OpenFlow1.3核心概念翻译与理解

Author: Wu Chenglin、Lin Jiashuo

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

# 译者言 - OpenFlow

## 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。

## 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。

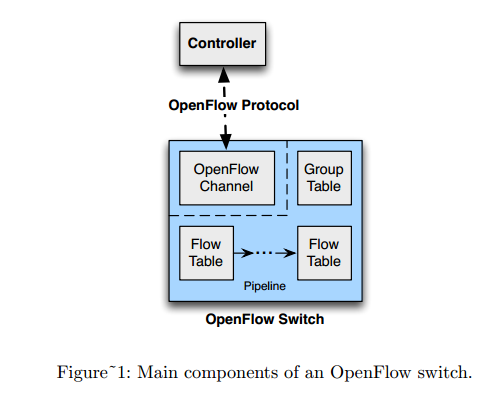
对于vlan，就有函数push\_vlan(int vid); pop\_vlan(); match\_vlan(int vid);

* 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。

# 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。



# 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）。

# 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）**。

# 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

# 表

## Pipeline Processing（流水线处理）

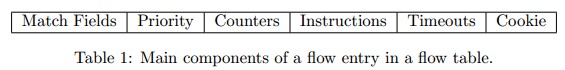
**表里面比较重要的概念是pipeline**，管道，或者也可以称作报文**“流水线”**。意思就是整个OpenFlow协议在交换机里处理的流程。管道可以用三个步骤来描述：

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。

## Flow Table（流表）

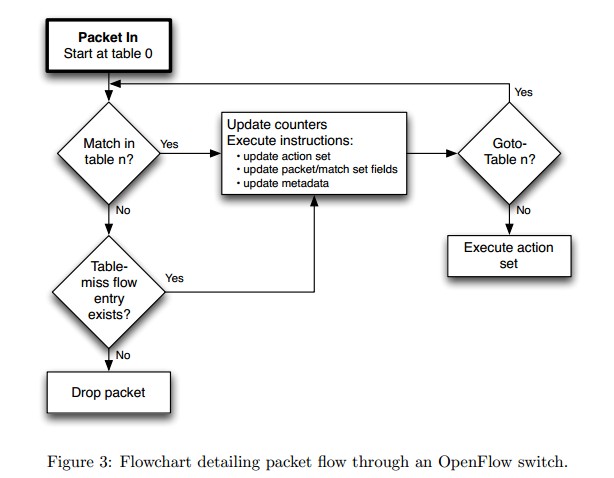
流表由Flow Entry组成，如图所示：

[](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。

## 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。

## 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。

## 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。

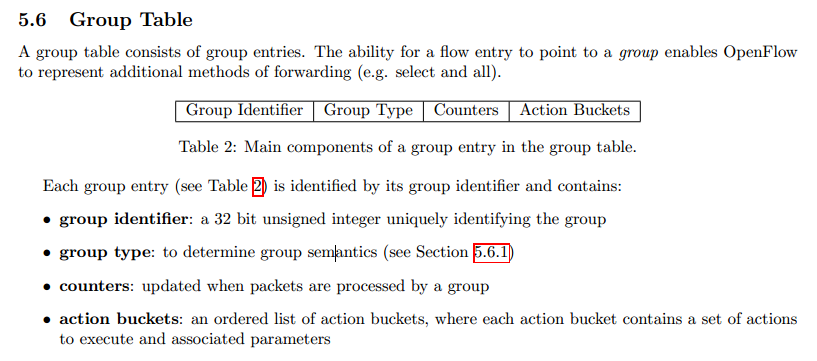
## Group Table（组表）

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：



### 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。

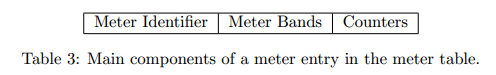
* *Required*: **all**: 执行group entry里的所有buckets。这种组一般用于多播或广播。对于每一个bucket，它高效地复制一个packet，即每个组的bucket处理对应的报文。但注意，如果该bucket将packet显式地发往ingress端口，那么这个packet将会被默认丢弃。如果控制器确实想把该packet发到ingress端口，那么这个组必须要包含一个额外的bucket，它包含了一个动作output:OFPP\_IN\_PORT。
* *Optional*: **select**: 选择group entry里的一个bucket运行。选择算法可以是hash，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。

## 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：

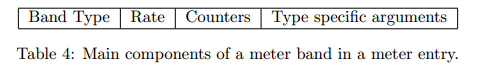


* Meter Identifier: 唯一标识符。
* Meter Bands: 一个包含了多个meter band的无序列表，每个meter band都指定了带宽、以及处理报文的方法。
* Counters: 通过本entry的报文计数器

### 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。
* **Counters**: 计数器

## 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。

## 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。

一个交换机可能不支持flow entry中的Instruction。那么在控制器给它下发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。

## 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。

|  |
| --- |
| * copy TTL inwards: 对报文进行向内层拷贝TTL 的动作 * pop:对报文执行所有pop tag的动作 * push-MPLS: 对报文进行打 MPLS tag 的动作 * push-PBB:对报文进行打 PBB tag 的动作 * push-VLAN: 对报文进行打 VLAN tag的动作 * copy TTL outwards: 对报文进行向外层拷贝TTL 的动作 * decrement TTL: 执行减少报文TTL的动作 * set:对报文应用所有 set-field的动作 * qos:应用所有QoS动作，如给报文设置队列ID。 * 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。 * 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）。

