OpenFlow1.3核心概念翻译与理解

Author: Wu Chenglin

Email: [alexapollo0@gmail.com](mailto:alexapollo0@gmail.com)

# 0 译者言 - OpenFlow

## 0.1前因与心得

* OpenFlow 1.0早已有了中文版本，但1.3迟迟没有译文，遂自己写写对OpenFlow的翻译与理解。
* OpenFlow是一个力图简洁的协议，事实上，协议大多很简洁，复杂都是因为不断的扩展导致的逻辑与对象的混乱。在OpenFlow 1.3里，很多概念已经逐渐不是那么容易理解，而协议本身也有许多未定义的地方，要靠交换机自己实现。
* OpenFlow的核心思想是将所有的协议都抽象出来，抽象成公共的flow概念，所以OpenFlow本身是“大而全”的，理论上它应该能组合出任意的协议。
* 注：本文使用官方SPEC的目录顺序来进行讲解。

## 0.2协议的抽象

**所有协议，无论如何，都不会脱离以下几个抽象概念：**

* 数据：在链路上传输的内容。
* 函数：某一种对数据的处理方式。
* 逻辑：数据与处理的对应关系，以及函数之间交互的行为与时序。

比如有函数： **add(int a, int b); minus(int a, int b);** 这里add/minus即函数，a/b即数据。

* 假如我们需要知道1+3-5等于多少，那么就要先调用add对1+3做处理，再调用minus对4-5进行处理——这就是逻辑了。我们可以将它称为(1+3-5)协议。

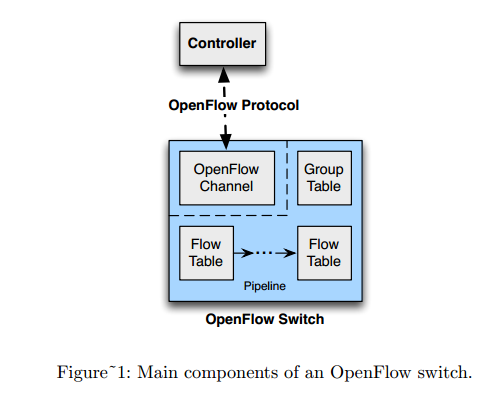
对于vlan，就有函数**push\_vlan(int vid); pop\_vlan(); match\_vlan(int vid);**

* 常见逻辑：对于一台虚拟机发往另一个虚拟机（经过虚拟交换机）的报文，虚拟交换机调用push\_vlan(10)给它加一个vlan，再送给match\_vlan(10)对应的端口集。假如对方也是一台虚拟机，那么就pop\_vlan()再发给它。这里vid是数据，push/pop/match是函数，逻辑如上，三者组合一起，就是基础的vlan协议了。

可以很容易的想到：协议大多是可替换/非唯一的。OpenFlow可以用RESTful API快速实现，不一定要单独定义OF的数据。但如果使用者想要获得最高的效率、普适的标准，那么单独定义一个协议是必须的，因为这样数据才能达到最简洁，没有任何冗余。坏处也很明显：缺乏扩展性，想要添加一个功能十分费力。

# 1 简介

本文阐述了实现一个OpenFlow交换机的要求。在阅读本文之前，我们建议读者阅读最新版的OpenFlow白皮书。白皮书可以在Open Networking Foundation website (<https://www.opennetworking.org/standards/intro-to-openflow>) 上找到。本文涵盖了OpenFlow中各个对象的概念、交换机的基础函数，以及控制交换机和控制器（Controller）交互的OpenFlow协议。



# 2 交换机对象

一个OpenFlow交换机由一个或多个**流表**和一个**组表**组成，它们决定了报文匹配与转发，以及一个与控制器建立连接的OpenFlow通道(Figure 1)。交换机和控制器通过OpenFlow协议通信，而控制器通过OpenFlow协议控制交换机。

通过OpenFlow协议，控制器可以**增删改**交换机流表中的流表项。相应的，控制器增删改流表项有两种模式：被动与主动。每个交换机中的流表都拥有一个流表项的集合。而每个**流表项**都由 **匹配域、计数器、指令集** 组成，报文通过流表项进行匹配与执行。

