Сравнительный анализ SDN-контроллеров

А. А. Семеновых, О.Р. Лапонина

***Аннотация* — SDN (Software Defined Networks) – новая сетевая парадигма, в которой архитектура переходит от традиционной полностью распределенной модели к более централизованному подходу. Этот подход также характеризуется разделением плоскости данных и плоскости управления. Плоскость данных включает в себя элементы, выполняющие пересылку (коммутаторы и маршрутизаторы), а плоскость управления включает контроллер. Контроллер обеспечивает высокий уровень абстракции управления элементами пересылки, который отсутствует в современных сетях. Следовательно, контроллер является фундаментальным компонентом архитектуры SDN, который будет способствовать успеху или провалу SDN. Поэтому необходимо оценивать и сравнивать различные существующие контроллеры на рынке и в исследовательских областях. В данной работе рассматриваются основные понятия SDN-сети или SDN, а также сравниваются существующие контроллеры по выбранным критериям. Контроллеры сравниваются по таким критериям как стоимость, эффективность работы, централизованность/распределенность, а также возможность поддерживать различные протоколы северного (RESTful API, Ad-hoc API) и южного интерфейсов (OpenFlow, OVSDB). Рассмотрены основные характеристики, в том числе языки программирования – C/C++, Java, Python, на которых они написаны, наиболее популярных контроллеров: OpenDaylight, ONOS, NOX, POX, Ryu, Beacon, Onix.**

***Ключевые слова*—SDN, SDN-контроллер, протоколы южного и северного интерфейсов, OpenFlow, RESTful.**

I. Введение

В настоящее время компьютерные сети — стратегический фактор развития современных ИТ [14], однако архитектура сети не всегда способна адекватно и эффективно реагировать на новые потребности. Несколько факторов обуславливают тот факт, что традиционные компьютерные сети имеют недостатки, преодолевать которые становится с каждым годом все сложнее и сложнее. Многие традиционные сети являются иерархическими, но такая статическая архитектура плохо приспособлена к потребностям динамических вычислений и систем хранения. Некоторые из ключевых компьютерных тенденций, определяющих необходимость новой сетевой парадигмы, включают: изменение схемы трафика, рост облачных сервисов, увеличение роли мобильных устройств, увеличение объема данных и т.д.

Более того отечественные телекоммуникационные инфраструктуры сегодня строятся на основе зарубежных средств, а значит, управление инфокоммуникационными сетями в России возможно лишь настолько, насколько это позволяют изделия зарубежных производителей [14].

В [14] представлены следующие проблемы современных компьютерных сетей:

• научно-технические — сегодня невозможно контролировать и надежно предвидеть поведение таких сложных объектов, как глобальные компьютерные сети;

• экономические — сети дороги, сложны и требуют для своего обслуживания высококвалифицированных специалистов;

• проблемы развития — в архитектуре современных сетей имеются существенные барьеры для экспериментирования и создания новых сервисов.

Ответом на кризис компьютерных сетей стало появление принципиально нового подхода к их построению — программно-конфигурируемых сетей (Software Defined Networking – SDN).

SDN – это новая сетевая архитектура, в которой управление сетью отделено от пересылки и напрямую программируется. Такая миграция управления, ранее тесно связанного в каждом сетевом устройстве с доступными вычислительными устройствами, позволяет приложениям и сетевым службам абстрагироваться от базовой инфраструктуры и рассматривать сеть как логическую или виртуальную сущность [10].

Основными идеями при использовании SDN в сравнении с традиционными компьютерными сетями являются следующие:

• разделение процессов передачи и управления данными;

• единый, унифицированный, не зависящий от поставщика интерфейс между уровнем управления и уровнем передачи данных;

• логически централизованное управление сетью, осуществляемое с помощью контроллера с установленной сетевой операционной системой (NOS) и реализованными поверх сетевыми приложениями;

• виртуализация физических ресурсов сети.

Далее будет рассмотрена типовая архитектура SDN.

II. Архитектура SDN

В архитектуре SDN можно выделить три уровня (Рис. 1):

- инфраструктурный уровень (плоскость данных), включающий набор сетевых устройств (коммутаторов и маршрутизаторов);

Статья получена 26 мая 2018.

Работа выполнена при поддержке РФФИ, проект 16-07-00873 А.

