Министерство образования и науки Российской Федерации
Московский физико-технический институт
(государственный университет)

ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ
КАФЕДРА ИНФОРМАТИКИ

Программно-конфигурируемые сети

Выпускная квалификационная работа
студента 4 курса 073а группы
Гончарова Федора Олеговича

Научный руководитель:
к. ф.-м. н. Емельянов Павел Владимирович

Долгопрудный,2014г.

Содержание

1 Введение

Термин *программно-конфигурируемые сети*[9] относительно не новый, первые работы и исследования по данной тематике начали появляться примерно с 1995 года[1]. Это связано с увеличением потребностей в использовании сетевых технологий для различных бизнес-задач, распределенных вычислений, появлении крупных дата-центров и началом виртуализации интернета[2].

Правильная настройка сети в крупном дата-центре и её поддержка - весьма сложный и дорогостоящий процесс. В задачу системного администратора входит настройка маршрутизаторов, балансировщиков нагрузки(load balancers), системы обнаружений вторжений(IDS) и.т.д.

Обычно администратор настраивает каждый сетевой элемент отдельно, причем зачастую у сетевого оборудования нет какого-либо унифицированного интерфейса для настройки. Сразу же можно сказать о проблемах возникающих в данном подходе: если необходимо настроить например не 10 или 20 маршрутизаторов, а например 20 000 или более(такие задачи весьма актуальны в облачных технологиях) - без автоматизации это сделать почти невозможно; вторая проблема заключается в зависимости интерфейса настройки от производителя сетевого оборудования(вендора), иногда интерфейс зависит даже от модели у одного и того же вендора – следовательно специалист в области сетевых технологий должен уметь работать с большим количеством протоколов, кол-во которых только возрастает.

Выше были перечислены лишь две наиболее очевидные проблемы в отрасли сетевых технологий, но все они являются следствием того, что софт, операционные системы, языки программирования быстрее следуют различным инновациям и современным требованиям, чем компьютерные технологии.

Программно-конфигурируемые сети(Software Developed Networks) – изменяют подход к дизайну и поддержке сети, как раз при помощи достижений в области ПО и операционных систем.

Программно-конфигурируемые сети(ПКС, SDN) – имеют две отличительные особенности:

- Разделение управления и передачей данных(*data and control plane separation*)
- Консолидация управления - т.е. управление множеством элементов уровня передачи данных при помощи специального приложения

Уровень управления передачей данных(control plane) – определяет по каким правилам будет пересылаться трафик в сети.
Уровень передачи данных(data plane) – непосредственно пересылает данные в соответствии с настройкой *control plane*.

Стоит отметить, что большинство современных маршрутизаторов обладают собственной операционной системой и локально решают задачу о направлении пересылки трафика. Маршрутизаторы, поддерживающие технологию *ПКС*(например *Open vSwitch*[5]) как правило, на практике не имеют собственной операционной системы и содержат лишь таблицу правил пересылки пакетов(data plane), которая в свою очередь настраивается при помощи специального приложения-контроллера.

В целом можно сказать, что *ПКС* – это технология, которая предоставляет способ программного управления(API) уровнем передачи данных роутеров, маршрутизаторов и другого сетевого оборудования. Один из ярких примеров такого API – *OpenFlow*[3][4] протокол настройки таблиц маршрутизации(более подробно он будет рассмотрен в данной работе).

Цель данной работы, обзор основных современных элементов *ПКС*, решение простейших задач конфигурирования, анализ уже существующих программных решений-контроллеров.

2 Обзор.

В данной работе последовательно рассматриваются основные элементы *data-plane* и *control-plane ПКС*:

- *OpenFlow Switch* – требования предъявляемые протоколом *OpenFlow* к маршрутизаторам
- *Open vSwitch* – виртуальный маршрутизатор, поддерживающий протокол *OpenFlow*(в описанной выше модели является примером элемента уровня *data-plane*)
- набор современных приложений-контроллеров (*POX-controller*[6], *Floodlight Project*[7], *OpenDaylight Project*[8]) - с помощью которых выполняется настройка сетевого устройства(пример элемента уровня *control-plane*).

Понимание основных принципов работы протокола *OpenFlow* облегчает понимание всей структуры *ПКС*; в спецификации указаны требования к сетевому оборудо-

ванию, которое поддерживает данный протокол, именно поэтому изложение начинается с этой темы.

Далее идет обзор *Open vSwitch* – как станет видно, виртуальный маршрутизатор соответсвует спецификации OpenFlow с некоторыми расширениями и дополнениями(в данной работе эти дополнения не будут рассмотрены), поэтому подробно мы на нем останавливаться не будем. Будут лишь рассмотрены некоторые классические задачи конфигурирования, такие как: *зеркалирование трафика*, *настройка QoS*, *настройка GRE туннелирования*. Данные простейшие задачи входили в цикл задач на ручное конфигурирование Open vSwitch.

Обзор *control-plane* представляет собой анализ и сравнение наиболее известных реализаций контроллеров.

Раздел *Постановки задач* и *Сравнение производительности* содержат графики тестов на пропускную способность и частоту транзакций и решение задачи о безопасной рассылке ARP запросов, реализованного в рамках данной работы.

3 Требования к OpenFlow Switch.

3.1 Обзор.

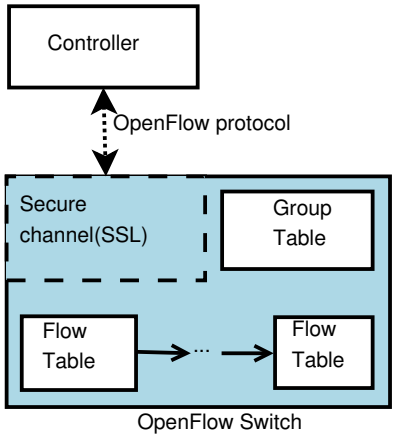


Рис. 1: Структура OpenFlow маршрутизатора(www.opennetworking.org)

OpenFlow маршрутизатор состоит из одной или нескольких *flow-таблиц*(*flow-table*), *групповой таблицы*(*group-table*) и *OpenFlow канала*(*OpenFlow channel*) к удаленному контроллеру(*controller*). Маршртизатор обменивается сообщениями с контроллером при помощи протокола OpenFlow.

Используя данный протокол, контроллер может *добавлять*, *обновлять* и *удалять flow-записи*(*flow-entries*) во flow-таблицах, как *реактивно(reactive)(ответы на входящие пакеты)*, так и проактивно(proactive). Каждая flow-таблица содержит набор flow-записей; каждая запись определяется *полями сравнения*(*matching fields*), *счетчиками*(*counters*) и набором *инструкций*(*instructions*), которые применяются к пакету с совпавшими полями.

Сравнение начинается с первой flow-таблицы и может продолжаться в других таблицах. Поля сравнений flow-записей сравниваются с заголовком пакета в порядке приоритета(*priority* – одно из полей сравнения). Если найдена совпадающая flow-запись, к пакету применяются инструкции(*instructions*) ассоциированные с данной записью. Если не найдено ни одной записи, результат зависит от конфигурации маршрутизатора(пакет может быть сброшен(dropped) либо передан контроллеру по OpenFlow каналу для анализа и принятия решения).

