

# 软件定义网络 (SDN) 实验 指 导 书 (学生用书)

北京邮电大学计算机学院 (国家示范性软件学院)

现代交换原理课程组

2025 年 5 月

# 实验一 SDN 基本原理与 OpenFlow 协议分析

## 一、实验目的

1. 了解软件定义网络控制平面和数据平面相分离的原理。
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、实验要求

1. 完成 Mininet 及 Ryu 环境配置, 了解软件定义网络控制平面和数据平面相分离的具体原理, 了解 Mininet 及 Ryu 各自的基本原理与功能, 以及两者之间连接的原理, 了解 OpenFlow 协议的原理。
2. 建立 Mininet 与 Ryu 的连接并抓包分析 OpenFlow 协议。4.2 节给出了一个基础示例, 通过创建简单的网络拓扑并建立连接, 为大家提供基本的实验步骤说明。  
**请大家自主构建一个相对复杂的网络拓扑, 至少包含 3 个以上的交换机, 并针对这个网络进行连接的建立, 以及后续的抓包和协议分析。**

## 三、实验环境及主要工具介绍

### 3.1 实验环境

虚拟机软件: VMware 17.6

Linux 系统: Ubuntu 20.04.6

Python: python 3.8.10

网络仿真平台: mininet 2.3.1b1

SDN 控制器: ryu 4.34

## 3.2 主要工具介绍

### ➤ Mininet 网络仿真平台

Mininet 是一个轻量级的网络仿真平台，通过在单个 Linux 系统上运行虚拟网络来模拟真实网络环境。Mininet 使用 Linux 的命名空间和虚拟以太网接口来创建虚拟网络设备，从而实现网络拓扑的仿真。具体而言，Mininet 利用命名空间为每个虚拟主机和交换机创建独立的网络环境，通过虚拟以太网接口连接虚拟主机和交换机，通过其之间的数据包传输来模拟物理链路，从而构建网络拓扑。

Mininet 在本实验中的主要功能为构建和管理虚拟网络拓扑。该仿真平台的具体功能包括：

- (1) 支持命令行和可视化界面两种方式，构建包含线性拓扑、树形拓扑等多种拓扑结构；
- (2) 提供命令行接口以管理网络设备、查看网络状态、执行网络操作等；
- (3) 支持使用包含 ierf 在内的多种工具进行网络性能测试；
- (4) 可以连接到本地或远程的 SDN 控制器，如本实验使用的 Ryu。

### ➤ Ryu 控制器

Ryu 为一个基于 Python 编写的开源 SDN 控制器。遵循软件定义网络 SDN 的控制平面和数据平面相分离原则，通过 OpenFlow 协议与网络设备（如交换机）进行通信。Ryu 采用事件驱动的架构，通过监听和处理各种网络事件（如数据包到达、端口状态变化等）来实现网络的动态管理和控制。

Ryu 控制器在本实验中的主要作用为，通过 OpenFlow 协议与网络中的交换机进行通信，实现 SDN 中的控制平面。该控制器的具体功能有：

- (1) 发现和管理 SDN 网络中的拓扑结构，包含主机、交换机、链路和路径

等信息；

(2) 控制网络流量的转发和策略，支持流表等操作，以实现流量控制（支持通过 OpenFlow 协议动态添加、删除和查询流表项）；

(3) 支持开发者编写自定义的 SDN 应用程序，通过 Ryu 的 API 与网络设备进行交互。

## 四、实验步骤

### 4.1 基础环境搭建

(1) 下载 VMware Workstation 虚拟机

按步骤安装并激活虚拟机软件。

(2) 环境搭建

**本实验前期配置环境的过程稍显繁琐，由于版本不统一、环境差异等问题，会导致大家花费较多时间在配环境上。为了便于大家在有限的时间内更好地专注于分析 OpenFlow 协议、了解并掌握流表下发等操作，在此向大家提供一个已经配置好 Mininet 和 Ryu 的虚拟机环境，帮助大家更便捷地掌握关键知识。但仍建议大家在集中实验之后，可以自己从创建新的虚拟机开始，按步配置实验环境，以加深大家对网络仿真平台和 SDN 控制器的理解和掌握程度。**

**使用已有的虚拟机环境步骤如下：**

【压缩包链接：[https://pan.baidu.com/s/1GW5Mr1\\_n1G0q3Zkzb8nLjw?pwd=jj7d](https://pan.baidu.com/s/1GW5Mr1_n1G0q3Zkzb8nLjw?pwd=jj7d)  
提取码: jj7d】

将提供给大家的“虚拟机”压缩包内的以下三个文件（如图 1.1 所示）复制到个人电脑的一个目录中。



 SDN.mf	2025/5/8 18:15	MF 文件
 SDN.ovf	2025/5/8 18:15	开放虚拟化格式程
 SDN-disk1.vmdk	2025/5/8 18:14	360zip