А.А. Семеновых – МГУ имени М.В. Ломоносова (e-mail: semenovykh.ann@yandex.ru)

О.Р. Лапонина – МГУ имени М.В. Ломоносова (e-mail: laponina@oit.cmc.msu.ru)

- уровень управления, включающий в себя сетевую операционную систему, которая обеспечивает приложениям сетевые сервисы и программный интерфейс для управления сетевыми устройствами и сетью;
- уровень сетевых приложений для гибкого и эффективного управления сетью

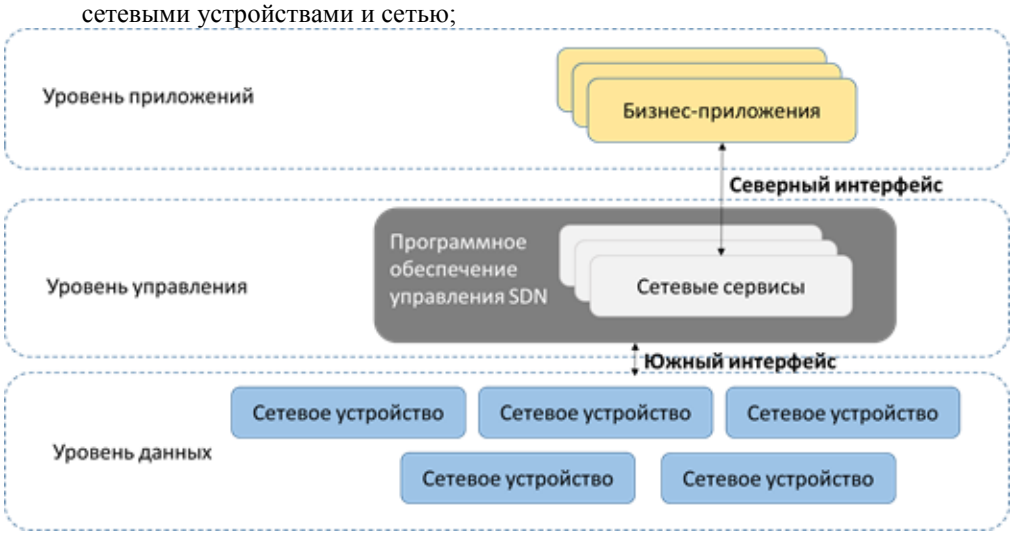

Рис.1 Архитектура SDN

Плоскость данных включает сетевые устройства, которые отвечают за эффективную передачу данных. На этом уровне находится сетевое оборудование, как и в традиционной сети. Однако отличие от традиционной сети в том, что в данном случае устройства, как физические, так и виртуальные, являются лишь средствами для пересылки потока и не могут принимать автономные решения. Способность принимать решения переносится на плоскость управления. Более того SDN строятся (концептуально) поверх открытых и стандартных интерфейсов (например, OpenFlow [1]). Открытые интерфейсы позволяют контроллерам динамически программировать гетерогенные устройства пересылки, что трудно в традиционных сетях из-за большого разнообразия закрытых интерфейсов и распределенного характера плоскости управления.

Плоскость управления использует протоколы, посредством которых заполняются таблицы пересылки в сетевых элементах плоскости данных. Платформа управления включает в себя программные службы, используемые для дистанционного контроля и настройки функций управления. Сетевая политика определена в плоскости управления, плоскость управления реализует политику, и плоскость данных выполняет ее, пересылая данные в соответствии с этой политикой.

В традиционных IP-сетях плоскости управления и данных тесно связаны, встроены в одни и те же сетевые устройства, и вся структура сильно децентрализована. Результатом этого является очень сложная и относительно статическая архитектура. Это также фундаментальная причина, по которой традиционные сети являются жесткими и сложными для управления и контроля.

Преимущества использования SDN достигаются за счет отделения плоскости данных от плоскости управления. Также с разделением плоскостей управления и данных сетевые коммутаторы становятся более простыми устройствами пересылки, а логика управления реализуется в логически централизованном контроллере (или сетевой операционной системе), упрощая применение политик, настройку и эволюцию сети [6].

Отличительной особенностью SDN является также пересылка на основе потока, а не на основе назначения. Поток в целом определяется набором значений полей пакета, действующим как критерий при выборе соответствия (фильтра) и набора действий (инструкций). В контексте SDN поток представляет собой последовательность пакетов между источником и назначением. Все пакеты потока получают идентичные политики обслуживания на пересылающих устройствах, что позволяет унифицировать поведение различных типов сетевых устройств, включая маршрутизаторы, коммутаторы, межсетевые экраны и другие промежуточные сетевые устройства. Гибкость использования потоков ограничивается лишь возможностями реализованных таблиц потоков.

