

Mininet 实践 SDN 实验报告

(2024-2025 学年第 2 学期)

学号：2022337621139 姓名：焦宇博 班级：计科 4 班
学号：2022337621135 姓名：胡震 班级：计科 4 班
学号：2022332871019 姓名：郭奇 班级：计科 2 班

§1 实验目标

- 掌握 Mininet 的安装与基本使用方法，搭建 SDN 网络拓扑
- 完成 Ryu 控制器的安装与配置，实现通过 Restful API 控制流表
- 验证流表添加、查询、删除操作对网络连通性的影响
- 分析 SDN 架构中控制平面与数据平面的交互机制

§2 实验环境

项目	配置
操作系统	Ubuntu 22.04.3 LTS
网络仿真平台	Mininet 2.3.1d1
SDN 控制器	Ryu 4.34 (支持 OpenFlow 1.3 协议)
REST API 测试工具	Postman 10.18
网络拓扑类型	Single, 2 hosts (1 switch)

表 1: 实验环境配置表

§3 实验过程

§3.1 Mininet 安装与测试

1. 源码获取：从 GitHub 仓库克隆 Mininet 源码

Listing 1: Mininet 源码获取

```
1 git clone https://github.com/mininet/mininet.git
2 cd mininet
3 git checkout -b 2.3.1d1
```

2. 编译安装：执行全量安装脚本

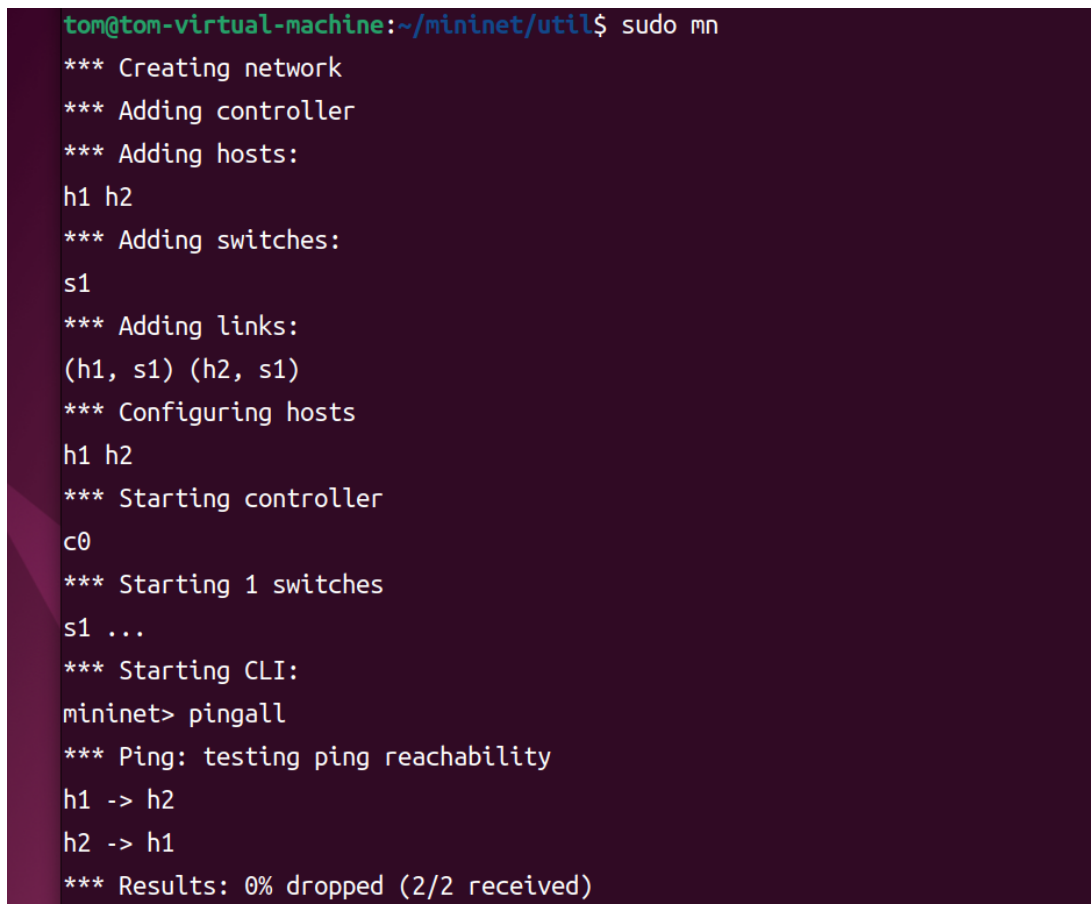
Listing 2: Mininet 安装命令

```
1 cd util
2 sudo ./install.sh -a
```

