**《现代交换原理》实验报告**

**实验名称 软件定义网络（SDN）实验**

**班 级 2022211305**

**学 号 2022211683、2022211124、2022211130**

**姓 名 张晨阳、梁维熙、金建名**

**指导教师 赵学达**

**目录**

[实验一 SDN 基本原理与 OpenFlow 协议分析 1](#_Toc198921165)

[一、 实验目的 1](#_Toc198921166)

[二、实验内容与实验步骤 1](#_Toc198921167)

[2.1. 基础环境搭建 1](#_Toc198921168)

[2.2. 建立Mininet与Ryu连接并抓包分析 1](#_Toc198921169)

[1. 启动Ryu控制器 1](#_Toc198921170)

[2. 利用Mininet工具构建网络拓扑 2](#_Toc198921171)

[3. 利用抓包工具Wireshark分析OpenFlow协议 2](#_Toc198921172)

[三、实验结果分析 4](#_Toc198921173)

[（1）HELLO 4](#_Toc198921174)

[（2）FEATURES\_REQUEST 5](#_Toc198921175)

[（3）FEATURES\_REPLY 5](#_Toc198921176)

[（4）MULTIPART\_REQUEST 6](#_Toc198921177)

[（5）MULTIPART\_REPLY 6](#_Toc198921178)

[（6）FLOW\_MOD 6](#_Toc198921179)

[（7）PACKET\_IN 7](#_Toc198921180)

[（8）PACKET\_OUT 8](#_Toc198921181)

[（9）PORT\_STATUS 8](#_Toc198921182)

[四、实验心得 10](#_Toc198921183)

[实验二 OpenFlow 流表操作实践 11](#_Toc198921184)

[一、 实验目的 11](#_Toc198921185)

[二、 实验内容与实验步骤 11](#_Toc198921186)

[2.1. 启动实验环境 11](#_Toc198921187)

[1. 首先通过执行命令 11](#_Toc198921188)

[2. 打开两个终端，在终端一中通过执行命令启动ryu控制器 11](#_Toc198921189)

[2.2. 实现二层交换 13](#_Toc198921190)

[2.3. 下发流表、流表匹配与数据包处理 14](#_Toc198921191)

[1. 删除s1流表 14](#_Toc198921192)

[2. 连通h1<->h3 14](#_Toc198921193)

[三、 实验结果分析 17](#_Toc198921194)

[四、 实验心得 19](#_Toc198921195)

# 实验一 SDN 基本原理与 OpenFlow 协议分析

# 实验目的

1. 了解软件定义网络控制平面和数据平面分离的原理。
2. 了解mininet网络仿真平台以及开源SDN控制器Ryu的基本原理和功能。
3. 掌握使用mininet平台自主构建一个较为复杂的网络拓扑，并建立与Ryu控制器的连接。
4. 掌握使用Wireshark工具抓取OpenFLow协议报文并进行分析。

# 二、实验内容与实验步骤

## 2.1. 基础环境搭建

1. 下载VMware Worstation虚拟机并安装。
2. 下载本实验提供的压缩包并根据实验指导书进行虚拟机环境的安装。

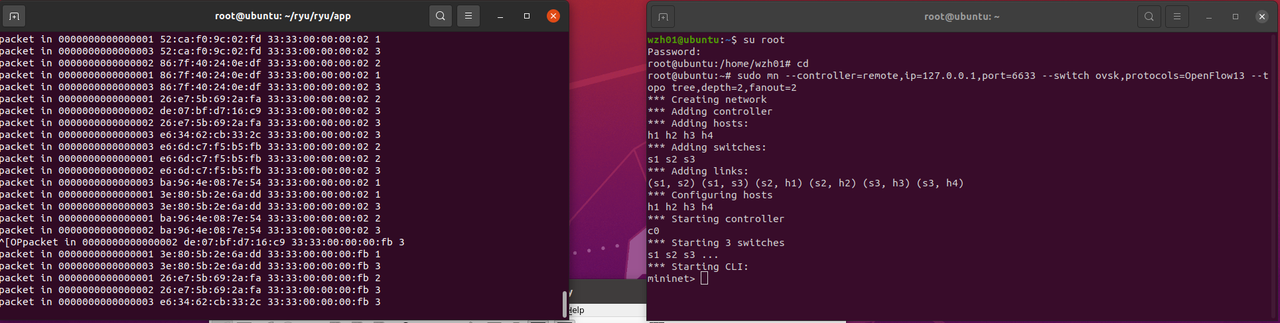
## 2.2. 建立Mininet与Ryu连接并抓包分析

### 启动Ryu控制器

在本实验提供的虚拟机环境中内置了多种控制器应用程序，在我们的实验中使用的是simple\_switch\_13.py，在ryu控制器对应的目录（本实验中的路径为root/ryu/ryu/app）中使用命令：

ryu-manager simple\_switch\_13.py

即可启动控制器。

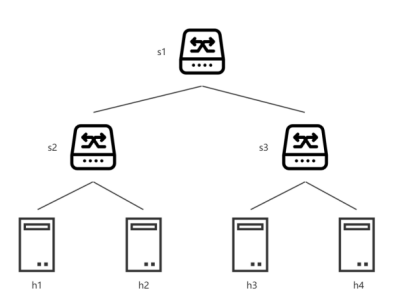


### 利用Mininet工具构建网络拓扑

该工具为模拟网络中的设备，在本实验中我们使用命令：

sudo mn --controller=remote,ip=127.0.0.1,port=6633 --switch ovsk,protocols=OpenFlow13 --topo tree,depth=2,fanout=2

其中，--controller=remote,ip=127.0.0.1,port=6633 的含义为连接到本地运行的Ryu 控制器，即步骤一中创建的控制器。--switch ovsk,protocols=OpenFlow13 的含义为使用 Open vSwitch，并指定使用 OpenFlow 1.3 协议。--topo tree,depth=2,fanout=2 的含义为创建一个树形拓扑网络，深度为2，每个叶子交换机连接两台设备，创建的拓扑网络如图所示。

