

# 软件定义网络：OpenFlow 和 VxLAN

作者： Vishal Shukla

# 目录

目录 .....	1
致谢 .....	5
前言 .....	6
第一章 软件定义网络.....	7
1.1 软件定义网络.....	7
控制平面和数据平面的分离.....	8
SDN 控制器.....	8
网络编程的可扩展性.....	9
SDN 的逻辑层.....	9
南向 API .....	9
北向 API .....	10
可行性的技术.....	10
1.2 什么是 OpenFlow? .....	10
OpenFlow 组件 .....	11
1.3 为什么发明 OpenFlow? .....	12
第二章 OpenFlow 概述 .....	14
2.1 OpenFlow 协议概述 .....	14
2.2 OpenFlow 组件概述 .....	16
2.2.1 OpenFlow 控制器(Controller) .....	16
2.2.2 OpenFlow 交换机 .....	17
2.2.3 流表.....	17
动作分类.....	21
第三章 OpenFlow 内部事件 .....	23

3.1 连接建立事件 .....	23
3.2 Packet-In 事件 .....	25
3.3 Packet-Out 事件 .....	27
3.4 端口状态 (Port Status) 消息事件 .....	28
3.5 设置配置 (Set Configuration) 事件 .....	31
3.6 获取配置 (Get Config) 请求与答复事件 .....	32
3.7 修改流 (Flow-Modification) 事件 .....	33
3.8 流移除 (Flow-Removed) 事件 .....	36
3.9 端口修改 (Port-Modify) 事件 .....	38
3.10 统计(Stats)请求和响应事件 .....	40
3.11 Barrier 请求和响应事件 .....	42
3.12 队列获取配置 (Queue Get Configuration) 请求和响应事件 .....	43
3.13 错误事件 .....	44
第四章 OpenFlow 案例研究 .....	46
4.1 拓扑结构的发现 .....	46
4.2 最短路径计算 .....	48
4.3 以 Ping 为例的流表操作 .....	50
4.4 以故障切换为例的流表修改 .....	56
第五章 OpenFlow 数据包详细说明 .....	60
5.1 OpenFlow 数据包概要 .....	60
控制器-至-交换机 .....	60
Asynchronous .....	60
异步 .....	60
Symmetric .....	60
对称 .....	60

5.2 数据包概述.....	60
5.3 数据包详细信息.....	62
5.3.1 通用协议数据头格式.....	62
5.3.2 Hello 数据包 .....	63
5.3.3 Error 数据包 .....	63
5.3.4 Echo 请求数据包.....	67
5.3.5 echo 响应数据包.....	68
5.3.6 各厂商数据包.....	68
5.3.7 特征请求.....	68
5.3.8 特征响应.....	69
5.3.9 获取配置请求.....	77
5.3.10 获取配置响应.....	77
5.3.11 设置配置.....	78
5.3.12 Packet-In .....	78
5.3.13 流移除通知.....	80
5.3.14 端口状态通知.....	85
5.3.15 数据包输出 .....	86
5.3.16 流修改.....	93
5.3.17 端口修改.....	95
5.3.18 统计请求 .....	96
5.3.19 状态响应.....	97
5.3.20 Barrier 请求 .....	99
5.3.21 Barrier 响应 .....	99
5.3.22 队列获取配置请求.....	100
5.3.23 队列获取配置响应.....	100

第六章 Vxlan 介绍.....	102
6.1 当前服务器虚拟化设计的难题.....	102
有限的 VLAN 范围.....	102
虚拟机操作.....	102
TOR 交换机的地址表.....	103
多租户.....	103
6.2 VXLAN 概述.....	103
- VNI（虚拟网络标识符）： .....	104
- VTEP（VXLAN Tunnel End Point）： .....	104
虚拟机对虚拟机的流量包.....	104
VXLAN 实现的具体方法.....	104
6.3 个案研究.....	106
6.4 VxLAN 数据包 .....	108
6.4.1 VxLAN 数据头 .....	108
6.4.2 外部 UDP 数据头 .....	109
6.4.3 外部 IP 数据头 .....	109
6.4.4 Outer Ethernet Header .....	110
6.4.5 内部以太网数据头和有效负载.....	110
参考书目 .....	111

## 致谢

感谢我的朋友们多次对此书提出建议，感谢他们对本书出版所作出的贡献。感谢 Ashish Kapur 帮助我完成 VxLAN 这一章节，同时也感谢 Thu Tran 对 OpenFlow 相关章节提供的有用信息。十分感谢我的妻子（Nandini）和儿子（Vinayak）以及我的印度大家庭，我在撰写文稿时未能时常伴其左右，感谢他们对此给予的谅解。

谨以此书献给我的父母，Kusum Shukla 和 P.N. Shukla，感谢他们的鼓励与支持。

## 前言

科学技术在不断发展并且总能够针对某一时段出现的问题，提供优质、简便、合算的解决方案。在后网络 2.0 时代，数据通信多次加速，超过百分之五十的数据通信属于实时视频数据流，因此，设计一个新的网络迫在眉睫。网络的发展需要跟上数据通信加速的步伐（从电路/数据包交换到基于 100G 网速的复杂网络协议），网络发展的下一阶段即是软件定义网络（SDN）。

在过去的几年里，就网络发展如何跟上服务器虚拟化的演变节奏，已经有许多相关研究。从很多企业开始投入大量资金研发 SDN 起，互联网的发展趋势就已经逐渐显现。关于 SDN 的精确定义仍在完善之中，但是 SDN 总的原则和基本协议已经明确了。

作为 SDN 的学习辅导书，本书主要讨论了 SDN 最具发展前景的一种协议：OpenFlow。同时本书也涉及到 SDN 的其他实现：虚拟可扩展局域网（VxLAN）。

本书的编写基于 OpenFlow1.0.0 规范和 VxLAN 草案，希望广大读者积极提供意见建议，以便下个版本的完善。我们的邮箱：[sdnbook@outlook.com](mailto:sdnbook@outlook.com)。

# 第一章 软件定义网络

## 1.1 软件定义网络

数据通信在过去的几年里经历了爆炸式的增长，这种增长逐渐暴露了传统数据网络的不足。软件定义网络就是在这种背景下，为解决传统交换或基于路由的网络部署的瓶颈问题而发展起来的。

SDN 遵循下面三个原则：

- 把控制平面功能从交换专用集成电路（the switching ASIC，交换 ASIC）中分离出来（通过将控制平面功能移入控制器），并使交换 ASIC 仅用于数据平面功能（这样，可以就使得交换 ASIC 商品化并对特定 ASIC 的复杂性进行了简化）
- 中央集中控制器和网络部署（包括网络设备、服务器及虚拟机）的中心观点是将复杂的网路进行具体细化地抽离（这需要基于复杂的路由/交换协议和虚拟机）
- 通过开放标准的验证得出的可扩展性能够借助简单的外部应用程序进行网络编程

基于对 SDN 的不同理解，各生产商研发出几款不同的 SDN 应用产品。一些生产商专注基于 OpenFlow 协议的控制器，一些则试图从虚拟机的角度（实现网络交换机进行抽离）对网络进行改进，例如基于 VxLAN 的方案。但是从本质上来说，SDN 均须满足上面三个原则。

在图 1.1 中，左边部分代表了目前的网络部署，以彼此相连的交换机/路由器及外部设施（服务器/虚拟机(VM)）为特征。目前的网络部署使用分布式协议来建立控制通路。一旦控制通路建立完成，数据平面就已安装在硬件上，数据包一般通过已设定好的路径转发。在目前的网络部署中，为建立控制和数据转发通道，控制协议分布在整個网络，主要分布在第二和第三层协议中，比如 BGP、OSPF、STP。

右边部分代表 SDN 定义的网络，其中，网络操作系统（OS）只在单一实体中运行（比如控制器），所有基于流的决策都由这种实体决定。决策做出后，控制器将对交换机的数据平面进行编程以建立网络。图 1.1 中右边虚线表示控制器和网络设备之间的关系，在这些关系中，控制器可以总揽所有的网络。实线表示交换/路由硬件安装完成后数据转发链路。



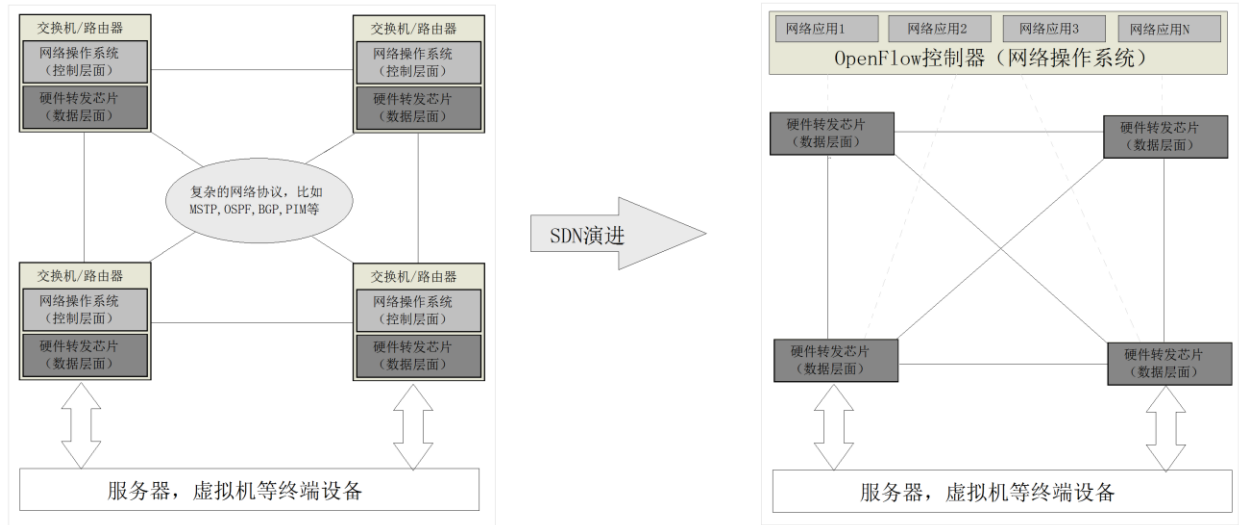


Figure 1.1.

图 1.1.

### 控制平面和数据平面的分离

SDN 的目标是保证所有控制层面上的逻辑决策都由一个中心实体发出。而传统网络控制层面上的决策是从其所所在位置发出，因此每个交换机都需要智能化。SDN 的这种中心决策方法能够降低节点拓扑内对智能节点数的需求量。

SDN 的发展基于 ASIC 硬件商品化的事实，而且，现在建立不同网络需要的 ASIC 并没有很大的不同，其真正的区别是软件。任何网络软件的使用都是通过编程，使数据通过特定的路径进行传输。现在，随着硬件的逐渐商品化和软件对硬件依赖性的减弱，已经没有在所有节点处都运行智能软件的必要了。SDN 概念的实现是通过在某个集中部件（比如控制器）进行软件的逻辑运行，并通过使用南向协议/应用程序接口（API）对交换机（硬件商品）进行指令编程。

### SDN 控制器

中央控制器（SDN 控制器）是一个软件实体，能够覆盖整个网络全局（包括虚拟机和业务流量）。正如在图 1.1 中所阐释的，网络操作系统要想实现所有路径选择的逻辑运算，就须以 SDN 中央控制器为基础，因为控制器了解整个网络的部署，决定最优数据转发路径并对硬件中的条目实现指令操作。现在，大多数的 SDN 控制器使用户图形界面，这样可以将整个网络以可视化的效果展示给管理员。

控制器可以被视为通过运行应用程序进而控制网络的平台。这些应用程序无需考虑控制器所管理的复杂物理网络结构。对流量分析和事件触发进行网络应用编程，就是此种应用的典例。

网络中可以有多个控制器，这样就使得控制平面的可用性增强。当网络中有多个控制器时，就可以进行常规的同步化操作。

## 网络编程的可扩展性

对物理网络拓扑和部署进行分离是 SDN 的主要优势之一。然而，SDN 最吸引人的地方却不在于此。管理员可以编写运行在 OpenFlow 控制器最高层上的应用程序，而且控制器可以与布置数据流的 OpenFlow 交换机进行对话，进而将实际的交换层从应用层分离出来。编写这些应用程序的 API 由控制器提供。

举例来说，管理员可以编写应用程序，使其所在公司的 CEO 无论何时召开网络会议都能被赋予最高的优先级。管理员可以编写一个含有简单触发条件的的基本应用程序，并保存在控制器中，控制器与管理员编写的 API 进行对话，将程序译为控制平面流语言，当程序被触发时，就可以支持该应用程序的运行。控制器可以控制网络硬件设备为 CEO 的组播服务器提供最大的缓冲区，而其他的数据流缓冲区将减少。

## SDN 的逻辑层

图 1.2 解释了 SDN 的三层逻辑层。最低层是硬件交换层，中间层是控制器层，最高层是应用层，在最高层中用户可以自定义应用程序进而触发网络中的流定义。

## 南向 API

南向 API 或协议是工作在两个最底层（交换 ASIC 或虚拟机）和中间层（控制器）之间一组 API 和协议。它主要用于通讯，允许控制器在硬件上安装控制平面决策从而控制数据平面。OpenFlow 协议有足够多的标准来控制网络，因而是最具发展前景的南向协议。

还有其他一些南向通讯实现方式正在研究中，比如 VxLAN。VxLAN 规范记录了终端服务器或虚拟机的详细框架，并把终端站地图定义为网络。VxLAN 的关键假设是交换网络（交换机、路由器）不需要指令程序，而是从 SDN 控制器中提取。VxLAN 对 SDN 的定义是通过控制虚拟机以及用 SDN 控制器定义基于这些虚拟机通信的域和流量而实现的，并不是对以太网交换机进行编程（这种假定可以使用现有协议为以太光纤网路提供路径）。

最后，在网络端到端视图方面（包括交换/路由设备和虚拟机/端节点），南向 API 需要改进，改进后，管理员可以借助单一的 SDN 控制器对网络设备（交换 ASIC）和虚拟机进行指令控制。

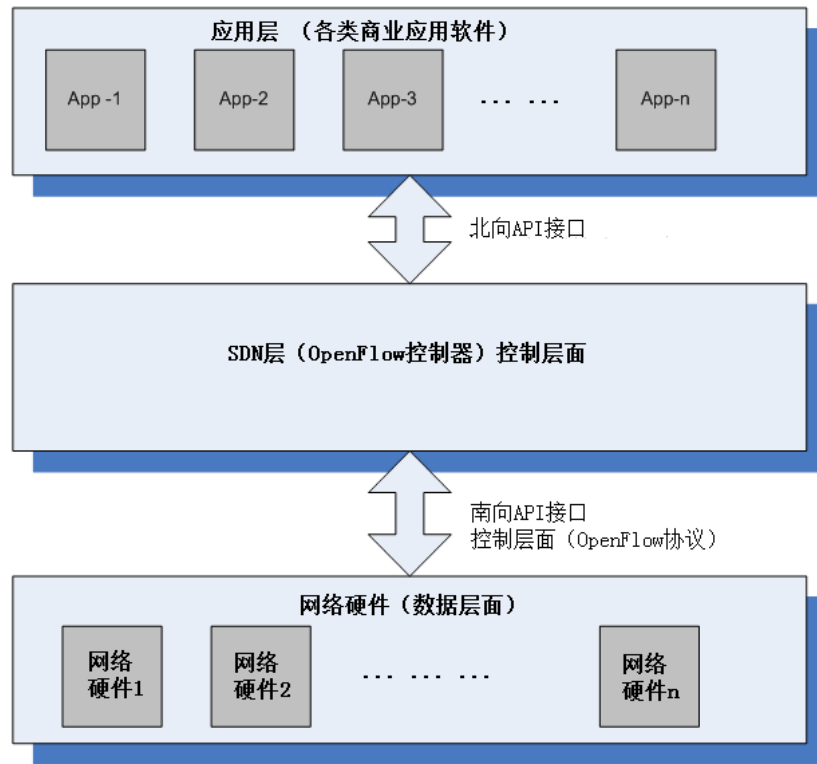


图 1.2.

### 北向 API

在 SDN 的逻辑中层和高层存在一个 API 级别的通信，其特征是，用户可以定义应用程序，该应用程序可以通过北向 API 与控制器实现通信，然后控制器将其编译后发送至最底层（比如硬件交换层）。到目前为止，还没有太多关于北向 API 的使用，但是随着 SDN 的发展，越来越多的北向 API 将自然渐进的发展起来。就现在来说，Quantum API 是一个与许多 OpenFlow/SDN 控制器集成在一起的开放的北向 API。

### 可行性的技术

总结下来，下面这些是支持 SDN 所有可行性的技术：

- 商品化（可编程）的交换机和服务/虚拟机
- 南向 API/协议（OpenFlow 协议）
- SDN 控制器
- 北向 API

本书主要关注 OpenFlow 技术，也对 VxLAN 做了概念层面上的简述。

## 1.2 什么是 OpenFlow?

正如在 1.1 节中所阐释的，OpenFlow 是一种南向协议。下面介绍关于 OpenFlow 更多的内容。

- OpenFlow 是软件定义网络的一个例子，顾名思义，OpenFlow 就是指两个端点之间的网络连接是通过运行在外部电脑/服务器上的软件来定义的，而且硬件（交换机）上的指令行为基于控制器的智能决策。
- OpenFlow 是一个开放的，基于一定标准的协议。它定义了如何由一个中心部件（控制器）对控制平面进行配置和控制。通过使用 OpenFlow，控制器可以管理数据包在网络中的传输。
- 在传统网络中，交换机和路由器的信息以不同的形式（路由表、Mac 表等）进行储存，需要在整个网络或一组网络中通过复杂的交换和路由协议进行计算才能得到。根据这些不同的表，可以对数据平面进行指令操作。OpenFlow 协议将协议规范化并集中化，通过创建和管理流信息表以代替其他所有的转发表。数据平面就是根据这些流信息表生成的。

## OpenFlow 组件

如图 1.3 所示，OpenFlow 最主要的组件是：OpenFlow 控制器，OpenFlow 交换机，OpenFlow 协议。

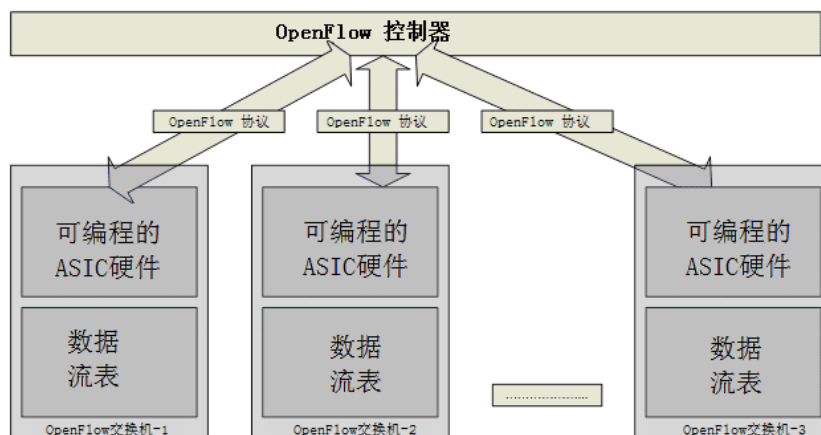


图 1.3.

## OpenFlow 控制器

这是该协议的大脑，使所有基于业务流的智能决策，并将这些决策发送给 OpenFlow 交换机。这些决策以指令的形式存在于流表中。典型的每个流信息决策的形式可以是：添加、删除及修改 OpenFlow 交换机中的流表；通过配置 OpenFlow 交换机，可以将所有未知数据包转发给控制器；也可以在 OpenFlow 流表进行其他一些指令。

定义流的时候需要考虑一些变量，这些变量叫做元组。根据 OpenFlow1.0 规范，在进行决策时，需要考虑的变量有 12 个。

**OpenFlow 交换机**

在硬件层面，OpenFlow 交换机类似于现在网络中使用的典型网路交换机。但其中不包含智能软件。在 OpenFlow 交换机中，OpenFlow 控制器管理硬件的流表，交换机的性能主要是数据平面上的转发。控制通路由 OpenFlow 控制器控制管理，而数据通路则建立在由 OpenFlow 控制器编制的 ASIC 指令基础之上。

**OpenFlow 协议**

同其他网络协议一样，OpenFlow 协议的最终目标是实现对数据通路的程序指令，但是，OpenFlow 实现数据通路指令的方法却有所不同。OpenFlow 是客户端服务器技术和各种网络协议的融合。本书通过深入研究数据包而对 OpenFlow 进行详细的阐述。第 2-5 章对 OpenFlow 进行了详细的阐述。

**1.3 为什么发明 OpenFlow?**

需求是发明的动力。要想知道 OpenFlow 缘何被发明出来，就需要从批判的视角来审视当前网络技术的缺陷。

目前的网络部署是基于大约 5000 个 RFC（注释请求文档）—或者是基于数以百计的专用硬件芯片组等。下面列举的是现有网络技术中的典型应用和存在的问题。

当前数据中心网络部署情况	存在的问题
由大量的路由器/交换机组成的网络，每个机架都安装 TOR 交换机用来连接服务器	所有交换机需要管理和单独配置，大量复杂和重复的工作， 随着数据中心服务器的不断更新网络变化，维护工作量会递增。
服务器的虚拟化技术让数据中心网络变得更复杂	服务器的虚拟化直接导致网络配置要经常地变动以适应业务的变化
数据中心网络设备多数是基于硬件与软件结构	灵活性和可扩展性不够
网络设备有大量的供应商	需要管理不同厂商的设备，使用不同的网管，相互的兼容性差，故障定位困难，对接问题多，维护成本较高。

表 1.1.

表 1.1 列举了目前网络部署的一些问题。其中绝大多数都可以通过 OpenFlow 协议解决：

- 控制器做所有控制层面上的决策，减少对智能交换机的需求，进而减少购买智能交换软件的巨大成本。
- 由于交换机不需要分开配置或管理，网络自动化的花费将降低，同时，网络的平面化视图使得管理更为简便。
- 不依赖于复杂的路由或交换协议。

根据 SDN 的概念，网络可以实现可编程，而管理员可以基于应用程序对网络编制程序。与当前网络使用的离散方法相比，把网络中各部分集合起来是一种更为直接的方法。

## 第二章 OpenFlow 概述

这一章从总体上对 OpenFlow 进行阐释。用一个很简单的例子解释当一个数据包进入 OpenFlow 交换机的时候会发生什么。OpenFlow 协议的工作方式与服务器-用户端工作模式很相似，一个服务器（运行控制器软件）通过程序指令告诉客户端（或是交换机）如何处理数据流。为了较快地了解 OpenFlow 工作原理，在这一章中，我们将对基于 OpenFlow 流程图中的步骤进行逐一介绍。

### 2.1 OpenFlow 协议概述

在图 2.1 所示的流程图中，字母代表了步骤，步骤从 A 开始（步骤 0 是发现阶段）。为了简化该流程图，我们假定其拓扑结构是 OpenFlow 交换机和 OpenFlow 控制器之间的单段链路。如果从步骤 0 开始，那么需要连通 OpenFlow 交换机（纯 OpenFlow 模式下运行）和 OpenFlow 控制器。

#### 步骤 0

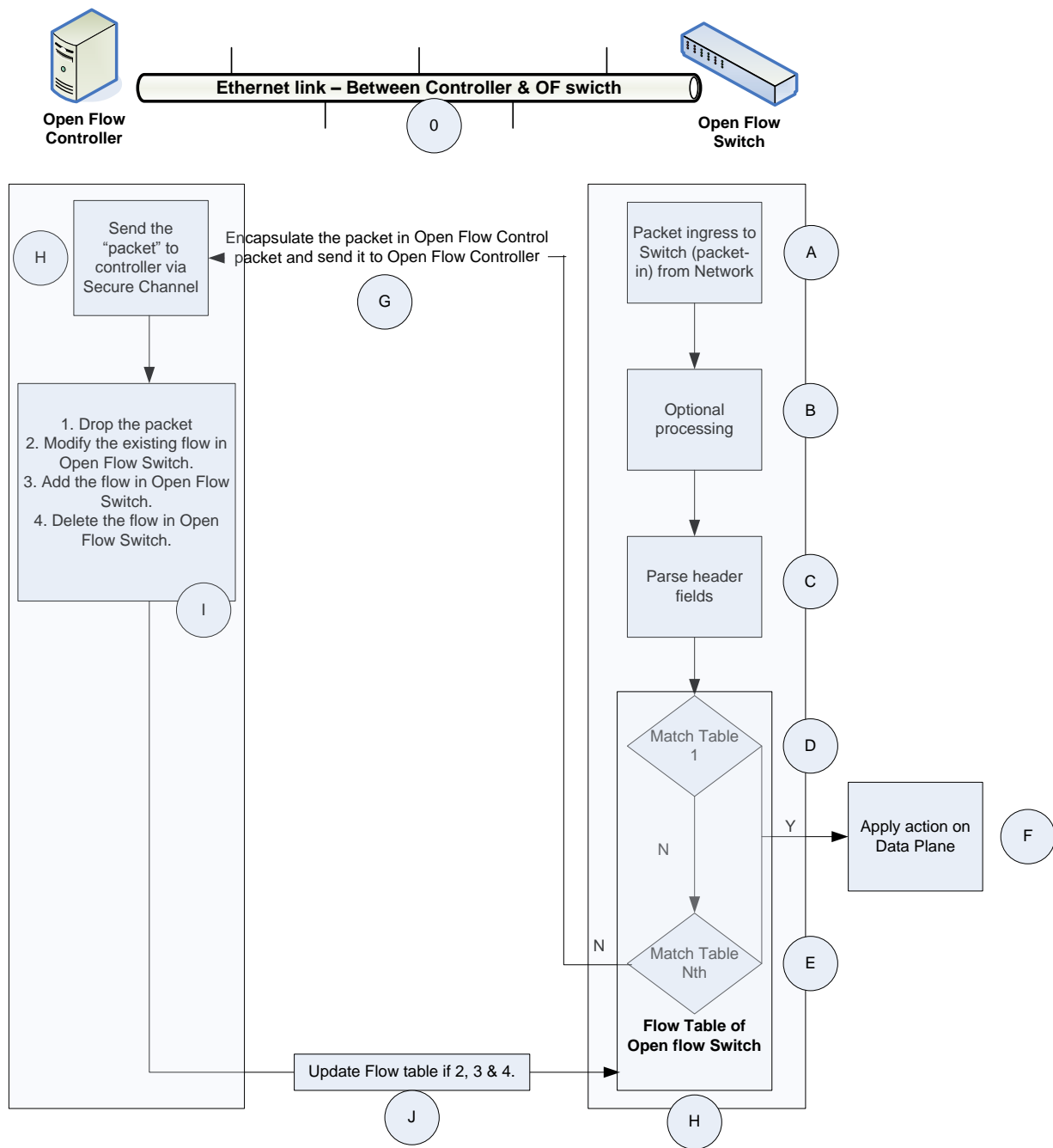
这一步骤表示的是在 OpenFlow 交换机与 OpenFlow 控制器之间建立连接。一个控制器与一个交换机之间的连接是一对一的（通过网络管理可以访问多个 OpenFlow 交换机）。用户端和服务器之间的连接需要 TCP 协议。后面章节我们将详细讨论这个连接是怎样建立的。

#### 步骤 A

这一步骤说明了一个数据包进入 OpenFlow 交换机的过程。在这一步骤中，我们假设 OpenFlow 交换机在 OpenFlow 模式下运行，这样，当数据包进入交换机后，将被交给交换机中运行的 OpenFlow 模块。这个数据包可以是控制报文或数据报文。

#### 步骤 B

在这一步骤中，交换机负责 PHY（物理层）级的处理—这取决于 OpenFlow 交换机使用的连接材料的材质（光纤或者铜）。处理完毕后，数据包将交给在客户端交换机上运行的 OpenFlow 客户端。



Fi

图 2.1. OpenFlow 协议流程图

### 步骤 C

当数据包在交换机内进行处理的时候，交换机内的 OpenFlow 协议将对数据包的数据头进行交换分析，从而判定数据包的类型。通常方法是提取数据包的 12 个元组之一，这样就可以根据后面步骤中安装的流表中对元组进行匹配。



#### **步骤 D 和 E**

当交换机获取了可进行匹配的元组信息后，就开始从头至尾对流表进行扫描。流表中的操作在扫描过后就将被提取出来（下一节做详细解释）。可以通过不同的实现方式来优化扫描，减少扫描量，提高流表使用效率。

#### **步骤 F**

元组匹配后，与特定流相关联的操作就完成了。这些操作通常包括转发数据包、丢弃数据包，或者将数据包发送至控制器。如果执行的是从交换机中转发数据包的操作，那么数据包就将被交换。如果没有针对流的操作，那么流就会被后台丢弃（从而数据包也将被丢弃）。如果流中没有与数据包匹配的条目，数据包则将被发送至控制器（此数据报将会被封装在 OpenFlow 控制报文中）。

