

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
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	12
3.3 第三题: 设计题	21
参考文献	24

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。联创团队的参赛队伍共有两支。本组名为 UniqueSDNStudio, 由三名大二本科生组成, 并安排有一位研究生作为指导老师。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组与在线讨论平台, 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“创非凡之事”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 指导老师

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。

华中科技大学联创团队

李英儒

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

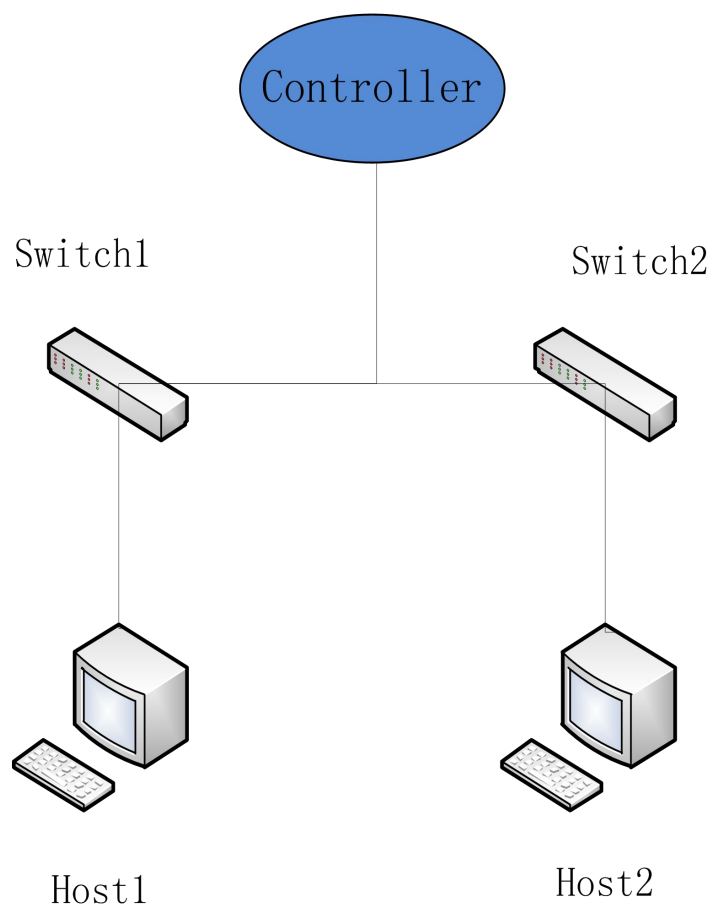


图 3-1 第一题第 1 小题:简单网络

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。

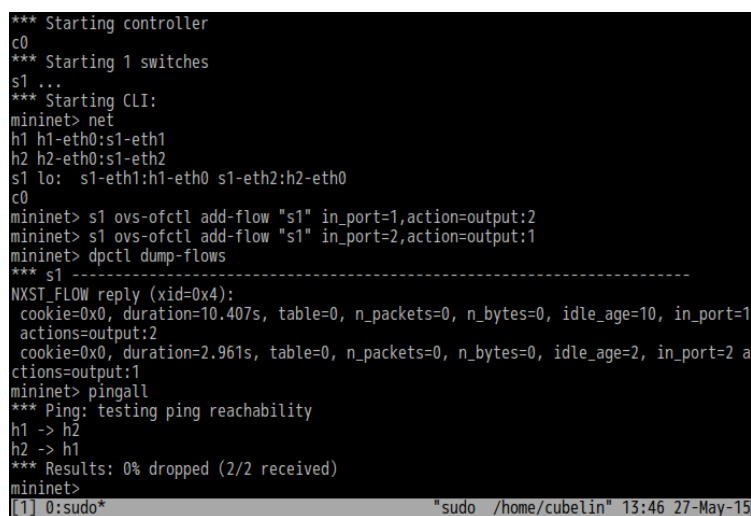


图 3-2 检查虚拟网络环境

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。

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。


```
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> 
```

