlowVisor. Он позволяет разрезать ПКС сеть на относительно независимые логические сегменты (slices) по заданному набору признаков. Принадлежность пакета к Slice определяется по сопоставлению его заголовка таблице FlowSpace. Каждый сегмент должен быть однозначно описан, например, по списку портов, номерам vlan, MAC адресам, IP адресам. Чтобы правила одного сегмента не влияли на работу другого, FlowVisor следит за корректностью их установки, по возможности модифицирует, но как правило отказывает в установке правил, написанных с нарушениями политик разделения сегментов. Это приводит к тому, что многие ПКС приложения не могут работать в сети, разделенной тем или иным способом. В частности, если контроллер посылает запрос flowmod, который не содержит в маске в явном виде идентификатора сегмента для FlowVisor, такой запрос может быть отклонен.

FlowVisor вносит дополнительную задержку в обработку потоков, а значит, при его применении стоит обращать внимание на технические характеристики сервера, на котором тот развернут. Благодаря тому, что его работа прозрачна для работы остального оборудования ПКС, FlowVisor можно ставить на несколько серверов и даже строить из них иерархию, как показано на рисунке 5.2.

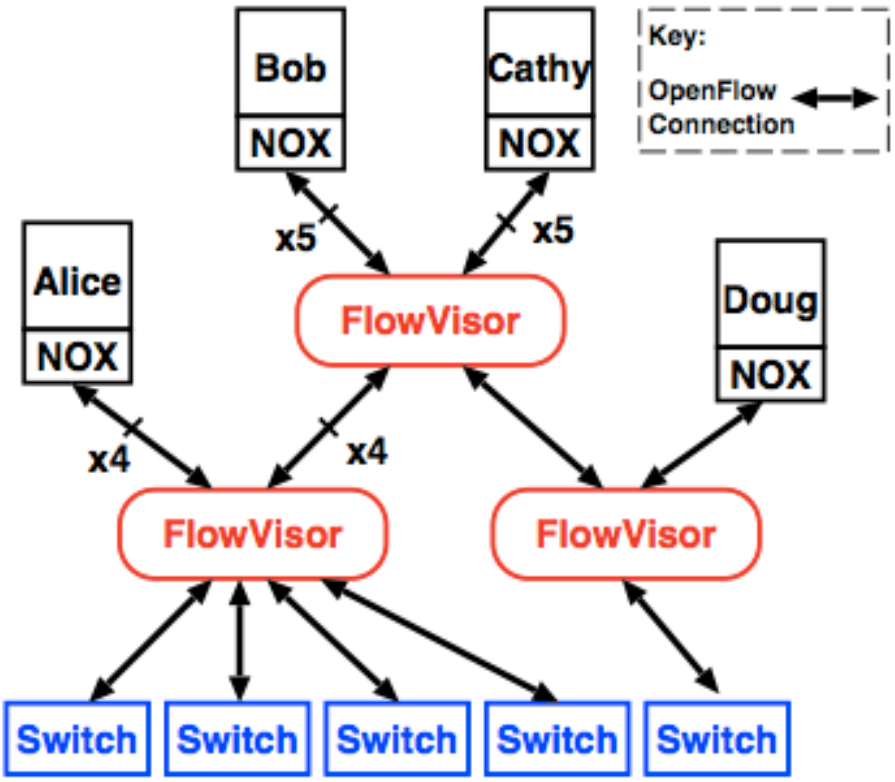

Рисунок 5.13 – Пример иерархии FlowVisor'ов

При использовании FlowVisor различные контроллеры могут сосуществовать в одной сети, функционально дополняя друг друга. Это особенно важно в условиях различной функциональной направленности контроллеров, их различных возможностей и набора приложений.

Установка FlowVisor возможна на большинстве систем Linux. Здесь для примера описана установка на сервере с уже предварительно установленной операционной системой Linux Ubuntu Server 12.04.2 LTS 64bit. На сервере должны быть настроены сетевые адаптеры для связи с коммутаторами, контроллерами и АРМ администратора. Сервер также должен иметь выход в Интернет для установки пакетов и обновления компонент программного обеспечения.

Пример настройки сетевого интерфейса для связи с 1-м коммутатором:

```
$ sudo vi /etc/network/interfaces
iface eth0 inet static
address 192.168.254.1
```

```
netmask 255.255.255.252
auto eth0
```

Для установки и работы FlowVisor необходим пакет Oracle-Java6-JDK. FlowVisor может работать и на других JDK, но на них тестирование не проводилось и производительность не гарантируется. Выбор 64-разрядной ОС обусловлен необходимостью выделения большого объема памяти, а также тем, что для современного сервера это уже стандарт де-факто.

5.8.5 Установка и настройка FlowVisor

Установка и настройка FlowVisor осуществляется по следующему алгоритму:

1) Для начала необходимо подготовить систему и установить необходимые пакеты:

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository "deb
http://archive.ubuntu.com/ubuntu hardy main multiverse"
$ sudo add-apt-repository "deb
http://archive.ubuntu.com/ubuntu hardy-updates main
multiverse"
$ sudo add-apt-repository "deb http://archive.canonical.com/
lucid partner"
$ sudo apt-get update
$ sudo apt-get install ant sun-java6-jdk
$ sudo apt-get install git build-essential eclipse
```

2) По соображениям безопасности создать пользователя, от имени которого будет работать FlowVisor:

```
$ sudo useradd flowvisor
```

3) Установим FlowVisor:

```
$ git clone git://github.com/OPENNETWORKINGLAB/flowvisor.git
$ cd flowvisor
```

```
$ make
$ sudo make fvuser=flowvisor fvgroup=flowvisor install
```

4) Запустить FlowVisor:

```
$ sudo –u flowvisor flowvisor
```

или так

```
$ sudo /etc/init.d/flowvisor start
```

5) Управление FlowVisor производится из CLI или через встроенный http-сервер посредством формирования запросов. Данные сохраняются во внутренней базе данных, поэтому при старте системы могут быть потеряны. Поэтому после сделанных изменений желательно сохранить БД в стартовый скрипт:

```
$ sudo fvctl save-config /etc/flowvisor/config.json
```

Как показано на рисунке 5.3, FlowVisor работает следующим образом: он получает сообщения OpenFlow от контроллеров (1) и, используя правила flow-space (2), прозрачно перезаписывает сообщение для того, чтобы контроллер мог управлять сетью только в рамках своего Slice (3). Сообщения от коммутаторов транслируются только на контроллеры, под чью политику flow-space они попадают (4).

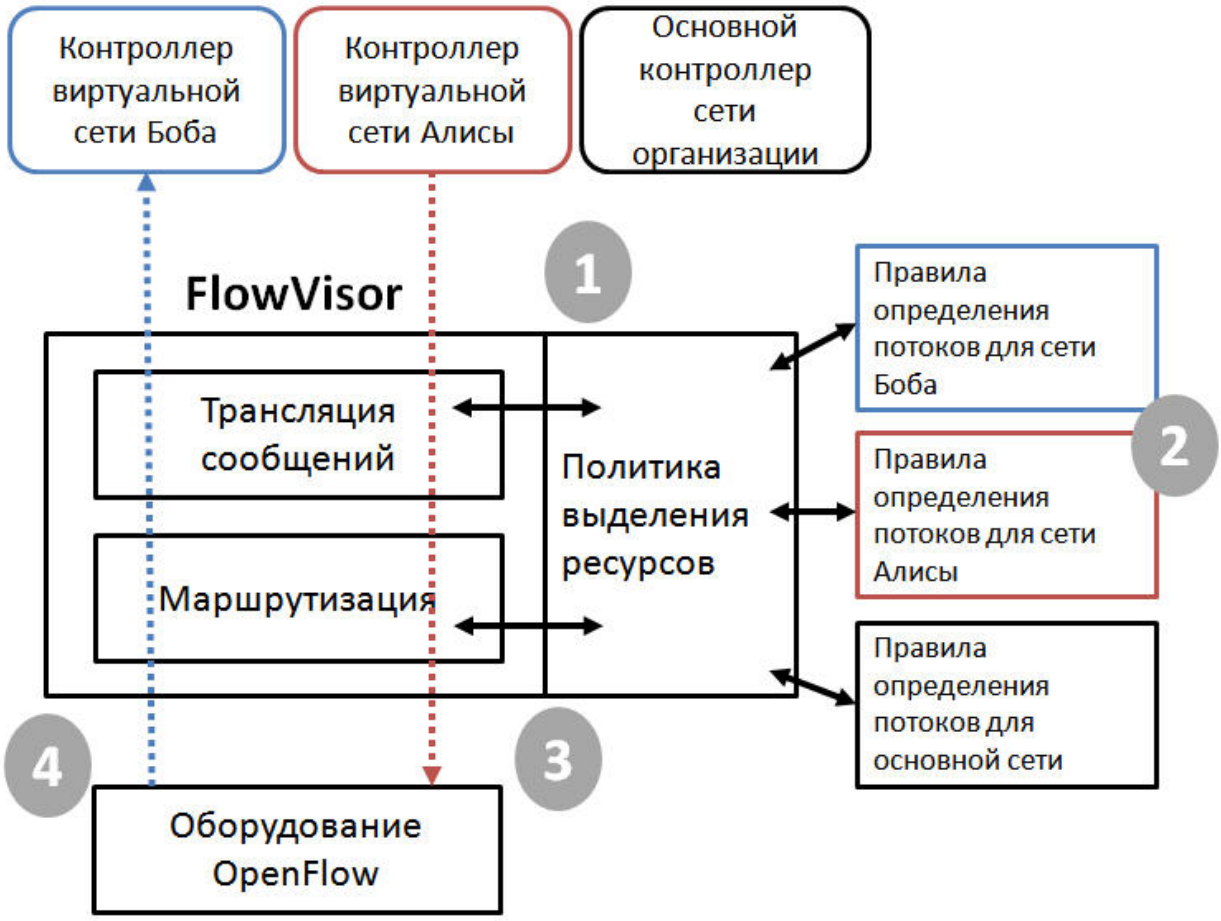

Рисунок 5.14 – Схема работы FlowVisor

6) Запустить FlowVisor, после чего создать Slice для тестового контроллера:

```
$ sudo /etc/init.d/flowvisor start
$fvctl add-slice test_sl tcp: 192.168.254.10:6633 your@email.ru
```

Будет создан Slice для контроллера. На указанный адрес электронной почты при соответствующей настройке FlowVisor будет слать сообщения.

7) Далее необходимо добавить маски правил для установления соответствия сообщений с контроллером. Здесь нужно определиться, какие параметры являются уникальными для этого Slice. Например выбрать MAC-адреса:

```
$fvctl add-flowspace name1 any 1 dl_src=00:00:00:00:00:01
test_sl=4
```

```
$fvctl add-flowspace name1 any 1 dl_src=00:00:00:00:00:02
test_sl=4
```

Таким образом, в Slice были добавлены два клиентских устройства с простыми MAC адресами 00:00:00:00:00:01 и 00:00:00:00:00:02 с приоритетом 1 и работающих на любом коммутаторе.

8) Для ограничения списка коммутаторов, можно ввести следующие записи:

```
$fvctl add-flowspace name1 00:00:00:00:00:aa:aa:01 1 nw_src=10.0.0.1 test_sl=4
$fvctl add-flowspace name1 00:00:00:00:00:aa:aa:01 1
nw_src=10.0.0.2 test_sl=4
```

Таким образом, были выделены ресурсы уже по IP адресам и только для 1-го коммутатора.

5.8.6 Установка и запуск контроллера

5.8.6.1 Предварительная подготовка сервер, на который устанавливая контроллер

Контроллер устанавливается на выделенный сервер, на который предварительно уже была установлена Linux Ubuntu Server 12.04.2 LTS 64bit и был корректно настроен сетевой адаптер для связи с сервером FlowVisor. Сервер имеет доступ в интернет.

5.8.6.2 Примеры вариантов установки контроллера на сервер

Далее приводится один из возможных алгоритмов установки контроллера ПКС:

1) Для установки контроллера запустить готовый скрипт установки:

```
$ git clone https://github.com/ARCCN/ctltest
$ cd ctltest
```

2) Для установки floodlight выполнить следующие команды:

```
$ cd 03_FloodLight
```

```
$ sudo ./install.sh
$ ./start.sh 1
```

Будет запущено стандартное простое приложение, эмулирующее работу L2 коммутации.

Стандартная установка контроллера осуществляется выполнением следующего набора команд:

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ git checkout stable
$ ant
```

5.8.6.3 Запуск контроллера

Для запуска контроллера необходимо выполнить следующую команду:

```
$ sudo java –jar target/floodlight.jar –cf
src/main/resources/floodlightdefault.properties
```

Через несколько секунд после запуска контроллера, коммутатор должен связаться с ним через FlowVisor. В случае успеха в консоли контроллера появится сообщение:

```
14:30:09.324 [New I/O server worker #2-1] DEBUG
n.f.core.internal.OFChannelHandler - Received role reply
message from OFSwitchBase [/192.168.254.9:48029
DPID[00:00:00:00:00:aa:aa:01]], setting role to MASTER
14:30:09.324 [New I/O server worker #2-1] DEBUG
n.f.core.internal.OFChannelHandler - Switch OFSwitchBase
[/192.168.254.9:48029 DPID[00:00:00:00:00:aa:aa:01]]
activated. Role is now MASTER
14:30:09.328 [New I/O server worker #2-1] INFO
n.f.core.OFSwitchBase - Clearing all flows on switch
```

```
OFSwitchBase [/192.168.254.9:48029
DPID[00:00:00:00:00:aa:aa:01]]
14:30:09.331 [main] WARN  n.f.c.i.C.syslog.notification -
Switch 00:00:00:00:00:aa:aa:01 connected.
```

5.8.7 Проверка работоспособности и поиск источника неисправности

Если соединение в течении таймаута подключения не произошло (обычно не более 1-й минуты), необходимо выяснить причину отказа.

Во-первых, необходимо проверить, что сеть управления собрана и настроена корректно, то есть проверить доступность коммутатора с сервера FlowVisor командой ping:

```
ping 192.168.254.2
PING 192.168.254.2 (192.168.254.2) 56(84) bytes of data.
64 bytes from 192.168.254.2: icmp_req=1 ttl=64 time=0.134 ms
64 bytes from 192.168.254.2: icmp_req=2 ttl=64 time=0.042 ms
64 bytes from 192.168.254.2: icmp_req=3 ttl=64 time=0.029 ms
64 bytes from 192.168.254.2: icmp_req=4 ttl=64 time=0.032 ms
```

Аналогично контроллер должен быть доступен с сервера FlowVisor.

Далее стоит посмотреть с помощью команды netstat, насколько успешно стартовал контроллер и FlowVisor (проверка ожидающих соединения портов), выполнив следующие действия:

1) Определить факт наличия запущенного FlowVisor и контроллера на стандартном tcp порту, запустив команду netstat со следующими атрибутами:

```
netstat –at | grep 6633
```

2) Определить наличие / отсутствие трафика на между FlowVisor и коммутатором, а также FlowVisor и контроллером соответственно, запустив команду netstat со следующими атрибутами:

```
sudo tcpdump src port 6633 or dst port 6633 –i eth0
sudo tcpdump src port 6633 or dst port 6633 –i eth2
```

Убедившись, что коммутатор успешно подключился к контроллеру, можно проверить работу ПКС на простом приложении.

Для следует подключить два сетевых устройства, например, ноутбука, включенные в порты DataPlane с настроенными сетевыми адаптерами из одной подсети (например, 10.10.0.1 и 10.10.0.2 с маской 255.255.0.0), выполнив следующие действия:

1) Внести адреса в наш Slice:

```
fvctl add-flowspace test_flowspace any 1 nw_src=10.0.0.1
test_sl=4
fvctl add-flowspace test_flowspace any 1 nw_src=10.0.0.2
test_sl=4
```

2) С первого ноутбука попробовать послать ICMP запросы на второй:

```
Ping 10.10.0.2
PING 10.10.0.2 (10.10.0.2) 56(84) bytes of data.
64 bytes from 10.10.0.2: icmp_req=1 ttl=64 time=234 ms
64 bytes from 10.10.0.2: icmp_req=2 ttl=64 time=0.170 ms
64 bytes from 10.10.0.2: icmp_req=3 ttl=64 time=0.042 ms
64 bytes from 10.10.0.2: icmp_req=4 ttl=64 time=0.044 ms
```

Большая задержка первого пакета связана с обработкой его на контроллере, при этом последующие пакеты пойдут между коммутаторами напрямую, без запроса контроллера.