Инструкции в соответствующих записях содержат либо *действия*(*actions*), которые применяются к пакету и определяют непосредственно сами правила пересылки, либо определяют дальнейшую обработку пакета, в последующих flow-таблицах(конвейерная обработка, (*pipeline processing*)). Конвейерная обработка позволяет передавать анализ пакета из одной таблиц в другую, дополнительно пересылая спецаильные метаданные(например числовые переменные, связанные с предыдущей обработкой). Такой подход позволяет создавать “ветвление” в обработке трафика. Обработка заканчивается, когда соответствующие инструкции не содержат команд по пересылки в следующую таблицу; в этот момент выполняется модификация пакета, выполнение “накопленных” действий(*actions*).

3.2 Порт OpenFlow.

Порт – сетевой интерфейс, для передачи пакетов между OpenFlow обработкой и остальной частью сети. OpenFlow маршрутизатор должен поддерживать три типа OpenFlow портов: *физический порт*(*physical port*), *логический порт*(*logical port*), *зарезервированный порт*(*reserved port*).

- Физический порт – порт который соответствует аппаратному сетевому интерфейсу(hardware).
- Логический порт – порт, который не обязательно соответствуют физическо-

му интерфейсу. Логический порт представляет собой более высокую абстракцию и может быть определен без использования OpenFlow(например: туннели, loopback интерфейсы). Единственное различие между *физическим портом* и *логическим* – пакет пересылаемый через данные порты в заголовке может иметь дополнительное поле *Tunnel-ID*, и когда пакет приходит с логического порта он пересылается на контроллер.

- Зарезервированный порт – представляет собой специальное действие по пересылки пакета:
  - **all** – представляет собой все порты, на которые может быть продублирован пакет(может быть использован, только как output port)
  - **controller** – канал связи с OpenFlow контроллером(может быть использован как для входящих, так и для исходящих пакетов)
  - **table** – действие по передачи обработки пакета в другую таблицу(см.выше)
  - **in_port** – представляет собой ID порта входящего пакета

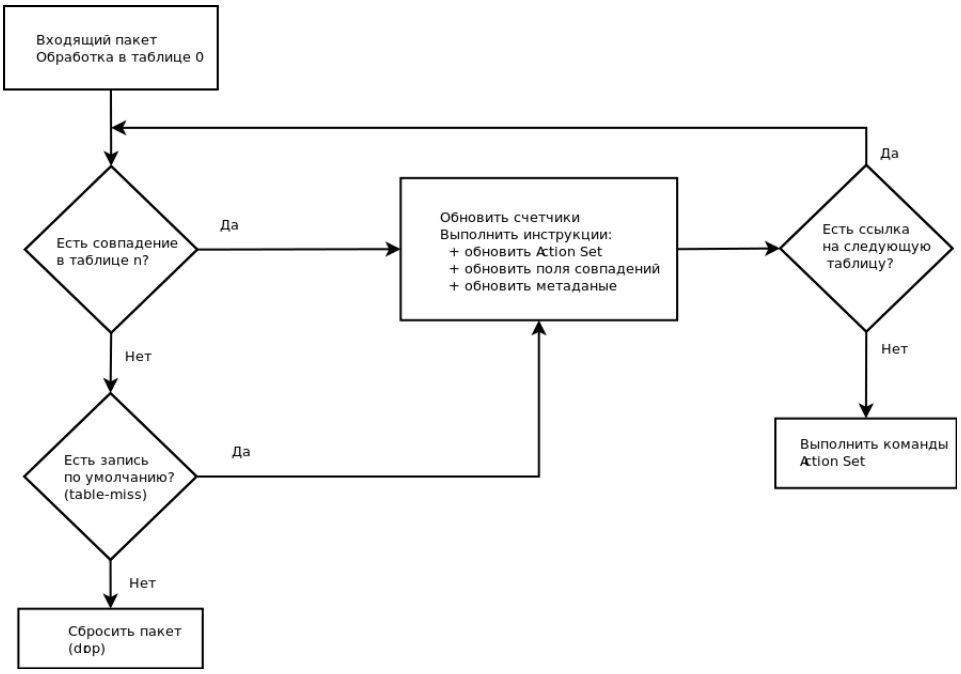
3.3 OpenFlow таблица.

Flow-таблица состоит flow-записей, где каждая состоит из:

Поля сравнения	**Приоритет**	**Счетчики**	**Инструкции**	**Временные метки**

- **Поля сравнения**(**match fields**) – параметры содержащиеся в заголовке пакета(данные L2-L4), а также некоторые OpenFlow данные(номер входящего порта, метаданные обработки процесса обработки пакета)
- **приоритет**(**priority**) – числовое поле от 0 до 65535, среди flow-записей совпавших с заголовком пакета, выбрана будет одна с большим приоритетом.
- **счетчики**(**counters**) – хранят статистические данные о работе записи(например сколько пакетов было обработано в соответствии с данной записью)
- **инструкции**(**instructions**) – набор команд, которые необходимо выполнить над пакетов в процессе обработки(см.Конвейер обработки пакетов)
- **временные метки**(**timeouts**) – максимальное время до удаление соответствующей flow-записи

Соответствующая запись определяется по полям сравнения и приоритету: среди записей, у которых совпали поля выбирается уникальная с наибольшим приоритетом.



Анализ начинается с нулевой таблицы(см.рисунок), из пакета извлекаются данные заголовка. Далее проходит сравнение этих данных со всеми полями во flow-таблице(стоит отметить, что поиск идет не только по заголовку пакета, но например по номеру входящего порта и специальным метаданным, которые могли быть получены на предыдущих этапах обработки)(если подходит несколько записей с одинаковым приоритетом, будет выбрана только одна запись, причем выбор конкретной записи неопределен).

После совпадения, записи инструкции добавляются в *Action Set*(своеобразный накопитель действий, которые будут выполнены после окончания анализа пакета)(есть возможность выставить бит `OFPFF_CHECK_OVERLAP` для запрета использования совпадающих правил).

3.4 Конвейер обработки пакетов.

OpenFlow – совместимые маршрутизаторы можно разедить на два типа: *только-OpenFlow*(*OpenFlow-only*) и *гибридные*(*OpenFlow-hybrid*).

Гибридные маршрутизаторы – поддерживают как и протокол OpenFlow так и стандартную Ethernet маршрутизацию, например L2 Ethernet маршрутизация(L2 Ethernet switching), VLAN, L3 маршрутизацию(L3 routing)(IPv4 routing, IPv6 routing...),

ACL, QoS. Такие маршрутизаторы должны иметь механизм отделения обработки пакетов по конвейеру OpenFlow или же по стандартными способами(в данной работе, эти механизмы не рассматриваются).

OpenFlow-конвейер состоит из множества flow-таблиц, в каждой таблицы множество flow-записей. Спецификация протокола OpenFlow требует, что маршрутизатор обязан иметь хотя бы одну flow-таблицу, хотя может иметь и несколько.