图 1.1 压缩包内文件

打开 VMware Workstation，点击菜单栏中的“文件” - “打开”，选择目录中的 SDN.ovf 文件并打开，如图 1.2 所示。

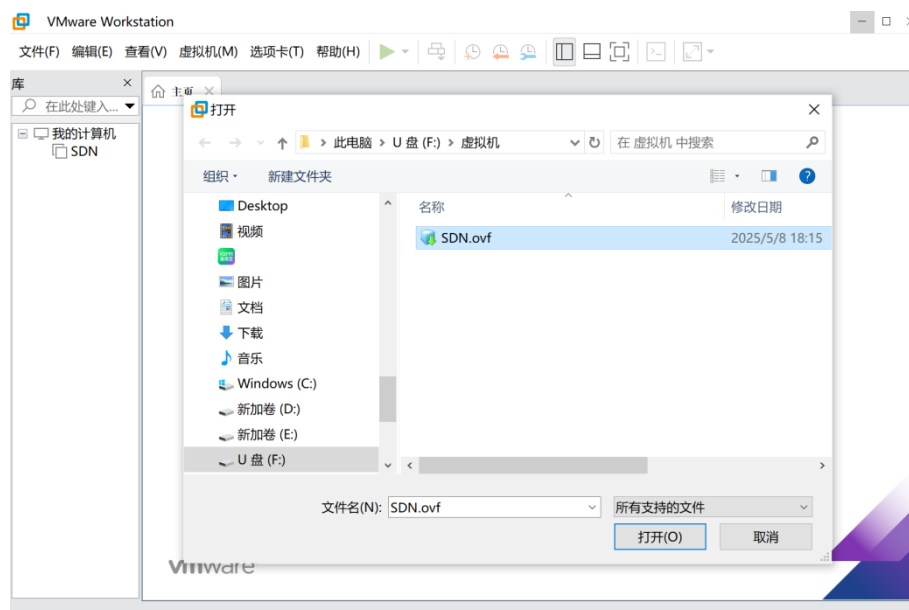


图 1.2 打开已有虚拟机 ovf 文件

如图 1.3 所示，在本地导入虚拟机。



图 1.3 导入虚拟机至本地

点击导入后，很快就能完成虚拟机的创建。选择虚拟机并点击“开启此虚拟机”后大概率会出现全黑的界面，此时正在自动配置各项环境，请耐心等待，此过程一般持续 5-8 分钟左右。

等待结束后，如图 1.4，大家可顺利进入虚拟机。此虚拟机的密码为：**asdf1122**。

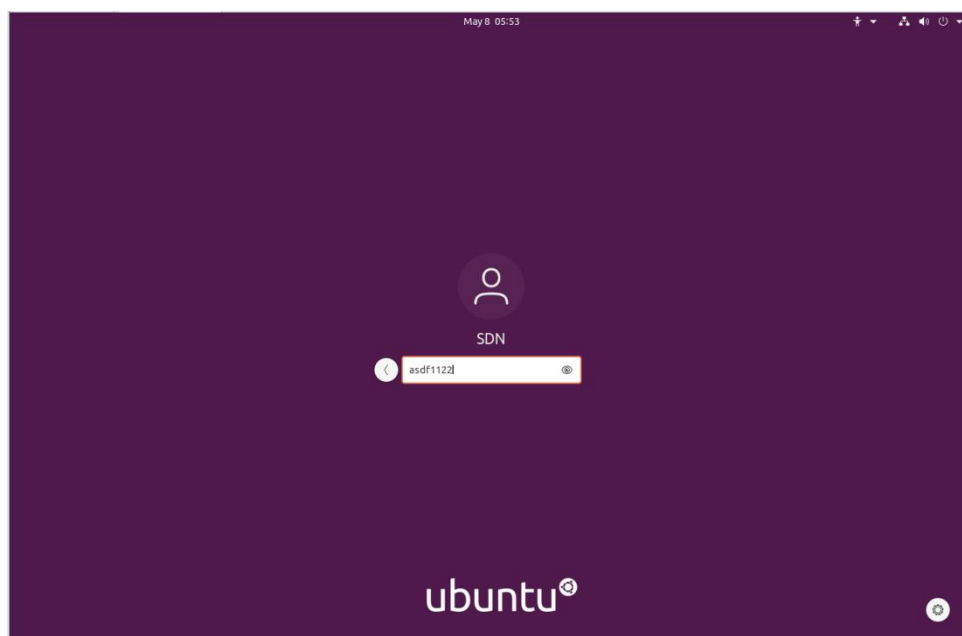


图 1.4 虚拟机起始界面

Mininet 和 Ryu 控制器均安装在 root 目录下, 可使用 `su root` 命令切换到 root 用户, 此过程中的密码仍为: **asdf1122**。root 目录内的内容如图 1.5 所示。