报文从第一个**流表**开始匹配，并且也可能跳转到另一个流表。流表项按照优先级对报文进行匹配，使用匹配到的第一个流表项(see 5.3)。如果报文匹配了一个流表项，那么报文将会执行流表项中的指令集。如果报文没有在流表中匹配到流表项，那么此时会以一个默认动作进行处理（这个动作由交换机配置决定），如：丢弃报文、对下一张流表进行匹配、转发给控制器等等(see 5.4)。

* **与流表项关联的指令集包含了动作**，或者会修改流水线处理流程。在指令中包含的动作描述了报文如何转发，报文的修改和组表的处理。在指令集中没有指定跳转表动作后，流水线停止，此时报文往往会被修改或者被转发。
* **流表项可能设定将报文转发到一个端口上**，通常是一个物理口，但也可以是逻辑口或保留口（4.1）。保留口可能指定一类转发动作，比如发给控制器，泛洪，用非OF方法转发，比如“normal”的交换机处理方式（译者注：传统的交换机MAC-PORT学习，4.5）；交换机定义的逻辑口也可能指定了链路聚合组，隧道，或者环回等接口。
* **与流表项关联的动作可以将报文定向到一个组上**。组指定对报文进行额外处理（5.6）。组代表了一个动作集，一般用于泛洪及更复杂的转发语义（如多路径，快速重路由，链路聚合）。作为间接、通用的一层，组可以接受由多条流定向到同一个组的做法（如IP转发到同样的下一跳）。可以快速的为不同流表项改变输出动作也是这个抽象层的优点。
* **组表包含了组**。每个组都包含了一个动作桶的列表，并也有一个指定的组类型。在报文送给组之后，将会执行组里的一个或多个动作（由组类型指定）。

**交换机设计者可以自由的用任何方法实现这些特性**，只需要保证必须的匹配项和指定语义是正确的就可以。举个例子，有一条流表项可能会指定一个类型为“all”的组，这个组是进行组播给对应的端口的，这时交换机设计者就可以选择一个硬件的端口掩码来让硬件进行组播。另一个例子是，OF交换机的匹配表可以用硬件表来实现（译者注：如CAM）。

# 3 常用术语

本节介绍OpenFlow的关键术语：

* **字节（Byte）**：一个8 bit位组。
* **报文（Packet）**：一个以太网帧，包含报文头和负载。
* **端口（Port）**：报文进入与退出OpenFlow流水线的地方。可能是物理口，交换机定义的逻辑口，OpenFlow协议定义的保留口。
* **流水线（Pipeline）**：报文的实际匹配、修改、转发等行为都在流水线中进行。
* **流表（Flow Table）**：包含了流表项，是流水线的一部分。
* **流表项（Flow Entry）**：流表的一个原子，用于匹配并处理报文，包含了一个匹配域的集合来匹配报文，一个优先级项来决定由哪个匹配报文的流表项对报文进行处理，一个计数器的集合用来记录报文，以及一个指令集来最终进行处理。
* **匹配域（Match Field）**：包括了报文头，入端口，元数据。一个匹配域应该可以被通配，可以拥有掩码。
* **元数据（Metadata）**：一个可屏蔽的记录值，用于携带交换机内的信息，从一张表到另一张表。
* **指令（Instruction）**：报文匹配之后就运行指令，可能是对流水线处理的修改，比如将报文送往另一个流表，或者给报文对应的动作集加上一系列的动作，也可能是马上对报文执行一个动作列表。
* **动作（Action）**：转发报文到端口，或者修改报文，比如递减TTL。动作可能被流表项中的指令集包含，也可能被包含在一个组里。动作可以通过动作集累积，或者通过动作列表立即对报文执行。
* **动作集（Action Set）**：当报文匹配时，往往会在动作集里累积动作；而在报文退出流水线处理时（指令集中没有跳转表），那么动作集就会按制定顺序执行。
* **组（Group）**：一个动作桶的列表。
* **动作桶（Action Bucket）**：动作集与相关参数，组专用。
* **标签（Tag）**：可以用push/pop来插入删除的报文头。
* **最外层标签（Outermost Tag）**：顾名思义，最外面的报文头。
* **控制器（Controller）**：一个和OpenFlow交换机通过OpenFlow协议进行交互的实体。
* **流控（Meter）**：交换机元素之一，用于检测和控制报文速率。一般用法是：当报文速率超过流控预先定义的限速阈值时，Meter将会触发一个meter band。如果该meter badnd丢弃该数据包，那么它被称为**限速器（Rate Limiter）**。

