

华中科技大学

第二届全国高校软件定义网络(SDN) 应用创新开发大赛

题目：初赛文档

姓 名 林家军 李英儒 胡云锐

专 业 计算机科学与技术

指 导 人 员 周世伟

院(系、所) 计算机科学与技术学院

华中科技大学联创团队制

摘 要

这是来自华中科技大学 UniqueSDNStudio 队伍的初赛文档,于 \LaTeX 编辑而成。

关键词： 软件定义网络, SDN, \LaTeX , 华中科技大学

目 录

摘要	I
插图索引	IV
表格索引	V
1 基础环境	1
1.1 Ryu	1
1.2 Mininet	1
2 参赛情况	2
2.1 竞赛组织及参与情况	2
2.2 参赛队伍构成	2
3 初赛题目	4
3.1 第一题: 基础题	4
3.2 第二题: 提高题	11
3.3 第三题: 设计题	16
3.4 字体	16
3.5 公式	16
3.6 罗列环境	16
4 其他格式测试	18
4.1 代码环境	18
4.2 定律证明环境	18
4.3 算法环境	18
4.4 表格	19
4.5 图片	19
4.6 参考文献示例	19
4.7 \autoref 测试	20
致谢	21
参考文献	22

附录 A UniqueSDNStudio 初赛文档	23
附录 B 这是一个附录	24

插图索引

图 3-1	第一题第 1 小题:简单网络	4
图 3-2	检查虚拟网络环境	5
图 3-3	删除流表后检查虚拟网络环境	6
图 3-4	第一题第 1 小题:访问限制	7
图 3-5	检查网络环境	10
图 3-6	11
图 3-7	流表过期	11
图 3-8	第二题第 1 小题:代理访问	12
图 3-9	拓扑图	12
图 3-10	输出流表	14
图 3-11	pingall 结果与流表变化	14
图 3-12	h1, h2 分别请求 h3, h4 网页的结果	15
图 3-13	第二题第 2 小题:流表管理	15
图 4-1	一个图片	19
图 4-2	多个图片	19

表格索引

表 4.1	一个表格	19
-------	----------------	----

一 基础环境

SDN 的基础环境由 SDN 控制器 (Controller) 和支持 SDN 的交换机构成的网络组成, 我们在本次比赛中使用 Ryu 作为 SDN 控制器, 使用 Mininet 构建虚拟试验网络。

1.1 Ryu

Ryu is a component-based software defined networking framework.

Ryu 是一个基于组件的软件定义网络框架。

Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4 and Nicira Extensions.

Ryu 提供了具有良好设计的 API 结构的软件组件, 使得开发者能够更加方便简单地创建新的网络管理控制应用。Ryu 支持许多管理网络设备的协议, 例如 OpenFlow, Netconf, OF-config 等。关于 Openflow, Ryu 支持全部 1.0, 1.2, 1.3, 1.4 和 Nicira 扩展。

All of the code is freely available under the Apache 2.0 license. Ryu is fully written in Python.

所有 Ryu 代码能够在 Apache 2.0 许可下自由地获取。Ryu 全部由 Python 写成。

1.2 Mininet

Mininet 是由一些虚拟的终端节点 (end-hosts)、交换机、路由器连接而成的一个网络仿真器, 它采用轻量级的虚拟化技术使得系统可以和真实网络相媲美。它可以很方便的创建一个支持 SDN 的网络, 有了这个网络, 就可以灵活的为网络添加新的功能并进行相关测试。

二 参赛情况

2.1 竞赛组织及参与情况

本次竞赛的组织活动由校教务处发起, 在全校范围内自主报名参加。联创团队组织三名大二学生参赛, 并安排一位研究生作为领队。在参赛过程中, 每周每人大约工作时间 40 小时且每周六进行一次讨论总结上周情况并对下周进行科学严谨的安排, 为保证作品按时完成我们建立了邮件列表与在线讨论平台, 同一管理协调。

2.2 参赛队伍构成

2.2.1 联创团队

联创团队 (*UniqueStudio*) 于 2000 年 6 月创建于华中科技大学, 是 *Teamwork* 和 *Creation* 为团队核心的学生团队。团队名称来源于“联众人之智”必能“创辉煌之事”的信念。联创团队建立了一个自主的精英学生平台, 在这个平台上学生自我管理, 通过这个平台激发无限的潜力和创意。自成立到现在, 联创团队已参加微软创新杯 11 次并 8 次进入全球总决赛拿下包括全球冠军等优秀成绩。除此之外, 联创团队也多次参加各种大小型比赛并取得了骄人的成绩, 并与 *MICROSOFT™*、*CSDN™* 等公司保持着非常好的合作关系, 同时加入微软中国组织的创新联盟。联创团队多年的发展积累了独特的文化和运作机制, 同时跟踪技术发展的最前沿, 这使得联创团队在华科乃至全国高校中独树一帜。

2.2.2 领队

周世伟

华中科技大学 2010 级本科生, 2014 级研究生, 现就读于光电与电子信息学院光电信息工程专业。华中科技大学联创团队 IT 组和嵌入式组成员。曾 2 次获得国家奖学金。参与微软创新杯 *Imagine Cup™ IT Challenge®* 比赛, 并进入中国区总决赛。曾获得中国区第一届 *RDMA®* 比赛一等奖。目前已获得华中科技大学保研资格。[周世伟个人博客](#)。

