Сети SDN на основе протокола OpenFlow

Протокол OpenFlow (OF) — одна из наиболее популярных реализаций протокола «контроллер SDN-коммутатор SDN».

Модель коммутатора и протокол OpenFlow были предложены в 2006 году исследователями Стэндфордского университета и Университета Беркли. Именно эта работа и положила начало утверждению концепции программно-управляемых сетей в том виде, в котором она существует сегодня. Изначально протокол и коммутатор OpenFlow создавались как средство быстрой разработки и тестирования новых сетевых протоколов, то есть как инструмент исследователей. Однако довольно быстро пришло понимание, что его гибкость и мощность могут быть использованы для изменения к лучшему функциональности сетей.

Разработкой стандартов OpenFlow занимается некоммерческий консорциум Open Networking Foundation (ONF), куда входят многие ведущие провайдеры, производители и исследовательские организации. Консорциум ONF также осуществляет поддержку открытой сетевой **операционной системы контроллера ONOS** (Open Network Operating System) и ряда приложений для этой ОС.

Первая версия стандарта OF 1.0 появилась в 2009 году, а последняя, OF 1.5 – в 2015 году. Наиболее устойчивой версией является версия OF 1.3 — ее в настоящее время поддерживают большинство производителей коммуникационного оборудования. Начнем рассмотрение протокола OpenFlow с версии 1.0, которая хорошо отражает основные принципы работы этого протокола, а затем обратимся к некоторым усовершенствованиям, сделанным в последующих версиях.

Сообщения протокола OpenFlow

По версии OF 1.0 модель коммутатора использует одну **таблицу продвижения**, состоящую из ряда записей – **правил обработки** пакетов. В исходном состоянии таблица продвижения коммутатора пуста. Ее формирование – это обязанность приложений контроллера SDN. Полученные от приложений правила обработки пакетов контроллер передает коммутатору по протоколу OF. Помимо сообщений-*правил* в число возможных сообщений протокола OF входят также сообщения-*запросы*, с помощью которых контролер запрашивает у коммутатора информацию о состоянии его портов (работоспособные или нет), а также статистику потоков. В протоколе OF предполагается, что коммутатор не только отвечает на запросы контроллера, но может передать контроллеру сообщения по *своей инициативе,* например, в случае изменения состояния порта или удалении некоторого правила по тайм-ауту. Канал обмена сообщениями между контроллером и коммутатором SDN называется **управляющим каналом**. Он представляет собой *TCP-сессию*, установленную в IP-сети контроллером и коммутатором.

Итак, каждое **правило** может включать элементы трех типов:

- *условия* выделения потока пакетов, к которым это правило должно быть применено;
- *действия*, которые должны быть выполнены над пакетом, который удовлетворяет условиям данного правила;
- *счетчики*, измеряющие характеристики потока пакетов.

Условия строятся на основе значений полей заголовка пакета, а также номера порта коммутатора, на который поступил кадр. Версия OF 1.0 выбрала в качестве *эталонного заголовка* достаточно типичный набор параметров кадра Ethernet, в которые вложены пакеты IP с сегментами TCP или дейтаграммами UDP:

- MAC-адрес источника (MAC Src) и MAC-адрес назначения (MAC Dst);
- Тип кадра Ethernet (Ethernet Type);
- Идентификатор VLAN (VLAN ID) и приоритет VLAN (VLAN Priority);
- IP-адрес источника (IP Src) и IP-адрес назначения (IP Dst);
- Коды вложенного протокола (IP protocol) и типа обслуживания (IP ToS);
- TCP/UDP-порт источника (TCP/UDP Src Port) и назначения (TCP/UDP Dst Port);
- номер входного физического интерфейса коммутатора (Port).

Примером условия может быть такой набор значений:

Conditions:

{Port = 5; VLAN ID = 254; IP Src = 194.81.18.227; IP Dst = 130.16.55.12; TCP Dst Port = 5009}

Это условие определяет поток пакетов, поступающих на порт 5, относящийся к VLAN 254, отправленных хостом с IP адресом 194.81.18.227 хосту с IP адресом 130.16.55.12 на TCP порт 5901. Условие считается выполненным, если выполняются все эго компоненты, то есть они объединяются логическим «И». Как видно из примера, в условии не обязательно использовать все возможные поля заголовка кадра; те поля, которые в условии не указаны, считаются замаскированными, то есть они могут иметь любые значения. Поля IP адресов могут иметь маску переменной длины, задающую значимую часть адреса, то есть возможно задать префикс 130.16.55.0/24 вместо адреса хоста 130.16.55.12.

Конструкция условий протокола OF позволяет очень тонко выделить поток данных и тем самым обеспечить его индивидуальную обработку коммутатором. Сравните это с возможностями стандартных коммутаторов и маршрутизаторов, которые считают одним потоком все пакеты, у которых совпадает лишь один признак – адрес назначения. Набор возможных **действий** некоторого правила включает передачу пакета на один из выходных портов коммутатора и операции по изменению значений полей заголовка пакета. Если действия в правиле отсутствуют, то пакет должен быть *отброшен*. Например, для того, чтобы пакеты потока были переданы на порт 8, и чтобы при этом номер VLAN был изменен на 1002, а MAC-адрес назначения — на bc:30:7e:d4:5d:8f, достаточно определить набор из трех действий:

Actions:

{Port=8; VLAN ID = 1002, MAC Dst = bc:30:7e:d4:5d:8f}

Счетчики позволяют коммутатору измерять статистические характеристики потока: его интенсивность (количество поступивших пакетов за секунду), продолжительность существования. Коммутатор также должен поддерживать агрегатные счетчики для каждого порта. Счетчики обновляются при поступлении каждого нового пакета в коммутатор, но непосредственно на обработку пакетов не влияют. Контроллер SDN может запросить у коммутатора значения счетчиков и на основании этих данных принять решение, например, заблокировать некоторый подозрительный поток, если его интенсивность слишком велика.