Логика управления сетью перекладывается на контроллер SDN, который также называют сетевой операционной системой, так как его назначение аналогично назначению традиционной операционной системы. Сеть программируется посредством приложений, которые взаимодействуют с NOS, а не с базовыми устройствами пересылки данных.

Взаимодействие плоскости управления и плоскости данных осуществляется с помощью южного интерфейса. Наиболее известным протоколом взаимодействия является OpenFlow, однако помимо него существуют и другие: ForCES, Open vSwitch Database (OVSDB), POF OpFlex, OpenState, revised open-flow library (ROFL), hardware abstraction layer (HAL), programmable abstraction of data path (PAD).

Взаимодействие плоскости управления и плоскости приложений осуществляется с помощью северного интерфейса. Обычно каждый контроллер имеет свой собственный северный интерфейс, так как стандарт для северного интерфейса еще не разработан. Например, такие контроллеры, как Floodlight, Trema, NOX, Onix и OpenDaylight используют собственные северные API.

III. Функции контроллера и коммутатора

Контроллер SDN определяет потоки, которые существуют в плоскости данных. Каждый поток по сети должен сначала быть разрешен контроллером, который проверяет, что поток не нарушает сетевую политику. Если контроллер разрешает поток, он вычисляет маршрут для потока и добавляет запись для этого потока в каждый коммутатор в пути. В отличие от сложных функций, входящих в состав контроллера, коммутаторы просто перенаправляют пакеты в соответствии с таблицами потоков, записи в которых могут быть заполнены только контроллером. Связь между контроллером и коммутаторами использует стандартизованный протокол и API. Чаще всего для этого используется протокол OpenFlow [8].

Архитектура SDN отличается исключительной гибкостью; возможна работа с различными типами коммутаторов и на разных уровнях протокола [11]. Контроллеры и коммутаторы SDN могут быть реализованы для Ethernet-коммутаторов (Layer 2), маршрутизаторов (Layer 3), транспорта (Layer 4) или маршрутизации на уровне приложений. SDN опирается на общие функции, имеющиеся в сетевых устройствах, которые в основном связаны с пересылкой пакетов на основе определения потока.

В архитектуре SDN-коммутатор выполняет следующие функции:

- Коммутатор инкапсулирует и перенаправляет первый пакет потока в контроллер SDN, чтобы контроллер мог решить, следует ли добавить поток в таблицу потока коммутатора.
- Коммутатор перенаправляет входящие пакеты из соответствующего порта на основе таблицы потоков. Таблица потоков может включать информацию о приоритете, продиктованную контроллером.
- Коммутатор может отбрасывать пакеты в определенном потоке, временно или постоянно, как это продиктовано контроллером. Выброс пакетов может использоваться для целей безопасности, сдерживания атак типа отказ в обслуживании (DoS) или требований к управлению трафиком.

Таким образом, контроллер SDN управляет состоянием пересылки коммутаторов в SDN. Это управление осуществляется с помощью API, что позволяет контроллеру удовлетворять самые разнообразные требования приложения без изменения каких-либо аспектов более низкого уровня сети.

Благодаря разделению плоскостей управления и данных SDN позволяет приложениям работать с одним абстрактным сетевым устройством, не заботясь о деталях работы устройства. Сетевые приложения видят в контроллере один API. Таким образом, можно быстро создавать и развертывать новые приложения для организации потока сетевого трафика в соответствии с конкретными корпоративными требованиями производительности и/или безопасности.

IV. Свойства SDN-контроллеров

Для сравнения контроллеров друг с другом можно выделить следующие характеристики:

1. Стоимость/лицензия.

По стоимости контроллеры различаются на две группы: коммерческие и OpenSource.

2. Эффективность.

Эффективность контроллера – это общий термин, используемый для обозначения различных параметров – производительности, масштабируемости, надежности и безопасности. Различные показатели, такие как количество интерфейсов, которые может обрабатывать контроллер, время ожидания, пропускная способность и т.д. определяют то, что мы будем называть производительностью. Аналогично, существуют различные метрики, определяющие масштабируемость, надежность и безопасность. Большая часть работы, проведенной для сравнения контроллеров, учитывает только критерии эффективности.