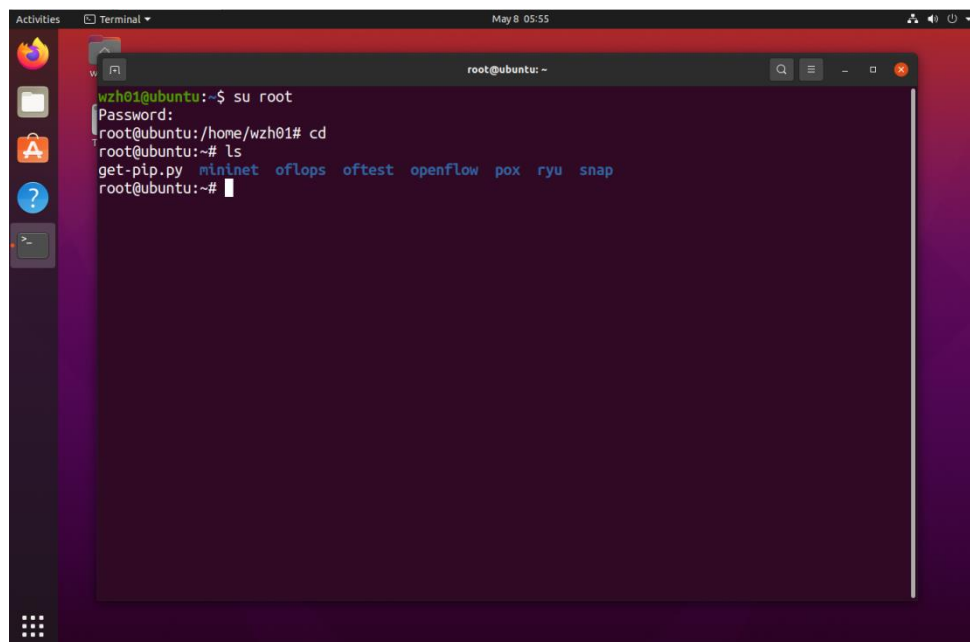


图 1.5 root 目录

## 4.2 建立 Mininet 与 Ryu 连接并抓包分析——基础示例

请注意本节给出的步骤为示例, 请大家自主构建一个更为复杂的网络拓扑 (至少包含 3 个以上的交换机) 进行本实验, 详见 4.3 节。

### 步骤一: 启动 Ryu 控制器

(1) Ryu 内置的控制器应用程序介绍:

如图 1.6 所示, 提供了多种内置的控制器应用程序 (如 `simple_switch_13.py`、`simple_switch.py`), 不同程序在 OpenFlow 协议版本、实现细节等方法存在差异。

```
root@ubuntu:~/ryu/ryu/app# ls
atuo_flow.py          rest_firewall.py      simple_switch_lacp_13.py
auto_flow.py          rest_qos.py           simple_switch_lacp.py
bmpstation.py         rest_router.py        simple_switch.py
cbench.py             rest_topology.py      simple_switch_rest_13.py
conf_switch_key.py    rest_vtep.py          simple_switch_snort.py
example_switch_13.py  simple_monitor_13.py  simple_switch_stp_13.py
gui_topology          simple_switch_12.py    simple_switch_stp.py
__init__.py           simple_switch_13.py    simple_switch_websocket_13.py
ofctl                 simple_switch_14.py    wsgi.py
ofctl_rest.py         simple_switch_15.py    ws_topology.py
__pycache__           simple_switch_igmp_13.py
rest_conf_switch.py   simple_switch_igmp.py
```

图 1.6 Ryu 内置的控制器应用程序

其中, `simple_switch_13.py` 是一个简单的 L2 学习交换机, 基于 OpenFlow 1.3 协议。展示了如何使用 Ryu 实现一个基本的二层交换机, 通过学习 MAC 地址来转发数据包。

