P4: Programming Protocol-Independent Packet Processors

P4 - программируемые, независимые от протокола процессоры пакетов

Pat Bosshart[1], Dan Daly[2], Glen Gibb[1], Martin Izzard[1], Nick McKeown[3], Jennifer Rexford[4], Cole Schlesinger[4], Dan Talayco[1], Amin Vahdat[5], George Varghese[6], David Walker[4]

Тезисы

P4 - язык высокого уровня для программирования независимых от протоколов сетевых процессоров. P4 работает совместно с протоколами управления SDN, такими как OpenFlow. В настоящее время OpenFlow явно задает протокольные заголовки, с которыми ведется работа. Набор этих полей расширился за несколько лет с 12 до 41, что усложнило спецификацию, но не обеспечивает достаточной гибкости добавления новых заголовков. В этой статье предлагается язык P4 как вариант будущего развития OpenFlow. Преследуются три цели: (1) возможность изменения конфигурации в «полевых» условиях, (2) независимость от протоколов (коммутатору не следует опираться на какой-либо из имеющихся протоколов) и независимость от платформы, позволяющая программистам описывать функциональность обработки пакетов без учета специфики оборудования, на котором будет происходить обработка. Например, будет описано, как можно использовать P4 для настройки коммутатора с целью добавления новой иерархической метки.

1. Введение

Программно-определяемые сети (SDN[7]) дают операторам возможность программного управления их сетями. В SDN уровень управления физически отделен от уровня пересылки и один уровень управления может контролировать множество устройств пересылки. Хотя устройства пересылки могут программироваться множеством способов, наличие общего, открытого и независимого от производителей интерфейса (такого как OpenFlow) позволяет уровню управления контролировать устройства с оборудованием и программами разных производителей.

Таблица 1. Поля, распознаваемые стандартом OpenFlow.

Версия	***Дата выпуска***	***Число полей заголовков***
OF 1.0	декабрь 2009	12 (Ethernet, TCP/IPv4)
OF 1.1	февраль 2011	15 (MPLS, межтабличные метаданные)
OF 1.2	декабрь 2011	36 (ARP, ICMP, IPv6 и т. п.)
OF 1.3	июнь 2012	40
OF 1.4	октябрь 2013	41

Интерфейс OpenFlow начинался с простой абстракции, содержащей 1 таблицу правил, которая позволяла сопоставлять пакеты по дюжине полей заголовков (например, адреса MAC и IP, протокол, номера портов TCP/UDP и т. п.). За 5 лет спецификация сильно расширилась и усложнилась (Таблица 1), были добавлены другие поля заголовков и многоэтапные таблицы правил, позволяющие коммутаторам раскрыть больше своих возможностей контроллеру.

Процесс расширения числа полей не показывает признаков остановки. Например, операторы ЦОД[8] все чаще хотят применять новые формы инкапсуляции пакетов (например, NVGRE, VXLAN, STT) и для этого развертывают программные коммутаторы, функциональность которых проще расширить. Вместо безостановочного расширения спецификации OpenFlow мы предлагаем поддерживать в будущих коммутаторах гибкие механизмы анализа и сопоставления полей пакетов, позволяющие контроллерам расширять свои возможности, используя базовый открытый интерфейс (например, новый OpenFlow 2.0 API). Такой обобщенный расширяемый подход будет проще, элегантней и перспективней сегодняшних стандартов OpenFlow 1.x.

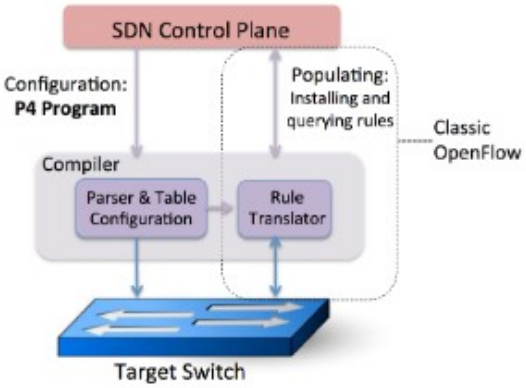

Рисунок 1. P4 как язык программирования коммутаторов.