#### **步骤 G 和 H**

如果从数据包中提取的元组与任何流表中的条目都不匹配，数据包则将被封装在数据包的数据头中，并发送到控制器进行进一步的操作。按照协议，控制器需要决定对于这个新的流应该进行哪些操作。

注意：关于数据头中数据包会在后面章节做进一步阐释。

#### **步骤 I**

控制器根据自身配置，决定如何处理数据包。处理方式包括丢弃数据包（不进行任何操作）、安装或修改 OpenFlow 交换机内的流表，或者修改控制器的配置。

#### **步骤 J**

控制器通过发送 OpenFlow 消息来对交换机中的流表进行增添或修改的处理。

当流表安装结束后，下一个数据包将从步骤 F 开始处理。对于每个新的流，流程是一样的。可以根据 OpenFlow 交换机与 OpenFlow 控制器的具体实现对流进行删除操作。

## **2.2 OpenFlow 组件概述**

### **2.2.1 OpenFlow 控制器(Controller)**

OpenFlow 控制器是所有基于 OpenFlow 软件定义网络的核心。OpenFlow 控制器运行控制平面，并通过控制平面逻辑对网络交换机中的流进行指令操作（比如，控制器计算最短路径，然后对交换机进行指令控制）。控制器运行于功能强大的服务器之上，保证所有 OpenFlow 交换机可以访问。管理员或者用户可以通过功能强大的图形用户界面以及很多控制功能对网络进行控制，从而方便了操作。

如果流表中没有列出新的事件发生，OpenFlow 交换机将向控制器寻求帮助。现在市场上销售的控制器一般都能够能够在图形用户界面上为用户提供整个网络的视图。很多控制器可以将网络中的流量形象化地表示出来。

SDN 还有其他一些改进的地方，比如网络管理员可以在 SDN 控制器上写入应用程序。目前的控制器都提供 API，可以用来开发可在控制器上运行的应用程序和自定义控制平面的逻辑。通过在控制器上运行应用程序，网络管理员可以实现对流的控制。

### 2.2.2 OpenFlow 交换机

OpenFlow 交换机运行软件与 OpenFlow 控制器相连，并进行流的安装和分析。在这一阶段，对 OpenFlow 交换机的定义是各厂商特定的。OpenFlow 交换机可以分为两类：

1. 纯 OpenFlow 交换机：这种交换机只支持 OpenFlow 协议。
2. 混合 OpenFlow 交换机：这种交换机同时支持传统以太网协议和 OpenFlow 协议。

不考虑交换机的类型，任何 OpenFlow 交换机都有模块用来负责与 OpenFlow 控制器进行 SSH 或 TCP 连接。

### 2.2.3 流表

流表存在于 OpenFlow 交换机，并指示交换机如何处理进入交换机的流量。OpenFlow 控制器将根据各种流和控制器的物理拓扑结构，将流表安装在 OpenFlow 交换机中。

OpenFlow 交换机将依照这个流表对所有进入其中的流量进行处理。如果流表中没有关于某特定流的条目与之对应，数据包信息则会被发往控制器，由控制器做出处理。控制器决定如何处理之后，就在 OpenFlow 交换机中通过相应的操作措施来对流进行处理。

图 2.2 总结了 OpenFlow 交换机中流表和流的匹配过程。

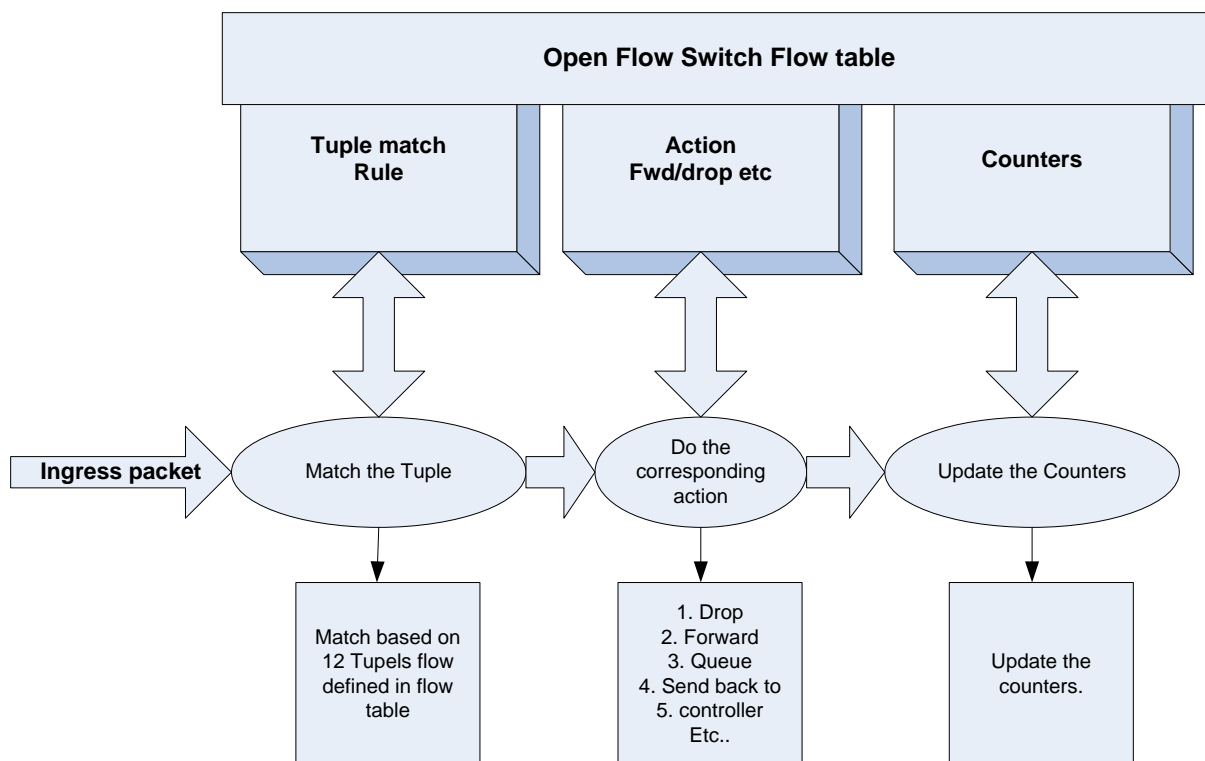


图 2.2.

每个流表中条目都包括：

- 数据头：与进入数据包或流相匹配（通常称为元组）
- 计数器：更新已经匹配的数据包或流字节
- 操作：流量匹配后进行的一些操作

#### 流表数据头（元组）

任何进入交换机的数据包都需要与数据头 12 个元组中的某一特定值进行匹配（更精确的匹配可以通过掩码实现）。目前来看，大多数元组都是各厂商预先设定的。

字段	Bits	适用范围	说明
Ingress Port	16	所有报文	入端口编号
Ethernet Source address	48	所有以太网报文	以太网源 MAC 地址
Ethernet Destination address	48	所有以太网报文	以太网目的 MAC 地址
Ether Type	16	所有以太网报文	以太类型

Vlan ID	12	以太类型为 0x8100 的报文	VLAN iD, 值为 0xffff 表示 untagged
Vlan priority	3	以太类型为 0x8100 的报文	VLAN 优先级, 值为 0-7
IP Source Address	32	IP 和 ARP 报文	源 IP 地址
IP Destination Address	32	IP 和 ARP 报文	目的 IP 地址
IP Protocol	8	IP 和 ARP 报文	IP 协议字段
IP ToS bits	6	所有 IP 报文	IP TOS 值
TCP/UDP Source Ports	16	TCP、UDP 和 ICMP 报文	源 TCP/UDP 端口
TCP/UDP Destination ports	16	TCP、UDP 和 ICMP 报文	目的 TCP/UDP 端口

**表 2.1** 数据头（12 个元组）

#### 匹配举例（基于 IPv4 目标的匹配）

图 2.3 是为除了目标 IP 地址（10.1.1.1）外的所有元组设置的带有通配符掩码的一个流表。

图中显示的进入交换机的数据包是以太 0x0800 型，目标 IP 地址为 10.1.1.1，使用 OpenFlow 协议。然后这个数据包将与流表中的条目相匹配。图中所示的流表，其他域都是通配符掩码。数据包需要寻找一个可以匹配的元组。

如果进入的数据包能够与这一流条目匹配，这一流所对应的操作域就会做出相应的操作（如果没有操作对应，数据包将会被抛弃）。当流表中没有条目可以匹配的时候，默认操作是封装数据包，并发送至控制器。

在决定对数据包采取某一操作后，计数器就会进行更新。计数器可以被用在很多方面。

OpenFlow流表中的一条流实例			
流表中的12元组	各元组可以用来匹配的值 (*号表示此项为空)	动作 (Action)	计数器
Ingress Port	*	Action list	
Ethernet Source address	*		
Ethernet destination address	*		
Ether Type	0x0800		
VLAN ID	*		
VLAN Priority	*		
IP Source Address	*		
IP Destination address	10.1.1.1		
IP protocol	*		
IP TOS bits	*		
TCP/UDP Source ports	*		
TCP/UDP Destination ports	*		

图 2.3.

### 计数器

这是流表的第二个组件，根据 OpenFlow 规范 1.0.0，计数器记录流、端口、队列和表。使用怎样的计数器同样是由各厂商来决定。现在主要使用计数器将统计信息发送至控制器。

### 动作 (Action)

这是流表的第三个组成部分，指定当进入数据包与流表中条目相匹配后应该进行怎样的动作（如图 2.3 所阐释的）。流条目可能与零个或者多个动作相关联。交换机根据流表中的动作要求对进入的数据包进行处理。如果没有与流相关联的动作，交换机将抛弃数据包。如果有超过一个的操作与流条目相关联，将按照流表中所给出的顺序进行执行。如果交换机无法执行动作，或者无法执行控制器指令的动作，将会发送错误信息到控制器，并不再对流进行操作。

Action 分类	Action	参数	说明
必选 Action	转发	指定端口(Port)	将报文转发到指定的物理端口或逻辑端口
		所有端口(All)	将报文转发到指定的物理端口或逻辑端口
		控制器(Controller)	将报文转发给控制器
		本地端口(Local)	将报文转发到指定的本机非 Openflow 端口（主要应用在 Openflow 混合模式交换机上）
		流表(Table)	将报文转发给其它的流表（适用于支持多流表的 OpenFlow 交换机）
		入端口(In Port)	将报文转发到接收此报文的端口
	丢弃		丢弃此报文
可选 Action	转发	普通的 (Normal)	将报文按正常的二层交换机的方式，查 MAC 表转发（主要应用在 Openflow 混合模式交换机上）
		洪泛 (Flood)	将此报文转发到生成树链路上，不包括入端口
	改变元组值		改变报文中指定字段的值，如果还存在其它的 Action, 此操作会被优先执行。详细信息见表 2.4

	入队列		将报文转发到指定端口的接定队列中，然后按照端口的 QoS 队列配置来转发。
--	-----	--	---------------------------------------

表 2.2 动作总结.

表 2.2 中展示了详细的动作分类说明。

### 动作分类

根据 OpenFlow 1.0，动作可以分为两类。就目前来说，OpenFlow 交换机需要支持必备动作。

#### 必备动作

- 转发数据包到已知的物理端口或虚拟端口（注：虚拟端口是一组具有类似特征的端口
- 如果没有任何动作与流条目对应，则丢弃数据包

#### 可选动作：

- 按照传统交换机的第二层或第三层的协议进行数据包转发。如果交换机支持这一功能，它就可以使用正常的第二层或者第三层的协议对数据包进行转发（比如 VLAN 转发）
- 在其他操作采取前对域进行修改
- 将数据包入队—指定数据包进入由端口服务质量定义的队列

### 修改域

这是 OpenFlow 规范 1.0 中的一个可选操作。该操作可以使协议更加灵活，并能够更好的控制流中的特定域。表 2.3 总结了所有可以进行修改域的操作。

Action 动作	需要修改的比特	描述
设置 VLAN ID	12	该动作指定 VLAN ID 可以修改（添加新的 VLAN ID 或者更改 VLAN ID）。
设置 VLAN 优先级	3	该动作可以修改指定的 VLAN 优先级。该操作可以修改 VLAN 优先级，从而为一些特殊应用场景提供了解决方案。
剥离 VLAN 数据头		剥离 VLAN 数据头
修改源 MAC 地址	48	用新 MAC 地址代替现有的以太网源 MAC 地址，可以提供更好的数据包层面的控制。
修改目的 MAC 地址	48	用新 MAC 地址代替现在的以太网目的 MAC 地址，可以提供更好的数据包层面的控制。

修改源 IPv4 地址	32	用新的 IP 地址代替现在的 IP 源地址并更新 IP 校验和（以及 TCP/UDP 校验和）。这一操作只适用于 IPv4 的数据包。
修改目的 IPv4 地址	32	用新的 IP 地址代替现在的 IP 目标地址并更新 IP 校验和（以及 TCP/UDP 校验和）。这一操作只适用于 IPv4 的数据包。
修改 IPv4 的 TOS 字段	6	更改 IP 的 TOS 域。这一动作仅适用于 IPv4 数据包。
修改 TCP/UDP 源端口	16	用新的 TCP/UDP 端口代替现在的 TCP/UDP 源端口，并更新 TCP/UDP 校验和。这一操作仅适用于 IPv4 数据包。
修改 TCP/UDP 目的端口	16	用新的 TCP/UDP 端口代替现在的 TCP/UDP 目标端口并更新 TCP/UDP 校验和。这一操作仅适用于 IPv4 数据包。

表 2.3 修改操作（Action）列表

## 第三章 OpenFlow 内部事件

本章主要讲述触发不同种类数据包的事件，并对触发条件进行详细的讨论。触发条件引起的数据包交换将在本章中进行简单讲述，在第五章中将做详细阐明。我们以梯形图的方式对这些事件进行讨论，从而使读者对事件顺序的能够有清楚的认识。虽然我们对每个事件分开阐述，但是在 OpenFlow 操作中，这些事件并不是独立发生的。分开讲述是为了更好的为读者作出解释。

我们对于这些事件的分析，都假设只有一台 OpenFlow 交换机与 OpenFlow 控制器。

### 3.1 连接建立事件

这一事件揭示了交换机如何与 OpenFlow 控制器建立连接以及数据包的序列，这一序列对于成功建立连接必不可少。这一事件同时也揭示了连接建立后的初次数据包交换。图 3.1 展示的是建立连接的一个简单的例子。图中的标识标注了不同的触发条件和过程。我们分步对于这些标签进行阐述。

#### 步骤 A

最初在 OpenFlow 交换机与 OpenFlow 控制器之间通过 TCP 三次握手过程建立连接，使用的 TCP 端口号为 6633。这一步骤就是指三次握手时发生。

#### 步骤 B

TCP 连接建立后，交换机和控制器就会互相发送 hello 报文。Hello 报文是使用 OpenFlow 协议的一个对称的数据包。Hello 报文中唯一的内容是 OpenFlow 报文头中的“类型值=0”。

#### 步骤 C

当 Hello 报文在交换机和控制器相互交换后，OpenFlow 控制器与交换机之间就建立的连接。根据各厂商提供的不同类型的实现方式，这一连接可以通过控制器的图形用户界面或者交换机的命令行界面观测到。在一个典型的控制器中，交换机应当出现在控制器的图形用户界面中。



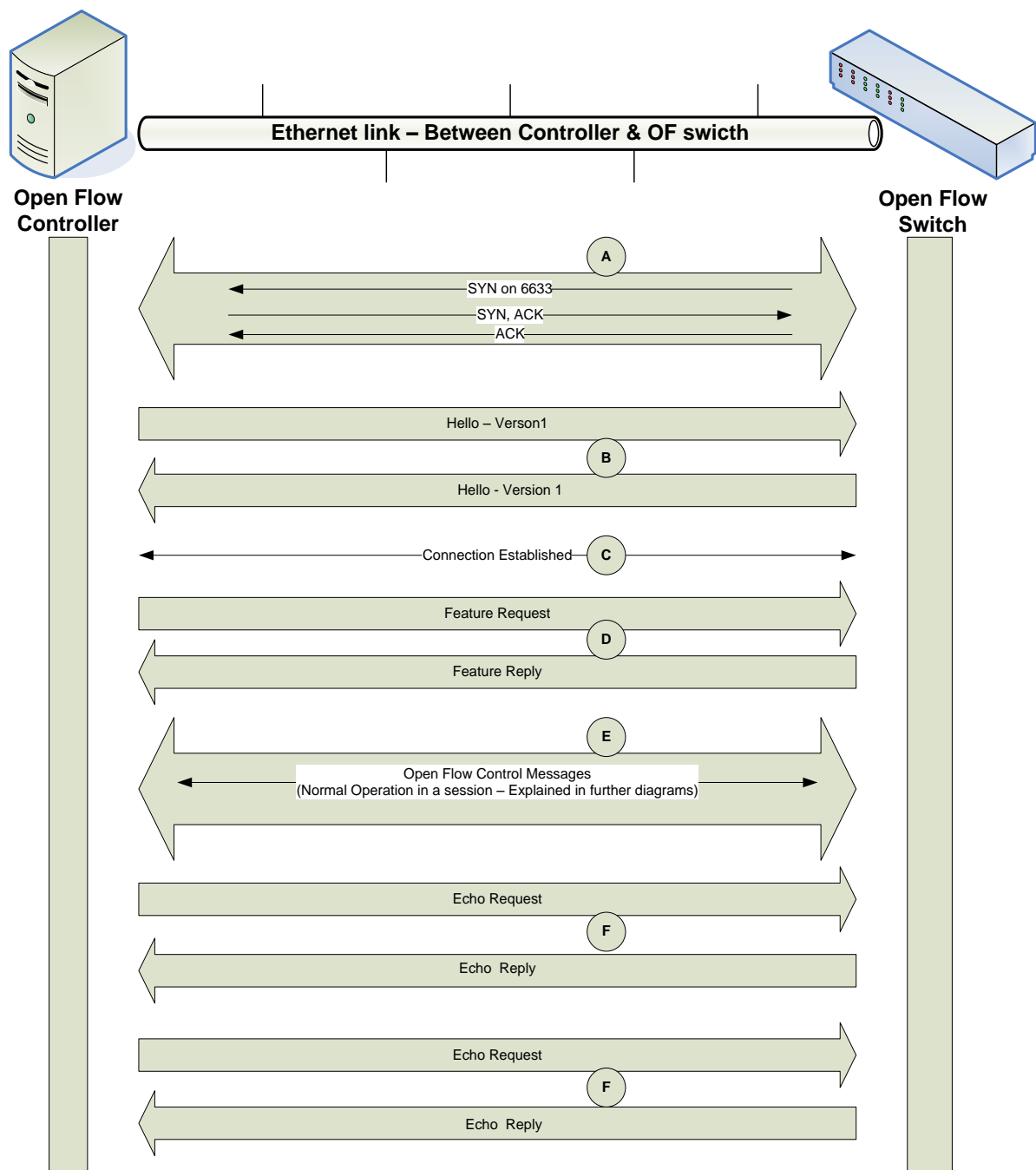


图 3.1 OpenFlow 交换机与控制器之间的连接建立。

#### 步骤 D

**功能请求 (Feature Request)**。这是控制器发向交换机的一条 Openflow 消息，目的是为了获取交换机性能，功能以及一些系统参数。该报文中 OpenFlow 数据头“类型值=5”。

**功能响应(Feature reply)**。这是由交换机响应功能请求报文 (Feature Request) 向控制器发送的功能响应 (Feature reply) 报文。这一报文中描述了 OpenFlow 交换机的详细信息，内容有：

- 数据路径 ID
- 流表数
- 可以缓冲处理的数据包数量
- 交换机性能
- 支持的操作标识
- 端口描述

#### 步骤 E

控制器获得交换机性能信息后，OpenFlow 协议特定操作就可以开始进行了。这些操作的细节在下面的事件表中列出。

#### 步骤 F

Echo 请求 (Echo request) 和 Echo 响应 (Echo reply) 属于 OpenFlow 中的对称型报文，他们通常作为在 OpenFlow 交换机和 OpenFlow 控制器之间保持连接的消息 (Keep-alive) 来使用。通常 echo 请求使用 OpenFlow 数据头 “类型值=2”，echo 响应使用 OpenFlow 数据头 “类型值=3”。不同各厂商提供的不同实现中，echo 请求和响应报文中携带的信息也会有所不同。

## 3.2 Packet-In 事件

这一节将阐释 Packet-In 事件的触发条件和过程。图 3.2 展示的是一个独立的 Packet-In 事件。图中的标识标注了不同的触发条件和协议报文，下面我们就对这些流程进行解释。

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是 Packet-In 事件发生的前提。

#### 步骤 B

这一步骤说明的是交换机怎样触发 Packet-In 事件。当 OpenFlow 交换机收到数据包后，如果流表中与数据包没有任何匹配条目，这时候 Packet-In 事件就被触发了，交换机会将这个数据包封闭到 Openflow 协议报文中发送至控制器，。

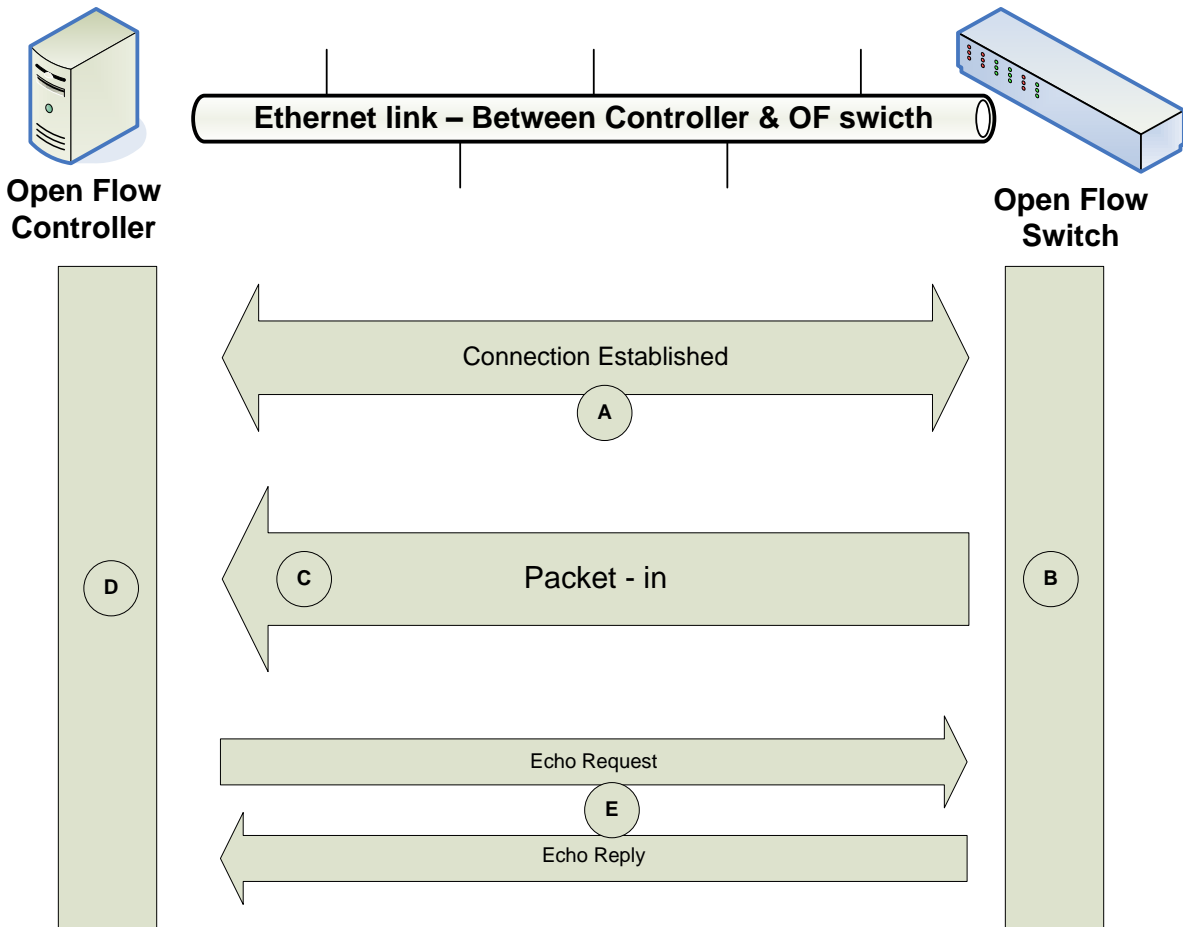


图 3.2 Packet-In 事件

### 步骤 C

一旦交换机触发了 Packet-In 事件，Packet-In 报文就将发送至控制器。

Packet-In 数据头包括了：

- 缓冲 ID
- 数据包长度
- 输入端口
- 原因
  - o 0: 无匹配
  - o 1: 流表中明确提到将数据包发送至控制器
- 数据结构—OpenFlow 交换机所实际收到的数据包

### 步骤 D

控制器收到 Packet-In 报文，并提取 OpenFlow 数据头。Packet-In 域提供数据包的信息，这些信息都是在那些特定的 Packet-In 被封装的。得到 Packet-In 信息后，控制器根据需要对原始数据包做出处理。

## Step E

### 步骤 E

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

## 3.3 Packet-Out 事件

这一节中我们将阐释 Packet-Out 事件的触发条件和过程。图 3.3 展示的是一个独立的 Packet-Out 事件。图中的标识标注了不同的触发条件和过程，下面我们就对这些流程进行解释。

### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是 Packet-Out 事件发生的前提。

### 步骤 B

控制器要发送数据包至交换机时，就会触发 Packet-Out 事件将数据包发送至交换机。这一事件的触发可以看做是控制器主动通知交换机发送一些数据报文的操作。通常，当控制器想对交换机的某一端口进行操作时，就会使用 Packet-Out 报文。

### 步骤 C

该数据包由控制器发往交换机，内部信息使用 Packet-Out 型，并由 OpenFlow 数据头封装。

OpenFlow Packet-Out 信息包括：

- 缓冲 ID
- 入口端口编号
- 动作明细（添加为动作描述符）
  - o 输出动作描述符
  - o VLAN VID 动作描述符
  - o VLAN PCP 动作描述符
  - o 提取VLAN标签动作描述符
  - o 以太网地址动作描述符
  - o IPv4地址动作描述符
  - o IPv4 DSCP动作描述符
  - o TCP/UDP端口动作描述符
  - o 队列动作描述符
  - o 各厂商动作描述符
- 数据结构

