Модел. и анализ информ. систем. Т. **20**, № **4** (2013) 110–124
©*Красотин А. А., Алексеев И. В.,* 2013

УДК 004.4:004.7

Программно-конфигурируемые сети как этап эволюции сетевых технологий

Красотин А. А., Алексеев И. В.[1]

ООО «Энергия-Инфо»
150008 Россия, г. Ярославль, ул. Союзная, д. 144
Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: a.krasotin@a-real.ru, aiv@yars.free.net

получена 10 октября 2013

Ключевые слова: программно-конфигурируемые сети, SDN, OpenFlow, контроллер сети

Рассматривается понятие программно-конфигурируемой сети. Вначале даётся короткая историческая справка о понятии программно-конфигурируемой сети как научно-технической концепции, кем оно было введено и что означает. Авторы статьи рассматривают технологию программно-конфигурируемых сетей как один из возможных этапов и направлений развития сетевых парадигм в целом, не абсолютизируя роли этой технологии. Наряду с достоинствами отмечаются и недостатки программно-конфигурируемых сетей, рассматриваются возможные варианты развития программно-конфигурируемых сетей в контексте гибридизации с другими технологиями, в частности - гибридизация MPLS и SDN. Значительное внимание уделяется протоколу OpenFlow. В конце статьи рассмотрены существующие библиотеки для программной реализации управления программно-конфигурируемой сетью с использованием протокола OpenFlow. Все эти библиотеки предоставляют API для создания модульных приложений управления программно-конфигурируемыми сетями. Для сравнения производительности библиотек приведены результаты сравнительных тестов по пропускной способности и латентности.

Введение

Первоначально глобальные IP-сети были построены на основе понятия автономной системы (AS). Это понятие позволяет сетям масштабироваться и расширяться через соединения, которые передают пакеты к следующему разумному транзитному участку. Такое представление сети просто и имеет доказанную отказоустойчивость и масштабируемость, именно на этом принципе построена сеть Интернет.

[1]Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации в рамках государственного контракта от 20 июня 2013 г. № 14.514.11.4084.

Принцип автономной системы не позволяет перемещать места назначения, т.е. менять топологию сети, не изменяя их идентификационные данные, поскольку они тесно связаны со службой доставки пакетов. Топологическое расположение места назначения, интерфейсы которого имеют подключения к сети, диктует идентификационные данные. Вдобавок использование только базовой автономной системы затрудняет определение других идентификационных качеств, таких как логическая группировка, контроль доступа, качество сервиса, межсетевое взаимодействие или определение аспектов, которые имеют отношение к последовательности пакетов, формирующих поток или сетевой сеанс связи. Для решения их проблем организация Internet Engineering Task Force (IETF) ввела новые стандарты, такие как виртуальные локальные сети (VLAN) и виртуальные приватные сети (VPN), среди множества других стандартов. Возрастающее число стандартов влечёт за собой увеличение сложности в спецификации сетевых элементов и конфигурации сетевых интерфейсов операторами сети.

1. Причины возникновения новой парадигмы

Взрывоподобный рост и распространение мобильных устройств и контента для них, виртуализация серверов и распространение облачных сервисов являются основными трендами для переосмысления традиционных сетевых архитектур.

Обозначим факторы более детально.

1. Изменение паттернов трафика. Внутри центров обработки данных происходит существенное изменение паттернов трафика. В отличие от клиент-серверной архитектуры, когда основной объем коммуникаций происходит между клиентами и сервером, современные приложения создают множество потоков данных между отдельными компьютерами, а не только с серверами. Кроме того, пользователи также меняют паттерны трафика, применяя мобильные устройства, которым необходим полноценный доступ к корпоративным ресурсам и приложениям из любой точки сети в любое время.

2. Использование персональных мобильных устройств. Пользователи все чаще применяют на работе различные персональные мобильные устройства, такие как планшеты, мобильные телефоны, коммуникаторы и ноутбуки для доступа в корпоративную сеть. Архитектура сети должна принять эти устройства с возможностью тонкого контроля и с учетом защиты корпоративных данных и интеллектуальной собственности при сохранении политики использования сети и данных в ней.

3. Развитие облачных сервисов. Предприятия активно используют публичные и частные облачные сервисы. Бизнес-логика диктует необходимость доступа к приложениям, инфраструктуре и другим ИТ ресурсам по запросу и с высокой степенью детализации. В дополнение к этому крайне важны вопросы безопасности при работе с частными и публичными облаками. Вся эта архитектура должна также учитывать возможность оперативных изменений в структуре предприятий, как внутренних, так и различных слияний и поглощений. Таким образом, вслед за вычислительны-

ми ресурсами сетевая инфраструктура должна также эластично масштабироваться, реорганизовываться на лету и желательно управляться единым инструментом вместе с инфраструктурой хранения, вычислительными мощностями и т.д.