# 4 端口

OpenFlow端口有三种类型：**物理、逻辑、保留**。物理口顾名思义就是对应具体的物理设备；逻辑口是一个抽象概念，往往用于非OF端口（tunnel口）；保留口就是特殊的代号，如“controller”，指的是一类端口。

**标准端口**：物理、逻辑，以及保留中的LOCAL端口，可以用于匹配和动作，也具有常规PORT的信息。

* **物理口**：连物理设备的
* **逻辑口**：多个tunnel id，其余信息和物理口一样
* **保留口：**
  + **ALL**, 除了设置不转发及入端口外的所有端口
  + **CONTROLLER**, 即和控制器通信的口
  + **TABLE**, 仅在packet-out消息中有用（控制器要求发送报文，可先走管道匹配）
  + **IN\_PORT**, 入端口
  + **ANY**, 只是一个端口通配符
  + **LOCAL**, 本地的网络协议栈，可以用于实现和in-band controller的通信。（需要特殊设置）
  + **NORMAL***,* 即MAC-LEARNING对应的口，自动转发
  + **FLOOD**, 泛洪，发给非BLOCK的口，可以指定VLAN

# 5 表

## 5.1 Pipeline Processing（流水线处理）

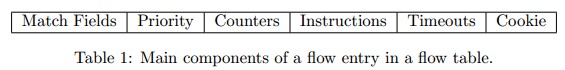
**表里面比较重要的概念是pipeline**，管道，或者也可以称作报文**“流水线”**。意思就是整个OpenFlow协议在交换机里处理的流程。管道可以用三个步骤来描述：

1. 找到最高优先级的匹配项
2. 将指令（instructions）应用到packet、match、action上，并更新对应信息（goto table也是个instruction，约束是只能往更大的table跳）
3. 将匹配项和动作集送往下一个table

如果没有在table中匹配，那么有多种默认行为可以实现，如丢包、继续下一个表、发往controller。

## 5.2 Flow Table（流表）

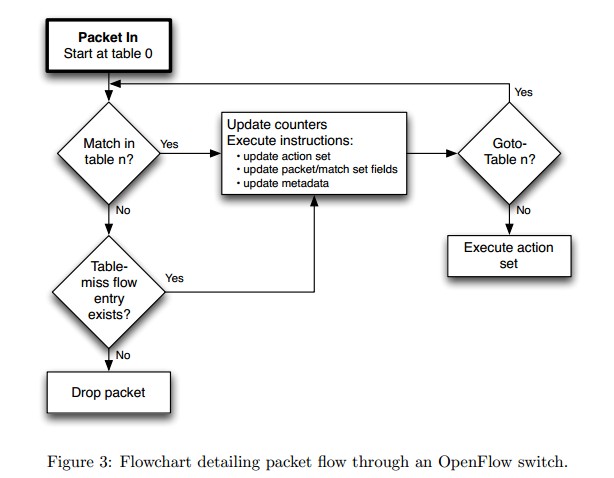
流表由Flow Entry组成，如图所示：

[](http://www.anwcl.com/wordpress/wp-content/uploads/2013/05/flow_entry.jpg)

**以上这几项就是Table章我们所要学习的所有东西了**。都在这儿，妥妥的。无非就是对packet根据某些字段进行匹配，取出匹配优先级最大的并计数，然后对这个packet做Instructions（比如更改动作集）。一条流表项本身具有timeout，就是过一会就消失了，需要重新去问controller来获得新的流表项；流也具有cookie，也即一个唯一标识符。

## 5.3 Matching（匹配）

流匹配，如图所示，先从table 0 开始匹配，不断的update action set（通过instructions），如果不跳转表那么就即刻执行action set。如果发生了table miss，也即没有match任何一条表项，那么可以丢包/发给controller/特殊处理等。

[](http://www.anwcl.com/wordpress/wp-content/uploads/2013/05/flow_matching.jpg)

大部分match的行为都是直观的，比如mac匹配，ipv4匹配。值得一提的是metadata，在spec中写到“Metadata may be used to pass information between tables in a switch.” 意思就是metadata(可以)用作table之间的标记。

有几个细节是值得说清楚的：报文在一个表中只会匹配优先级最大的一条flow，但如果有多条优先级相同的flow，那么这个flow选取动作在spec中是未定义的。OVS会直接随机挑选一条。（这个场景仅在CONTROLLER不设置OFPFF\_CHECK\_OVERLAP项时才会出现，此时存在“重叠流”）