Каждое правило имеет тайм-аут неактивности и срок жизни. Если *срок жизни* правила превышен или если оно не применяется в течение интервала времени, превышающего *тайм-аут неактивности* (то есть поток этого типа перестал поступать в коммутатор), то это правило должно быть удалено. Правила также имеют *приоритеты*, причем правило с большим приоритетом проверяется раньше правила с меньшим приоритетом. Приоритеты позволяют контроллеру легче изменять таблицу продвижения. Так, правило с высоким приоритетом может быть добавлено в таблицу без удаления правила с тем же набором условий, но с более низким приоритетом. Например, такая пара правил может использоваться при задании основного и резервного маршрутов: высокоприоритетное правило определяет основной маршрут, а низкоприоритетное – резервный; в случае отказа на основном пути контроллер просто удаляет высокоприоритетное правило и пакеты идут по резервному пути. При обработке пакета таблица продвижения коммутатора просматривается только *один раз*, в порядке, задаваемом приоритетами правил. Если для пришедшего пакета ни одно из условий таблицы не выполняется, то он отбрасывается.

Виртуальные порты

Контроллер и его приложения должны иметь возможность реагировать на появление новых потоков в сети, иначе гибкость сети SDN не будет достаточной. С этой целью в технологии SDN предусмотрен **виртуальный порт CONTROLLER.** Для того, чтобы все пакеты, принадлежащие неизвестным коммутатору потокам (то есть для которых не нашлось условия в таблице, вызывающего совпадение) не отбрасывались, а обрабатывались особым, предусмотренным для них способом, необходимо поместить в таблицу следующее правило, имеющее нулевой приоритет:

Priority = 0

Conditions: {}

Actions: {port=CONTROLLER}

Данное правило будет играть роль «стопора» для неизвестных ранее потоков, так как с пустым списком условий оно будет вызывать совпадение для любого пакета, но применяться оно будет всегда после проверки всех остальных правил, из-за своего самого низкого приоритета. Если такое правило в таблице отсутствует, то все нераспознанные пакеты просто отбрасываются, но при его наличии они направляются в порт CONTROLLER.

Пакет, посланный в порт CONTROLLER, передается по управляющему каналу с помощью **сообщения**

packet_in в контроллер и далее – в приложение, которое должно решать, что нужно делать в такой ситуации, например, оно может сформировать для нового потока новое условие с ненулевым приоритетом и послать его в коммутатор. В сообщении packet_in, наряду с полученным пакетом, содержится *номер порта*, через которой он был получен. После этого последующие пакеты данного потока уже не будут посылаться в контроллер, а будут переданы на некоторый физический порт в соответствии с набором действий нового правила.

Из приведенного описания видно, что основным режимом работы коммутатора OF является *автономный режим*, когда пакеты продвигаются им без непосредственного участия контроллера, за счет сформированной заранее таблицы продвижения. Контроллер вмешивается и перестраивает таблицу продвижения только в случае необходимости, когда в сети происходят события, такие как появление новых потоков или отказы ее элементов. Степень адаптивности сети SDN целиком зависит от эффективности приложений, работающих на сервере (или кластере серверов), выполняющем роль контроллера.

Протокол автоматического распознавания связей BDDP

Как известно, для нормального функционирования сети необходимо, чтобы в ней работали *служебные протоколы* (протоколы слоя управления): протоколы маршрутизации (OSPF, IS-IS, BGP, Spanning Tree) или протоколы мониторинга соединений (CFM). Важную часть этих протоколов составляет процесс генерации служебных сообщений и обмен ими со своими соседями. Но такие действия не входят в функциональность коммутаторов SDN. Чтобы сделать возможным работу служебных протоколов в сети SDN, разработчики добавили в набор сообщений протокола OF **сообщение packet_out.** Этим сообщением контроллер говорит коммутатору о том, что последний должен передать пакет, находящийся в данном сообщении, на один из своих портов, номер которого также содержится в этом сообщении.

То есть контроллер генерирует некоторый пакет, подобный объявлениям маршрутизаторов при построении топологии сети, после чего помещает его в сообщение packet_out и передает по протоколу OF коммутатору для передачи на определенный сетевой интерфейс. Наличие команды packet out делает возможным работу в контроллере OF SDN приложений, реализующих служебные протоколы, например, протокол OSPF или BGP. Контроллер в этом случае *эмулирует работу маршрутизатора* по автоматическому построению топологии сети и выбору в ней маршрута в соответствии с некоторой метрикой. Такая функциональность может быть полезной при поэтапном внедрении островков OF SDN в традиционные сети. Однако для автоматического построения топологии сети OF SDN обычно используется другой подход, более соответствующий централизованной схеме управления сетью. Он предусматривает следующие действия:

- контроллер генерирует служебные пакеты, передаваемые каждому коммутатору с указанием направить этот пакет на все его выходные порты, так что сгенерированный пакет поступает *всем непосредственным соседям* некоторого коммутатора:
- в таблицу каждого коммутатора вносится правило, в соответствии с которым пакеты это типа, пришедшие на любой входной порт, коммутатор должен всегда передавать *котроллеру* и никогда не распространять далее по сети.