5.9 Мониторинг работы приложений сегмента ПКС и их отладка

5.9.1 Запись Openflow пакетов в файл

При создании приложений, отладке стендов, тестировании оборудования полезно отследить Openflow сообщения. Для этого может понадобиться собрать

трафик между коммутатором и FlowVisor, а также между контроллером и FlowVisor в файл:

```
$sudo tcpdump src port 6633 or dst port 6633 –i eth0 –w
commutator.log
$sudo tcpdump src port 6633 or dst port 6633 –i eth2 –w
controller.log
```

Полученные файлы имеют бинарный формат и их анализ без соответствующей графической утилиты невозможен.

5.9.2 Установка и настройка Wireshark с поддержкой OpenFlow

Бинарные файлы tcpdump могут быть просмотрены программой WireShark. Парсер OpenFlow в комплекте поставки WireShark отсутствует, поэтому проще всего для его установки на десктоп администратора или разработчика воспользоваться скриптом установки mininet.

Установим mininet на Ubuntu Desktop 12.04 LTS.

```
$git clone git://github.com/mininet/mininet
$mininet/util/install.sh –a
```

Будет установлен пакет mininet с полным набором утилит, такие как OpenVSwitch, Wireshark и др. Операция занимает довольно много времени и закачивает большой объем данных.

Проверить наличие парсера можно войдя в меню Help->About Wireshark->Plugins и увидев там openflow.so, как это показано на рисунке 5.4.

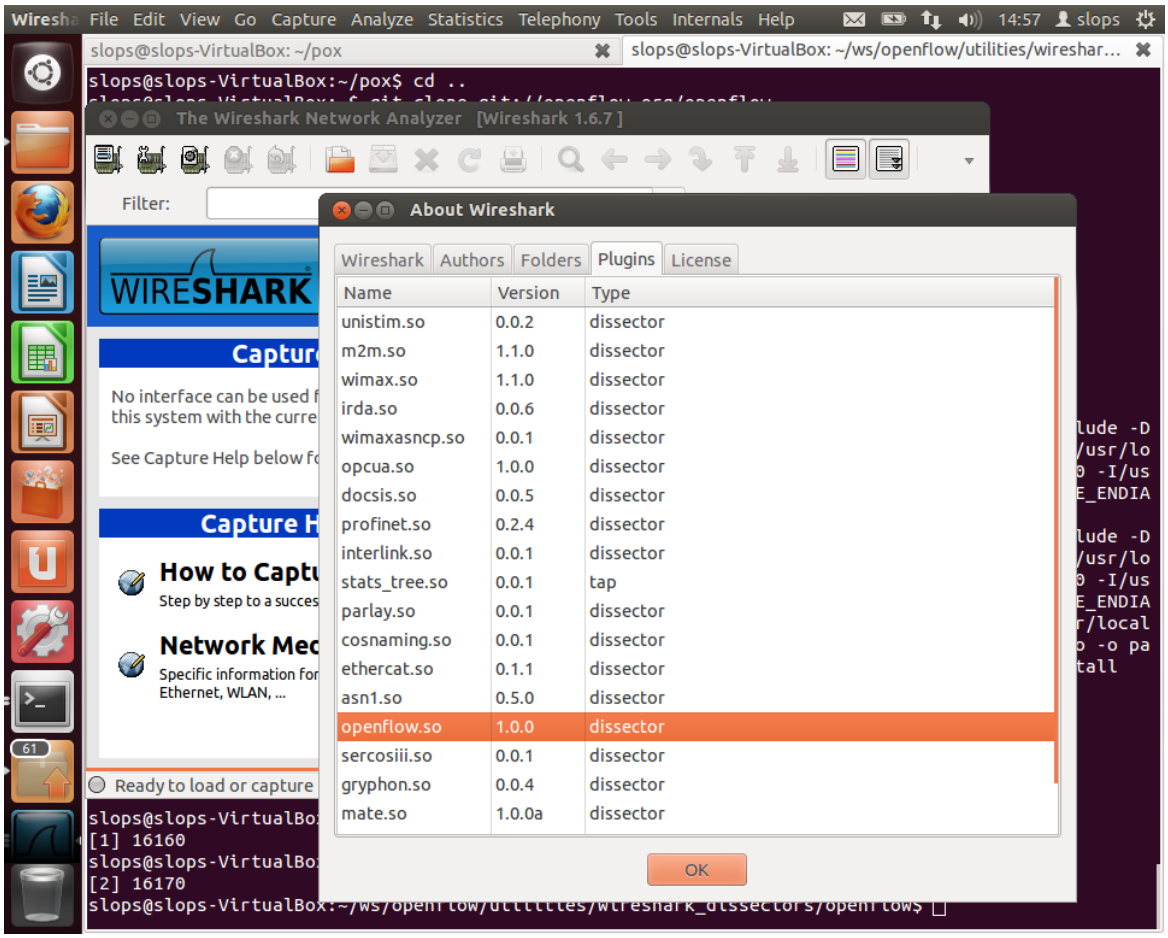

Рисунок 5.15 – Список плагинов Wireshark

Wireshark позволяет также в реальном времени разбирать пакеты, в частности если контроллер установлен на АРМ администратора.

Полученный через tcpdump файл можно загрузить в Wireshark, после чего для фильтрации трафика OpenFlow надо в строке «фильтр» написать «of» и применить фильтр «Apply», как это показано на рисунке 5.5.

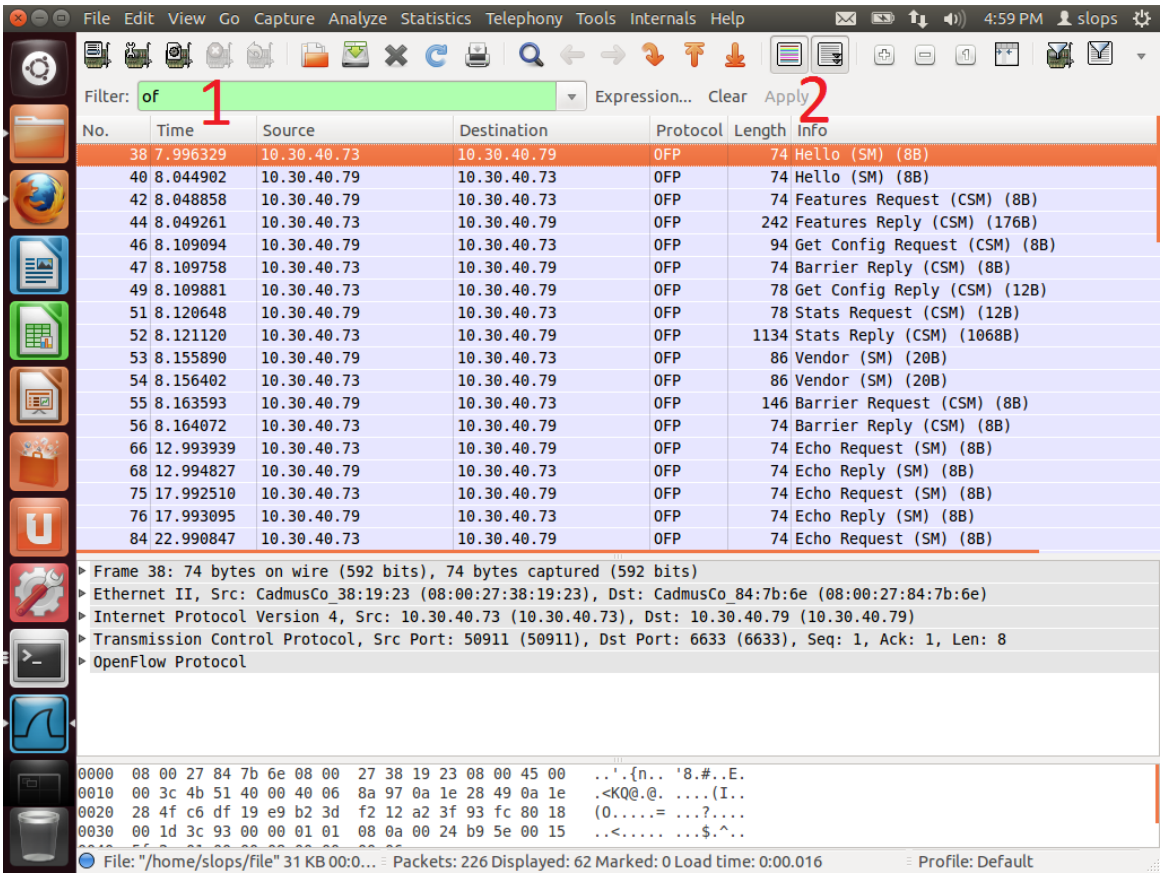

Рисунок 5.16 – Вывод Wireshark с плагином OpenFlow

Далее можно увидеть, что коммутатор посылает сообщение Hello. На данное сообщение следует ответ. Контроллер запрашивает список поддерживаемых опций и набор портов и т.д.