Недавно разработанные микросхемы показывают, что такая гибкость может быть достигнута в ASIC при терабитных скоростях [1, 2, 3]. Программирование нового поколения микросхем коммутации далеко не просто. Каждая микросхема имеет свой низкоуровневый интерфейс, похожий на программирование микрокода. В этой статье кратко описан язык

[1]Barefoot Networks
[2]Intel
[3]Stanford University
[4]Princeton University
[5]Google
[6]Microsoft Research
[7]Software-Defined Networking
[8]Центр обработки данных

высокого уровня для программирования независимых от протоколов процессоров пакетов P4[1]. На рисунке 1 показаны связи между P4, используемым для настройки коммутатора (как обрабатывать пакеты), и имеющимися интерфейсами API (такими, как OpenFlow), которые предназначены для заполнения таблиц пересылки в коммутаторах с фиксированной функциональностью. P4 поднимает уровень абстракции для программирования сети и может служить базовым интерфейсом между контроллером и коммутаторами. Предполагается, что будущие версии OpenFlow позволят кнтроллеру сообщать коммутаторам, как им следует работать, а не ограничиваться заданием фиксированной конфигурации. Основной задачей является поиск баланса между выразительностью языка и простотой реализации на разных программных и аппаратных коммутаторах. При разработке P4 ставились три основных цели, описанные ниже.

- *Настраиваемость*. Контроллер должен иметь возможность переопределять анализ и обработку пакетов в процессе работы.
- *Независимость от протоколов*. Коммутатор не должен быть привязан к определенному формату пакетов. Вместо этого контроллеру следует предоставить возможность задать (i) анализатор для извлечения полей заголовков по именам и типам, а также (ii) набор типизованных таблиц «сопоставление-действие» (match+action) для обработки этих заголовков.
- *Независимость от платформы*. Как программист С может абстрагироваться от специфики процессора (CPU), так и программисту контроллеров следует обеспечить возможность не знать деталей работы коммутатора, для которого программа предназначена. Компилятор должен учитывать возможности коммутатора при преобразовании независимого от платформы описания (на языке P4) в предназначенную для целевой платформы программу для найтройки конфигурации коммутатора.

В статье сначала вводится абстрактная модель пересылки в коммутаторе. Затем разъясняется необходимость создания нового языка для описания независимой от протоколов обработки пакетов. После этого рассматривается простой пример, когда оператор хочет поддерживать новое поле заголовка и обрабатывать пакеты в несколько этапов. Это служит для объяснения того, как P4 программирует заголовки, анализатор пакетов, множество таблиц match+action и потоки управления. В заключение описано, как компилятор P4 может отображать программы на целевые коммутаторы.

Предшествующие работы. В 2011 году Yadav с соавторами [4] предложили абстрактную модель пересылки для OpenFlow, но без акцента на компиляторы. Kangaroo [1] ввел понятие программируемого анализа. Недавно Song [5] предложил не связанную с протоколом пересылку, которая соответствует заявленным в статье целям, но ориентирована на сетевые процессоры. В ONF были разработаны шаблоны типизации таблиц для описания возможностей сопоставления в коммутаторах [6]. Недавняя работа NOSIX [7] соответствует цели гибкого задания таблиц match+action, но не рассматривает независимость от протоколов и язык для задания анализаторов, таблиц и управления потоками. В других недавних работах предлагается программируемый интерфейс с уровнем данных для мониторинга, контроля перегрузок и управления очередями [8, 9]. Модульный маршрутизатор Click [10] поддерживает гибкую обработку пакетов на программном уровне но не отображает ее на разные аппаратные платформы коммутации.

2. Абстрактная модель пересылки

В предлагаемой абстрактной модели (Рисунок 2) коммутатор пересылает пакеты через программируемый анализатор, за которым следует несколько этапов «сопоставление-действие», выполняемых последовательно, параллельно или в комбинации этих вариантов. Выведенная из OpenFlow, эта модель имеет три обобщения. Во-первых, OpenFlow предполагает фиксированный анализатор, а новая модель - программируемый, который позволяет определять новые заголовки. Во-вторых, OpenFlow предполагает последовательные этапы match+action, а в новой модели могут использоваться также параллельные. В третьих, новая модель предполагает транзакции на основе независимых от протокола примитивов, поддерживаемых коммутатором.

Эта модель обобщает обработку пакетов в разных устройствах пересылки (например, в коммутаторах Ethernet, балансировщиках нагрузки, маршрутизаторах) на основе разных технологий (например, коммутаторы на основе ASIC и NPU с фиксированной функциональностью, перенастраиваемые и программные коммутаторы, FPGA). Это позволяет разработать общий язык (P4) описания обработки пакетов. Это дает программистам возможность создавать независимые от платформы программы, которые компилятор преобразует в разные устройства пересылки — от небольших и сравнительно медленных коммутаторов до высокопроизводительных устройств на базе ASIC.