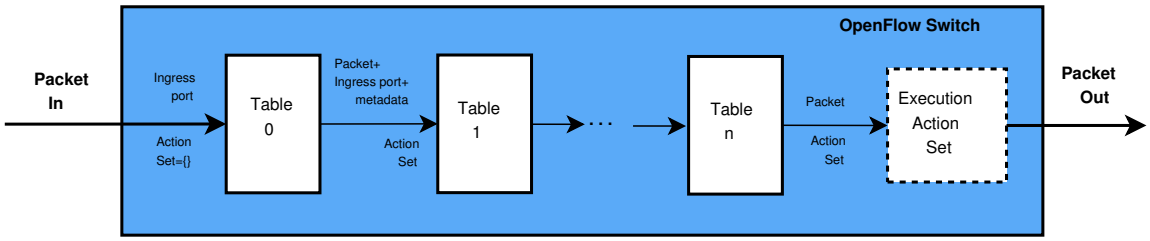


Рис. 2: Конвейер обработки сетевых пакетов(www.opennetworking.org)

Flow-таблицы последовательно пронумерованы начиная от 0. Обработка всегда начинается с начальной таблицы: заголовок пакета сравнивается с полями flow-записей в таблице 0. Если найдена соответствующая flow-запись, выполняются инструкции, определенные в данной записи(одна из возможностей – передача пакета в другую таблицу, где данный процесс повторяется заново). Для дальнейшей обработки пакет можно передавать только в таблицу с большим порядковым номером, чем текущий. В последней таблице, в качестве инструкций запрещено использовать передачу далее по конвейеру(разрешен только список непосредственных действий над пакетом). Если в найденной flow-записи нет инструкций передачи пакета далее по конвейеру, обработка на этом месте заканчивается и к пакету применяются действия, добавленные в *action set* в процессе обработки.

Если ни одна запись не подходит для обрабатываемого пакета, генеририруется событие *table miss*. Действие примененое к пакету зависит от статической конфигурации маршрутизатора(можно сбросить пакет, отправить на контроллер или отправить на обработку в специальную таблицу).

4 Open vSwitch

4.1 Структура Open vSwitch.

Open vSwitch – это виртуальный *многоуровневый*(*multilayer*) маршрутизатор разрабатываемый под лицензией Apache 2.0.

Open vSwitch условно можно разделить на две части – *user-space* и *kernel-space*.

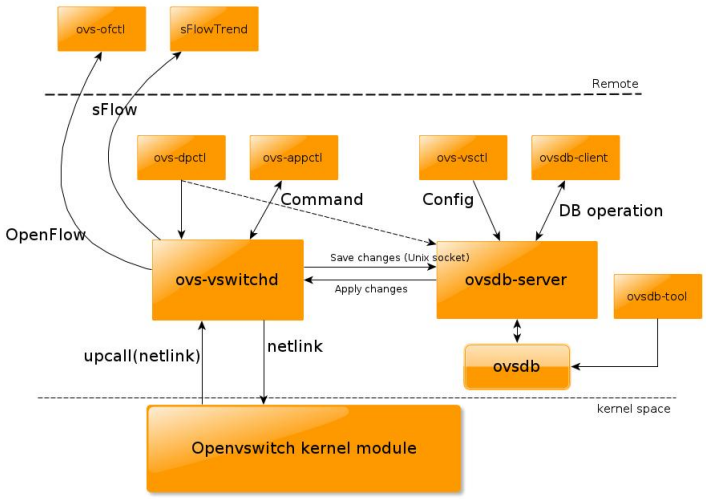


Рис. 3: Структура Open vSwitch(www.openvswitch.org)

User-space состоит из нескольких наиболее важных компонент: это демоны(daemons), которые реализуют свитч с таблицей потоков, ядро Open vSwitch, и набор утилит, которые позволяют конфигурировать свитч, его базу данных(ovsdb) и напрямую посылать сообщения в ядро.

Openvswitch service запускает три демона: **ovs-vswitchd** – сохраняет и отвечает за изменение конфигурации маршрутизатора, а также сохраняет состояние в базу данных(ovsdb), **ovsdb-server** – манипулирует базой данных, конфигурацией и набором потоков, **ovs-brcompatd** – создает совместимость Open vSwitch с обычными Linux мостами(которые создаются командой brctl).

Демон ovs-vswitchd – единственная компонента, связанная с kernel-space через протокол netlink, также меняет конфигурацию Open vSwitch и сохраняет ее в базу данных(ovsdb), которая управляется при помощи демона ovsdb-server(коммуникация ovs-vswitchd и ovsdb-server осуществляется при помощи сокетов)

- **ovs-dpctl** – позволяет напрямую обращаться к kernel-space, без сторонних обращений к базе данных.
- **ovsdb-client** – утилита для конфигурирования базы данных ovsdb.
- **ovs-ofctl** – команда для работы с протоколом OpenFlow, модификацией flow-таблиц, настройки соединения с удаленным контроллером.

Стоит отметить, что Open vSwitch является **гибридным** маршрутизатором – помимо поддержки OpenFlow, маршрутизатор поддерживает специализированные команды(Nicira Extensions)(L2 routing, QoS, tunneling и.т.д), но наибольший интерес он представляет как проект поддерживающий протокол OpenFlow(на данный момент есть поддержка OpenFlow 1.0 и тестовые версии с поддержкой OpenFlow 1.3).

4.2 Настройка QoS.

В некоторых случая администратору необходимо ограничить пропускную способность выделенных виртуальных машин, в частности, когда различные пользователи используют одно и то же виртуальное пространство.

Open vSwitch дает простую возможность ограничения максимальной скорости передачи выбранного интерфейса. Пропускная способность до установления QoS:

```
[root@localhost /]# netperf -H 192.168.0.2
MIGRATED TCP STREAM TEST from 0.0.0.0 () port 0 AF_INET to 192.168.0.2 ()
port 0 AF_INET
Recv Send Send
Socket Socket Message   Elapsed
Size Size Size   Time   Throughput
bytes bytes bytes   secs   10^6bits/sec

87380 16384 16384   13.07   1382.34
```

Выставляются ограничения пропускной способности(inrgress_policing_rate – максимальная скорость передачи в килобайтах в секунду, через выбранный интерфейса, inrgress_policing_burst – максимальный размер данных(в килобайтах), которые можно передать с превышением пропускной способности):

```
[root@localhost /]# ovs-vsctl set Interface <target interface> \
ingress_policing_rate=1000
[root@localhost /]# ovs-vsctl set Interface <target interface> \
ingress_policing_burst=100
```

Тест пропускной способности после установления ограничений:

```
[root@localhost /]# netperf -H 192.168.0.2
MIGRATED TCP STREAM TEST from 0.0.0.0 () port 0 AF_INET to 192.168.0.2 ()
port 0 AF_INET
Recv Send Send
Socket Socket Message   Elapsed
Size Size Size   Time   Throughput
bytes bytes bytes   secs   10^6bits/sec

87380 16384 16384   12.31   0.99
```

Приведенный выше способ ограничения пропускной способности не буферизует пакеты, а сбрасывает, при превышении пропускной способности.
Есть также возможность применять политику QoS на выделенный порт. Для этого создается буфер(queue), который и влияет на скорость передачи определенных пакетов на выбранный порт(например в следущей конфигурации, будет введено ограничение 2Мб/с на порт eth1):

```
1) Создание буфера(q0)
q0 (2 Mbps):
ovs-vsctl --id=@q0 create queue other-config:min-rate=2000000 \
other-config:max-rate=2000000

2) Создание QoS(newqos) и добавление новой очереди в таблицу QoS
ovs-vsctl --id=@newqos create qos type=linux-htb queues=0=@q0

3) Добавление QoS (newqos) к выбранному порту (eth1)
ovs-vsctl set port eth1 qos=@newqos
```

Следует также отметить, что на выбранный порт можно определить несколько очередей, и таким образом гарантировать различное качество обслуживания для разных видов трафика.

4.3 Зеркалирование трафика.

