实验三

一． 实验目的

使用 Pyretic 实现第二层 Mac 地址上的防火墙

二． 实验环境

1. VMWare 虚拟机
2. 安装 mininet virtual machine，这在实验二的时候介绍过了，就不重复介绍了。
3. 安装 putty，安装 putty 的方法在实验二也介绍过了，也不重复介绍了。

三． 实验内容

1. 网络拓扑结构

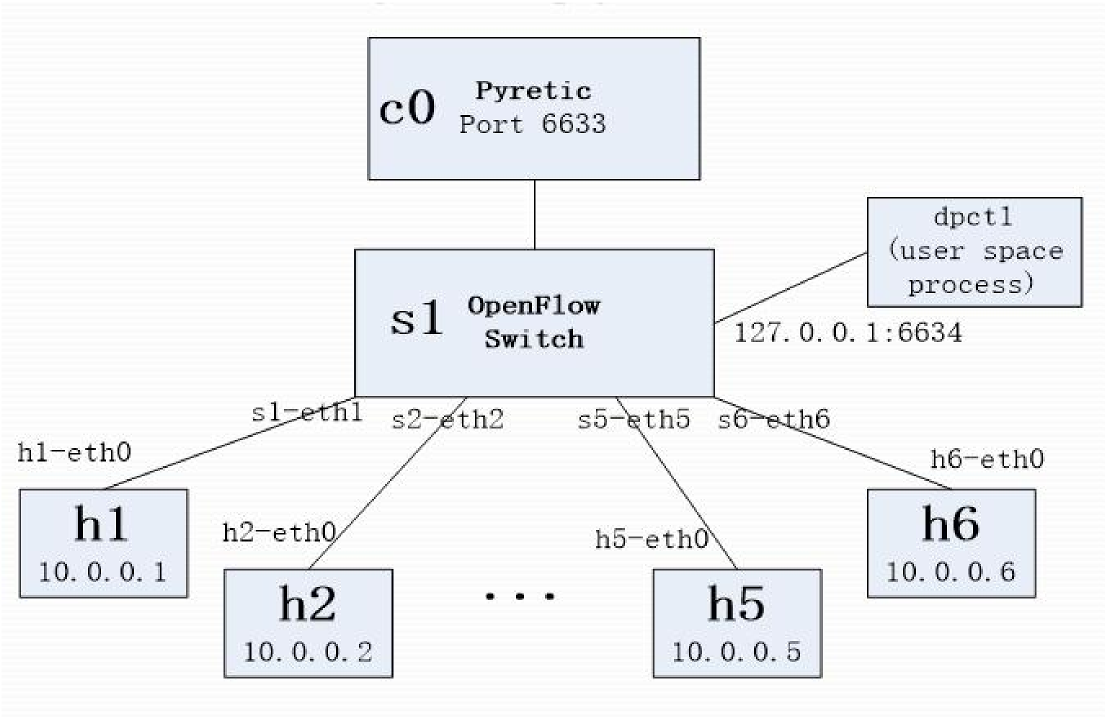

图 1 网络拓扑结构

我们需要实现的网络拓扑结构如图 1。这是一个简单的网络拓扑结构。一个 OpenFlow 交换机控制着 6 个主机，而交换机又由 Pyretic 控制器控制着。

2. 防火墙代码与相关解释

（1）引入头文件

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.examples.pyretic_switch import act_like_switch

import os, csv
from csv import DictReader
```

（2）确定防火墙规则文件地址

```
policy_file = "%s/pyretic/pyretic/examples/firewall-policies.csv" % os.environ['
HOME']
```

（3）主函数：

① 初始化防火墙规则为 none

```
not_allowed = none
```

② 读入防火墙规则文件的数据

```
# read data from policy file
with open(policy_file, "r") as policy_content:
```

③ 根据防火墙规则文件的数据确定不训练连接的网络链路

```
 # read data from policy file
with open(policy_file, "r") as policy_content:
  dictReader = csv.DictReader(policy_content)
  # add the forbidden policy and not allow the two side communication
  for d in dictReader:
    not_allowed = not_allowed + (match(srcmac=MAC(d['mac_0']))&match(dstmac=MA
C(d['mac_1'])))+(match(srcmac=MAC(d['mac_1']))&match(dstmac=MAC(d['mac_0'])))
# add the allowed rules
```

④ 确定允许连接的网络链路，它相当于不允许的链路取反

```
# add the allowed rules
allowed = ~not_allowed
```

⑤ 输出当前的允许的链路连接规则

```
# print allowed
print allowed
```

⑥ 将允许的规则传递 act_like_switch()

```
# regard the allowed switch input as the act_like_switch of pyretic_switch
return allowed>>act_like_switch()
```

我将这个代码保存在了$HOME/pyretic/pyretic/examples/pyretic_firewall.py 中

3. 防火墙规则