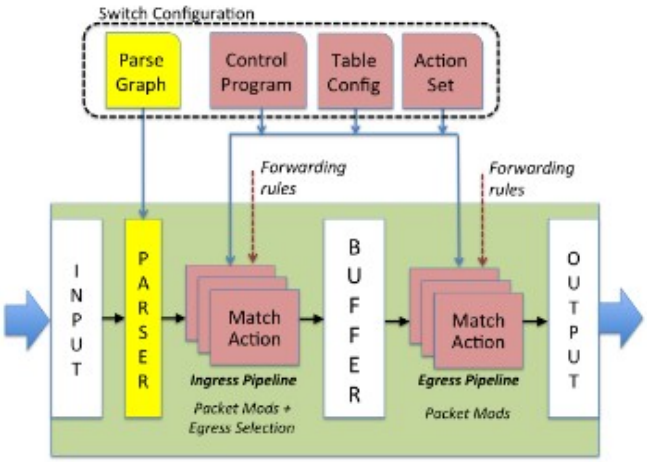

Рисунок 2. Абстрактная модель пересылки.

Управление моделью пересылки включает два типа операций - настройку (Configure) и заполнение (Populate). Операции настройки программируют анализатор, задают порядок этапов match+action, а также указывают поля заголовков, обрабатываемые на каждом этапе. Конфигурация определяет поддерживаемые протоколы и способы

[1]Programming Protocol-independent Packet Processors.

обработки пакетов коммутатором. Операции заполнения добавляют и удаляют записи в таблицы match+action, заданные при настройке конфигурации. Заполнение определяет правила, применяемые к пакетам в каждый момент.

В этой статье фазы настройки и заполнения считаются раздельными. В частности, коммутатору не требуется обрабатывать пакеты в процессе настройки конфигурации. Однако предполагается, что реализации будут поддерживать обработку пакетов в процессе полной или частичной перенастройки конфигурации, что позволит обновлять коммутаторы без прерывания работы. Модель осознанно допускает и поддерживает перенастройку конфигурации без прерывания пересылки.

Очевидно, что фаза настройки имеет мало значения для коммутаторов ASIC с фиксированной конфигурацией. В этом случае задача компилятора состоит лишь в проверке поддержки коммутатором программ P4. Instead, our goal is to capture the general trendtowards fast reconfigurable packet-processing pipelines, asdescribed in [2, 3].

Принимаемые пакеты сначала обрабатываются анализатором. Предполагается, что тело пакета (без заголовков) буферизуется отдельно и недоступно для операций сопоставления. Анализатор распознает и извлекает поля заголовка, определяя протоколы, поддерживаемые коммутатором. Модель не включает допущений о назначении протокольных заголовков , считая лишь, что анализируемое представление определяет набор полей для которых применяются сопоставления и действия.

Извлеченные поля заголовков передаются в таблицы match+action. Эти таблицы делятся на входные и выходные. Хотя оба типа таблиц могут менять заголовки пакетов, входные таблицы определяют выходной порт или порты и очередь, в которую помещается пакет. На основе входной обработки пакет может быть переслан, реплицирован (для групповой рассылки проброса или отправки на уровень управления), отброшен, а также могут выполняться действия потока управления. Выходные таблицы match+action вносят изменения в заголовки отдельных пакетов (например, для группового копирования). С потоком можно связать таблицы действий (счетчики, указатели и т. п.) для отслеживания состояний кадр за кадром.

Пакеты могут переносить между этапами дополнительную информацию (метаданные), которые трактуются аналогично полям заголовков. Примерами метаданных могут служить входной порт, направление передачи и очередь, временная метка и передаваемые между таблицами данные, которые не связаны с полученным при анализе пакета представлением (например, идентификатор VLAN).

Дисциплины очередей обрабатываются как в OpenFlow - действия сопоставляют пакеты с очередями, для которых настроена определенная дисциплина обслуживания. Эти дисциплины (например, минимальная скорость, DRR) выбираются при настройке конфигурации коммутатора.

Хотя это и не рассматривается в статье, могут добавляться примитивы действий, позволяющие программистам реализовать новые или имеющиеся протоколы контроля перегрузок. Например, коммутатор можно запрограммировать на установку бита ECN на основании новых условий или задать свой механизм контроля перегрузок с использованием таблиц match+action.

3. Язык программирования

Абстрактная модель пересылки используется для определения языка, позволяющего выразить конфигурацию коммутатора и способы обработки пакетов. Основной целью статьи является предложение языка программирования P4. Однако ясно, что возможно существование множества языков, обладающих описываемыми здесь характеристиками. Например, языки должны выражать способы программирования анализаторов, чтобы те знали какой формат пакетов следует ожидать. Поэтому программисту нужен способ указания возможных типов. Например, программист может задать формат заголовков IPv4 и указать какие заголовки могут следовать после заголовка IP. Это служит мотивом определения анализа в P4 путем объявления допустимых типов заголовков. Например, поля TTL должны декрементироваться и проверяться, может потребоваться добавление новых туннельных заголовков, а также расчет контрольных сумм. Это побуждает использовать в P4 обязательную программу для описания обработки полей заголовков, использующую объявленные типы заголовков и примитивы набора действий.