Для использования системы обнаружения вторжения, необходимо использовать зеркалирование траффика на порт для анализа. Для настройки зеркалирования необходимо создать порт-зеркало(mirror-port) и добавить его в соответствующий маршрутизатор:

```
[host]# ovs-vsctl create mirror name=mirror select_all=1 output_port=port_id
```

Для того, чтобы узнать ID порта, который необходим:

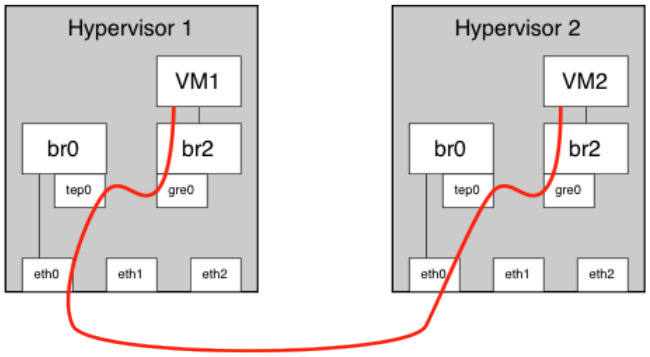
```
[host]# ovs-vsctl list port <IDS target port>
```

Добавление зеркала:

```
[host]# ovs-vsctl add bridge extern0 mirrors=mirror_id
```

4.4 Настройка GRE туннелирования.

Open vSwitch – позволяет использовать GRE туннелирование между хостами, как способ инкапсуляции трафика и создание оверлейных сетей(overlay networks). На следующей схеме показано, как настраивать GRE туннель между двумя виртуальными машинами:



Процесс настройки состоит из трех частей:

- Создать изолированный мост для виртуальной машины
- Создать конечные точки GRE туннеля на каждом из гипервизоров.
- Добавить GRE интерфейс и инициализировать туннель.

Создание изолированного моста Open vSwitch(его можно назвать изолированным, т.к. он не содержит реальных физических интерфейсов):

```
[hypervisor1]# ovs-vsctl add-br br2
```

Смысл использования “изолированного моста” и отдельного GRE интерфейса в том, чтобы отделить системный трафик гипервизора от трафика GRE, что в свою очередь отделяет управление сетью для виртуальной машины, от особенностей сети хоста, на котором запущен гипервизор. Для этого создается внутренний интерфейс и ему присваивается IP адрес:

```
[hypervisor1]# ovs-vsctl add-port br0 tep0 -- set Interface tep0
type=internal
[hypervisor1]# ifconfig tep0 192.168.200.20 netmask 255.255.255.0
```

На втором гипервизоре прописываются те же команды, выставляется другой IP адрес интерфейса tep0. После этого в маршрутизатор добавляется порт GRE туннеля:

```
[hypervisor1]# ovs-vsctl add-port br2 gre0 -- set interface gre0 type=gre \
options:remote_ip=<GRE tunnel enpoint on other hypervisor>
```

После данных команд на каждом из гипервизоров, между маршрутизаторами инициализирован GRE туннель.

5 OpenFlow-контроллер.

В данной главе будет проведено сравнение современных OpenFlow контроллеров. Основная идея использования ПКС, это унифицирование интерфейсов конфигурирования сетевых устройств, а также предоставление удобного API для рядового программиста для работы с сетью(соответственно данное API позволяет создавать приложения, использующие полный спектр возможностей сети). Для полноты изложение введем два простейших термина:

- Southbound API – программный интерфейс, который обеспечивает связь между control-plane и сетевыми устройствами(OpenFlow яркий пример такого API)
- Northbound API – програмный интерфейс, предоставляемый контроллером, который использует бизнес-приложение(для обычного программиста наиболее важным является Northbound API, чем гибче и проще данное API, тем легче программисту создавать целевые сетевые приложения)

5.1 POX-контроллер.

POX контроллер – open-source проект, разрабатываемый на Python под лицензией GNU License(официально поддерживает Windows, Mac OS, Linux)

POX в настоящий момент поддерживает OpenFlow 1.0(частично поддерживает под Openflow 1.1), также имеется поддержка Open vSwitch/Nicira extensions(специальное расширение протокола OpenFlow, позволяющее создавать более гибкие настройк маршрутизаторов, пример: action=normal – flow который заставляет обрабатывать совпавший пакет, как обычный L2 Linux bridge, ставить числовые метки на пакеты, которые идут по forwarding pipeline и таким образом контролировать обработку).

POX сам по себе никак не управляет трафиком – функциональность контроллера создается запуском отдельных компонент вместе с контроллером(components), следовательно целевая аудитория проекта POX – разработчики, которые хотят реализовать свои проекты. Стандартный дистрибутив включает в себя набор базовых компонент.

Разработка компонент

POX компоненты по сути это модули, написанные на Python, следовательно в контроллер можно добавить любой код, который разработчик захочет. По соглашению, каждая компонента содержит специальную функцию, которая выполняется при запуске контроллера:

```
def launch(args..):
    //код компоненты
```

Стандартно, каждая компонента может быть запущена только один раз, но специальными командами можно запустить одну и ту же компоненту в нескольких экземплярах.

POX API

POX содержит объект – ”ядро(core)“, который реализует основу POX’s API. Основная цель данного объекта – коммуникация между запущенными компонентами. При запуске, указанные компоненты регистрируются на этом объекте и всевозможные пересылки между компонентами реализованы через его использование. Если необходимо “зарегистрировать” компоненту в “ядре” необходимо выполнить следующие две команды:

```
core.register(component_class)
core.registerNew(component_class)
```

В итоге “ядро”(core) и стандартные компоненты библиотеки pox предоставляют следующее API's:

- Dependency and Event Management(компоненты могут создавать события, а другие компоненты на них реагировать)
- Working with packets and addresses(парсинг пакетов, создание Openflow пакетов с “нуля”)
- Using threads, timers, tasks(POX's recoco library специальная библиотека для многопоточности; например требуется собирать статистику с контроллера каждые N секунд)
- Working with asynchronous sockets(коммуникация между контроллером и удаленным хостом)
- Openflow protocol management(разработка Openflow приложений)

Общий обзор

POX в основном используется для исследований ПКС, т.к. предоставляет простое и удобное Northbound API для небольших задач. Основная цель этого проекта – разработка и исследование парадигмы ПКС, для полнценных коммерческих приложений этот проект не подходит из-за ограниченного API и медленной работы(т.к. написан на Python)

5.2 Floodlight-контроллер.

Контроллер разрабатываемый на Java 1.7+ под лицензией Apache 2.0. Поддерживает только топологии без циклов между различными компонетами связности. Имеет модульную структуру, т.е. перед запуском в специальный конфигурационный файл добавляются названия сервисов, которые будут использованы. Поддерживает следующие сервисы:

- Forwarding – стандартный модуль, обеспечивающий реактивное конфигурирование OpenFlow маршрутизаторов.

- Static Flow Entry Pusher – специальное приложение для для добавления flow-записей в выбранный маршрутизатор.
- Firewall(брэндмауэр) – применение ACL правил, для ограничения траффика(настраивается через Rest API)
- Learning Switch – обычный L2 маршрутизатор(возможна настройка через Rest API)
- Load Balancer(балансировщик нагрузки) – балансировка tcp, ping, udp трафика.