[1] 0:sudo* "sudo /home/cubelin" 13:47 27-May-15

图 3-3 删除流表后检查虚拟网络环境

3.1.2 第 2 小题：访问限制

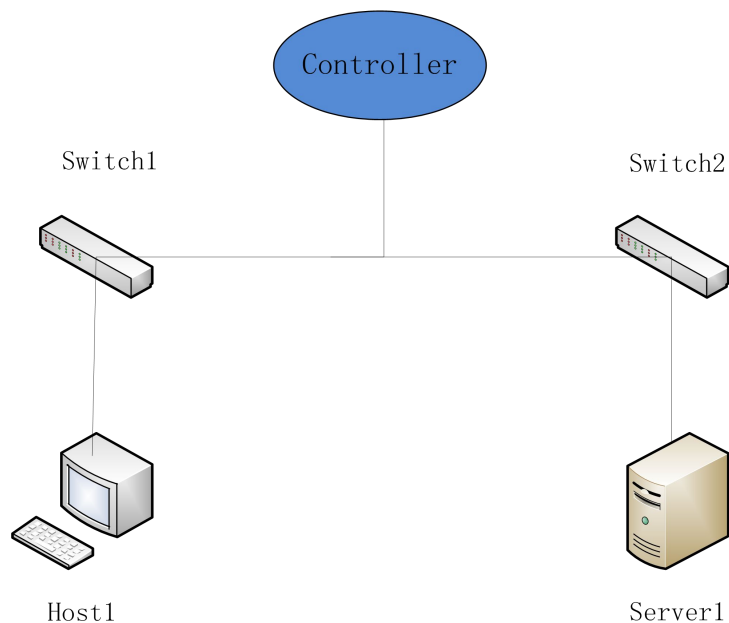


图 3-4 第一题第 1 小题:访问限制

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。

方法描述

拓扑环境如图 3-4, 其中 Host1 为 h1, Server1 为 h2。

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。与第一小题类似的, 每个交换机各有两个 flow entry 保证主机 h1 和 h2 的正常通信。

s2 中有一个额外的 flow entry, 实现将 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, 检查此包来源的 mac 是否在白名单中, 并判断此包是否为 TCP 三次握手包 ($ACK \leq 1$)。若此包并不是 TCP 三次握手

包, 而且其来源 mac 地址不在白名单中, Controller 将立即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, 阻断其请求, 但是仍可接收来自 h2 的返回, 之后将此 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, 此后的请求将不会被阻止。
流程伪代码如下:

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_packet from h1 do
4   | Send TCP_packet to port 2 and its copy to RyuController ;
5 end
6 CONTROL PLANE[Software Layer];
7 while Recieve TCP_packet do
8   | if TCP_packet.source_mac  $\notin$  WhiteList and TCP_packet.ACK > 1 then
9     |   Send Flow Entry with hard_timeout = 60 to Drop HTTP Package from
        |   TCP_packet.source_mac to h2.mac;
10    |   White List  $\leftarrow$  White List  $\cup$  {TCP_packet.source_mac} ;
11    | end
12 end

```

实验步骤

1. 进入附件 1.1 目录, 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>

```

图 3-5 检查网络环境

5. 等待 60 秒,多次执行 `h1 curl h2`,均能得到如下正常返回

****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>
```

[9] 0:sudo* "sudo /home/cubelin/D" 14:48 27-May-15

图 3-6 检查流表

```
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>
```

[9] 0:sudo* "sudo /home/cubelin/D" 14:51 27-May-15

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

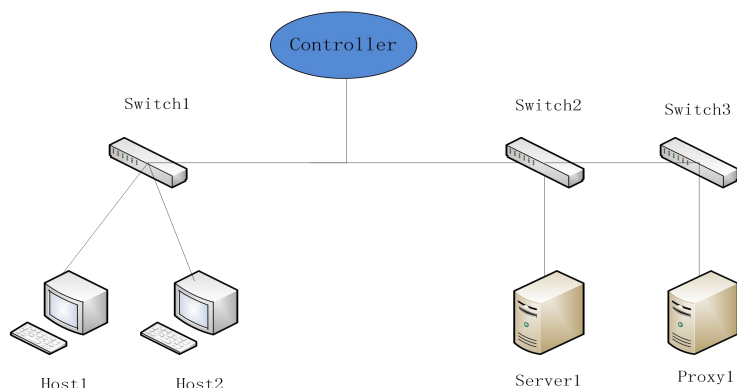


图 3-8 第二题第 1 小题:代理访问

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)

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。

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。

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. 进入附件 2.1 目录, 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。

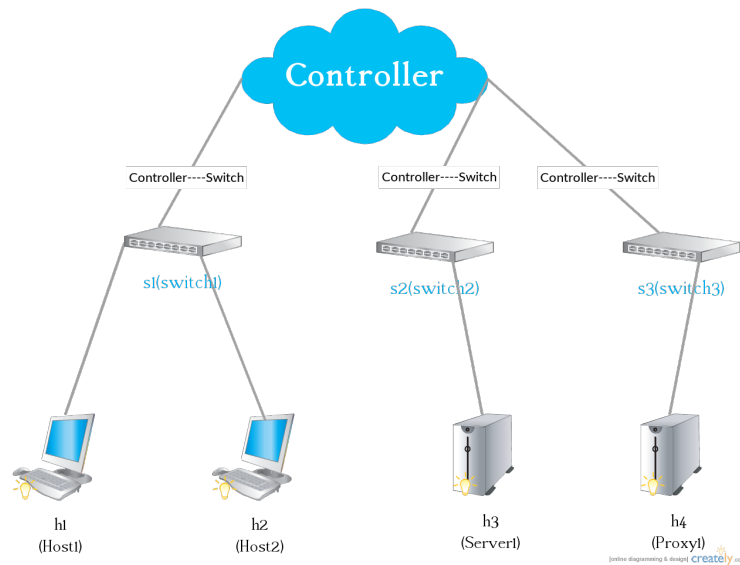


图 3-9 拓扑图