Можно применять такие языки как Click [10], которые создают коммутаторы из модулей, написанных на C++. Язык Click является очень выразительным и хорошо подходит для описания обработки пакетов в ядре CPU. Но он недостаточно ограничен для рассматриваемой задачи, поскольку она требует языка, отображающего конвейеры parse-match-action на конкретное оборудование. Кроме того, язык Click не предназначен для архитектуры «контроллер-коммутатор» и не позволяет программистам описать таблицы match+action, которые динамически заполняются четко типизованными правилами. Наконец, в Click сложно вывести зависимости, ограничивающие параллельное выполнение.

Язык описания обработки пакетов должен позволять программисту выразить (явно или неявно) любую последовательность зависимостей между полями заголовка. Зависимости определяют какие таблицы можно применять параллельно. Например, требуется последовательное применение для таблицы маршрутизации IP и таблицы ARP, поскольку данные этих таблиц связаны зависимостью. Зависимости можно найти анализируя графы зависимости таблиц (TDG[1]). Эти графы описывают входные данные полей, действия и поток управления обменом данными между таблицами. На рисунке 3 показан пример графа зависимостей между таблицами для коммутатора L2/L3. Узлы TDG отображаются непосредственно на таблицы match+action и анализ зависимостей показывает, где каждая из таблиц может размещаться в конвейере обработки. К сожалению TDG неохотно воспринимаются большинством программистов, которые предпочитают использовать для алгоритмов обработки пакетов императивные конструкции вместо графов.

В результате предлагается двухэтапный процесс компиляции. На верхнем уровне программисты задают программы обработки пакетов с использованием императивного языка, представляющего поток управления (P4), затем компилятор транслирует представление P4 в TDG для упрощения анализа зависимостей. В заключение TDG отображается на конкретную платформу коммутации. Компилятор P4 предназначен для упрощения трансляции программ P4 в TDG. Таким образом, P4 можно рассматривать как компромисс между общностью (скажем, Click), которая осложняет обнаружение зависимостей и их отображение на оборудование, и гибкостью OpenFlow 1.0, которая не позволяетперенастраивать протокольную обработку.

[1]Table Dependency Graph

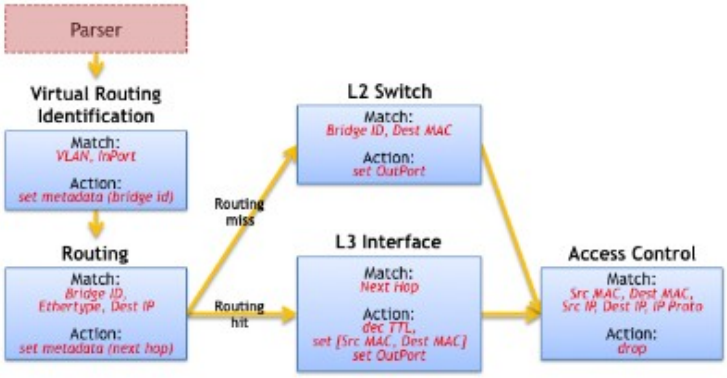

Рисунок 3. Граф связей между таблицами в коммутаторе L2/L3.

4. Пример использования языка P4

Рассмотрим P4 на простом примере. Многие сети разделены на ядро (core) и периферию (edge), оконечные хосты подключаются к периферийным устройствам, которые соединены между собой через высокопроизводительное ядро. Для поддержки такой архитектуры разработаны специальные протоколы (например, MPLS [11] и PortLand [12]), нацеленные на упрощение пересылки в ядре.

Рассмотрим пример сети L2 со стоечными коммутаторами (ToR[1]), связанные через двухуровневое ядро. Предположим, что число хостов возросло и таблицы L2 переполняются. MPLS помогает упростить ядро, но реализация протокола распространения меток со множеством тегов является непростой задачей. Протокол PortLand представляется интересным, но он требует переписывания MAC-адресов (это может препятствовать работе отладочных инструментов в сети), а также использования новых агентов для обработки запросов ARP. P4 позволяет описать решение с минимальными изменениями архитектуры сети. В приведенном примере mTag объединяет иерархическую маршрутизацию PortLand с простыми тегами в стиле MPLS. Маршруты через ядро представляются 32-битовыми метками, состоящими из 4 однобайтовых полей. Такая метка может указывать маршрут, заданный отправителем (source route) или указатель получателя (destination locator), подобно PortLand Pseudo MAC. Каждому коммутатору в ядре нужно проверить лишь один байт метки и выполнить коммутацию не его основе. В примере это метка, добавленная первым коммутатором ToR, но она может добавляться и сетевым адаптером хоста. NIC. Пример mTag преднамеренно упрощен для фокусировки на языке P4. Программа P4 для реального коммутатора будет значительно больше и сложнее.