4. Резкое увеличение объемов данных. Современные приложения, например бизнес-аналитики, и другие предусматривают работу с огромными объемами данных и обмен между множеством серверов, которые могут иметь непредсказуемый заранее набор связей друг с другом. Увеличение объемов передаваемых по сети данных и непредсказуемость типовых путей передачи также требует от сети возможности по существенному ее масштабированию, которое невозможно в рамках традиционной сетевой парадигмы. Все эти факторы, по сути новые функциональные требования к архитектуре сети, приводят к смене парадигмы, поскольку существующие архитектурные принципы не способны удовлетворить новые потребности. Сетевые технологии на сегодняшний день представляют собой набор протоколов, разработанных для обеспечения коннективности хостов по различным типам каналов, топологий и на различных дистанциях. Эти протоколы разрабатывались в изоляции друг от друга (собственно такая изоляция и лежит в основе многоуровневой модели сетевой архитектуры, будь то OSI или TCP/IP). Результатом является повышенная сложность сетей. Например, чтобы добавить новое устройство, необходимо изменить конфигурацию коммутаторов, маршрутизаторов, межсетевых экранов и т.д., обновить списки контроля доступа, информацию о ВЛВС, качестве обслуживания и другие механизмы, используя инструменты администрирования, специфические для конкретного оборудования, причем в зависимости от изготовителя оборудования, типа и версий ПО. Таким образом, современные сетевые архитектуры испытывают сложности в оперативной поддержке меняющихся требований приложений и бизнес логики, имеют недостаточную масштабируемость и сложны в настройке и обслуживании, зависят от специфики вендорных решений.

Поскольку развиваются облачные архитектуры и динамическое распределение ресурсов, растёт использование мобильных компьютеров и виртуальных машин, возникает потребность в дополнительном уровне программно-конфигурируемой сети. Программно-конфигурируемые сети (SDN от англ. Software-defined Network) – это вид компьютерных сетей, разработанный совместно университетами Беркли и Стэнфорда в 2005 году. SDN сети позволяют администраторам сети управлять сетевыми сервисами посредством абстракций над функциональностью более низкого уровня, перемещать сетевые объекты без изменения идентификационных данных или нарушения технических требований, упрощает сетевые операции. В парадигме SDN функции управления и контроля поведения сетевых устройств полностью отделены от процессов пересылки данных. Более того, функции контроля программируются непосредственно приложениями. Благодаря такому отделению контроля, функции которого ранее были жестко привязаны к каждому сетевому устройству, инфраструктура сети теперь выступает отдельным уровнем абстракции для приложений и сетевых служб, которые могут рассматривать сеть как логическую или виртуальную сущность.

Всё это достигается путём разделения уровня управления сетью и устройств передачи данных (Рис. 1).