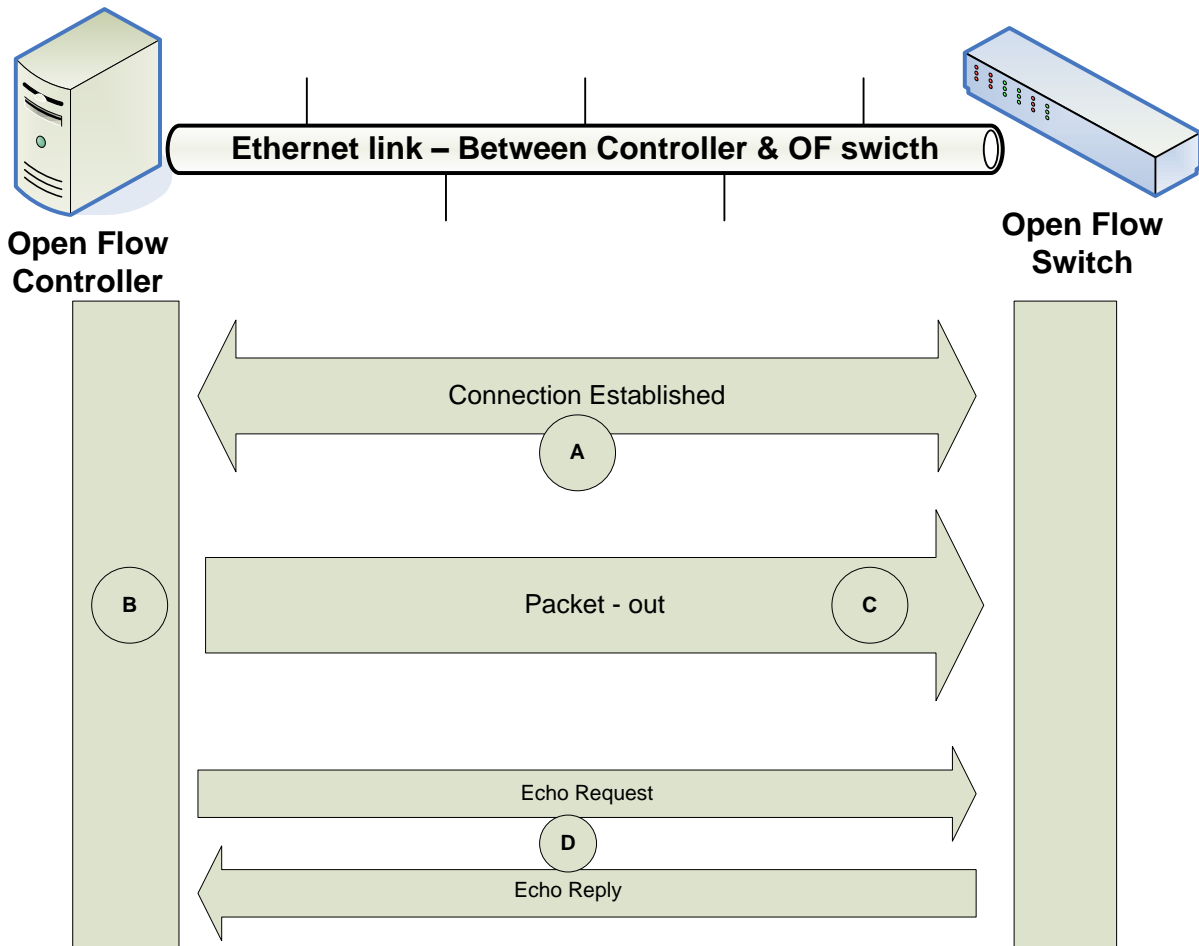


图 3.3 Packet-Out 事件

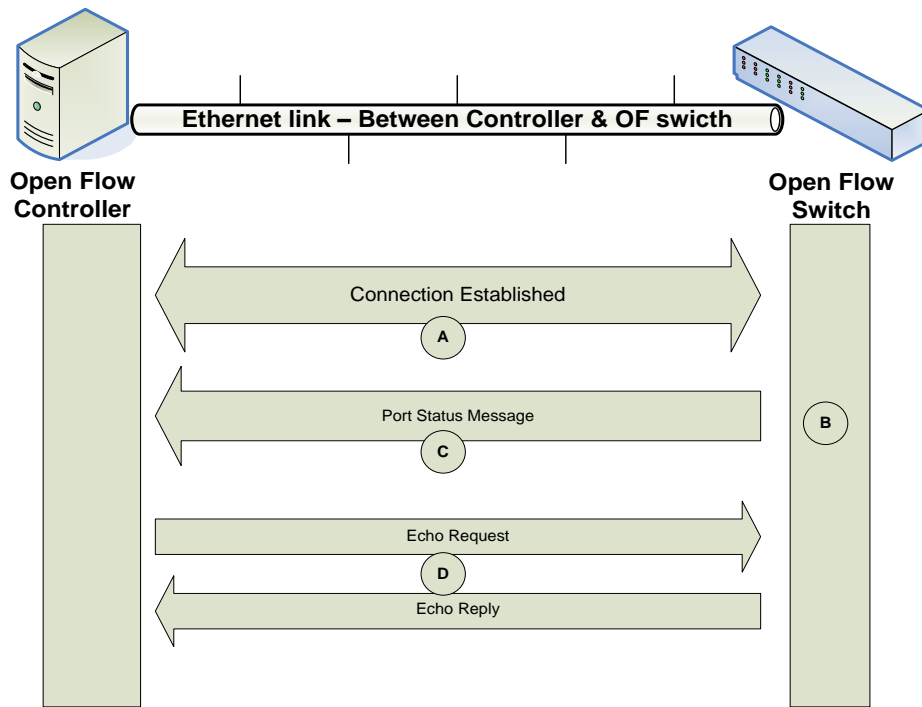
## Step D

### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

## 3.4 端口状态（Port Status）消息事件

这节将阐述端口状态消息的触发条件和报文。图 3.3 展示的是一个独立的端口状态消息事件。图中的标识标注了不同的触发条件和过程，下面我们就对这些流程进行解释。



Fi

图 3.4 端口状态（Port Status）消息事件

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是端口状态消息（Port Status）事件发生的前提。

#### 步骤 B

交换机端口状态发生改变（端口 up/down、增添或移除）或者端口配置标志发生改变时，会触发端口状态（Port Status）消息事件的发生。这一消息由 OpenFlow 交换机触发，端口状态(Port Status)消息由 OpenFlow 交换机发往控制器，用于通告交换机端口状态的变化。

#### 步骤 C

交换机发送的端口状态（Port Status）信息包括：

- 原因
  - 0 表示增加端口
  - 1 表示增加端口
  - 2 表示修改端口
- 端口描述符

- 端口数
- 以太网(MAC)地址
- 端口描述（名称）
- 端口配置标志
  - 管理员故障
  - 没有STP
  - 没有接收
  - 没有接收STP
  - 没有洪泛
  - 没有转发
  - 没有Packet-In
- 端口状态标志
  - 连接状态
    - STP状态
- 即时端口特性标志
  - 连接速度
  - 媒介（连接介质：铜、光纤，等等）
  - 自动协商
  - 暂停
- 通告端口特性标志
  - 连接速度
  - 媒介（连接介质：铜、光纤，等等）
  - 自动协商
  - 暂停
- 支持端口特性标志
  - 连接速度
  - 媒介（连接介质：铜、光纤，等等）
  - 自动协商
  - 暂停
- 链路层相邻通告端口特性标志
  - 连接速度
  - 媒介（连接介质：铜缆、光纤，等等）
  - 自动协商
  - 暂停

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.5 设置配置（Set Configuration）事件

本节将阐述设置配置（Set Configuration）信息的触发条件和过程。图 3.5 展示的是一个独立的设置配置（Set Configuration）事件。图中的标识标注了不同的触发条件和过程。

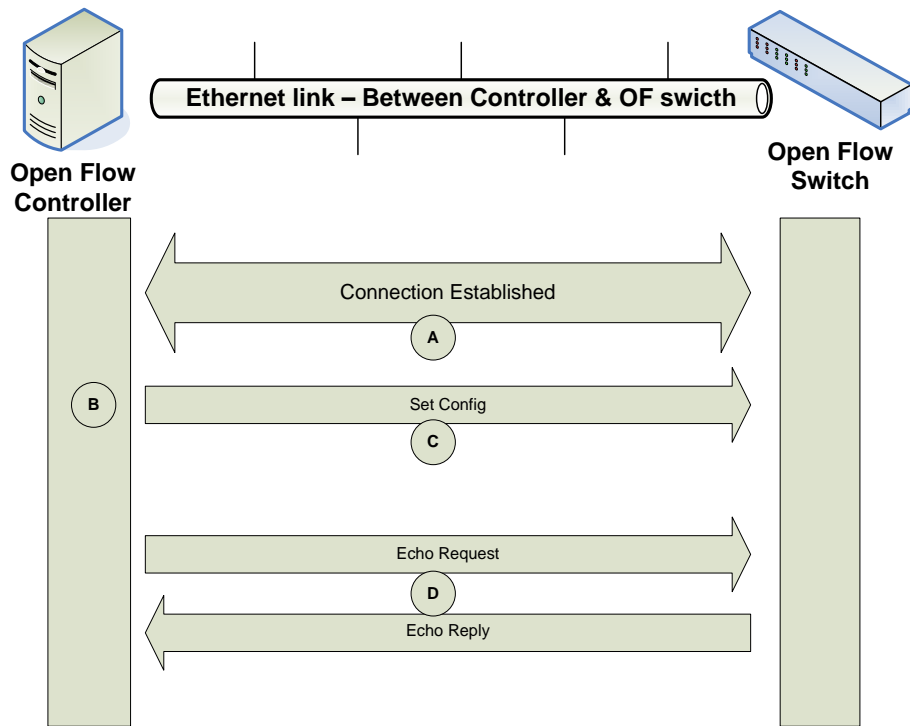


图 3.5 设置配置（Set configuration）事件

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是设置配置（Set configuration）事件发生的前提。

#### 步骤 B

当 OpenFlow 控制器设置 OpenFlow 交换机的配置时，触发设置配置（Set configuration）数据包。



#### 步骤 C

当控制器想要改变交换机的配置时，就会发送一个设置配置信息，这信息是控制器—>交换机信息。设置配置信息包括：

- 交换机配置标志
- Miss发送长度
  - 表示重新配置后发送至控制器的流的最大8位字节，默认值128。

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.6 获取配置（Get Config）请求与答复事件

本节将阐述配置请求与答复消息的触发条件和过程。图 3.6 展示的是一个独立的获取配置请求与答复的事件。图中的标识标注了不同的触发条件和过程。

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是获取配置请求与答复事件发生的前提。

#### 步骤 B

当控制器想要获取交换机中的配置信息时，就会发送消息“Get Config”

#### 步骤 C

获取配置请求的报文中没有内容（只包含 OpenFlow 常规数据头）；OpenFlow 交换机通过“TypeCode = 7”识别这个报文。

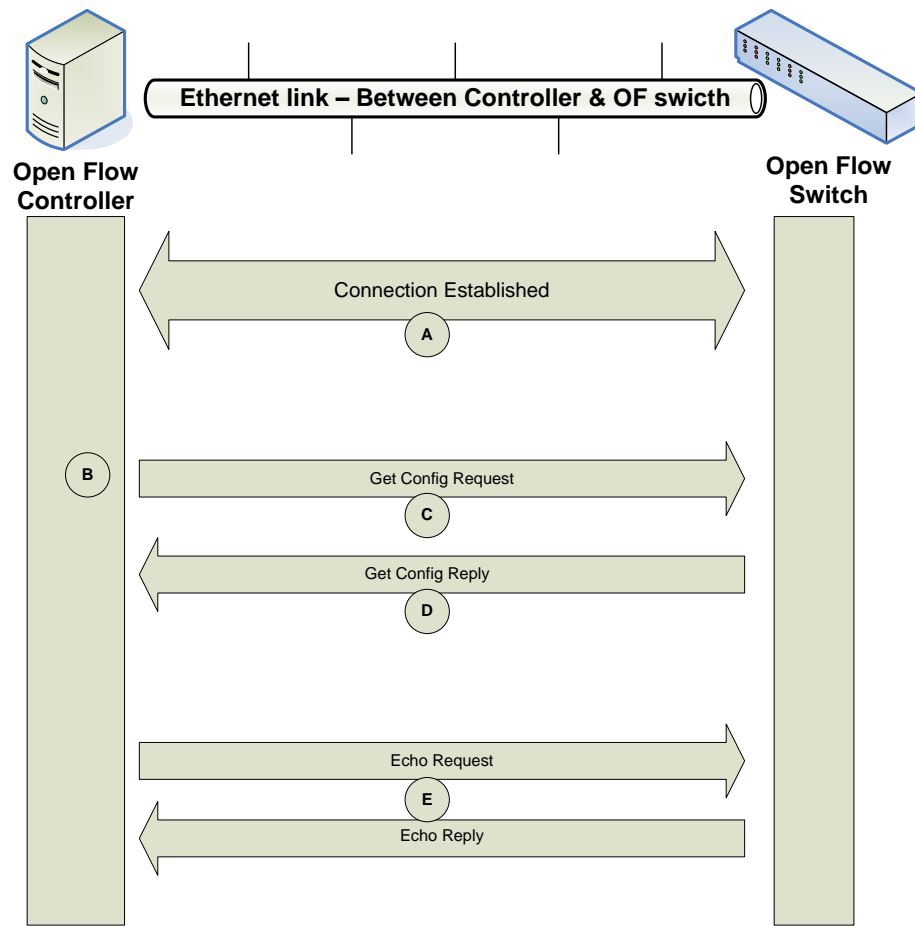


图 3.6 配置请求与答复事件

#### 步骤 D

交换机发出配置答复消息作为反馈，该消息包含了交换机的所有配置信息，配置答复消息包括：

- 交换机配置标志
- Miss发送长度
- 表示发送至控制器的新的流的最大8位字节，默认值128。

#### 步骤 E

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.7 修改流（Flow-Modification）事件

这一节将阐释修改流消息的触发条件和过程。图 3.7 展示的是一个独立的修改流事件。图中的标识标注了不同的触发条件和过程。

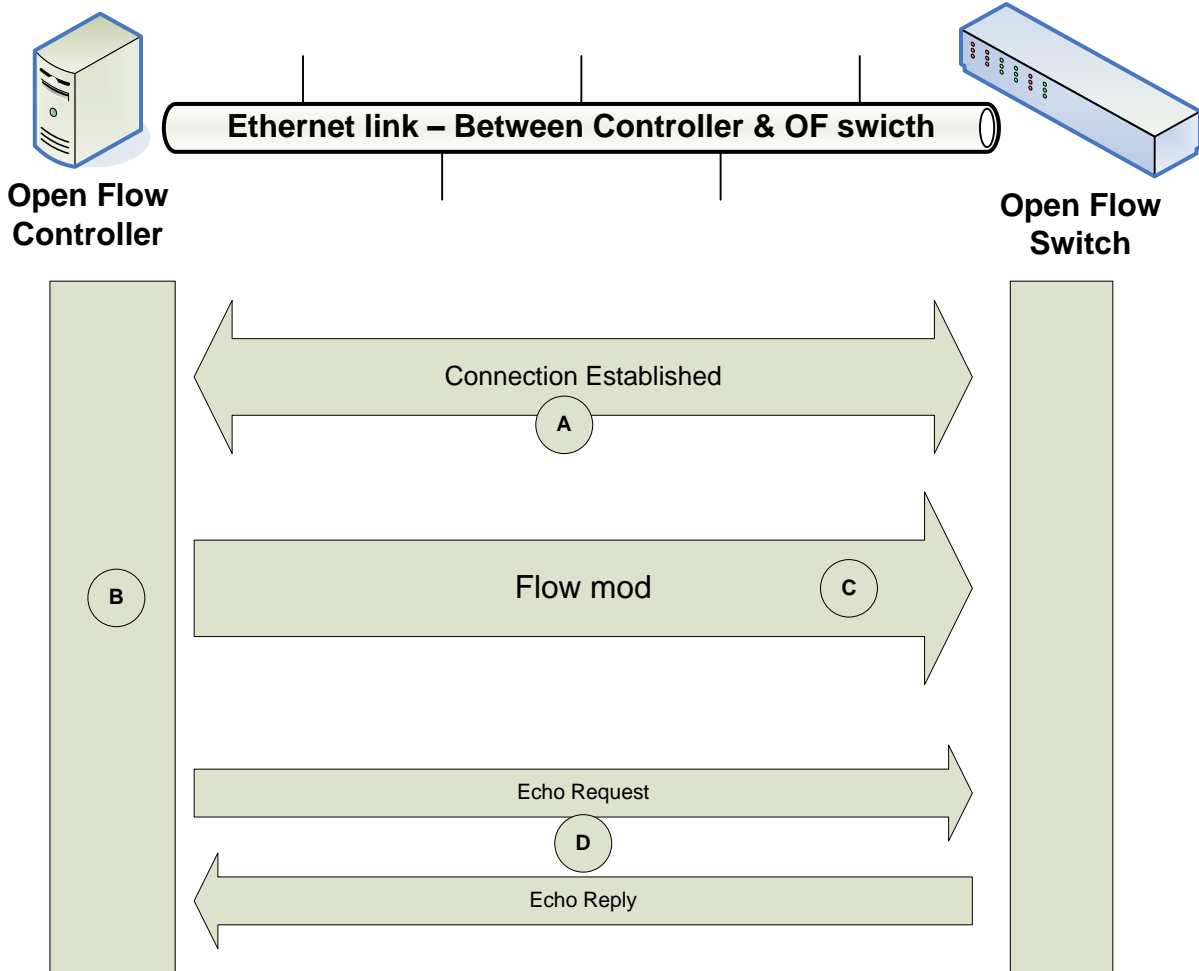


图 3.7 修改流（Flow-Modification）事件

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是修改流事件发生的前提。

#### 步骤 B

当控制器需要增添、修改或删除交换机中流表的时候，触发该事件。

#### 步骤 C

修改流（Flow-Modification）事件包括下面这些信息：

- 流匹配描述符
  - 通常是12个元组
- 命令
  - 0 增加新流

- 1 修改所有已匹配流
- 2 修改与通配符掩码匹配的条目
- 3 删除所有已匹配流
- 4 删除与优先权和通配符掩码匹配的流

- 软超时
- 硬超时
- 优先权
- 数据包缓冲ID
- 出口端口号

I. 值0xffff (无)意味着这些域没有意义，可以抛弃。

II. 端口0x0000–0xff00表示交换机端口（物理或逻辑定义端口）；对在序列号码内的端口进行指令操作

III. 如果端口不在0x0000–0xff00序列内，就意味着端口属于虚拟端口

表3.1 不同类型的端口号

范围	说明	类型
0x0000	预留	预留
0x0001-0xff00	交换机端口	物理端口
0xffff8	入端口	In-Port
0xffff9	流表	Table
0xffffa	常规二层转发	Normal
0xffffb	洪泛	Flood
0xffffc	所有端口	All
0xffffd	控制器	Controller
0xffffe	本地非OpenFlow端口	Local
0xfffff	无效	None

表 3.1

- 标志
  - 发送流移除

- 检查重复
- 紧急流
- 动作列表（添加为动作描述符）
  - 输出动作描述符
  - VLAN VID 动作描述符
  - VLAN PCP 动作描述符
  - 提取VLAN标签动作描述符
  - 以太网地址动作描述符
  - IPv4地址动作描述符
  - IPv4 DSCP动作描述符
  - TCP/UDP 端口动作描述符
  - 队列动作描述符
  - 各厂商Action描述符

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.8 流移除（Flow-Removed）事件

这一节将阐释了流移除（Flow-Removed）消息的触发条件和过程。图 3.8 展示的是一个独立的移除流事件。图中的标识标注了不同的触发条件和过程。

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是移除流（Flow-Removed）事件发生的前提。

#### 步骤 B

控制器试图删除交换机中流表的时候，就会触发该事件形成数据包。只有控制器触发移除流（Flow-Removed）事件时候，这一步才有必要。

#### 步骤 C

如果删除流的操作是由控制器触发（即，如果步骤 B 正确），那么只会发送删除流的数据包。该数据包类似其他修改流数据包（在前面已经阐述），但是通常该数据包还含有另外一个指令，标识着对于该数据包的修改操作是进行删除。

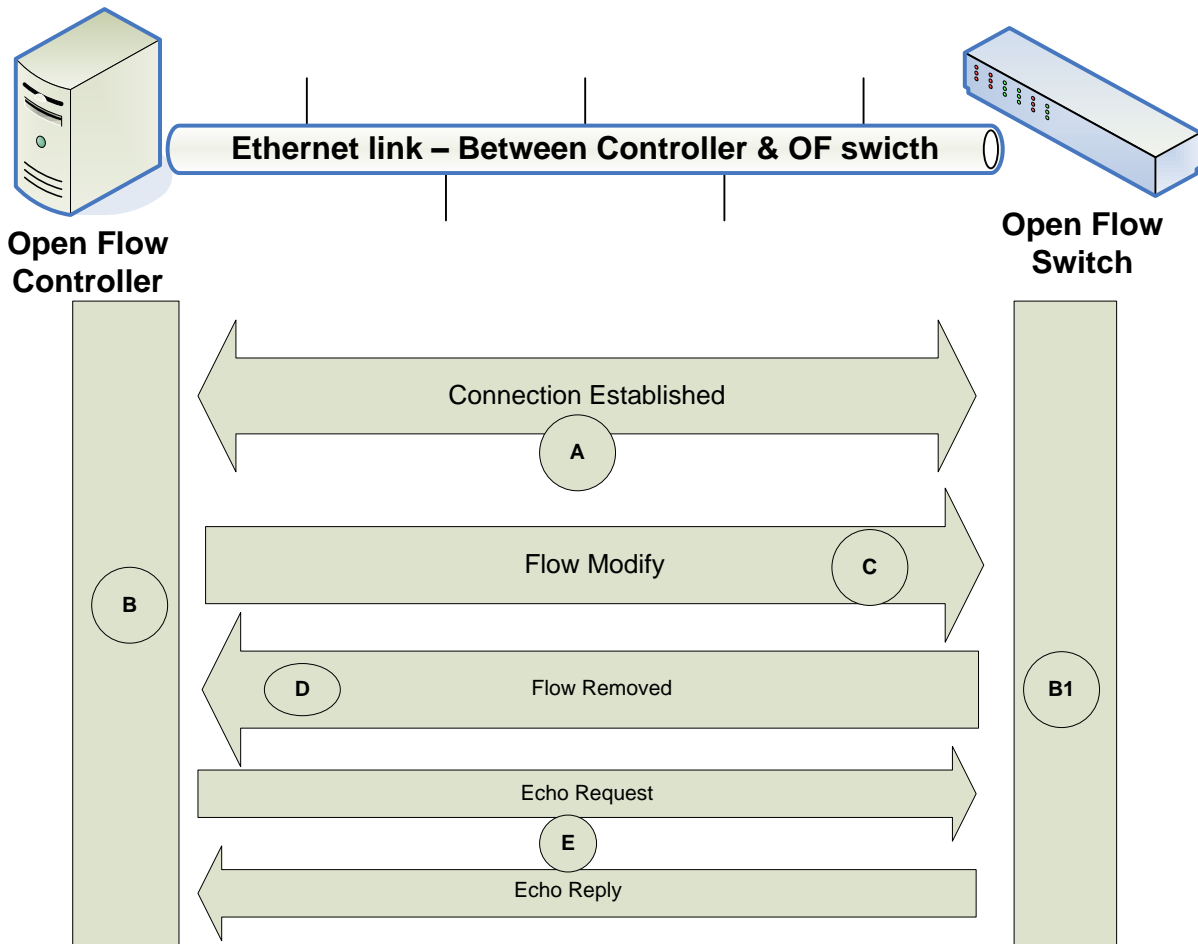


图 3.8 流移除（Flow-Removed）事件

#### 步骤 B1

如果流的删除是因为硬件超时（Hard time-out）或者空闲超时（Idle time-out），并且发送流移除标识是针对流条目，那么这一步骤就是必要的。

#### 步骤 D

无论删除流的请求是怎样触发的（步骤 B 或者步骤 B1），在步骤 D 中，移除流（Flow-Removed）消息被发送至控制器。这是选择性的，取决于交换机中发送流移除（Flow-Removed）消息的标识是否被设置。如果设置，交换机就会发送移除流（Flow-Removed）消息给控制器，用来通知流表的删除。

该事件数据包包括：

- 流匹配描述符
  - 不同的流匹配种类
- 优先权

- 对于移除流数据包来说，该项内容仅仅是提供信息，并不在移除流事件中发挥作用。但是，在常规的流表中，优先权的数值越高，流的优先权就越大
- 原因
  - 0 流的空闲超时超过了软超时。
  - 1 时间超过了硬超时。
  - 2 被修改流的删除操作剔除。
- 寿命持续时间（秒）
  - 流在交换机中的存活持续时间（秒）
- 寿命持续时间（毫微秒）
  - 流在交换机中的存活持续时间（毫微秒）
- 软超时
  - 初始流修改中得出的软超时
- 发送数据包数量
  - 命中流条目的数据包数量
- 发送字节数量
  - 命中流条目的字节数量

#### 步骤 E

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.9 端口修改（Port-Modify）事件

这节将阐述修改端口（Port-Modify）消息的触发条件和过程。图 3.9 展示的是一个独立的修改端口（Port-Modify）事件。图中的标识标注了不同的触发条件和过程。

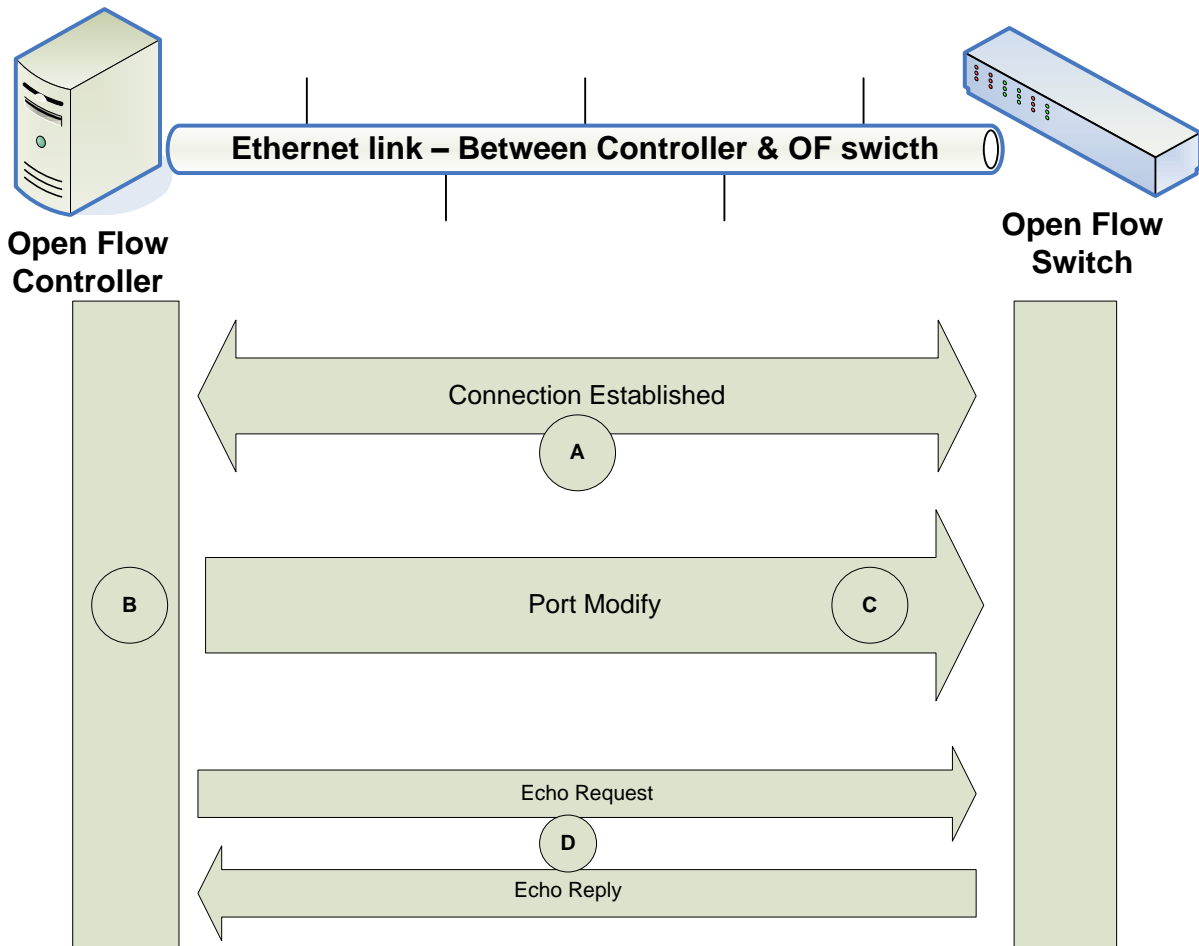


图 3.9 端口修改（Port-Modify）事件

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是修改端口（Port-Modify）事件发生的前提。

#### 步骤 B

控制器（管理员）试图修改端口配置标志——“admin down,” “no STP,” “no receive,” “no receive STP,” “no flood,” “no FWD,” “no packet-in” 的时候，会触发该事件。

#### Step C

#### 步骤 C

The port modification message has the following fields in the packet, and, using these fields, the port properties can be modified:



修改端口（Port-Modify）消息包括以下这些内容，通过这些内容，可以对端口的性能做出调整：

- Port number
- 端口号
- Ethernet address
- 以太网地址
- Port configuration flags
- 端口配置标志
- Port configuration flags mask
- 端口配置标志掩码
- Advertisement (port feature flags)
- 通告（端口特征标志）

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.10 统计(Stats)请求和响应事件

本节将阐述统计请求和响应消息的触发条件和过程。图 3.10 展示的是一个独立的统计请求和响应事件。图中的标识标注了不同的触发条件和过程。

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是统计请求和响应事件发生的前提。

#### 步骤 B

当控制器试图从交换机处获得不同类型的统计数据信息时，该事件被触发。

#### 步骤 D

该统计数据包包括：

- 类型
  - 0 OpenFlow交换机描述符
  - 1 单个流消息
  - 2 聚合流消息
  - 3 流表统计信息
  - 4 端口统计信息
  - 5 队列统计信息