2.2.3 参赛队员

林家军

华中科技大学 2013 级本科生, 现就读于计算机科学与技术学院计算机科学与技术专业。华中科技大学联创团队 IT 组成员。

李英儒

华中科技大学联创团队

华中科技大学 2013 级本科生, 现就读于计算机科学与技术学院计算机科学与技术专业。华中科技大学联创团队 IT 组成员。

胡云锐

华中科技大学 2013 级本科生, 现就读于计算机科学与技术专业。华中科技大学联创团队 IT 组成员, 主要进行网络通讯、大数据方面的研究。

三 初赛题目

3.1 第一题: 基础题

3.1.1 第 1 小题: 简单网络

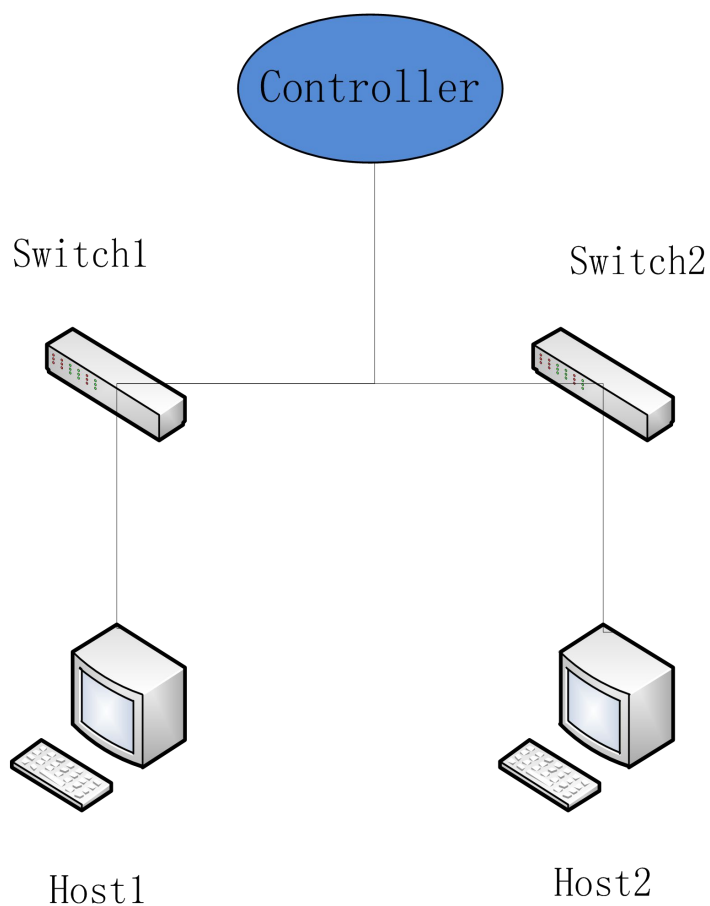


图 3-1 第一题第 1 小题:简单网络

网络环境要求

要求搭建的网络环境如图 1 所示 (Controller 表示控制器, switch 表示交换机, host 表示主机)。控制器可以自主选择, 例如各种开源的控制器 (Floodlight、Ryu、Nox、Beacon、Trema、OpenDaylight 等)。拓扑中各网络部件既可以是仿真环境实现 (例如 mininet, OpenvSwitch 等), 有条件的队伍也可以通过物理设备实现, 各种方案不影响评分。试题中

每道题都遵循这些要求, 不再说明。

操作要求

1. 先使 Host1 可以 ping 通 Host2, Host2 也可以 ping 通 Host1。
2. 然后对流表进行操作, 使 Host1 不能 ping 通 Host2, Host2 也不能 ping 通 Host1。

方法描述

利用 mininet 创建虚拟环境, 在虚拟交换机上直接修改流表。

实验步骤

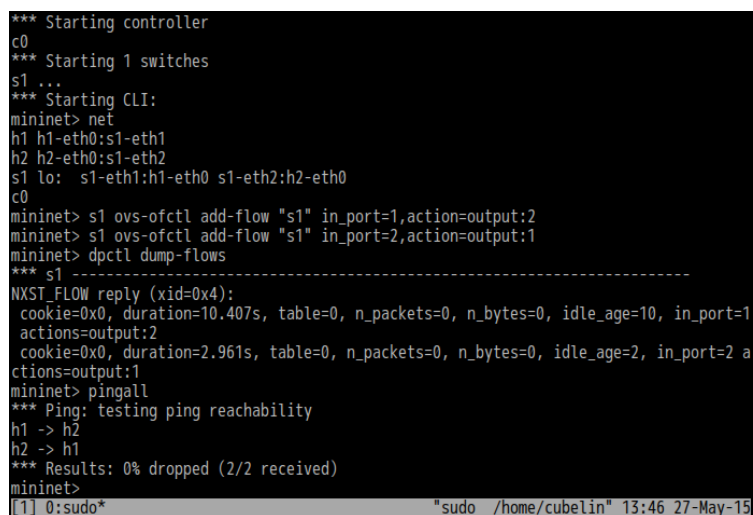
1. 打开 mininet 创建一个拓扑, 它包含一个交换机和两个主机。命令如下:

```
1 mn --topo single,2
```