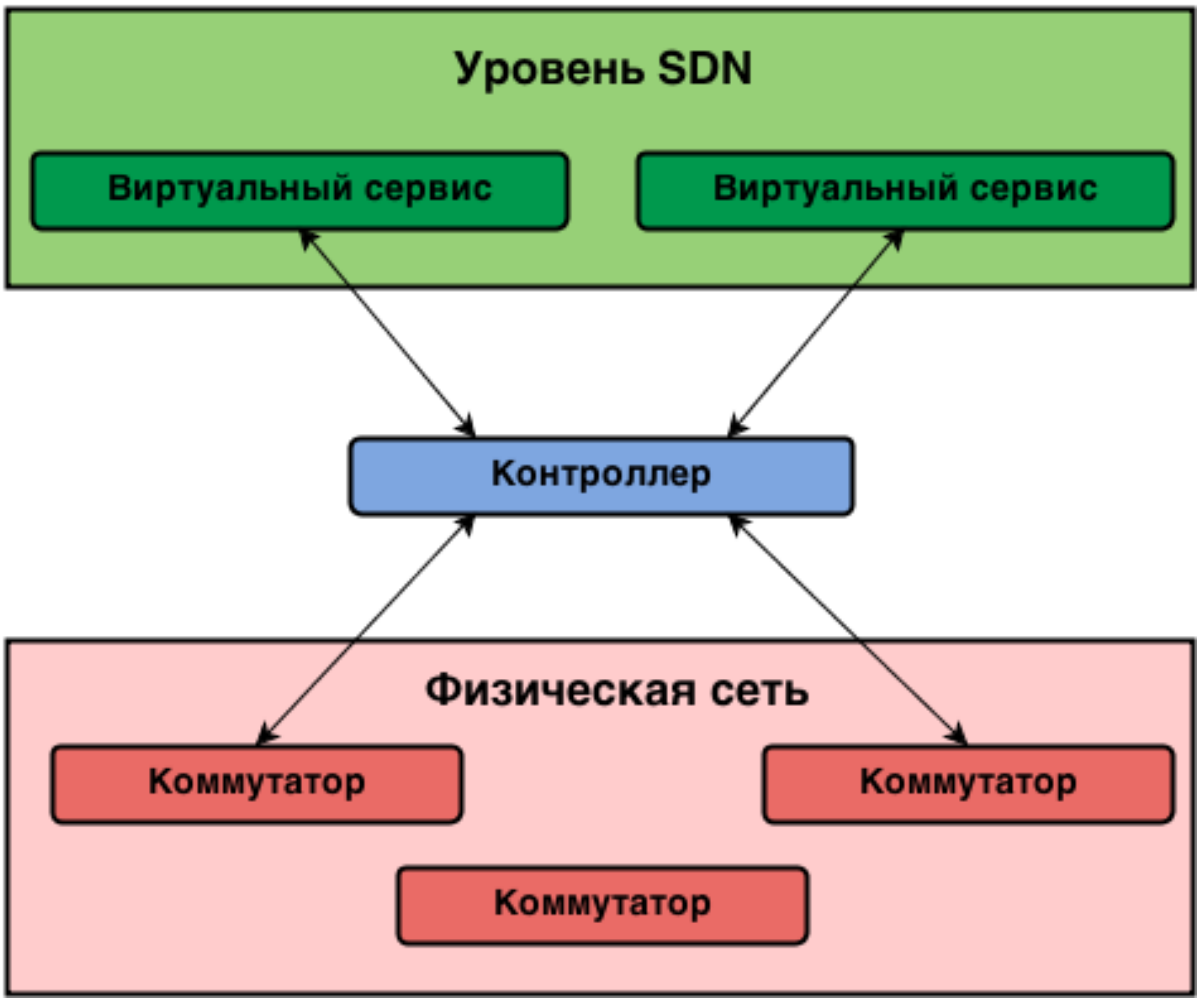

Рис. 1. Контроллер выполняет роль интерфейса между физической сетью и уровнем SDN

Интеллектуальные функции сети сосредоточены в программном сетевом контроллере, который отслеживает общее состояние сетевой инфраструктуры и потоков в ней. Для приложений сеть благодаря такой абстракции выглядит как единый логический коммутатор. В такой концепции управление всей сетью происходит в единой логической точке, что значительно упрощает задачи конфигурации и управления. Кроме того, проще выглядит и функционирование сетевых устройств, потому что в отличие от традиционной модели им не надо больше поддерживать и обрабатывать множество различных протоколов, а достаточно только получать и обрабатывать инструкции от контроллеров SDN.

Очень важным практическим следствием является также то, что для настройки сети достаточно настроить программный контролер сети, вместо того, чтобы изменять сотни строк кода во множестве сетевых устройств сети. Также поведение сети можно изменять в реальном времени, а новые приложения внедрять за гораздо более короткое время, чем в традиционной архитектуре. Централизуя состояние

сети в едином уровне контроля, SDN сети могут конфигурироваться с помощью программных средств. Сетевые контроллеры также обладают набором прикладных интерфейсов, которые позволяют реализовать типичные задачи по маршрутизации, в том числе многоадресной, безопасности, контролю доступа, управлению полосой пропускания, качеством обслуживания, очень тонко настроены под задачи конкретного потребителя.

2. Протокол OpenFlow

Программно-конфигурируемая сеть требует некоторых методов для обеспечения связи между уровнем управления и устройствами передачи данных. На данный момент самым популярным и активно развивающимся является протокол OpenFlow. Последняя версия протокола на момент написания статьи 1.3.2.

Наподобие набора инструкций процессора, OpenFlow определяет базовые примитивы, с помощью которых стороннее приложение может программировать схемы пересылки данных сетевых устройств.

Основная идея протокола OpenFlow проста: большинство современных Ethernet-коммутаторов и маршрутизаторов содержат таблицы потоков, которые используются для реализации брандмауэров, NAT, QoS, а также для сбора статистики. В то же время таблицы потоков разных производителей различаются. OpenFlow использует единый набор функций и предоставляет открытый протокол для программирования таблиц потоков на различных коммутаторах и маршрутизаторах.

Протокол OpenFlow реализуется на обеих сторонах интерфейса между сетевыми устройствами и сетевым программным контроллером. OpenFlow использует концепцию потоков для идентификации трафика, базируясь на заранее определенным правилах сопоставления, которые задаются статически или динамически сетевым контроллером. Таким образом можно контролировать, как трафик проходит через сетевые устройства в зависимости от таких параметров, как особенности использования сети, загрузка, доступные ресурсы облачных и распределенных приложений, доступные ресурсы хранения данных. Соответственно сеть программируется с гранулярностью отдельных потоков.

Простой OpenFlow-коммутатор представляет собой элемент передачи данных, который пересылает пакеты между портами, как определено удалённым контроллером (Рис. 2).

Коммутатор OpenFlow должен содержать не менее трёх частей:

1. Таблица потоков. Она содержит потоки и ассоциированное с каждым потоком действие, которое говорит коммутатору, каким образом обрабатывать поток.
2. Защищённый канал. Используется для передачи пакетов и команд между удалённым контроллером и коммутатором.