Также на производительность влияют и поддерживаемые языки программирования. Python, C/C++ и Java являются наиболее часто используемыми языками для программирования контроллеров SDN. В общем, контроллеры, написанные на Java, обладают кросс-платформенностью и хорошей модульностью, написанные на C/C++ контроллеры обеспечивают высокую производительность, но не обладают высокой степенью модульности, хорошим управлением памятью и хорошим графическим интерфейсом, а написанным на Python контроллерам не хватает реальной многопоточной обработки.

3. Централизованность / распределенность.

Централизованный контроллер – это единый объект, который управляет всеми устройствами пересылки сети. Естественно, он представляет собой единую точку отказа и может иметь ограничения масштабирования. Для управления сетью с большим количеством элементов плоскости данных может оказаться недостаточно одного контроллера.

В отличие от централизованного проектирования распределенная NOS может быть масштабирована в соответствии с требованиями потенциально любой среды, от небольших до крупномасштабных сетей. Распределенный контроллер может быть централизованным кластером узлов или физически распределенным набором элементов. Хотя первая альтернатива может предлагать высокую пропускную способность для очень плотных центров обработки данных, последняя может быть более устойчивой к различным видам логических и физических сбоев. Облачный провайдер, который охватывает несколько центров обработки данных, связанных между собой глобальной сетью, может потребовать гибридного подхода с кластерами контроллеров внутри каждого центра обработки данных и распределенными контроллерами на разных сайтах.

При использовании распределенных контроллеров, также возникает понятие западного/восточного интерфейса. Это интерфейсы, позволяющие взаимодействовать различным элементам плоскости управления.

4. Северный интерфейс.

API северного интерфейса используются прикладным уровнем для взаимодействия с контроллером. Они являются наиболее важной частью архитектуры контроллера SDN, поскольку назначение SDN связано с инновационными приложениями. Поскольку они

достаточно критичны, ориентированные на север API должны поддерживать широкий спектр приложений. Эти API должны также позволять подключаться к автоматическим стекам, таким как OpenStack или CloudStack, используемым для управления облаком. Недавно Open Networking Foundation (ONF) сфокусировался на API северного интерфейса SDN после стандартизации южного интерфейса (OpenFlow). Они создали рабочую группу Northbound, которая будет писать код, разрабатывать прототипы и формировать стандарт для северного интерфейса. В настоящее время протокол REST является наиболее часто используемым северным интерфейсом, и большинство контроллеров его реализуют.

От поддержки северного интерфейса зависит и свойство программируемости сети – самое важное преимущество внедрения SDN для решения сложных задач управления в современной сети с большим числом подключенных устройств и развертывания новых услуг.

Приложения могут быть развернуты поверх платформы контроллера для выполнения предопределенных задач и функций управления. Возможности контроллера в программировании сети в основном определяются степенью интеграции большого числа северных интерфейсов, хорошего графического интерфейса пользователя и интерфейса командной строки (CLI).

5. Южный интерфейс и поддержка OpenFlow.

Для взаимодействия контроллера с пересылающими устройствами используются протоколы южного интерфейса (например, OpenFlow, OVSDB, ForCES) и расширения существующих протоколов управления физическими или виртуальными устройствами (например, SNMP, BGP, NetConf). Использование южного интерфейса важно как для обратной совместимости, так и для поддержки гетерогенных сетей. Однако, в настоящее время одним из наиболее популярных протоколов южного интерфейса является OpenFlow. Более того, OpenFlow, является одновременно протоколом между контроллерами SDN и сетевыми устройствами, а также спецификацией логической структуры функций сетевого коммутатора [8].

V. Сравнение SDN-контроллеров

На данный момент имеется более 20 реализаций сетевых ОС для программно-определяемых сетей: NOX, POX, Beacon, Maestro, Trema, BigSwitch, FloodLight и др. [6]. Каждая из реализаций имеет свои особенности.

A. Стоимость

Различные поставщики контроллеров SDN, такие как Brocade, Cisco, HPE, Juniper, NEC и Nuage, не имеют широкого спектра собственных SDN-контроллеров с открытым исходным кодом [13].

Существует множество разрабатываемых SDN-контроллеров с открытым исходным кодом, от POX до Beacon, который является одним из самых популярных. Начатый в начале 2010 года, Beacon является Java-контроллером OpenFlow, лицензированным под комбинацией лицензии GPL v2 и лицензии FOSS License Exception v1.0 Стэнфордского университета. Другие реализации контроллеров SDN включают Trema (Ruby-based от NEC), а также Ryu (поддерживается NTT). Также среди OpenSource контроллеров можно выделить NOX, OpenDaylight, ONOS, POX и пр.