- 65535 各厂商扩展
- 标志
  - 值应为零
- 状态主体
  - 不同类型值具有不同主体

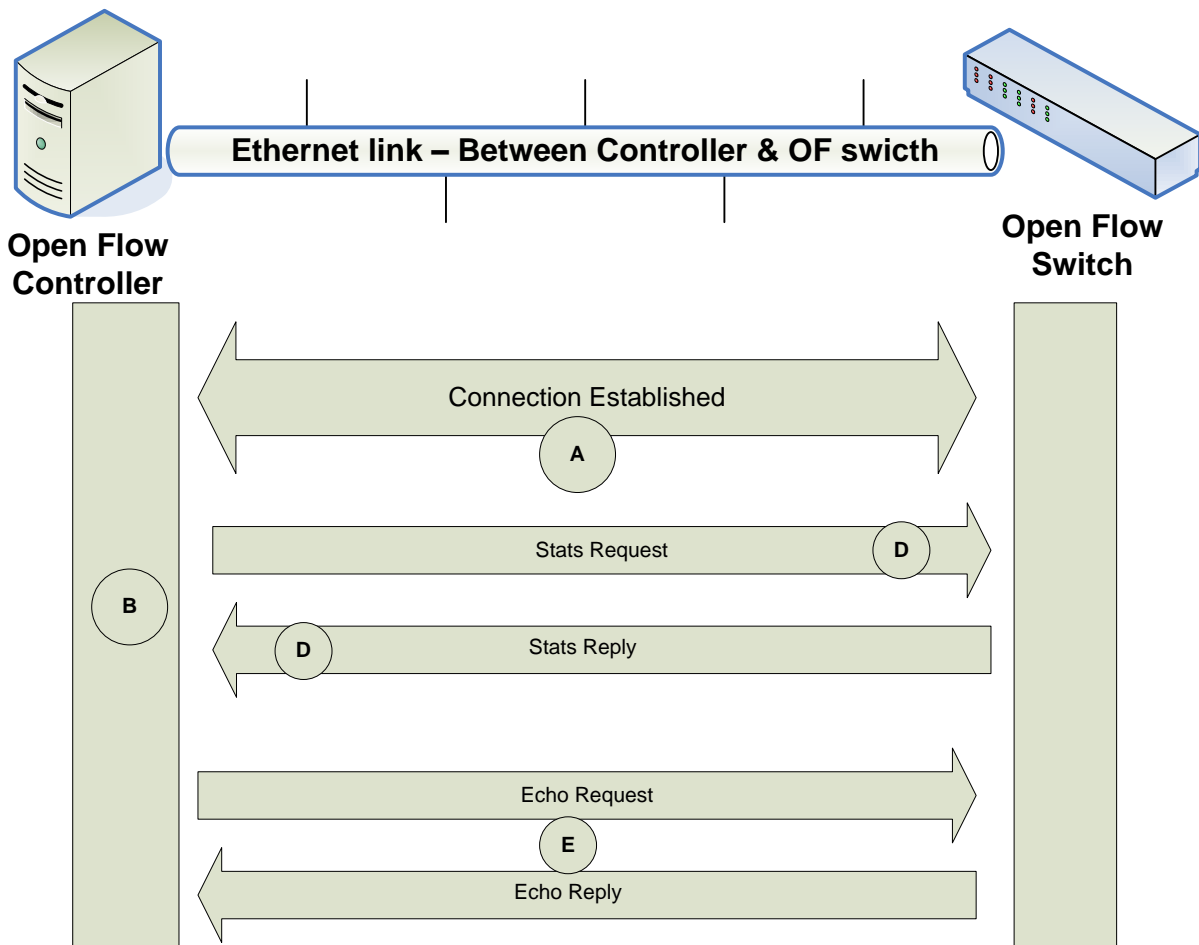


图 3.10 统计请求和响应事件

#### 步骤 E

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.11 Barrier 请求和响应事件

本节将阐述 Barrier 请求和响应消息的触发条件和过程。图 3.11 展示的是一个独立的 Barrier 请求和响应事件。图中的标识标注了不同的触发条件和过程。

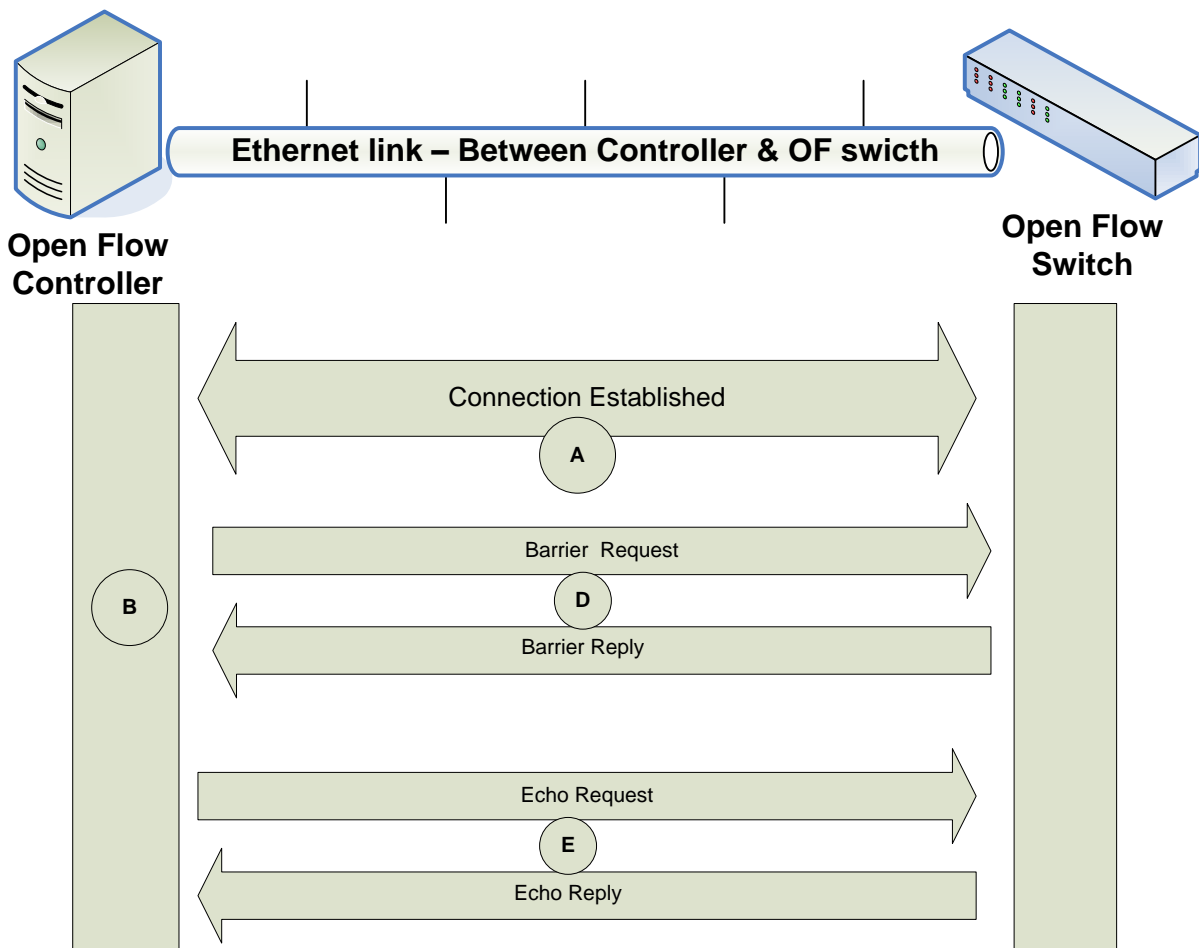


图 3.11 Barrier 请求和响应

#### Step A

##### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是 Barrier 请求和响应事件发生的前提。

#### **步骤 B**

当控制器试图了解其分配给 OpenFlow 交换机的任务是否完成或将在何时完成的时候，该事件将被触发。

#### **步骤 D**

Barrier 请求消息用 OpenFlow 数据头消息“类型值=19”表示。

收到请求消息的交换机，在完成控制器分配的任务后，会发送响应消息至控制器。

障碍响应消息用 OpenFlow 数据头消息“类型值=20”表示，并附有 Barrier 请求消息的交换标识。

#### **步骤 E**

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### **3.12 队列获取配置（Queue Get Configuration）请求和响应事件**

本节将阐述队列配置请求和响应消息的触发条件和过程。图 3.11 展示的是一个独立的队列获取配置请求和响应事件。图中的标识标注了不同的触发条件和过程。

#### **步骤 A**

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是队列获取配置请求和响应事件发生的前提。

#### **步骤 B**

当控制器试图询问 OpenFlow 交换机端口的队列配置时候，触发该事件。

#### **步骤 C**

队列请求包括所请求队列信息的端口号。队列配置响应消息包括端口号和该端口的队列配置信息。

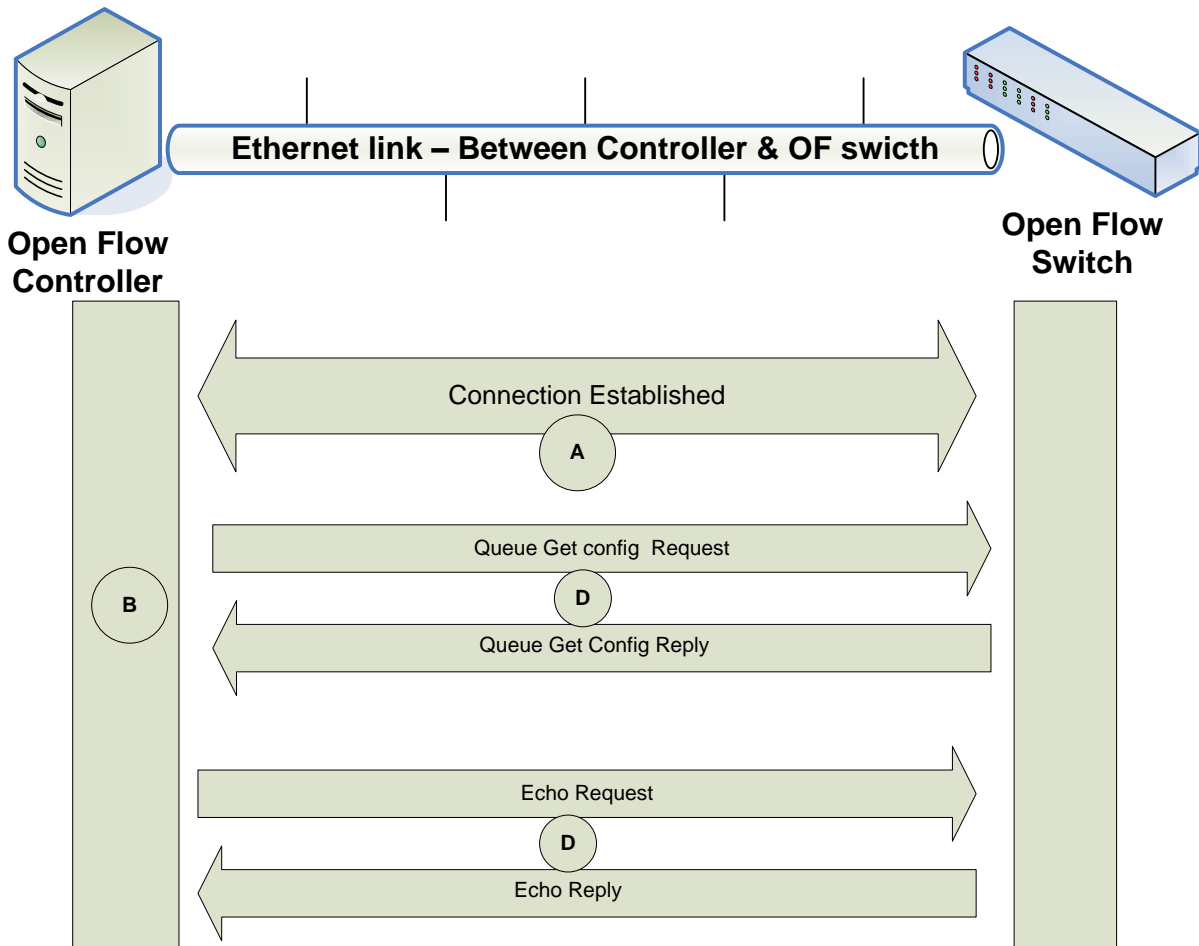


图 3.12 队列获取配置请求和响应

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

### 3.13 错误事件

这节将阐述错误消息的触发条件和过程。图 3.13 展示的是一个独立的错误事件。图中的标识标注了不同的触发条件和过程。

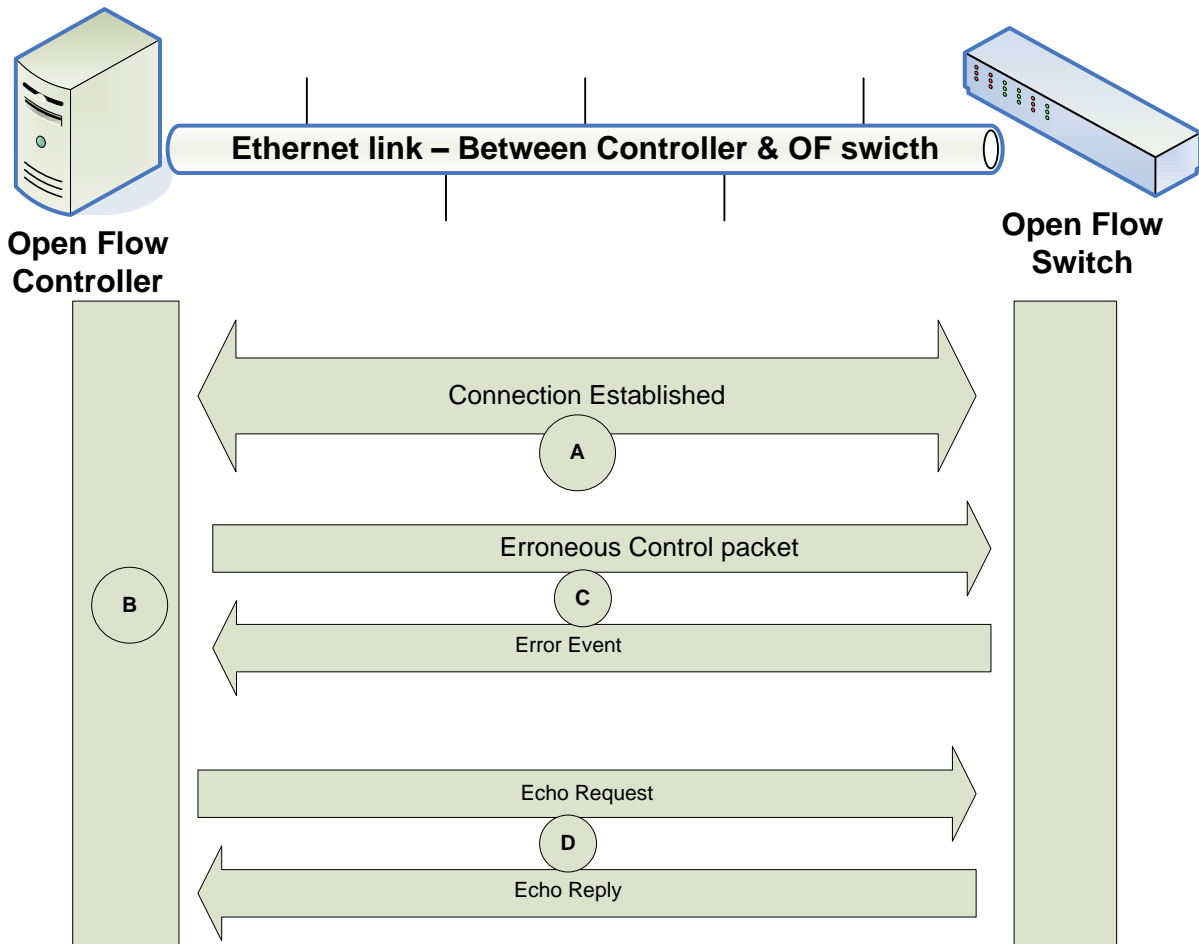


图 3.13 错误事件

#### 步骤 A

控制器和交换机之间的连接经过 TCP 建立、Hello 报文、功能请求与响应环节后建立。这些连接的存在是错误事件发生的前提。当然，错误也可以发生在 hello 报文中。

#### 步骤 B

当控制器发送的数据包不能被读出或支持，或者交换机不能执行的时候，就产生了错误事件。所以任何发送至交换机的控制数据包都可能触发该事件。

#### 步骤 C

如果 OpenFlow 交换机不能读出、支持或者执行控制器发出的 OpenFlow 控制数据包，就会发送错误数据包至控制器，说明错误原因。数据包中包含说明错误信息的类型值或者是代码值。

#### 步骤 D

Echo 请求和响应用于保持 OpenFlow 控制器与 OpenFlow 交换机之间的连接状态。

## 第四章 OpenFlow 案例研究

本章主要讲述了一个简单的 OpenFlow 组网，以研究起作用的基本协定。组网中有三个交换机呈三角形连接，每个交换机在管理网络上都可以通过 OpenFlow 控制器来控制。三个交换机之间的链路的开销不同，所以我们可以很容易地分析出 OpenFlow 是怎样计算出最短路径的；在出现故障时，流转发链路又是如何重新选择的。为了对其做出更好的解释，还使用了一些 wire-shark 网络抓包程序，并使用相关信息对其进行了标注。

### 4.1 拓扑结构的发现

对于基于 OpenFlow 的网络，第一步是发现 OpenFlow 控制器控制的 OpenFlow 交换机。控制器需要首先了解拓扑，才能在 OpenFlow 网络中实际寻找到两个不同主机之间的路径，并且为数据平面安装流表。

OpenFlow 控制器通过监听 TCP 端口号 6633 来检测 OpenFlow 交换机。一旦 OpenFlow 控制器连接到 OpenFlow 交换机，下一步就是要发现网络的总体视图（即获取单个 OpenFlow 交换机的细节，以及不同 OpenFlow 交换机之间连接的实际链路和端口）。这一发现通过两个步骤完成：第一步是连接单个交换机，第二步是检测这些交换机之间的链路。

第一步由特性请求和特性响应机制维护。一旦 TCP 信号交换完成，控制器就会发送一个特性请求消息。新连接的交换机就会通过一个特性响应消息进行响应。这个特性响应消息会将交换机的功能、端口细节及活动能力告知控制器。在第二步中，交换机间链路的发现，需通过不同 OpenFlow 交换机之间发送的链路层发现协议（LLDP）报文来实现。

LLDP 报文如何发送？在 OpenFlow 协议中没有定义 LLDP，Openflow 交换也不能识别 LLDP 报文（只当普通报文处理），所以所有的 LLDP 报文实际上是由控制器生成，并封装在 Packet-Out 报文中，然后由控制器发送 Packet-Out 报文给交换机，Packet-Out 报文中动作（Action）字段为：转发到交换机的指定端口。这些 LLDP 报文将从交换机的各个端口转发出去，在对端交换机将会收到传入的 LLDP 数据包，交换机并不能识别这个报文为 LLDP 协议报文，只当普通数据报文处理，这个 LLDP 报文由交换机封装入 Packet-In 报文发送到控制器中（Packet-In 报文中会携带入端口信息）。当控制器收到从交换机发来的 Packet-In 报文（封装了 LLDP 协议报文），交换机就可以了解整个网络的拓扑。（LLDP 发送的频率是十秒一次。）

数据包发起自：	数据包发向：	数据包类型	数据包有效载荷及细节

控制器	所有交换机	Feature 请求	控制器向所有交换机发送 Features 请求报文
所有交换机	控制器	Feature 响应	所有交换机发送 Feature 响应消息（携带交换机支持的所有功能）
控制器	交换机-A	Packet-out	控制器要求交换机-A 向端口 32、52、41 发送 LLDP 数据包
控制器	交换机-B	Packet-out	控制器要求交换机-B 向端口 30、50、41 发送 LLDP 数据包
控制器	交换机-C	Packet-out	控制器要求交换机-C 向端口 42、41 发送 LLDP 数据包
交换机-A	控制器	Packet-in	交换机-A 封装从端口 52 上收到来自交换机-B 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器
交换机-A	控制器	Packet-in	交换机-A 封装从端口 41 上收到来自交换机-C 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器
交换机-B	控制器	Packet-in	交换机-B 封装从端口 50 上收到来自交换机-A 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器
交换机-B	控制器	Packet-in	交换机-B 封装从端口 41 上收到来自交换机-C 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器
交换机-C	控制器	Packet-in	交换机-C 封装从端口 41 上收到来自交换机-A 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器
交换机-C	控制器	Packet-in	交换机-C 封装从端口 42 上收到来自交换机-B 的 LLDP 数据包，并通过 Packet-In 报文发送给控制器



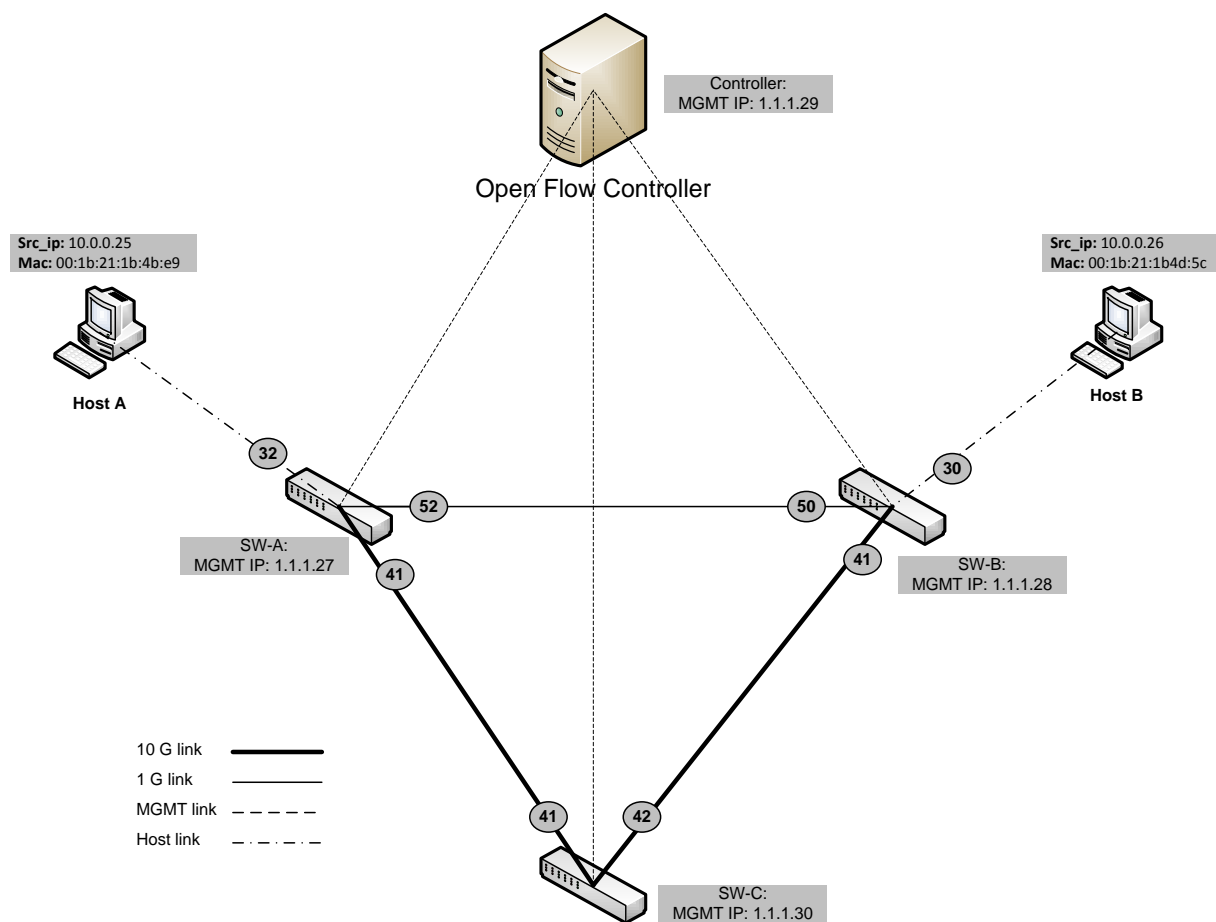


图 4.1.

根据上述步骤，控制器可以得到拓扑图如图 4.1。(此时控制器只是得了网络的拓扑，并不了解连接交换机的主机信息，如图 4.1 所示)。

整个拓扑发现的操作可以列在以下步骤中（对于图 4.1 中解释的拓扑发现）：

表 4.1

## 4.2 最短路径计算

当控制器完成了拓扑发现后，就可以计算出最短路径。这本书解释了基于Dijkstra的最短路径计算（同传统STP）。在端节点之间计算出最短路径后，控制会使用端口修改（Port-Modify）命令为OpenFlow交换机上的端口进行参数设置。这样做的目的是使交换机收到一个数据包时，它也只会泛洪至属于最短路径一部分的端口。

目前，该算法是基于链路的带宽。我们举一个例子来说明在现有拓扑中是如何实现最短路径计算的，如图4.1:

按照我们所举的例子，我们认为数据包需要在主机A和主机B之间发送，且有两条可以采取的路径。此外每条路径上链路的成本:

1. 交换机-A <-> 交换机-B = 1G
2. 交换机-A <-> 交换机-C <-> 交换机-B = 10G

基于最短路径算法:

1. 当网络发现完成时，控制器将通过向交换机-A、交换机-B、交换机-C 发送端口改变（Port-Modify）消息，用来设置一个无泛洪的端口标志：
  - a. 交换机-A 将会在端口（32、52、41）上收到“端口修改（Port-Modify）”消息——在所有端口上设置“无泛洪”的标志
  - b. 交换机-B 将会在端口（30、50、41）上收到“端口修改（Port-Modify）”消息——在所有端口上设置“无泛洪”的标志
  - c. 交换机-C 将会在端口（41、42）上收到“端口修改（Port-Modify）”消息——在所有端口上设置“无泛洪”的标志
2. 一旦控制器使用 LLDP Packet-In/Out 消息发现拓扑，最短路径树将被计算出，每个交换机上的几个链路将通过使用端口修改（Port-Modify）包再次启用。
  - a. 交换机-A 将会在端口（32、41）上收到“端口修改（Port-Modify）”消息——将端口设到转发状态
  - b. 交换机-B 将会在端口（41、30）上收到“端口修改（Port-Modify）”消息——将端口设到转发状态
  - c. 交换机-C 将会在端口（41、42）上收到“端口修改（Port-Modify）”消息——将端口设到转发状态

图 4.2 控制器计算出的最短路径的步骤

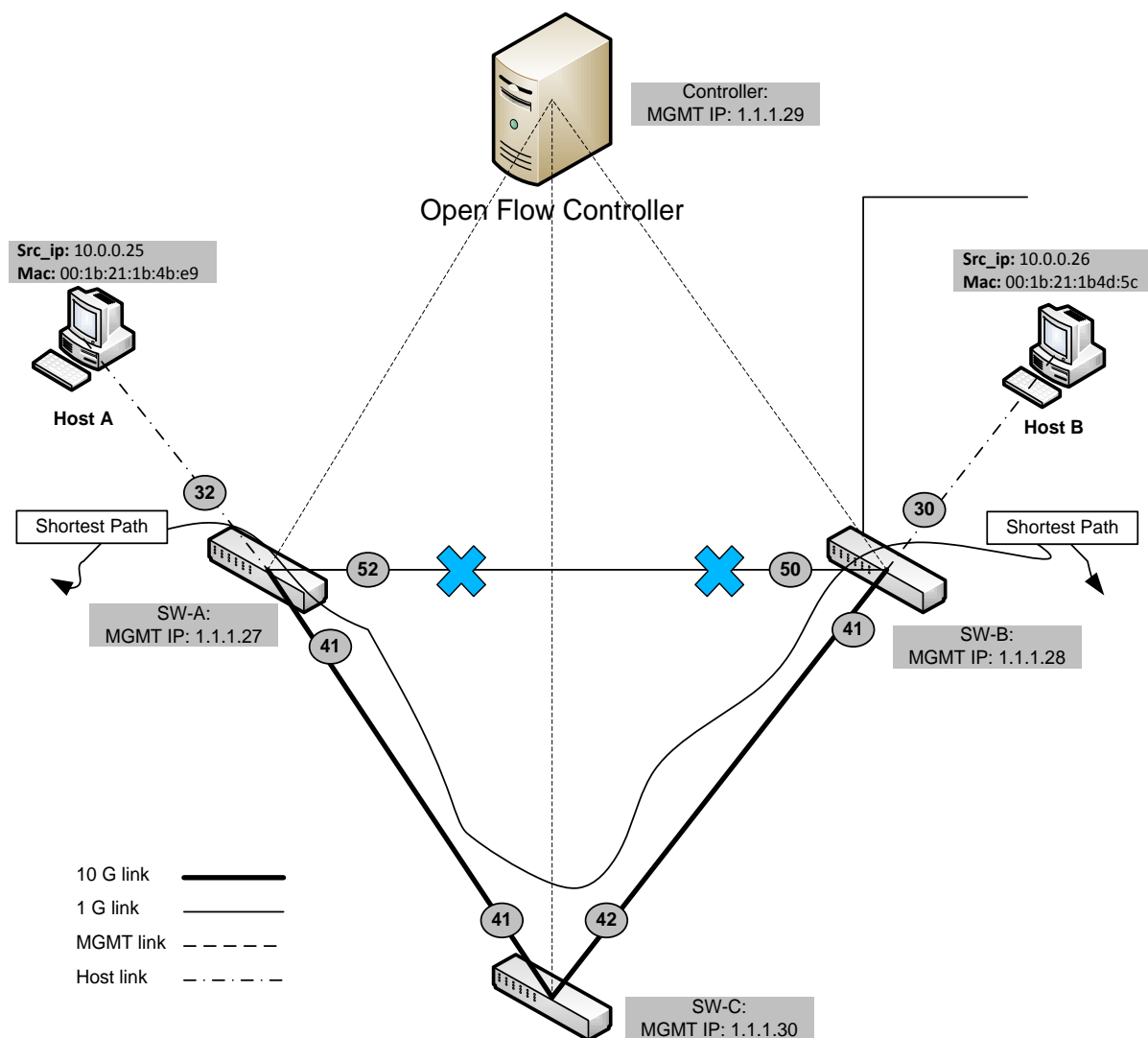


图 4.2.