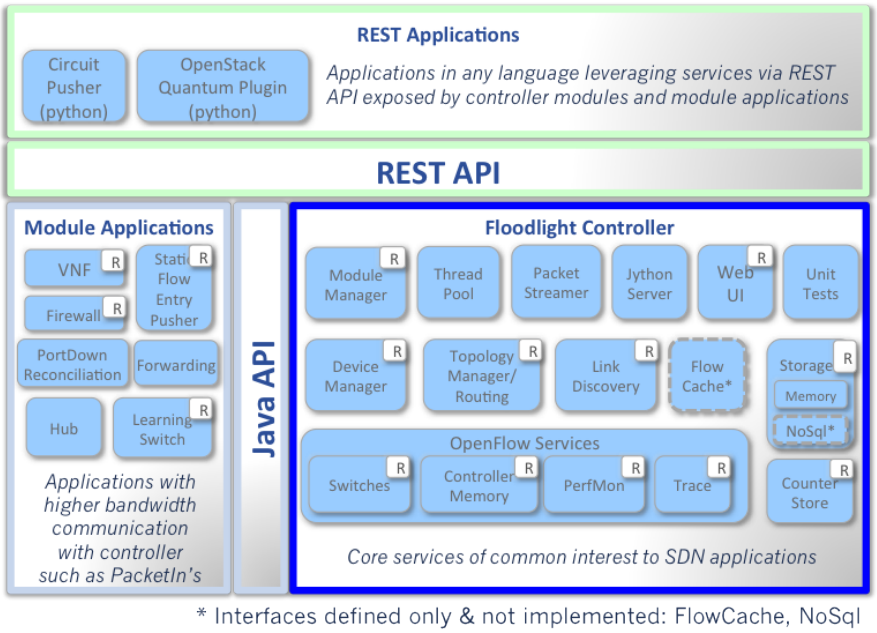


Рис. 4: Структура Floodlight(www.docs.projectfloodlight.org)

На картинке показана архитектура контроллера. По сути контроллер выполняет стандартные функции сканирования сети, подключение новых устройств, работа с OpenFlow(1.1), наличие графического интерфейса.

Для разработки приложений на базе данного контроллера, предоставляется Restful Java API.

5.3 Opendaylight-контроллер.

OpenDaylight — open-source проект с гибкой модульной платформой для встраивания различных плагинов. Проект разрабатывается под лицензией EPL v1.0(Eclipse public license) при поддержке Linux Foundation(считается на данный момент самым крупным проектом среди контроллеров ПКС). Т.к. проект разрабатывается на

языке Java(Java 1.7), следовательно является кроссплатформенным. Последний релиз Hydrogen полностью поддерживает протокол OpenFlow 1.3, BGP-LS, протокол OVSDB, PCEP, SNMP.

Архитктурные принципы OpenDaylight

- **Модульность и расширяемость** – модульный, расширяемый контроллер поддерживает установку, удаление и обновление используемых служб, во время работы(без выключения)
- **Множественная поддержка Southbound протоколов** – использование различных сетевых протоколов(OpenFlow, BGP и.т.д)
- **Служебный абстрактный уровень(Service Abstraction Layer)** – использование различных Northbound протоколов
- **Consistent clustering(распределенная версия)** – предоставляет отказоустойчивость и распредленность
- **Разделяемость** – управление различными частями сети, при помощи разных компонент контроллера

Функциональный обзор

The OSGi framework позволяет динамически подключать плагины для использова-

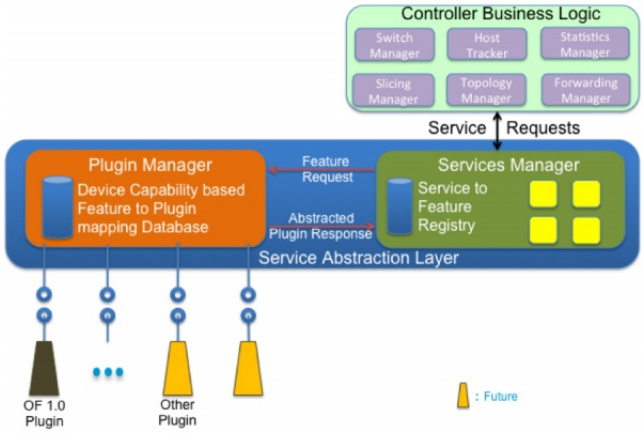


Рис. 5: Структура OpenDaylight(www.opendaylight.org)

ния новых Southbound протоколов. SAL предоставляет такие базовые сервисы, как Device Discovery(обнаружение сетевых устройств в сети и подключение к ним), который в свою очередь используются такими модулями как Topology Manager(хранение,

обновление информации о топологии сети) и.т.д. Основываясь на запросе сервиса – SAL маппирует нужный плагин и использует наиболее подходящий southbound протокол.

Одним из важных достоинств OpenDaylight является поддержка распределенной версии(*High Avalability Model*), позволяет развертывать контроллер на нескольких физических серверах, гарантируя тем самым отказоустойчивость и балансировку нагрузки.

Общий обзор

OpenDaylight controller огромный быстро развивающийся проект спонсируемый такими крупными организациями как: Cisco, IBM, Red Hat. Из достоинств можно выделить: работает на JVM, т.е. работает везде, где есть Java; модульность - поддерживает несколько различных Northbound APIs(Rest API, OSGi frameowork) и Southbound APIs которые между собой независимы(важная особенность, что модули можно загружать прямо во время работы контроллера. Есть дополнительная поддержка OVSDB protocol, следовательно может использовать все особенности и расширения Open vSwitch.

5.4 Итоги сравнения.

Проект / Характеристика	POX	Floodlight	OpenDaylight
Язык	Python 2.6	Java 1.7+	Java 1.7+
Строк кода	56961	90578	332317
Поддержка OVS	1.11	2.0+	2.0+
OpenFlow Hardware	Да	Да	Да
Тип API	Rest API	Rest API	Rest API, OSGi framework
Поддержка GRE	Нет	Да	Да
Распределенная модель	Да	Да	Да

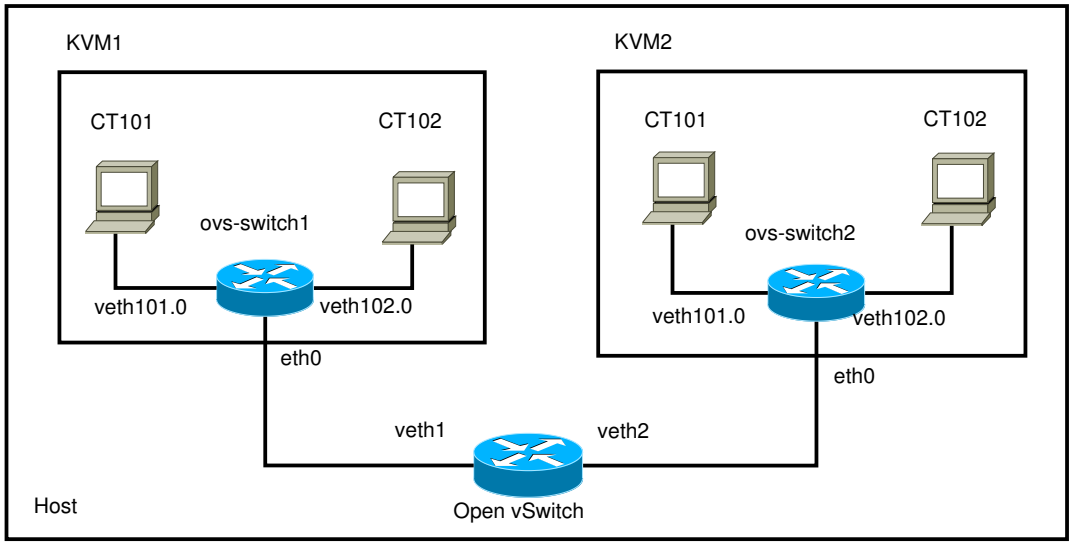
В целом можно сказать, что POX контроллер используется для проверки производительности ПКС на различных топологиях сетей и тестирования протокола OpenFlow. Floodlight и OpenDaylight можно объеденить в одну категорию, как два проекта, разрабатываемых на языке Java. Хотя Floodlight можно назвать переходным между POX и OpenDaylight, т.к. последний поддерживает любые топологии сетей, большое

кол-во протоколов маршрутизации, является более гибким и лучше подходит для решения реальных бизнес-задач.