Таким образом, контроллер по принятым от коммутаторов пакетам этого служебного протокола может судить о связях между коммутаторами. Так, если пакет A был отправлен коммутатору S1, а принят от коммутатора S2, то между ними имеется непосредственная связь. Связь коммутаторов идентифицируется двумя парами параметров (MAC-адрес коммутатора, номер порта), описывающих каждый из оконечных коммутаторов. Данный подход реализуется **протоколом BDDP** (Broadcast Domain Discovery Protocol). Он является модификацией протокола LLDP (Link Layer Discovery Protocol), с помощью которого коммутаторы локальных сетей обнаруживают своих непосредственных соседей по сети. Чтобы традиционные коммутаторы отличали пакеты BDDP от пакетов других протоколов, для него зарегистрирован код EtherType, равный 0x8942.

Контроллер посылает коммутатору сообщение packet_out, в которое вложен готовый к отправке кадр Ethernet с широковещательным адресом назначения, несущий в своем поле данных пакет BDDP. В пакете BDDP содержится *MAC-адрес коммутатора*, на который контроллер направил данный пакет, а также *номер порта*, через который коммутатор должен отослать кадр Ethernet (рис.16.7). Кадр Ethernet должен быть отправлен коммутатором с порта, номер которого указан в поле сообщения packet_out. Заметим, что номер этого порта (P) уже присутствует в пакете BDDP. Но здесь нет избыточности, так как содержимое пакета BDDP предназначено исключительно для контроллера и недоступно для коммутаторов.

Рис. 16.7. Структура сообщения, несущего пакет BDDP

Рассмотрим работу протокола BDDP по автоматическому распознаванию связей на примере сети из трех коммутаторов и контроллера SDN (рис. 16.8).

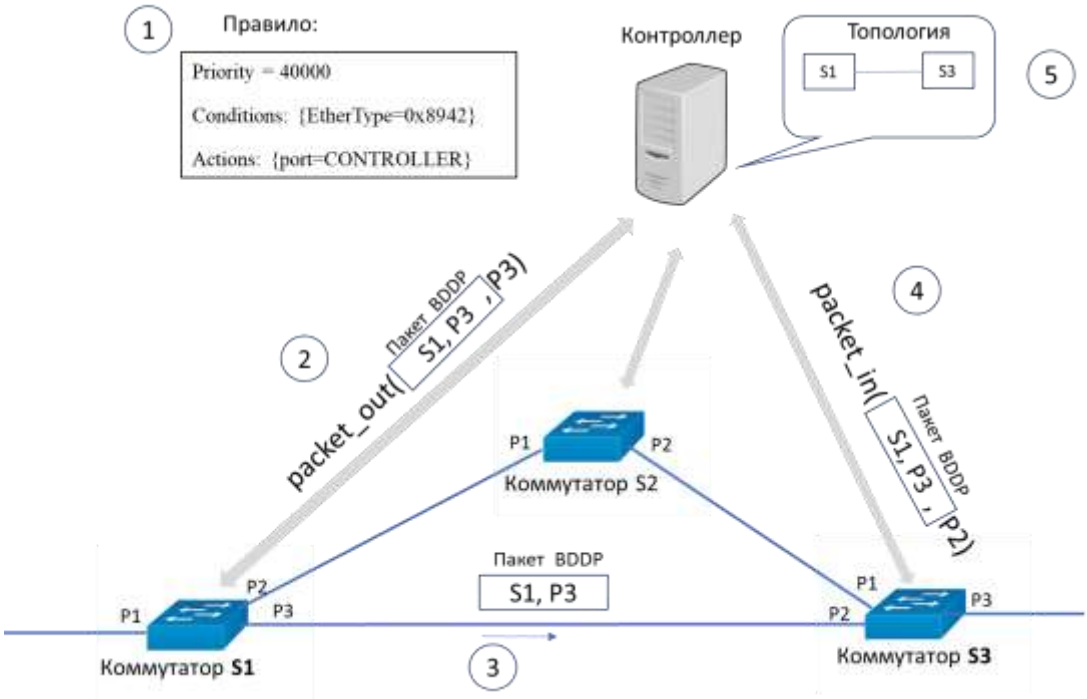

Рис. 16.8. Автоматическое распознавание связи с помощью пакетов BDDP

В соответствии с протоколом OF каждый коммутатор при старте устанавливает управляющий канал с контроллером (для этого коммутатору при конфигурации должен быть указан IP-адрес контроллера) и сообщает ему свои параметры, в том числе свои MAC- и IP-адреса, а также данные о каждом из своих портов – тип (например, Ethernet), скорость (например, 10G), MAC-адрес и состояние – работоспособен или нет. Этой информации достаточно для того, чтобы контроллер начал процедуру автоматического распознавания связей между коммутаторами.