如果交换机-A 和交换机-B 之间的链路速率改为 100G，那么端口状态标志将改变控制器处的拓扑图，将会重新计算最短路径，在改变后的拓扑基础上再次触发“端口修改（Port-Modify）”消息。

### 4.3 以 Ping 为例的流表操作

为了演示如何在交换机上安装流表，以下举一个 Ping 的例子，其中主机 A 将会发送 Ping 命令到主机 B。图 4.3 中显示了所有安装流表的操作，如图 4.2，由拓扑计算出了最短路径。

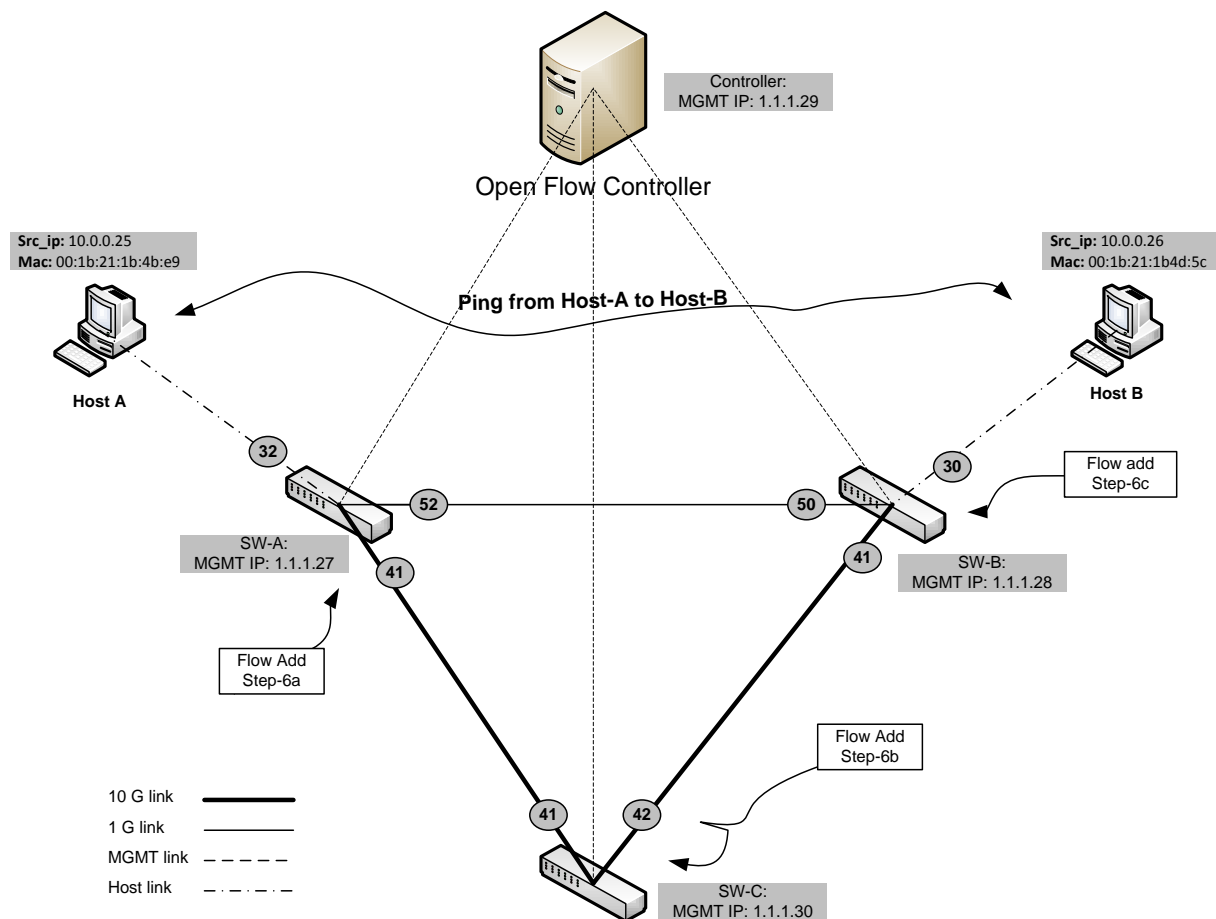


图 4.3.

流表安装操作见以下几个步骤。

#### 步骤 1

从主机 A ping 主机 B (10.0.0.26)，主机 A 将生成一个 ARP 请求, 发送至交换机 A。当交换机 A 收到 ARP 请求后，它会发送一个 Packet-In 数据包（其中封装了这个 ARP 请求）至控制器。

图 4.4 为 Wireshark 抓包后解析的 Packet-In 报文。

```

  ▸ Internet Protocol, Src: 1.1.1.27, Dst: 1.1.1.29
  ▸ Transmission Control Protocol, Src Port: 53706 (53706), Dst Port: 6633 (6633), Seq: 9, Ack: 531, Len: 78
  ▾ OpenFlow Protocol → ARP request encapsulated in OF Packet In
    ▸ Header
    ▾ Packet In
      Buffer ID: 4294967295
      Frame Total Length: 60
      Frame Recv Port: 32 → The incoming port for this ARP request
      Reason Sent: Action explicitly output to controller (1)
    ▾ Frame Data: FFFFFFFF001B211B4BE908060001080006040001001B...
      ▸ Ethernet II, Src: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
      ▾ Address Resolution Protocol (request) → Actual ARP Request
        Hardware type: Ethernet (0x0001)
        Protocol type: IP (0x0800)
        Hardware size: 6
        Protocol size: 4
        Opcode: request (0x0001)
        [Is gratuitous: False]
        Sender MAC address: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9)
        Sender IP address: 10.0.0.25 (10.0.0.25)
        Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
        Target IP address: 10.0.0.26 (10.0.0.26)

```

图 4.4.

## 步骤 2

控制器收到这个 Packet-In 报文后，将解封装数据包并查看到 ARP 请求。控制器会发送一个 Packet-Out 报文（封装原始 ARP 请求）至所有边缘交换机，动作字段（Action）为发送 ARP 请求至所有边缘端口。在拓扑中，唯一的边缘交换机为交换机-B（交换机-C 是一个中间传送交换机节点）。

控制器如何知道非边缘交换机？当 LLDP 交换发生时，LLDP 数据包被发送至交换机所有接通状态的端口。如果交换机没有收到发送至 up 状态的端口的 LLDP 报文，便可以知道这些 up 状态的端口没有连接其它的交换机，为边缘端口。

图 4.5 为 Wireshark 抓包后解析的 Packet-Out 报文

```

  ▸ Internet Protocol, Src: 1.1.1.29 , Dst: 1.1.1.28
  ▸ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 41101 (41101), Seq: 1, Ack: 351, Len: 84
  ▾ OpenFlow Protocol
    ▸ Header
    ▾ Packet Out
      Buffer ID: None
      Frame Recv Port: None (not associated with a physical port)
      Size of action array in bytes: 8
      ▾ Output Action(s)
        ▾ Action
          Type: Output to switch port (0)
          Len: 8
          Output port: 30
          Max Bytes to Send: 65535
          # of Actions: 1
      ▾ Frame Data: FFFFFFFF001B211B48E908060001080006040001001B...
        ▸ Ethernet II, Src: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
        ▸ Address Resolution Protocol (request)

```

图 4.5.

### 步骤 3

当交换机 B 收到 Packet-Out 报文后，解析报文中动作（Action）后，将原 ARP 请求转发给所有边缘端口，这时主机 B 将后收到原始的 ARP 请求报文，主机 B 则会回应一个 ARP Reply 报文。同样的，交换机 B 收到此报文后，发送 Packet-In 报文（封装了 ARP Reply 报文）至控制器。图 4.6 为 Wireshark 抓包后解析的 Packet-In 报文（封装了 ARP Reply 报文）

```

  ▸ Internet Protocol, Src: 1.1.1.28 , Dst: 1.1.1.29
  ▸ Transmission Control Protocol, Src Port: 41101 (41101), Dst Port: 6633 (6633), Seq: 351, Ack: 85, Len: 78
  ▾ OpenFlow Protocol
    ▸ Header
    ▾ Packet In
      Buffer ID: 4294967295
      Frame Total Length: 60
      Frame Recv Port: 30
      Reason Sent: Action explicitly output to controller (1)
      ▾ Frame Data: 001B211B48E9001B211B4D5C08060001080006040002001B...
        ▸ Ethernet II, Src: IntelCor_1b:4d:5c (00:1b:21:1b:4d:5c), Dst: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9)
        ▾ Address Resolution Protocol (reply)
          Hardware type: Ethernet (0x0001)
          Protocol type: IP (0x0800)
          Hardware size: 6
          Protocol size: 4
          Opcode: reply (0x0002)
          [Is gratuitous: False]
          Sender MAC address: IntelCor_1b:4d:5c (00:1b:21:1b:4d:5c)
          Sender IP address: 10.0.0.26 (10.0.0.26)
          Target MAC address: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9)
          Target IP address: 10.0.0.25 (10.0.0.25)

```

图 4.6.

#### 步骤 4

同样的，控制器将主机 B 回应的 ARP Reply 消息封装在 Packet-Out 报文中发送给交换机 A，交换机 A 解释 Packet-Out 报文,执行动作（Action）字段中的操作，将 ARP Reply 报文发送给主机 A。主机 A 则会收到主机 B 发送的 ARP Reply 报文，完成主机 B 的 ARP 表项安装。图 4.7 Wireshark 抓包后解析的 Packet-Out 报文（封装了 ARP Reply 报文）

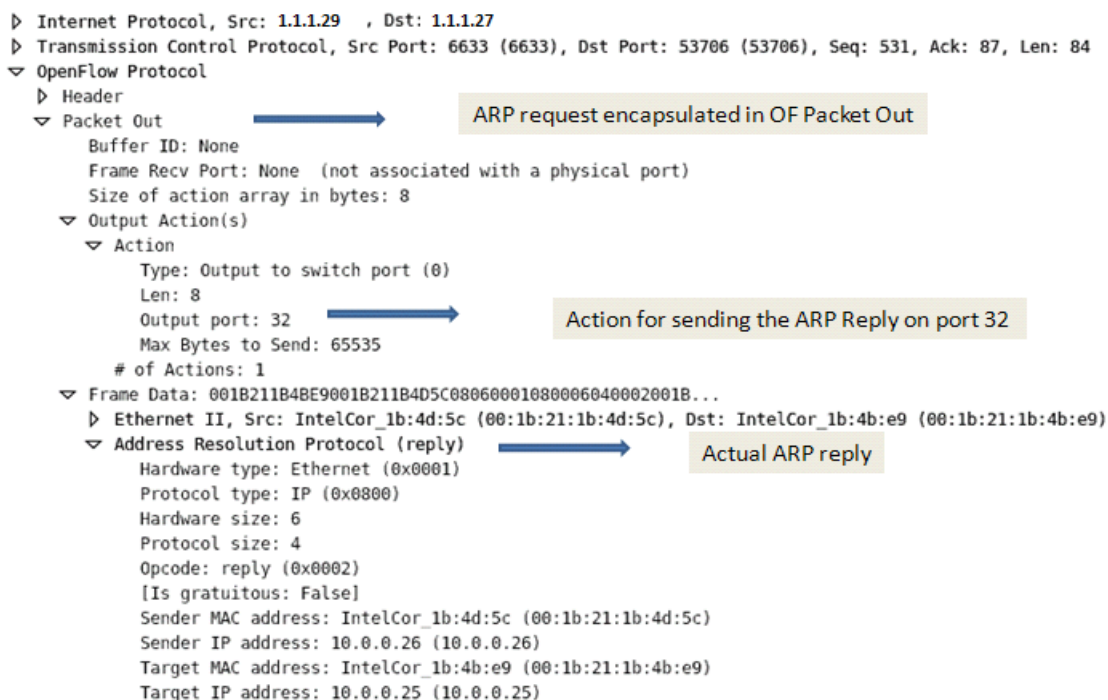


图 4.7.

#### 步骤 5

现在主机 A 的 ARP 表项中有了主机 B 的 MAC 和 IP 信息，它将发送 ICMP 请求数据包。当这个 ICMP 数据包到达交换机 A 时，它将被封装在 Packet-Out 报文发送给控制器。图 4.8 为 Wireshark 抓包后解析的 Packet-In 报文（封装了 ICMP 请求报文）

```

  ▸ Internet Protocol, Src: 1.1.1.27 , Dst: 1.1.1.29
  ▸ Transmission Control Protocol, Src Port: 53706 (53706), Dst Port: 6633 (6633), Seq: 87, Ack: 615, Len: 92
  ▾ OpenFlow Protocol
    ▸ Header
    ▾ Packet In
      Buffer ID: 4294967295
      Frame Total Length: 74
      Frame Recv Port: 32
      Reason Sent: Action explicitly output to controller (1)
      ▾ Frame Data: 001B211B4D5C001B211B4BE908004500003C005600008001...
        ▸ Ethernet II, Src: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9), Dst: IntelCor_1b:4d:5c (00:1b:21:1b:4d:5c)
        ▸ Internet Protocol, Src: 10.0.0.25 (10.0.0.25), Dst: 10.0.0.26 (10.0.0.26)
        ▸ Internet Control Message Protocol

```

图 4.8.

#### 步骤 6（6a、6b 及 6c）

步骤 5 当控制器收到主机发送的 ARP Request 报文主机 A 发送 ICMP 请求数据包时，同时控制器也开始安装流（现在它从 ARP 中得到了 MAC 地址）。流修改数据包由控制器发送至交换机 A、B、C，将流双向地安装在每个交换机上。此为交换机 A 上流修改的例子（类似的数据包也会由控制器发送，在交换机 B、C 上安装条目）。

```

  ▸ Internet Protocol, Src: 1.1.1.29 , Dst: 1.1.1.27
  ▸ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 53706 (53706), Seq: 615, Ack: 179, Len: 80
  ▾ OpenFlow Protocol
    ▸ Header
    ▾ Flow Modification
      ▾ Match
        ▸ Match Types
          Input Port: 41
          Ethernet Src Addr: IntelCor_1b:4d:5c (00:1b:21:1b:4d:5c)
          Ethernet Dst Addr: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9)
          Input VLAN ID: 65535
          IP Src Addr: 0.0.0.0 (0.0.0.0)
          IP Dst Addr: 0.0.0.0 (0.0.0.0)
          Cookie: 0xc00261a80002c4b8
          Command: Modify entry strictly matching wildcards (2)
          Idle Time (sec) Before Discarding: 300
          Max Time (sec) Before Discarding: 0
          Priority: 25000
          Buffer ID: None
          Out Port (delete* only): 0
        ▸ Flags
      ▾ Output Action(s)
        ▾ Action
          Type: Output to switch port (0)
          Len: 8
          Output port: 32
          Max Bytes to Send: 0
          # of Actions: 1

```



图 4.9.

#### 步骤 7

交换机 A 发送至控制器的封装有 ICMP 请求的包入指令，将被封装入包出指令中而送至交换机 B（由控制器发送）。

#### 步骤 8

基于在步骤 6a、6b、6c 中安装的流，ICMP 数据包的响应将由交换机 B 被发送至交换机 A。（注意，如 4.2 节中解释，交换机 A 和交换机 B 之间的最短路径将通过交换机 C。）

### 4.4 以故障切换为例的流表修改

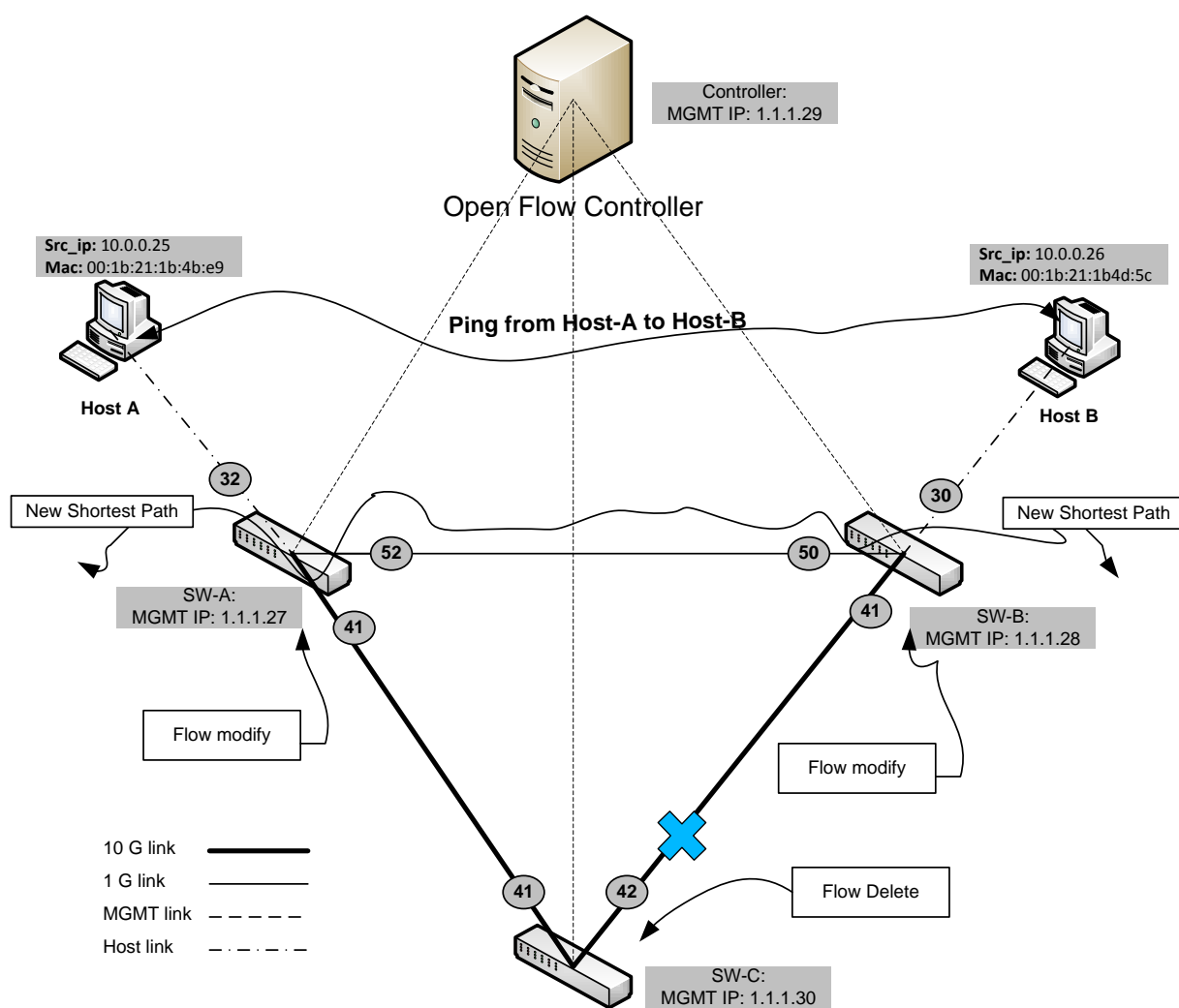


图 4.10.

如图 4.2，最短路径是基于更好的带宽（交换机-A <-> 交换机-B <-> 交换机-C）。对于故障切换，当交换机 C 和交换机 B 之间的链路发生故障（图 4.10），迫使流表修改。新流表将被修改并安装在交换机网络中，如以下步骤所示。

## Step 1

### 步骤 1

当交换机 C 和交换机 B 之间的链路发生故障，交换机 C 会发送端口状态消息至控制器，通告交换机 C 与交换机 B 相连链路状态变为 down；同样地，交换机 B 也会向控制器通告链路状态改变。图 4.11 显示了该消息的详细内容。

```

> Internet Protocol, Src: 1.1.1.30 , Dst: 1.1.1.29
> Transmission Control Protocol, Src Port: 44685 (44685), Dst Port: 6633 (6633), Seq: 9, Ack: 9, Len: 64
▼ OpenFlow Protocol
  > Header
  ▼ Port Status
    Reason: Some attribute of the port has changed (2)
    ▼ Physical Port
      Port #: 42
      MAC Address: BladeNet_6f:33:00 (08:17:f4:6f:33:00)
      Port Name: 42
      > Port Config Flags
      ▼ Port State Flags
        .... 1 = No physical link present: 1
        STP state: Learning and relaying frames
      > Port Current Flags
      > Port Advertised Flags
      > Port Supported Flags
      > Port Peer Flags
```

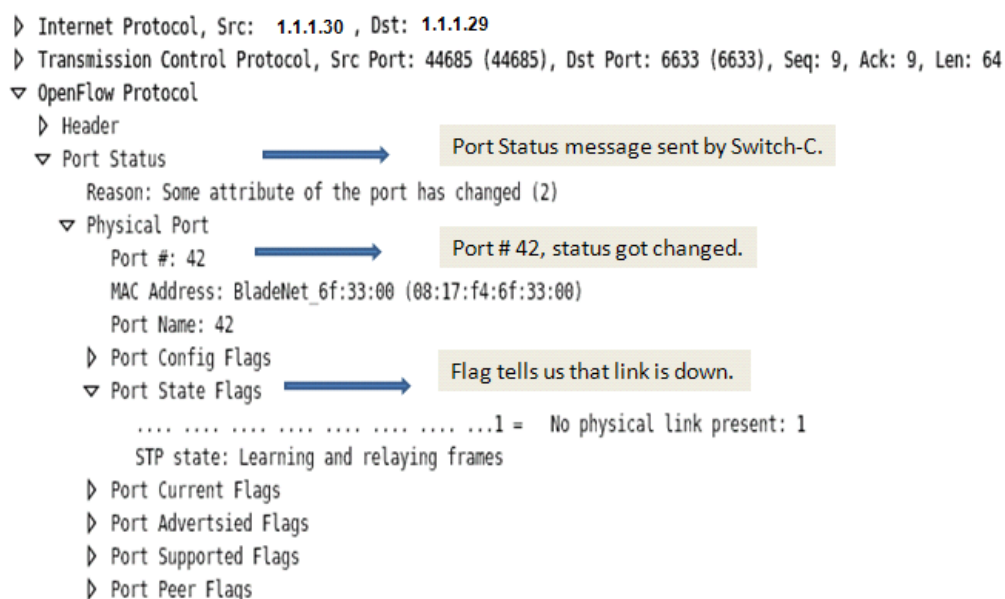


图 4.11.

### 步骤 2

当控制器收到链路 down 消息后，再次计算最短路径，随后会发送流修改的信息来安装一个新的路径。（根据拓扑，新路径将会切换到交换机 A 和交换机 B 之间的 GE 链路。）

图 4.12 显示了流修改消息的详细内容。

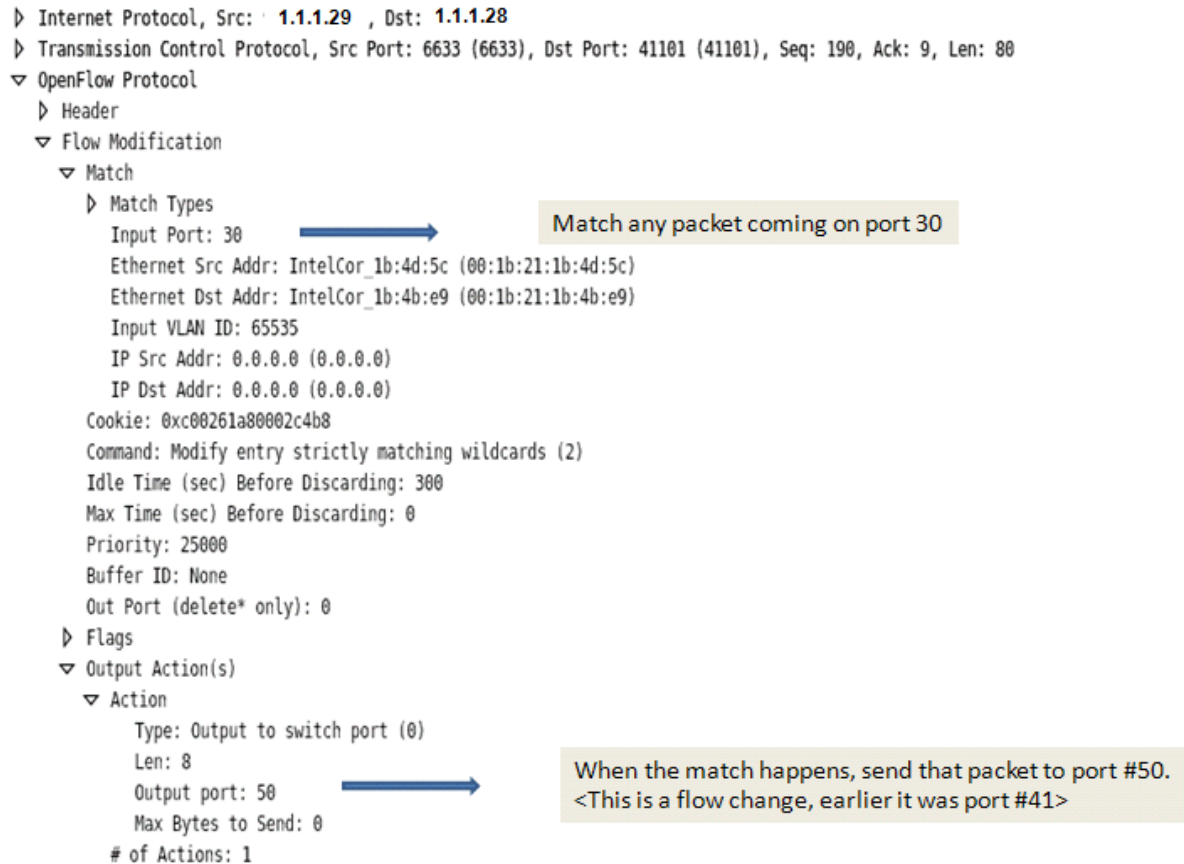


图 4.12.

### 步骤 3

现在交换机 C 和 B 之间没有任何可用链路，交换机 C 上的转到到交换机 B 的流表将被删除，以下是用于流修改的报文。

图 4.13 显示了流删除报文的详细内容。（同样，交换机 A 和 B 上的流也将被修改）

```

    ▸ Internet Protocol, Src: 1.1.1.29 , Dst: 1.1.1.30
    ▸ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 44685 (44685), Seq: 9, Ack: 73, Len: 72
    ▾ OpenFlow Protocol
      ▸ Header
        ▾ Flow Modification
          ▾ Match
            ▸ Match Types
              Input Port: 42
              Ethernet Src Addr: IntelCor_1b:4d:5c (00:1b:21:1b:4d:5c)
              Ethernet Dst Addr: IntelCor_1b:4b:e9 (00:1b:21:1b:4b:e9)
              Input VLAN ID: 65535
              IP Src Addr: 0.0.0.0 (0.0.0.0)
              IP Dst Addr: 0.0.0.0 (0.0.0.0)
              Cookie: 0xc00261a80002c4b8
              Command: Delete entry strictly matching wildcards and priority (4)
              Idle Time (sec) Before Discarding: 300
              Max Time (sec) Before Discarding: 0
              Priority: 25000
              Buffer ID: None
              Out Port (delete* only): None (not associated with a physical port)
            ▾ Flags
              .... 1 = Send flow removed: Yes (1)
              .... 0 = Check for overlap before adding flow: No (0)
              .... 0 = Install flow into emergency flow table: No (0)
            ▾ Output Action(s)
              Warning: No actions were specified

```

Match entry with input port 40 (port which is now down)

Command to delete the flow

图 4.13.