2. 在 mininet CLI 中检查环境, 确认流表为空。
3. 执行如下两条命令, 添加流表, 使两个虚拟主机可以通信。

```
1 s1 ovs-ofctl add-flow "s1" in_port=1,action=output:2
2 s1 ovs-ofctl add-flow "s1" in_port=2,action=output:1
```

4. 执行 pingall 检查网络环境, 结果见图 3-2, h1 与 h2 可以正常通信。



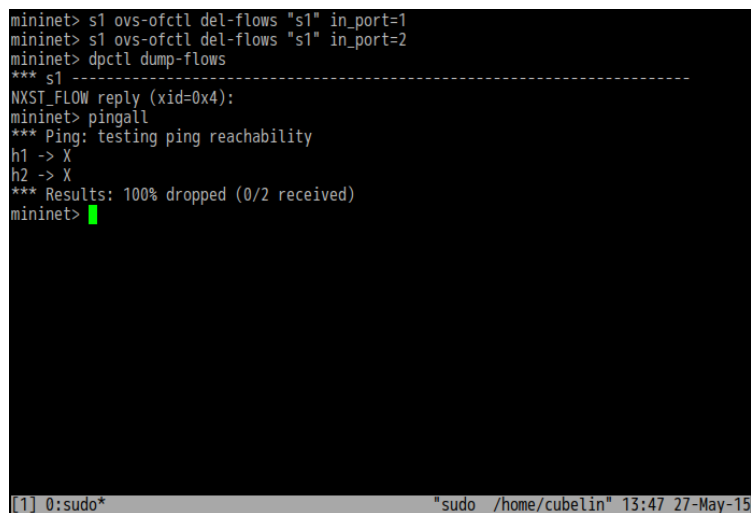
```
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> s1 ovs-ofctl add-flow "s1" in_port=1,action=output:2
mininet> s1 ovs-ofctl add-flow "s1" in_port=2,action=output:1
mininet> dpctl dump-flows
*** s1 ***
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.407s, table=0, n_packets=0, n_bytes=0, idle_age=10, in_port=1
 actions=output:2
 cookie=0x0, duration=2.961s, table=0, n_packets=0, n_bytes=0, idle_age=2, in_port=2 a
 ctions=output:1
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

图 3-2 检查虚拟网络环境

5. 执行如下两条命令, 删除添加的流表。

```
1 s1 ovs-ofctl del-flows "s1" in_port=1
2 s1 ovs-ofctl del-flows "s1" in_port=2
```

6. 执行 pingall 再次检查网络环境,结果见图 3-3,h1 与 h2 无法正常通信。



```
mininet> s1 ovs-ofctl del-flows "s1" in_port=1
mininet> s1 ovs-ofctl del-flows "s1" in_port=2
mininet> dpctl dump-flows
*** s1 ***
NXST_FLOW reply (xid=0x4):
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

图 3-3 删除流表后检查虚拟网络环境

实验结果

通过修改交换机的流表,先后成功实现虚拟主机 h1 与 h2 能互相 ping 通,与 ping 不通。

3.1.2 第 2 小题：访问限制

网络环境要求

设有一台 PC 机 (Host1), 一台 Web 服务器 (Server1) 提供简单的静态网页访问服务,如图 2 所示。(PC 机、Web 服务器、交换机、控制器可以选择物理设备或者虚拟设备实现)

操作要求

1. 先使得 PC 机访问服务器成功 (即看到服务器的网页)。
2. 限制该 PC 机一定时间 (比如一分钟) 内再次访问服务器。限制时间过后,PC 机可以成功访问服务器。

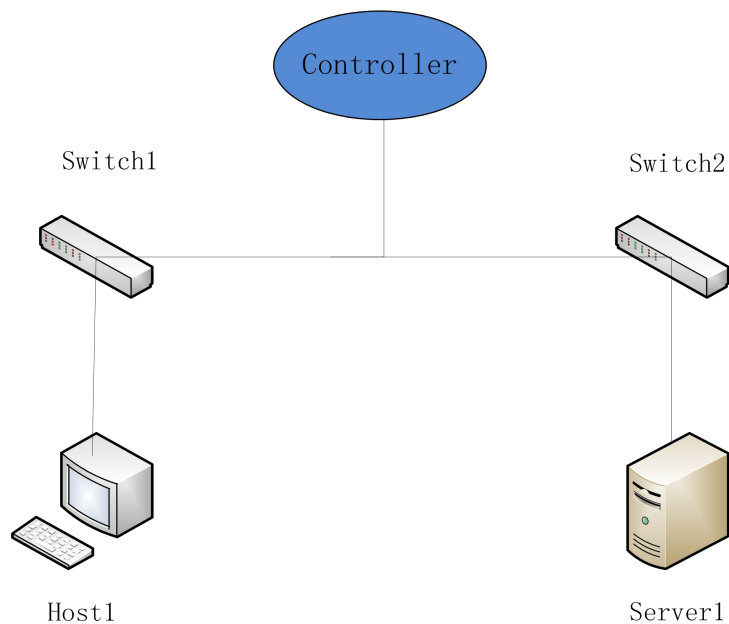


图 3-4 第一题第 1 小题:访问限制

方法描述

拓扑环境如图 3-4, 其中 h1 为客户机, h2 为服务器。

为了一般性, 我们编写了第三方 Ryu Controller App, 接收交换机连接 Controller 时发来的 **Feature message**, 并为交换机添加默认流表。