IP分片会在进入pipeline过程前直接重组好。所以在这里可能要考虑分片的性能问题。

## 5.4 Table-miss

每个流表都有一条(隐形)默认规则，也即table-miss规则。当一个报文进入一张流表，但没有任何flow entry匹配时，就会默认进入table-miss规则（实际上它是一条优先级0的通配规则）。默认行为可以是以下几种（包括但不限于）：

* 丢包
* 发给controller
* 发给另一张表

实际上这条规则和其他规则几乎是完全等同的，可以被controller动态的控制，也可以有老化时间。但table-miss而送往controller时必须要标明这是一个table-miss事件。

如果table-miss不存在，那么也会有默认行为，可能是drop（较大概率），当然配给controller也是有道理的，这要看具体的实现了。

## 5.5 Flow Removal（流移除）

流移除当前只有两种情况：一种是controller要求移除，另一种是超时（idle或者hard）老化。但不管如何流移除后都会告诉controller。

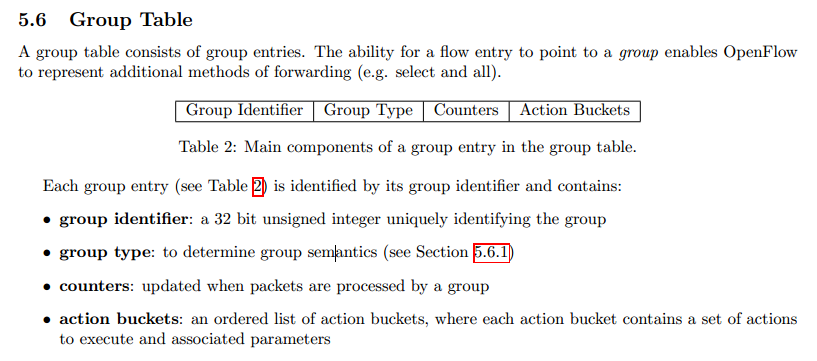
## 5.6 Group Table（组表）

一个比较大的更新，group table，俗称组表。组表里最小的元素都是动作。但我们需要注意spec里是这么描述组表的：组表由group entry组成，而group entry最重要的元素就是action buckets。而action buckets是an ordered list of action buckets.. 实际上表示的就是无序动作集。

好像上面逻辑写的有点不清晰。简单的说，group entry就是一个列表，列表里每个元素都是一个action set（无序动作集）。猜想的一条group entry是这样的：

{ **{encap\_vxlan(10000)}**, {encap\_vlan(2), flood, normal}, {send to vtep} }

更细节的原文如下：



### 5.6.1 Group Types

注意到还有一个比较重要的字段：group type（实际上应该说是group entry type），OpenFlow 1.3指定了四种类型的Group Type。交换机不必支持所有的Group Type。但有一些Group Type是交换机必须支持的，如all和indirect；而标注Optional的Group Type为可选项。

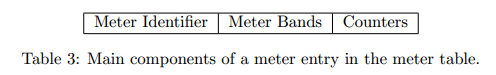
* *Required*: **all**: 执行group entry里的所有buckets。这种组一般用于多播或广播。对于每一个bucket，它高效地复制一个packet，即每个组的bucket处理对应的报文。但注意，如果该bucket将packet显式地发往ingress端口，那么这个packet将会被默认丢弃。如果控制器确实想把该packet发到ingress端口，那么这个组必须要包含一个额外的bucket，它包含了一个动作output:OFPP\_IN\_PORT。
* *Optional*: **select**: 选择group entry里的一个bucket运行。选择算法可以是hash，也可以是个简单的循环（这个算法不由OF协议定义）。当某个bucket指定的端口的状态变为down时，交换机不应该再选择该bucket，而应该采取其他的bucket，从而减小端口down时带来的链路中断的影响。
* *Required*: **indirect**: 执行group entry中的一个指定bucket（唯一），这个entry理应只有一个bucket，这样可以有更高的效率。和all type时只有一个bucket其实是完全等同的。
* *Optional*: **fast failover**: 在和controller断链时使用。执行第一个存活的bucket（存活指的是具有对应的端口、可通信），实际上这个bucket往往是一个底层自动转发的命令。如normal。注：这种group类型需要有一个完整的“存活检验”机制来保证。