## (2) 启动 Ryu 控制器:

在 root 目录下使用 `cd ryu/ryu/app` 命令进入 mininet 的应用程序目录。

如图 1.7 所示, 利用 `ryu-manager simple_switch_13.py` 命令运行 SDN 控制器。

```
root@ubuntu:~/ryu/ryu/app# ryu-manager simple_switch_13.py
loading app simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

图 1.7 启动 Ryu 控制器

## 步骤二: 利用 Mininet 工具构建网络拓扑

### (1) 网络拓扑构建方法介绍:

Mininet 创建网络拓扑包含多种方式, 常见的方法有:

- 方法一: 内置拓扑选项

直接通过命令行参数指定网络拓扑构建, 例如 `sudo mn --topo linear,3` 可创建一个线性拓扑 (包含 3 个主机和 2 个交换机); `sudo mn --topo tree,depth=2,fanout=2` 创建一个树形拓扑 (深度为 2, 每个节点的分支数为 2)。

- 方法二: 使用自定义 Python 脚本

Mininet 支持使用 Python 脚本自定义复杂的网络拓扑, 通过编写 Python 脚本来定义拓扑结构、主机、交换机和链路。



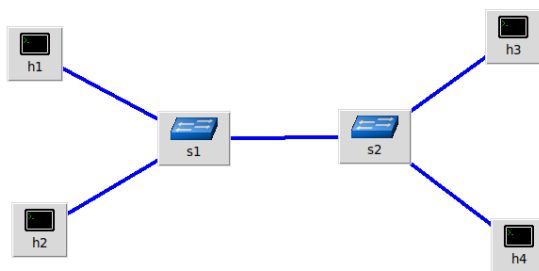


图 1.8 示例拓扑结构

例如，创建如图 1.8 所示的网络拓扑，可创建脚本程序 custom\_topo.py：

```

1. from mininet.topo import Topo
2.
3. class CustomTopo(Topo):
4.     def build(self):
5.         # 添加主机
6.         h1 = self.addHost('h1')
7.         h2 = self.addHost('h2')
8.         h3 = self.addHost('h3')
9.         h4 = self.addHost('h4')
10.
11.        # 添加交换机
12.        s1 = self.addSwitch('s1')
13.        s2 = self.addSwitch('s2')
14.
15.        # 添加链路
16.        self.addLink(h1, s1)
17.        self.addLink(h2, s1)
18.        self.addLink(s1, s2)
19.        self.addLink(s2, h3)
20.        self.addLink(s2, h4)
21.
22. # 创建拓扑实例
23. topo = CustomTopo()

```

`sudo mn --custom custom_topo.py --topo custom` 运行 Mininet 并加载自定义拓扑可

完成如图 1.8 所示的网络拓扑创建。

### ● 方法三：使用 Mininet CLI

Mininet 的命令行界面 (CLI) 支持动态创建和管理网络拓扑。可以在 Mininet CLI 中使用 `add-host`、`add-switch` 以及 `add-link` 等命令来动态添加主机、交换机和

链路。

## (2) 创建一个简单的网络拓扑并建立与 Ryu 控制器的连接：

在本示例中，使用内置拓扑选项方式，在另一终端使用 `sudo mn --controller=remote,ip=127.0.0.1,port=6633 --switch ovsk,protocols=OpenFlow13 --topo single,2` 命令构建一个如图 1.9 所示的简单网络拓扑并建立与前述 Ryu 控制器的连接。

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 single,2` 的含义为创建一个单交换机拓扑，包含两个主机。



图 1.9 简单网络拓扑

如图 1.10 所示，成功建立连接。