4.1 节至 4.4 节解释的步骤可以应用于任何大型的部署，可以用来了解基于 OpenFlow 的控制器是怎样在物理链路故障情况下的工作原理。

这里要说明的是，故障切换的性能将主要受下面几个因素影响；

1. 控制器处理能力；
2. 交换机与控制器之间网络传送延迟；
3. 交换机处理 Openflow 协议报文的能力；
4. 交换机上流表的数量；

## 第五章 OpenFlow 数据包详细说明

本章详细描述了 OpenFlow 报文的类型。每个报文中都细化到 byte 和 bit 级别的信息。与本书其它章节相似，这些数据包都基于 OpenFlow standards 1.0 规范。

### 5.1 OpenFlow 数据包概要

#### 控制器-至-交换机

- 控制器-至-交换机的消息由控制器生成，不一定需要交换机的响应。

#### Asynchronous

##### 异步

- 在未获取控制器触发这些消息情况下，交换机即发送这些消息至控制器。交换机向控制器发送异步消息来确认报文收到、交换机状态的改变或发生的错误信息。

#### Symmetric

##### 对称

- 在未收到对端触发的情况,交换机或控制器互相主动发送一些消息。

### 5.2 数据包概述

下表是规范 1.0 数据包类型中给出和归纳的所有 OpenFlow 数据包简况。第一列解释了包的类型，第二列解释了该包基于规范的分类型，第三列给出了特定类型的数据包在 OpenFlow 数据头中的类型值。基于这些数据包实现的功能，数据包进一步的分组已经完成。

OpenFlow 数据包类型	数据包分类	通用数据头中的类型值
对称消息		
OpenFlow hello	对称消息	0
OpenFlow error	对称消息	1
OpenFlow echo request	对称消息	2
OpenFlow echo reply	对称消息	3

OpenFlow vendor message	对称消息	4
<b>交换机配置消息</b>		
OpenFlow Feature 请求	控制器-至-交换机消息	5
OpenFlow Feature 响应	控制器-至-交换机消息	6
OpenFlow get configuration 请求	控制器-至-交换机消息	7
OpenFlow get configuration 响应	控制器-至-交换机消息	8
OpenFlow set configuration	控制器-至-交换机消息	9
<b>异步消息</b>		
OpenFlow packet-in	异步消息	10
OpenFlow flow removed	异步消息	11
OpenFlow port status	异步消息	12
<b>控制器指令消息</b>		
OpenFlow packet-out	控制器-至-交换机消息	13
OpenFlow flow modification	控制器-至-交换机消息	14
OpenFlow port modification	控制器-至-交换机消息	15
<b>统计消息</b>		
OpenFlow stats 请求	控制器-至-交换机消息	16
OpenFlow stats 响应	控制器-至-交换机消息	17
<b>Barrier 消息</b>		
OpenFlow barrier 请求	控制器-至-交换机消息	18
OpenFlow barrier 响应	控制器-至-交换机消息	19
<b>队列配置消息</b>		
Queue get configuration 请求	控制器-至-交换机消息	20



Type value in OpenFlow header	OpenFlow Packet Type
0	Hello
1	Error
2	Echo Request
3	Echo Reply
4	Vendor
5	Features Request
6	Features Reply
7	Get Configuration Request
8	Get Configuration Reply
9	Set Configuration
10	Packet Input Notification
11	Flow Removed Notification
12	Port Status Notification
13	Packet Output
14	Flow Modification
15	Port Modification
16	Stats Request
17	Stats Reply
18	Barrier Request
19	Barrier Reply
20	Queue Get configuration request
21	Queue get Configuration Reply

表 5.2.

**Message length(消息长度)** 此字段代表封装在一个特定通用OpenFlow数据头中的OpenFlow数据包的总长度。整个数据包的长度（数据头+有效负载）使用八位字节表示。

**Transaction ID** 此Transaction ID字段用于将请求指令与响应指令相对应。某数据包中响应指令的事务ID将必须与数据包中请求指令的事务ID相匹配，以使得配对明了清晰。

### 5.3.2 Hello 数据包

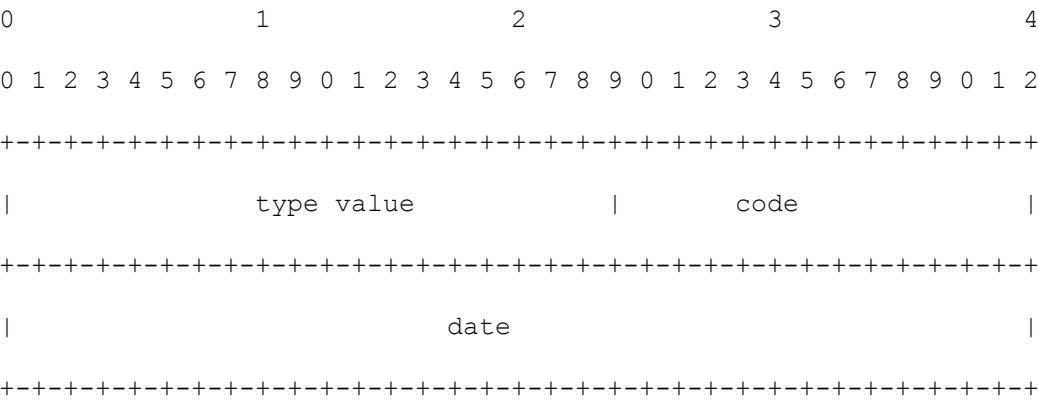
Hello数据包没有具体的主体，它只包含一个类型值为0的OpenFlow数据头。接收者和发送者通过查看OpenFlow数据头的类型字段来隔离这些数据包。

### 5.3.3 Error 数据包

以下是一个 Error 数据包的图形展示。它被封装在一个通用的 OpenFlow 数据头中，并在某些 Error 情况发生时被发送。



数据包5.3.3



**Type value.** Error数据包中的类型值字段，其数值为零至五，且对于每一类型值可能会有不同的代码类型。

- “Type Value （类型值）= 0”
  - o 原因： 问候协议失效

当一个hello 消息失效时，则发送类型值为零的误差数据包。其可能会有以下代码值。

Code	Description
0	No Compatible version
1	Permission Error

表 5.3.

- “Type Value（类型值）=1”
  - o 原因：请求未被识别

当OpenFlow请求未被解析，发送Type Value为1的误差数据包。其可能会有以下代码值。

Code	Description
0	Version not supported
1	Header Type not supported
2	Type of Statistics request not supported
3	Vendor not supported
4	Vendor subtype not supported
5	Permission Error
6	Wrong request length for type
7	Specified Buffer is used already
8	Specified buffer does not exist.

表 5.4.

- “Type Value（类型值）=2”
  - o 原因：action描述中的误差

当action描述符消息中有一个误差时，则发送Type Value为2的误差数据包。其可能会有以下代码值。

Code	Description
0	Unknown Action type
1	Length problem in actions
2	Unknown Vendor ID specified
3	Unknown action type for Vendor ID.
4	Problem Validating output actions
5	Bad action arguments
6	Permission Error
7	Can't handle too many actions
8	Problem Validating output queue

表 5.5.

- “Type Value （类型值）=3”
  - 原因：修改flow entry中的问题

当修改flow entry的action失效时，则发送Type Value为3的误差数据包。其可能会有以下代码值。

Code	Description
0	Flow not added because of full tables
1	Attempted to add overlapping flow with CHECK_OVERLAP flag set.
2	Permission Error
3	Flow not added because of non-zero idle/ time out.
4	Unknown command
5	Unsupported action list - cannot support in the specified order.

表 5.6.

- “Type Value （类型值）=4”
  - 原因：端口修改（port modification）请求失效

当端口修改（port modification）请求指令中有一个误差时，则发送Type Value为4的数据包其可能会有以下代码值。

Code	Description
0	Specified port does not exists
1	Specified HW address is wrong.

表 5.7.

- “Type Value（类型值）=5”
  - o 原因：队列操作失效

当队列操作中有误差时，则发送类型值为五的误差数据包。其可能会有以下代码值。

Code	Description
0	Invalid Port
1	Queue does not exists
2	Permission Error

表 5.8.

**数据。**基于上节中解释的类型及代码值，其长度是可变的。通常其包含至少六十四字节的失效请求。

#### 5.3.4 Echo 请求数据包

一个echo-request数据包仅为一个OpenFlow数据头加一个数据有效负载用作offset。Data offset可能有时间戳来检查延迟，它可以是各种长度来确定控制器和交换机之间的带宽。在OpenFlow数据头中，可以通过其类型值将其识别出。

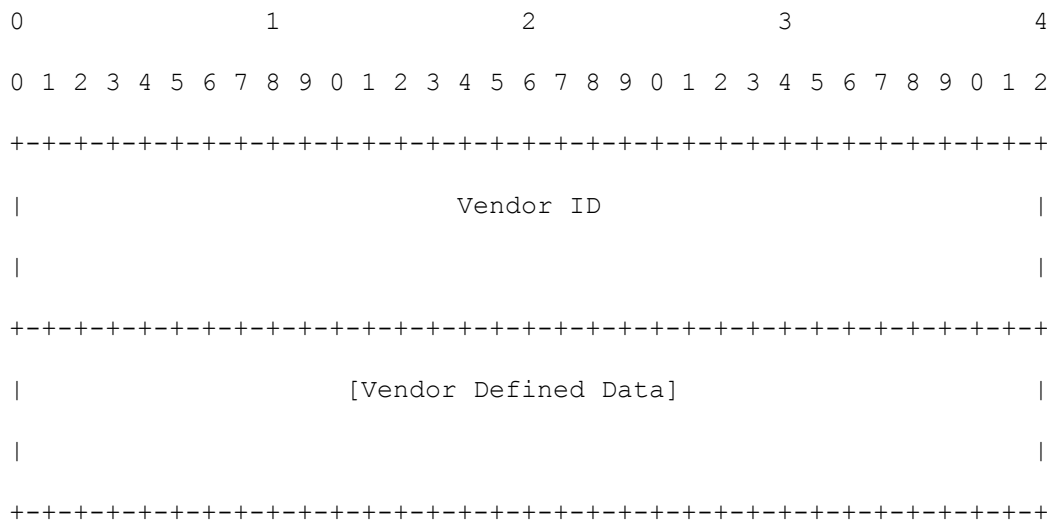
### 5.3.5 echo 响应数据包

一个echo-request数据包仅为一个OpenFlow数据头加一个数据有效负载用作offset。Data offset可能有相同的数据作为echo请求。在OpenFlow数据头中，可以通过其类型值将其识别出。这个数据包的发送是作为对echo-request数据包的响应。

### 5.3.6 各厂商数据包

此包用来确定各厂商的唯一性。如果一个交换机不理解一个各厂商ID的有效负载，它应该以误差数据包回应。以下是各厂商数据包的图形展示。

数据包5.3.6



**Vendor ID.**此为三十二比特的数据，来唯一确定一个各厂商。如果OpenFlow交换机不理解各厂商ID，那么它应该发送误差消息“类型=1，代码=3。”

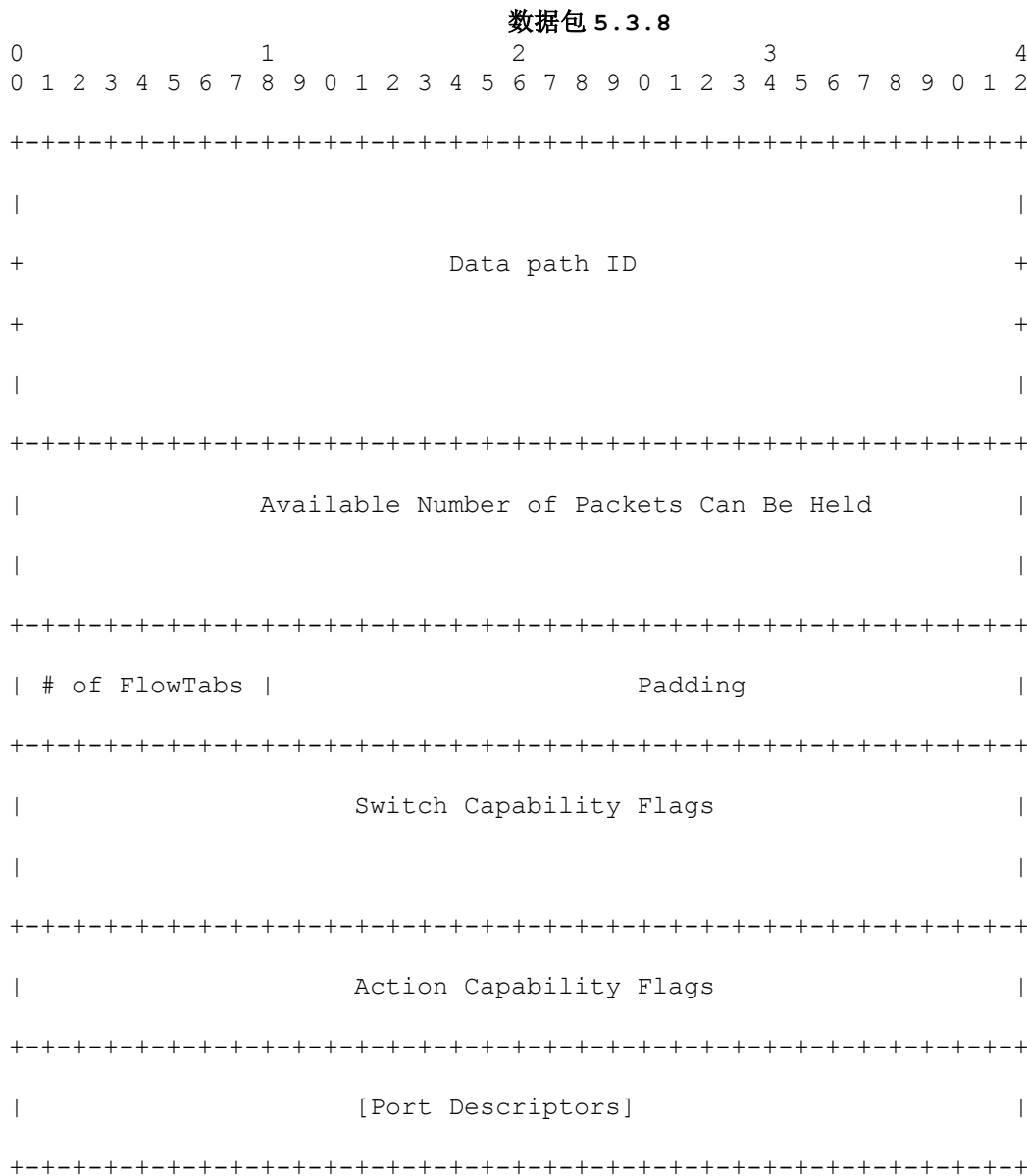
**Vendor-defined data.**各厂商定义数据要按具体实现而定。

### 5.3.7 特征请求

一个feature-request数据包为一个OpenFlow数据头加一个数据有效负载用作offset。控制器和交换机之间建立会话之后，控制器发送feature-request消息至交换机。

### 5.3.8 特征响应

**Feature-reply** 消息的发送是作为对 **feature-request** 消息的回应。在收到 **feature-reques** 消息后，交换机必须以带有“具体负载”的 **Feature-reply** 消息进行回应。通常在一个 **feature-request** 消息中，控制器向交换机寻求可用端口的细节及其详情等。交换机使用可用端口及其细节进行回复。以下是特征响应数据包的图形演示。