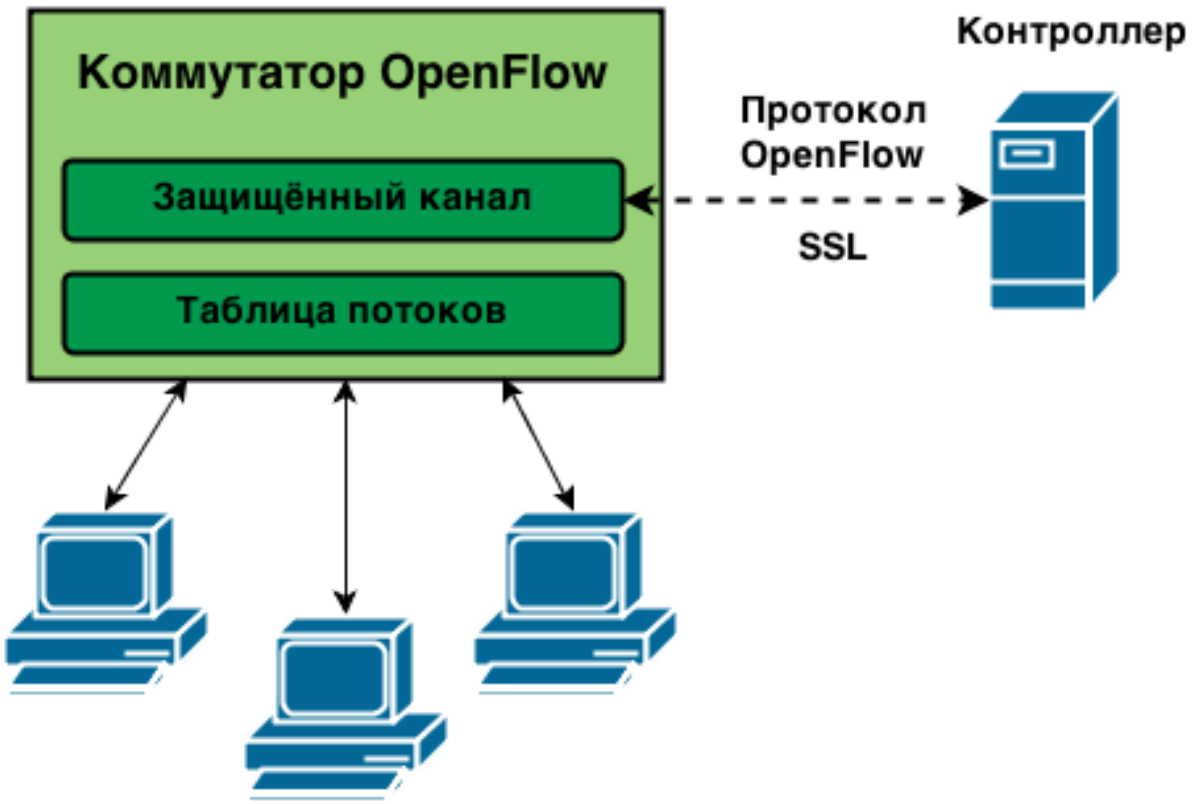

Рис. 2. Коммутатор OpenFlow. Таблица потоков управляется удалённым контроллером через защищённый канал

3. Протокол OpenFlow. Предоставляет открытый и стандартный путь для взаимодействия контроллера с коммутатором.

В таком представлении потоки могут определяться по различным признакам и ограничены только возможностями реализации таблицы потоков. Например, потоком может быть TCP сессия, или все пакеты с определённого MAC или IP адреса, или все пакеты с одинаковым тегом VLAN, или все пакеты с одинаковым номером порта коммутатора.

Каждая запись потока в таблице имеет простое ассоциированное с ней действие. Три основных действия (все коммутаторы OpenFlow должны их поддерживать):

1. Передача пакета, принадлежащего потоку, в указанный порт. Это позволяет маршрутизировать пакеты в сети.

2. Инкапсулирование и передача пакета, принадлежащего потоку, контроллеру. Пакет доставляется через защищённое соединение после того, как инкапсулируется и отправляется контроллеру. Обычно это действие используется для первого пакета нового потока, таким образом, контроллер может принять решение, добавлять поток в таблицу потоков или нет.

3. Отбросить пакет, принадлежащий потоку. Может быть использовано для обеспечения безопасности, чтобы защитить сервисы от атак или уменьшить паразитный широковещательный трафик от конечного хоста.

Запись в таблице потока имеет три поля:

1. Заголовок пакета, который определяет поток.

2. Действие, которое определяет, каким образом должны быть обработаны пакеты.

3. Статистика, которая отслеживает и хранит число прошедших пакетов и переданных байт, а также время, прошедшее с момента появления последнего пакета, попадающего в данный поток (для вычисления и удаления неактивных потоков).

Как вариант быстрого, но не самого эффективного решения, коммерческие коммутаторы и маршрутизаторы могут быть модифицированы для работы по протоколу OpenFlow путём добавления таблицы потоков, защищённого канала и самого протокола OpenFlow. Обычно в качестве таблицы потоков можно использовать аппаратно реализованные таблицы, например TCAM (Ternary Content Addressable Memory); защищённый канал и протокол могут быть портированы в операционную систему устройства.

Другие, более эффективные принципы работы коммутаторов и маршрутизаторов также часто рассматриваются в литературе в привязке к подходу SDN. Так, например, в работе [9] предлагаются усовершенствования, которые могут как затрагивать принципы работы аппаратной логики коммутаторов, так и устранять определенные недостатки протокола OpenFlow. К недостаткам логических схем современных коммутаторов отнесено то, что они позволяют выполнять последовательность совпадение/действие только над крайне ограниченным набором полей. А сам протокол OpenFlow сильно ограничивает набор возможностей по обработке пакетов. Для изменения этого положения авторами работы [9] предлагается механизм RMT (переконфигурируемые таблицы соответствия), который позволит менять логику работы специализированных микросхем коммутаторов, изменяя на ходу характеристики полей, по которым нужно искать соответствие, и предлагая более широкий спектр действий по пакетам, на которых сработало правило. Причем вся эта система может работать на терабитных скоростях без потерь производительности.