```

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。

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> 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

实验结果

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

```
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,dst=00:00:00:00:00:04 actions=output:3
cookie=0x0, duration=147.024s, table=0, n_packets=33, n_bytes=2406, priority=1,in_port=3,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,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,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,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,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,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,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,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,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,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,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,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,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,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,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,dst=00:00:00:00:00:02,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,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,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,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,dst=00:00:00:00:00:03 actions=output:1
mininet>
```

图 3-11 pingall 结果与流表变化

```
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

3.2.2 第 2 小题：流表管理

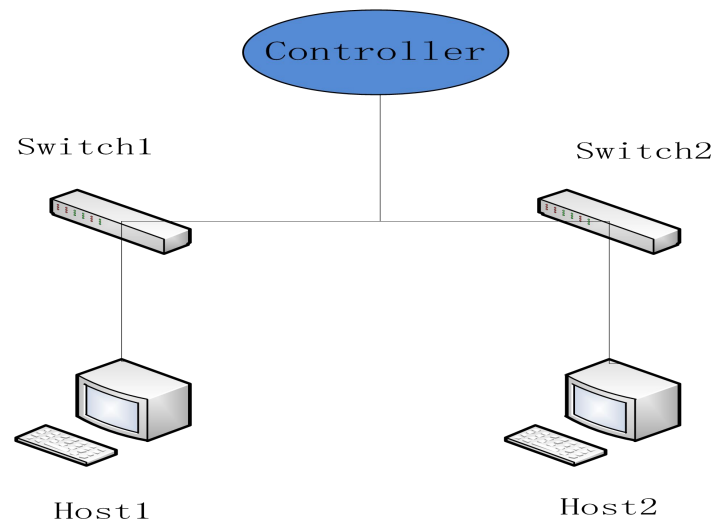


图 3-13 第二题第 2 小题:流表管理

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。

方法描述

基于 Ryu 提供的北向接口开源实现, 添加了少量独立开发的北向接口作为管理工具与 Controller 交互的接口, 封装实现网络及流表的管理后台, 前端使用 d3.js 绘制拓扑结构, 并使用 Bootstrap 框架构建网页端。

实验步骤

1. 进入附件 2.2 目录, 执行如下代码运行管理工具后端。

```
1 ryu-manager --observe-links ryu.app.ofctl_rest rest_topology.py simple_switch_13.py
2 python wrapper.py
```

此时便可以通过 <http://localhost:8000/static/index.html> 预览管理工具, 但是在交换机连接 Controller 之前, 网页端不会显示任何相关信息。

补充说明: 为了支持在网络拓扑中显示主机, 我们对 Ryu 提供的 `rest_topology.py` 进行了简单的修改, 通过监测网络中通过交换机的 ARP 包和 IPv4 包, 维护一个主机的状态表, 并在前端请求拓扑结构时返回。我们还对 `simple_switch_13.py` 进行了修改, 让其添加的路由 `flow entry` 拥有过期时间, 便于 Controller 及时更新状态表。

2. 执行如下命令, 用 mininet 构建虚拟网络环境, 并连接 Controller。