```
id,mac_0,mac_1
1,00:00:00:00:00:01,00:00:00:00:00:04
2,00:00:00:00:00:02,00:00:00:00:00:05
3,00:00:00:00:00:03,00:00:00:00:00:06
~
~
```

我制定的规则是节点 1 和节点 4 之间不能相互通信，节点 2 和节点 5 之间不同相互通信，节点 3 和节点 6 之间不同相互通信。我将这个防火墙规则文件保存在$HOME/pyrertic/pyretic/examples/firewall-policies.csv 中。

四．实验结果

1. 启动控制器

输入 pyretic.py -v high pyretic.examples.pyretic_firewall

```
mininet@mininet:~/pyretic$ pyretic.py -v high pyretic.examples.pyretic_firewall_
```

得到如下的执行结果：

```
        match:
            ('srcmac', 00:00:00:00:00:04)
        match:
            ('dstmac', 00:00:00:00:00:01)
    sequential:
        match:
            ('srcmac', 00:00:00:00:00:02)
        match:
            ('dstmac', 00:00:00:00:00:05)
    sequential:
        match:
            ('srcmac', 00:00:00:00:00:05)
        match:
            ('dstmac', 00:00:00:00:00:02)
    sequential:
        match:
            ('srcmac', 00:00:00:00:00:03)
        match:
            ('dstmac', 00:00:00:00:00:06)
    sequential:
        match:
            ('srcmac', 00:00:00:00:00:06)
        match:
            ('dstmac', 00:00:00:00:00:03)
```

这是允许的链路规则，说明我们的防火墙代码已经生效了。

2. 创建拓扑

(1) 首先先打开 putty，将其连接到我的虚拟机上。连接虚拟机需要 ip 地址，所以我先用 ifconfig 查看了我的 ip 地址。我当前的 ip 地址如下：

```
attempting get output of of_client:
INFO:core:POX 0.2.0 (carp) is up.
INFO:core:Going down...

pyretic.py done
mininet@mininet:~/pyretic$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b1:a3:7f
          inet addr:192.168.43.131  Bcast:192.168.43.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb1:a37f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15946 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2273 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1092496 (1.0 MB)  TX bytes:323753 (323.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3829 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3829 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:737406 (737.4 KB)  TX bytes:737406 (737.4 KB)

mininet@mininet:~/pyretic$ _
```

从图中的 eth0 的 inet addr 可以看出我的 ip 地址是 192.168.43.131

(2) 通过 putty 连接上虚拟机，然后输入账号密码登陆。登陆后的结果如下图：

mininet@mininet: ~/pyretic

```
login as: mininet
mininet@192.168.43.131's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-23-generic i686)

 * Documentation:  https://help.ubuntu.com/

  System information as of Sat Jun  2 04:59:33 PDT 2018

  System load:  0.0               Processes:           104
  Usage of /:   8.4% of 18.94GB   Users logged in:     1
  Memory usage: 8%                IP address for eth0: 192.168.43.131
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/

New release '14.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Jun  1 07:04:08 2018
```

(3) 创建一个 6 个节点，一个交换机的网络。

mininet@mininet: ~/pyretic

```
*** Stopping 1 controllers
c0
*** Done
completed in 179.150 seconds
mininet@mininet:~/pyretic$ sudo mn --topo single,6 --mac ovsk --controller remot
e
[sudo] password for mininet:
Sorry, try again.
[sudo] password for mininet:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

(4) 此时可以发现在虚拟机上，出现了以下信息：

```
OpenFlow switch 1 connected
2018-06-02 06:16:38.332554  | clear_all
2018-06-02 06:16:38.334117  | clear_all
2018-06-02 06:16:38.335051  | clear_all
2018-06-02 06:16:38.335862  | clear_all
2018-06-02 06:16:38.336878  | clear_all
2018-06-02 06:16:38.338677  | clear_all
2018-06-02 06:16:38.339574  | clear_all
2018-06-02 06:16:38.995153  | clear_all
_
```

这说明网络拓扑正式建立，网络拓扑中的控制机也连接上了。

3. Pingall 测试

(1) 我们在 mininet 中进行 pingall 测试，可以得到如下的结果：

```
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X h5 h6
h2 -> h1 h3 h4 X h6
h3 -> h1 h2 h4 h5 X
h4 -> X h2 h3 h5 h6
h5 -> h1 X h3 h4 h6
h6 -> h1 h2 X h4 h5
*** Results: 20% dropped (6/30 lost)
mininet> 
```