Среди рассмотренных проектов сильно выделяется OpenDaylight. Есть поддержка распределенной модели, OSGi фреймворк позволяет загружать модули не перезагружая контроллер, +активная поддержка Linux сообщества(спонсоры: IBM, Cisco, Nicira, OpenFlow Network Foundation и.т.д)

6 Постановки задач.

6.1 Ограничение рассылки широковещательных ARP запросов в VPN



Дана следующая схема сети, необходимо настроить *ovs-switch1* и *ovs-switch2* так, чтобы arp-broadcast запросы контейнеров CT102 не попадали на интерфейсы CT101. Соответственно контейнер не можеть получить mac-адрес какого-либо контейнера не из своей подгруппы.

Решение – использование broadcast-unicast и global-local битов в mac-адресе посылаемого arp пакета(см.приложение).

mac-адрес в первом октете содержит 2 бита, первый указывает broadcast это пакет или unicast, а второй global-local – является ли адрес глобально или локально администрируемым. Соответственно, при помощи утилиты *ovs-ofctl* выставляем local и broadcast для стандартных arp broadcast(ff:ff:ff:ff:ff:ff) пакетов, а остальные 46 бит можно использовать как уникальный ID соответствующей группы.

Настройка:

На главном хосте Open vSwitch – работает как обычный маршрутизатор, никаких дополнительных потоков в нем прописывать не нужно. Пусть *mac1-broadcast* и *mac2-broadcast* – уникальные для соответствующих подсетей(CT101,CT102) глобальные MAC-адреса(например: 0f:00:00:00:00:01, 0f:00:00:00:00:02), тогда:

```
kvm1$ ovs-ofctl add-flow ovs-switch1 priority=10,in_port=1,
dl_dst=ff:ff:ff:ff:ff:ff,dl_type=0x0806,actions=mod_dl_dst:mac1_broadcast,all
kvm1$ ovs-ofctl add-flow ovs-switch1 priority=10,in_port=2,
dl_dst=ff:ff:ff:ff:ff:ff,dl_type=0x0806,actions=mod_dl_dst:mac2_broadcast,all

kvm1$ ovs-ofctl add-flow ovs-switch1 priority=10,in_port=3,
dl_dst=mac1_broadcast,dl_type=0x0806,actions=1
kvm1$ ovs-ofctl add-flow ovs-switch1 priority=10,in_port=3,
dl_dst=mac2_broadcast,dl_type=0x0806,actions=2

kvm1$ ovs-ofctl add-flow ovs-switch1 priority=5,actions=normal
```

На KVM2 команды дублируются с учетом портов, к которым подключены контейнеры к маршрутизатору *ovs-switch2*.

Первая группа команд преобразует стандартные arp-broadcast запросы, выставляя уникальные mac адреса в поле mac destination у сетевого пакета. Вторая группа обрабатывает пришедшие пакеты извне, и рассылает на соответствующие порты. И наконец последняя команда обрабатывает arp unicast пакеты и ip пакеты.

В итоге, поставленная задача решена полностью – arp-broadcast пробрасываются только на хосты из одной подгруппы и отлкик(ping) есть только у хостов из одной подгруппы. Этим можно дополнительно контролировать трафик между различными подсетями.

Данная задача имеет приложение и в области безопасности – например пользователь виртуальной машины сделав даже ARP broadcast запрос не узнает, какие есть виртуальные машины в облаке, кроме тех, к которым у него есть доступ.

Дополнительно был создана компонента для POX контроллера, которая по входным данным(топологии сети) генерирует для каждой VPN свой уникальный широковещательный адрес и делает соответствующие записи в маршрутизаторах, с кото-

рыми соединены хосты.

6.2 Реализация распределенной версии POX-контролллера.

Среди контроллеров, которые сравниваются в данной работе, только в POX[11] не реализована распределенная версия.

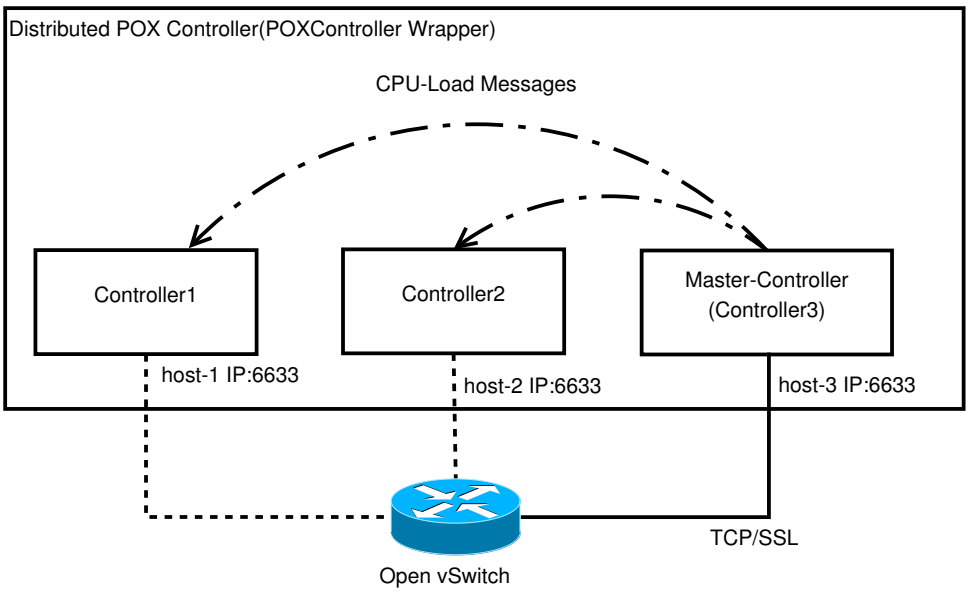
POX поддерживает версию 1.0 протокола OpenFlow, в которой не реализована синхронизация между несколькими подключенными контроллерами одновременно, но маршрутизатор можно настроить так, чтобы он подключался сразу к экземплярам сразу:

```
host$ ovs-vsctl set-controller <BRIDGE> controller1 .. controllerN
```

Проблема заключается в том, что при подключении все контроллеры будут иметь *равные права* на запись и удаление *flow-потоков* в таблицах маршрутизатора.

Следовательно возникает проблема синхронизации – в спецификации OpenFlow 1.3 добавлен обмен сообщениями между маршрутизатором и подключенными контроллерами для выбора *главного(master)* и *дочерних(slave)* экземпляров. *Master* имеет права на *запись и удаление flow-потоков*, а *дочерние* экземпляры имеют права лишь на чтение данных с маршрутизатора(аггрегирование статистики, чтение состояния *flow-таблиц*).

6.2.1 Алгоритм работы.



- Список хостов(IP - адрес, порт), на которых будет запущен экземпляр контроллера определяется заранее администратором, после чего в базу данных

маршрутизатора записываются эти адреса

- В единый момент времени, активен только 1 контролллер(таким образом исключается возможность рассинхронизации), остальные члены кластера проводят мониторинг хоста на котором запущен контроллер.

- master-controller с определенной периодичностью рассылает сообщения, в которых указывает загрузку своего CPU и также этим сигнализирует о том, что связь не потеряна.