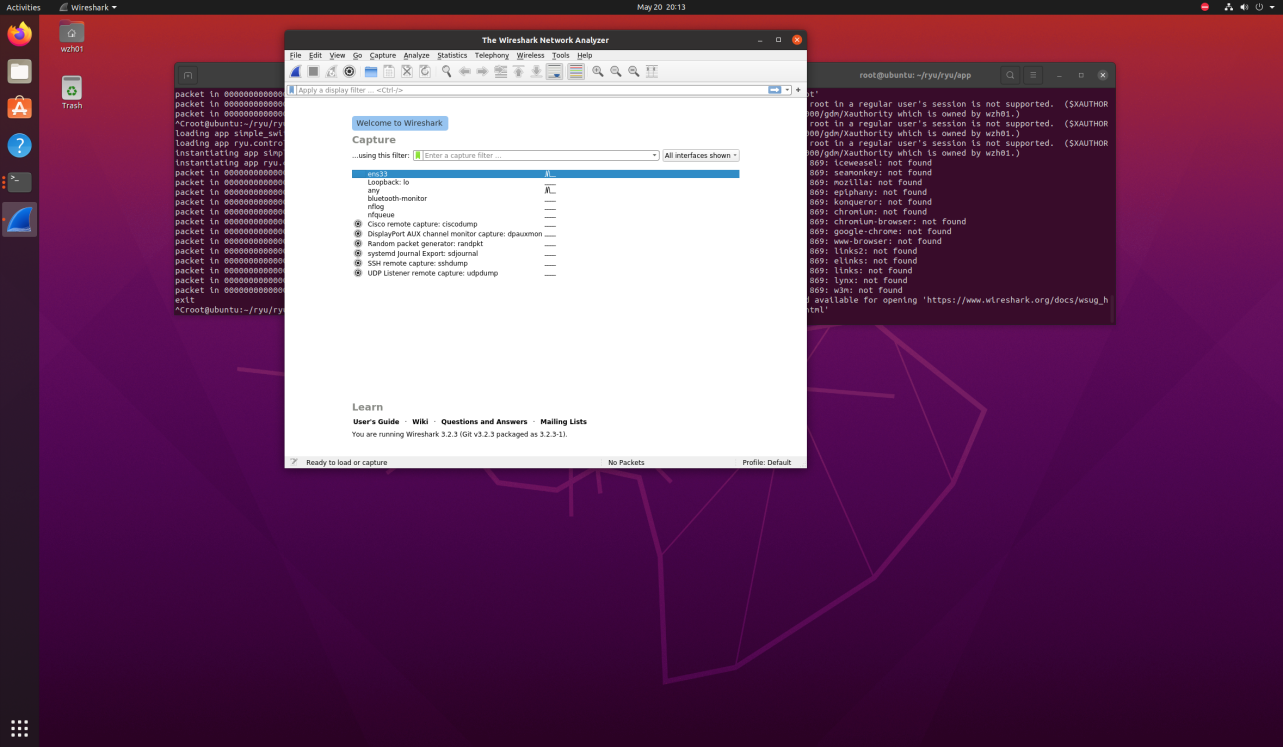


### 利用抓包工具Wireshark分析OpenFlow协议

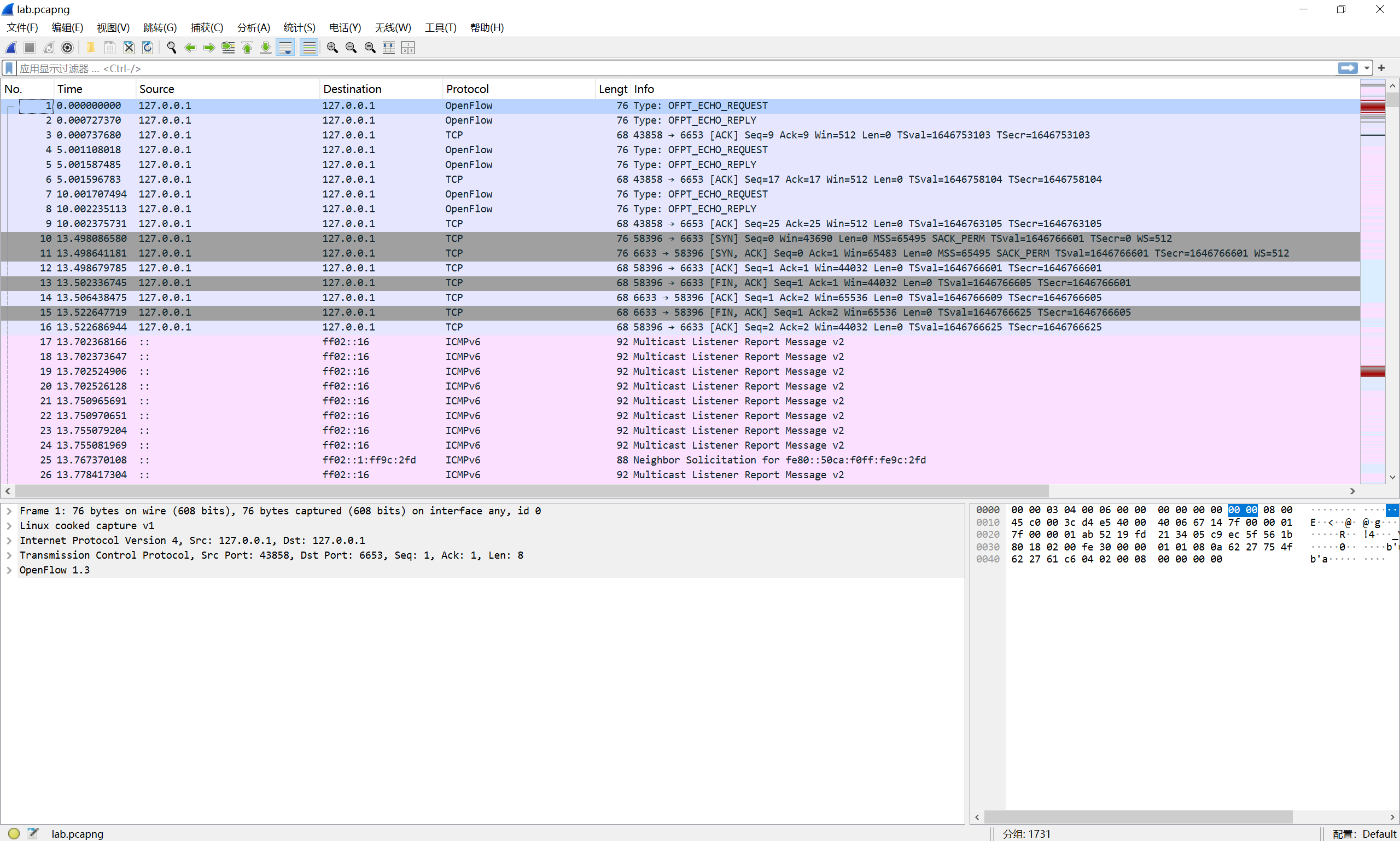
Wireshark为Linux自带的抓包工具，在系统中我们可以通过命令：

sudo wireshark

启动Wireshark进行抓包，启动界面如下图所示：



在Mininet与Ryu建立连接之前，选择Wireshark的any模式进行抓包，抓包得到的结果如下：



# 三、实验结果分析

由于实验中存在三台路由器，一台作为三层交换机，两台台作为二层交换机，二者仅存在一处不同，因此接下来先对三层交换机抓包进行说明：

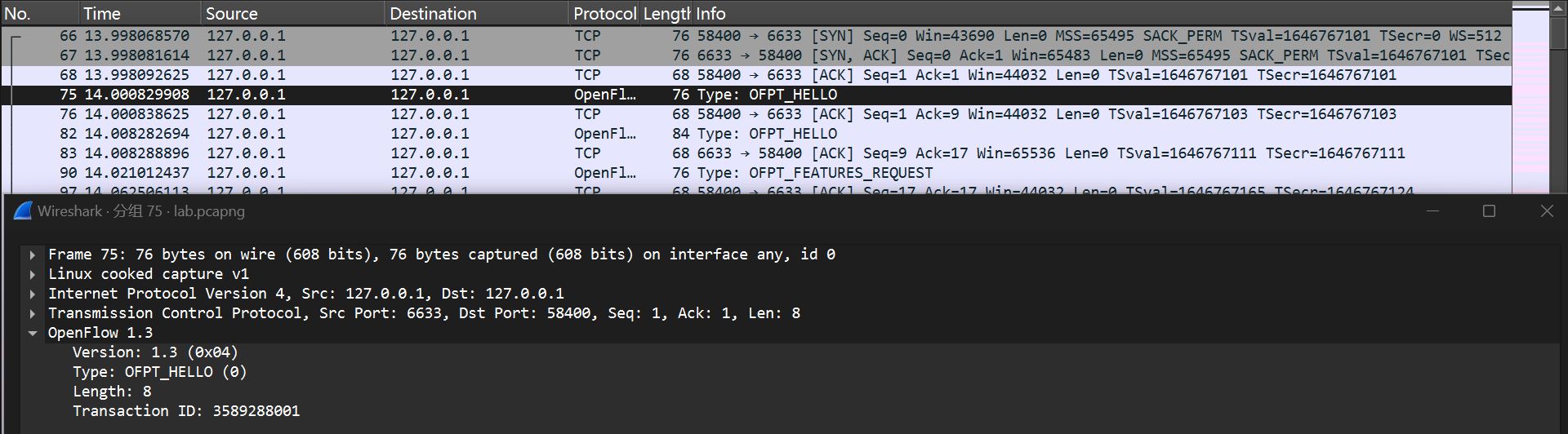
在实验中，三层交换机的端口为58400，因此在wireshark中查找：

tcp.dstport == 58400 || tcp.srcport == 58400

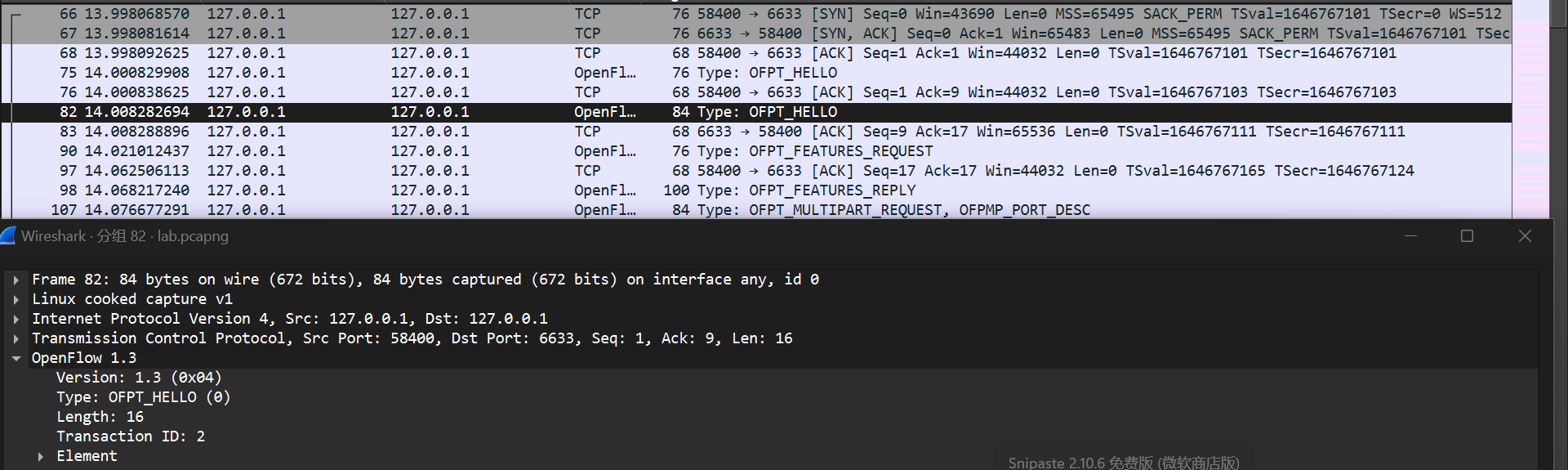
进行筛选。

## （1）HELLO

控制器端口：6633 -> 交换机端口：58400