## 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。

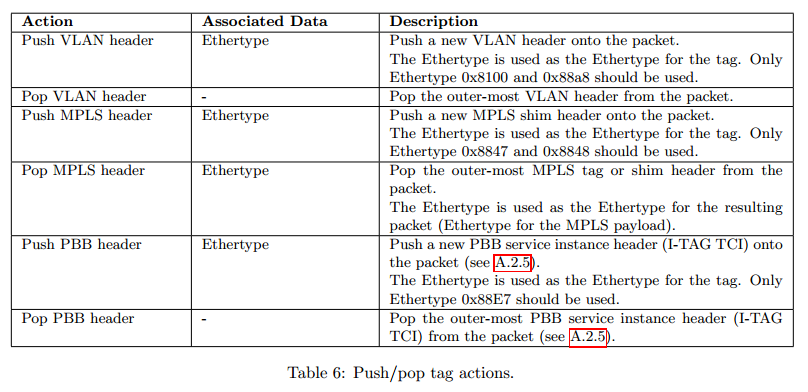
在Apply-Actions指令中的Action list执行完后，pipeline直接在被执行的报文上继续做处理。但报文对应的action set是不变的。

## Actions

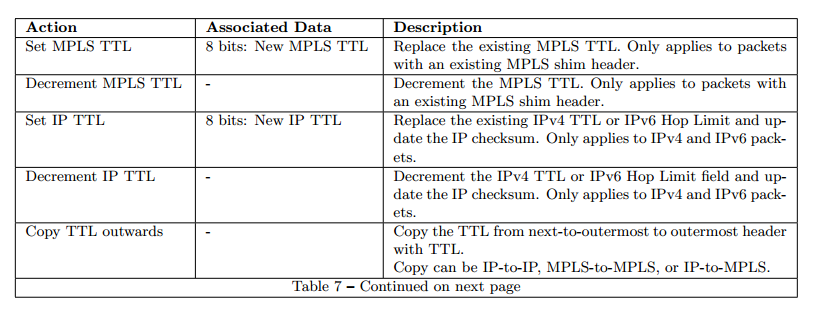
交换机没有必要支持所有的动作，只需要优先支持“必选动作”，Controller会在需要时请求交换机支持动作的列表（理应在握手时获得）。

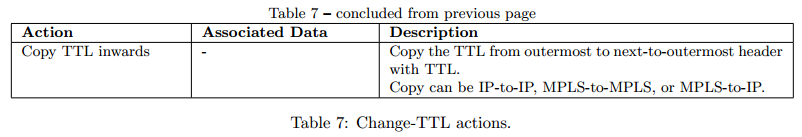
|  |
| --- |
| * 必选动作: Output. 将报文发给指定的OpenFlow端口(4.1)。OpenFlow交换机必须支持转发给物理口、交换机定义的逻辑口，以及必须的保留口(4.5)。 * 可选动作: Set-Queue. 设置报文的队列ID。每当一个报文被使用output命令转发到某个OpenFlow端口上，队列ID决定了这个报文要使用该端口上的哪一个队列来进行调度和转发。转发行为由队列的配置来决定，并且这个队列是用于基础的流量控制 (QoS) 支持的 (see section 7.2.2). * 必选动作: Drop. 没有显示的动作表示丢弃报文。取而代之的判断方法是：假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标签如Table 6所示。为了拥有与现有网络良好的融合能力，我们建议交换机支持push/pop vlan标签。 |

详细描述（每个域的默认值见5.12.1）：



|  |
| --- |
| * 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.) |

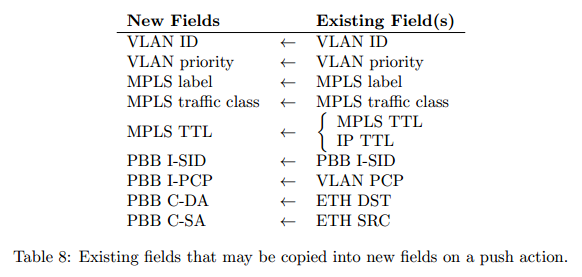




OpenFlow交换机会检验报文的TTL或MPLS-TTL字段，如果非法则会拒绝报文。但注意一点：不是每个报文都需要检验TTL，但当一个action是减小TTL的值时，交换机必须在TTL值减小后检验该报文。交换机发asynchronous消息给控制器的配置可被修改为：将非法TTL的报文packet-in给控制器。

### Push时各个域的默认值

当交换机执行一个push动作时，交换机必须将报文包头的值（在Table 8指定的所有域的值）复制到新的包头。原包头不包含的域的值将会在新的域中被设置为0。那些无法通过OpenFlow set-field动作修改的域应该被初始化为适当的值。



push一个头之后，报文可以用set动作来修改。

注意：这里没有set vlan的动作。

# 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。

## 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，以管理和视察交换机的状态
* **asynchronous**：switch发给controller的消息，用以告知控制器网络事件和交换机状态的变更
* **symmetric**：双方都可发送，并不需要回复。

每个消息都包含若干子消息类型。具体如下。

## Controller-to-Switch

这种类型的消息由控制器发给交换机，交换机并不一定需要回复。

**Features**：控制器通过发送一个features request给交换机来请求获得交换机的信息；交换机必须回复features reply来告知控制器它的信息。这个过程经常发生在OpenFLow通道（channel）建立的时候

**Configuration**：控制器能设置并且查询交换机的配置参数。交换机只需回复控制器的查询。

**Modify-State**：控制器发送Modify-State消息来管理交换机的状态。它们的主要目的是添加、删除和修改流表项或group entry，或者设置交换机的端口属性。

**Read-State**：控制器使用Read-State消息来获取交换机的各个信息，如当前配置、统计信息或容量等等。

**Packet-out**：控制器使用这种消息来发送报文给交换机的一个指定端口，或者将Packet-in接收的报文转发出去。Packet-out消息必须包含了一个完整的报文或者一个buffer ID（对应交换机存储的报文）。该消息也必须包含一个要实施的动作列表；如果该动作列表为空，则交换机丢弃该数据包。