3. Направления современных исследований SDN

Несмотря на то что SDN несомненно является новой и достаточно перспективной парадигмой в развитии сетевой архитектуры, однако научная дискуссия в рамках направлений дальнейшего развития SND крайне далека от завершения.

Одно из потенциальных направлений дальнейшей эволюции SDN подходов – гибридизация концепции SDN и MPLS. В своей работе [10] авторы предлагают вернуться к технологии MPLS и использовать ее решения для дальнейшего развития SDN. Считая, что тремя важнейшими требованиями к инфраструктурной части современных сетей является: простота конструкции/логики работы оборудования, что позволяет повышать скорость обработки данных, независимость от модели/марки

оборудования, и готовность к изменениям, чтобы не требовалось замены оборудования при развитии сетевых технологий. К программному же компоненту сетей применяется одно условие – максимальная гибкость и программируемость.

Рассмотрим сеть как систему интерфейсов, считая интерфейсы точками, в которых передается контрольная информация между объектами сетевой инфраструктуры. С такой точки зрения существуют три интерфейса:

1. Узел-сеть, в котором конечные источники данных информируют сеть о своих потребностях.

2. Оператор-сеть, в котором операторы сети (сервис-провайдеры или администраторы корпоративных сетей) информируют сеть о своих потребностях.

3. Пакет-коммутатор, каким образом пакет идентифицируется перед коммутатором (или в более общем случае любым сетевым устройством, осуществляющим форвардинг пакетов).

В традиционной модели сети Интернет сеть просто перемещает пакеты узлов от источника к получателю. Каждый маршрутизатор принимает решение индивидуально по каждому пакету. Поэтому интерфейсы узел-сеть и пакет-коммутатор идентичны, а интерфейс оператор-сеть не формализован.

В технологии сетей MPLS, которая предлагает явное различие между границей сети и ее ядром, имеется различие между интерфейсами узел-сеть и пакет-коммутатор. Тогда как в первом случае этот интерфейс представляет собой IP заголовок пакета, а во втором случае специальную метку, при помощи которой трафик помечается на пограничном маршрутизаторе. Однако технология MPLS не формализует интерфейс оператор-сеть.

В концепции SDN формализации подвергся интерфейс оператор-сеть, в то время как два других интерфейса остались без изменений по сравнению с традиционной архитектурой сети. Протокол OpenFlow обеспечивает контроллеру возможность манипулирования коммутационным/маршрутизирующим оборудованием, используя стандартные компоненты заголовков Ethernet, IP, протоколов транспортного уровня, вызывая либо отправку пакета на определенный порт, либо модификацию этих полей. Т.е. проблемой SDN остается необходимость интерпретации заголовков пакетов сетевыми устройствами. Это в свою очередь не позволяет достичь больших скоростей в обработке пакетов в оборудовании, привязывает сеть к протоколу (например переход с IPv4 на IPv6 приведет к необходимости рассмотрения устройствами других полей заголовка и соответственно необходимость замены оборудования с сети или его апгрейда).

Таким образом, одним из направлений возможной эволюции SDN является разделение интерфейсов узел-сеть и пакет-коммутатор для того, чтобы выдержать требования к «идеальной» инфраструктуре сети. Для этого вводятся понятия ядра и границы сети так же, как с MPLS. В таком случае все три интерфейса являются формализованными и используются каждый на своем месте. При этом устрой-

ства, обеспечивающие функционирование ядра, и граничные устройства управляются при помощи протокола, аналогичного OpenFlow, но отдельно друг от друга, посредством отдельных программных контроллеров, поскольку ядро и граница выполняют разные задачи. Ядро сети отвечает за транспортировку данных (одно- и многоадресная пересылка) и за интеллектуальную политику обслуживания очередей в случае возникновения перегрузки определенных направлений.

Граница сети и устройства, расположенные в ней, отвечают за весь спектр сетевых сервисов, востребованных современными приложениями – изоляцию, безопасность, качество обслуживания. Поэтому пограничные устройства также управляются отдельно от ядра и пользуются сервисами ядра для пересылки трафика.

4. Библиотеки контроллеров OpenFlow

На данный момент существует более десятка библиотек с открытым исходным кодом для создания модульных приложений управления программно-конфигурируемыми сетями. Далее приведено краткое описание наиболее популярных и развивающихся библиотек с открытым исходным кодом на данный момент.