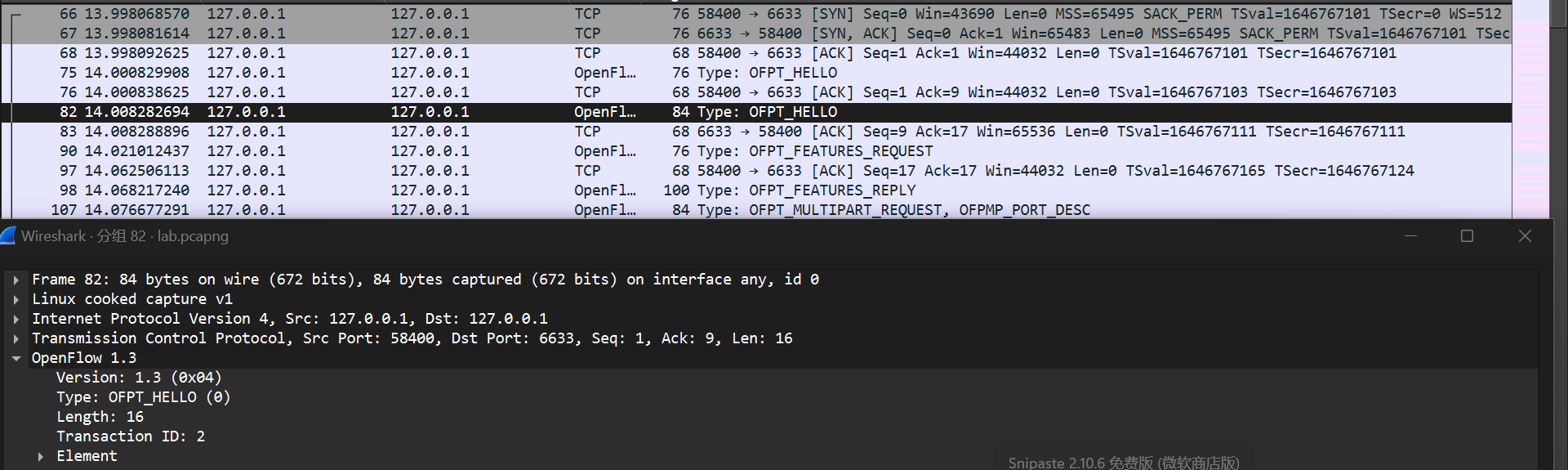
交换机端口：58400 -> 控制器端口：6633



这两个消息用于使交换机和控制器建立连接。

## （2）FEATURES\_REQUEST

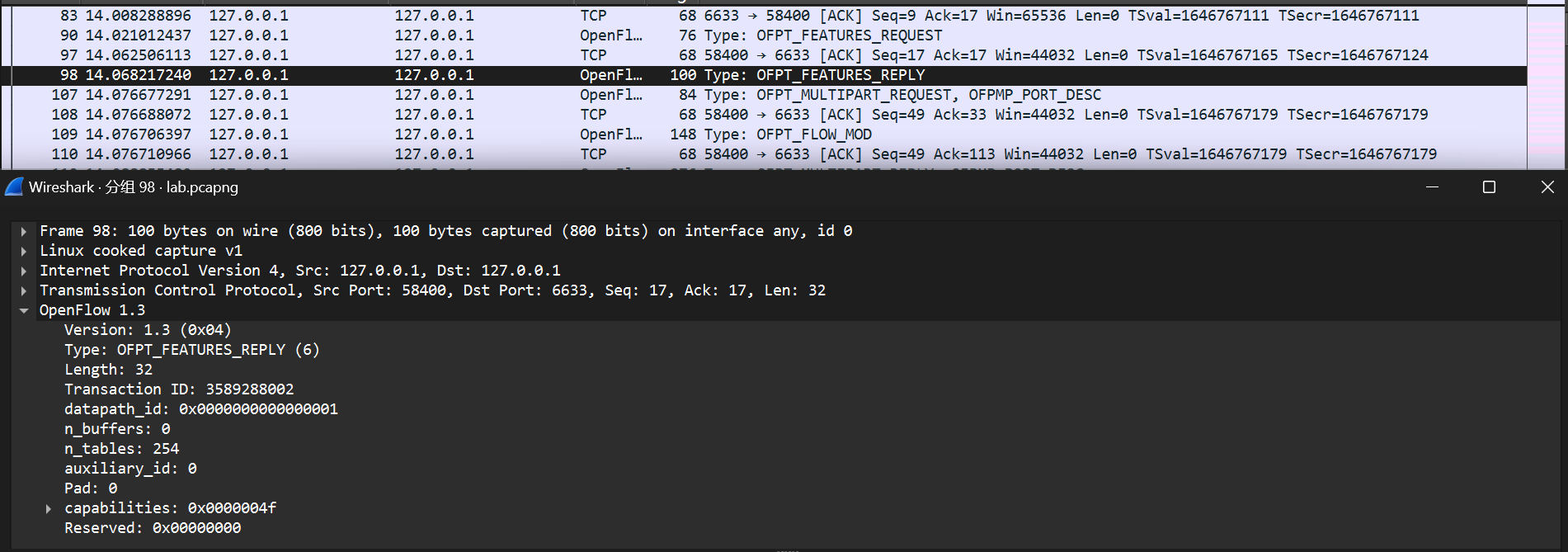
控制器端口：6633 -> 交换机端口：58400



此消息表示控制器需要了解路由器的一些基本资源和配置，因此发送请求。

## （3）FEATURES\_REPLY

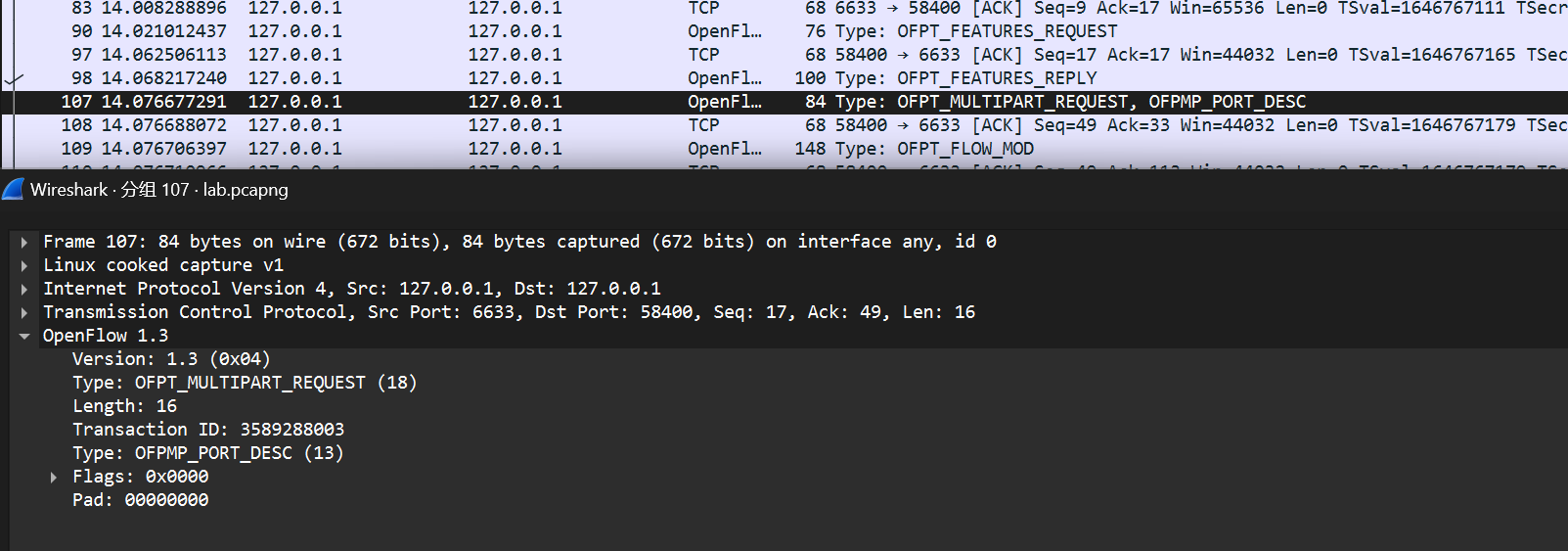
交换机端口：58400 -> 控制器端口：6633



此消息是对消息FEATURES\_REPLY的回复，也就是交换机向控制器发送控制器请求了解的信息。

## （4）MULTIPART\_REQUEST

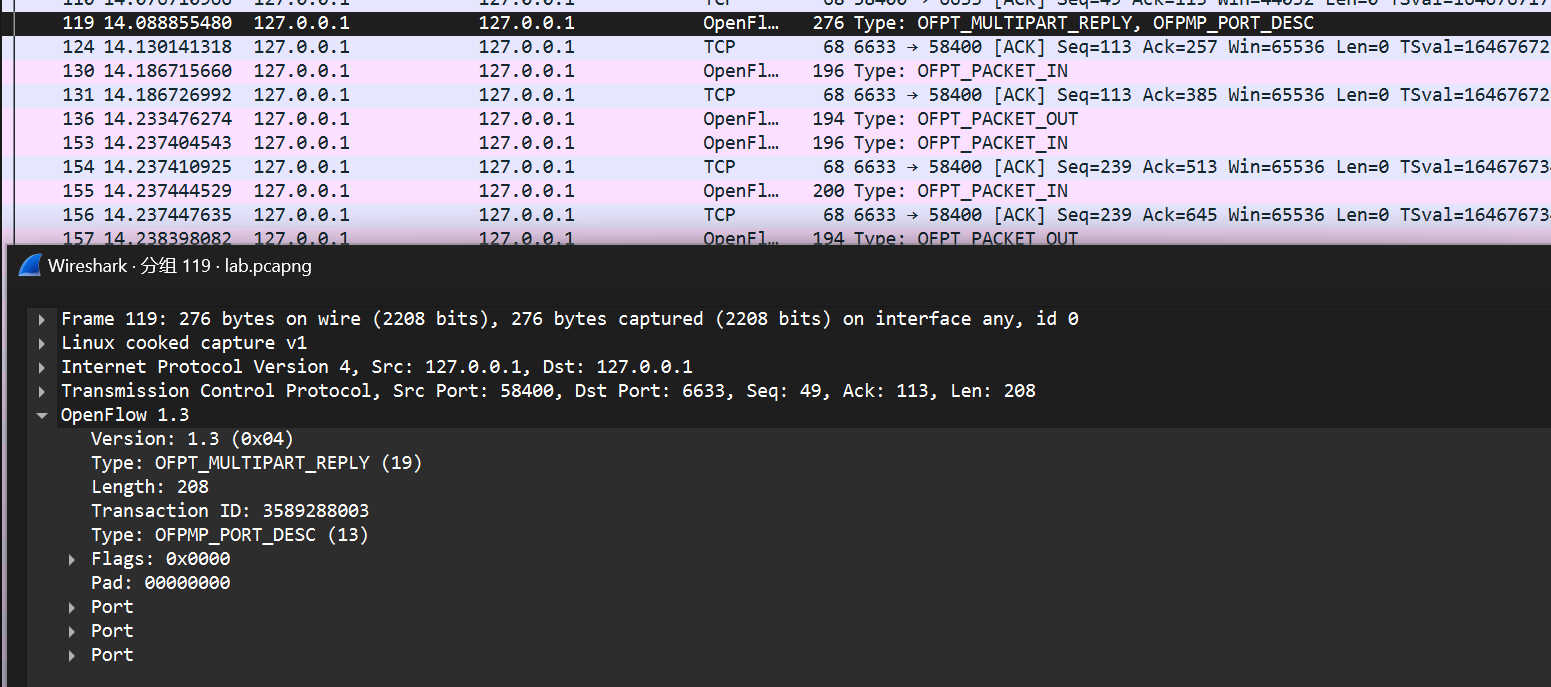
控制器端口：6633 -> 交换机端口：58400



此消息表示控制器需要进一步了解交换机的各种详细信息，因此发送的请求。

## （5）MULTIPART\_REPLY

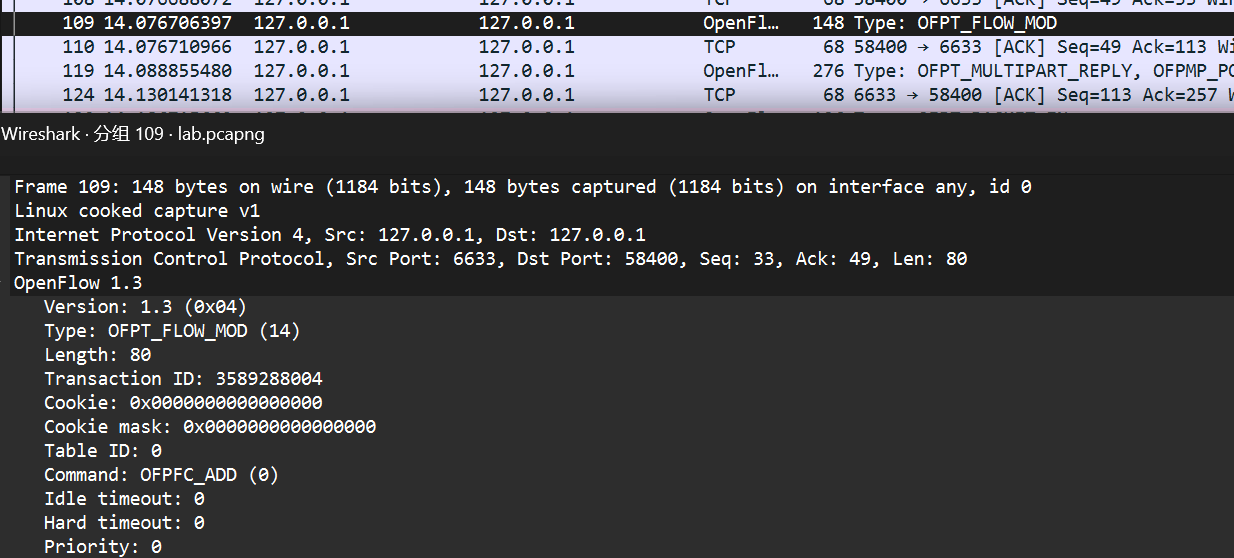