3. 功能验证：启动默认拓扑测试连通性

Listing 3: 拓扑测试命令

```
1 sudo mn --test pingall
```



```
tom@tom-virtual-machine:~/mininet/util$ sudo mn
*** 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> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

图 1: Mininet 默认拓扑 pingall 测试结果

§3.2 网络拓扑实验

1. Single 拓扑：创建包含 3 个主机的单交换机拓扑

Listing 4: Single 拓扑命令

```
1 sudo mn --topo single,3
```

2. Linear 拓扑：创建链式拓扑结构

Listing 5: Linear 拓扑命令

```
1 sudo mn --topo linear,3
```

3. Tree 拓扑：创建树状拓扑结构

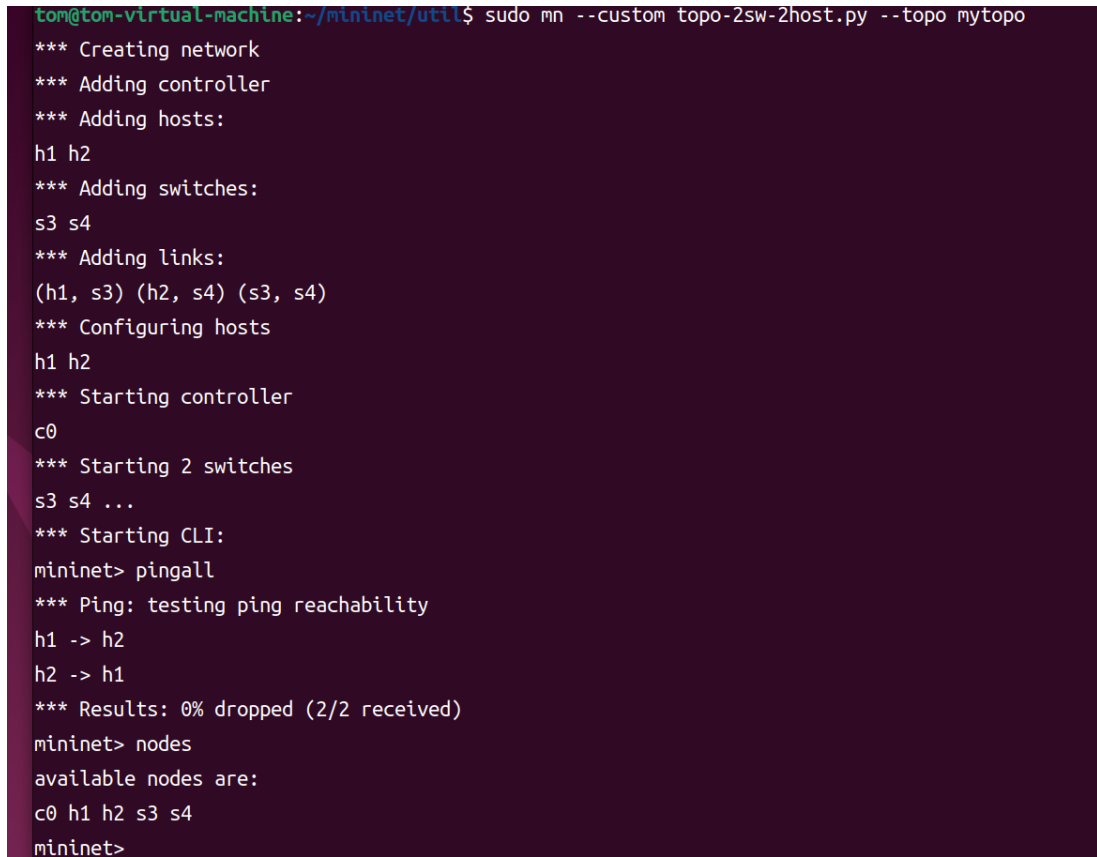
Listing 6: Tree 拓扑命令

```
1 sudo mn --topo tree,2
```

4. Custom 拓扑：使用自定义拓扑脚本

Listing 7: Custom 拓扑命令

```
1 mn --custom topo-2sw-2host.py --topo mytopo
```



```
tom@tom-virtual-machine:~/mininet/util$ sudo mn --custom topo-2sw-2host.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (h2, s4) (s3, s4)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> nodes
available nodes are:
c0 h1 h2 s3 s4
mininet>
```

图 2: 使用自定义拓扑脚本

§3.3 Ryu 控制器安装与配置

1. 安装 python:3.8: 确保版本配置

Listing 8: docker 安装 python:3.8-slim