6 Обобщение и оценка результатов исследований

Цель выполнения НИР по теме «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» заключалась:

– в разработке научно-технического задела в области создания программных средств с открытым кодом для управления сетевыми ресурсами и потоками данных на основе подхода программно-конфигурируемых сетей с оценкой возможности применения ПКС для повышения эффективности управления компьютерными сетями и потоками данных;

– в проведении исследований и сравнительного анализа существующих сетевых операционных систем для ПКС, выявление перспективных направлений их развития, сравнительный анализ перспектив последующего развития существующих СОС на основе критериев производительности, масштабируемости, надежности, безопасности;

– в разработке программной модели платформы управления сетевыми ресурсами ПКС с открытым программным кодом.

Для достижения поставленных целей НИР в соответствии с пунктами технического задания и календарного плана были решены следующие задачи:

– Проведен аналитический обзор современной научно-технической, нормативной, методической литературы и публикаций по принципам построения, управления сетевыми ресурсами и архитектур ПКС в сравнении с традиционными сетями.

– Проведен 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.

– Сформирован набор критериев эффективности управления инфраструктурой КС и потоками данных. Были выделены 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.

– Разработана методика проведения экспериментального исследования выбранных СОС по показателям:

1) производительности (определение максимального количества обработанных запросов на установку соединений в секунду, минимальное время обработки одного запроса),
2) ресурсоёмкости (загрузка ядер процессора, расход физической памяти),
3) масштабируемости (изменение показателей производительности при увеличении числа подключённых коммутаторов и при увеличении числа ядер процессора),
4) функциональных возможностей (поддержка всех функций протокола OpenFlow),
5) надёжности (количество отказов за время тестирования, время безотказной работы при заданном профиле нагрузок) и безопасности (устойчивость к некорректно сформированным сообщениям протокола OpenFlow).

– Проведен сравнительный анализ методов и средств управления ПКС с традиционными методами и средствами управления сетевыми ресурсами и потоками данных в КС.

– Проведен анализ традиционных сервисов/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.

– Проведен анализ средств формирования OpenFlow таблиц для OpenFlow сетевых коммутаторов, возможности формирования сетевых срезов с помощью таких коммутаторов

– Проанализированы существующие подходы к виртуализации сетей.

– Проведены исследование, обоснование и выбор методов обеспечения QoS для ПКС.

– Разработаны методы и алгоритмы управления сетевыми ресурсами и потоками с помощью сетевой операционной системы.

– Разработана программная модель платформы управления сетевой инфраструктурой ПКС в соответствии с сформированным набором требований.

– Проведено экспериментальное исследование сетевых ОС и разработанных алгоритмов, проведен их анализ.

– Проведена технико-экономическая оценка рыночного потенциала полученных результатов.

– Сформированы рекомендации и предложения по использованию результатов НИР в реальном секторе экономики, а также дальнейших исследованиях и разработках.

– Разработаны методические материалы по построению и настройке ПКС в российских университетах и научно-исследовательских организациях.

– Разработан проект технического задания на проведение ОКР по теме: «Разработка отечественной распределенной платформы управления с открытым программным кодом для ПКС».

В результате выполнения исследований в рамках НИР были получены следующие результаты:

– Разработана программная модель прототипа отечественной платформы управления сетевыми ресурсами ПКС с открытым программным кодом

– Разработаны алгоритмы управления сетевой инфраструктурой ПКС:

1) алгоритм автоматического построения топологии;
2) алгоритм кодирования/декодирования пакетов;
3) алгоритм поддержки механизма подписки на события для модулей сетевой ОС;
4) алгоритм поиска кратчайших маршрутов.

– Реализованы разработанные алгоритмы в следующих программных модулях и компонентах прототипа платформы управления:

1) модуль обработки полученных пакетов;
2) модуль построения топологии ПКС;
3) модуль построения маршрутов в сети;
4) модуль формирования правил для сетевых элементов ПКС;
5) модуль, реализующий механизм подписки на события для других модулей СОС;
6) модуль распараллеливания обработки запросов по ядрам процессора;
7) модуль контроля состояния сетевых ресурсов.

– Разработаны методики экспериментальной оценки показателей производительности, масштабируемости, ресурсоемкости, надежности и безопасности для сетевых ОС.

– Разработаны методики экспериментальной проверки правильности работы разработанных алгоритмов и методов.

– Создан экспериментальный сегмент ПКС, на котором проводилось исследование сетевых ОС (NOX, POX, Beacon, FloodLight, MUL и Maestro) и алгоритмов в соответствии с методиками.

Анализ результатов экспериментального исследования позволяет сделать следующие выводы:

– Большинство свободно распространяемых контроллеров используются для решения исследовательских задач и в текущем состоянии не подходят для работы в сетях предприятий, провайдеров, глобальных сетях и ЦОД в силу их недостаточной производительности, масштабируемости, надежности и безопасности.

– Алгоритмы и методы, заложенные в существующих контроллерах, не являются оптимальными, поэтому открыты широкие возможности по разработке новых архитектур, алгоритмов и методов для контроллеров.

– Перспективным направлением развития является создание распределенных сетевых операционных систем, в которых устранены перечисленные выше недостатки масштабируемости, надежности и безопасности.

Полученные результаты были отражены в следующих документах, соответствующих техническому заданию:

– промежуточный отчёт о НИР за первый этап;

– программа и методики экспериментальных исследований;

– отчет о патентных и маркетинговых исследованиях;

– программная документация на программную реализацию экспериментального сегмента ПКС;

– программная документация (тексте и описании программы) на каждый программный модуль сетевой ОС;

– методические материалы по построению и настройке ПКС в российских университетах и научно-исследовательских организациях;

– проект технического задания по теме «Разработка отечественной распределенной платформы управления с открытым программным кодом для ПКС»;

– заключительный отчет о НИР.

В рамках данной работы были также сформулированы задачи и направления работ, требующие проведения дальнейших исследований, разработок и экспериментов:

– разработка распределенной платформы управления в соответствии с разработанным проектом ТЗ и разработанной программной моделью платформы управления;

– доказательство корректности разработанных алгоритмов для контроллера;

– совершенствование предложенных и разработка новых алгоритмов и методов управления сегментом сети ПКС;

– развитие методик для экспериментальной оценки показателей производительности, масштабируемости, надежности и безопасности сетевых ОС.

Таким образом, задачи, поставленные в рамках данной НИР, успешно выполнены, цели достигнуты, а полученные результаты исследований обоснованы и проанализированы. НИР открывает новые перспективные направления для разработок и исследований в области разработки отечественной платформы управления сетевой инфраструктурой и потоками данных в программно-конфигурируемых сетях.

7 Проведение технико-экономической оценки рыночного потенциала полученных результатов

7.1 Краткое описание результатов НИР, их сравнение с мировым уровнем

В ходе настоящей НИР были получены следующие результаты:

– Разработаны алгоритмы управления сетевой инфраструктурой ПКС, в том числе алгоритмы автоматического построения топологии, кодирования/декодирования пакетов, поддержки механизма подписки на события для модулей сетевой ОС, поиска кратчайших маршрутов.

– Реализованы разработанные алгоритмы в следующих программных модулях и компонентах прототипа платформы управления:

1) модуль обработки полученных пакетов;
2) модуль построения топологии ПКС;
3) модуль построения маршрутов в сети;
4) модуль формирования правил для сетевых элементов ПКС;
5) модуль, реализующий механизм подписки на события для других модулей СОС;
6) модуль распараллеливания обработки запросов по ядрам процессора;
7) модуль контроля состояния сетевых ресурсов.

– Разработаны методики экспериментальной оценки показателей производительности, масштабируемости, ресурсоемкости, надежности и безопасности для сетевых ОС.

– Разработаны методики экспериментальной проверки правильности работы разработанных алгоритмов и методов.