B. Эффективность

Для определения эффективности различных контроллеров проводят тесты. Cbench является наиболее часто используемым инструментом тестирования для контроллеров SDN. По сути Cbench работает в двух режимах: пропускной и латентной. По результатам исследования, представленного в [9], контроллеры, написанные на языке C/C++, показали наивысшую производительность (Mul, Libfluid_msg), уступают в производительности контроллеры, написанные на Java, такие как Beacon, Iris и Maestro. Однако в режиме латентности Maestro, используя адаптивный режим пакетной обработки, выдавал наилучшие показатели задержки. Кроме того контроллеры, написанные на C/C++ и Java, предлагают более высокую производительность при изменении количества потоков. Однако POX, написанный на Python, не показывает существенных различий при использовании многопоточности, потому что поддержка многопоточности в Python не очень эффективна.

C. Централизованность/распределенность

Централизованные контроллеры, такие как NOX-MT, Maestro, Beacon и Floodlight, были спроектированы как высокопроизводительные системы для достижения пропускной способности, требуемой, например, при использовании в ЦОД. Эти контроллеры основаны на многопоточных проектах для изучения параллелизма многоядерных компьютерных архитектур. Например, Beacon может обрабатывать более 12 миллионов потоков в секунду, используя большие вычислительные узлы облачных провайдеров, таких как Amazon [2]. Другие централизованные контроллеры, такие как Trema, Ryu NOS, Meridian и ProgrammableFlow ориентированы на конкретные среды, такие как центры обработки данных, облачные инфраструктуры и сети операторского класса. Кроме того, контроллеры, такие как Rosemary, предлагают определенную функциональность и гарантии защиты и изоляции приложений. Используя контейнерную архитектуру под названием micro-NOS, он достигает своей основной цели – изолировать приложения и предотвратить распространение сбоев по всему SDN-стеку.

ONIX, HyperFlow, HP VAN SDN, ONOS, DISCO, yanc, PANE, SMaRt-Light являются примерами распределенных контроллеров [12], [5]. Большинство распределенных контроллеров предлагают семантику слабой согласованности, что означает, что обновления данных на всех узлах при изменении хотя бы на одном занимают какой-то период времени. С другой стороны, сильная согласованность гарантирует, что все узлы контроллера будут считывать самое последнее значение свойства после операции записи. Несмотря на влияние такого обновления на производительность системы, сильная согласованность предлагает более простой интерфейс для разработчиков приложений. На сегодняшний день только Onix, ONOS и SMaRtLight предоставляют эту модель согласованности данных [5], [7]. Другим общим свойством распределенных контроллеров является отказоустойчивость.

Таким образом, для управления небольшой сетью может быть достаточно одного контроллера, однако он представляет собой единую точку отказа. Аналогичным образом, независимые контроллеры могут быть распределены по сети, каждый из которых управляет сегментом сети, уменьшая влияние отказа одного контроллера. Тем не менее, если доступ к плоскости управления имеет решающее значение, кластер контроллеров может использоваться для достижения большей степени доступности и/или для поддержки большего количества устройств. В конечном счете, распределенный контроллер может улучшить устойчивость и масштабируемость плоскости управления и, например, уменьшить влияние проблем, вызванных сетевым разделением.

Восточно-западные API являются примером интерфейсов, которые необходимы для распределенных контроллеров. В настоящее время каждый контроллер реализует собственный API восток/запад. Функции этих интерфейсов включают в себя импорт/экспорт данных между контроллерами, алгоритмы для моделей согласованности данных и возможности мониторинга / уведомлений.

Подобно южным и северным интерфейсам, API «Восток-Запад» являются важными компонентами распределенных контроллеров. Чтобы определить и обеспечить общую совместимость и совместимость между различными контроллерами, необходимо иметь стандартные интерфейсы восток/запад.

D. Северный интерфейс

Современные контроллеры предлагают широкий спектр северных API, таких как API ad-hoc, API-интерфейсы RESTful, многоуровневые интерфейсы программирования, системы межсетевых экранов и другие специализированные API. Некоторые контроллеры, такие как Floodlight, Trema, NOX, Onix и OpenDaylight предлагают и определяют собственные северные API, однако каждый из них имеет свои специфические интерфейсы [3], [4], [5].