首先保证主机 h1 和 h2 的正常通信, 如果流表匹配到 h1 发来的 TCP 包, 在送出交换机 2 号端口的同时复制一份送给 Controller。

Controller 分析后, 判断此包来源的 mac 是否在白名单中;

若不在, 放行 TCP 包 ACK 标志小于等于 1 的包, 为交换机添加一个带有 60 秒 **hard_timeout** 的 flow entry, 丢弃所有来自此 mac 并且目的 mac 地址是主机 h2 的 http 包, 同时将此 mac 加入白名单。

60 秒过后, 此 flow entry 过期, 而来源 mac 已在白名单中, 下次请求将不会阻止。

算法 3.1: Visiting Constraints using SDN

```

1 Initial Ryu Controller & White List =  $\emptyset$  ;
2 DATA PLANE[Hardware Layer];
3 while Flowtable Match TCP_package from h1 do
4   | Send TCP_package to port 2 and its copy to RyuController ;
5 end
6 CONTROL PLANE[Software Layer];
7 while Recieve TCP_package do
8   | if TCP_package.source_mac  $\notin$  WhiteList then
9     | if TCP_package.ACK  $\leq 1$  then
10      | Pass TCP_package ;
11      | Send Flow Entry with hard_timeout = 60 to Drop HTTP Package from
12      | TCP_package.source_mac to h2.mac;
13      | White List  $\leftarrow$  White List  $\cup$  {TCP_package.source_mac} ;
14    | end
15 end

```

实验步骤

1. 执行如下命令, 其中 `setup_network.py` 用于搭建虚拟网络环境, 执行该脚本需要 root 权限。1_2.py 是我们编写的 Ryu Controller App。

```

1 python2 setup_network.py
2 ryu-manager 1_2.py

```

交换机连接 Controller 后, 会往 Controller 发送 Feature message, 以下代码将发挥作用。

```

1 ...
2 @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
3 def switch_feature_handler(self, ev):
4     datapath = ev.msg.datapath
5     ofproto = datapath.ofproto
6     parser = datapath.ofproto_parser
7     dpid = datapath.id
8
9     self.logger.info("Switch %d connected, adding flow entries.", dpid)
10    match = parser.OFPMatch(in_port=1)

```

```

11     actions = [parser.OFPActionOutput(2, ofproto.OFPCML_NO_BUFFER)]
12     # Flow entry match structure: priority=0,in_port=1 actions=output:2
13     self.add_flow(datapath, 0, match, actions)
14
15     match = parser.OFPMatch(in_port=2)
16     actions = [parser.OFPActionOutput(1, ofproto.OFPCML_NO_BUFFER)]
17     # Flow entry match structure: priority=0,in_port=2 actions=output:1
18     self.add_flow(datapath, 0, match, actions)
19
20     if dpid == 2:
21         match = parser.OFPMatch(in_port=1, eth_type=PROTO_IP,
22                                 ip_proto=PROTO_TCP, tcp_dst=80)
23         actions = [parser.OFPActionOutput(2, ofproto.OFPCML_NO_BUFFER),
24                   parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
25                                           ofproto.OFPCML_NO_BUFFER)]
26         # Flow entry match structure:
27         # priority=1,tcp,in_port=1,tp_dst=80
28         # actions=output:2,CONTROLLER:65535
29         self.add_flow(datapath, 1, match, actions)
30
31     ...

```

2. 在 mininet CLI 中检查网络环境,结果如图 3-5。

3. 在 mininet CLI 中执行 `h2 xterm &`,并在打开的 xterm 中执行 `python -m HTTPServer 80` 打开 Web 服务器。

4. 回到 mininet CLI 中执行 `h1 curl h2`,得到以下返回

```
**this_is_the_web_page_of_problem_1.2**
```

一分钟内再次执行 `h1 curl h2 -m 10`,超时。检查流表,结果如图 3-6。

Controller 接受到 PacketIn message 时,以下代码将会发挥作用。

```

1     ...
2     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
3     def _packet_in_handler(self, ev):
4         msg = ev.msg
5         datapath = msg.datapath
6         parser = datapath.ofproto_parser
7         in_port = msg.match['in_port']
8         dpid = datapath.id
9         pkt = packet.Packet(msg.data)
10        eth = pkt.get_protocols(ethernet.ethernet)[0]
11        dst = eth.dst
12        src = eth.src
13        tcp_p = pkt.get_protocols(tcp.tcp)
14
15        if dpid != 2 or src in self.macs or len(tcp_p) == 0:
16            return
17
18        tcp_p = tcp_p[0]

```

```

19     if tcp_p.ack <= 1: # TCP three-way handshake and HTTP request
20         return
21
22     self.logger.info("First HTTP request from %s, adding flow entry.", src)
23
24     self.macs.append(src)
25     actions = []
26     match = parser.OFPMatch(in_port=in_port, eth_src=src, eth_dst=dst,
27                             eth_type=PROTO_IP, ip_proto=PROTO_TCP,
28                             tcp_dst=80)
29     # Flow entry match structure:
30     # priority=2,tcp,in_port=1,dl_src=xx:xx:xx:xx:xx:xx,
31     # dl_dst=xx:xx:xx:xx:xx:xx,tp_dst=80 actions=drop
32     self.add_flow(datapath, 2, match, actions, hard_timeout=60)
33
34     ...