Этап 1 этой процедуры заключается в том, что контроллер устанавливает в каждом коммутаторе правило, в соответствии с которым BDDP-пакет, поступивший на любой входной порт коммутатора, должен быть направлен контроллеру:

Priority = 40000

Conditions: {EtherType=0x8942}

Actions: {port=CONTROLLER}

Этап 2. Контроллер начинает периодически, скажем, один раз секунду, посылать всем коммутаторам сообщения packet_out со сгенерированными им пакетами BPPD для каждого из портов каждого коммутатора. В нашем примере контроллер посылает каждую секунду коммутатору S1 *три* сообщения packet_out для портов P1, P2 и P3. На рис. 16.8 показано одно такое сообщение packet_out, в поле данных которого содержится пакет BDDP {MAC-адрес коммутатора S1, номер порта P3}. В заголовке сообщения packet_out также есть указание, что пакет BDDP нужно передать в сеть через порт P3.

Этап 3. Получив сообщение packet_out, коммутатор извлекает пакет BDDP и передает его в сеть через

указанный в сообщении порт, в нашем примере порт P3.

Этап 4. Если связь между коммутаторами S1 и S3 работоспособна, то коммутатор S3 получает пакет BDDP на порт P2 и, в соответствии с правилом для пакетов BDDP пересылает этот пакет на порт CONTROLLER в сообщении packet_in, указывая также номер порта, на который пришел этот пакет, в данном случае порт P2.

Этап 5. Контроллер получает им же сформированный пакет BDDP и на основе его содержимого (S1, P3) и данных сообщения packet_in (P2) делает вывод, что между коммутаторами S1 и S3 существует односторонняя связь (S1, P3)=>(S3, P2), находящаяся в работоспособном состоянии. Для обнаружения связи в противоположном направлении (связи (S1, P3) => (S3, P2)) используются пакеты BDDP, переданные контроллером коммутатору S3 для порта P2. Если контроллер перестает периодически получать пакеты BDDP, тестирующие некоторую связь, то по истечении тайм-аута связь помечается как неработоспособная. Отметим, что контроллер обнаруживает связи только между коммутаторами своего **домена OF SDN –** набора коммутаторов, которые присоединились к данному контроллеру в результате процедуры установления с ним управляющего канала.

Алгоритм автоматического построения топологии сети домена OF SDN использует факт *централизованного управления* сетью. Этим он принципиально отличается от распределенных алгоритмов маршрутизации сетей TCP/IP. Как следствие, данный алгоритм не страдает от периодов нестабильной работы сети, вызванной временным рассогласованием таблиц разных маршрутизаторов.

Развитие протокола OF:

В версии 1.0 протокола OpenFlow предполагается, что коммутатор имеет только одну таблицу продвижения. Такое ограничение может приводить к неэффективным решениям в тех случаях, когда обработка пакета, во-первых, требует проверки нескольких условий, а, во-вторых, проверяемые параметры принимают независимо друг от друга значения из широкого диапазона.

Типичным примером именно такого случая является реализация коммутатором OF *алгоритма прозрачного моста*: здесь требуется проверка двух условий, и параметры – MAC-адреса – принимают значения из множества адресов некоторой локальной сети. Обработка мостом каждого пакета состоит в проверке двух независимых правил, одно из которых связано с проверкой совпадения MAC-адреса источника с адресами, имеющимися в таблице продвижения, а второе – с проверкой совпадения MAC-адреса назначения с этими же адресами. Первая проверка нужна для корректировки таблицы продвижения в тех случаях, когда адрес источника либо отсутствует в таблице, либо соответствует новому значению порта источника. Вторая проверка нужна для принятия решения о продвижении пакета (на один из портов либо на все порты) или же о его отбрасывании. Так как просмотр таблицы коммутатора OF может быть сделан *только один раз*, для учета всевозможных комбинаций MAC-адресов источника и назначения необходимо иметь в таблице OF NxN правил, если в сети имеется N различных хостов. Каждое такое правило может выглядеть так (N1 и N2 взяты из диапазона [1-N]):

Conditions: {MAC Src = N1, Port = m, MAC Dst = N2}

Actions: {Port = p}

В том случае, когда MAC-адреса пакета и порт источника совпадают с условиями одного из правил, пакет передается на порт, заданный в действии. Если же нет, то срабатывает правило-стопор, которое в данном случае говорит, что пакет должен быть послан на виртуальный порт CONTROLLER. Контроллер, получив пакет, должен решить, почему произошел отказ в обработке пакета таблицей и как нужно модифицировать таблицу коммутатора. Например, если отказ таблицы произошел по причине изменения соответствия MAC-адреса источника порту источника, то контроллер должен отозвать старое правило с некорректным соответствием и заменить его новым. Если же появился ранее неизученный MAC-адрес источника, то для него нужно создать новые правила, содержащие этот адрес.

Алгоритм прозрачного моста — не единственный, который приводит к таблицам большой размерности. Любой алгоритм обработки пакетов, который требует нескольких независимых проверок, включающих все пространство MAC-адресов или IP-адресов, порождает проблему размерности таблицы продвижения, если эта обработка должна быть сделана за *один проход* таблицы. Например, рассмотренная ранее в этой главе процедура проверки продвижения по реверсивному пути при маршрутизации траффика группового вещания также требует проверки условий, связанных как с адресом источника, так и назначения. Еще одним примером является поддержка техники VLAN с портами доступа (см. главу 11), которая требует выполнения двух действий с пришедшим пакетом: добавление к заголовку пакета тега с номером VLAN, приписанному к входному порту пакета, и продвижение пакета по его MAC-адресу назначения.