Barrier：控制器使用Barrier request/reply消息来保证信息的独立性，或者用于接收操作完成的通知。

**Role-Request**：控制器使用这种消息来设置或者查询它的OpenFlow通道（channel）。这种消息经常在交换机连接到多个控制器的情况下使用（见6.3.4）。

**Asynchronous-Configuration**：控制器使用这种消息来设置一个额外过滤器，该过滤器指定了控制器想要接收的asynchronous消息类型。控制器也可以使用这种消息来查询该过滤器。这种消息经常在交换机连接到多个控制器的情况下被使用，并且一般在OpenFlow通道建立的过程中发生。

### Asynchronous

Asynchronous消息由交换机发送给控制器，并且控制器不必回复该消息。当有报文需要packet-in时、或者交换机状态变更时，或者发送错误（error）时，交换机发送这种消息。这种消息主要有4种类型：

**Packet-in**：将报文发送给控制器。对于所有的匹配到flow entry或者table-miss flow entry的报文，交换机将该报文转发给控制器（见5.12）。其他的处理，如检测TTL，也可能会发送packet-in事件给控制器。

Packet-in事件可被配置，使得交换机可以缓存报文。对于flow entry或者group bucket的output action产生的packet-in事件，它可被output action配置（见7.2.5）；而对于其他的packet-in事件，它可在交换机配置中配置（见7.3.2）。如果packet-in事件配置为缓存报文，同时交换机有足够的内存来缓存报文，那么packet-in事件将只包含报文包头的一部分，和一个buffer ID。当控制器想要转发该报文时，Buffer ID可被指定对应的报文。那些不支持缓存的交换机，应被配置不对packet-in事件缓存报文。当交换机内存占满从而无法缓存报文时，它必须将完整的报文转发给控制器。控制器经常使用**Packet-out**消息来使用缓存的报文。如果在一定时间内缓存的报文没有得到处理，那么它将会自动的消失。

在缓存的情况下，packet-in包含的原报文的字节数可被配置。默认情况下交换机缓存128个字节。对于flow entry或者group bucket的output action产生的packet-in事件，它可被output action配置（见7.2.5）；而对于其他的packet-in事件，它可在交换机配置中配置（见7.3.2）。

**Flow-Removed**：告知控制器流表中的某一个flow entry被移除了。只有flow entry有OFPFF\_SEND\_FLOW\_REM标志，交换机才可能会发送Flow-Removed消息。当控制器想要主动删除一个流表项时、或者flow entry过期时，产生了这个标志。

**Port-Status**：告知控制器端口的信息。当端口被配置时，或者端口的状态发生改变时，交换机需要发送这种消息给控制器。这些事件包含了端口配置的改变（如用户直接关掉端口）或者端口状态的改变（如链路失效down）。

**Error**：当错误发生时，交换机发送这种消息告知控制器。

### Symmetric

Symmetric消息可被交换机或者控制器发送，并且都不期望能够得到对方回复。

**Hello**：当控制器与交换机的连接建立时，它们交换Hello消息。

**Echo**：Echo request/reply消息可被交换机或者控制器发送，并且对方必须回复一个echo reply消息。它们经常在控制器或者交换机想要确认链路的存活时使用，它们也可以用来测量延迟及带宽。

**Experimenter**：Experimenter消息为OpenFlow交换机提供了一个标准的方式，它使得交换机可以在OpenFlow消息的类型空间（type space）中添加额外的功能。它用来测试并改善未来的OpenFlow版本。

## Message Handling

OpenFlow协议提供了可靠的消息传输和处理，但它不会自动地提供ack，也不会保证消息按序处理。本章描述了在主连接和使用可靠传输的辅助连接的OpenFlow消息处理行为。注意这些在使用不可靠传输的辅助连接中并不适用（见6.3.5）

**Message Delivery**：除非在OpenFlow通道完全失效，消息在一般情况下都能得到保证传输。在OpenFLow通道失效的情况下，控制器没办法获得交换机状态的任何信息（例如，交换机可能一进入了“fail standalone”模式）

**Message Processing**：交换机必须处理控制器发来的每一个消息，并在必要时应回复控制器。如果一个交换机无法有效处理控制器发来的消息，那么它必须发回一个error消息给控制器。对于packet-out的消息，交换机并不保证包含在该消息中的报文能离开交换机。由于交换机拥塞、QoS服务、或报文被发送到一个关闭的或者不合法的端口，该报文可能被交换机（悄悄地）丢弃。

此外，当OpenFlow的状态发生改变时（如flow-removed，port-status或packet-in消息），交换机必须将所有的asynchronous消息发送给控制器，这样控制器才能与交换机的状态保持实时一致。这些消息可能会由于Asynchronous配置而被过滤掉（见6.1.1）。而且，有时这些会触发OpenFlow状态变更的事件可能会在触发这些事件之前就已被丢掉。比如，一些本应发到控制器的报文可能因为交换机中的拥塞或者QoS服务而被丢掉，从而不会产生packet-in消息。那些动作为“发送给控制器”的报文、和那些table-miss而发到控制器的报文都可能因此而被丢弃。我们推荐使用QoS动作或者限速来对发送给控制器的报文进行限制，从而保护控制器不受到DoS攻击，这超出了本文所讨论的范围。

控制器可以不处理它们接收到的信息，但必须回复echo消息从而保证交换机不会主动的关掉连接。

Message Ordering：使用barrier消息可以保证序列。如果不使用barrier消息，交换机可能对消息任意排序来优化性能；因此，控制器不能依赖一个特定的处理顺序。特别地，交换机将flow entry添加到流表中的顺序可能与它接受到的flow mod消息的顺序不一样。而barrier消息可以保证消息按序处理，并且只有当所有先前的消息都被处理后，barrier才能被处理。下列是详细细节：

1.在barrier消息之前的消息都必须在处理barrier之前得到处理，这包含发送回复或者错误信息；

2.然后交换机才能处理barrier信息并且发送一个barrier回复；

3.最后交换机才处理barrier之后的消息。

如果控制器发出的两个消息互相依赖，那么他们必须通过一个barrier消息来分开彼此。这种情况有：1）添加一个group mod和一个flow mod，其中flow mod指定了group mod；2）添加一个port mod和一个packet-out，其中packet-out将消息转发到该port；3）添加一个flow mod和一个packet-out，其中packet-out指定了OFPP\_TABLE。