```
root@ubuntu:~# sudo mn --controller=remote,ip=127.0.0.1,port=6633 --switch ovsk,
protocols=OpenFlow13 --topo single,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> [2]+ Stopped ryu-manager simple_switch_13.py
root@ubuntu:~/ryu/ryu/app# ryu-manager simple_switch_13.py
loading app simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 0000000000000001 ea:3f:90:e8:7d:e6 33:33:00:00:00:16 2
packet in 0000000000000001 ea:3f:90:e8:7d:e6 33:33:ff:e8:7d:e6 2
packet in 0000000000000001 d6:48:a7:67:11:0f 33:33:00:00:00:16 1
packet in 0000000000000001 d6:48:a7:67:11:0f 33:33:00:00:00:16 1
packet in 0000000000000001 d6:48:a7:67:11:0f 33:33:00:00:00:02 1
packet in 0000000000000001 ea:3f:90:e8:7d:e6 33:33:00:00:00:16 2
```

图 1.10 连接建立

## 步骤三：利用抓包工具 Wireshark 分析 OpenFlow 协议

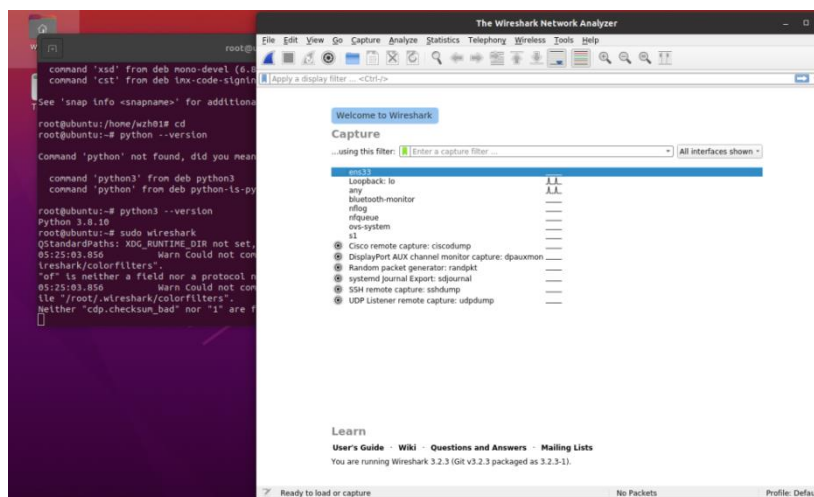


图 1.11 WireShark 工具

如图 1.11 所示, 在新终端使用 `sudo wireshark` 命令打开 Wireshark 抓包工具。

在 Mininet 与 Ryu 建立连接之前, 选择 Wireshark 的 any 模式进行抓包, 抓包结果如图 1.12、图 1.13 所示。

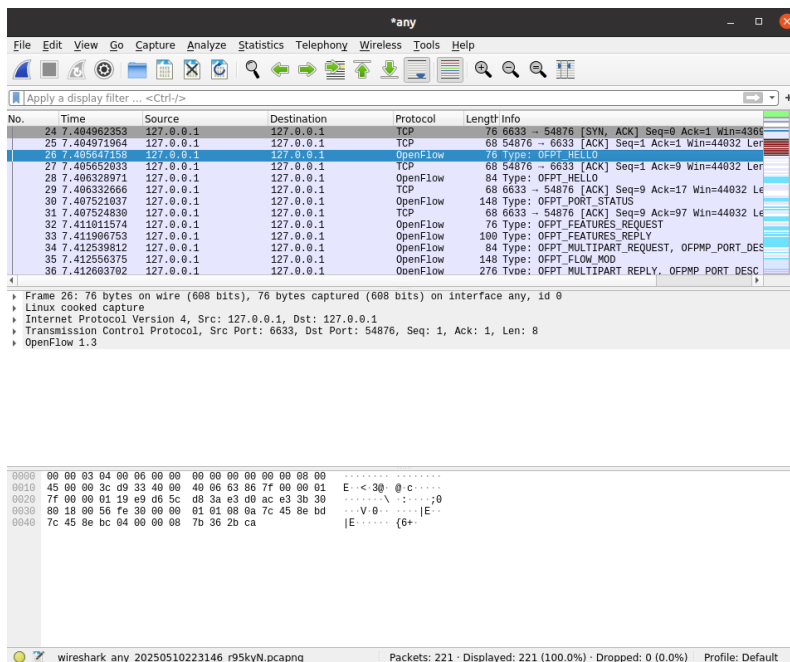


图 1.12 抓包结果一

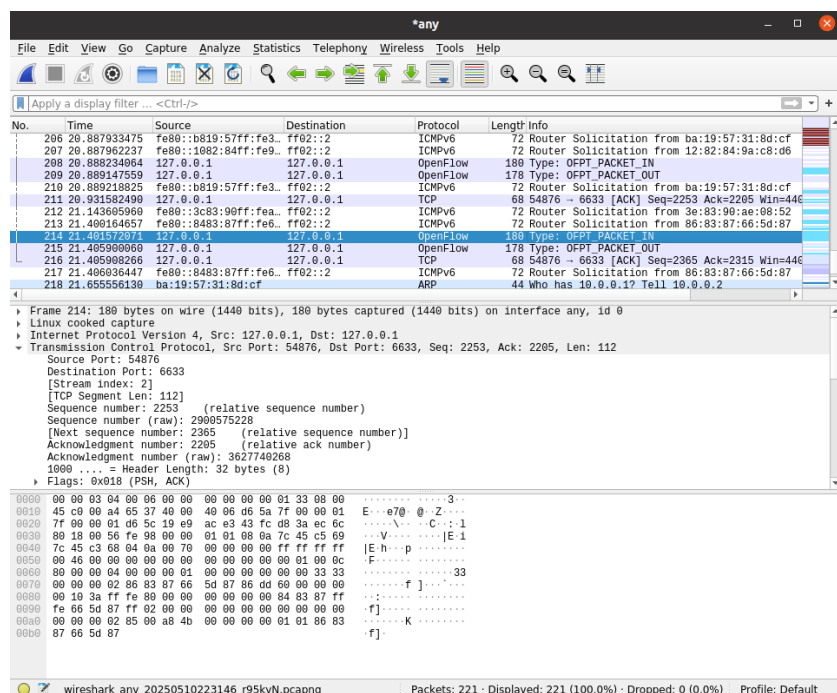


图 1.13 抓包结果二

### 4.3 建立 Mininet 与 Ryu 连接并抓包分析

在了解上述实验基本步骤后，请大家选用自己一种网络拓扑构建方法（也可选用除 4.2 节所述三种方法外的其他方案）构建一个更为复杂的网络拓扑（**至少包含 3 个以上的交换机**）。选用合适的 Ryu 控制器应用程序，建立 Mininet 与 Ryu 之间的连接。

通过 Wireshark 抓包工具，针对 HELLO、PORT\_STATUS、FEATURES\_REQUEST、FEATURES\_REPLY、FLOW\_MOD、PACKET\_IN、PACKET\_OUT 等 OpenFlow 协议主要报文类型进行分析，**至少包含：主要功能及作用分析、主要字段分析、报文出现阶段分析。并在宏观层面分析各类报文之间是如何相互配合的，给出流程图。**

## 五、实验报告

实验报告应包含以下几个部分：

### 1. 实验目的。

2. 实验内容。
3. 实验详细步骤（包含命令、代码、实验结果截图、实验结果分析）。
4. 实验分析。
5. 实验过程中遇到的问题及解决办法。
6. 实验心得

## 实验二 OpenFlow 流表操作实践

### 一、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。

### 二、实验要求

1. 建立 Mininet 与 Ryu 的连接，实现简单的二层交换功能。
2. 通过命令行工具 curl 进行下发、查看、删除流表操作。
3. 通过 Mininet 指令和 ryu 命令行内容验证流表匹配与数据包处理。

**注：可选择使用指导书中示范的拓扑，或自己重新构建不同拓扑（视实验设计难度与个人思考给分）**

### 三、实验环境及主要工具介绍

虚拟机软件：VMware 17.6

Linux 系统：Ubuntu 20.04.6

Python：python 3.8.10

网络仿真平台：mininet 2.3.1b1

SDN 控制器：ryu 4.34

### 四、实验步骤

#### 4.1 启动实验环境

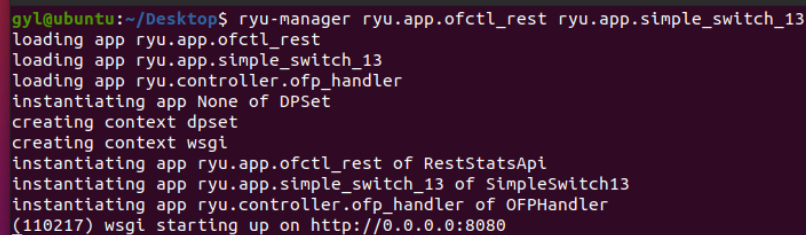
安装 curl（Client URL）。