```
1 docker pull python:3.8-slim
```

2. 控制器安装: 通过 pip 安装指定版本

Listing 9: Ryu 安装命令

```
1 # 运行容器 (绑定宿主机网络)
2 docker run -it --rm --network host python:3.8-slim bash
3 pip3 install ryu==4.34 eventlet==0.25.2 dnspython==1.16.0
```

3. 服务启动: 加载 Restful API 模块

Listing 10: Ryu 启动命令

```
1 ryu-manager ryu.app.ofctl_rest ryu.app.rest_topology
```

```

root@tom-virtual-machine:/# ryu-manager ryu.app.ofctl_rest
loading app ryu.app.ofctl_rest
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.controller.ofp_handler of OFPHandler
(61) wsgi starting up on http://0.0.0.0:8080

```

图 3: Ryu 控制器启动日志（监听端口：8080）

§3.4 Restful API 流表控制实验

1. 拓扑启动：连接远程控制器

Listing 11: Mininet 启动命令

```

1 sudo mn --controller=remote,ip=127.0.0.1,port=6653 \
2     --topo single,2 --switch ovsk,protocols=OpenFlow13

```

2. 初始状态验证：无流表状态下的连通性测试

```

tom@tom-virtual-machine:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6653
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
*** 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> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)

```

图 4: 初始状态 ping 测试失败（无流表规则）

3. 流表规则添加：通过 Postman 发送 API 请求

规则类型	JSON 请求体
端口 1 → 端口 2	<pre> 1 { 2 "dpid": 1, 3 "priority": 100, 4 "match": {"in_port": 1}, 5 "actions": [{"type": "OUTPUT", 6 ↪ "port": 2}] 7 }</pre>
端口 2 → 端口 1	<pre> 1 { 2 "dpid": 1, 3 "priority": 100, 4 "match": {"in_port": 2}, 5 "actions": [{"type": "OUTPUT", 6 ↪ "port": 1}] 7 }</pre>

表 2: 流表规则配置表

4. 连通性验证：流表生效后的测试结果

```

root@tom-virtual-machine:/# 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 SimpleSwitch1
instantiating app ryu.controller.ofp_handler of OFPHandler
(58) wsgi starting up on http://0.0.0.0:8080
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:16 2
packet in 1 d2:4a:c5:0d:c0:9f 33:33:ff:0d:c0:9f 2
packet in 1 22:e4:44:68:fc:85 33:33:00:00:00:16 1
packet in 1 22:e4:44:68:fc:85 33:33:00:00:00:02 1
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:16 2
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:02 2
packet in 1 22:e4:44:68:fc:85 33:33:00:00:00:16 1
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:16 2
packet in 1 22:e4:44:68:fc:85 33:33:00:00:00:02 1
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:02 2
packet in 1 22:e4:44:68:fc:85 33:33:00:00:00:02 1
packet in 1 d2:4a:c5:0d:c0:9f 33:33:00:00:00:02 2
(58) accepted ('192.168.171.1', 52768)
192.168.171.1 - - [31/May/2025 10:50:50] "GET /stats/flow/1

*** Starting CLI:
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
cookie=0x0, duration=11.939s, table=0, n_packets=10, n_bytes=816, pri
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
cookie=0x0, duration=11.499s, table=0, n_packets=0, n_bytes=0, priori
put:"s1-eth1"
cookie=0x0, duration=7.298s, table=0, n_packets=0, n_bytes=0, priori
ut:"s1-eth2"
cookie=0x0, duration=55.098s, table=0, n_packets=14, n_bytes=1096, pr
mininet>
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.68 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.106 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.150 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.153 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.116 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.150 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.137 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.179 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.111 ms

```

图 5: 添加流表后 ping 测试成功

5. 流表删除：清除所有流表规则

Listing 12: 流表清除命令