```

```

mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:h2-eth0
c0
mininet> dpctl dump-flows -O openflow13
*** s1 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=15.055s, table=0, n_packets=7, n_bytes=558, priority=0,in_port=1
 actions=output:2
 cookie=0x0, duration=15.055s, table=0, n_packets=20, n_bytes=2681, priority=0,in_port
 =2 actions=output:1
*** s2 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=15.055s, table=0, n_packets=0, n_bytes=0, priority=1,tcp,in_port
 =1,tp_dst=80 actions=output:2,CONTROLLER:65535
 cookie=0x0, duration=15.055s, table=0, n_packets=20, n_bytes=2681, priority=0,in_port
 =1 actions=output:2
 cookie=0x0, duration=15.055s, table=0, n_packets=7, n_bytes=558, priority=0,in_port=2
 actions=output:1
mininet>
[9] 0:sudo* "sudo /home/cubelin/D" 14:45 27-May-15

```

图 3-5 检查网络环境

5. 等待 60 秒,多次执行 `h1 curl h2`,均能得到正常返回

****this_is_the_web_page_of_problem_1.2****

此时再检查流表,发现第 4 步时新增的流表已经过期。结果见图 3-7

实验结果

利用 Ryu Controller 提供的 Python API,自动化实现问题要求。


```
mininet> h2 xterm &
mininet> h1 curl h2
**this_is_the_web_page_of_problem_1.2**
mininet> h1 curl h2 -m 10
curl: (28) Connection timed out after 10000 milliseconds
mininet> dpctl dump-flows -O openflow13
*** s1 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=50.900s, table=0, n_packets=20, n_bytes=1480, priority=0,in_port
=1 actions=output:2
 cookie=0x0, duration=50.900s, table=0, n_packets=30, n_bytes=3864, priority=0,in_port
=2 actions=output:1
*** s2 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=50.900s, table=0, n_packets=1, n_bytes=74, priority=1,tcp,in_por
t=1,tp_dst=80 actions=output:2,CONTROLLER:65535
 cookie=0x0, duration=50.900s, table=0, n_packets=23, n_bytes=3170, priority=0,in_port
=1 actions=output:2
 cookie=0x0, duration=50.900s, table=0, n_packets=16, n_bytes=1336, priority=0,in_port
=2 actions=output:1
 cookie=0x0, duration=23.326s, table=0, n_packets=10, n_bytes=764, hard_timeout=60, pr
iority=2,tcp,in_port=1,dl_src=f6:c9:da:9d:5f:a5,dl_dst=f6:7f:67:e8:9b:88,tp_dst=80 act
ions=drop
mininet>
```

图 3-6

```
mininet> h1 curl h2
**this_is_the_web_page_of_problem_1.2**
mininet> h1 curl h2
**this_is_the_web_page_of_problem_1.2**
mininet> h1 curl h2
**this_is_the_web_page_of_problem_1.2**
mininet> dpctl dump-flows -O openflow13
*** s1 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=209.927s, table=0, n_packets=69, n_bytes=5146, priority=0,in_por
t=1 actions=output:2
 cookie=0x0, duration=209.927s, table=0, n_packets=82, n_bytes=9657, priority=0,in_por
t=2 actions=output:1
*** s2 ***
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=209.928s, table=0, n_packets=48, n_bytes=3656, priority=1,tcp,in
_port=1,tp_dst=80 actions=output:2,CONTROLLER:65535
 cookie=0x0, duration=209.928s, table=0, n_packets=28, n_bytes=4469, priority=0,in_por
t=1 actions=output:2
 cookie=0x0, duration=209.928s, table=0, n_packets=65, n_bytes=5914, priority=0,in_por
t=2 actions=output:1
mininet>
```

图 3-7 流表过期

3.2 第二题: 提高题

3.2.1 第 1 小题: 代理访问

网络环境要求

设有两台 PC 机 (Host1,Host2), 一台 Web 服务器 (Server1) 提供简单的静态网页访问服务, 一台代理服务器 (Proxy1) 和 Web 服务器提供同样的服务, 两台服务器所显示的网页大致相同, 但要有显著差别, 可以是不同的网页内容或者不同颜色, 能够区分彼此即可, 如图 3 所示. (PC 机、Web 服务器、代理服务器、交换机、控制器可以选择物理设备或者虚拟设备实现)