– Создан экспериментальный сегмент ПКС, на котором проводилось исследование сетевых ОС (NOX, POX, Beacon, FloodLight, MUL и Maestro) и алгоритмов в соответствии с методиками.

В мировых научно-исследовательских центрах, занимающихся развитием ПКС-технологий, разработка контроллеров и приложений к ним является одним из ключевых направлений работ.

Например, совместная исследовательская лаборатория университетов Стенфорда и Беркли Open Networking Laboratory (ON.Lab), специализирующаяся на разработке open-source решений для ПКС-сетей, работает над проектами:

– NOX - первый OpenFlow контроллер, написан с поддержкой двух языков: C++ и Python, был опубликован в 2008 г. под лицензией GPL и с тех пор этот контроллер является базовым для многих научно-исследовательских групп, которые только приступают к изучению SDN. Содержит сервисы для построения топологии сети и L2-L3 коммутацииВ настоящий момент проект не развивается.

– POX – часть проекта NOX, написанная на Python. По своей сути POX – это платформа для быстрой разработки и прототипирования ПО управления сетью. К примеру, исследовательская группа в Стэнфорде использует POX для исследования ключевых проблем ПКС. POX находится в стадии активного развития: все удачные идеи постоянно перемещаются из лабораторных экспериментов в официальные релизы контроллера POX. Дата релиза июнь 2011 г.

– Beacon – один из ведущих проектов на сегодняшний момент в данной области. Beacon - достаточно быстрый, кросс-платформенный, модульный OpenFlow контроллер на Java. Этот контроллер разрабатывается уже более двух лет. Beacon используется во многих научно-исследовательских проектах и тестовых развертываниях. Beacon применяется в экспериментальном ЦОДе Стэнфорда, в котором он управляет 100 виртуальными и 20 физическими коммутаторами. Релиз проекта состоялся в сентябре 2011 г.

– На основе Beacon разработан контроллер FloodLight - контроллер корпоративного уровня. Этот контроллер разрабатывается коммерческой компанией Big Switch Networks. FloodLight также как и другие контроллеры на Java является модульным, что очень удобно для разработчиков. FloodLight помимо open-source, имеет также коммерческую версию.

Помимо указанных проектов, также можно отметить операционную систему Maestro (разработана в Rice University), контроллер FlowER и Mul. Всего на сегодняшний день существует свыше 24 ПКС-контроллеров.

Проведенный анализ результатов экспериментального исследования в рамках данной работы, позволяет сделать следующие выводы:

- Большинство open-source контроллеров используются для решения исследовательских задач и в текущем состоянии не подходят для работы в сетях предприятий, провайдеров, WAN в силу их ненадежности, небезопасности и плохой масштабируемости.
- Алгоритмы и методы, заложенные в существующих контроллерах, не являются оптимальными, поэтому существуют широкие возможности по разработке новых архитектур, алгоритмов и методов для контроллеров.
- Создание распределенного контроллера, в которых устранены перечисленные выше недостатки масштабируемости, надежности и безопасности, является очень перспективным направлением развития.

Проведенное экспериментальное исследование является крайне актуальным, поскольку на сегодняшний день известно очень мало работ, связанных с экспериментальным исследованием характеристик контроллеров. Исследования по надежности и безопасности отсутствуют. Существующие исследования имеют недостаточный набор контроллеров (обычно до 5), для них также характерны слабая объективность результатов, поскольку в большинстве случаев лучшим оказывается проект, разработанный авторами исследований.

Разработанные алгоритмы управления сетями ПКС позволяют обеспечить заданное качество обслуживания для виртуальных сетей за счет меньшей сложности алгоритмов маршрутизации по сравнению с таковыми в традиционных сетях и эффективной организации работы модулей сетевой ОС, что также позволит поддерживать сети большого масштаба (500-1000 коммутаторов).

Кроме того, экспериментальная реализация алгоритмов управления ПКС должна обеспечить производительность, не меньшую по сравнению с таковой для мировых аналогов (контроллеры NOX, Beacon) (на сопоставимой аппаратной базе).

7.2 Технические перспективы применения результатов НИР

Результаты проведенных в НИР работ могут быть использованы для проведения нескольких прикладных НИР по следующим направлениям:

– разработка распределенной платформы управления в соответствии с разработанным проектом ТЗ и разработанной программной моделью платформы управления;

– доказательство корректности разработанных алгоритмов для контроллера;

– совершенствование программной модели и архитектуры платформы управления сетевой инфраструктурой ПКС;

– разработка новых модулей сетевой ОС;

– совершенствование алгоритмов функционирования модулей;

– развитие методик для экспериментальной оценки показателей производительности, масштабируемости, надежности и безопасности сетевых ОС.

Одним из результатов выполненной НИР является разработка требования к прототипу отечественной платформе управления ПКС. Такая система может занять важное место в ряду приоритетных направлений развития в сфере современных сетевых технологий, поскольку контроллер является основным элементов управления ПКС-сетью. В связи с этим результаты исследований по данной тематике и их практическая реализация может быть интересна самому широкому кругу заказчиков: государственный органы, телекоммуникационные компаний, финансовые организации, организации, работающие с большой инфраструктурой информационных технологий, научные и образовательные организации.

Для коммерциализации результатов научного исследования должно быть запланировано проведение следующих мероприятий:

– подготовка заявки на охранный документ (патент, свидетельство);

– организация участия в мероприятиях, направленных на освещение и популяризацию промежуточных и окончательных результатов НИР (конференции, семинары, симпозиумы, выставки и т.п., в том числе, международных);

– проведение оценки РИД, полученных при выполнении НИР, с целью их вовлечения в хозяйственный оборот;

– проведение маркетинговых исследований с целью изучения перспектив коммерциализации РИД, полученных при выполнении НИР;

– организация и ведение рекламной кампании; направленная на продвижение разработанных решений в целевой аудитории, маркетинговая работа на внешнем рынке.

7.3 Ожидаемые преимущества применения технологий ПКС в технической инфраструктуре заказчика

Технология ПКС нацелена на решение следующих проблем компьютерных сетей у заказчиков:

– повышение эффективности использования пропускной способности каналов, балансировка нагрузок в сети;

– упрощение управление сетью, повышение масштабируемости сети;

– повышение безопасности сетей;

– эффективная маршрутизация;

– снижение капитальных затрат.

Целевая аудитория заказчиков: новая технология отвечает запросам операторов связи, производителей компьютерного и телекоммуникационного оборудования, владельцев дата-центров, финансовые структуры и банки, хостинг и сервис провайдеров.

Преимущества ПКС-подхода:

– Производительность: благодаря снятию с коммутаторов нагрузки по обработке тракта управления (маршрутизации потоков данных), все ресурсы этих устройств направлены на оптимальное перемещение трафика.

– Экономичность: за счет упрощения коммутаторов, виртуализации управления сетью ПКС снижает расходы на построение и сопровождение сетей. По результатам тестов на базе крупнейших провайдеров США использование ПКС-технологий позволяет на 20-30 % увеличить загрузку незадействованных ресурсов центров обработки данных и в несколько раз снизить эксплуатационных расходы.

– Программирование сетевых сервисов: в ПКС сети пользователи могут сами писать программы для реализации новых сервисов обработки и управления трафиком в сети. Эти сервисы реализуют программы (виртуальные сетевые сервисы - наборы команд коммутатора), формируемые приложениями контроллера. Эти виртуальные сервисы не зависимы от поставщика коммутатора.

– Администрирование: поскольку контроллер консолидирует информацию о состоянии всех ресурсов ПКС сети, то существенно упрощаются вопросы администрирования сети. За счет специальных приложений контроллера можно запускать отладку и тестирования наборов правил Open Flow коммутаторов, генерируемых приложениями контроллера.

– Безопасность. Поскольку в ПКС сети тракт данных и тракт управления разделены, то нет возможности атаковать тракт управления из тракта данных, что является типичным источником атак в традиционных сетях. В ПКС сетях проще контролировать физический доступ к критическим компонентам, например маршрутизаторам, так как они сконцентрированы на контроллере, а не распределены в разных местах, доступ в которых бывает трудно проконтролировать. Оперативность управления потоками данных позволяет эффективнее, чем в традиционных сетях, реагировать на действия злоумышленников или появление вредоносного программного обеспечения.

– Виртуализация. Разделение логического понятия потока данных и физического маршрута в сочетании с динамической привязкой потока данных к

физическому каналу данных делают ПКС сети удобным средством для виртуализации сетевых ресурсов, для создания виртуальных сетей надежно изолированных друг от друга.