4.1 Концепции P4

Программа P4 содержит определения перечисленных ниже компонент.

- *Заголовки* (Header). Определение заголовка описывает порядок и структуру последовательности полей, включая их размеры и ограничения для возможных значений.
- *Анализаторы* (Parser). Определение унализатора указывает способы идентификации заголовков и корректный порядок заголовков в пакетах.
- *Таблицы* (Table). Таблицы «сопоставление-действие» (match+action) служат механизмами обработки пакетов. Программа P4 задает поля, которые могут использоваться в сопоставлениях и действиях.
- *Действия* (Action). P4 поддерживает создание сложных действий (операций) из простых примитивов, независимых от протоколов. Эти операции доступны в таблицах match+action.
- *Программы управления* (Control Program) определяют порядок применения к пакету таблиц match+action. Это простая императивная программа, описывающая прохождения пакета через таблицы.

Далее описано использование этих компонент для определения идеализированного процессора mTag в P4.

4.2 Форматы заголовков

Разработка начинается с задания форматов заголовков. Для этого было предложено несколько языков с разными сферами применения [13, 14, 15], которые были заимствованы в P4. В общем случае каждый заголовок указывается путем задания упорядоченного списка имен полей с их размерами. Необязательные аннотации полей позволяют ограничить диапазоны значений или максимальный размер полей переменного размера. Например, стандартные заголовки Ethernet и VLAN можно задать, как показано ниже.

```
header ethernet {
       fields {
              dst_addr : 48; // размер в битах
              src_addr : 48;
              ethertype : 16;
       }
}
header vlan {
       fields {
              pcp : 3;
              cfi : 1;
              vid : 12;
              ethertype : 16;
       }
}
```

[1]Top-of-rack - букв., на вершине стойки.

Заголовок mTag можно добавить без влияния на имеющиеся определения. Имена полей показывают, что ядро имеет два уровня агрегирования. Каждый коммутатор ядра программируется с использованием правил для проверки одного из байтов по его местоположению в иерархии и направлению (вверх или вниз).

```
header mTag {
       fields {
              up1 : 8;
              up2 : 8;
              down1 : 8;
              down2 : 8;
              ethertype : 16;
       }
}
```

4.3 Анализатор пакетов

P4 предполагает, что коммутатор может реализовать машину состояний (конечный автомат), которая перебирает заголовоки пакета от начала к концу, извлекая значения по мере их поступления. Полученные значения полей передаются в таблицы match+action для обработки. P4 описывает машину состояний напрямую как набор переходов от одного заголовка к другому. Каждый из переходов может быть вызван значениями в текущем заголовке. Например, машину состояний mTag можно описать, как показано ниже.

```
parser start {
       ethernet;
}
parser ethernet {
       switch(ethertype) {
              case 0x8100: vlan;
              case 0x9100: vlan;
              case 0x800: ipv4;
              // Другие варианты
       }
}
parser vlan {
       switch(ethertype) {
              case 0xaaaa: mTag;
              case 0x800: ipv4;
              // Другие варианты
       }
}
parser mTag {
       switch(ethertype) {
              case 0x800: ipv4;
              // Другие варианты
       }
}
```

Анализ начинается из состояния start и выполняется до следующего явного состояния stop или возникновения особой (не обрабатываемой) ситуации, которая может считаться ошибкой. Извлеченные заголовки передаются в обработку match+action в конвейере коммутатора.

Анализатор для mTag очень прост и имеет лишь 4 состояния. В реальных сетях анализаторы включают гораздо больше состояний. Например, анализатор, определенный Gibb и др. [16, рисунок 3(e)], имеет сотни состояний.

4.4 Задание таблиц

Затем программист описывает, как указанные поля заголовков сопоставляются на этапах match+action (например, точное или шаблонное соответствие) и какие действия выполняются при совпадении.

В примере mTage периферийный коммутатор сопоставляет получателя L2 destination и VLAN ID, а затем выбирает mTag для добавления в заголовок. Программист определяет таблицу для сопоставления с этими полями и применения действия по добавлению заголовка mTag (см. ниже). Атрибут reads указывает сопоставляемые поля с учетом типа сопоставления (точное или иное). Атрибут actions указывает действия, которые могут быть выполнены в таблице применительно к пакету. Эти действия рассматриваются в следующем параграфе. Атрибут maxsize задает число элементов, которые могут быть включены в таблицу.