## OpenFlow Channel Connections

OpenFlow通道被用于在OpenFlow交换机与OpenFlow控制器之间交换信息。一个典型的OpenFlow控制器管理多个OpenFlow通道，分别连接到各个OpenFlow交换机。一个OpenFlow交换机可能只有一个OpenFlow通道连接到一个控制器，也可能有多个通道（为了可靠性）连接到不同的控制器（见6.3.4）。

一个典型的OpenFlow控制器在一个或者多个网络上远程地控制器一个OpenFlow交换机。OpenFlow通道使用的网络的specification不在本文讨论的范围之内。它可能是一个分开的网络，或者，OpenFlow通道也可能使用OpenFlow交换机管理的网络（带内（in-band）控制连接）。它只需要提供TCP/IP连接即可。

一般情况下，控制器与交换机之间使用TLS或者纯TCP（见6.3.3）的OpenFlow通道。或者，OpenFlow通道也可能包含了多个并行的控制器连接（见6.3.5）。OpenFLow交换机必须主动创建一个OpenFlow通道来连接到OpenFlow控制器上（见6.3.1）。在某些情况下，交换机可让控制器主动连接到交换机上，但这种情况下，必须使用安全连接（见6.3.3）来防止攻击者连接。

### Connection Setup

### Connection Interruption

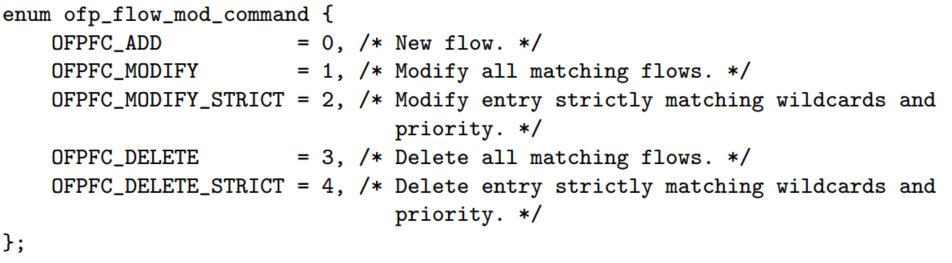
### Encryption

### Multiple Controllers

### Auxiliary Connections

## Flow Table Modification Messages（更改流表信息）

流表更改信息有以下几种类型：



对于有OFPFF\_CHECK\_OVERLAP标志的**add请求**（OFPFC\_ADD），交换机必须首先检查在被请求修改的流表中，是否有冲突的flow entry。两个flow entry冲突是指，一个报文可能同时match这两个flow entry，并且这两个flow entry有相同的优先级。如果**add请求**添加的flow entry与当前某个flow entry冲突，交换机必须拒绝该**add请求**，并回复一个ofp\_error\_msg，类型为OFPET\_FLOW\_MOD\_FAILED，编码为OFPFMFC\_OVERLAP。

对于没有冲突、和没有检测冲突的add请求， 交换机必须将flow entry添加到对应的流表中。如果流表中早已存在一个匹配域和优先级都相同的flow entry，那么该flow entry必须被删除，然后交换机将新的flow entry添加到交换机中。如果该add请求设置了OFPFF\_RESET\_COUNTS标志，那么该flow entry的计数器应该被清零，否则，新替换的flow entry的计数器仍然继承了旧的flow entry的计数器。对于这种方式被删除掉的flow entry，交换机并不会发送一个flow-removed消息给控制器；如果控制器想要获得该种flow-removed消息，那么它必须先发送一个delete请求删除旧的flow entry，然后再添加新的flow entry。

对于**modify请求**（OFPFC\_MODIFY或者OFPFC\_MODIFY\_STRICT），如果流表中存在匹配的项，则该流表项的instruction域被更新为与该请求的instruction一致，而其他的域，如cookie，idle\_timeout，hard\_timeout，flags，counters和duration都不变。如果该请求设置有OFPP\_RESET\_COUNTS标志，那么该流表项的计数器必须清零。如果流表中没有流表项与该请求匹配，那么交换机忽略该请求，并不会发出error消息，也不会对流表做任何修改。

对于**delete请求**（OFPFC\_DELETE或者OFPFC\_DELETE\_STRICT），如果流表中存在匹配的流表项，并且该流表项设置了OFPFF\_SEND\_FLOW\_REM标志，那么交换机必须发送一个flow removed消息。如果流表中没有流表项与该请求匹配，那么交换机忽略该请求，并不会发出error消息，也不会对流表做任何修改。

**Modify**和**delete**命令有两个类型：non-strict类型（OFPFC\_MODIFY和OFPFC\_DELETE），strict类型（OFPFC\_MODIFY\_STRICT和OFPFC\_DELETE\_STRICT）。在strict类型中，流表项的掩码（mask）和优先级必须严格匹配，并且最后只会有一个流表项被修改或者移除。例如，如果控制器发出一个没有包含匹配域的delete消息，那么OFPFC\_DELETE会删除所有的流表项，而OFPFC\_DELETE\_STRICT只会删除该优先级的一个流表项。

对于non-strict **modify**和**delete**命令，所有与flow\_mod匹配的流表项都会被修改，或者移除。在non-strict版本中，flow\_mod没有指定的匹配域都是通配的。例如，如果一个OFPFC\_DELETE想要删除所有目标端口为80的流表项，那么所有目标端口为80的流表项都会被删除，无论它其他的匹配域是否存在。同样的，strict和non-strict规则同样适用于flow stats请求。