Поскольку контроллер является основным элементов управления ПКС-сетью, то без этого элемента эффективная реализация нового подхода к архитектуре компьютерных сетей невозможна.

В настоящий момент можно сказать, что решений на основе технологии программно-конфигурируемых сетей представлены в пяти сегментах рынка информационных технологий:

- оборудование для центров обработки данных;
- телекоммуникации;
- облачные технологии;
- корпоративные решения;
- безопасность.

Первые два сегмента: оборудование для центров обработки данных и телекоммуникации будут включать гораздо больше программного обеспечения, где ПКС будет включен в качестве элемента управления в оборудование, что позволит отделить некоторые функции от коммутаторов и маршрутизаторов.

Значительный интерес решения на основе ПКС-технологий представляют для отрасли телекоммуникаций. Телекоммуникационные провайдеры последнее десятилетие ищут способы оптимизации затрат и повышения эффективности своих сетей. В то же время, промышленность также дает стимул развитию новых форм сетей, которые будут быстрее, проще и легче в управлении. Технология ПКС помогает непосредственно провайдерам телекоммуникационных услуг, перенаправить сетевой трафик и снижает перегрузки сети. В конечном итоге это приводит к значительной экономии средств, которые могут быть перенаправлены на основные цели бизнеса.

Сетевая операционная система (контроллер) облегчает управление приложениями и сервисами, установленными поверх топологии сети. ПКС также привносит ряд новых возможностей, таких как сегментирование и виртуализация используемой сети. Эта свойство ПКС позволяет осуществлять логические манипуляции конкретным сегментом сети, тем самым повышая ее гибкость. ПКС обещает дать больше возможностей по управлению инфраструктурой ЦОД и сетей операторов связи, позволяя повысить оптимизацию и адаптивность сетей под задачи клиентам, а, следовательно, приведет к сокращению общих капитальных и эксплуатационных затрат.

Основными движущими силами ПКС на рынке являются следующие тенденции:

– растущая потребность в мобильности;

– набирающая популярность концепция «Принеси свое собственное устройство» (Bring Your Own Device, BYOD) – использование сотрудниками компании своих собственных устройств на рабочем месте;

– острая необходимость в снижении операционных расходов операторов и ЦОД (OPEX) и капитальных затрат (CAPEX).

7.4 Оценка зарубежного рынка

На сегодняшний день существуют несколько оценок потенциала роста рынка ПКС-решений на ближайшие 3-5 лет. В связи с «молодостью» технологии и отсутствием значимого количества практических внедрений (т.е. практической базы для исследований и изучения опыта), эти прогнозы очень сильно различаются и носят очень приблизительный характер. Однако, для оценки потенциала рынка и анализа основных трендов на рынке, данные отчеты представляют большой интерес.

Рынок ПКС-решений только движется от лабораторных исследований к рыночной продукции, до сих пор не существует четких стандартов в ПКС. Кроме того, такие факторы как современное состояние исследований, зрелости продукта, различное видение перспектив развития технологии, а также малое количество

внедрений в реальную инфраструктуру могут препятствовать широкому коммерческому внедрению технологии ПКС в будущем.

Аналитическое агентство Markets&Markets прогнозирует, что объем рынка ПКС-продуктов к 2016 г. будет превышать 2 миллиарда долларов. Прогноз этого агентства для рынка ПКС и OpenFlow включается в себя не только оценку рынка коммутаторов и маршрутизаторов, но и рынка сервисов и программного обеспечения. В 2013 г. объем рынка предположительно будет составлять 168 миллионов долларов и к 2016 г. он вырастет до 2 миллиардов долларов. Аналитик Кейси Куиллин из агентства Dell'Oro Group предполагает в период с 2010 до 2016 год общая сумма расходов компаний на ПКС и OpenFlow возрастут более, чем в 17 раз [21].

В соответствии с отчетом Dell'Oro Group почти все производители сетевого оборудования к 2016 году будут поддерживать ПКС и предлагать продукты на основе этой технологии, но оно будет использовать только часть потенциала этой технологии. Даже если все функции коммутатора перенести на программный уровень, сами коммутаторы тем е менее будут необходимы для соединения серверов с пользователями и ресурсами [21].

В апреле 2013 г. вышел отчет, проведенный информационным агентством SDNCentral, компанией Plexxi и венчурным агентством Lightspeed Ventures, в котором утверждается, что объем рынка ПКС-решений к 2018 г. может преодолеть сумму в 35 миллиардов долларов. Одним из ключевых факторов такого роста аналитики называют рост венчурных инвестиций в компаниях, ориентированных на программно-конфигурируемые сети. Как показало исследование, венчурный капитал ориентированных на ПКС компаний вырос с 10 миллионов долларов в 2007 году до 454 миллионов долларов в 2012 году.

Исследования, проведенные на основе общедоступных данных и закрытых рыночных моделях компании SDNCentral, показали, что:

– доля расходов на оборудование с поддержкой ПКС-технологий увеличится с 2% в этом году до 40% в 2018 году;

– объем рынка программного обеспечения виртуализации сети вырастет с 10 миллионов долларов в этом году до 1,2 миллиардов долларов в 2018 году [22].

Как показало исследование, несмотря на переход к ПКС, сетевое оборудование по-прежнему будет играть значительную роль в инфраструктуре сети. К 2018 году 46% от общего объема расходов ЦОД будет уходить на оптическое, коммутационное и маршрутизационное оборудование, поддерживающие ПКС-технологии, и более значительная сумма – 49% - на оборудование, не поддерживающие ПКС.

Отдельную аналитику по развитию рынка ПКС-решений проводить достаточно затруднительно, поскольку часть доходов от ПКС-решений (особенно в части программного обеспечения, к которым относится и контроллеры, и приложения для них) косвенно учитываются в показателях других рынка, например решений для управления сетями, сетевой безопасности, приложений для уровней L4-7.

Doyle Research и GigaOM Research предполагают, что рынок ПКС-решений достигнет объема в 320 миллионов долларов в 2014 году и вырастет до 2,8 миллиардов долларов к 2018 г. Этот рост будет обусловлен расширением возможностей использования ПКС-технологий для улучшения операций в корпоративных сетях, ЦОД, WAN и кампусных сетях. Сетевые аппаратные средства в 2014 году все еще будут составлять наибольшую долю рынка ПКС-решений (уровень L2-7) – около 66%. Эта доля снизится до 47 % к 2018 году, поскольку значительно возрастут доходы от программного обеспечения и различных сервисов для ПКС-сетей. Соответственно, сектор только программного обеспечения и сетевых сервисов к 2018 году будет составлять не менее 1,3 миллиарда долларов [23].

Практически все лидеры рынка сетевого оборудования работают над созданием собственного контроллера. Так, Juniper в настоящее время работает над созданием своего, третьего «стандарта» для контроллеров ПКС. Он должен стать альтернативой с открытым кодом тем технологиям, которые уже предлагаются или

будут предложены компаниями Cisco и Vmware [24]. Juniper поддерживает контроллер OpenFlow, предложенный компанией-стартапом Big Switch Networks. Этот проект поддерживается и многими другими отраслевыми игроками, в том числе Arista Networks, Broadcom, Brocade, Citrix, Dell, Extreme Networks, F5, Mellanox и Microsoft. Между лидеры индустрии ЦОД HP и IBM предлагают свои собственные контроллеры на основе OpenFlow. Выпуск первого продукта — распределенного контроллера сети с поддержкой протоколов BGP и XMPP намечен на 2013 год. Компания IBM также предлагает собственный программный контроллер ПКС на базе OpenFlow — Programmable Network Controller (PNC), для развертывания которого необходим сервер с процессором Intel Xeon под управлением ОС Red Hat Enterprise Linux.

Корпорация Hewlett-Packard (HP) предлагает рынку собственный контроллер - HP Virtual Application Networks Controller. На 2013 год запланированы поставки контроллеров Virtual Application Networks SDN Controller, с 2014 года в HP намерены сосредоточиться на собственных и разработанных партнерами ПКС-приложениях, а в 2015-м планируют перейти к развертыванию ПКС в масштабе предприятий. Компания намерена со временем открыть «супермаркет» решений, подобный App Store, который должен упростить развертывание ПКС.