NOX

Самая первая платформа для создания приложений управления сетью, разработана Nicira Networks. В 2008 году Nicira Networks пожертвовала NOX научному сообществу, с тех пор NOX стал основой для многих научно-исследовательских проектов в освоении ПКС. Особенности:

- Предоставляет C++ API для OpenFlow 1.0;
- Быстрый и асинхронный I/O;
- Ориентирован на последние версии Linux дистрибутивов (в частности, Ubuntu 11.10 и 12.04, также Debian и RHEL 6);
- Включает примеры компонентов:
  - обнаружение топологии;
  - обучающийся коммутатор;
  - коммутатор широкого спектра задач (коммутация, простая маршрутизация).

POX

POX – это ответвление от платформы NOX. По своей сути, это платформа для быстрой разработки и прототипирования программного обеспечения управления сетью с помощью Python. POX постоянно развивается. Основная цель проекта – исследовательские задачи. Особенности:

- Предоставляет Python API для OpenFlow 1.0;

- Повторно используемые примеры компонентов (обнаружение топологии, обучающийся коммутатор и т.д.);
- Работает везде, где установлен Python;
- Ориентирован на работу в Linux, Mac OS и Windows;
- Поддерживает одинаковые инструменты визуализации и GUI, как в NOX.

Beacon

Это быстрый, кроссплатформенный и модульный контроллер OpenFlow, который реализован на языке Java. Beacon поддерживает многопоточную, а также асинхронную событийную модель работы. Был создан в 2010 году в университете Стэнфорда, использовался для обучения и проведения исследований, также использовался в качестве основы для платформы Floodlight. Особенности:

- Предоставляет Java API для OpenFlow 1.0;
- Кроссплатформенность;
- Код в Beacon может быть запущен/остановлен/обновлён/установлен во время работы без прерывания работы других не связанных частей кода;
- Имеет опциональную поддержку графического интерфейса.

Floodlight

Floodlight Open SDN Controller – это OpenFlow контроллер корпоративного класса под лицензией Apache, написан на языке Java. Он поддерживается сообществом разработчиков, среди которых инженеры из Big Switch Networks (лидирующая компания в области разработки и исследований ПКС, в её команду входят разработчики OpenFlow из Стэнфордского Университета). Особенности:

- Предоставляет Java API для OpenFlow 1.0;
- Предлагает систему загрузки модулей, что упрощает расширение функционала;
- Лёгкая установка с минимумом зависимостей;
- Поддерживает широкий спектр виртуальных и физических OpenFlow коммутаторов;
- Может обрабатывать совмещённые OpenFlow и не-OpenFlow сети;
- Высокая производительность, является ядром коммерческого продукта Big Switch Network.

Maestro

Это сетевая операционная система, управляющая приложениями контроля сети. Создан в университете Райса. Предоставляет интерфейсы для реализации модульных приложений управления сетью. Особенности:

- Имеет готовые модули для реализации сети с обучающимися коммутаторами или маршрутизаторами на основе OpenFlow коммутаторов;
- Кроссплатформенность;
- Поддержка многопоточности.

Ryu

Ryu – это компонентный фреймворк для построения OpenFlow контроллеров. Предоставляет хорошо продуманное API, что упрощает разработку новых компонентов. Поддерживается и активно разрабатывается этот фреймворк группой NTT laboratories OSRG. Их целью является разработка операционной системы для ПКС, которая будет иметь достаточное качество для использования в большой корпоративной среде. Особенности:

- Предоставляет Python API для OpenFlow 1.0, 1.2 и 1.3;
- Имеет готовые к использованию модули;
- Работает только на Linux-системах;
- Поддерживает OpenvSwitch, что позволяет использовать в виртуальных сетях OpenStack.

Далее приводятся тесты производительности описанных выше платформ. Текущий стандарт оценки производительности OpenFlow контроллеров – это утилита Cbench. Cbench эмулирует OpenFlow коммутаторы, каждый из которых посылает Packet In сообщение на тестируемый контроллер.

Тесты были проведены в Amazon Elastic Computer Cloud с использованием экземпляра Cluster Compute Eight Extra Large, который содержит 16 физических ядер из двух процессоров Intel Xeon E5-2670, 60.5 Гбайт оперативной памяти, использует образ Ubuntu 11.10 VM 64-bit.

Тесты проводились на пропускную способность и латентность. При тестировании пропускной способности каждый эмулируемый коммутатор посылает множество Packet In сообщений, насколько это возможно, таким образом гарантируя, что контроллер всегда будет иметь сообщения для обработки. На Рис. 3 и Рис. 4 показаны результаты тестов для однопоточного и многопоточного режима соответственно.

Тест на латентность использует Cbench, чтобы эмулировать один коммутатор, который посылает один пакет на контроллер, ждёт ответа, затем повторяет этот процесс

настолько быстро, насколько это возможно. Общее число ответов, полученное за период времени, может быть использовано, чтобы вычислить среднее время, которое заняло у контроллера для обработки каждого пакета. Результаты теста показаны на Рис. 5.