**Data path ID.** 数据路径ID从根本上说就是一个交换机ID。其用于数据路径辨别。底部的四十八个比特为交换机MAC地址，顶部的十六个比特按具体实现而定。

**Number of flow tables.**由数据路径（OpenFlow交换机）支持的流表数量可在零至254之间。每一个表都可以有其自己支持的通配符比特及条目数量。控制器可选择使用一个表类型状态请求指令调查每个表大小上的更多细节。

**Switch capability flags.**此数据包描述符中的标志给出了交换机所支持物的消息。以下是数据包描述符的图形展示。

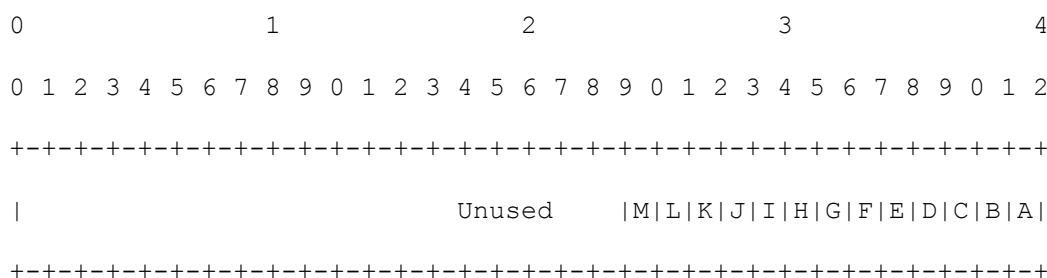
```
0             1             2             3             4
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               Padding                                |H|G|F|E|D|C|B|A|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Bit	Description
A	Flow statistics support.
B	Table statistics support.
C	Port statistics support.
D	IEEE 802.1D spanning tree support.
E	Reserved (Must be 0)
F	IP fragmentation support
G	Queue statistics support
H	Matching of IP address in ARP packets

70

**Action 性能标志。**这里详细讲述了交换机支持哪些 action。以下是 action 性能标志数据包描述符的图形展示。

数据包5.3.8 (ii) (Action性能标志)



Bit	Actions associated with flow table or Packets.
A	set Output to switch port
B	Set 802.1Q VID
C	Set 802.1Q PCP
D	Strip 802.1Q tag
E	Set Ethernet source address
F	Set Ethernet destination address
G	Set IPv4 source address
H	Set IPv4 destination address
I	Set IPv4 DSCP
J	Set TCP/UDP source port
K	Set TCP/UDP destination port
L	Output to queue
M	Vendor

表 5.10.

### 5.3.8.d1 端口描述符

Following is the pictorial presentation of a port descriptor packet.

以下是端口描述符数据包的图形展示。

数据包5.3.8.d1 (端口描述符)



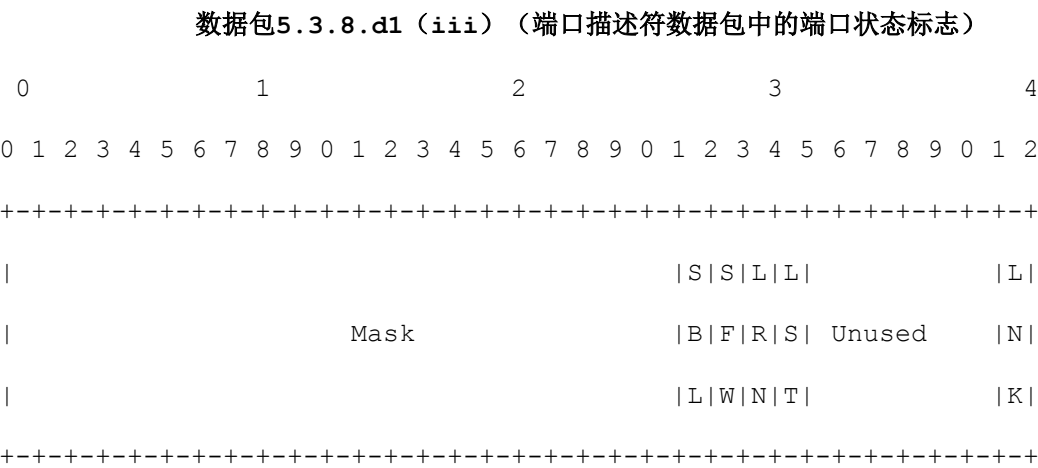
[illegible]





- ADM（管理员状态）：** 这表示某端口管理员是否出故障
- NST（无\_STP）：** 表示 STP 是否有效；控制器可将此比特设置为一或零来使端口上的 STP 有效或失效。
- BPD（仅收到 BPDU）：** 可将此比特配置为放弃端口上收到的所有数据包（除 BPDU）
- NBP（未收到 BPDU）：** 可将此比特设置为放弃端口上收到的任何 BPDU
- NFL（无泛洪）：** 若 STP 在端口上有效（如，NST=0），则由 STP 设置端口上的比特；若 STP 无效（如，NST=1），那么比特设置为零
- NFD（无转送）：** 端口上运行的 STP 决定了端口是否位于转送状态
- PIN：** 表示不向端口发送包入消息
- 未使用：** 必须为零

**端口状态标志。** 这些比特表示了某端口的当前状态。这些比特不可由控制器配置，只可用于读数。



- **LNK（链路状态）**
  - o 特定链路是否有物理上的故障
- **LST（监听）**

- 端口在监听状态
- **LRN (Learning)**
- **LRN (学习)**
  - 端口在学习状态
- **SFW (STP转送状态)**
  - 通信在端口上呈转送状态
- **SBL (STP阻塞状态)**
  - 端口不是STP的一部分
- **掩码**
  - 掩码来决定STP状态

**端口特征标志。**端口描述符中的电流、通告、支持及对等标志以这种形式发送。

## 数据包5.3.8.d1 (iv) (端口描述包中的端口特征标志)

[illegible]

比特的详细消息见表5.12。

Bits	Description
A	10 Mbps Half Duplex
B	10 Mbps Full Duplex
C	100 Mbps Half Duplex
D	100 Mbps Full Duplex
E	1 Gbps Half Duplex
F	1 Gbps Full Duplex
G	10 Gbps Full Duplex
H	Copper medium
I	Fiber Medium
J	Auto Negotiation
K	Pause
L	Asymmetric pause

表 5.12.

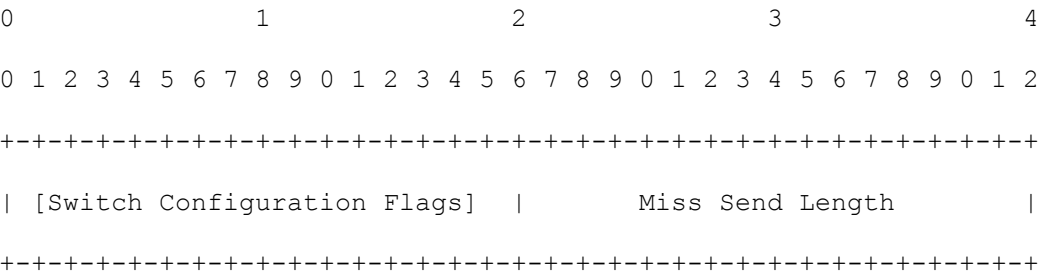
5.3.9 获取配置请求

一个获取配置请求数据包仅包含一个OpenFlow数据头。此包由控制器发送，以从交换机处得到配置消息。

5.3.10 获取配置响应

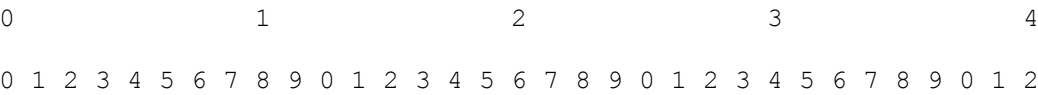
交换机将此包发送至控制器，解释交换机配置的状态。以下是数据包的图形展示。

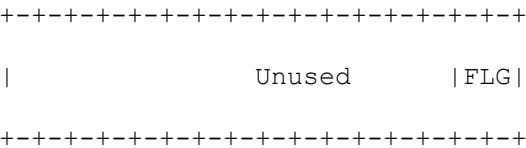
数据包5.3.10



交换机配置标志。

交换机5.3.10 (i) (交换机配置标志)





FLG	Action
0	无分片处理
1	放弃片段数据包
2	重新装配 IP 片段（仅当交换机性能标志支持时）

表 5.13.

**Miss 发送长度。** 此为应发送至控制器的新流的最大八位字节。缺省值为128。

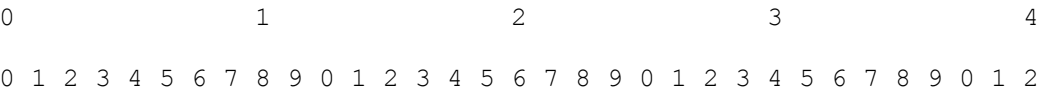
5.3.11 设置配置

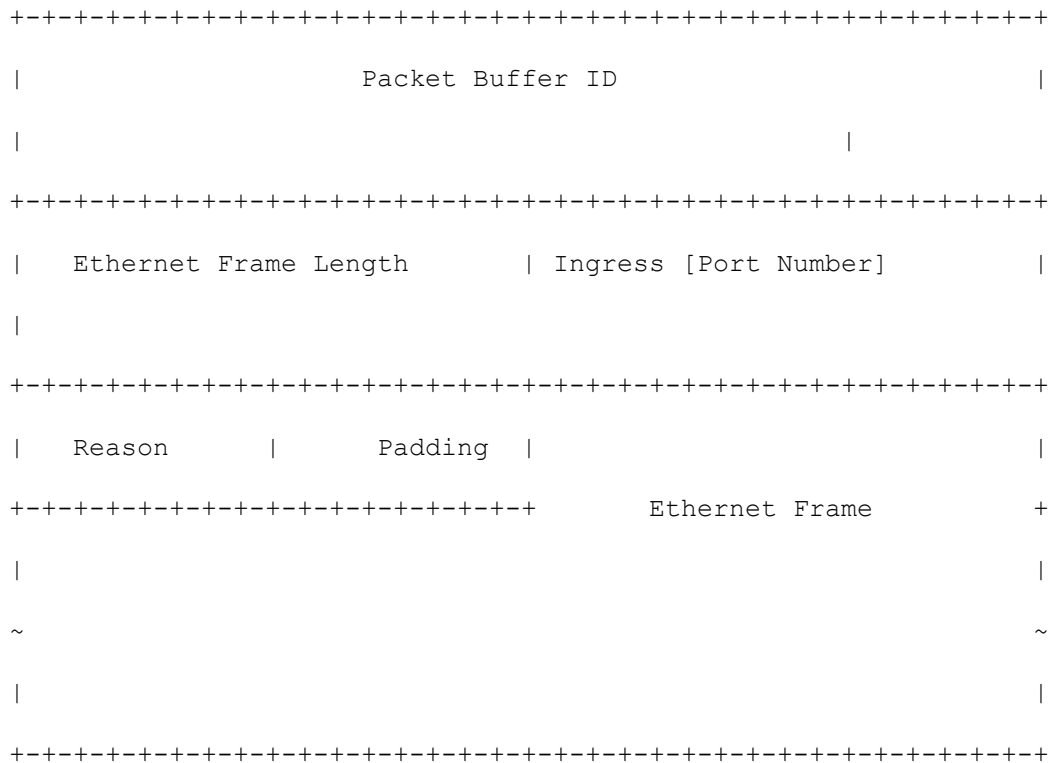
此包由控制器发送至交换机，此报文包含设置了配置的具体状态。数据包的详细消息与获取配置响应相同。如果控制器不得不改变其通过获取配置响应收到的配置，则其将发送有着不同交换机配置标志或 *Miss send length* 的相同数据包（*Miss send length* 定义了一个 OpenFlow 交换机可发送至控制器的数据包大小——仅在没有流相匹配的情况下——缺省值为 128 字节）来设置新的配置。

5.3.12 Packet-In

对于所有不包括匹配 flow entry 的数据包，或者某数据包通过一个发送至控制器的 action 与某 entry 相匹配，则 packet-in 事件被发送至控制器。如果交换机有足够的内存来存放将发送至控制器的缓冲数据包，则 packet-in 事件包含数据头的一些片段（缺省值 128 字节），当交换机准备好要转送数据包时，控制器将使用一个缓冲 ID。不支持内部缓冲（或已用尽内部缓冲）的交换机必须发送完整数据包至控制器作为事件的一部分。以下是包入数据包的图片展示。

数据包5.3.12





**Packet buffer ID.**缓冲在交换机中的数据包标识号

**Ethernet frame length.**完整数据包长度

**Ingress port number.**收到数据包的端口

**原因。**0：无匹配流；1： action明确输出至控制器

**填充。**如有需要，填充

**以太网框架。**部分数据包（如果数据包可被缓冲）或全部数据包（如果数据包不可被缓冲）



### 5.3.13 流移除通知

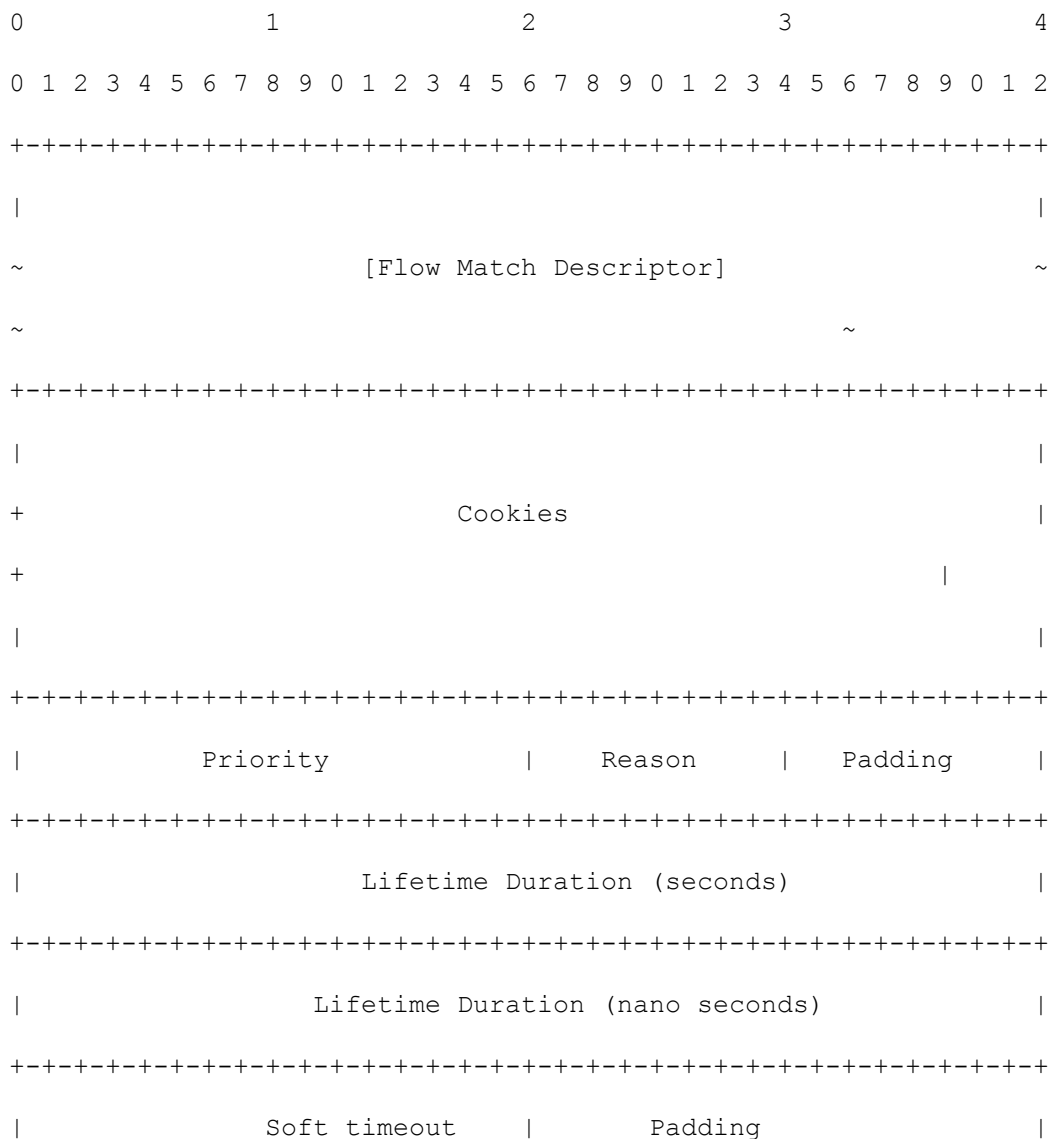
当流从交换机移除时，即发送数据包（某些情况下）。为了对其详细了解，我们需要看一下流是如何安装的。当流条目通过流修改消息添加至交换机上，一个 **idle time-out value** 表示条目由于缺少或者非 **activity** 状态而被移除，而一个 **hard time-out value** 表示条目应当被移除，而不管是否在 **activity** 状态。

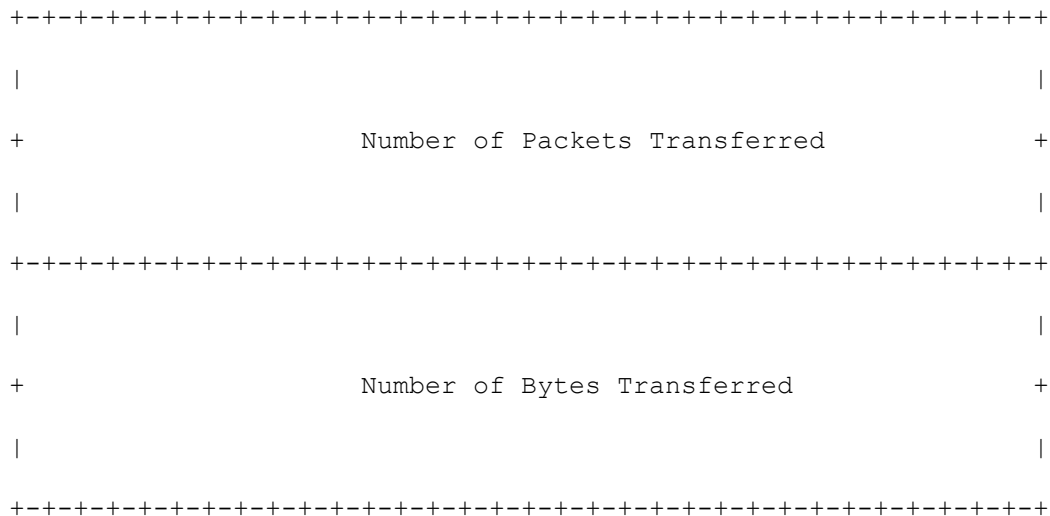
流修改消息同时也指明，当流时间终止时交换机是否应发送流移除消息至控制器（见 5.3.16 节是否设置了字节 R）。

如果流因任何原因被删除，则发送数据包，当流被安装时，设置 R 字节。

以下是本数据包的图形展示。

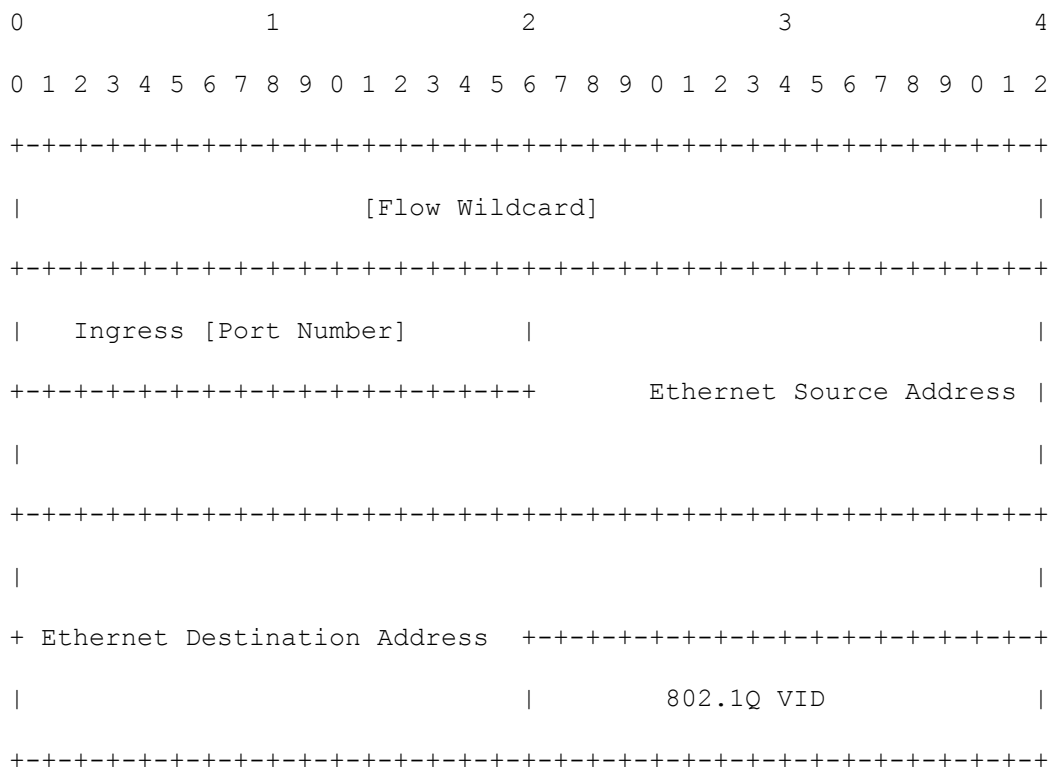
数据包 5.3.13

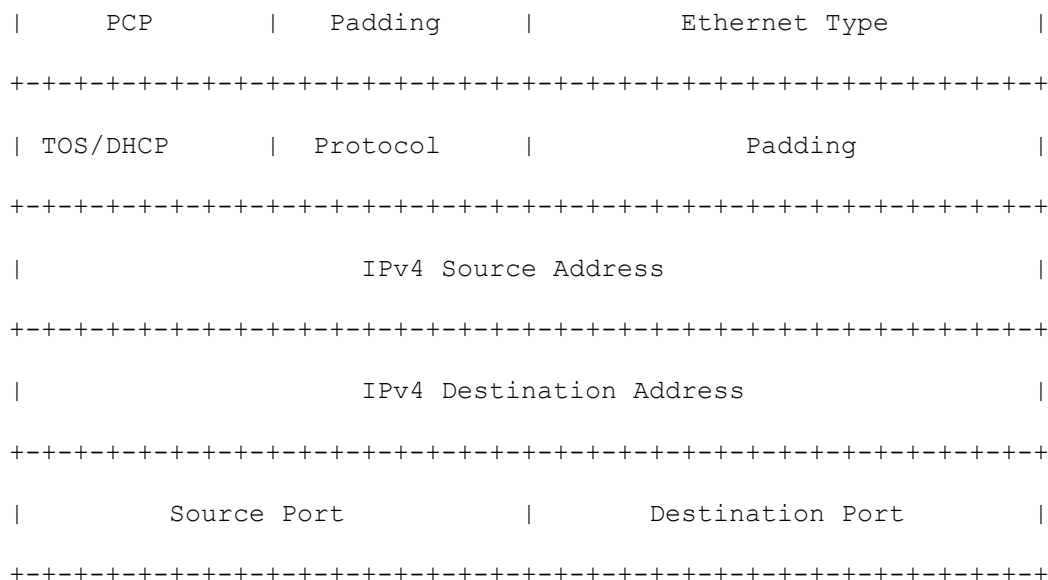




### 5.3.13.d1 流匹配描述符报文

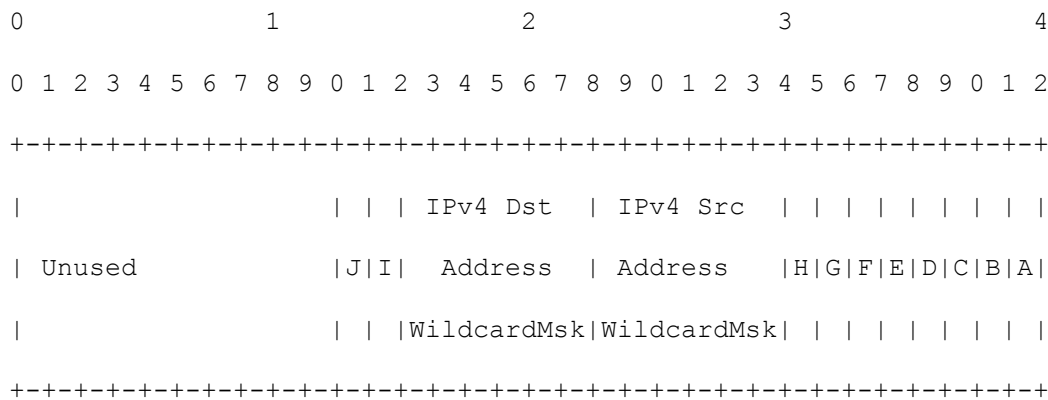
数据包5.3.13.d1（流匹配描述符）





流通配符

数据包5.3.13.d1 (i) (流匹配描述符中的流通配符)



- 比特标识:

Bits	Description
A	Switch Input port
B	Vlan ID
C	Layer-2 Source address
D	Layer-2 destination address
E	Ethernet frame type
F	IP Protocol
G	TCP/UDP Source Port
H	TCP/UDP Destination Port
I	Vlan Priority
J	ToS/DHCP

表 5.14.

- 未使用
  - 一定为零
- IPv4目标地址通配符掩码
  - IPv4目标地址通配符掩码计数。零为正好匹配，一忽略了IPv4地址的LSB，二忽略了两个最不显著的比特……三十二至更高，整个字段为通配符。这与通常的传统，即/24表示忽略了八个比特不同。  
 例：值=0表示：255.255.255.255 1.1.1.1/32（正好匹配）  
 值=8表示：255.255.255.0 1.1.1.1/24（前三个八位字节匹配）
- IPv4源地址通配符掩码
  - 同上解释的逻辑

如果没有设置通配符，那么流匹配描述符通过使用流表的所有十二元组恰好描述了一个流；如果在不管字段值的情况下设置了所有通配符，那么每个流都会匹配。

- 入口端口编号
  - 收到数据包的端口
- 以太网源地址
  - 流的2层源地址

- 以太网目标地址
  - 流的2层目标地址
- **802.1Q VID**
  - IEEE 802.1Q VLAN标识符；0xffff表示无IEEE 802.1q标签存在
- **802.1Q PRI**
  - IEEE 802.1Q优先级
- 以太网类型/长度
  - 以太帧类型
- **IPv4协议**
  - 以太网类型为0x0806时IPv4协议或ARP类型
- **IPv4源地址**
  - IPv4源地址
- **IPv4目标地址**
  - IPv4目标地址
- **TCP/UDP源端口**
  - Layer-4 源 port 或者ICMP 类型
- **TCP/UDP目标端口**
  - Layer-4 目的 port 或者ICMP 类型

流匹配描述符数据包详细消息在此点结束。

**消息记录程序：**控制器发布的一个标识符。

**优先级：**对于流移除数据包，本字段只用于消息传递（其不在流移除事件中起任何作用）。但是，在正常的流表中，优先级值越高，流的优先级越高。

原因:

- 0代表流空闲超时超过了soft time-out。
- 1代表时间超过了hard time-out。
- 2表示删除一个flow通过发送flow mod消息。

**填充:** 填充至三十二比特。

**寿命持续时间（秒）：**流在交换机中存活持续时间（秒）。

寿命持续时间 (毫微秒)；流在交换机中存活持续时间 (毫微秒)。

**软超时：** 原始流修改中得出的软超时。

**填充:** 填充至六十四比特。

**转送数据包编号：**命中流条目的数据包数量。

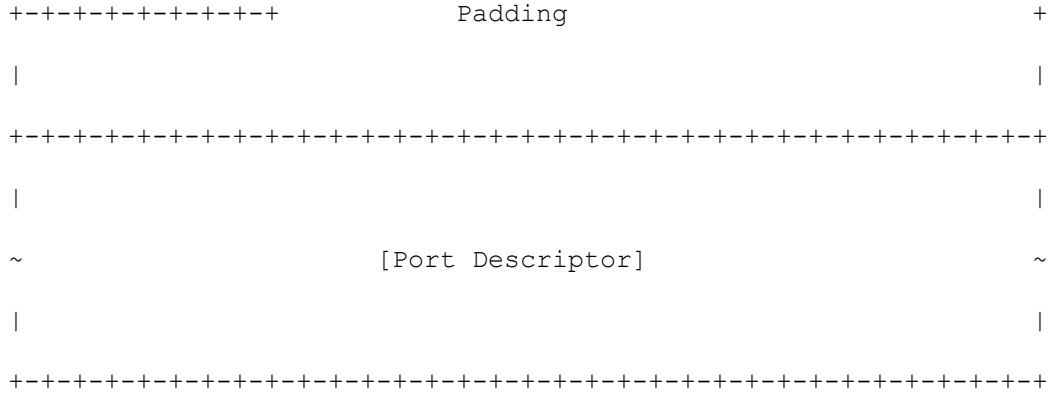
**转送比特数量：**命中流条目的比特数量。

### 5.3.14 端口状态通知

当端口配置状态改变后，交换机预计将发送端口状态消息至控制器。这些事件包括端口状态改变（比如，若其直接由用户改变）或者 802.1D 中指定的端口状态改变。以下为本数据包的图形展示。

### 数据包5.3.14

0										1										2										3										4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+



## 原因

- 0: 端口增加
- 1: 端口删除
- 2: 端口编辑

## 填充

- 填充至六十四比特

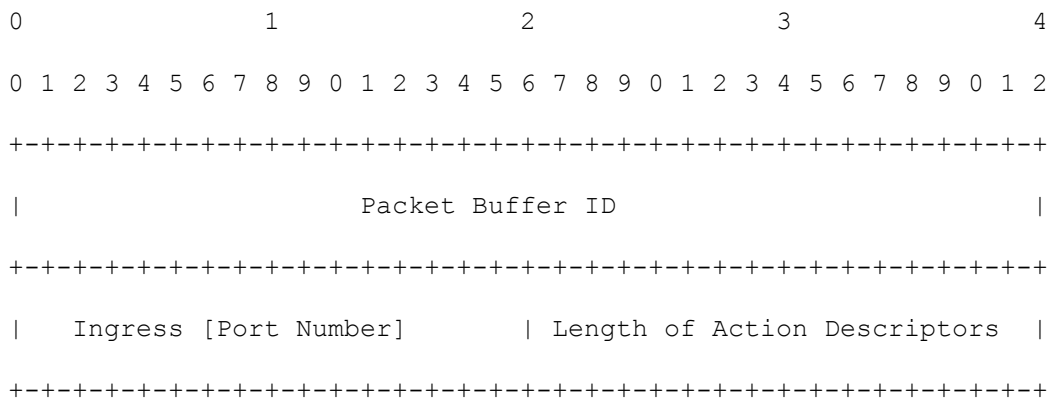
## 端口描述符

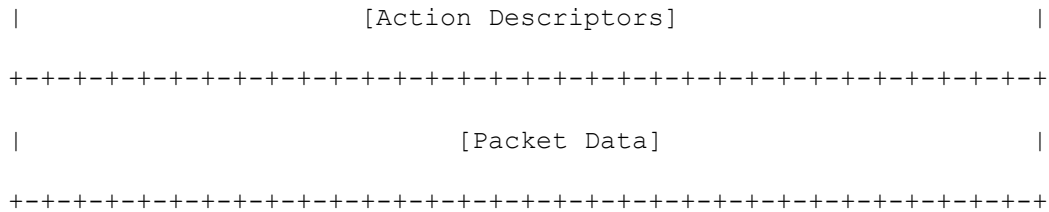
- 在5.3.8.d1节中有此解释

### 5.3.15 数据包输出

当控制器想要发送任何数据包至交换机时，便是触发了一个 **packet-out** 事件，然后数据包被发送至交换机。此触发可能是控制器<->交换机消息的响应，或由控制器发送的一些明确消息。以下是此数据包的图形展示。

**数据包5.3.15**





**数据包缓冲ID**

- 缓冲在交换机中的数据包标识号

**入口端口编号**

- 入口端口编号

**Actions长度**

- action列表的总长度

**Actions描述符**

- 参照5.3.15.d1节

**数据包数据**

- 交换机需要发送的实际数据包

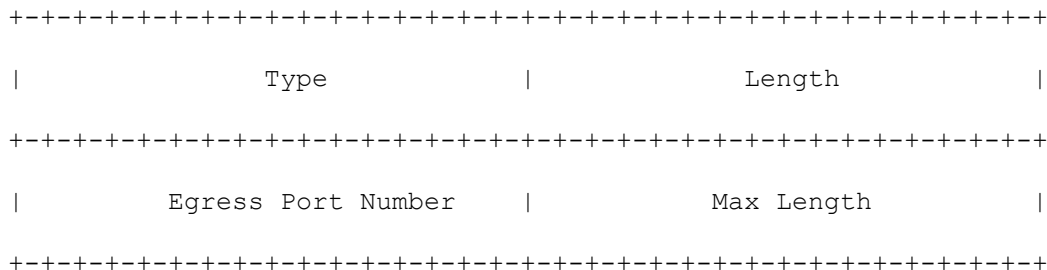
**5.3.15.d1 Actions 描述符**

可以采取不同类型的 action。以下为 action 描述符中可被定义的 Actions 列表。

输出至交换机端口：

数据包5.3.15.d1 (i) (Action描述符中的类型0)																							
0	1								2	3								4					
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	

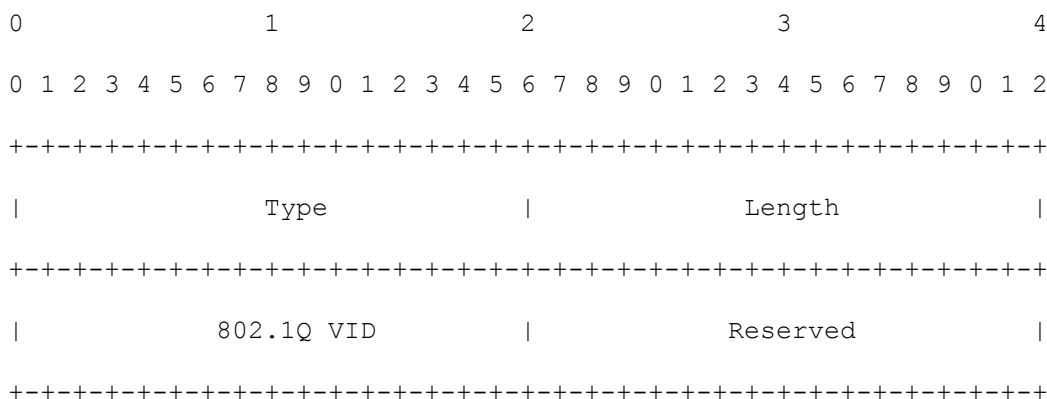




- 类型=0
  - o Action: 输出至交换机端口
- 长度=8
- 出口端口编号: 数据包需要发送出去的端口编号 (参考表3.1.)
- 最大长度: 可发送至控制器的最大长度。

#### 设置802.1Q VID:

数据包5.3.15.d1 (ii) (Action描述符中的类型1)



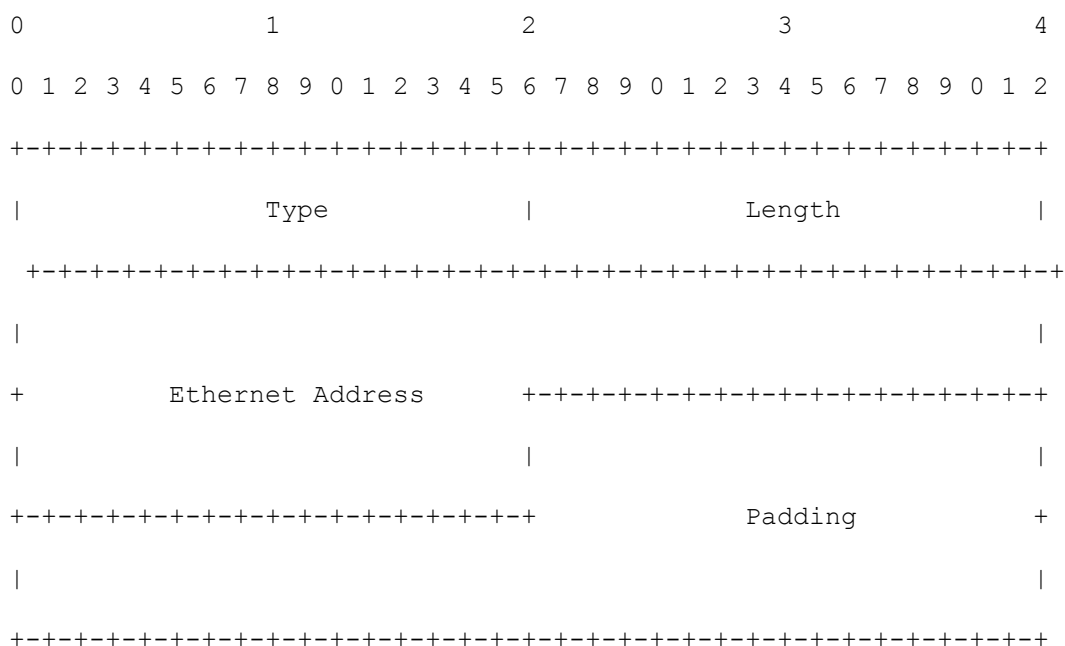
- 类型=1
  - o Action: 设置IEEE 802.1Q VLAN
- 长度=8
- 802.1Q VID = IEEE 802.1Q VLAN标识符

#### 设置802.1Q PCP:

数据包5.3.15.d1 (iii) (Action描述符中的类型2)



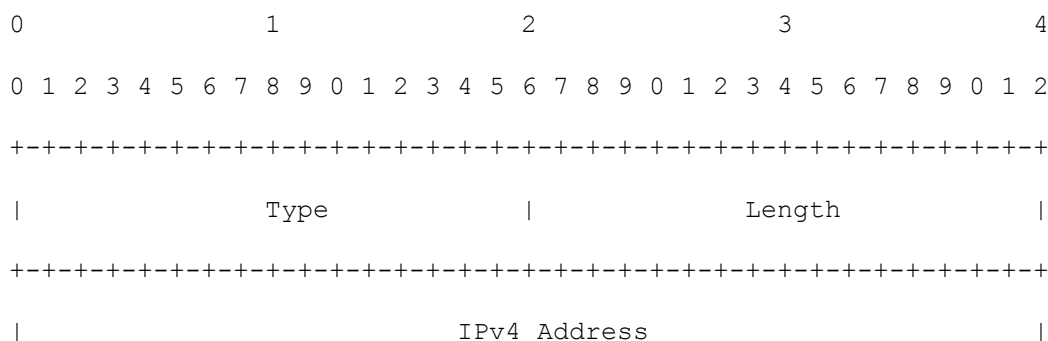
### 数据包5.3.15.d1 (v) (Action描述符中的类型4&5)



- 类型=4和5
  - o Action: 设置源以太网地址（类型4）；设置目标以太网地址（类型5）
- 长度=16
- 以太网地址=需要设置的以太网地址

设置IPv4地址:

### 数据包5.3.15.d1 (vi) (Action描述符中的类型6&7)



[illegible]

- 类型=6或7
  - o Action: 设置IPv4源地址（类型6）； action: 设置IPv4目标地址（类型7）
- 长度=8

### 设置IPv4 DSCP:

## 数据包5.3.15.d1 (vi) (Action描述符中的类型8)

[illegible]

- 类型=8
  - o Action: 为IPv4 DSCP设置值
- 长度=8
- IPv4 DSCP=将要设置的IPv4 DSCP值
- UN=未使用

### 设置TCP/UDP端口:

## 数据包5.3.15.d1 (vii) (Action描述符中的类型9&amp;10)

0	1										2										3										4																		
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-

Type	Length
UDP/TCP Port	Padding

- 类型=9和10
  - o Action: 设置TCP/UDP源端口（类型9）； action: 设置TCP/UDP目标端口（类型10）
- 长度=8

流修改消息由控制器发送至交换机，以修改（增加/编辑/删除）交换机流表中的一个流。以下为此数据包的图形展示。

[illegible]

- 细节见于5.3.13.d1节中的流匹配描述符

**指令：**

- 0 增加新流
- 1 修改所有匹配流
- 2 修改严格匹配通配符流的条目
- 3 删除所有匹配流
- 4 删除严格匹配的通配符和优先级

**软超时：**

- 流终止之前的空闲超时（秒）

**硬超时：**

- 流终止之前的最大时间（秒）

**优先级：**

- 值范围为0.65535

**数据包缓冲ID：**

- 在交换机中缓冲的数据包标识号

**Egress Port编号：**

- 细节见数据包描述符节（端口编号）

**比特-E：**

- 标记紧急流的比特

**比特-O：**

- 首先检查重叠条目的比特

**比特-R：**

- 流时间终止或被删除时，发送流移除消息的比特

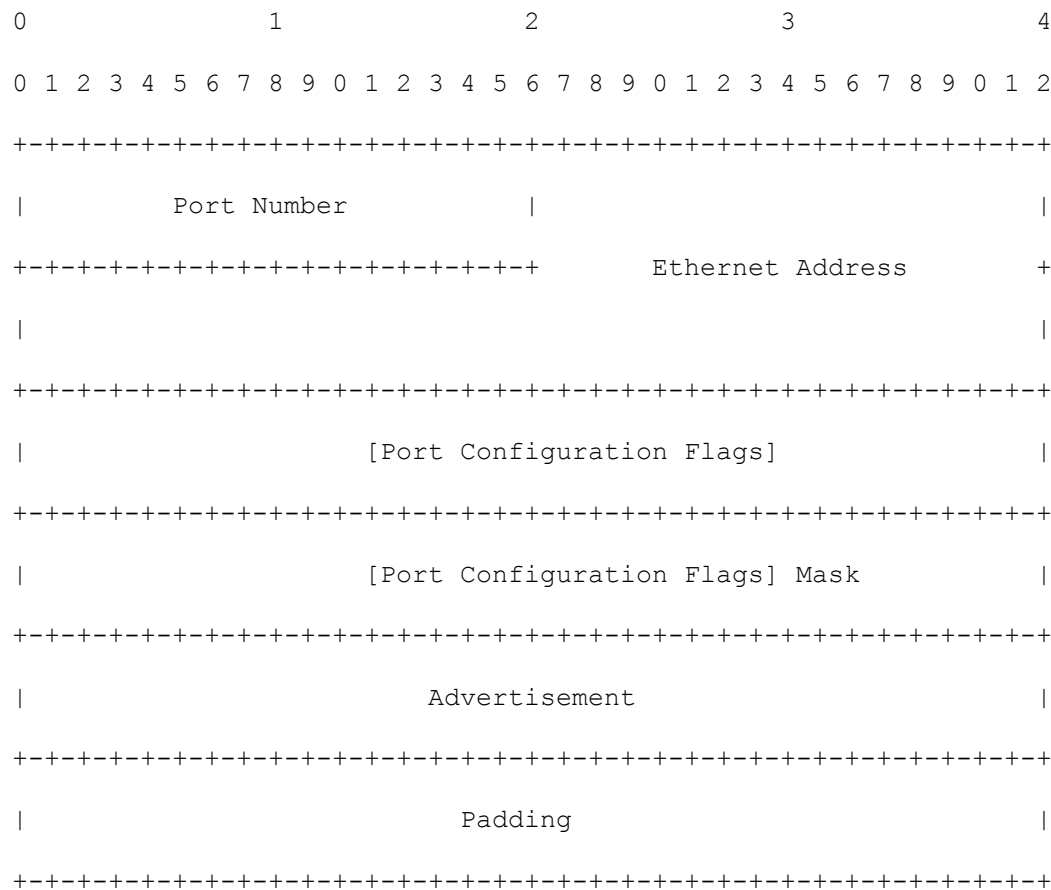
Action描述符:

- 见5.3.15.d1节

5.3.17 端口修改

控制器发送端口修改消息至交换机用以修改端口上的配置。以下为此数据包的图形展示。

数据包5.3.17



端口编号:

- 参照表3.1

以太网地址:

- 端口上2层MAC地址



端口配置标志:

- 5.3.8.d1节中有此解释（端口配置标志）

端口配置标志掩码:

- 5.3.8.d1节中解释的标志掩码（端口配置标志）

通告（端口特征标志）

- 5.3.8.d1节中有此解释（端口特征标志）

Padding:

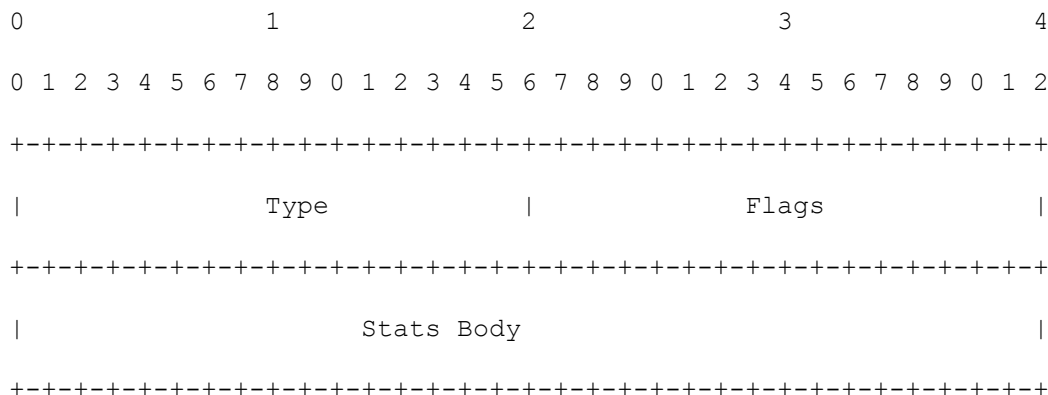
填充:

- 填充至六十四比特

### 5.3.18 统计请求

控制器发送此消息来查询统计。以下为此数据包的图形展示。

数据包5.3.18



类型:

- 0 OpenFlow交换机描述
- 1 单个流消息
- 2 聚合流消息

- 3 流表统计消息
- 4 端口统计消息
- 5 队列统计消息
- 65535各厂商扩展

标志:

- 未使用

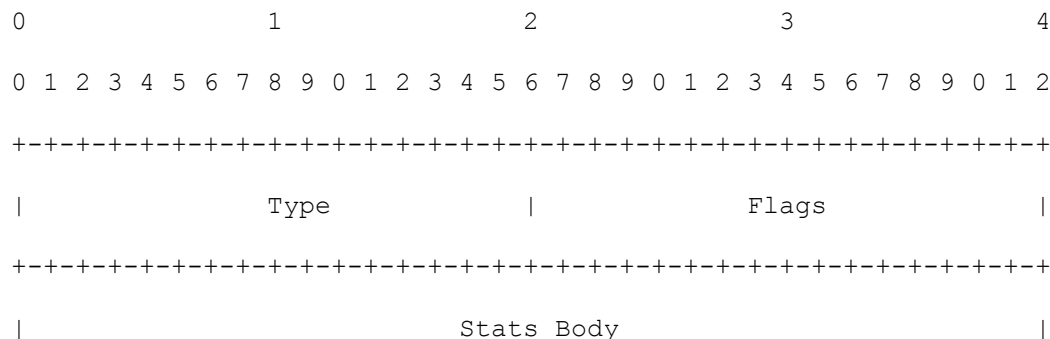
统计body:

- 类型=0（交换机描述）时，统计body为空
- 类型=1（单个流消息）时，以下消息为统计body字段。
  - o 匹配的字
  - o 读取表格的ID
  - o 要求匹配条目将其包括在输出端口中
- 类型=2（聚合流消息）时，以下消息为统计body
  - o 匹配的字
  - o 读取表格的ID
  - o 要求匹配条目将其包括在输出端口中
- 类型=3（流表上的消息）时，统计body为空
- 类型=4（端口数据消息）时，以下消息为统计body
  - o 端口编号，可以为单个端口或端口列表
- 类型=5（队列数据消息）时，以下消息为统计body
  - o 所有消息需要的端口
  - o 所有队列ID

### 5.3.19 状态响应

交换机发送此消息至控制器以回应状态请求。以下为此数据包的图形展示。

数据包5.3.19



类型:

- 标志:**

- ### 统计body:

- 98

- 流动数据包数量
  - 流动字节数量
  - 流数量
- 类型=3（流表上的消息）时，以下消息为统计主体
  - 表的标识符（首先查阅较小编号的表）
  - 表支持的通配符
  - 支持条目的最大数量
  - 活动条目的数量
  - 表中查阅的数据包数量
  - 命中表的数据包数量
- 类型=4（端口数据消息）时，以下消息为stats body
  - 收到数据包的数量
  - 传送数据包的数量
  - 收到字节的数量
  - 传送字节的数量
  - RX放弃的数据包数量
  - TX放弃的数据包数量
  - 接收错误的数量（这是更加具体的接收错误的超级组，应大于或与所有RX错误值总和相等）
  - 传送错误的数量（这是更加具体的传送错误的超级组，应大于或与所有TX错误值总和相等——无当前定义）
  - 数据帧校准错误数量
  - RX超限运行的数据包数量
  - CRC错误数量
  - 冲突数量
- 类型=5（队列数据消息）时，以下消息为状态主体
  - 队列ID
  - 传送字节的数量
  - 传送数据包的数量
  - 由于超限运行而放弃的数据包数量

### 5.3.20 Barrier 请求

一个Barrier请求数据包为一个OpenFlow数据头加一个数据有效负载用作补偿。数据补偿可按照各厂商的实现而使用。

### 5.3.21 Barrier 响应

一个Barrier响应数据包即为一个OpenFlow数据头加一个数据有效负载用作补偿。数据补偿可按照各厂商的实现来使用。

控制器可向交换机索询某端口上的配置队列。控制器发送此消息至交换机来了解某特定端口上配置队列。以下为此数据包的图形展示。

数据包 5.3.22

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Content	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
Label	Port Number																Padding							

交换机以此消息做出响应，包括被请求的特定端口上配置队列的详细消息。以下为此数据包的图形展示。

**数据包 5.3.23**

The diagram illustrates the structure of packet 5.3.23, which is 24 bytes long. The bytes are indexed from 0 to 23. The packet is divided into three main sections:

- Port Number:** Occupies bytes 0 through 11 (12 bytes).
- Padding:** Occupies bytes 12 through 19 (8 bytes).
- Queuing information:** Occupies bytes 20 through 23 (4 bytes).

The diagram uses vertical lines to separate these sections and horizontal lines to mark the byte boundaries.

**队列消息。**由以下参数表示:

- 队列 ID

- 队列的 ID。可以有多个队列 ID 根据已给信息。对于每个队列 ID，队列消息都有不同的变量（与此队列 ID 相关）。
- 对于每个队列 ID，将会有以下变量：
  - 此队列 ID 的队列消息长度
  - 属性
    - 值=空；如果值为空，此队列 ID 的队列消息在此终止。
    - 值=最低比率限度；如果设置了此标志，队列消息页同样设置了为此队列 ID 指定了最低比率的变量。

## 第六章 Vxlan 介绍

本章是专门针对 SDN 的实施操作以及虚拟机技术，因此，围绕解决出现的问题的技术已经随着虚拟机演变应运而生。这项技术主要基于一个假设：服务器的视图将被抽象到交换机/路由器为基础的网络中。与 OpenFlow 不同（该开关是可编程的），这项技术专注于虚拟机之间的终端到终端之间的通信，而虚拟机是由网络边界所分隔。专注于这项技术的主要理由是：

- 服务器虚拟化导致网络上活跃着数量庞大的虚拟机（VM），这正造成现有的网络基础设施找到一个使需求增长的方法。
- 随着虚拟被分割成组，数据中心每分每秒都在增长。原来这都是使用 VLAN 来完成的。然而，4094 个 VLAN 可能不足够应付这种增长的需求。
- 数据中心托管多个租户，每个租户有自己的应用程序。每个租户都存在于自己的逻辑网络。

### 6.1 当前服务器虚拟化设计的难题

本节介绍了服务器虚拟化的广泛应用下暴露的具体问题。

#### 有限的 VLAN 范围

随着数据中心规模和容量扩大，当前的虚拟局域网（VLAN），使用一个 12 位 VLAN 账号，可能不足以提供必要的传播的独立性。随着这个虚拟化的需求越来越大，需要一个更具可扩展性的解决方案产生。

#### 虚拟机操作

随着虚拟化和云计算的诞生，虚拟机可以从一台服务器无缝地移动到另一个上去，并且对被移动的虚拟机没有任何影响。然而，在目前的设计中，只有 VM 是在相同的 IP 子网上，这才能达到。这个限制将不会允许虚拟机跨过不同的 IP 子网移动，并且随着数据中心的规模和复杂性增长，这可能是一个重大的问题。（请参阅图 6.1，它描述了在不同的子网中使用 VMotion 的问题）。

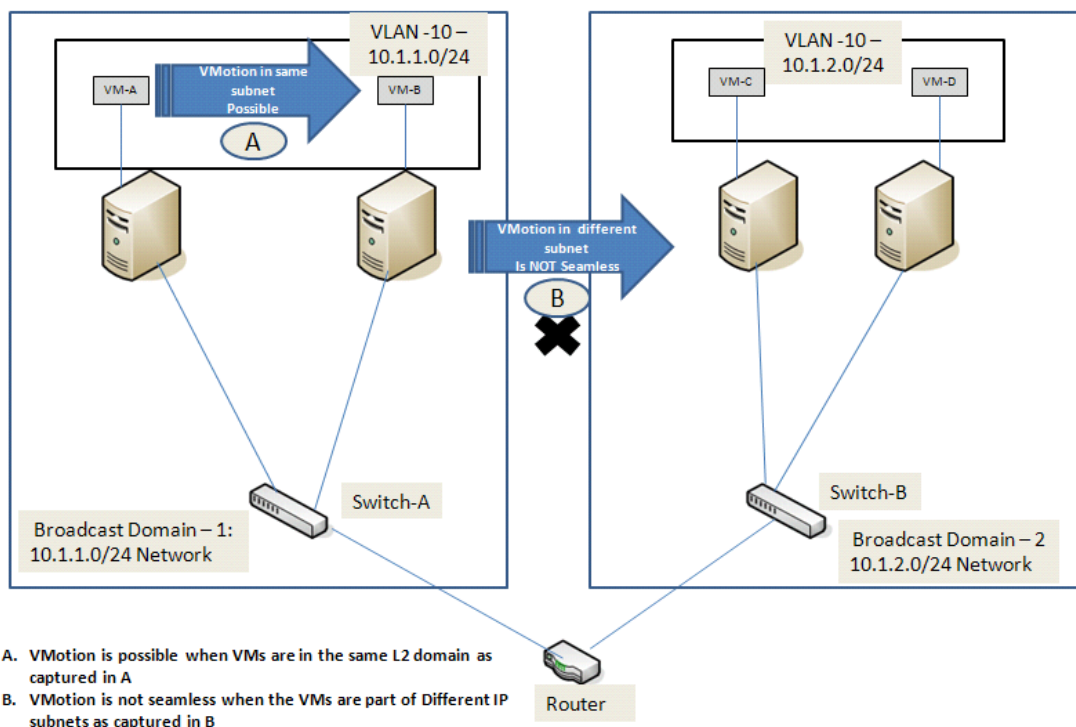


图 6.1

### TOR 交换机的地址表

随着虚拟机扩展的限制，TOR 通常连接到服务器，不得不载入数量不断增加的 MAC 地址。此外，由于每个数据中心都有许多 racks，每个 TOR 需要为虚拟机保持一个地址表，因为其需要跟其他虚拟机在数据中心通信。

### 多租户

数据中心托管多个租户。在云中，需要向许多租户提供弹性服务，而每个租户需要流量隔离。这种隔离处于第二层，正如虚拟局域网（12 位，4094）所提供的。在第 3 层网络的情况下，有可能有两个客户使用相同的第 3 层的寻址方案，这可能需要以不同的方式提供隔离。

## 6.2 VXLAN 概述



VXLAN（**虚拟可扩展局域网**）可以解决 6.1 中所述的问题。

VXLAN 可以在现有的基础设施上运行，并提供了一种方法来扩展在第 3 层网络上扩展第 2 层网络。VXLAN 可以被作为在第 3 层上第 2 层网络覆盖方案。每个 overlay 被称为 VXLAN segment。在相同 VXLAN segment 的虚拟机可以互相通信。

为了实现 VXLAN，引入以下技术：

**- VNI（虚拟网络标识符）：**

这是一个 24 位的 ID。VNI 标识一个 VXLAN。这给出了将近 16M（2 的 24 次方）可以使用的 VXLANs。VNI 将内部的帧封装（帧起源在虚拟机）。使用 VNI 封装有助于 VXLAN 建立 tunnel，该 tunnel 在第 3 层网络之上覆盖率第二层网络。

**- VTEP（VXLAN Tunnel End Point）：**

这条 tunnel 发起于一个称为 VXLAN Tunnel End Point（VTEP）。该 tunnel 从一个 VTEP 延伸到另一 VTEP，并且由 VNI 识别。VTEP 将从虚拟机发出/接受的帧封装/解封装，而虚拟机并不区分 VNI 和 VXLAN tunnel。

两个 VXLAN 可以具有相同的 MAC 地址，但一个段不能有一个重复的 MAC 地址。

**虚拟机对虚拟机的流量包**

考虑到一个 VXLAN 中的虚拟机覆盖了网络。此虚拟机不知道 VXLAN。要与不同的主机上的虚拟机进行通信，和之前一样，它将发送一个 MAC 帧来锁定目标。物理主机上 VTEP 查询 VNI 与哪一个 VM 相关。然后确定目标 MAC 是否在同一网段。如果是这样，一个外部 header 连接器将包括一个外围的 Mac，外部的 IP 地址，和 VXLAN header（参见图 1，在第 5 帧格式），这个外围接头连接器将被内置到原始的 MAC 帧的前部。最终的数据包被发送到目的地。这就是 VTEP 的 IP 地址和远程目标虚拟机连接（由 MAC 内部目的地址所显示）。

**VXLAN 实现的具体方法**

VXLAN 的控制计划可以由几种方法实现。其中有两个已经在 VXLAN 草案中讨论过。

- **- Learning Based on Data Flows:**

在数据平面的录入时，由虚拟机的 MAC 到 VTEP 的 IP 都是通过源代码录入发现。对于未知的目标，广播和多播流量，用于多播。

– **Central Repository Based:**

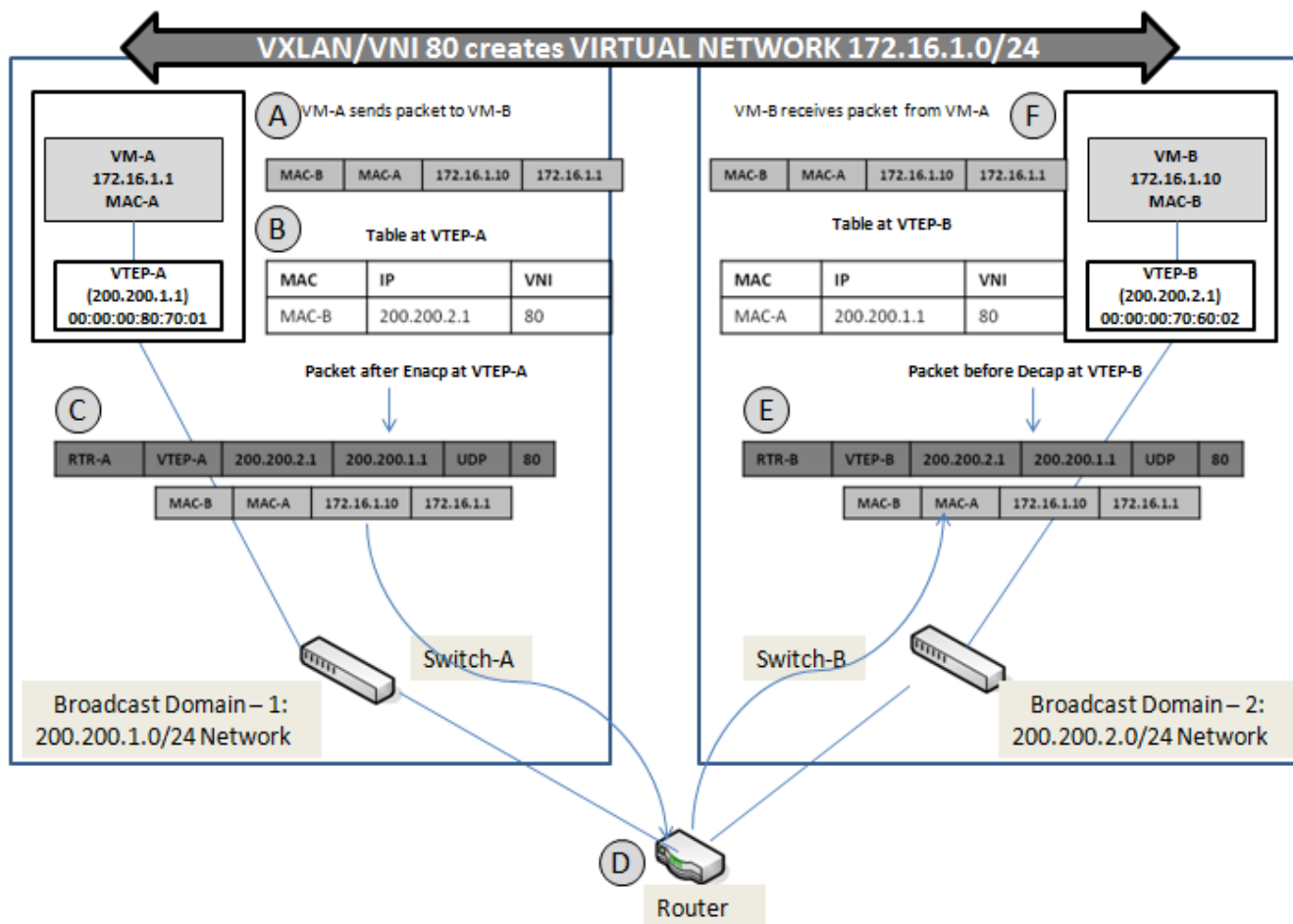
另一种方式来将虚拟机的 MAC 分配到 VTEP 的 IP 地址，可以是一个中央分配为基础的查找方式，通过个人 VTEP 或通过中央目录将地图信息分配到 VTEPs。

– **Treatment of Broadcast/Unknown Unicast/Multi-Cast**

在上述的虚拟机到虚拟机的数据流包时，当源虚拟机不知道目标 VM 的 MAC，将出现一个情形，并且它将发送一个 ARP 的数据包，该数据包是由 VTEP 使用 multi-cast 机制发送的。为了这个 multi-cast 机制工作，需要在 VXLAN 和 VNI 还有 IP multi-cast 组之间有映射，来用于发送这些数据包。

这的映射将启用 VTEP 提供 IGMP 组成部分报告到上游交换机/路由器到加入/断开的 VXLAN 相关的 IP multi-cast 组。基于这种结构，如果组成部分在没有提供具体的 multi-cast 组的主机上，这些叶节点可以被修改。独立 multi-cast 协议（PIM）sparse mode PIM dense mode，和 PIM 双向模式可以由 VXLAN 部署相应功能。

### 6.3 个案研究



本例研究用一个部署网络（在图 6.2 中有解释）为例，来解释 vxLAN 的功能。在图 6.2 中，字母代表了每一步骤。以下是每一步的解释（从 A 开始）

在两个 VM 发生真正交互前，可能需要解决交互 VM 间的 ARP。这些用于解决 ARP 的步骤被标记为第 0 步。

#### 0 步

VM-A 想要和一个在不同的主机上的 VM-B 进行交互。它需要发送一个框架到指定的 VM,但是它不知道指定 VM 的 MAC。

IV.未知 vxLAN 的 VM-A 发送一个 ARP 数据包去获取 VM-B 的 MAC 地址。

V.这个 ARP 通过物理服务器的 VTEP-A 封装成一个多址传送的数据包，而且这个是多址传送到一个和 VNI 有关的组织。

VI.所有和 VNI 有关的 VTEP 接收那个数据包，并且把 VTEP-A/VM-A MAC 的映像加到它们的表格中。

VII. VTEP-B 也接收了这个多址传送的数据包，它解封装这个数据包，然后填满内部的数据包，即 ARP 需要主机中 VNI 中某部分的所有端口。

VIII.VM-B 接收 ARP 的指令，然后建造一个 ARP 回复的数据包，并发送给和物理服务器的 VM-B 相关的 VTEP-B。

IX.当 VM-B 在它的为 VM-A 准备的表格中建立一个映射，且这个映像指向 VTEP-A 时，它将封装 ARP 反馈信息成单一传播的数据包，并把它发送给 VTEP-A。注意，目标 IP 将是 VTEP-A 的 IP。如果已经选择路径，或者，它是在同在二层网域，目标 MAC 会成为下一个路由器的 MAC，目标 MAC 成为 VTEP-A 的 MAC。

X. VTEP-A 接收这个数据包，并解封装，之后发送这个 ARP 反馈给 VM-A。

XI. VTEP-A 在它的表格中加上一个映射：VTEP-B IP/VM-B MAC。

#### **A 步**

VM-A 想要和 VM-B 交互，发送出一个附上源 MAC 的数据包（“MAC-A”），目标 MAC（“MAC-B”），源 IP 172.16.1.1 和目标 IP 172.16.1.10。

#### **B 步**

VM-A 是未知 VxLAN 的，然而，VM-A 归附的物理服务器是 VxLAN 80 的一部分。这个 VTEP 结点，在这个例子中是 VTEP-A，检查表格来确认是否它有一个到达目标 MAC-B 的入口。

#### **C 步**

VTEP-A 封装这个来自于 VM-A 的数据包，加上一个 VNI as 80 的 VxLAN 数据头，和有着特定目标 VxLAN 端口的 UDP 数据头，一个新的源 IP 作为 VTEP-A 的 IP，新目标 IP 作为 VTEP-B 的 IP，源 MAC 作为 VTEP-A 的 MAC，而且目标 MAC 作为链接开关 A 的路由器的接口的 MAC 地址。

#### **D 步**

一旦这个数据包到达路由器，它将执行正常的路由和依据相应接口进行转发，然后调整外层 header 源 MAC 和目标 MAC。

## E 步

这个数据包到达 VTEP-B，并且当数据包有一个带有 VxLAN 端口的 UDP 的数据头，VTEP-B 解封装这个数据包（所有内部框架被剥夺）然后发送内部的数据包给目标 VM。

## F 步

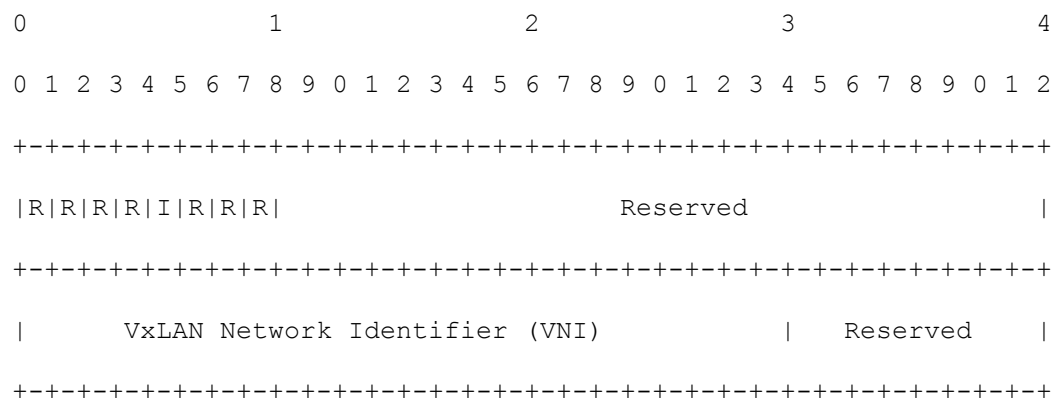
内部的数据包被 VM-B 接收，这是正确的目标。

## 6.4 VxLAN 数据包

这节解释了 VxLAN 封装的分组格式（这个数据包是取自 VxLAN 草案）

### 6.4.1 VxLAN 数据头

## Packet 6.1



## 标志寄存器 (8 位)

I 标记为一个有效的 VxLAN 网络 ID (VNI) 必须设为 1。保留 7 位作为保留字段并且必须设为 0。

**VxLAN 段 ID/ VxLAN 网络 ID(VNI):**

这是一个 24 位的字段，用于识别个别的 VxLAN 重叠网，交互 VM 坐落在这个网中。VM 在不同的重叠网不能和彼此交互。

保留字段:

(24 位和 8 位) — 为了达到填补目的必须设为 0。

### 6.4.2 外部 UDP 数据头

## Packet 6.2

[illegible]

这个带有源端口外部的 UDP 数据头由 VTEP 提供，并且这个目标端口也是熟知的售方专用 UDP 端口。这个 UDP 校验和应该设为 0。作为每份 VXLAN 草案，如果 UDP 校验和为 0 的话，一个数据包要能被剥离。

### 6.4.3 外部 IP 数据头

## Packet 6.3