交换机端口：58400 -> 控制器端口：6633



对消息MULTPART\_REQUEST的回复。告知相关信息。

## （6）FLOW\_MOD

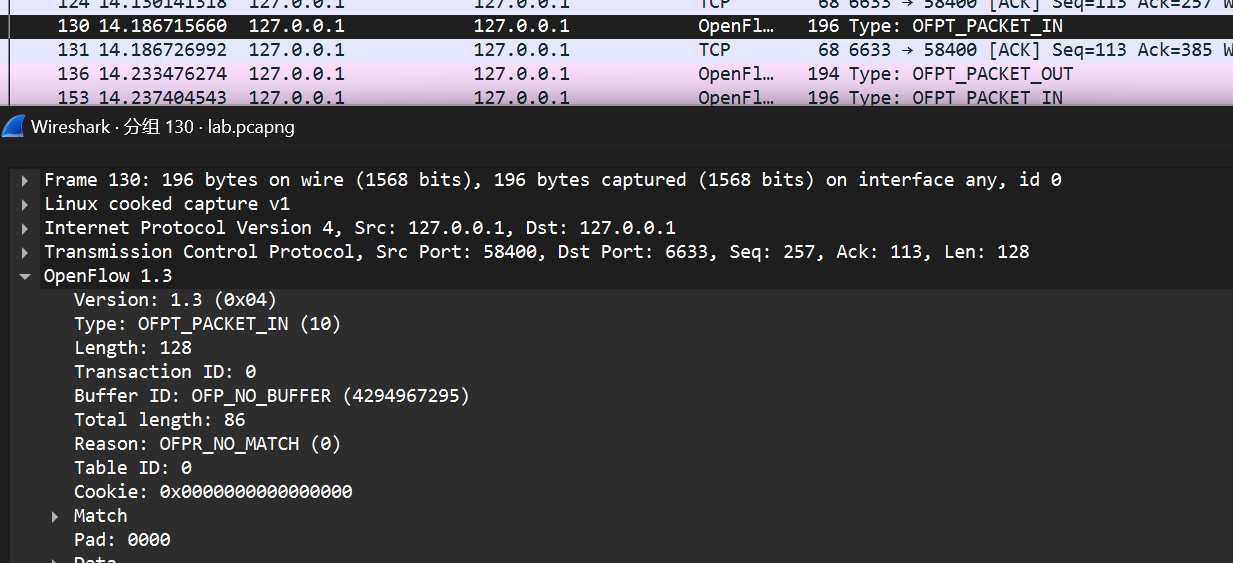
控制器端口：6633 -> 交换机端口：58400



控制器通过此消息指导交换机进行数据转发。

## （7）PACKET\_IN

交换机端口：58400 -> 控制器端口：6633



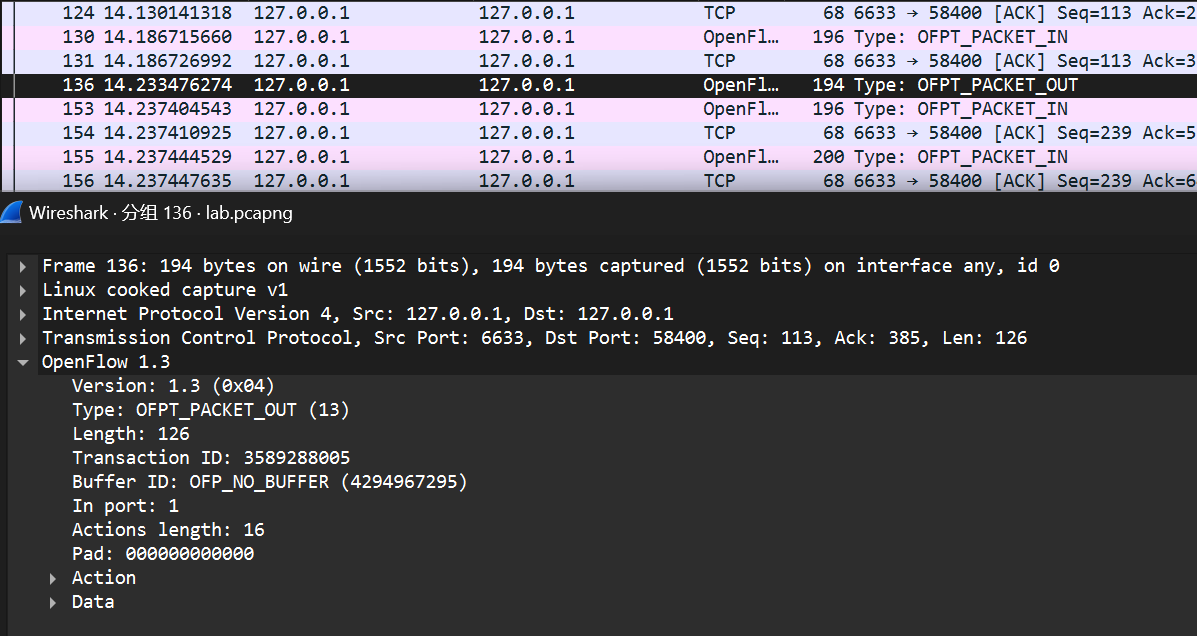
有两种情况：

**Ⅰ：**交换机查找流表，发现没有匹配条目时，向控制器询问如何转发；

**Ⅱ：**有匹配条目但对应的action是OUTPUT=CONTROLLER时。

## （8）PACKET\_OUT

控制器端口：6633 -> 交换机端口：58400

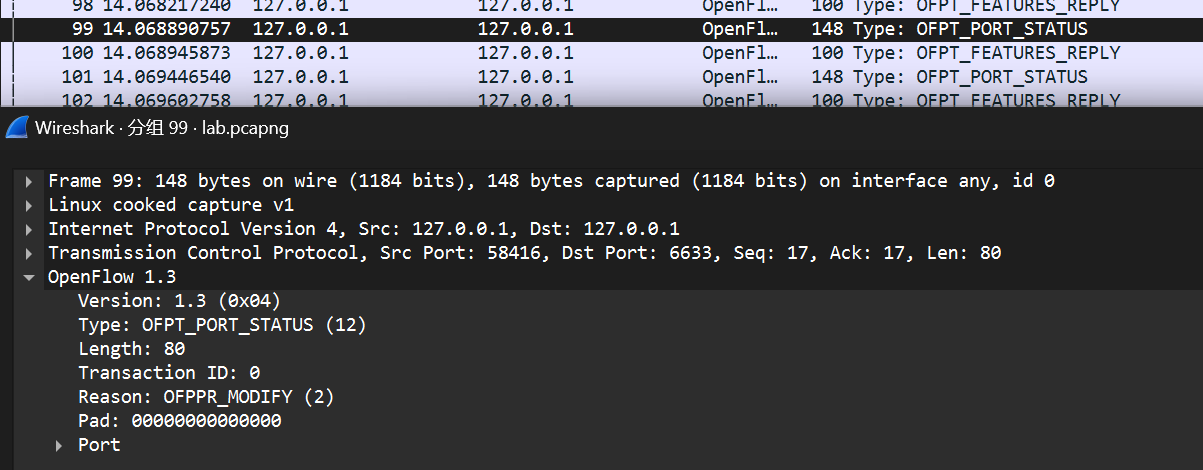


用于控制器指导交换机对数据包处理。

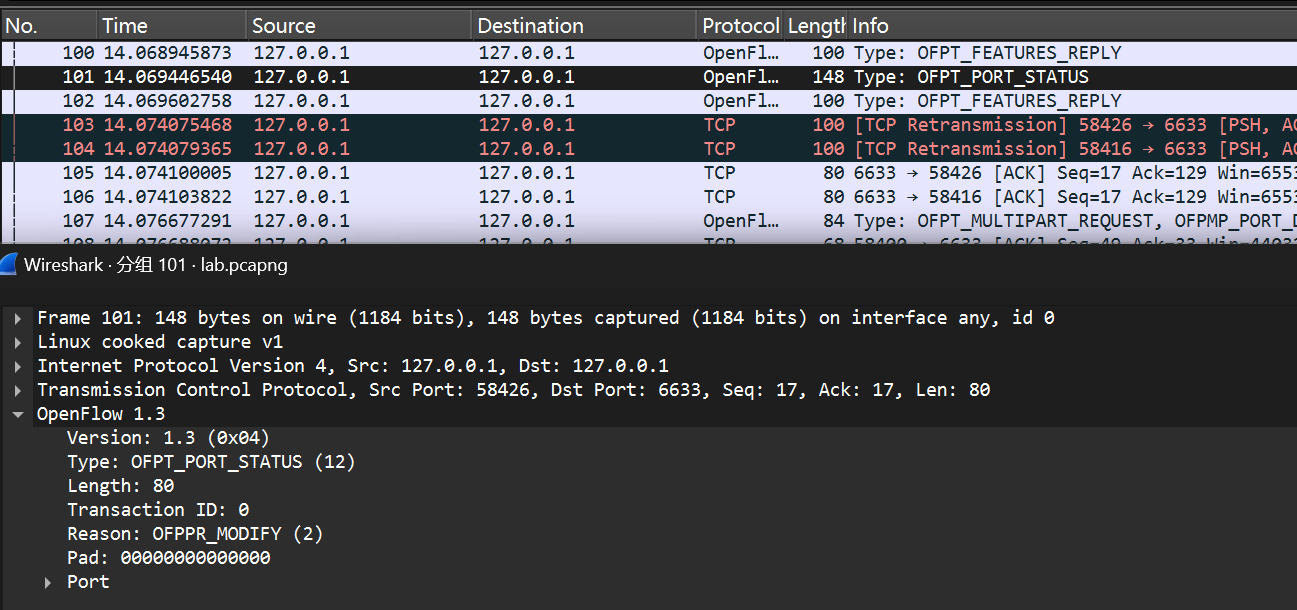