在某些情况下，Delete命令可指定destination group或者output port来指定流表项。如果out\_port域的值不是OFPP\_ANY，那么该值指定了匹配的流表项必须包含一个output到该端口（即out\_port域的值）的动作。必须注意的是，该域只限定了流表项，并不匹配那些同样匹配到该域的group。同样的，如果out\_group域的值不是OFPG\_ANY，它约束了对应的group。注意，OFPFC\_ADD，OFPFC\_MODIFY和OFPFC\_MODIFY\_STRICT消息会直接忽略这两个域。

当cookie\_mask的值非0时，modify和delete命令也可以通过cookie值来指定。此时，flow mod中cookie\_mask指定的位的cookie值必须与flow entry的cookie值相等。即，(flow entry:cookie & flow mod:cookie mask) == (flow mod:cookie & flow mod:cookie mask)。

Delete命令也可以使用OFPTT\_ALL值指定所有的流表（不止一个），来将所有流表中所有匹配到的flow entry删除。

如果一个更改流表的命令指定了一个非法的table-id，那么交换机必须发送一个ofp\_error\_msg信息（类型：OFPET\_FLOW\_MOD\_FAILED，代码：OFPFMFC\_BAD\_\_TABLE\_ID）。如果一个add或modify request命令指定了OFPTT\_ALL，则交换机必须发送相同的错误信息。

如果交换机因为流表已满而无法在指定的流表中添加流表项，则交换机必须发送一个ofp\_error\_msg信息（类型：OFPET\_FLOW\_MOD\_FAILED，代码：OFPFMFC\_TABLE\_FULL）。

如果交换机不知道flow mod中的instructions（unknown），则该交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_INSTRUCTION，代码：OFPBIC\_UNKNOWN\_INST）。如果交换机不支持（unsupport）flow mod的instructions，则交换机也必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_INSTRUCTION，代码：OFPBIC\_UNSUP\_INST）。

如果instructions指定了一个Goto-Table但该table-id是非法的，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_INSTRUCTION，代码：OFPBIC\_BAD\_TABLE\_ID）。

如果instructions包含了一个Write-Metadata然而交换机不支持该metadata的值或者掩码（mask），则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_INSTRUCTION，代码：OFPBIC\_UNSUP\_METADATA或OFPBIC\_UNSUP\_METADATA\_MASK）。

如果match指定了一个交换机不支持的域，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_MATCH，代码：OFPBMC\_BAD\_FIELD）。如果match对一个相同的域指定了2次以上，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_MATCH，代码：OFPBMC\_DUP\_FIELD）。如果一个match指定了某个域但没有指定该域的先决条件的域（如指定了一个IPv4地址，但没有指定EtherType为0x0800），则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_MATCH，代码：OFPBMC\_BAD\_PREREQ）。

如果match对数据链路或者网络地址指定了一个交换机不支持的位掩码，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_MATCH，代码：OFPBMC\_BAD\_DL\_ADDR\_MASK或OFPBMC\_BAD\_NW\_ADDR\_MASK）。如果match对数据链路和网络地址都指定了一个交换机不支持的位掩码，则返回错误代码应为OFPBMC\_BAD\_DL\_ADDR\_MASK。如果match对其他的域指定了一个交换机不支持的位掩码，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_MATCH，代码：OFPBMC\_BAD\_VALUE）。

如果某个action指定了一个永远不可能为合法的port，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：OFPBAC\_BAD\_OUT\_PORT）。如果该port可能在未来某个时刻是合法的（例如，新添加了一块网卡（端口）到交换机），则交换机有两个选择：1）丢掉所有发往该端口的报文并不发出任何消息；2）立即返回一个OFPBAC\_BAD\_OUT\_PORT错误并且拒绝添加该flow mod。

如果flow mod中的action指定了一个group而交换机当前还未定义该group，或指定了一个保留组（reserved group）（如OFPG\_ALL），则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：OFPBMC\_BAD\_OUR\_GROUP）。

如果action包含一个非法的值（如Set VLAN ID指定了一个大于4095的值，或者Push指定了一个非法的Ethertype），交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：OFPBAC\_BAD\_OUT\_PORT）。

如果action与match不一致时（如action为pop VLAN而match没有指定任何的VLAN，或者设置一个IPv4地址而match没有指定Ethertype），交换机可以拒绝该flow mod并立即返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：OFPBAC\_MATCH\_INCONSISTENT）。我们仍未定义不一致的action和match会造成什么影响。我们强烈建议，控制器不将table entry组合在一起，从而避免产生不一致的action。

如果一个action list包含了一系列的动作但交换机不支持该次序，则交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：OFPBAC\_UNSUPPORTED\_ORDER）。

除了上述所述的错误意外，如果交换机在处理flow mod的时候发生了其他的错误，则交换机可以返回一个ofp\_error\_msg信息（类型：OFPET\_FLOW\_MOD\_FAILED，代码：OFPFMC\_UNKNOWN）。

## Group Table Modification Messages

组表更改信息（Group Table Modification Messages）有下列几种类型：

/\* Group commands \*/

enum ofp\_group\_mod\_command {

OFPGC\_ADD = 0, /\* 新组。 \*/

OFPGC\_MODIFY = 1, /\* 更改所有匹配到的组。 \*/

OFPGC\_DELETE = 2, /\* 删除所有匹配到的组 \*/

};

一个组可以不包含桶（bucket），也可以包含多个。没有桶的组将不会修改报文对应的动作集（action set）。一个组也可以包含转发到其他组的桶（如果交换机支持的话）。

每一个桶的动作集都应遵循与flow mod相同的规则（见6.4），并且还应遵循额外的规则。如果某个桶的某个动作是非法的或者是不支持的，交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_BAD\_ACTION，代码：与该错误对应）（见6.4）。

对于add请求（OFPGC\_ADD），如果组表中早已存在一个相同标识的组表项（group entry），那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_GROUP\_EXISTS）。

对于modify请求（OFPGC\_MODIFY），如果组表中早已存在一个相同标识的组表项，那么该组表项（包含它的类型及动作桶）都应该被移除，然后交换机添加该新组表项。如果组表中并不存在一个相同标识的组表项，那么交换机应该拒绝该group mod并且返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_UNKNOWN\_GROUP）。