```
1 sudo mn --switch ovsk --topo tree,3,fanout=3 --controller remote,ip=127.0.0.1
```

3. 在 mininet CLI 中执行 `pingall`, 让 Controller 发现主机。

4. 访问 <http://localhost:8000/static/index.html>, 可以看到拓扑图以及所有交换机上的流表。之后如若交换机甚至是整个网络环境断开重新连接, 只需要点击两个 Refresh 按钮刷新即可。

网络拓扑部分显示如图 3-14。

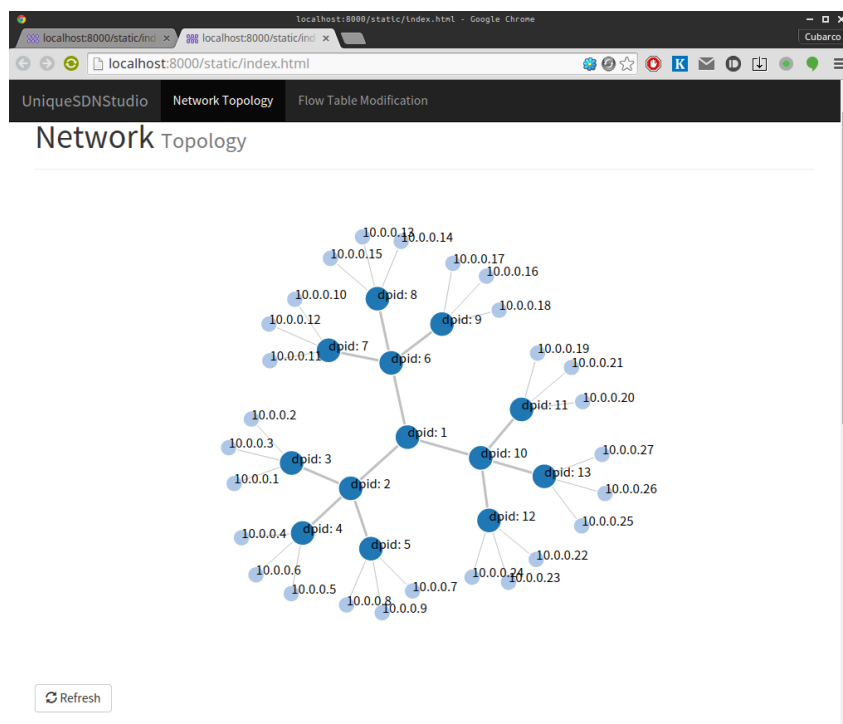


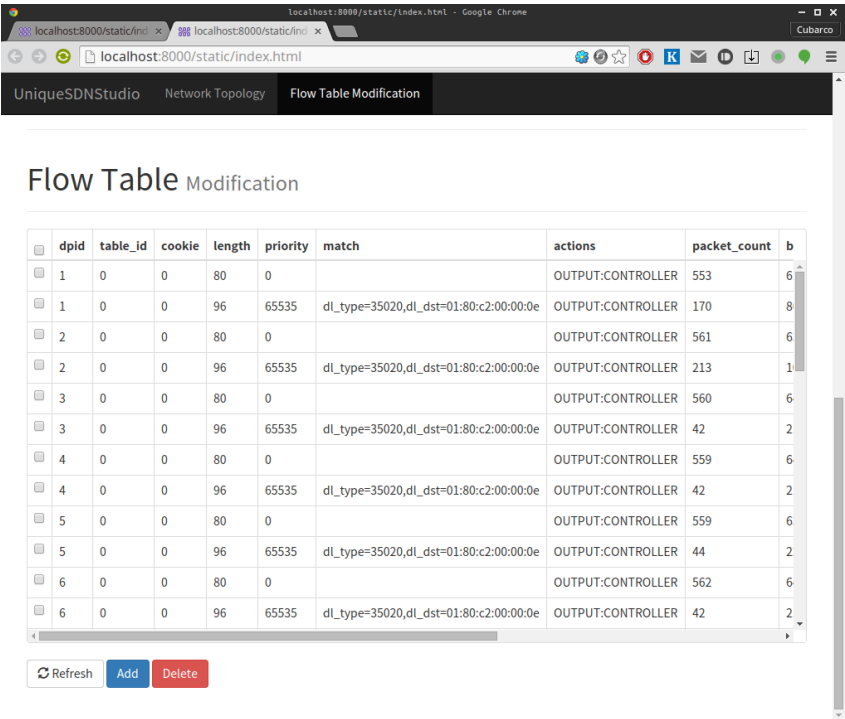
图 3-14 网络拓扑图

流表部分显示如图 3-15。

5. 预设若 6 分钟无某主机的包通过交换机, 将会从状态表中删除该主机的记录。6 分钟后点击 Refresh 按钮, 网络拓扑如图 3-16。

6. 测试删除 flow entry 的功能: 勾选相应的 flow entry, 点击 Delete 按钮, 确认后, 相应的 flow entry 将会被删除, 并从网页中消失, 过程见图 3-17。

注意: 选择相应 flow entry 之前务必点击 Refresh 按钮刷新。



	dpid	table_id	cookie	length	priority	match	actions	packet_count	b
<input type="checkbox"/>	1	0	0	80	0		OUTPUT:CONTROLLER	553	6
<input type="checkbox"/>	1	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	170	8
<input type="checkbox"/>	2	0	0	80	0		OUTPUT:CONTROLLER	561	6
<input type="checkbox"/>	2	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	213	1
<input type="checkbox"/>	3	0	0	80	0		OUTPUT:CONTROLLER	560	6
<input type="checkbox"/>	3	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	42	2
<input type="checkbox"/>	4	0	0	80	0		OUTPUT:CONTROLLER	559	6
<input type="checkbox"/>	4	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	42	2
<input type="checkbox"/>	5	0	0	80	0		OUTPUT:CONTROLLER	559	6
<input type="checkbox"/>	5	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	44	2
<input type="checkbox"/>	6	0	0	80	0		OUTPUT:CONTROLLER	562	6
<input type="checkbox"/>	6	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	OUTPUT:CONTROLLER	42	2

图 3-15 流表管理

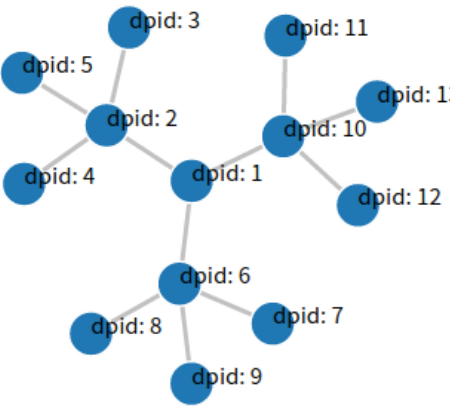


图 3-16 6 分钟后的网络拓扑图

从图 3-14 可以看到,主机 h1 直接接在 s3 交换机上,所以在 mininet CLI 中执行如下命令:

```
1 h1 ping h4
2 h1 ping h5
3 h4 ping h5
```

结果如图 3-18 所示, h1 无法与外界通信, 而 s1 交换机外的环境通信正常, 说明 flow entry 成功删除。

Flow Table Modification

<input type="checkbox"/>	dpid	table_id	cookie	length	priority	match	a
<input type="checkbox"/>	1	0	0	80	0		C
<input type="checkbox"/>	1	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input type="checkbox"/>	2	0	0	80	0		C
<input type="checkbox"/>	2	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input checked="" type="checkbox"/>	3	0	0	80	0		C
<input checked="" type="checkbox"/>	3	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input type="checkbox"/>	4	0	0	80	0		C
<input type="checkbox"/>	4	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C

(a) 选中与 s3 交换机相关的 flow entry

Flow Table Modification

<input type="checkbox"/>	dpid	table_id	cookie	length	priority	match	a
<input type="checkbox"/>	1	0	0	80	0		C
<input type="checkbox"/>	1	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input type="checkbox"/>	2	0	0	80	0		C
<input type="checkbox"/>	2	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input type="checkbox"/>	4	0	0	80	0		C
<input type="checkbox"/>	4	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C
<input type="checkbox"/>	5	0	0	80	0		C
<input type="checkbox"/>	5	0	0	96	65535	dl_type=35020,dl_dst=01:80:c2:00:00:0e	C

(b) 删除后

图 3-17 删除流表操作

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
^C
--- 10.0.0.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4000ms
pipe 4
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
^C
--- 10.0.0.5 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6008ms
pipe 4
mininet> h4 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=6.06 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.349 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.102 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.076 ms
^C
--- 10.0.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.076/1.647/6.064/2.552 ms
mininet>
```

图 3-18 ping 结果图

7. 测试添加 flow entry 的功能: 在上述实验步骤的前提下, 点击 Add 按钮, 如图 3-19 填写相关信息 (match 项填空格表示匹配所有), output 端口填 4294967293 表示发给 Controller. 再次点击 Add 按钮添加, 如果添加成功将在网页上显示新的 flow entry.