## （9）PORT\_STATUS

在实验过程中，我们在端口为58400的交换机没有发现存在这种包，但在其他两个交换机（端口分别为：58416和58429）的包中发现存在。

**58416：**



**58426：**

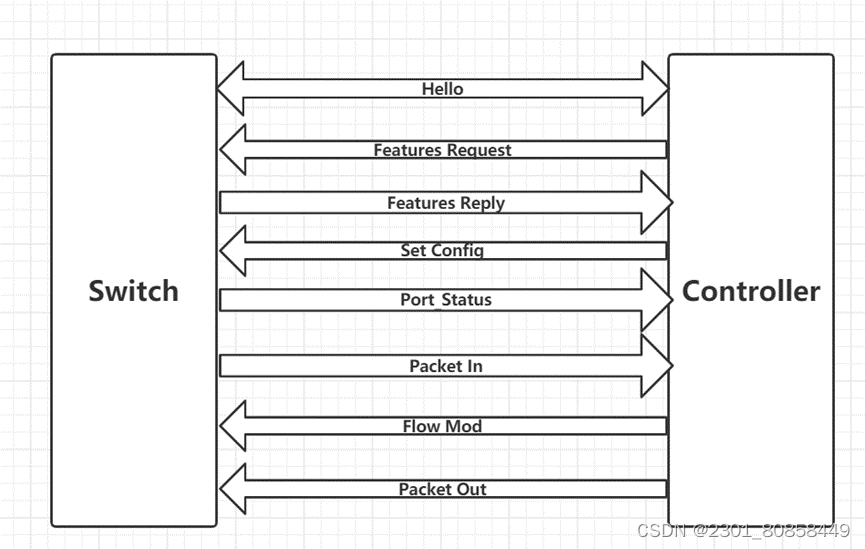


这种包的作用为当交换机端口发生改变时，告知控制器相应的端口状态。

在我们分析之后，发现只有两个二层交换机才存在这种包的原因是：

控制器需要直接与三台交换机连接，但我们的拓扑结构中还需要两个二层交换机各自选出一个空闲端口与三层交换机相连，这就导致每个二层交换机中各有一个端口的状态发生变化，因此向控制器发送这种包。

**交互图：**



# 四、实验心得

我们在使用VMware Workstation时遇到了以下问题：



但经过资料查阅等方法，都成功解决了。

本次实验我们学习了SDN和OpenFlow协议的基本原理，通过使用Ryu和mininet这些与网络编程相关的python包构建出一个虚拟网络拓扑，再使用wireshark对实验网络包进行全程追踪。

# 实验二 OpenFlow 流表操作实践

# 实验目的

1. 了解OpenFlow协议的基本原理及流表结构。