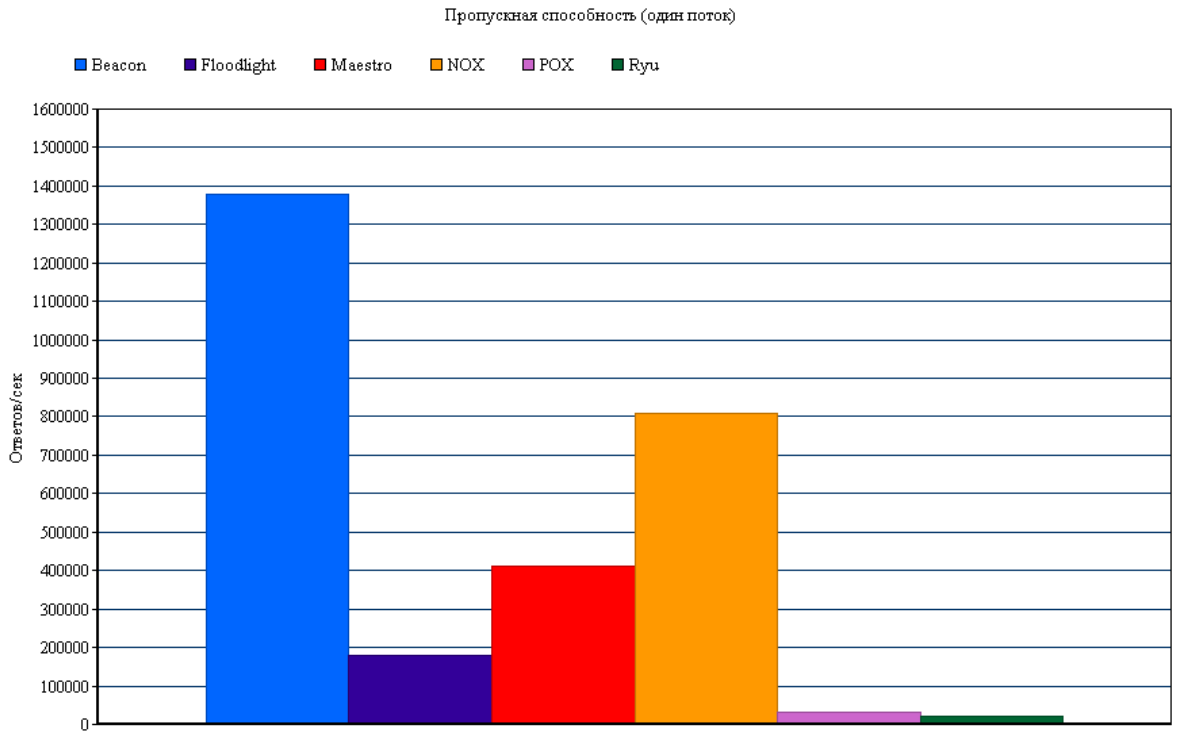

Рис. 3. Тест на производительность в однопоточном режиме

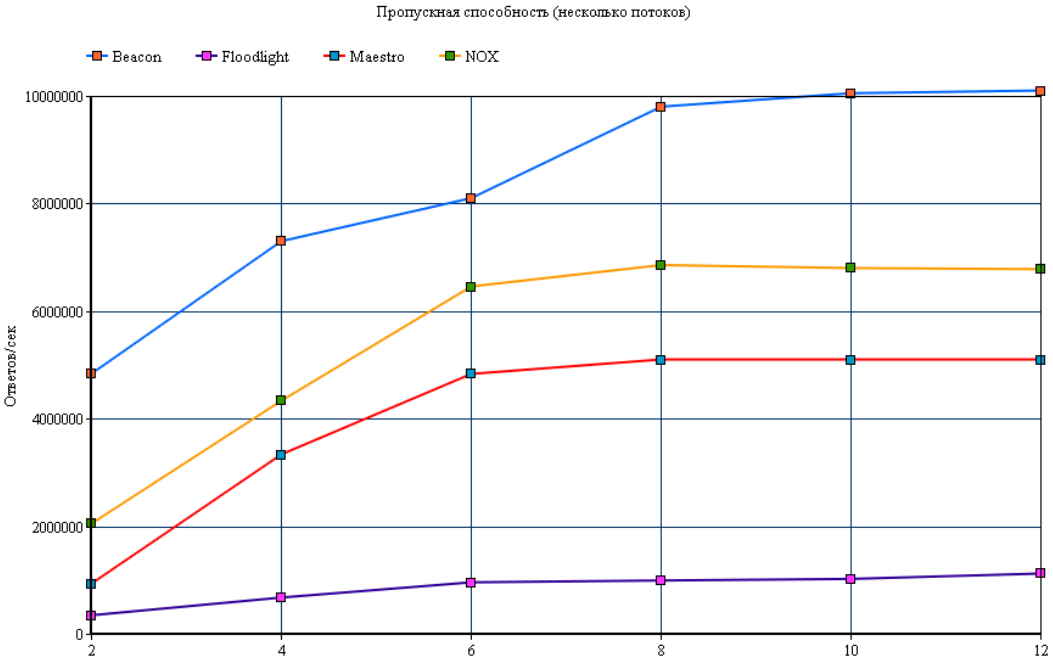

Рис. 4. Тест на производительность в многопоточном режиме

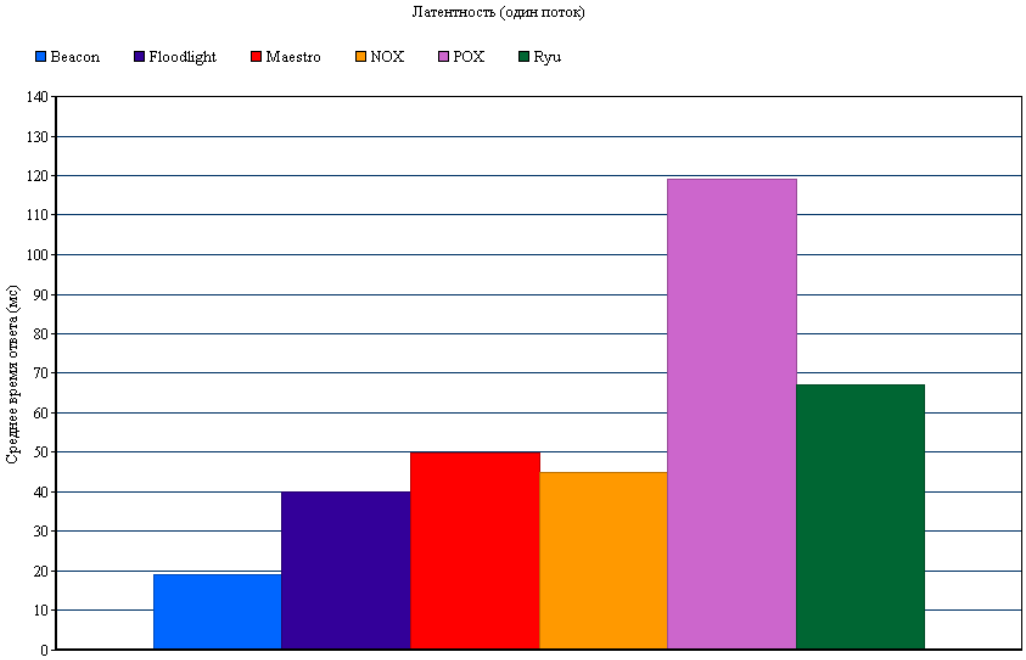

Рис. 5. Тест на латентность

Список литературы

1. OpenFlow Switch Specification, Version 1.1.0 Implemented // www.openflow.org/documents/openflow-spec-v1.1.0.pdf

2. OpenFlow white paper // www.archive.openflow.org/documents/openflow-wp-latest.pdf

3. *McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.* OpenFlow: Enabling Innovation in Campus Networks // SIGCOMM Comput. Commun. Rev. March 2008. Vol. 38. P. 69–74.

4. *Koponen T., Casado M., Gude N., Stribling J., Poutievski L., Zhu M., Ramanathan R., Iwata Y., Inoue H., Hama T.* Onix: A Distributed Control Platform for Large-scale Production Networks. OSDI, 2010

5. *Canini M., Venzano D., Peresini P., Kostic D., Rexford J.* A NICE Way to Test OpenFlow Applications // NSDI, 2012.

6. *Rotsos C., Sarrar N., Uhlig S., Sherwood R., Moore A. W.* OFLOPS: An Open Framework for OpenFlow Switch Evaluation // PAM, 2012.

7. *Kazemian P., Varghese G., McKeown N.* Header Space Analysis: Static Checking For Networks // NSDI, 2012.

8. *Khurshid, W. Zhou, M. Caesar, P. B. Godfrey* VeriFlow: Verifying Network-Wide Invariants in Real Time // HotSDN, 2012.

9. *Bosshart P., Gibb G., Kim Hun-Seok, Varghese G., McKeown N., Izzard M., Mujica F., Horowitz M.* Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN // SIGCOMM '13: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM. P. 99–110.

10. *Casado M., Koponen T., Shenker S., Tootoonchian A.* Fabric: a retrospective on evolving SDN // HotSDN '12: Proceedings of the first workshop on Hot topics in software defined networks, 2012.