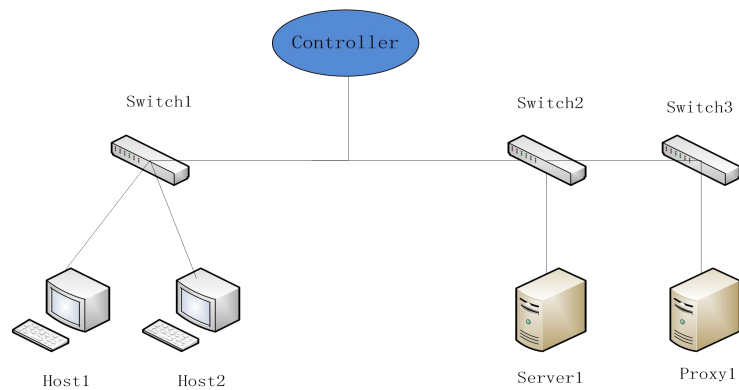


图 3-8 第二题第 1 小题:代理访问

操作要求

1. Web 服务器是 Host1 和 Host2 都可以访问的, 而代理服务器是只有代理用户才可以使用的。
- 2 可以设置 Host1 或 Host2 为代理用户, 可以直接从代理服务器访问到网页。

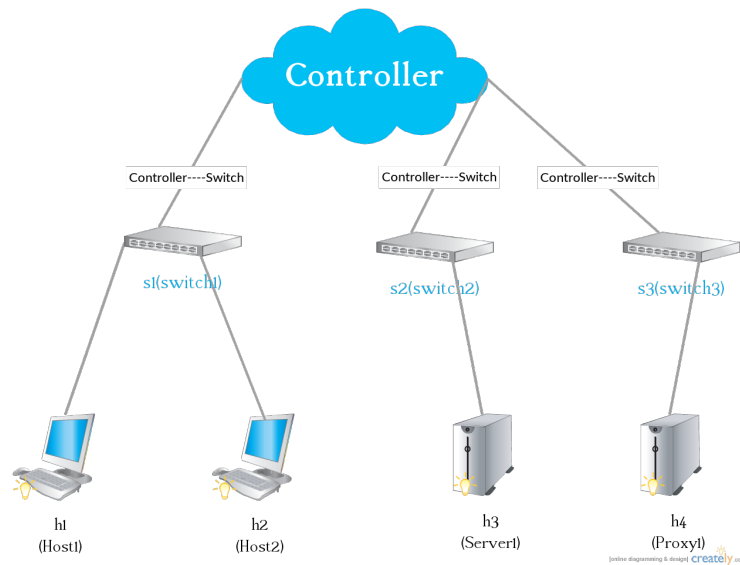


图 3-9 拓扑图

方法描述

拓扑图见图 3-9。为了一般化, 本题同样用 Ryu Controller Python API 实现。采用 Ryu 内置的 simple_switch_13 简单处理路由。编写第三方 Ryu Controller App, 在交换机 s3 连

接 Controller 时为其添加默认流表, 丢弃所有来自 h2 并且发往 h4 的数据包。

实验步骤

1. 执行如下命令, 其中 `setup_network.py` 用于搭建虚拟网络环境, 执行该脚本需要 root 权限。2_1.py 是我们编写的 Ryu Controller App。

```
1 python2 setup_network.py
2 ryu-manager 2_1.py ryu.app.simple_switch_13
```

交换机连接 Controller 后, 会往 Controller 发送 Feature message。此时 Ryu 的 `simple_switch_13` 和 2_1.py 中注册的回调都会被调用。2_1.py 中往 s3 交换机添加流表的如下代码将会发挥作用。

```
1 ...
2 @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
3 def switch_feature_handler(self, ev):
4     datapath = ev.msg.datapath
5     parser = datapath.ofproto_parser
6     dpid = datapath.id
7
8     print 'Switch %d connected.' % dpid
9     if dpid == 3:
10         match = parser.OFPMatch(in_port=1,
11                                 eth_src=self.h2_mac, eth_dst=self.h4_mac)
12         actions = [] # Drop
13         # Flow entry:
14         # table=0, priority=65535,in_port=1,dl_src=00:00:00:00:00:02,
15         # dl_dst=00:00:00:00:00:04
16         # actions=drop
17         self.add_flow(datapath, 65535, match, actions)
18
19 ...
```

2. 在 mininet CLI 中检查网络环境, 输出流表如图 3-10, 其中

`priority=65535,in_port=1,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=drop`

为 2_1.py 为交换机 s3 添加的默认流表。

3. 在 mininet CLI 中执行 `pingall`, 检查网络环境, 结果如图 3-11, `simple_switch_13` 添加了多个流表保证网络通信。只有 h2 与 h4 互相无法 ping 通。

4. 搭建 Web 服务器。先在 mininet CLI 中执行 `h3 xterm &`, 在打开的 xterm 中执行 `cd web && python -m SimpleHTTPServer 80`, 再在 mininet CLI 中执行 `h4 xterm &`, 在打开的 xterm 中执行 `cd proxy && python -m SimpleHTTPServer 80`, 至此, Web Server 和 Web Proxy Server 搭建完毕。