如果某个组的类型是非法的（即，包含了诸如weight等该组类型未定义的域），那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_INVALID\_GROUP）。

如果交换机不支持组的权重负载均衡（即，桶的weight值不等于1），那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_WEIGHT\_UNSUPPORTED）。

如果交换机因为内存已满而无法添加组表项时，那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_OUT\_OF\_GROUPS）。

如果交换机因为动作桶的数量限制（可能是硬件原因或其他原因）而无法添加组表项时，那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_OUT\_OF\_GROUPS）。

如果交换机因为不支持生存配置（liveliness configuration）而无法添加组表项时，那么交换机必须拒绝添加该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_WATCH\_UNSUPPORTED）。它应说明不支持该生存配置的watch\_port或者watch\_group。

对于delete请求（OFPGC\_DELETE），如果组表中并不存在一个相同标识的组表项，那么交换机不会产生error信息，并且不更改任何组表。否则，与之匹配的组表项将会删除，并且所有在Group动作中包含了该组的流表项都会被删除。Delete请求不必指定组的类别。（Delete also differs from an add or modify with no buckets specified in that future attempts to add the group identifier will not result in a group exists error.）如果用户想要删除一个组，但同时不想要删除相应的流表项，那么可发送一个没有指定任何动作的modify请求。

用户如果想要只用一个消息来删除所有的组，那么应指定组的值为OFPG\_ALL。

组也可以链接在一起（如果交换机支持的话）。此时某个组将流转发到另外一个组，或者更加复杂。比如，一个快速的重路由组可能有两个桶，这两个桶各自转发到其他的组。如果一个交换机不支持这种操作（组链接），它必须返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_CHAINING\_UNSUPPPORTED）。

当组链接在一起时，交换机可支持检验组的链接，确保无环产生：如果产生了环，那么交换机必须拒绝该group mod，并且返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_LOOP）。如果交换机不支持检验，那么转发的行为是未定义的（？）。

一个交换机可支持检验，转发的目标组确实存在（没有被删除）：如果一个交换机无法删除一个组（因为它被其他的组所指定），那么它应该拒绝删除该组表项，并返回一个ofp\_error\_msg信息（类型：OFPET\_GROUP\_MOD\_FAILED，代码：OFPGMFC\_CHAINED\_GROUP）。如果交换机不支持该检验，那么转发的行为是未定义的（？）。

如果交换机支持快速故障重恢复（Fast failover）组，那么它应检测存活（liveness monitoring），来决定执行哪一个桶。其他类型的组并不需要检测存活（当然交换机自己可以检测存活）。如果一个交换机无法检测组中的桶的存活，那么它应该拒绝该group mod并且返回一个错误。决定存活的规则包含：

* 如果一个端口的标志（flag）是OFPPS\_LIVE，那么该端口是存活的。端口的存活不由交换机的OpenFlow部分来决定，如生成树或KeepAlive等机制。它的存活与否不在本文的讨论范围。如果交换机的一个端口存活机制将该端口设为死的，或者如果端口的配置位OFPPC\_PORT\_DOWN表明了该端口是down的，或者如果端口的状态位OFPPS\_LINK\_DOWN表明了该链路是down的，则该端口不应该被认为是存活的（此时标志OFPPS\_LIVE不应该被设置）。
* 如果watch\_port不是OFPP\_ANY或者监视的端口是存活的，或者watch\_group不是OFPG\_ANY并且监视的组是存活的，那么痛应该被认为是存活的。
* 如果某个组有至少一个桶是存活的，那么这个组应该被认为是存活的。

控制器可以通过监视各个端口的状态来推断组的存活的状态。

## Meter Modification Messages

Meter更改消息有下列几种类型

/\* Meter commands \*/

enum ofp\_meter\_mod\_command {

OFPMC\_ADD, /\* New meter. \*/

OFPMC\_MODIFY, /\* Modify specified meter. \*/

OFPMC\_DELETE, /\* Delete specified meter. \*/

};

对于添加（add）请求（OFPMC\_ADD），如果交换机已存在一个同一标识的meter entry，那么交换机必须拒绝添加该meter entry，并且返回一个ofp\_error\_msg信息（类型：OFPET\_METER\_MOD\_FAILED，代码：OFPMMFC\_METER\_EXISTS）。

对于更改（modify）请求（OFPMC\_MODIFY），如果交换机已存在一个同一标识的meter entry，那么该entry，包含它的带宽，则会被移除，同时交换机将添加新的meter entry。如果交换机不存在一个同一标识的meter entry，那么交换机必须拒绝该meter mod，并且返回一个ofp\_error\_msg信息（类型：OFPET\_METER\_MOD\_FAILED，代码：OFPMMFC\_UNKNOWN\_METER）。

如果交换机因为内存已满而无法添加meter entry，那么交换机必须返回一个ofp\_error\_msg信息（类型：OFPET\_METER\_MOD\_FAILED，代码：OFPMMFC\_OUT\_OF\_METERS）。

如果交换机因为带宽的数量限制（硬件上的或者其他的原因）而无法添加meter entry，那么它应该拒绝添加该meter entry并返回一个ofp\_error\_msg信息（类型：OFPET\_METER\_MOD\_FAILED，代码：OFPMMFC\_OUT\_OF\_BANDS）。

对于删除（delete）请求（OFPMC\_DELETE），如果交换机不存在一个同一标识的meter entry，则不发生任何meter更改，也不产生任何错误信息。否则，对应的meter将会被删除，并且所有在instruction中包含了应该meter的流表也将会被删除。Meter delete请求只需指定identifier，其他的域（如带宽band）可以被忽略。

如果想要只用一个消息来删除所有的meter，控制器可指定meter的值为OFPM\_ALL。此时，虚拟（virtual）meter将不会被删除

# OpenFlow协议（Appendix）

本文的核心在于OpenFlow协议消息结构的设置。

下文定义的结构源于文件inclue/openflow/openflow.h，该文件是OpenFlow specification的一部分。所有的结构都是8字节对齐的（assertion statement检测对齐）。所有的OpenFlow消息都是以大端模式发送的。

## OpenFlow包头（OpenFlow Header）

/\* Header on all OpenFlow packets. \*/

struct ofp\_header {

uint8\_t version; /\* OFP\_VERSION. \*/

uint8\_t type; /\* One of the OFPT\_ constants. \*/

uint16\_t length; /\* Length including this ofp\_header. \*/

uint32\_t xid; /\* Transaction id associated with this packet.