2. 熟悉流表项的增删改查操作及其对数据转发的影响。
3. 通过主机间连通状况验证流表行为。

# 实验内容与实验步骤

## 启动实验环境

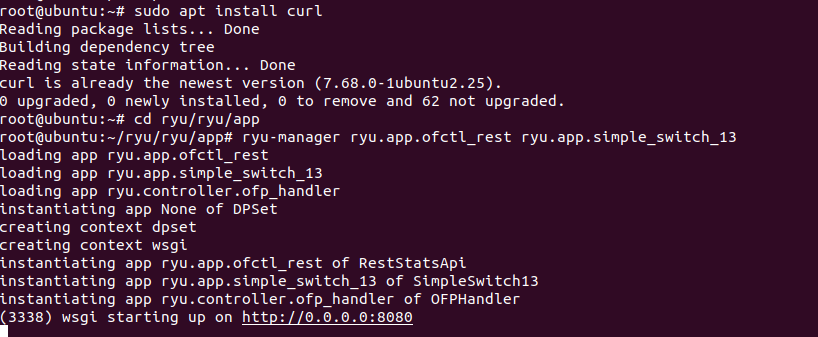
### 首先通过执行命令

sudo apt install curl

安装开源命令行工具curl，该工具通过各类网络协议传输数据，支持get、post、delete等http方法，本实验中用于进行下发、查看、删除流表。

### 打开两个终端，在终端一中通过执行命令启动ryu控制器

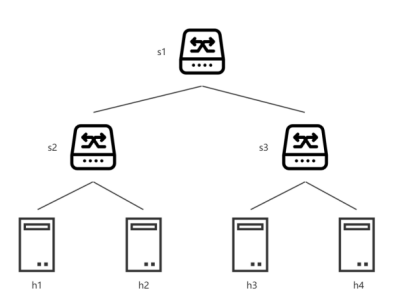
ryu-manager ryu.app.ofctl\_rest ryu.app.simple\_switch\_13



在终端二中执行以下命令创建一个二层高的、每个叶子节点有两台主机的树形拓扑网络：

sudo mn --controller remote --topo tree,depth=2

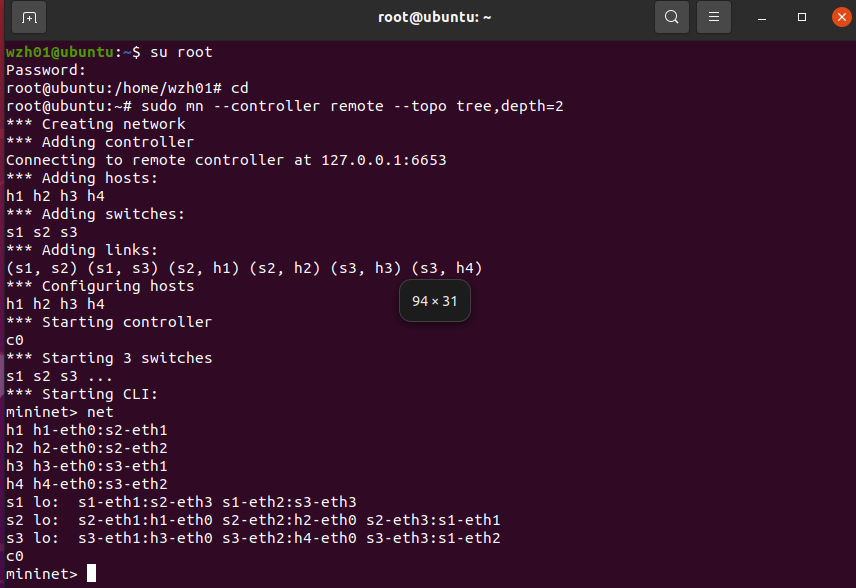
其中，--topo中的参数tree表示创建的网络为树形，depth=2表示树形网络的高度为2，其网络拓扑结构如图所示。