在 mininet CLI 中执行 pingall, 发现 h1 能 ping 通所有主机, 表示 flow entry 添加成功, 结果见图 3-20。

实验结果

显示网络拓扑图, 查看流表, 增加流表, 删除流表等功能测试成功。

Add a flow entry

dpid*:

3

table_id:

cookie:

cookie_mask:

idle_timeout:

hard_timeout:

priority:

buffer_id:

flags:

match*:

actions*:

type:output,port=4294967293

Close

Add

图 3-19 添加 flow entry

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h17 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
h18 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h19 h20 h21 h22 h23 h24 h25 h26 h27
h19 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h20 h21 h22 h23 h24 h25 h26 h27
h20 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h21 h22 h23 h24 h25 h26 h27
h21 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h22 h23 h24 h25 h26 h27
h22 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h23 h24 h25 h26 h27
h23 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h24 h25 h26 h27
h24 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h25 h26 h27
h25 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h26 h27
h26 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h27
h27 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26
*** Results: 0% dropped (702/702 received)
mininet>
[8] 0:sudo* *sudo /home/cubelin/D* 18:17 31-May-15
```

图 3-20 添加 flow entry 后 pingall

3.3 第三题: 设计题

选题背景

随着互联网逐渐渗透到人们生活的每一个角落,人们的生活也变得越来越便利了。但是网络也是一把双刃剑。在它越来越多的控制人们的生活的同时,它不安全的一面也渐渐暴露出来。网络攻击随处可见,造成了难以估计的损失。对于一般民众而言,恶意网站、钓鱼链接是对他们数据、财产安全最大的威胁,需要小心应对。

设计细节

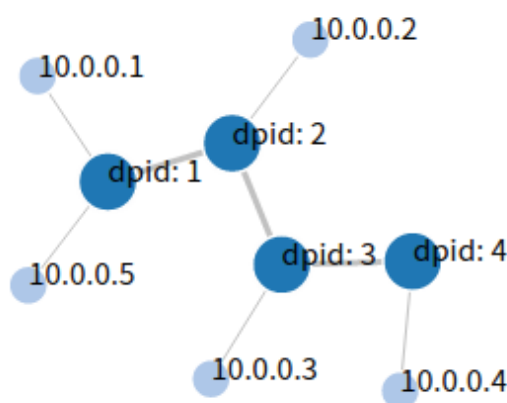


图 3-21 设计题网络拓扑图

如图 3-21 所示,网络为线性拓扑,共 4 个 host,分别与 4 个交换机相连(s1,s2,s3,s4)。这四个交换机互相之间也有连接。为了使 hosts 可以访问外网,故使用了 nat 参数使得本机桥接到 mininet 虚拟网络环境内,作为 h5 连接在 s1 上。h1,h2 的网关是 h3。将 controller 上的 dnsserver 设置为 h4 的 mac 地址,将 dns 请求全部劫持到 h4 上。h4 上运行 fakedns.py 来处理所有 dns 请求。

设计与实现方案: controller 运行一个经过修改的交换机算法。对于除 dns 包以外的数据包按正常的交换机进行学习和转发,对于 dns 查询数据包则将目的 mac 地址强制设为 dnsserver 然后再进行交换算法。这样就可以将 dns 查询报文劫持到 h4 上。而 h4 上运行的 fakedns 将过滤出所有 dns 包并检查其查询的网址是否在黑名单中,若在黑名单中则不进行响应,否则将该数据包正常送往网关,然后将回应数据包转回来源 host。

实验步骤

1. 执行如下命令创建网络拓扑,得到如图 3-22 的网络结构。


```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
nat0 nat0-eth0:s1-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2 s1-eth3:nat0-eth0
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo: s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo: s4-eth1:h4-eth0 s4-eth2:s3-eth3
c0
```

图 3-22 创建网络拓扑

```
1 mn --controller=remote --topo=linear,4 --nat
```

2. 进入附件的目录 3, 执行如下命令运行 Controller:

```
1 ryu-manager 3.py
```

3. 在 mininet CLI 中执行 pingall, 结果如图 3-23, 说明通信正常。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 nat0
h2 -> h1 h3 h4 nat0
h3 -> h1 h2 h4 nat0
h4 -> h1 h2 h3 nat0
nat0 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

图 3-23 检查网络环境

4. 检查各主机网络设置, 结果见图 3-24a, 图 3-24b, 图 3-24c。在 mininet CLI 中执行 h1 ip r replace default via 10.0.0.3 配置 h1 的默认路由, 并执行 h1 ip r 检查结果, 见图 3-24d。

```
mininet> h1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred lft forever
2: h1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether f2:e5:26:c2:e3:14 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.1/8 brd 10.255.255.255 scope global h1-eth0
        valid lft forever preferred lft forever
    inet6 fe80::f0e5:26ff:fe2c:e314/64 scope link
        valid lft forever preferred lft forever
```

(a) h1 网络设置

```
mininet> h3 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred lft forever
2: h3-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether b2:f0:68:3b:f7:dc brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.3/8 brd 10.255.255.255 scope global h3-eth0
        valid lft forever preferred lft forever
    inet6 fe80::b0f0:68ff:fe3b:f7dc/64 scope link
        valid lft forever preferred lft forever
```

(b) h3 网络设置

```
mininet> h4 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred lft forever
2: h4-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 3a:12:8c:e5:53:c5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.4/8 brd 10.255.255.255 scope global h4-eth0
        valid lft forever preferred lft forever
    inet6 fe80::3812:8cff:fe55:53c5/64 scope link
        valid lft forever preferred lft forever
```

(c) h4 网络设置

```
mininet> h1 ip r
default via 10.0.0.3 dev h1-eth0
10.0.0.0/8 dev h1-eth0 proto kernel scope link src 10.0.0.1
```

(d) h1 的路由表配置

图 3-24 各主机网络设置

5. 在 h4 上运行 fakedns.py, 配置文件中默认屏蔽 baidu.*, 如图 3-25

```
.root@hyrslaptop FakeDns-master]# ./fakedns.py
JARNING: No route found for IPv6 destination :: (no default route?)
JARNING: Please, report issues to https://github.com/phaethon/scapy
> Parsing rules...
> baidu.* -> 127.0.0.1
> 1 rules parsed
```

图 3-25 运行 fakedns.py

6. 在 mininet CLI 中执行 `h1 nslookup www.google.com`, 结果如图 3-26a, 正常得到返回。fakedns.py 的日志见图 3-26b。

```
mininet> h1 nslookup www.google.com
Server:          202.114.0.242
Address:         202.114.0.242#53

Non-authoritative answer:
Name:   www.google.com
Address: 64.233.187.99
Name:   www.google.com
Address: 64.233.187.106
Name:   www.google.com
Address: 64.233.187.104
Name:   www.google.com
Address: 64.233.187.103
Name:   www.google.com
Address: 64.233.187.105
Name:   www.google.com
Address: 64.233.187.147
```

(a) 查询 www.google.com 的 DNS 记录

```
Begin emission:
*Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
*
Sent 1 packets.
b'www.google.com.'
```

(b) fakedns.py 的日志

图 3-26 www.baidu.com 的 DNS 记录查询情况

7. 在 mininet CLI 中执行 `h1 nslookup www.baidu.com`, 结果如图 3-27a, 查询超时。fakedns.py 的日志见图 3-27b。

```
mininet> h1 nslookup www.baidu.com
;; connection timed out; no servers could be reached
```

(a) 查询 www.baidu.com 的 DNS 记录

```
url in blacklist, dropping.
b'www.baidu.com.'
```

(b) fakedns.py 的日志

图 3-27 www.baidu.com 的 DNS 记录查询情况

试验结果

此方案可以成功阻止对恶意网址的 dns 查询, 进而保护用户安全

参考文献

- [1] Nadeau, Thomas D. SDN: Software Defined Networks. O'Reilly Media; 1 edition (September 7, 2013).
- [2] Ryu sdn framework, 2014. <http://osrg.github.io/ryu-book/en/Ryubook.pdf>.
- [3] Openflow switch specification, version 1.3.1 (wire protocol 0x04), September 6, 2012. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf>.
- [4] Openflow switch specification, version 1.1.0 implemented (wire protocol 0x02), February 28, 2011. <http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf>.

2015 年 5 月 31 日