## 5.7 Meter Table（米表）

Meter表包含了许多meter entry。Meter对各个流进行监控，它使得OpenFlow可以实现各种简单的QoS（如流控），并且它能和per-port queues（见5.12）结合来实现复杂的QoS功能，如DiffServ等。

一个meter对特定的数据包的速率进行监控并对它进行控制。Meter直接与流表对应（与队列相反，队列与端口对应）。一个流表项可以在它的instruction中指定一个meter（见5.9），该meter对所有指定了它为meter的数据包的总速率进行监控。在同一张流表中，可以指定多个meter，但它们不能被指定在同一个流表项中。

meter entry的细节如图：

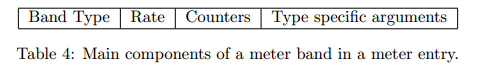


* **Meter Identifier**: 唯一标识符。
* **Meter Bands**: 一个包含了多个meter band的无序列表，每个meter band都指定了带宽、以及处理报文的方法。
* **Counters**: 通过本entry的报文计数器

### 5.7.1 Meter Band

每一个meter可以含有一个或多个meter band。每一个band指定了报文的速率和超过该速率时的处理方式。交换机基于当前检测的meter速率对报文进行处理。Meter将执行低于当前速率的最高配置速率的meter band。如果当前的速率低于任何meter band的速率，则不执行meter band。

一个meter band的具体构成如图所示：



* **Band Type**: 决定packet处理方式（超过Rate后）
  + *Optional*: **drop**: 和正常的QoS一样，超过限速就丢包
  + *Optional*: **dscp remark**: 修改报文的DSCP字段，详见DSCP协议。[一个第三方的讲解](http://kanghestef.blog.51cto.com/77886/191587)。
* **Rate**: 指定了meter执行该meter band的最低速率，往往称为保证带宽。
* **Counters**: 计数器

## 5.8 Counters（计数器）

没有什么特别的，就是走了规则就计数。但实际上有很多特性（统计）交换机不必去实现。这里的原则是大类都得实现（如port、flow），但里面很多子特性不要求每一条都能支持。

## 5.9 Instructions（指令）

这个词不是很好理解。Instructions的意思是“立即执行的命令”。在这里我简单的把它译为**指令**。

指令的应用场景：在flow table中，某一个报文匹配到了一条flow，那么就会执行flow entry中写好的Instructions，这个Instructions可能是修改报文字段，改metadata，以及增加或清空action set等等。

* 可选指令: **Meter meter id** : 将报文定向到一个meter上进行QoS。
* 可选指令: **Apply-Actions action(s)**: 应用一个或多个具体的动作，但不变更当前的action set。如果是多个actions，那么应该为一个链表。这个Instruction的场景一般是执行多个同类型action，或者修改报文等。
* 可选指令: **Clear-Actions**: 清空动作集。
* 必选指令: **Write-Actions action(s)**: 将action加入到action set中，如果这个action类型已经在set中存在，那么覆盖set中对应的action，否则添加。
* 可选指令: **Write-Metadata metadata / mask** : 使用mask和metadata来修改当前的metadata，使用mask是为了表示哪些位需要更新。
* 必选指令: **Goto-Table next-table-id** : 跳转到另一张表。跳表的原则是只能往更大的table id跳，不能往更小的跳。

一个交换机可能对flow entry中的Instructions没有支持能力。那么在给它下发flow entry时（flow mod）交换机就会回复一个“错误：不支持该flow”信息。

注：在一条流表项的Instruction字段上，每种类型的Instruction最多只能出现一次。Instruction按顺序执行。**当前对Instruction的约束**是：Meter Instruction必须要在Apply-Actions Instruction之前执行、Clear-Actions Instruction必须要在Write-Actions Instruction之前执行，Goto-Table Instruction必须在末尾执行。

## 5.10 Action Set（动作集）

**动作集（Action set）是从属于每个报文的**。默认该set为空。

Action set随着报文进入流表，一旦匹配了流表项，那么流表项中的指令就会得到执行。如果执行的是Write-Action / Clear-Action的指令，那么就会更新Action set。

如果在Instructions中有Goto-Table指令，那么Action set也会被携带到下一张表中。如果没有Goto-Table指令，那么Action set在当前Instructions处理完后就会马上执行。

如上文所讲，Action set中每个field类型的set-field动作都只能有一个。如果用户需要执行多个相同类型的动作，那么应该要用Apply-Actions指令。