Коммутатор может справится с этой задачей, только если в таблицу продвижения будет добавлено большое количество записей, описывающих все возможные комбинации (Nport x Nmac) проверяемых параметров.

Конвейер таблиц

Кардинальным решением проблемы размерности таблицы продвижения коммутатора стала замена одной большой таблицы **конвейером** (pipeline), составленным из нескольких компактных таблиц (рис. 16.9).

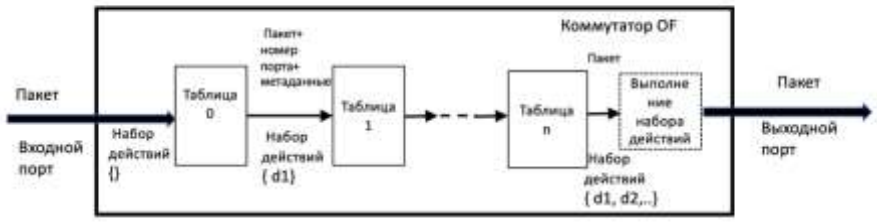

Рис. 16.9. Конвейер таблиц коммутатора SDN

Возможность использования нескольких таблиц упрощает многие задачи. Обратимся снова к примеру обработки трафика по алгоритму прозрачного моста. Если одно условие (проверка MAC-адреса источника для обучения моста) описывается одной таблицей, а второе условие (проверка MAC-адреса назначения для выбора выходного порта) — другой таблицей, то последовательное применение правил каждой из таблиц позволяет решить задачу даже если разрешен только один проход. При этом обе таблицы имеют относительно небольшой размер.

Начиная с версии 1.1 стандарта OF, коммутатор получил возможность поддерживать до 255 таблиц. Каждая таблица состоит из некоторого количества правил, того же формата и назначения, что и правила коммутатора с единственной таблицей. Обработка пакета начинается с таблицы 0. Переход к следующей таблице *всегда* выполняется по **команде GO TO table k,** где k-номер следующей таблицы. Процесс обработки пакета может с помощью команды GO TO k может перемещаться к следующей соседней таблице или «прыгать» через несколько таблиц, при этом перемещение должно быть только вперед, к таблице с большим номером, чем текущая. Это ограничение исключает зацикливание обработки пакета при использовании различных таблиц. Последовательная обработка пакета последовательностью таблиц *без возможности возврата* является конвейерной обработкой, что и дало название этому варианту организации таблиц коммутатора. Пакет переходит от таблицы к таблице вместе со связанными с ним набором отложенных действий и метаданными.

В наборе **отложенных действий** накапливаются действия, которые предполагалось выполнять над пакетом при осуществлении условия каждой из таблиц, через которые прошел пакет. Все отложенные действия выполняются на выходе из коммутатора после прохождения пакетом всех таблиц. И только если таким действием является изменение значения некоторого поля заголовка пакета, коммутатор выполняет его, не откладывая, передавая пакет на обработку следующей таблице в измененном виде.

Метаданные представляют собой поле размером в 64 бита, ассоциированное с пакетом на время прохождении им конвейера таблиц. Это поле служит для того, чтобы одна таблица могла сгенерировать и передать другой таблице некоторую информацию, которую вторая и последующие таблицы могли бы учесть при обработке пакета. Правила таблиц могут включать значение поля метаданных в свои условия так же, как значения полей заголовка пакета. Правила могут и изменять значение метаданных аналогично изменению значений полей заголовка пакета. Например, в правиле некоторой таблицы указано действие, в результате которого в поле метаданных обрабатываемого пакета помещается отметка времени. Значение этого поля затем будет передаваться «по конвейеру таблиц», каждая из которых может использовать отметку времени в правилах обработки пакета.

Рассмотрим другой пример использования метаданных. Пусть коммутатор предоставляет пользователям два различных транспортных сервиса, s1 и s2. Каждый пользователь подключен к индивидуальному порту коммутатора и за каждым портом жестко закреплен тип предоставляемого сервиса. Таблица 0 организована так, что она задает отображение портов на сервисы, то есть содержит два правила, по одному

на каждый пользовательский порт коммутатора. Правила имеют вид:

Conditions:	*Conditions:*
{Port = 1}	*{Port = 2}*
Actions:	*Actions:*
{Metadata = 1, GO TO table 1}	*{Metadata = 2, GO TO table 1}*

Все последующие таблицы используют значение метаданных, созданных таблицей 0, чтобы определить, какой сервис они должны предоставить пришедшему пакету. Для пакетов, у которых Metadata=1, выполняются проверки и действия, соответствующие сервису s1, а для тех, у которых Metadata = 2 – проверки и действия для сервиса s2.

Метаданные дополняют стандартный набор параметров пакета, которые он несет в своем заголовке — это позволяет сделать обработку пакетов более гибкой.

Таблицы коммутатора могут отличаться функциональностью. Например, таблица 0 может допускать только условия, работающие с полями заголовка Ethernet, а таблица 1 – только с полями заголовков IP и TCP/UDP. Эта специфика не определяется стандартом OpenFlow — она оставлена на усмотрение производителя коммутатора.