Replies use the same id as was in the request

to facilitate pairing. \*/

};

OFP\_ASSERT(sizeof(struct ofp\_header) == 8);

Version指定了OpenFlow的版本。在OpenFlow的早期阶段中，最重要的位用于指示该版本是否是一个实验版本。低位表示了协议的修订号。本文所描述的版本是1.3.2，其ofp\_version为0x04。

Length表示了该消息的总长度，所以无需额外的帧来区分相邻的帧。Type有以下几种类型：

enum ofp\_type {

/\* Immutable messages. \*/

OFPT\_HELLO = 0, /\* Symmetric message \*/

OFPT\_ERROR = 1, /\* Symmetric message \*/

OFPT\_ECHO\_REQUEST = 2, /\* Symmetric message \*/

OFPT\_ECHO\_REPLY = 3, /\* Symmetric message \*/

OFPT\_EXPERIMENTER = 4, /\* Symmetric message \*/

/\* Switch configuration messages. \*/

OFPT\_FEATURES\_REQUEST = 5, /\* Controller/switch message \*/

OFPT\_FEATURES\_REPLY = 6, /\* Controller/switch message \*/

OFPT\_GET\_CONFIG\_REQUEST = 7, /\* Controller/switch message \*/

OFPT\_GET\_CONFIG\_REPLY = 8, /\* Controller/switch message \*/

OFPT\_SET\_CONFIG = 9, /\* Controller/switch message \*/

/\* Asynchronous messages. \*/

OFPT\_PACKET\_IN = 10, /\* Async message \*/

OFPT\_FLOW\_REMOVED = 11, /\* Async message \*/

OFPT\_PORT\_STATUS = 12, /\* Async message \*/

/\* Controller command messages. \*/

OFPT\_PACKET\_OUT = 13, /\* Controller/switch message \*/

OFPT\_FLOW\_MOD = 14, /\* Controller/switch message \*/

OFPT\_GROUP\_MOD = 15, /\* Controller/switch message \*/

OFPT\_PORT\_MOD = 16, /\* Controller/switch message \*/

OFPT\_TABLE\_MOD = 17, /\* Controller/switch message \*/

/\* Multipart messages. \*/

OFPT\_MULTIPART\_REQUEST = 18, /\* Controller/switch message \*/

OFPT\_MULTIPART\_REPLY = 19, /\* Controller/switch message \*/

/\* Barrier messages. \*/

OFPT\_BARRIER\_REQUEST = 20, /\* Controller/switch message \*/

OFPT\_BARRIER\_REPLY = 21, /\* Controller/switch message \*/

/\* Queue Configuration messages. \*/

OFPT\_QUEUE\_GET\_CONFIG\_REQUEST = 22, /\* Controller/switch message \*/

OFPT\_QUEUE\_GET\_CONFIG\_REPLY = 23, /\* Controller/switch message \*/

/\* Controller role change request messages. \*/

OFPT\_ROLE\_REQUEST = 24, /\* Controller/switch message \*/

OFPT\_ROLE\_REPLY = 25, /\* Controller/switch message \*/

/\* Asynchronous message configuration. \*/

OFPT\_GET\_ASYNC\_REQUEST = 26, /\* Controller/switch message \*/

OFPT\_GET\_ASYNC\_REPLY = 27, /\* Controller/switch message \*/

OFPT\_SET\_ASYNC = 28, /\* Controller/switch message \*/

/\* Meters and rate limiters configuration messages. \*/

OFPT\_METER\_MOD = 29, /\* Controller/switch message \*/

};

### Padding

## 一般结构

## Controller-to-Switch消息

### 握手消息

### 交换机配置消息

### 流表配置消息

### 更改状态消息

#### 更改流表项消息

#### 更改组表项消息

#### 更改端口消息

#### 更改Meter消息

OFPT\_METER\_MOD消息用于更改meter。

/\* Meter configuration. OFPT\_METER\_MOD. \*/

struct ofp\_meter\_mod {

struct ofp\_header header;

uint16\_t command; /\* One of OFPMC\_\*. \*/

uint16\_t flags; /\* Bitmap of OFPMF\_\* flags. \*/

uint32\_t meter\_id; /\* Meter instance. \*/

struct ofp\_meter\_band\_header bands[0]; /\* The band list length is

inferred from the length field

in the header. \*/

};

OFP\_ASSERT(sizeof(struct ofp\_meter\_mod) == 16);

Meter\_id唯一地标识了交换机中对应的meter。Meter的编号从meter\_id=1开始增加，直到交换机能够支持的最大条目。OpenFlow协议也定义了一些额外的虚拟meter，这些虚拟meter不能用于与流进行交互：

/\* Meter numbering. Flow meters can use any number up to OFPM\_MAX. \*/

enum ofp\_meter {

/\* Last usable meter. \*/

OFPM\_MAX = 0xffff0000,

/\* Virtual meters. \*/

OFPM\_SLOWPATH = 0xfffffffd, /\* Meter for slow datapath. \*/

OFPM\_CONTROLLER = 0xfffffffe, /\* Meter for controller connection. \*/

OFPM\_ALL = 0xffffffff, /\* Represents all meters for stat requests

commands. \*/

};