7.5 Оценка российского рынка

Говорить об оценках рынка ПКС-решений в России пока преждевременно, однако стоит отметить интерес со стороны крупных заказчиков к этой технологии. Среди них отдельно следует отметить ОАО «Ростелеком», который нацелен на разработку оригинальной платформы для управления собственными ЦОД на основе ПКС. Важным элементом будущей платформы будет составлять распределенный контроллер. Кроме того, потенциально возможную заинтересованность в развития ПКС-технологии на российском рынке показывают такие крупные игроки как АФК «Система», Yandex, системных интеграторов («КРОК»), ОАО «Сбербанк».

Поскольку в настоящий момент российский рынок сетевых технологий является зависимым от западных компаний, то соответственно, на него воздействуют такие же тренды, но с небольшим временным интервалом. Поэтому для оценки возможного потенциала разрабатываемой технологии следует проанализировать общие тенденции на рынке оборудования и программного обеспечения для компьютерных сетей. Рынок в последние несколько лет показывает устойчивые темпы роста около 10-12 % в год. Аналитики отмечают также тенденцию увеличения объема поставок при одновременном снижении оборота в деньгах, что это происходит потому, что конкуренция заставляет вендоров снижать цены. Представители большинства игроков сетевого рынка отмечают, что в прошлом году динамичнее всего развивались сегменты домашнего оборудования и конечных устройств. Следует отметить, что в целом рынок сетевых устройств прибавлял пропорционально, существенных структурных изменений в 2011 и 2012 гг. не было. Сегмент решений корпоративного уровня по-прежнему существенно опережает по обороту решения для дома и малого офиса. Среди мировых поставщиков лидерство сохраняет Cisco, что объясняется помимо прочего еще и тем, что во многих государственных структурах и крупных корпорациях этот бренд выбран в качестве корпоративного стандарта. В России в 2011 г. также был отмечен значительный рост в секторе оборудования для развертывания ЦОДов, а также в сегменте беспроводных решений. Это связано с растущей мобильностью сотрудников, когда компании переходят к концепции BYOD (Bring Your Own Device). Наибольший рост демонстрировали новые ниши, а также те сегменты, которые пострадали в кризис. Это прежде всего оборудование для инфраструктурных проектов, мощные серверы, СХД, высокопроизводительное оборудование уровня ядра сети.

По данным компании IDC, в сегменте расходов на оборудование в области информационных технологий Россия в 2010 г. входила в десятку ведущих стран, с показателем общей суммы расходов, на 12% превышающим среднемировое значение, и всего в 3–5 раз отставала от стран Западной Европы и США в расчете на душу населения. Однако по расходам на ПО мы занимали уже 16-е место, отставая

от среднемирового значения на 55%, а от США и Западной Европы – соответственно в 20 и 10 раз [25].

По данным Минкомсвязи, сейчас объем отрасли информационных технологий в России составляет более 250 миллиардов рублей. Эта сфера обеспечивает 300 тыс. рабочих мест и экспорт в объеме 4 миллиардов долларов.

При поддержке государством эта отрасль должна серьезно вырасти. Целевой показатель – рост в в 3 раза быстрее валового внутреннего продукта в целом по стране. К 2018 г. целевой объем производства должен составить 450 миллиардов рублей, экспорта –9 миллиардов долларов, а венчурного финансирования - 40 миллиардов рублей.

В рамках поддержки научных исследований в сфере информационных технологий планируется создать с участием государства до 50 центров исследований на базе высших учебных заведений и научных центров. Процедуры их отбора, которые планируется утвердить до конца 2013 г., будут проводиться с участием представителей индустрии информационных технологий.

В сфере развития инфраструктуры к 2015 г. должно быть завершено строительствои вывод на проектную мощность всех строящихся в рамках госпрограммы технопарков. Также к развитию инфраструктуры Минкомсвязиь относит поддержку экспорта российской продукции в сфере информационных технологий.

Наконец, более привлекательные условия для ведения бизнеса в области информационных технологий должны появиться за счет снижения ставки страховых взносов для компаний этой сферы с численностью от 7 до 29 человек [26].

7.6 Выводы

Технико-экономическая эффективность реализации предложений по тематике исследований является высокой. Учитывая анализ направлений зарубежных лабораторий и университетов по сходной тематике можно сделать вывод, что

выполненные научно-исследовательские работы находятся в общем направлении исследований и разработок по данной тематике.

Использование результатов НИР предполагается в одной из наиболее перспективных отраслей ИКТ, которая прогнозирует темпы роста не менее 60 % в год при среднем показателей по рынку ИКТ около 10 %.

Результаты НИР потенциально могут быть интересны следующим категориям заказчиков: операторы связи, корпорации со сложной инфраструктурой информационных технологий, финансовые организации, государственные органы.

Использование результатов НИР позволит заказчикам осуществлять централизованный контроль за инфраструктурой информационных технологий, снизить зависимость от продукции одного вендора, уменьшит сложность управление сетями, повысит качество обслуживания.

В целом российский рынок развивается в общем тренде с западными рынками. Наибольшие темпы роста показывает сектор оборудования для развертывания ЦОДов, сегмент беспроводных решений и сегмент решений корпоративного уровня. Основной объем продукции – решений зарубежных производителей. При указанных темпах роста и активном стимулировании развития отрасли информационных технологий со стороны государства на российском рынке будут больше востребованы новые технологии, разработанные российскими исследовательскими центрами.

8 Разработка ррекомендаций и предложений по использованию результатов НИР в реальном секторе экономики, а также в дальнейших исследованиях и разработках

8.1 Перспективы применения результатов НИР в дальнейших исследованиях и разработках

Результаты проведенных в НИР работ могут быть использованы для проведения нескольких прикладных НИР по следующим темам:

– разработка распределенной платформы управления в соответствии с разработанным проектом ТЗ и разработанной программной моделью платформы управления$

– доказательство корректности разработанных алгоритмов для контроллера;

– совершенствование программной модели и архитектуры платформы управления сетевой инфраструктурой ПКС;

– разработка новых модулей сетевой ОС;

– совершенствование алгоритмов функционирования модулей;

– развитие методик для экспериментальной оценки показателей производительности, масштабируемости, надежности и безопасности сетевых ОС.

Одним результатов выполненной НИР является разработка требования к прототипу отечественной платформе управления ПКС. Такая система может занять важное место в ряду приоритетных направлений развития в сфере современных сетевых технологий, поскольку контроллер является основным элементов управления ПКС-сетью. В связи с этим результаты исследований по данной тематике и их практическая реализация может быть интересна самому широкому кругу заказчиков: государственный органы, телекоммуникационные компаний, финансовые организации, организации, работающие с большой инфраструктурой информационных технологий, научные и образовательные организации.

Кроме того, разработанный экспериментальный сегмент ПКС позволяет проводить экспериментальные исследования алгоритмов и приложений в сетях ПКС, воспроизводить различные сценарии функционирования сети.

8.2 Перспективы применения результатов НИР в реальном секторе экономики

Результаты НИР могут быть использованы, как основа для систем управления сетями национального масштаба, корпоративными сетями, сетями ЦОД, сетями сервис-провайдеров, а также в домашних сетях.

Применение технологий ПКС в сетях национального масштаба и сетях поставщиков телекоммуникационных услуг должно дать следующие преимущества:

– Снижение операционных расходов – централизованное управление сетью из одного места при помощи ограниченного количества протоколов снижает издержки на поддержку сети. По сути, контроллеры берут на себя часть рутинной работы сетевых инженеров. Сбор статистики, мониторинг и управление доступны через один открытый протокол. Конфигурация сети больше не разбросана по устройствам.

– Упрощение логики сети – уменьшение числа протоколов в сети способствует снижению рисков их неверной настройки (человеческий фактор).

– Простота внедрения новых сервисов – по сути, надо только перепрограммировать контроллеры и новые сетевые услуги будут внедрены.

– Снижение коэффициента потерь пакетов – сеть имеет возможность реагировать на изменения сразу после поступления события на контроллер. Больше нет нужды в ожидании времени сходимости протоколов маршрутизации и STP.

Корпоративные сети характеризуются повышенными требованиями безопасности и ограничения прав доступа. При применении ПКС-подхода большая часть нужной функциональности реализуется, как приложение для разрабатываемой платформы управления ПКС, например, NAT, межсетевой экран, система балансировки нагрузки.