Software-Defined Networks as a Stage of the Network Technology Evolution

Krasotin A. A., Alexseev I. V.

Energiya-Info LLC
Soyuznaya str., 144, Yaroslavl, 150008, Russia
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: software-defined networks, SDN, OpenFlow, network controller

The authors of the article focus on the concept of a software defined network. In the beginning, the brief historical account is given concerning software defined networks as a scientific concept, its formation and technological and scientific meaning. The software defined network concept is treated in the article not as the final state-of-the-art in networking, but rather as a possible step and direction in the development of a networking paradigm. The article touches on pros and cons as well of the software defined networking and gives an account of possible stages of development of this technology in the context of other technologies, considering its hybrid with MPLS, as an example. OpenFlow protocol constitutes the main part of the article. The authors further discuss various kinds of existing libraries realizing programmable management routines for a software defined network using OpenFlow. All of these libraries provide API for building modular applications for software defined network management. Touching on practical side of implementation the results of comparative tests of throughput and latency, achieved with these libraries are shown.

Сведения об авторе:
Красотин Артем Александрович,
ООО "Энергия-Инфо", технический руководитель;
Алексеев Игорь Вадимович,
Ярославский государственный университет им. П.Г. Демидова,
канд. физ.-мат. наук, директор Университетского Центра Интернет