Спецификация таблицы позволяет компилятору определить размер требуемой для нее памяти и тип этой памяти (например, TCAM или SRAM).

```
table mTag_table {
       reads {
              ethernet.dst_addr : exact;
              vlan.vid : exact;
       }
       actions {
              // В процессе работы записи программируются
              // параметрами действия mTag.
              add_mTag;
       }
       max_size : 20000;
}
```

Для полноты и последующего обсуждения представлены краткие определения других таблиц, на которые ссылается программа управления (4.6 Программа управления).

```
table source_check {
	// Проверять mtag только на портах в ядро
	reads {
		mtag : valid; // mtag был проанализирован?
		metadata.ingress_port : exact;
	}
	actions { // Каждая запись таблицы задает 1 действие.
		// Если mTag не применим, выполняется передача в CPU
		fault_to_cpu;
		// При наличии метки mtag она вырезается и записывается в метаданные
		strip_mtag;
		// В противном случае обработка пакета продолжается
		pass;
	}
	max_size : 64; // Одно правило на порт
}
table local_switching {
	// Определяется получатель и проверяется его локальность
	// При отсутствии переход в таблицу mtag.
}
table egress_check {
	// Проверяется определение выходного порта.
	// Пакеты м меткой не помечаются заново
	// Считывается выходной порт и проверяется наличие mTag
}
```

4.5 Задание действий

P4 определяет набор примитивой действий (операций), из которых можно создавать более сложные операции. Каждая программа P4 задает набор функций-действий, состоящих из таких примитивов. Эти функции просто определяют спецификации и заполнение таблиц. P4 предполагает параллельное выполнение примитивов внутри функций (коммутаторы, не поддерживающие параллельное выполнение могут исключить его).

Действие по добавлению mTag, упомянутое выше, реализуется в показанной ниже форме.

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
	add_header(mTag);
	// Копируется VLAN ethertype в mTag
	copy_field(mTag.ethertype, vlan.ethertype);
	// Устанавливается ethertype для VLAN с целью указания mTag
	set_field(vlan.ethertype, 0xaaaa);
	set_field(mTag.up1, up1);
	set_field(mTag.up2, up2);
	set_field(mTag.down1, down1);
	set_field(mTag.down2, down2);
	// Указывается выходной порт
	set_field(metadata.egress_spec, egr_spec);
}
```

Если действию нужны параметры (например, значение up1 для mTag), они представляются из таблицы сопоставления.

В этом примере коммутатор помещает mTag после тега VLAN, копирует VLAN Ethertype в mTag и устанавливает Ethertype для тега VLAN в 0xaaaa для указания mTag. Обратная операция вырезания mTag из пакета и таблица для применения этого действия в периферийных коммутаторах не показаны.

Примитивы действий P4 включают:

- *set_field* устанавливает значение заданного поля в заголовке (поддерживаются маски полей);
- *copy_field* копирует одно поле в другое;
- *add_header* указывает конкретный экземпляр заголовка (и все его поля) как действительный (пригодный);
- *remove_header* удаляет (выталкивает - pop) заголовок (и все его поля) из пакета;
- *increment* инкрементирует или декрементирует значение поля;
- *checksum* рассчитывает контрольную сумму для некоторых полей заголовка (например, IPv4).

Предполагается, что большинство реализаций коммутаторов будет ограничивать выполнение действий изменениями заголовков, совместимыми с заданным форматом пакетов.

4.6 Программа управления

После определения таблиц и действий остается лишь задать поток управления из одной таблицы в следующую. Этот поток задается как программа из набора функций, условий и ссылок на таблицы.

На рисунке 4 приведено графическое представление желаемого потока управления для реализации mTag на периферийном коммутаторе. После анализа таблица source_check проверяет согласованность полученного пакета и входного порта. Например, меткам mTag следует появляться только на портах, подключенных к коммутаторам ядра. таблица source_check также вырезает метки mTag из пакетов, записывая факт их наличия в метаданные. Последующие табоицы конвейера могут сопоставлять эти метаданные, чтобы пакет не был помечен заново. Затем

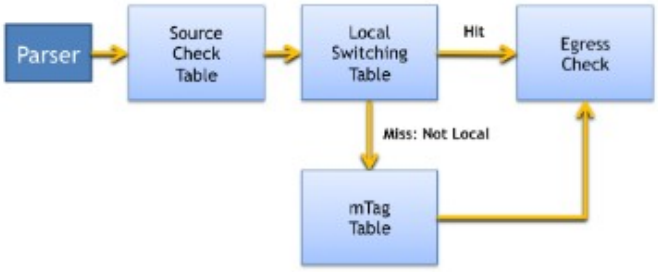

Рисунок 4. Диаграмма переходов для mTag.

применяется локальная таблица коммутации. Если таблица показывает «отсутствие», это говорит о том, что пакет не адресован ни одному из локально подключенных хостов. В этом случае к пакету применяется таблица mTag (см. выше). Управление локальной пересылкой и пересылкой в ядре может осуществляться в таблице egress_check, которая обрабатывает пакеты для неизвестных адресатов путем отправки уведомления стеку управления SDN. Императивное представление такого конвейера обработки пакетов приведено ниже.