- Если по какой-либо причине происходит потеря связи с активным контроллером либо загрузка его CPU превышает некоторый заранее установленный предел, запускается атомарный процесс выбора нового активного контроллера. Выбранный хост, запускает у себя экземпляр контроллера.

- Хост, первый заметивший потерю сигнала от главного контроллера запускает атомарный процесс смены мастера. В случае “развала” кластера на несколько несвязных компонент, в каждой из них будет выбран свой мастер(это может привести к рассиннхронизации, поэтому за такой ситуацией должен следить администратор), но как только связь между компонентами будет восставлена – автоматически запустится процесс слияния компонент.

6.2.2 Особенности реализации. Тестирование.

Программа представляет собой “обертку”, которая проводит процесс мониторинга соединения между хостами, на которых запущен контроллер. При запуске программы на удаленных хостах, программа рассылает BROADCAST UDP сообщения, с помощью которых члены кластера ”находят“ друг друга и отправляют запрос на подключение.

В основу программы легла библиотека JGroups[10](www.jgroups.org), которая предоставляет возможность создания кластерных приложений и обмена сообщениями между членами кластера(можно отметить, что Floodlight Project и OpenDaylight Project также используют эту библиотеку для распределенных версий). JGroups предоставляет возможность использовать атомарные операции и *shared-variables* внутри кластера.

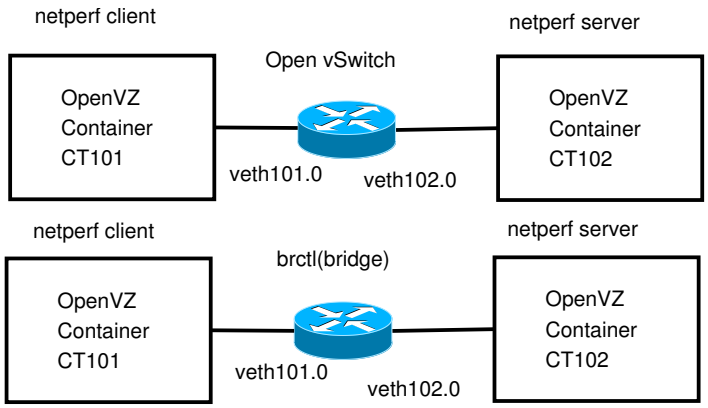
Тесты проводились на (2-7) виртуальных машинах, время переключения на нового мастера составляло порядка 2-2.5 секунд при нарушении связи(резком выклю-

чении виртуальной машины с запущенным мастером). С учетом того, что время на регистрацию утери связи проходило около 2с., выбор нового master-controller происходил за 0.1-0.3 секунды, независимо от количества запущенных контроллеров.

Ссылка на репозиторий проекта:
https://github.com/fedor-goncharov/PoxWrapper

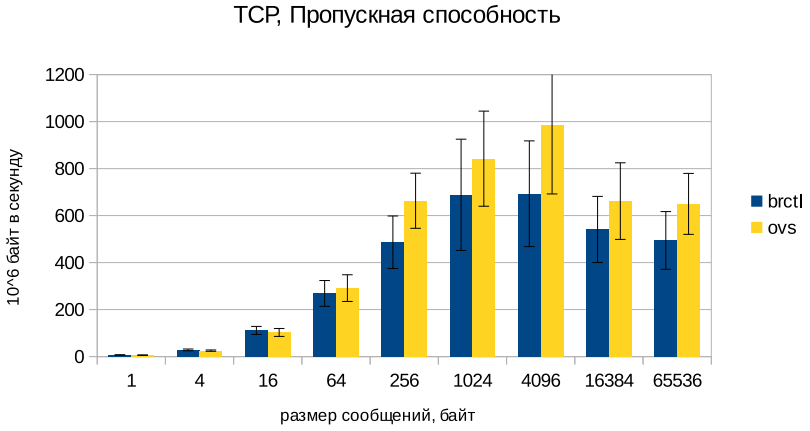
6.3 Сравнение производительности Open vSwitch и библиотеки brctl.

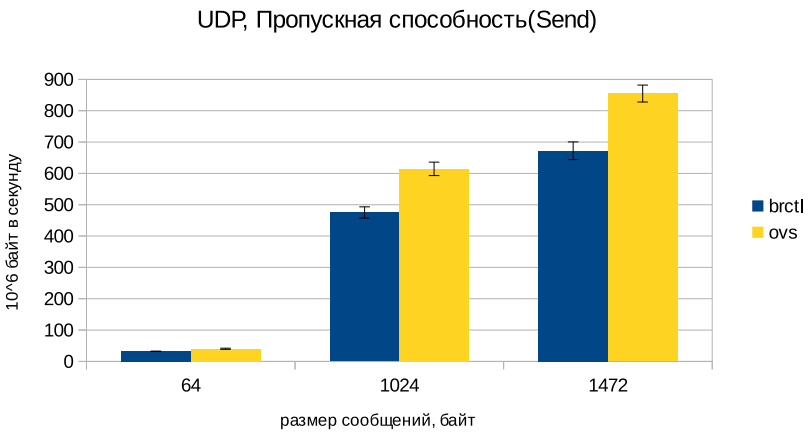
На схеме(см.внизу) представлена модель тестирования производительности:

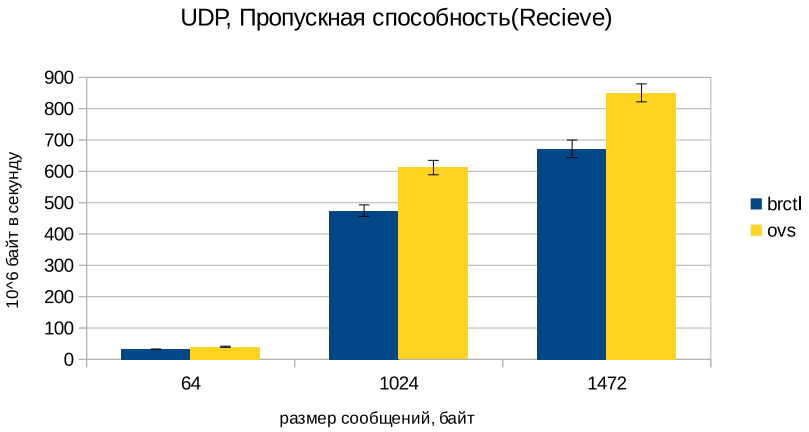


Был проведен набор стандартных тестов при помощи утилиты *netperf*. Измерена пропускная способность, а также частота передачи сообщений(request-response per second).