**无论Instruction的顺序如何，Action set中的动作都应该按下面的顺序执行**。如果Action set中含有group（组），那么组里的action bucket中的动作也应该按照以下顺序来执行。

注：交换机可以选择性支持Apply-Actions中的乱序动作。

|  |
| --- |
| 1. copy TTL inwards: 对报文进行向内层拷贝TTL 的动作 2. pop:对报文执行所有pop tag的动作 3. push-MPLS: 对报文进行打 MPLS tag 的动作 4. push-PBB:对报文进行打 PBB tag 的动作 5. push-VLAN: 对报文进行打 VLAN tag的动作 6. copy TTL outwards: 对报文进行向外层拷贝TTL 的动作 7. decrement TTL: 执行减少报文TTL的动作 8. set:对报文应用所有 set-field的动作 9. qos:应用所有QoS动作，如给报文设置队列ID。 10. group: 如果在动作中制定了一个group，那么应用这个group：但注意应用动作的顺序也是按照本表。 11. output: 如果没有制定group动作，那么可以使用output命令你个执行转发报文。 |

Output动作是最后才进行处理的。如果在Action set中既有output动作又有group动作，那么output此时自动被忽略，只会执行group动作。如果没有output动作也没有group动作，那么这个报文就应该被丢弃（可能是其他默认行为？）。理论上，group动作是可以递归的（前提是交换机支持）。

## 5.11 Action List

Apply-Actions指令以及Packet-out消息都包含了一个动作列表（action list）。顾名思义，在action list上的所有动作都是**按顺序**执行的，并且会马上应用到对应的packet上。

* Action list的效果是“递增”的，意思就是如果用户尝试PUSH了两次VLAN，那么报文确确实实会得到两个VLAN头。
* 如果在Action list上有output动作，那么应该有一个报文拷贝通过对应的端口发送出去。
* 如果在Action list上有group动作，那么应该有一个报文拷贝给group并执行对应的group buckets。

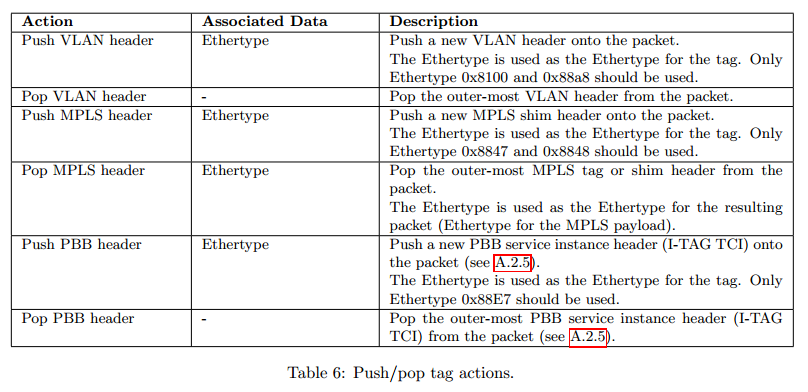
在Apply-Actions指令中的Action list执行完后，pipeline直接在被执行的报文上继续做处理。但报文对应的action set是不变的。

## 5.12 Actions

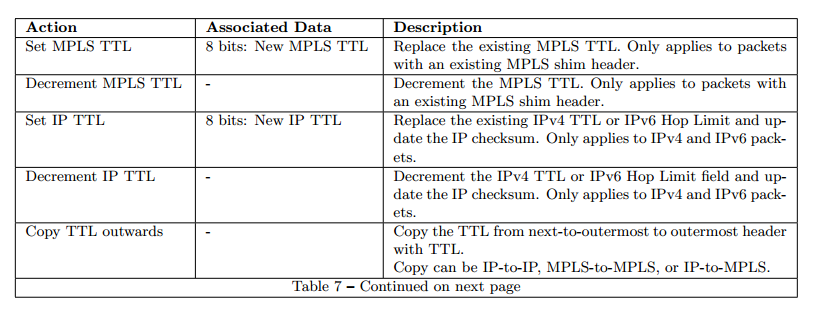
交换机没有必要支持所有的动作，只需要优先支持“必选动作”，Controller会在需要时请求交换机支持动作的列表（理应在握手时获得）。