然后，我们使用以下命令检查拓扑网络中各个交换机与主机的连接情况：

net

得到的结果如下：



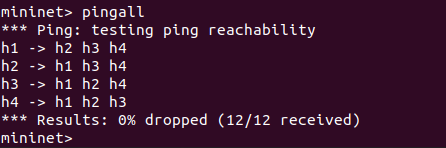
输出表明h1、h2与s2相连，h3、h4与s3相连，s2、s3与s1相连，与预期树形拓扑网络相同。

## 实现二层交换

首先，我们使用以下命令进行各个节点之间的通信测试

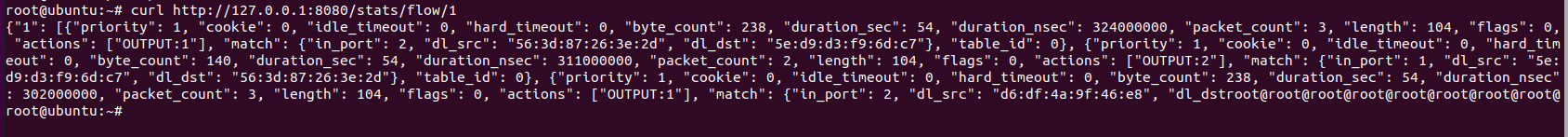
pingall

得到如下结果：

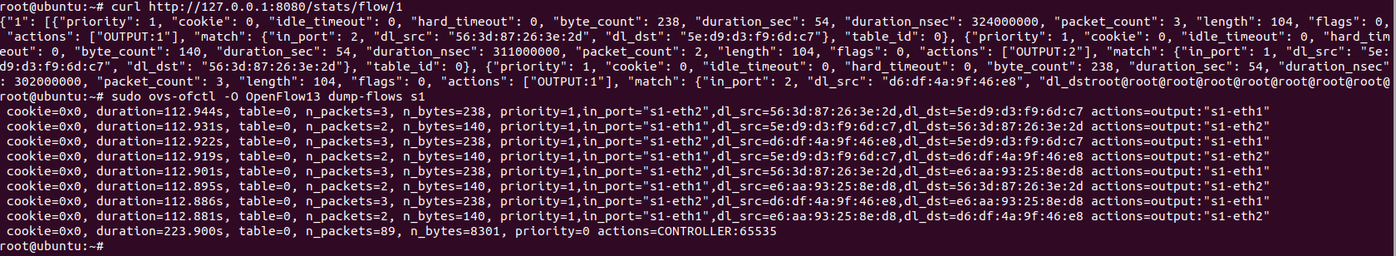


结果说明所有主机之间可以正常通信，这是因为Ryu的simple\_switch\_13中已经实现了二层自学习交换功能，无需额外配置。在新建的终端三中分别使用以下两种命令查看s1的流表，得到结果：

curl http://127.0.0.1:8080/stats/flow/1



sudo ovs-ofctl -O OpenFlow13 dump-flows s1



我们可以发现s1已经自动学习了MAC地址。

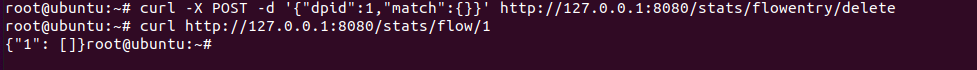
## 下发流表、流表匹配与数据包处理

### 删除s1流表

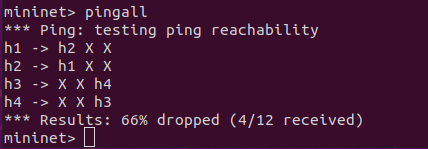
我们在终端中执行以下命令删除s1中的流表：

curl -X POST -d '{"dpid":1,"match":{}}' http://127.0.0.1:8080/stats/flowentry/delete

然后使用以下命令再次查看s1的流表：



输出表明s1的流表成功删除。然后我们再次使用pingall命令进行主机之间的通信测试，得到结果如下：

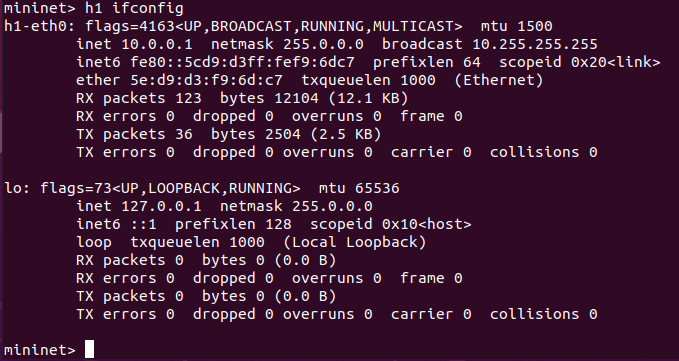


结果说明此时h1与h2相通，h3与h4相通。

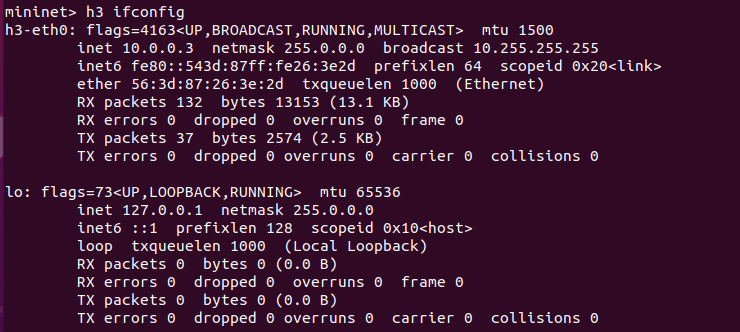
### 连通h1<->h3

首先，我们在终端二中使用以下两条命令查看h1和h3的MAC地址：

h1 ifconfig



h3 ifconfig



然后，我们使用curl工具给s1添加一个流表，流表的源地址为h1，目的地址为h3：

curl -X POST -d '{

"dpid": 1,

"priority": 100,

"match": {

"in\_port": 1,

"dl\_src": "5e:d9:d3:f9:6d:c7",

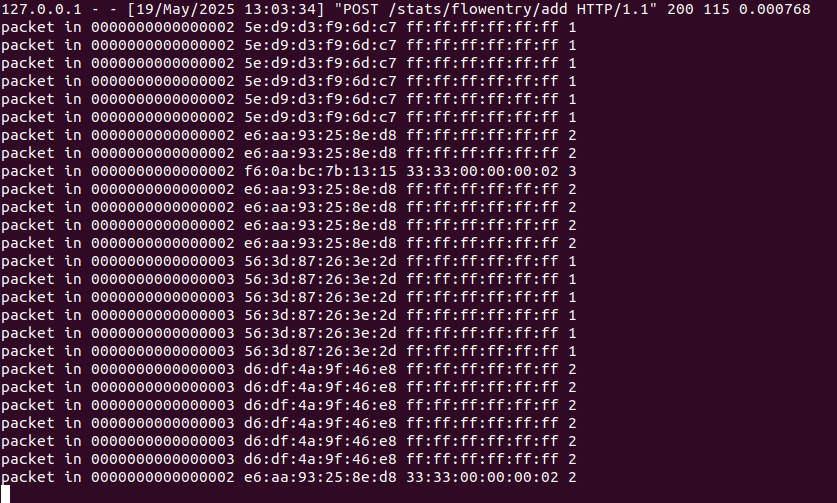
"dl\_dst": "56:3d:87:26:3e:2d"