Развитие функциональности коммутатора

В старших версиях протокола OpenFlow предполагается, что коммутатор может иметь две специальные таблицы — групповую таблицу и таблицу измерителей, которые отличаются по формату и назначению от таблиц продвижения пакетов. Обе эти таблицы повышают автономность и функциональность коммутатора, его способность совершать *без непосредственного участия контроллера* более сложные действия, чем простое продвижение пакетов. Это очень важное свойство, так как любое вмешательство контроллера приводит у существенному замедлению процесса передачи пакетов коммутатором.

ПРИМЕЧАНИЕ

> Однако при усложнении функций коммутатора появляется потенциальный риск скомпрометировать саму идею SDN разделения сетевых устройств на две категории: «умные» контроллеры и «глупые», работающие под их управлением коммутаторы. К тому же увеличение функциональной нагрузки коммутаторов SDN угрожает свести на нет еще одно преимущество устройств SDN – унификацию. Действительно, чем сложнее модель, тем больше вероятность того, что производители начнут реализовывать ее по-своему и снова появятся фирменные коммутаторы, отличные друг от друга.

Групповая таблица (Group Table) состоит из записей типа *{номер группы, тип группы, счетчик, наборы действий}*. Правила таблиц продвижения пакетов могут ссылаться на ту или иную группу из групповой таблицы. Если условие некоторой таблицы продвижения выполняется и в этом правиле есть ссылка на номер группы групповой таблицы, то дальнейшая обработка пакета зависит от типа группы.

Группа типа «Быстрое переключение» (Fast Failover) нужна для автономного принятия решения коммутатором по переходу на резервный путь при отказе какого-либо его порта. В группе данного типа определяется столько наборов действий, сколько существует портов для альтернативных маршрутов к какому-нибудь адресу назначения (если существуют только основной и резервный маршрут, то таких наборов будет два). Коммутатор должен автоматически выявлять работоспособность своих портов и на этом основании определять «жизнеспособность» каждого набора действий. Процедура определения жизнеспособности порта стандартом не оговаривается. Если определены несколько наборов действий, то пакет обрабатывается в соответствии с первым из них. В случае отказа порта соответствующий набор помечается как «нежизнеспособный» и выбирается следующий «жизнеспособный» набор. Обычно набор действий группы этого типа включает как минимум одно действие – отправка пакета на выходной порт. Остальные действия могут быть связаны со спецификой резервного маршрута, например, может быть изменено значение приоритета пакета.

Для *групп типа «Выбор»* (Select) выбирается один из заданных наборов действий случайным или псевдослучайным образом, например, с помощью хэш-функции от адресов источника и назначения. Этот тип группы предназначен для поддержания алгоритмов баланса нагрузки выходных портов, когда в сети существуют альтернативные маршруты.

Для *групп типа «Все»* (ALL) создается столько копий пакета, сколько наборов действий определено в описании группы. Одним из очевидных применений такого типа группы является обработка пакетов с групповыми адресами, которая, напомним, состоит в том, что по мере перемещения такого пакета по сети, он копируется на каждой из «развилок», ведущих к тому или иному члену группы, указанной в адресе данного сообщения. Каждый набор в этой группе действий соответствует порту, на который должен быть отправлен пакет с групповым адресом. Так как маршруты группового трафика проходят в общем случае через несколько портов коммутатора, то число наборов (и число копий пакета) должно быть равно числу участвующих в групповой рассылке выходных портов для данной группы.

Таблица измерителей (Meter Table) служит для автономного выполнения коммутатором операции по *профилированию трафика* (ограничению его скорости). Каждая запись этой таблицы описывает некоторый **измеритель** (Meter) и состоит из двух компонент:

- **диапазон** (band) описывается двумя значениями: скоростью и пульсацией (как мы помним, период усреднения T является производной от этих двух величин). В стандарте определено только два возможных действия при превышении скорости диапазона – отбрасывание пакета и изменение значения поля DSCP пакета;
- действия, которые нужно выполнить над пакетом, если интенсивность потока попадает в данный диапазон.

Измерители, описанные в данной таблице, имеют номера. Правила таблиц продвижения могут ссылаться на эти измерители, указывая их в наборе действий. Само описание измерителя в таблице измерителей не оказывает никакого влияния на потоки пакетов, пока на него не ссылается некоторое правило из таблицы продвижения. Если же такая ссылка есть, то к потоку, который удовлетворяет условиям данного правила, начинает применяться профилирование, то есть отбрасывание пакетов или же их маркирование, что создает предпосылки для последующего отбрасывания другими коммутаторами.

Рассмотрим использование измерителя на следующем примере. Пусть на порт 4 коммутатора поступает поток IP-пакетов от внешней сети, которая не находится под нашим контролем. Мы хотим защитить сеть от перегрузок и ограничить скорость входного потока пределом в 4 Гбит/с. Прежде, чем задать правило обработки этого потока, нужно создать измеритель с нужными параметрами.