E. Южный интерфейс и поддержка OpenFlow

На более низком уровне платформ управления южные API можно рассматривать как слой драйверов устройств. Они обеспечивают общий интерфейс для верхних уровней, позволяя платформе управления использовать разные южные API (например, OpenFlow, OVSDB и ForCES) и расширения протоколов для управления существующими или новыми физическими или виртуальными устройствами (например, SNMP, BGP и NetConf). Это важно как для обратной совместимости, так и для неоднородности, то есть для обеспечения возможности использования нескольких протоколов и соединителей управления устройствами. Поэтому в плоскости данных соединение физических устройств, виртуальных устройств и устройств с различными интерфейсами могут сосуществовать.

Большинство контроллеров поддерживают только OpenFlow как южный API. Тем не менее, некоторые из них, такие как OpenDaylight, Onix и HP VAN SDN Controller, предлагают более широкий спектр южных API и/или подключаемых модулей протокола. Onix поддерживает протоколы OpenFlow и OVSDB. Контроллер HP VAN SDN имеет другие соединители на юг, такие как L2 и L3.

OpenDaylight выходит за рамки, предоставляя абстракцию уровня сервиса (SLA), которая позволяет нескольким южным API и протоколам сосуществовать на платформе управления. Например, его первоначальная архитектура была разработана для поддержки по меньшей мере семи различных протоколов и подключаемых модулей: OpenFlow, OVSDB, NETCONF, PCEP, SNMP, BGP и LISP Flow. Следовательно, OpenDaylight является одной из немногих управляющих платформ, которые разрабатываются для поддержки более широкой интеграции технологий в единую платформу управления.

VI. Итоги сравнения

Важно отметить, что ONOS и OpenDaylight являются наиболее популярными контроллерами. Эти два написанных на Java контроллера обеспечивают кроссплатформенность и предоставляют высокую модульность, используя контейнер OSGI, который позволяет загружать пакеты во время выполнения. Наследуя мощь Java в программировании графических пользовательских интерфейсов, они предоставляют хорошую возможность графического интерфейса. Будучи в партнерстве с известными сетевыми провайдерами и исследовательскими сообществами, они имеют не только хорошую расширенную документацию, но и стратегию развития с подробным планом релизов. Кроме того, их поддержка распределенной схемы делает их способными проводить развертывание в реальном масштабе SDN. Контроллер ONOS в основном предназначен для сетей операторов связи. Это дает им возможность предоставлять новые услуги SDN наряду с их первоначальными проприетарными услугами.

Архитектура ONOS предназначена для поддержки высокоскоростных и крупномасштабных сетей. Его отличительной особенностью является поддержка гибридных сетей.

OpenDaylight был в основном сосредоточен для использования в центрах обработки данных, но его последний выпуск (Lithium) показал возможность поддержки различных приложений. Были добавлены многие южные и северные интерфейсы, такие как COAP, PCEP, LACP, OpFlex, SNMP и т.д. Также были реализованы новые модули, такие как IoT data broker (IoTDM), унифицированный безопасный канал USC и т.д. Таким образом, это первый контроллер, который можно использовать для IoT.

Ryu в основном предназначен для малых предприятий и исследовательских приложений. Будучи написанным на Python, этот контроллер предоставляет средства для разработки приложений и модулей. Однако отсутствие хорошей модульности и невозможность запуска на других платформах ограничивают его широкое использование в реальных промышленных приложениях.

Следует заметить, что API, ориентированные на север, очень разнообразны.

В частности, пять контроллеров (Onix, Floodlight, MuL, Meridian и SDN Unified Controller) уделяют

больше внимания этому интерфейсу, как подтверждение его важности.

В таблице 1 приведена сравнительная информация по контроллерам. Как видно из таблицы, контроллер OpenDayligt имеет наиболее широкий набор как северных, так и южных интерфейсов. Более того, OpenDaylight является свободно распространяемым, распределенным и за счет поддержи языка Java многоплатформенным.