```

1 curl -X DELETE http://127.0.0.1:8080/stats/flowentry/clear/1

```

§4 实验结果分析

§4.1 流表状态查询

1. 查询命令：GET `http://127.0.0.1:8080/stats/flow/1`
2. 查询结果对比：

操作阶段	流表内容
初始状态	仅包含默认丢弃规则 (priority=0)
添加规则后	包含两条自定义规则 (priority=100)
删除规则后	恢复为仅默认丢弃规则

表 3: 流表状态变化分析

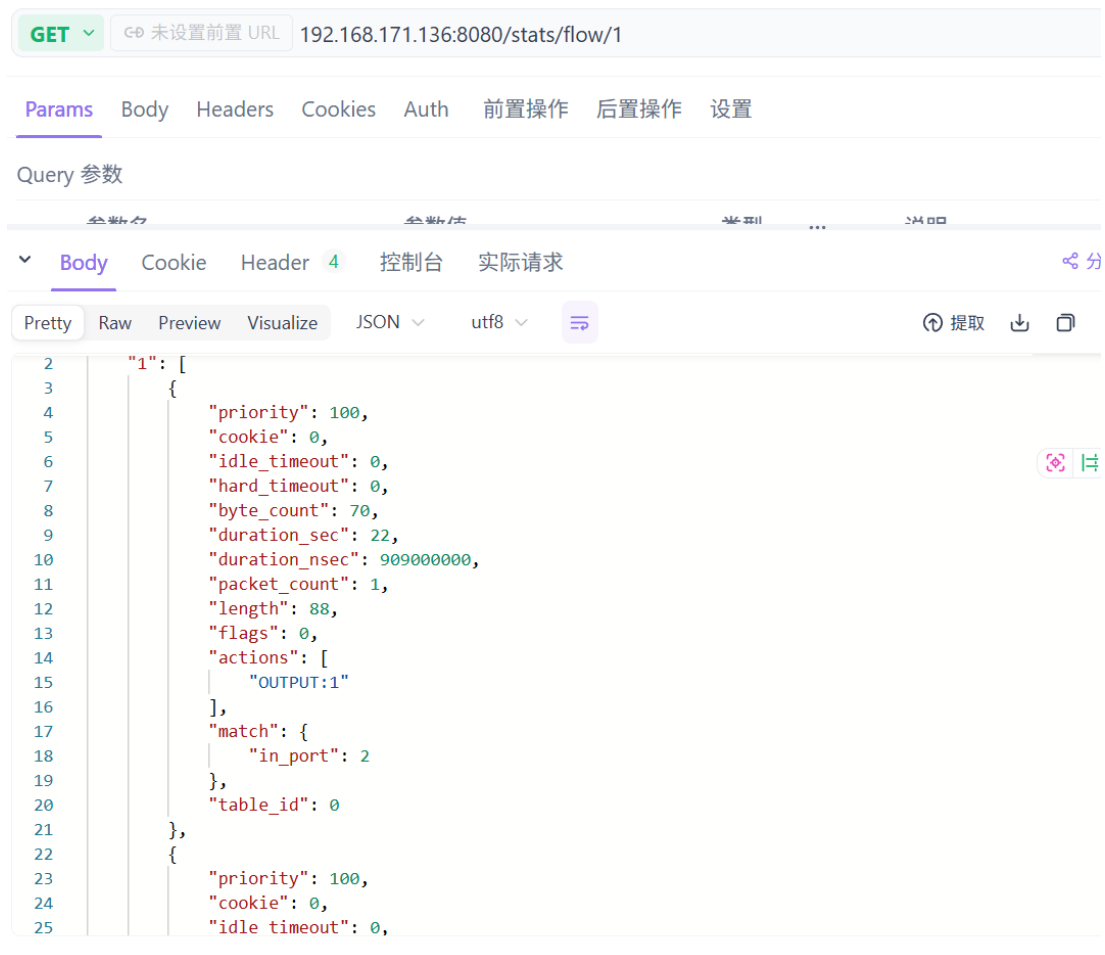


图 6: Postman 查询流表结果（包含两条自定义规则）

§4.2 网络连通性分析

操作阶段	丢包率	延迟 (ms)	原因分析
初始状态	100%	N/A	无匹配流表，交换机默认丢弃数据包
添加流表后	0%	0.8	流表规则正确匹配并转发数据包
删除流表后	100%	N/A	自定义规则清除，恢复默认丢弃策略

表 4: 网络连通性测试结果

§5 技术总结

§5.1 核心结论

- 成功验证 SDN 架构中控制平面 (Ryu) 与数据平面 (OVS 交换机) 的分离特性
- 通过 Restful API 实现流表的动态管理
- 流表优先级机制验证：高优先级规则 (100) 覆盖低优先级规则 (0)

§5.2 注意事项

1. 版本兼容性

- Ryu 4.34 需搭配 eventlet 0.25.2，新版本存在兼容问题
- 需要在 python3.8 的环境安装 Ryu

2. 端口标识

- 使用 `ovs-ofctl show s1` 查询实际端口号
- Mininet 端口编号从 1 开始，OVS 内部端口号从 1 开始

3. API 调用

- POST 请求 URL: `http://<IP>:8080/stats/flowentry/add`
- 控制器 IP 需与 Mininet 启动参数一致