Meter 1:

{Type=DROP; Rate = 4G; Burst = 5 GB}

Эта запись создает измеритель с номером 1 в таблице измерителей с диапазоном, определяемым скоростью 4 Гбит/с и пульсацией 5 Гбайт (это дает период усреднения скорости T = Burst x 8/Rate = 10 с). Тип диапазона DROP говорит о том, что пакеты, превышающие предел скорости, будут отброшены, то есть измеритель этого типа работает как элемент профилирования трафика. Наличие этого измерителя позволяет создать правило для потока, поступающего на порт P4 и передаваемого на порт P1:

Priority = 40000

Conditions: {Port = P4; EtherType=0x0800; IP Dest = 194.81.18.0/24*}*

Actions: {Meter 1; Port=P1}

Измерители создают предпосылки для применения методов QoS в сетях OF SDN. Централизованное управление коммутаторами облегчает задачу согласованного применения механизмов QoS, так как приложение, работающее на контроллере сети, знает полные маршруты потоков. Это облегчает выполнение процедур проверки и резервирования ресурсов вдоль пути следования потока.

Проблемы протокола OpenFlow

Распространению сетей OF SDN мешают несколько принципиальных проблем.

Проблема унификации конвейера таблиц. Конвейер таблиц представляет собой мощное и гибкое средство обработки пакетов, одновременно лишая коммутаторы OpenFlow одного из фундаментальных преимуществ, присущих сетевым устройствам SDN перед традиционными коммутаторами – их *унифицированности.* Ситуация, когда коммутатор одного производителя поддерживает конвейер таблиц, несовместимый с конвейером другого производителя, возвращает сетевую отрасль к старому вопросу: как обеспечить независимость разработчиков приложений от конкретной модели коммутатора?

Один из ответов заключается в том, чтобы *переложить все заботы, связанные со спецификой конвейера, на драйверы контроллеров.* Так делается, например, в ОС ONOS, где существует промежуточный слой

Flow Objectives между приложениями и драйверами OF коммутаторов. Этот слой с помощью программного интерфейса предоставляет приложениям возможность задавать правила обработки пакетов потока абстрактно, не уточняя, в какой таблице должно помещаться каждое правило. Драйвер может отказать приложению, обнаружив, что не может реализовать запрос слоя Flow Objectives с помощью имеющегося в его распоряжении конвейера.

Проблема разбора (анализа) заголовка пакета. Заголовок пакета может иметь весьма разнообразную структуру, она зависит от того, какие и в какой последовательности протоколы инкапсулированы в этот пакет. Как было показано выше, версия OpenFlow 1.0 работает с так называемым эталонным заголовком пакета, включающим 11 полей достаточно типичного кадра: это кадр Ethernet с полем тега VLAN, в который вложен пакет IP с сегментами TCP или UDP. Коммутатор OF 1.0, пользуясь такой структурой заголовка, может корректно разбирать заголовки пакетов, но *только этой структуры*. Такой коммутатор не может выделять потоки по меткам MPLS, или же по IPv6 адресам, он просто о них не знает. К тому же такая популярная инкапсуляция как Q-in-Q тоже не будет полностью поддерживаться так как коммутатор знает только о теге VLAN верхнего уровня, но не внутреннего.

Чтобы повысить применимость коммутаторов разработчики последующих версий протокола OpenFlow стали усложнять эталонный заголовок, так в версии OF 1.3 заголовок может состоять уже из 40 различных полей, включая поля протоколов MPLS, ARP, ICMP и IPv6, а в OF 1.5 - 44 поля. Но это не решает проблему окончательно: трудно учесть даже все популярные протоколы в некотором универсальном заголовке, не говоря уже о новых.

Проблема масштабируемости. Каким бы мощным ни был сервер, на котором работает контроллер и приложения SDN, его возможности ограничены и, значит, существует предельное число управляемых коммутаторов, после которого действия контроллера становятся слишком медленными, а память исчерпывается. Централизованное управление всегда страдает от недостаточной масштабируемости, будь то сервер баз данных или сервер службы DNS. Это — та неизбежная плата за привилегию концентрировать вокруг себя все информационные потоки и принимать единоличные решения. Архитекторы Интернета использовали иерархический подход к решению проблемы масштабирования централизованных решений, если они считались по каким-то причинам более выгодными, чем распределенные. Наиболее известным успешным случаем такого подхода является система DNS, построенная на иерархии серверов имен. Так же иерархически строятся справочные системы, например, Microsoft Directory Services, или системы управления сетью NMS (см. главу 25). Но на момент написания этой книги иерархическая архитектура контроллеров OF SDN не утвердилась и протокол взаимодействия контроллеров в такой иерархии не стандартизован, хотя отдельные инициативы в этой области от различных организации по стандартизации появлялись. Соответственно, пока технология OF SDN остается однодоменной.

Прогресс в области производства микросхем за время, прошедшее с момента зарождения технологии OF SDN, привел к тому, что стало возможным построение коммутатора на сравнительно недорогих программируемых микросхемах, в которых функции не являются фиксированными, так как реализуются процессорами. Возможность применения специализированных пакетных процессоров в качестве основы коммутатора привело к появлению новой технологии SDN – технологии P4 SDN, которая является развитием OF SDN и решает ее проблемы на другом уровне.