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth2
h4 h4-eth0:s3-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth1
s2 lo: s2-eth1:s1-eth3 s2-eth2:h3-eth0 s2-eth3:s3-eth1
s3 lo: s3-eth1:s2-eth3 s3-eth2:h4-eth0
c0
mininet> dpctl dump-flows -O openflow13
*** s1 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=18.016s, table=0, n_packets=96, n_bytes=13177, priority=0 action
s=CONTROLLER:65535
*** s2 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=18.016s, table=0, n_packets=96, n_bytes=13177, priority=0 action
s=CONTROLLER:65535
*** s3 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=18.015s, table=0, n_packets=96, n_bytes=13177, priority=0 action
s=CONTROLLER:65535
 cookie=0x0, duration=18.015s, table=0, n_packets=0, n_bytes=0, priority=65535, in_port
=1, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:04 actions=drop
mininet>
[17] 0:sudo* "sudo /home/cubelin/D" 15:16 27-May-15
```

图 3-10 输出流表

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
mininet> dpctl dump-flows -O openflow13
*** s1 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=192.813s, table=0, n_packets=137, n_bytes=22415, priority=0 actions=CONTROLLER:65535
 cookie=0x0, duration=128.756s, table=0, n_packets=25, n_bytes=1442, priority=1, in_port=2, dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=147.024s, table=0, n_packets=35, n_bytes=2406, priority=1, in_port=3, dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=128.783s, table=0, n_packets=15, n_bytes=1134, priority=1, in_port=1, dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=147.023s, table=0, n_packets=16, n_bytes=1208, priority=1, in_port=1, dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=128.767s, table=0, n_packets=25, n_bytes=1666, priority=1, in_port=3, dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=128.802s, table=0, n_packets=16, n_bytes=1232, priority=1, in_port=2, dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=128.800s, table=0, n_packets=15, n_bytes=1190, priority=1, in_port=1, dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=128.766s, table=0, n_packets=14, n_bytes=1092, priority=1, in_port=2, dl_dst=00:00:00:00:00:03 actions=output:3
*** s2 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=192.814s, table=0, n_packets=137, n_bytes=22303, priority=0 actions=CONTROLLER:65535
 cookie=0x0, duration=128.773s, table=0, n_packets=15, n_bytes=1134, priority=1, in_port=2, dl_dst=00:00:00:00:00:02 actions=output:1
 cookie=0x0, duration=118.753s, table=0, n_packets=14, n_bytes=980, priority=1, in_port=2, dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=128.789s, table=0, n_packets=16, n_bytes=1176, priority=1, in_port=3, dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=128.785s, table=0, n_packets=40, n_bytes=2576, priority=1, in_port=1, dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=147.025s, table=0, n_packets=30, n_bytes=2300, priority=1, in_port=1, dl_dst=00:00:00:00:00:03 actions=output:2
 cookie=0x0, duration=128.761s, table=0, n_packets=10, n_bytes=532, priority=1, in_port=3, dl_dst=00:00:00:00:00:02 actions=output:1
 cookie=0x0, duration=147.031s, table=0, n_packets=17, n_bytes=1230, priority=1, in_port=2, dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=118.754s, table=0, n_packets=15, n_bytes=1022, priority=1, in_port=3, dl_dst=00:00:00:00:00:03 actions=output:2
*** s3 -----
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=192.813s, table=0, n_packets=133, n_bytes=22047, priority=0 actions=CONTROLLER:65535
 cookie=0x0, duration=192.813s, table=0, n_packets=25, n_bytes=1442, priority=65535, in_port=1, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=128.766s, table=0, n_packets=10, n_bytes=532, priority=1, in_port=2, dl_dst=00:00:00:00:00:02 actions=output:1
 cookie=0x0, duration=128.794s, table=0, n_packets=16, n_bytes=1176, priority=1, in_port=2, dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=128.787s, table=0, n_packets=29, n_bytes=2114, priority=1, in_port=1, dl_dst=00:00:00:00:00:04 actions=output:2
 cookie=0x0, duration=118.759s, table=0, n_packets=15, n_bytes=1022, priority=1, in_port=2, dl_dst=00:00:00:00:00:03 actions=output:1
mininet>
[17] 0:sudo* "sudo /home/cubelin/D" 15:19 27-May-15
```

图 3-11 pingall 结果与流表变化

5. 回到 mininet CLI 分别执行如下命令, 得到输出如图 3-12, 说明只有 h2 无法访问 h4(proxy)。

```
1 h1 curl h3 -m 10
2 h1 curl h4 -m 10
3 h2 curl h3 -m 10
4 h2 curl h4 -m 10
```

```
mininet> h3 xterm &
mininet> h4 xterm &
mininet> h1 curl h3 -m 10
**this_is_the_web_page_from_WEB_Server**
mininet> h1 curl h4 -m 10
**this_is_the_web_page_from_Proxy_Server**
mininet> h2 curl h3 -m 10
**this_is_the_web_page_from_WEB_Server**
mininet> h2 curl h4 -m 10
curl: (28) Connection timed out after 10001 milliseconds
mininet>
```