Таблица 1 Сравнительная таблица SDN-контроллеров

Контроллер	***Стоимость***	***Язык программирования***	***Централизованность***	***Южный интерфейс***	***OpenFlow***	***Северный интерфейс***
ONOS	OpenSource	Java	Распределенный	NETCONF	1.0, 1.3	RESTful API
OpenDaylight	OpenSource, EPL v1.0	Java	Распределенный	NETCONF/YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP	1.0, 1.3, 1.4	REST, RESTCONF
NOX	OpenSource, GPLv3	C++	Централизованный		1.0	Ad-hoc API
POX	OpenSource, GPL v3	Python	Централизованный		1.0	Ad-hoc API
Ryu	OpenSource, Apach 2.0	Python	Централизованный многопоточный	NETCONF, OFCONFIG	1.0, 1.3, 1.4	Ad-hoc API
Beacon	OpenSource, GPL v2	Java	Централизованный многопоточный		1.0	Ad-hoc API
Onix	Коммерческий	Python, C	Распределенный	OVSDB	+	NVP NBAPI

VII. Заключение

В данной работе были рассмотрены основные понятия SDN сетей, а также по выбранным критериям был проведен сравнительный анализ SDN-контроллеров.

Основная особенность SDN-сетей заключается в разделении плоскости управления от плоскости данных. В качестве плоскости данных выступает SDN-контроллер, на который переносится вся нагрузка по управлению потоком с маршрутизаторов и коммутаторов, в отличие от традиционных сетей. Поэтому плоскость управления является одним из критических для успеха SDN.

Библиография

[1] Braun W., Menth M. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices //Future Internet. – 2014. – Т. 6. – №. 2. – С. 302-336.

[2] Erickson D. The beacon openflow controller //Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. – ACM, 2013. – С. 13-18.

[3] Floodlight OpenFlow Controller -Project Floodlight URL: http://www.projectfloodlight.org/floodlight/

[4] Home – OpenDaylight URL: https://www.opendaylight.org/

[5] Koponen T., Casado M., Gude M. Onix: A Distributed Control Platform for Large-scale Production Networks URL: http://yuba.stanford.edu/~casado/onix-osdi.pdf

[6] Kreutz D. et al. Software-defined networking: A comprehensive survey //Proceedings of the IEEE. – 2015. – Т. 103. – №. 1. – С. 14-76.

[7] ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out. URL: https://onosproject.org/

[8] OpenFlow Switch Specification Version 1.4.0 (Wire Protocol 0x05). – 2013. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[9] Salman O. et al. SDN controllers: A comparative study //Electrotechnical Conference (MELECON), 2016 18th Mediterranean. – IEEE, 2016. – С. 1-6.

[10] Software-Defined Networking: The New Norm for Networks ONF White Paper. – 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[11] Stallings W. Software-defined networks and openflow //The internet protocol Journal. – 2013. – Т. 16. – №. 1. – С. 2-14. URL: http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html

[12] Tootoonchian A., Ganjali Y. HyperFlow: A Distributed Control Plane for OpenFlow // Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. – 2010. URL: http://dl.acm.org/citation.cfm?id=1863133.1863136

[13] What are SDN Controllers (or SDN Controller Platforms)? URL: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/

[14] Смелянский Р. Л. Программно-конфигурируемые сети //Открытые системы. СУБД. – 2012. – Т. 9. – С. 23-26. URL: https://www.osp.ru/os/2012/09/13032491/

Comparative analysis of SDN controllers

Anna A. Semenovykh, Olga R. Laponina

Abstract **- SDN (Software Defined Networks) is a new network paradigm in which the architecture moves from a traditional fully distributed model to a more centralized approach. This approach is also characterized by the separation of the data plane and the control plane. The data plane includes the elements that perform the forwarding (switches and routers), and the control plane includes the controller. The controller provides a high level of abstraction control for the forwarding elements, which is not present in modern networks. Consequently, the controller is a fundamental component of the SDN architecture, which will contribute to the success or failure of the SDN. Therefore, it is necessary to evaluate and compare the various existing controllers in the market and in research areas. In this paper, we discuss the basic concepts of SDN-network or SDN, and compare existing controllers according to the selected criteria. Controllers are compared by such criteria as cost, efficiency, centralization / distribution, as well as the ability to support various protocols of the northbound (RESTful API, Ad-hoc API) and southbound interfaces (OpenFlow, OVSDB). The main characteristics are considered, including programming languages - C / C ++, Java, Python, on which they are written, the most popular controllers: OpenDaylight, ONOS, NOX, POX, Ryu, Beacon, Onix.**

Keywords **- SDN, SDN-controller, the protocols of the southbound and northbound interfaces, OpenFlow, RESTful.**