Программируемые **П**ротокольно-независимые **П**акетные **П**роцессоры (P4)

Основу коммутатора технологии P4 составляет программируемый пакетный процессор – микросхема, функциональные возможности которой программируются на предварительном *этапе конфигурирования* коммутатора. На этом этапе программист задает, какие поля заголовка пакета должны приниматься во внимание в правилах продвижения, какой набор условий и действий допускается применять в правилах продвижения, а также какой набор таблиц может поддерживать конвейер таблиц коммутатора. Упрощенно можно сказать, что программист может задать конфигурацию OF-коммутатора определенной модели компании Cisco, так как эта конфигурация в полной мере подходит требованиям приложения A, для приложения B — определить конфигурацию модели коммутатора HP, а для приложения C — создать уникальную конфигурацию, не встречавшуюся в моделях OF-коммутаторов. После определения архитектуры коммутатора P4 наступает второй этап, который соответствует *этапу загрузки правил* в OF-коммутатор. На этом этапе приложение, работающее на котроллере, создает и загружает в таблицы коммутатора P4 набор правил, определяющих способ обработки и продвижения пакетов при поступлении их на входные порты коммутатора. И, наконец, на третьем *этапе управления* коммутатор P4 начинает обрабатывать пакеты в соответствии с загруженными в него правилами. Этапы работы коммутатора P4

иллюстрирует рис. 16.10.

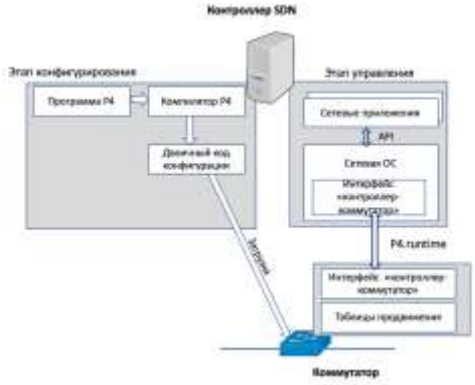

Рис. 16.10. Программное конфигурирование коммутатора P4

На этапе конфигурирования разрабатывается **программа** P4, которую **компилятор** P**4** транслирует в двоичный код конфигурации. Этот двоичный код загружается в коммутатор, после чего он готов принимать правила продвижения пакетов и выполнять их в реальном времени. Программа P4 состоит из нескольких секций: секции описателей полей заголовка, секции синтаксического анализатора (parser) заголовка, секции описания таблиц конвейера, секции условий и секции действий. Для описания элементов конфигурации язык P4 использует синтаксис, близкий к синтаксису языка программирования C. Например, известные структуры заголовков Ethernet и VLAN описываются так:

```
header ethernet {
fields {
dst_addr : 48; //размер поля в битах
src_addr : 48;
ethertype : 16;
  }
}
```

```
header vlan {
fields {
pcp : 3;
cfi : 1;
vid : 12;
ethertype : 16;
  }

}
```

Но одного описания структуры заголовков недостаточно для того, чтобы синтаксический анализатор коммутатора смог понять, в каком порядке заголовки различного типа следуют друг за другом в общем заголовке пакета. Поэтому секцию заголовков дополняет секция синтаксического анализа, которая указывает, заголовок какого типа появляется в пакете первым, и как можно узнать, каким является следующий заголовок. Например, кадр Ethernet всегда начинается с заголовка ethernet, поэтому первым оператором этой секции должен быть оператор:

```
parser start {
ethernet;
}
```

Тип следующего заголовка определяется значением поля ethertype заголовка ethernet, например, значение 0x8100 или 0x9100 говорит о том, что следующим заголовком будет заголовок vlan, а значение 0x8000 – заголовок ipv4. На языке P4 такие указания синтаксического анализатору записываются так:

```
parser ethernet {
switch(ethertype) {
  case 0x8100: vlan;
  case 0x9100: vlan;
  case 0x8000: ipv4;
 //другие случаи
   }
}
```

```
parser vlan {
  switch(ethertype) {
  case 0x9100: vlan;
  case 0x800: ipv4;
 //другие случаи
    }

}
```

Конструкция switch-case взята из языка C, в данном случае она говорит синтаксическому анализатору о том, какой тип будет у следующего заголовка при определенном значении поля ethertype.

Для реализации второго и третьего этапов можно было бы использовать уже существующий протокол OpenFlow, но архитекторы P4 решили заменить его новым **протоколом p4.runtime**, так как в сообщениях OpenFlow за типами заголовков и действий жестко закреплены определенные числовые значения, и это

делает его зависимым от конфигурации коммутатора. Программируемость архитектуры контроллера P4 решает проблему унификации коммутаторов OF – вместо описания всеобъемлющей архитектуры, которая должна включать все возможные варианты структуры заголовка пакета, все возможные условия и действия над пакетами создается возможность запрограммировать желательную для определенного класса приложений архитектуру коммутатора. Во время написания книги коммутаторы P4 еще не получили поддержки основных производителей сетевого оборудования, но ее гибкость и естественное развитие базовых принципов, заложенных в технологию SDN OpenFlow, позволяют рассчитывать на ее успех в будущем.

Мы описали две технологии SDN, основанные на совершенно новой модели архитектуры коммутатора. Существует и более осторожный подход к воплощению идеи программно-определяемых сетей, развиваемый специалистами ITEF. Этот подход базируется на принятии решения о маршрутизации некоторым центральным элементом, называемым **элементом вычисления маршрута** (PCE, Path Computation Element). В сети IETF SDN используются традиционные маршрутизаторы и в этом проявляется «осторожность» данного подхода, так как здесь не требуется замены традиционных устройств на устройства совершенно нового типа. Для того, чтобы маршрутизатор мог отработать маршрут, вычисленный PCE, он должен поддерживать маршрутизацию от источника — эту давно известную, но пока мало востребованную функция маршрутизаторов.