```
sudo apt install curl
```

curl 是一个开源命令行工具，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。

终端 1 启动 Ryu 控制器。

```
ryu-manager ryu.app.ofctl_rest ryu.app.simple_switch_13
```



```
gyl@ubuntu:~/Desktop$ ryu-manager ryu.app.ofctl_rest ryu.app.simple_switch_13
loading app ryu.app.ofctl_rest
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
(110217) wsgi starting up on http://0.0.0.0:8080
```

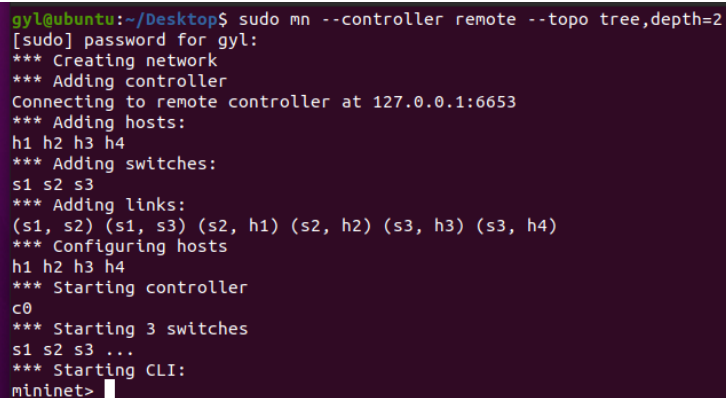
ofctl\_rest：一个 REST API 模块，允许通过 http 请求（如 get/post/delete）管理 OpenFlow 交换机的流表、端口状态等。

simple\_switch\_13：支持 OpenFlow 1.3 的二层交换应用，会自动学习 MAC 地址并下发流表。

二层交换：基于 OSI 模型的数据链路层进行转发，核心是通过 **MAC 地址** 决定数据的转发路径。

终端 2 启动 Mininet 拓扑。

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



```
gyl@ubuntu:~/Desktop$ sudo mn --controller remote --topo tree,depth=2
[sudo] password for gyl:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