图 3-12 h1, h2 分别请求 h3, h4 网页的结果

实验结果

利用 Ryu Controller 提供的 Python API, 自动化实现问题要求。

3.2.2 第 2 小题：流表管理

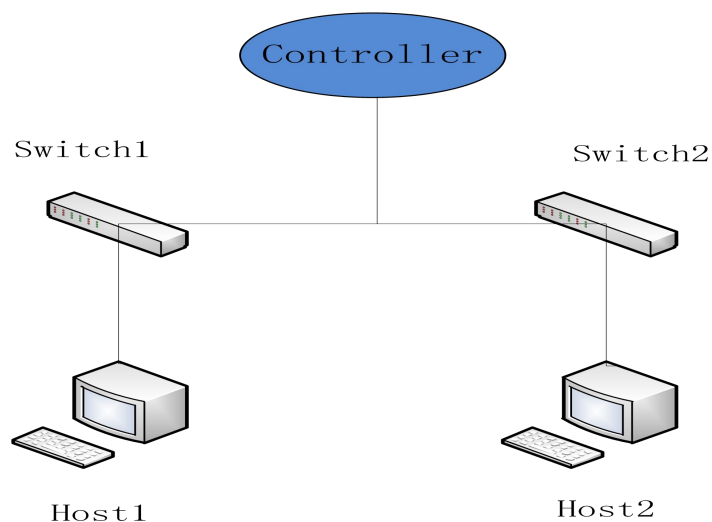


图 3-13 第二题第 2 小题:流表管理

网络环境要求

设有若干台 PC 机 (Host1,Host2), 若干台交换机 (Switch1,Switch2), 如图 4 所示 (作为

示例拓扑, 实际拓扑可自由选择)。(PC 机、交换机、控制器可以选择物理设备或者虚拟设备实现)

操作要求

利用控制器提供的 API(例如 REST API), 开发一个网络及流表管理工具 (客户端, 网页端均可)。管理工具可以显示网络拓扑结构, 查看流表, 增加流表, 删除流表。

3.3 第三题: 设计题

3.3.0.1 第三层

测试测试测试测试测试测试测试测试测试测试测试测试。¹

3.4 字体

普通**粗体**斜体

黑体楷体仿宋

3.5 公式

单个公式, 公式引用: [公式 3.1](#)。

$$c^2 = a^2 + b^2 \quad (3.1)$$

多个公式, 公式引用: [公式 3.2a](#), [公式 3.2b](#)。

$$F = ma \quad (3.2a)$$

$$E = mc^2 \quad (3.2b)$$

3.6 罗列环境

1. 第一层
2. 第一层
 - 2.1 第二层
 - 2.2 第二层

¹脚注

a) 第三层

b) 第三层

解释环境 解释内容

四 其他格式测试

4.1 代码环境

```
1 import os
2
3 def main():
4     '''
5     doc here
6     '''
7     print 'hello, world' # Abc
8     print 'hello, 中文' # 中文
```

4.2 定律证明环境

定义 4.1. 这是一个定义。

命题 4.1. 这是一个命题。

公理 4.1. 这是一个公理。

引理 4.1. 这是一个引理。

定理 4.1. 这是一个定理。

证明. 这是一个证明。

□

4.3 算法环境

算法 4.1: How to write algorithms

Data: this text

Result: how to write algorithm with L^AT_EX2_ε

```
1 initialization;
2 while not at end of this document do
3     read current;
4     if understand then
5         go to next section;
6         current section becomes this one;
7     else
8         go back to the beginning of current section;
9     end
10 end
```

4.4 表格

表格见[表 4.1](#)。

表 4.1 一个表格

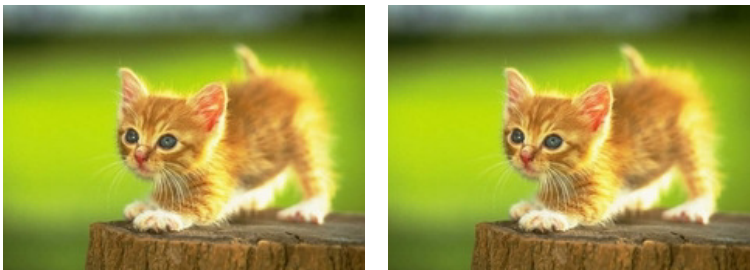
a	b
c	d

4.5 图片

图片见[图 4-1](#)。图片格式支持 eps, png, pdf 等。多个图片见[图 4-2](#), 分开引用:[图 4-2a](#), [图 4-2b](#)。



图 4-1 一个图片



(a) 图片 1

(b) 图片 2

图 4-2 多个图片

4.6 参考文献示例

这是一篇中文参考文献^[1];这是一篇英文参考文献^[2];同时引用^[1,2]。

4.7 \autoref 测试

公式 [公式 3.1](#)

脚注 [脚注 1](#)

项 [第 1 项](#),[第 2.1 项](#),[第 2.2a 项](#)

图 [图 4-1](#)

表 [表 4.1](#)

附录 [附录 B](#)

章 [第一章](#)

小节 [3.1 小节](#),[3.1.1 小节](#),[3.2 小节](#)

算法 [算法 4.1](#),[第 1 行](#)

证明环境 [定义 4.1](#),[命题 4.1](#),[公理 4.1](#),[引理 4.1](#),[定理 4.1](#),[证明 1](#)

致 谢

致谢正文。

参考文献

- [1] T_EXGuru. L^AT_EX 2_ε Manual. 1999.
- [2] Donald E Knuth. The T_EXbook. MA: Addison–Wesley Pub. Co., 1984.

附录 A UniqueSDNStudio 初赛文档

- [1] 论文 1
- [2] 论文 2

附录 B 这是一个附录

附录正文。

2015 年 5 月 31 日