Для эффективного использования сетевых ресурсов ЦОД может быть применена виртуализация сетей, которая активно развивается в рамках концепции ПКС сетей. Под виртуализацией сети понимается изоляция сетевого трафика, то есть группирование (мультиплексирование) нескольких потоков данных с различными характеристиками в рамках одной логической сети, которая может разделять единую физическую сеть с другими логическими сетями или сетевыми срезами (network slices). Каждый срез может использовать свою адресацию, свои алгоритмы маршрутизации, управления качеством сервисов и т.д. Таким образом, каждый пользователь виртуализированного multi-tenancy ЦОД имеет не только набор виртуальных машин, но и свою виртуальную сеть, которой он может управлять.

Домашние сети характеризуются большим количеством различных сетевых устройств (сетевой жесткий диск, принтер, медиа устройства и т.п.), обменивающиейся разными типами трафика с различными качествами сервиса. Разработываемая программная платформа управления позволяет управлять такими системами через единый графический интерфейс, реализованный как сетевое приложение для этой платформы. Что в целом позволит повысить экономическую эффективность использования домашних сетей.

Также представляет интерес исследование применимости технологий ПКС в таких современных направлениях, как «умный город», «Интернет вещей».

ЗАКЛЮЧЕНИЕ

В результате выполнения работ по заключительному этапу НИР «Создание и развитие отечественной платформы с открытым программным кодом для управления программно-конфигурируемыми сетями (ПКС)» были получены следующие результаты:

– Разработаны следующие функциональные алгоритмы для сегмента ПКС:

1) алгоритм автоматического построения топологии ПКС;
2) алгоритм маршрутизации в рамках сегмента ПКС;
3) алгоритм кодирования/декодирования пакетов;
4) алгоритм для поддержки механизма подписки на события для модулей сетевой ОС.

– Разработана программная модель платформы управления на основе выбранной сетевой ОС и алгоритмов управления сегментами ПКС.

– Создан стенд для проведения экспериментальных исследований характеристик сетевых ОС.

– Проведено экспериментальное исследование существующих сетевых ОС и программной модели платформы управления с разработанными алгоритмами.

– Проведены обобщение и оценка результатов проведенных исследований в рамках НИР.

– Проведена технико-экономическая оценка рыночного потенциала полученных результатов НИР.

– Сформулированы рекомендации и предложения по использованию результатов НИР в реальном секторе экономики, а также в дальнейших исследованиях и разработках.

– Разработаны методические материалы по построению и настройке ПКС в российских университетах и научно-исследовательских организациях.

– Разработан проект технического задания на проведение ОКР по теме: «Разработка отечественной распределенной платформы управления с открытым программным кодом для ПКС».

Результаты работы по заключительному этапу НИР нашли отражение в следующих документах: настоящий отчёт о НИР, программная документация, отражающая экспериментальную реализацию разработанных программно-технических решений, методические материалы по построению и настройке ПКС в российских университетах и научно-исследовательских организациях, проект технического задания на проведение ОКР по теме: «Разработка отечественной распределенной платформы управления с открытым программным кодом для ПКС».

Таким образом, цели и задачи, поставленные в рамках заключительного второго этапа НИР, успешно выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Отчёт о НИР по теме: Создание отечественной ПКС платформы управления 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, надёжности, безопасности (Промежуточный) [Текст]. — Москва: МГУ имени М.В. Ломоносова, 2013.

2 About POX [Electronic resource] // NOXRepo. — Mode of access: http://www.noxrepo.org/pox/about-pox/ (accessed date: 15.07.2013).

3 NOX: towards an operating system for networks [Текст] / N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker // SIGCOMM Comput. Commun. Rev. — 2008. — Т. 38, № 3. — С. 105–110.

4 Web-страница проекта Mul [Электронный ресурс] — Режим доступа: http://sourceforge.net/p/mul/wiki/Home/ (дата обращения: 06.05.2013).

5 Maestro: A System for Scalable OpenFlow Control [Электронный ресурс] / Zheng Cai, Alan L. Cox, T. S. Eugene Ng // Rice University. — [2010]. — Режим доступа: http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf (дата обращения: 28.04.2013).

6 Full-Stack OpenFlow Framework in Ruby and C [Electronic resource] // GitHub. — Mode of access: https://github.com/trema/trema (accessed date: 15.07.2013).

7 Beacon Guides [Электронный ресурс] — Режим доступа: https://openflow.stanford.edu/display/Beacon/Guides (дата обращения: 14.07.2013).

8 Jaxon:Java-based OpenFlow Controller [Electronic resource] // Jaxon home page. — Mode of access: http://jaxon.onuos.org/ (accessed date: 15.07.2013).

9 Floodlight Wiki [Электронный ресурс] — Режим доступа: http://www.projectfloodlight.org/documentation/ (дата обращения: 16.07.2013).

10 The repository for the SNAC OpenFlow Controller [Электронный ресурс] — Режим доступа: https://github.com/bigswitch/snac (дата обращения: 15.07.2013).

11 Ryu SDN Framework [Electronic resource] — Mode of access: http://osrg.github.io/ryu/ (accessed date: 13.07.2013).

12 NodeFlow: An OpenFlow Controller Node Style [Electronic resource] // blog.garyberger.net. — [2012]. — Mode of access: http://garyberger.net/?p=537 (accessed date: 15.07.2013).

13 ovs-controller [Электронный ресурс] — Режим доступа: http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities%2Fovs-controller.8 (дата обращения: 12.07.2013).

14 Carving research slices out of your production networks with OpenFlow [Текст] / R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, G. Parulkar // SIGCOMM Comput. Commun. Rev. — 2010. — Т. 40, № 1. — С. 129–130.

15 Virtual routers as a service: the RouteFlow approach leveraging software-defined networks [Текст] / M.R. Nascimento, C.E. Rothenberg, M.R. Salvador, C.N.A. Corrêa, S.C. de Lucena, M.F. Magalhães // Proceedings of the 6th International Conference on Future Internet Technologies: CFI '11. — New York, NY, USA: ACM, 2011. — С. 34–37.

16 Controller Performance Comparisons [Электронный ресурс] // OpenFlow Wiki. — Режим доступа: http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons (дата обращения: 17.07.2013).

17 Production Quality, Multilayer Open Virtual Switch [Электронный ресурс] — Режим доступа: http://openvswitch.org/ (дата обращения: 15.07.2013).

18 Pantou: OpenFlow 1.0 for OpenWRT [Electronic resource] — Mode of access: http://www.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT (accessed date: 15.07.2013).

19 CPqD/ofsoftswitch13 [Электронный ресурс] — Режим доступа: https://github.com/CPqD/ofsoftswitch13 (дата обращения: 14.07.2013).

20 Indigo [Electronic resource] // Project Floodlight. — Mode of access: http://www.openflowhub.org/display/Indigo/ (accessed date: 15.07.2013).

21 High hopes for an SDN revolution [Electronic resource] / Roberta Prescott // RCRWireless. — Mode of access: http://www.rcrwireless.com/americas/20120727/componets/sdn-arrived-will-take-us-would-openflow-drive-sdn-revolution/ (accessed date: 15.07.2013).

22 SDNCentral Exclusive: SDN Market Expected to Reach $35B by 2018 [Electronic resource] / Matthew Palmer // SDN Central. — [2013]. — Mode of access: http://www.sdncentral.com/market/sdn-market-sizing/2013/04/ (accessed date: 13.07.2013).

23 Forecast: sizing the software-defined networking market [Electronic resource] / Lee Doyle // ADARA. — [2013]. — Mode of access: http://www.adaranet.com/pdf/PR-20130207.pdf (accessed date: 15.07.2013).

24 Боб Маглиа: Cisco, VMware и OpenFlow фрагментируют SDN [Электронный ресурс] / Джим Даффи // Computerworld Россия. — Режим доступа: http://www.osp.ru/cw/2013/02/13033613/ (дата обращения: 15.07.2013).

25 ИТ-рынок России [Электронный ресурс] // Сайт компании «Технологии поддержки»". — [2012]. — Режим доступа: http://www.techsupp.ru/company/news/2012/2106/ (дата обращения: 15.07.2013).

26 Правительство одобрило «дорожную карту» развития ИТ-отрасли [Электронный ресурс] / Денис Легезо // Cnews | информатизация. — [2013]. — Режим доступа: http://corp.cnews.ru/news/top/index.shtml?2013/07/15/535475 (дата обращения: 15.07.2013).