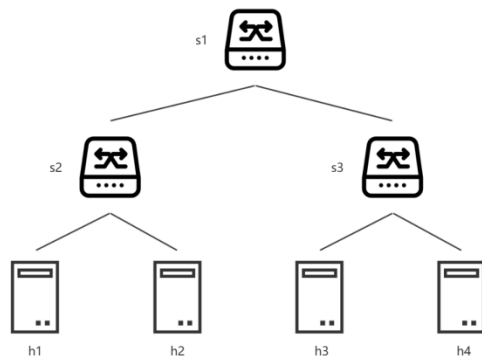
tree,depth=2: 创建了一个两层树形拓扑。

使用 net 指令查看其拓扑。

net

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo: s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo: s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
c0
```

分析可知，具体拓扑如下：



## 4.2 实现二层交换

在 mininet 终端中 pingall，应能直接互通。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

这是因为 Ryu 的 simple\_switch\_13 已经实现了二层自学习交换功能，无需额外配置。

在终端 3 中查看交换机 s1 的流表，已经自动学习 MAC 地址：

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



```

65535
gyl@ubuntu:~/Desktop$ curl http://127.0.0.1:8080/stats/flow/1
{"1": [{"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 238, "duration_sec": 127, "duration_nsec": 275000000, "packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match": {"in_port": 2, "dl_src": "36:ef:27:d1:f0:16", "dl_dst": "c2:ce:06:9c:ca:90"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 140, "duration_sec": 127, "duration_nsec": 272000000, "packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match": {"in_port": 1, "dl_src": "c2:ce:06:9c:ca:90", "dl_dst": "36:ef:27:d1:f0:16"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 238, "duration_sec": 127, "duration_nsec": 259000000, "packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match": {"in_port": 2, "dl_src": "26:b3:e0:25:ed:8d", "dl_dst": "c2:ce:06:9c:ca:90"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 140, "duration_sec": 127, "duration_nsec": 259000000, "packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match": {"in_port": 1, "dl_src": "c2:ce:06:9c:ca:90", "dl_dst": "26:b3:e0:25:ed:8d"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 238, "duration_sec": 127, "duration_nsec": 251000000, "packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match": {"in_port": 2, "dl_src": "36:ef:27:d1:f0:16", "dl_dst": "26:4d:6b:20:82:b5"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 140, "duration_sec": 127, "duration_nsec": 249000000, "packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match": {"in_port": 1, "dl_src": "26:4d:6b:20:82:b5", "dl_dst": "36:ef:27:d1:f0:16"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 238, "duration_sec": 127, "duration_nsec": 237000000, "packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match": {"in_port": 2, "dl_src": "26:b3:e0:25:ed:8d", "dl_dst": "26:4d:6b:20:82:b5"}, "table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count": 140, "duration_sec": 127, "duration_nsec": 235000000, "packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match": {"in_port": 1, "dl_src": "26:4d:6b:20:82:b5", "dl_dst": "26:b3:
gyl@ubuntu:~/Desktop$

```

另一种查看当前 s1 流表的方式:

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

```

gyl@ubuntu:~/Desktop$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for gyl:
cookie=0x0, duration=61.858s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=36:ef:27:d1:f0:16,dl_dst=c2:ce:06:9c:ca:90 actions=output:"s1-eth1"
cookie=0x0, duration=61.855s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=c2:ce:06:9c:ca:90,dl_dst=36:ef:27:d1:f0:16 actions=output:"s1-eth2"
cookie=0x0, duration=61.844s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=26:b3:e0:25:ed:8d,dl_dst=c2:ce:06:9c:ca:90 actions=output:"s1-eth1"
cookie=0x0, duration=61.842s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=c2:ce:06:9c:ca:90,dl_dst=26:b3:e0:25:ed:8d actions=output:"s1-eth2"
cookie=0x0, duration=61.834s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=36:ef:27:d1:f0:16,dl_dst=26:4d:6b:20:82:b5 actions=output:"s1-eth1"
cookie=0x0, duration=61.832s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=26:4d:6b:20:82:b5,dl_dst=36:ef:27:d1:f0:16 actions=output:"s1-eth2"
cookie=0x0, duration=61.820s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=26:b3:e0:25:ed:8d,dl_dst=26:4d:6b:20:82:b5 actions=output:"s1-eth1"
cookie=0x0, duration=61.818s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=26:4d:6b:20:82:b5,dl_dst=26:b3:e0:25:ed:8d actions=output:"s1-eth2"
cookie=0x0, duration=1298.489s, table=0, n_packets=123, n_bytes=11336, priority=0 actions=CONTROLLER:65535

```

思考题：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？

## 4.3 下发流表、流表匹配与数据包处理

首先删除 s1 的所有流表。

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

再次查看当前 s1 流表，发现已经为空。

```
gyl@ubuntu:~/Desktop$ curl -X POST -d '{"dpid":1,"match":{}}' http://127.0.0.1:8080/stats/flowentry/delete
gyl@ubuntu:~/Desktop$ curl http://127.0.0.1:8080/stats/flow/1
{"1": []}gyl@ubuntu:~/Desktop$
```

尝试 pingall, 仅有 h1 和 h2 之间、h3 和 h4 之间能 ping 通。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 X X
h3 -> X X h4
h4 -> X X h3
*** Results: 66% dropped (4/12 received)
```