```
control main() {
       // Проверка согласованности состояния mTag и порта
       table(source_check);
       // При отсутствии ошибок в source_check обработка продолжается
       if (!defined(metadata.ingress_error)) {
              // Попытка коммутации оконечным хостам
              table(local_switching);
              if (!defined(metadata.egress_spec)) {
                     // Нет известного локального хоста, попытка установить mTag
                     table(mTag_table);
              }
              // Проверка неизвестного выходного состояния или неверной
              // перемаркировки (retagging) с использованием mTag.
              table(egress_check);
       }
}
```

5. Компиляция программ P4

Для реализации в сети программ P4 нужен компилятор, отображающий независимое от платформы описание на оборудование конкретного целевого коммутатора или программную платформу. Это включает выделение ресурсов целевой платформы и создание подходящей для устройства конфигурации.

5.1 Компиляция анализатора пакетов

Для устройств с программируемыми анализаторами компилятор преобразует описание анализатора в машину состояний, а для фиксированных анализаторов лишь проверяет совместимость описания с анализатором целевой платформы. Создание машины состояний и записей талицы состояний описано в [16].

В таблице 2 показаны записи таблицы состояний для разделов анализатора vlan и mTag (4.3 Анализатор пакетов). Каждая запись указывает текущее состояние соответствующее значение поля и следующее состояние. Остальные элементы записи для краткости опущены.

Таблица 2. Записи таблицы состояний анализатора для примера mTag.

Текущее состояние	***Искомое значение***	***Следующее состояние***
vlan	0xaaaa	mTag
vlan	0x800	ipv4
vlan	*	stop
mTag	0x800	ipv4
mTag	*	stop

5.2 Компиляция программ управления

Императивное представление потока управления в параграфе 4.6 Программа управления является удобным способом задания логического поведения коммутатора при пересылке, но не связано явно с зависимостями между таблицами и возможностью параллельного выполнения. Поэтому компилятор используется для обнаружения зависимостей и определения возможностей параллельного выполнения. В финале компилятор генерирует целевую конфигурацию для коммутатора. Имеется множество разных целевых систем, например, программные коммутаторы [17], многоядерные программные коммутаторы [18], NPU [19], коммутаторы с фиксированной конфигурацией [20], конвейеры с перенастраиваемой таблицей сопоставления (RMT[1]) [2].

Как было отмечено в раздел 3. Язык программирования, компилятор использует двухэтапную трансляцию. Сначала преобразуется программа управления на языке P4 с промежуточный граф зависимостей, который анализируется для поиска зависимостей между таблицами. Затем зависящий от целевой платформы транслятор (back-end) отображает граф на соответствующие ресурсы коммутатора.

Ниже кратко описана возможная реализация mTag в коммутаторах разных типов.

Программные коммутаторы обеспечивают полную гибкость - число таблиц, их конфигурация и анализатор управляются на программном уровне. Компилятор напрямую отображает граф таблицы mTag на таблицы коммутатора. Компилятор использует информацию о типе таблиц для ограничения их размеров и критериев

[1]Reconfigurable match table

сопоставления (например, точное, по префиксу или шаблону) для каждой таблицы. Компилятор может также оптимизировать сопоставление со структурами данных программы.

Аппаратные коммутаторы с RAM и TCAM. Компилятор может настроить хэширование для проверки точного совпадения с использованием RAM для периферийных коммутаторов mTag. В коммутаторах ядра таблица пересылки mTag, соответствующая подмножеству битов тега, будет отображаться в TCAM.

Коммутаторы, поддерживающие параллельные таблицы. Компилятор может обнаружить зависимости между данными и настроить таблицы для параллельного или последовательного применения. В примере mTag таблицы mTag и локальная коммутация могут выполняться параллельно выполнению действий по установке mTag.

Коммутаторы, применяющие действия в конце конвейера. Для таких коммутаторов компилятор может задать на промежуточных этапах создание метаданных, которые будут применяться при финальной записи. В примере mTag добавление или удаление mTag может быть представлено в метаданных.

Коммутаторы с малым числом таблиц. Компилятор может отображать большое число таблиц P4 на меньшее число физических таблиц. В примере mTag локальная коммутация может комбинироваться с таблицей mTag. При установке коммутатором новых правил в процессе работы, компилятор транслирует «составные» правила в две таблицы P4 для генерации одной физической таблицы.