Пропускная способность:

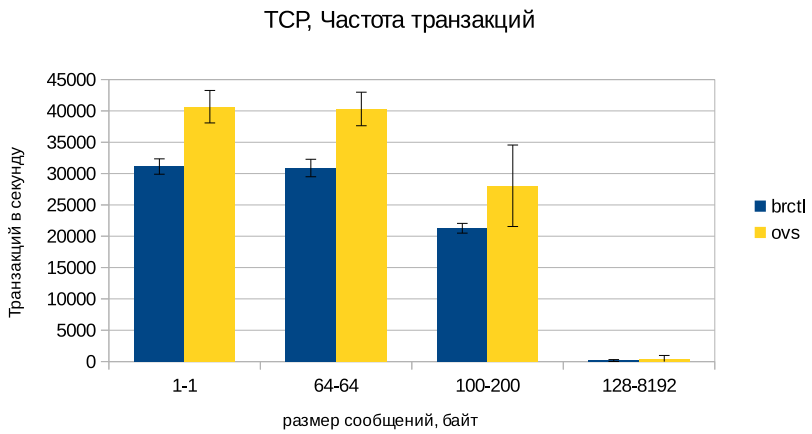


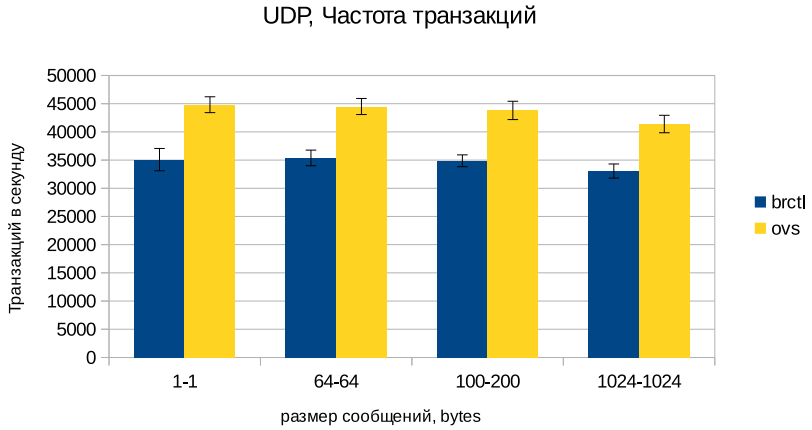




Каждое измерение повторялось по 15 раз, затем результаты усреднялись и вычислялось стандартное отклонение. Из данных графиков видно преимущество *Open vSwitch* в скорости обмена сообщениями по протоколу UDP(прирост в скорости составил в среднем 20%, причем стоит заметить, что области стандартных отклонений не пересекаются – это говорит в пользу достоверности сделанных выводов). В тестах с использованием протокола TCP, при размерах сообщений от 64 байт и выше прирост в скорости составил не менее 18%(максимум 30% при размере сообщений – 4096 байт)(к сожалению при проведении тестов, обеспечить малость стандартного отклонения не удалось).

Частота транзакций:





По частоте обмена сообщениями request-response *Open vSwitch* также превосходит мост из библиотеки brctl(в тесте TCP в среднем на 23%, UDP в среднем на 21%); Стоит также отметить что интервалы стандартных отклонений не пересекаются, что свидетельствует о корректности сделанных выводов.

Все тесты проводились при одной записи во *flow-таблице*, где в качестве поля *action* было задано действие NORMAL, т.е. стандартная обработка пакетов L2/L3 маршрутизатора.

7 Заключение.

В данной работе проведен обзор основных элементов *ПКС(SDN)*, показано, что *Open vSwitch* полностью соответствует спецификации *OpenFlow Switch*, а также содержит ряд удобных расширений и дополнений(*Nicira Extensions*). Обработка входящего трафика полностью определяется записями во *flow-таблицах*, которые можно конфигурировать вручную при помощи утилиты *ovs-ofctl*, или автоматически подключая к маршрутизатору OpenFlow-контроллер. Гибкость настройки *OpenFlow*

маршрутизатора достигается при помощи модели обработки трафика - *Forwarding pipeline*; данная модель основана на использовании ряда *flow-таблиц* и возможности передавать обработку пакета из одной таблицы в другую – это позволяет делать настройки расширяемыми и облегчает настройку маршрутизатора для сетевого администратора.

Итогом обзора OpenFlow маршрутизатора стало сравнение производительности *Open vSwitch* и библиотеки *brctl*: тесты на OpenVZ контейнерах показали, что *Open vSwitch* превосходит *brctl(bridge)* как и в STREAM PERFOMANCE тестах, так и в TRANSACTION PERFOMANCE тестах(для протоколов TCP,UDP).

Также в работе проведено качественное сравнение современных OpenFlow-контроллеров, показаны сильные и слабые стороны рассмотренных проектов(*POX, Floodlight Project, OpenDaylight Project*) – указаны приоритетные цели использования данных проектов:

Так *POX* и *Floodlight* представляют скорее научный инструмент, для быстрой разработки и тестирования парадигмы *ПКС*, в то время как *OpenDaylight Project* и его последняя версия *Hydrogen* подходят для интеграции и решения реальных бизнес задач. В отличие от других проектов *OpenDaylight* поддерживает не только протокол *OpenFlow*, но возможны встраивания других *Southbound* протоколов и имеет наибольшее число спонсоров, среди которых такие крупные как Linux Foundation, Cisco, Citrix, Juniper, IBM, RedHat, Windows и некоторые другие.

Итогом сравнения контроллеров является таблица, которая находится в параграфе “OpenFlow-контролллеры”, в которой указаны основные интересующие характеристики.

В конце можно упомянуть реализацию распределенной версии POX-контроллера основанной на использовании библиотеки JGroups; приоритетом написания данной версии было решить задачу отказоустойчивости, тесты показали, что время переключения на новый экземпляр составляет порядка 0.1-0.3 секунды, независимо от количества запущенных контроллеров.

В целом можно сказать, что идеологически все проекты контроллеров не сильно различаются, т.к. по сути делают одно и то же(предоставляют удобное API программистам для работы с сетью), следовательно приоритетным является тот проект, который предоставляет наиболее удобное и гибкое API.

8 Приложения.

8.1 Структура MAC адреса

Стандарты IEEE определяют 48-разрядный (6 октетов) MAC-адрес, который разделен на четыре части.

Первые 3 октета (в порядке их передачи по сети; старшие 3 октета, если рассматривать их в традиционной бит-реверсной шестнадцатеричной записи MAC-адресов) содержат 24-битный уникальный идентификатор организации (OUI), или (Код MFG — Manufacturing, производителя), который производитель получает в IEEE. При этом используются только младшие 22 разряда (бита), 2 старшие имеют специальное назначение:

- первый бит (младший бит первого байта) указывает, для одиночного (0) или группового (1) адресата предназначен кадр
- следующий бит указывает, является ли MAC-адрес глобально (0) или локально (1) администрируемым.
- Следующие три октета выбираются изготовителем для каждого экземпляра устройства. За исключением сетей системной сетевой архитектуры SNA.

Список литературы

[1] K.Calvert, S.Bhattacharjee, E.Zegura, and J.Sterbenz, «Directions in Active Networks» *IEEE Communications magazine*, стр. 72-78, October 1998

[2] Nicira. It's time to virtualize network, 2012. http://nicira.com/en/network-virtualization-platform

[3] N.Mc Keown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J.Rexford, S.Shenker, and J. Turner «OpenFlow: Enabling innovation in campus networks» *ACM SIGCOMM Computer Communications Review, April 2008*

[4] «OpenFlow Switch Specification» Open Networking Foundation. June 2012

[5] Open vSwitch Project. http://openvswitch.org/documentation

[6] NOXRepo. http://noxrepo.org

[7] Floodlight Project. http://projectfloodlight.org/floodlight

[8] Opendaylight Project. http://opendaylight.org

[9] N. Feamster, J. Rexford, E. Zegura,
«The Road to SDN: An Intellectual History of Programmable Networks», http://cs.princeton.edu/courses/archive/fall13/cos597/papers/sdnhistory.pdf

[10] JGroups. http://jgroups.org

[11] Volkan Yazici, M. Oguz Sunay, Ali O. Ercan «Controlling a Software-Defined Network via Distributed Controllers»

[12] A.Tannenbaum, D. Wetherall «Computer Networks. 5th Edition»