思考题：请解释这种现象的原因。

下面尝试让 h1 <-> h3 能 ping 通。

查看 h1 的 mac 地址：

```
h1 ifconfig
```

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::c0ce:6ff:fe9c:ca90 prefixlen 64 scopeid 0x20<link>
    ether c2:ce:06:9c:ca:90 txqueuelen 1000 (Ethernet)
    RX packets 170 bytes 16029 (16.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 50 bytes 3316 (3.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2 bytes 224 (224.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 224 (224.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

同理得到 h3 或其他主机的 mac 地址。

给 s1 添加一个流表，该流表的源地址为 h1，目标地址为 h3。

```
curl -X POST -d '{
    "dpid": 1,
    "priority": 100,
    "match": {
        "in_port": 1,
```

```
"dl_src": "c2:ce:06:9c:ca:90",  
  
"dl_dst": "36:ef:27:d1:f0:16"  
  
},  
  
"actions": [{"type": "OUTPUT", "port": 2}]  
}' http://127.0.0.1:8080/stats/flowentry/add
```

思考题：此时尝试 ping，应 ping 不通。为什么？

提示：请观察 ryu 终端显示的包信息的目的地址。

```
127.0.0.1 - - [12/May/2025 02:01:13] "POST /stats/flowentry/add HTTP/1.1" 200 11  
5 0.000644  
packet in 2 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 1  
packet in 3 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 3  
packet in 2 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 1  
packet in 3 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 3  
packet in 2 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 1  
packet in 3 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 3  
packet in 2 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 1  
packet in 3 c2:ce:06:9c:ca:90 ff:ff:ff:ff:ff:ff 3  
packet in 2 26:4d:6b:20:82:b5 ff:ff:ff:ff:ff:ff 2  
packet in 2 26:4d:6b:20:82:b5 ff:ff:ff:ff:ff:ff 2  
packet in 2 26:4d:6b:20:82:b5 ff:ff:ff:ff:ff:ff 2  
packet in 2 26:4d:6b:20:82:b5 ff:ff:ff:ff:ff:ff 2  
packet in 2 26:4d:6b:20:82:b5 ff:ff:ff:ff:ff:ff 2
```

给 s1 添加一个用于 h1 广播的流表：

```
curl -X POST -d '{  
  
  "dpid": 1,  
  
  "priority": 100,  
  
  "match": {  
  
    "in_port": 1,  
  
    "dl_src": "c2:ce:06:9c:ca:90",  
  
    "dl_dst": "ff:ff:ff:ff:ff:ff"  
  
  },  
  
  "actions": [{"type": "OUTPUT", "port": 2}]  
}' http://127.0.0.1:8080/stats/flowentry/add
```

同理，给 s1 添加 h3->h1 的流表，随后 pingall，即可发现 h1 与 h3 之间能够 ping 通。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X
h2 -> h1 X X
h3 -> h1 X h4
h4 -> X X h3
*** Results: 50% dropped (6/12 received)
mininet>
```

再次检查 s1 的流表项：

```
gyl@ubuntu:~/Desktop$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for gyl:
 cookie=0x0, duration=1764.033s, table=0, n_packets=3, n_bytes=238, priority=100,in_port="s1-eth1",dl_
src=c2:ce:06:9c:ca:90,dl_dst=36:ef:27:d1:f0:16 actions=output:"s1-eth2"
 cookie=0x0, duration=1532.346s, table=0, n_packets=4, n_bytes=280, priority=100,in_port="s1-eth2",dl_
src=36:ef:27:d1:f0:16,dl_dst=c2:ce:06:9c:ca:90 actions=output:"s1-eth1"
 cookie=0x0, duration=508.107s, table=0, n_packets=4, n_bytes=168, priority=100,in_port="s1-eth1",dl_s
rc=c2:ce:06:9c:ca:90,dl_dst=ff:ff:ff:ff:ff:ff actions=output:"s1-eth2"
```

思考题：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？

## 五、实验报告

实验指导书应包含以下几个部分：

1. 实验目的
2. 实验内容
3. 实验详细步骤（包含命令和实验截图）
4. 实验分析
5. 实验过程中遇到的问题及解决办法
6. 实验心得

**注：**可以使用指导书的示范拓扑，或自己构建其他形式的拓扑。在使用示范拓扑的情况下，应在“实验分析”部分中应回答前述的四个思考题；在使用自己构建的拓扑时，应给出拓扑图，以及主机之间联通的思路，详细说明是如何设计实验以验证流表匹配与数据包处理的。