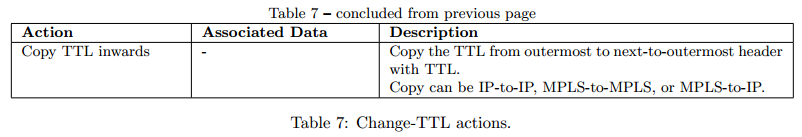
|  |
| --- |
| * 必选动作: Output. 将报文发给指定的OpenFlow端口(4.1)。OpenFlow交换机必须支持转发给物理口、交换机定义的逻辑口，以及必须的保留口(4.5)。 * 可选动作: Set-Queue. 设置报文的队列ID。每当一个报文被使用output命令转发到某个OpenFlow端口上，队列ID决定了这个报文要使用哪个关联到端口上的队列来进行调度和转发。转发行为由队列的配置来决定，并且这个队列是用于基础的流量控制 (QoS) 支持的 (see section A.2.2). * 必选动作: Drop. 没有显示的动作表示丢弃报文。取而代之的判断方法是：假如报文的动作集中没有output的动作，那么它就应该被丢弃。有以下几个可能导致这个动作：报文对应一个空指令集，或者空的动作桶，或者执行了一个“清空所有动作”的指令。 * 必选动作: Group. 用指定的组来处理报文。具体的处理方法由group的类型决定。 * 可选动作: Push-Tag/Pop-Tag. 交换机可以支持push/pop tags 的能力。对应Table 6。为了拥有与现有网络良好的融合能力，我们建议支持push/pop vlan标签。 |

详细描述：



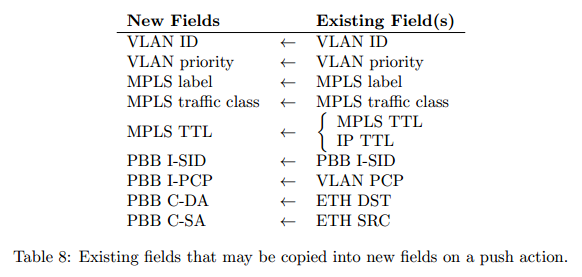
|  |
| --- |
| * 可选动作: Set-Field. 各种Set-Field动作均由它们的field type指定，并且它们会修改对应的报文头中的字段。尽管这个动作不是必选的，但是它大幅提高了OpenFlow的易用性。（毕竟重写header的动作比pop/push快很多）   为了帮助与现有网络的融合，我们建议支持修改VLAN的动作。Set-Field动作应当以最外层的头作为修改目标 (如：一个设置VLAN ID的动作总是设置最外层的VLAN标签)。只在field type不同时，才可能选择不同层。   * 可选动作: Change-TTL. 各种Change-TTL动作修改IPv4 TTL的值，IPv6 跳数限制，或者报文中的MPLS TTL。尽管此动作不是必须的，但在表7中列出的动作非常有用，尤其是在用OpenFlow对路由协议进行实现时。   Change-TTL动作必须仅修改最外层头。(should always be applied to the outermost-possible header.) |





OpenFlow交换机会检验报文的TTL或MPLS-TTL字段，如果非法则会拒绝报文。但注意一点：不是每个报文都需要检验TTL，但在做减少TTL的action时，必须检验。每当交换机发asynchronous配置消息时，可能发出带有非法TTL的packet-in报文。(?)

### 5.12.1 Push的默认值



push一个头之后，报文可以用set的操作来修改。

注意：这里没有set vlan的动作。

# 6 通道

这里通道指的是switch和controller之间连接的通道。一般来说通道传输的报文都是基于TLS的。而TLS一般也是基于TCP的，所以看起来和SSL很相似。

通过通道，OpenFlow 交换机和Controller建立连接，发送/接收标准OpenFlow消息。

## 6.1 OpenFlow协议概览

OpenFlow协议支持三种协议类型：

* controller-to-switch：controller发给switch的消息
* asynchronous：switch发给controller的消息
* and symmetric：双方打哈哈的消息

每个消息都包含若干子消息类型。具体如下。

### 6.1.1 Controller-to-Switch

### 6.1.2 Asynchronous

### 6.1.3 Symmetric

## 6.2 Message Handling

## 6.3 OpenFlow Channel Connections

### 6.3.1 Connection Setup

### 6.3.2 Connection Interruption

### 6.3.3 Encryption

### 6.3.4 Multiple Controllers

### 6.3.5 Auxiliary Connections

## 6.4 Flow Table Modification Messages

## 6.5 Group Table Modification Messages

## 6.6 Meter Modification Messages

# 7 OpenFlow协议（Appendix）