```
0                                     1                                     2                                     3                                     4
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|                    Total Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Identification                               |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Time to Live   |     Protocol    |           Header Check sum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                           Outer Source Address (IPv4)                                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```

|               Outer Destination Address               |
+-----+

```

这个外部 IP 数据头包括源 IP/发生 VM 交互的 VTEP 的目标 IP。其他的 IP 数据头跟随典型的 IP 数据头定义，这些可以在 VTEP 获得。

#### 6.4.4 Outer Ethernet Header

Packet 6.4

```

0               1               2               3               4
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-----+
|               Outer Destination MAC Address               |
+-----+
| Outer Destination MAC Address |   Outer Source MAC Address   |
+-----+
|               Outer Source MAC Address               |
+-----+
Optional Ethertype=C-Tag 802.1Q |   Outer.VLAN Tag Information   |
+-----+
|           Ethertype 0x0800           |
+-----+

```

这个外部以太网数据头包括源 VTEP 的源 MAC。在这个数据头的外部目标 MAC 地址可能成为中间第三层路由器的目标 VTEP 的地址。

#### 6.4.5 内部以太网数据头和有效负载

这个是当地的来自源 VM 的以太网数据包（这个需要做 VxLAN 封装）。

## 参考书目

- [1] OpenFlow Specification 1.0.0 : <http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf>
- [2] IPRFC : [www.ietf.org/rfc/rfc791.txt](http://www.ietf.org/rfc/rfc791.txt)
- [3] OSPF RFC : [www.ietf.org/rfc/rfc1583.txt](http://www.ietf.org/rfc/rfc1583.txt)
- [4] BGP RFC : [www.ietf.org/rfc/rfc1771.txt](http://www.ietf.org/rfc/rfc1771.txt)
- [5] TCP RFC : [www.ietf.org/rfc/rfc793.txt](http://www.ietf.org/rfc/rfc793.txt)
- [6] UDP RFC : [www.ietf.org/rfc/rfc768.txt](http://www.ietf.org/rfc/rfc768.txt)
- [7] OpenFlow Shortest Path : <http://staff.science.uva.nl/~delaat/rp/2011-2012/p25/report.pdf>  
: <http://www.dis.uniroma1.it/~demetres/docs/dapsp-full.pdf>
- [8] Quantum API : <http://openvswitch.org/openstack/2011/07/25/openstack-quantum-and-open-vswitch-part-1/>
- [9] SDN overview : <http://www.networkcomputing.com/data-networking-management/searching-for-an-sdn-definition-what-is/240000171>  
: <https://www.opennetworking.org/>
- [10] VxLAN Specification :
- [11] VxLAN Packet Flow :  
[http://www.nvc.co.jp/pdf/product/arista/Arista\\_Networks\\_VxLAN\\_White\\_Paper.pdf](http://www.nvc.co.jp/pdf/product/arista/Arista_Networks_VxLAN_White_Paper.pdf)
- [12] OpenFlow Packets :  
[http://www.openflow.org/wk/images/c/c5/Openflow\\_packet\\_format.pdf](http://www.openflow.org/wk/images/c/c5/Openflow_packet_format.pdf)