6. Заключение

SDN позволяет одному уровню управления напрямую контролировать целую сеть коммутаторов. OpenFlow поддерживает эту цель, обеспечивая один, независимый от производителя интерфейс API. Однако сегодняшние коммутаторы OpenFlow с фиксированной функциональностью распознают лишь предопределенный набор полей и обрабатывают пакеты, используя лишь небольшой набор предопределенных действий. Уровень управления не может выразить способ наиболее эффективной обработки пакетов в соответствии с требованиями программ управления.

Здесь предложен новый шаг к созданию более гибких коммутаторов, конфигурация которых задается (и может быть изменена) в полевых условиях. Программист задает обработку пакетов уровнем пересылки, не заботясь о деталях реализации. Компилятор преобразует императивную программу в граф зависимостей между таблицами, который может быть отображен на конкретные целевые коммутаторы с учетом оптимизации под конкретное оборудование.

Подчеркнем, что это лишь первый шаг, подготовленный как вклад в обсуждение OpenFlow 2.0. В этом предложении некоторые аспекты коммутаторов остаются неопределенными (например, примитивы контроля перегрузок, дисциплины очередей, мониторинг трафика). Однако предполагается, что наличие языка управления конфигурацией и компилятора, генерирующего низкоуровневые конфигурации для целевых устройств, обеспечат будущим коммутаторам большую гибкость и раскроют возможности программно-определяемых сетей.

7. Литература

[1] C. Kozanitis, J. Huber, S. Singh, and G. Varghese, “Leaping multiple headers in a single bound: Wire-speed parsing using the Kangaroo system,” in IEEE INFOCOM, pp. 830–838, 2010.

[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese,N. McKeown, M. Izzard, F. Mujica, and M. Horowitz,“ Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN,” in ACM SIGCOMM, 2013.

[3] “Intel Ethernet Switch Silicon FM6000.” http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf.

[4] N. Yadav and D. Cohn, “OpenFlow Primitive Set.” http://goo.gl/6qwbg, July 2011.

[5] H. Song, “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,” in SIGCOMM HotSDN Workshop, Aug. 2013.

[6] “Openflow forwarding abstractions working group charter.” http://goo.gl/TtLtw0, Apr. 2013.

[7] M. Raju, A. Wundsam, and M. Yu, “NOSIX: A lightweight portability layer for the SDN OS,”ACM SIGCOMM Computer Communications Review, 2014.

[8] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazieres, “Tiny packet programs for low-latency network control and monitoring,” in ACM SIGCOMM HotNetsWorkshop, Nov. 2013.

[9] A. Sivaraman, K. Winstein, S. Subramanian, andH. Balakrishnan, “No silver bullet: Extending SDN to the data plane,” in ACM SIGCOMM HotNets Workshop, Nov. 2013.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F.Kaashoek, “The Click modular router,” ACM Transactions on Computer Systems, vol. 18,pp. 263–297, Aug. 2000.

[11] “Multiprotocol Label Switching Charter.” http://datatracker.ietf.org/wg/mpls/charter/.

[12] R. Niranjan Mysore, A. Pamboris, N. Farrington,N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand: A scalable fault-tolerant layer 2 data center network fabric,” in ACM SIGCOMM, pp. 39–50, Aug. 2009.

[13] P. McCann and S. Chandra, “PacketTypes: Abstract specification of network protocol messages,” in ACM SIGCOMM, pp. 321–333, Aug. 2000.

[14] G. Back, “DataScript - A specification and scripting language for binary data,” in Generative Programming and Component Engineering, vol. 2487, pp. 66–77,Lecture Notes in Computer Science, 2002.

[15] K. Fisher and R. Gruber, “PADS: A domain specific language for processing ad hoc data,” in ACM Conference on Programming Language Design and Implementation, pp. 295–304, June 2005.

[16] G. Gibb, G. Varghese, M. Horowitz, andN. McKeown, “Design principles for packet parsers,” in ANCS, pp. 13–24, 2013.

[17] “Open vSwitch website.” http://www.openvswitch.org.

[18] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G.Andersen, “Scalable, high performance ethernet forwarding with CuckooSwitch,” inCoNext, pp. 97–108, 2013.

[19] “EZChip 240-Gigabit Network Processor for Carrier Ethernet Applications.” http://www.ezchip.com/p_np5.htm.

[20] “Broadcom BCM56850 Series.” https://www.broadcom.com/products/Switching/Data-Center/BCM56850-Series.

Николай Малых

nmalykh@protocols.ru