尽管OpenFlow协议包含了虚拟meter，但是我们建议用户使用常规的方法，如基于每个流的meter（见5.7章）或者优先级队列（见7.2.2章）。

* OFPM\_CONTROLLER：虚拟meter控制所有通过packet-in消息发往控制器的报文（使用CONTROLLER保留端口发往控制器的报文或者其他过程（见6.1.2章））。能够被用于限制发往控制器的报文流量。
* OFPM\_SLOWPATH：虚拟meter控制所有由交换机的慢通道（slow datapath）处理的报文。有许多交换机拥有一个快速通道和一个慢速通道（fast and slow datapath）。比如，一个硬件交换机可能有一个慢速软件通道，或者一个软件交换机可能有一个较慢的用户控件通道。

Command必须为以下值：

/\* Meter commands \*/

enum ofp\_meter\_mod\_command {

OFPMC\_ADD, /\* New meter. \*/

OFPMC\_MODIFY, /\* Modify specified meter. \*/

OFPMC\_DELETE, /\* Delete specified meter. \*/

};

Flags包含了以下一个或多个标志（flag）：

/\* Meter configuration flags \*/

enum ofp\_meter\_flags {

OFPMF\_KBPS = 1 << 0, /\* Rate value in kb/s (kilo-bit per second). \*/

OFPMF\_PKTPS = 1 << 1, /\* Rate value in packet/sec. \*/

OFPMF\_BURST = 1 << 2, /\* Do burst size. \*/

OFPMF\_STATS = 1 << 3, /\* Collect statistics. \*/

};

Band是一个速率带宽列表。它可以包含任何数量的band，并且每一个band的类型都可以是重复的（只要有意义的话）。当当前的报文速率超过了多个band的速率时，交换机只会使用一个band，该band为那个最高的配置速率。

所有band的包头都如下所示：

/\* Common header for all meter bands \*/

struct ofp\_meter\_band\_header {

uint16\_t type; /\* One of OFPMBT\_\*. \*/

uint16\_t len; /\* Length in bytes of this band. \*/

uint32\_t rate; /\* Rate for this band. \*/

uint32\_t burst\_size; /\* Size of bursts. \*/

};

OFP\_ASSERT(sizeof(struct ofp\_meter\_band\_header) == 12);

其中rate配置了band的速率（见5.7.1章）。如果flags域中指定了OFPMF\_PKTPS的标志，则rate的单位是pakcet/s；否则，rate的单位为kbit/s。

仅当flags中指定了OFPMF\_BURST的标志，busrt\_size才会被使用。它定义了meter band的细粒度：当突发的报文或者字节的长度大于burst\_size的值时，meter rate就会被执行。如果flags域中指定了OFPMF\_PKTPS的标志，则burst\_size的单位为kbit；否则，单位为packet。

Type的值必须如以下所示：

/\* Meter band types \*/

enum ofp\_meter\_band\_type {

OFPMBT\_DROP = 1, /\* Drop packet. \*/

OFPMBT\_DSCP\_REMARK = 2, /\* Remark DSCP in the IP header. \*/

OFPMBT\_EXPERIMENTER = 0xFFFF /\* Experimenter meter band. \*/

};

OpenFlow交换机可能不支持所有的band类型，并且也不一定支持在所有meter中使用它能支持的band类型，即，可能有一些meter是比较特殊的。

类型为OFPMBT\_DROP的band定义了一个简单的限流器（rate limiter），它将所有超过band rate的值的报文全部丢弃。它的结构如下所示：

/\* OFPMBT\_DROP band - drop packets \*/

struct ofp\_meter\_band\_drop {

uint16\_t type; /\* OFPMBT\_DROP. \*/

uint16\_t len; /\* Length in bytes of this band. \*/

uint32\_t rate; /\* Rate for dropping packets. \*/

uint32\_t burst\_size; /\* Size of bursts. \*/

uint8\_t pad[4];

};

OFP\_ASSERT(sizeof(struct ofp\_meter\_band\_drop) == 16);

类型为OFPMBT\_DSCP\_REMARK的band定义了一个简单的DiffServ策略。所有超过band rate值的报文的IP包头中的DSCP域值中的丢弃优先级都会被重新标记，它的结构如下所示：

/\* OFPMBT\_DSCP\_REMARK band - Remark DSCP in the IP header \*/

struct ofp\_meter\_band\_dscp\_remark {

uint16\_t type; /\* OFPMBT\_DSCP\_REMARK. \*/

uint16\_t len; /\* Length in bytes of this band. \*/

uint32\_t rate; /\* Rate for remarking packets. \*/

uint32\_t burst\_size; /\* Size of bursts. \*/

uint8\_t prec\_level; /\* Number of drop precedence level to add. \*/

uint8\_t pad[3];

};

OFP\_ASSERT(sizeof(struct ofp\_meter\_band\_dscp\_remark) == 16);

其中，prec\_level字段指示，如果当前速率超过该band，则应该增加的报文丢弃优先级的大小。该band通过prec\_level指定了丢弃优先级的增加大小，而不是原来的DSCP值；增加丢弃优先级后的DSCP值必须是合法的，并且没有指定丢弃优先级的DSCP值将不会被修改。

类型为T\_EXPERIMENTER的band用于实验。它的结构如下所示：

/\* OFPMBT\_EXPERIMENTER band - Experimenter type.

\* The rest of the band is experimenter-defined. \*/

struct ofp\_meter\_band\_experimenter {

uint16\_t type; /\* One of OFPMBT\_\*. \*/

uint16\_t len; /\* Length in bytes of this band. \*/

uint32\_t rate; /\* Rate for this band. \*/

uint32\_t burst\_size; /\* Size of bursts. \*/

uint32\_t experimenter; /\* Experimenter ID which takes the same

form as in struct

ofp\_experimenter\_header. \*/

};

OFP\_ASSERT(sizeof(struct ofp\_meter\_band\_experimenter) == 16);

## Asynchronous消息

## Symmetric消息