},

"actions": [{"type": "OUTPUT", "port": 2}]

}' http://127.0.0.1:8080/stats/flowentry/add

终端一中的ryu终端输出如下：



然后，我们使用以下命令给s1分别添加h1广播与h3->h1的流表：

curl -X POST -d '{

"dpid": 1,

"priority": 100,

"match": {

"in\_port": 1,

"dl\_src": "5e:d9:d3:f9:6d:c7",

"dl\_dst": "ff:ff:ff:ff:ff:ff"

},

"actions": [{"type": "OUTPUT", "port": 2}]

}' http://127.0.0.1:8080/stats/flowentry/add

curl -X POST -d '{

"dpid": 1,

"priority": 100,

"match": {

"in\_port": 2,

"dl\_src": "56:3d:87:26:3e:2d",

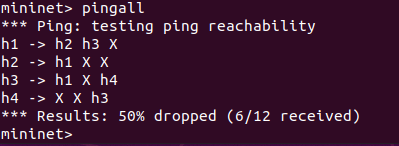
"dl\_dst": "5e:d9:d3:f9:6d:c7"

},

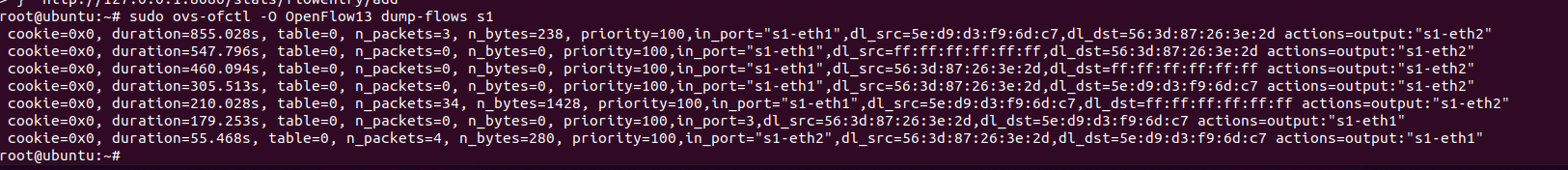
"actions": [{"type": "OUTPUT", "port": 1}]

}' http://127.0.0.1:8080/stats/flowentry/add

此时，我们再次进行pingall操作，可以发现h1与h3之间能够通信。



再次检查s1中的流表项：



# 实验结果分析

**两种查看流表的方式有什么不同？这些流表表示了什么信息？**

**1. 输出格式方面**

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

输出格式较为直观和简洁，通常以文本形式直接显示流表的详细信息，包括流表的优先级、匹配字段（如源地址、目的地址、协议类型等）、动作（如转发到某个端口、丢弃等）等，这些信息是按照 OpenFlow 协议规定的格式直接从交换机中读取并展示出来的，方便网络管理员快速查看和理解流表的配置情况。

curl http://127.0.0.1:8080/stats/flow/1

输出格式是 JSON 格式。JSON 格式是一种轻量级的数据交换格式，它以键值对的形式组织数据。在 Mininet 中，当使用 Ryu 控制器等基于 REST API 的控制器时，通过该命令获取的流表信息会以 JSON 格式返回。例如，它可能会包含多个键，如 “flows” 键下包含了流表的详细信息，其中包括流表的 id、cookie、priority 等字段，以及每个流表的 match 条件和 actions 等，这种格式便于程序进行解析和处理，适合用于开发自动化脚本或与其他基于 JSON 的系统进行交互。

**2. 获取流表的途径方面**

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

通过 OpenFlow 协议直接与交换机进行通信来获取流表信息的。它利用了 OpenFlow 协议中的消息机制，向交换机发送请求消息，交换机接收到请求后，会将流表信息以 OpenFlow 消息的形式返回，这种方式是基于 OpenFlow 协议的直接交互，不依赖于其他中间件或控制器。

curl http://127.0.0.1:8080/stats/flow/1

通过与控制器（如 Ryu 控制器）的 REST API 进行交互来获取流表信息的。控制器会维护一个流表的视图，当收到该 curl 请求时，控制器会从自己的内部数据结构中获取流表信息，并将其转换为 JSON 格式返回给请求方。这种方式依赖于控制器的正常运行以及控制器与交换机之间的通信，控制器起到了中间代理的作用，它从交换机获取流表信息后进行存储和管理，然后通过 REST API 向外部提供这些信息。

**请解释这种现象的原因。**

由于创建的拓扑网络为树形结构，所以与s2相连的主机要与s3相连的主机需要经过s1进行转发，但是我们通过命令将s1中的流表删除，此时s1不知道如何进行通信包的转发，所以两侧的主机不能够进行相连。但是与s2直接相连的两台主机只需要经过s2进行转发即可进行通信，所以二者之间能够ping通；与s3直接相连的两台主机能够ping通原因相同。

**此时尝试 ping，应 ping 不通。为什么？**

ping命令通过发送ICMP包到目标地址并接受从目标地址发回的ICMP包进行分析判断两个地址之间能否连通，我们在s1中配置了h1->h3的流表，但是并未配置h3->h1的流表，即使h3通过ARP的方式获取到了h1的MAC地址，s1交换机不会将h3回复的ICMP包转发给h1。

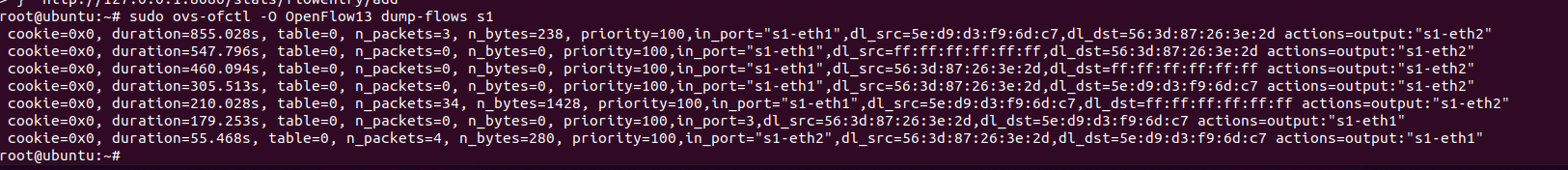
**为什么仅设置了 h1 广播的流表项，未设置 h3 广播的流表项，h3->h1 仍能 ping 通？**

h3首先使用ARP广播包请求h1的MAC地址，虽然我们在s1中没有配置h3广播的流表项，但是在SDN环境中，交换机的默认行为通常是由控制器配置的流表决定的。如果流表中没有明确禁止某种类型的帧（如广播帧）的转发，交换机会根据帧的类型（如广播帧）进行默认处理。对于广播帧，交换机会将其转发到所有其他端口，以确保所有设备都能接收到该帧。所以该ARP包能够正确转发至h1，使h1能够构建ARP回复包告诉h3自己的MAC地址。

# 实验心得

**遇到的问题与解决方案**

1. 在配置流表时尝试使用其他端口（例如端口3）作为目标端口连通失败：



经过排查，我们发现在mininet自动生成树形拓扑结构时，s1中给s2、s3分配的端口分别为1和2，所以在配置流表时使用其他端口会导致配置失败。

通过这次实验，我深刻理解了如何利用 Ryu 控制器、Mininet 以及 curl 工具来构建软件定义网络（SDN）。这次实践不仅加深了我对网络交换机运作机制的认识，还让我掌握了搭建 SDN 环境的基本技能。

在实验过程中，我们遇到了一些挑战，但通过查阅相关资料和进行故障排查，我们最终成功解决了这些问题。这一过程不仅增强了我们的问题解决能力，还提高了我们的故障排查技能。这些经验对于我们未来在网络领域的学习和工作都具有重要的意义。

通过这次实验，我更加坚信软件定义网络是未来网络发展的重要方向。它为我们提供了更灵活、更高效的网络管理方式，使我们能够更好地应对不断变化的网络需求。我期待在未来的学习和工作中，能够继续探索和应用 SDN 技术，为网络的发展做出自己的贡献。