(2) 同时我们可以发现在虚拟机会显示如下的信息：

下面是部分信息截图

```
        ('srcmac', 00:00:00:00:00:06)
        ('srcport', 8)
        ('dstmac', 00:00:00:00:00:05)
        ('inport', 6)
        ('switch', 1)
        ('ethtype', 2048)
        ('tos', 0)
        ('srcip', 10.0.0.6)
        ('dstport', 0)
[{'outport': 5}]
2018-06-02 06:19:50.088408  | install rule
match:
        ('dstip', 10.0.0.6)
        ('protocol', 1)
        ('srcmac', 00:00:00:00:00:05)
        ('srcport', 0)
        ('dstmac', 00:00:00:00:00:06)
        ('inport', 5)
        ('switch', 1)
        ('ethtype', 2048)
        ('tos', 0)
        ('srcip', 10.0.0.5)
        ('dstport', 0)
[{'outport': 6}]
_
```

4. 单个交换机 Ping 测试

以节点 1 对其他节点的连通性为例，进行测试：

节点 1 与节点 2：

```
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_req=1 ttl=64 time=64.0 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.029 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_req=6 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_req=7 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_req=8 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_req=9 ttl=64 time=0.036 ms
64 bytes from 10.0.0.2: icmp_req=10 ttl=64 time=0.028 ms
64 bytes from 10.0.0.2: icmp_req=11 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_req=12 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_req=13 ttl=64 time=0.049 ms
^C
--- 10.0.0.2 ping statistics ---
```

节点 1 与节点 3：

```
mininet@mininet: ~/pyretic
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=85.4 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.029 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.213 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=0.062 ms
64 bytes from 10.0.0.3: icmp_req=5 ttl=64 time=0.042 ms
64 bytes from 10.0.0.3: icmp_req=6 ttl=64 time=0.066 ms
^C
--- 10.0.0.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.029/14.312/85.462/31.819 ms
```

节点 1 与节点 4：

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
From 10.0.0.1 icmp_seq=11 Destination Host Unreachable
From 10.0.0.1 icmp_seq=12 Destination Host Unreachable
^C
--- 10.0.0.4 ping statistics ---
15 packets transmitted, 0 received, +3 errors, 100% packet loss, time 13999ms
pipe 3
```

节点 1 与节点 5：

```
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_req=1 ttl=64 time=89.6 ms
64 bytes from 10.0.0.5: icmp_req=2 ttl=64 time=0.058 ms
64 bytes from 10.0.0.5: icmp_req=3 ttl=64 time=0.052 ms
64 bytes from 10.0.0.5: icmp_req=4 ttl=64 time=0.032 ms
64 bytes from 10.0.0.5: icmp_req=5 ttl=64 time=0.086 ms
^C
--- 10.0.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.032/17.975/89.651/35.838 ms
```

节点 1 与节点 6：

```
mininet> h1 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_req=1 ttl=64 time=50.1 ms
64 bytes from 10.0.0.6: icmp_req=2 ttl=64 time=0.056 ms
64 bytes from 10.0.0.6: icmp_req=3 ttl=64 time=0.046 ms
64 bytes from 10.0.0.6: icmp_req=4 ttl=64 time=0.034 ms
64 bytes from 10.0.0.6: icmp_req=5 ttl=64 time=0.034 ms
64 bytes from 10.0.0.6: icmp_req=6 ttl=64 time=0.044 ms
64 bytes from 10.0.0.6: icmp_req=7 ttl=64 time=0.041 ms
64 bytes from 10.0.0.6: icmp_req=8 ttl=64 time=0.043 ms
64 bytes from 10.0.0.6: icmp_req=9 ttl=64 time=0.034 ms
^C
--- 10.0.0.6 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8002ms
rtt min/avg/max/mdev = 0.034/5.613/50.192/15.761 ms
mininet>
```

从以上的结果图，我们可以很容易的看出节点 1 和节点 2，3，5，6 之间是连通的，而与节点 4 是不连通的。

五．总结

1. 从实验结果上我们可以看出我们防火墙生效了。因为在 pingall 的时候，节点 1 与节点 4、节点 2 与节点 5、节点 3 与节点 6 是两两不连通的，而其余的节点是相互联通的。这和我们设定的防火墙规则是符合的。同时从节点 1 的单节点 ping 测试，我们也可以直观的看出防火墙的效果。
2. 本次实验使用了 Pyretic 控制器，使用控制器实现了 Mac 层上的防火墙相关的策略。也很容易从本次 Project 中看出，使用 SDN 网络实现防火墙策略的管理比传统网络更加容易。
3. 在做本次实验的时候有一个需要注意的就是，我们在写防火墙规则的时候，每一列之间要用逗号空格，因为 Python csv 的 DictReader 只能识别这种格式。我刚开始使用 tab 相隔，结果代码不能执行。