Computer Network* Lab 2

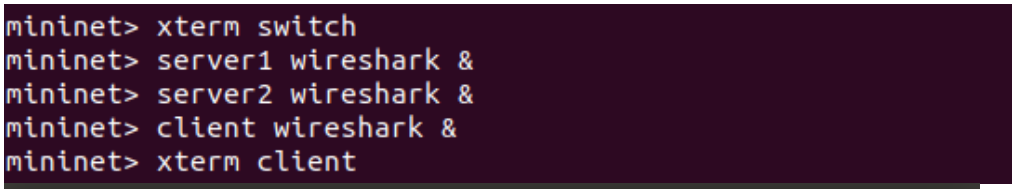
- **徐思源 191220133**
- Department of Computer Science and Technology
- Nanjing University
- 1357307497@qq.com

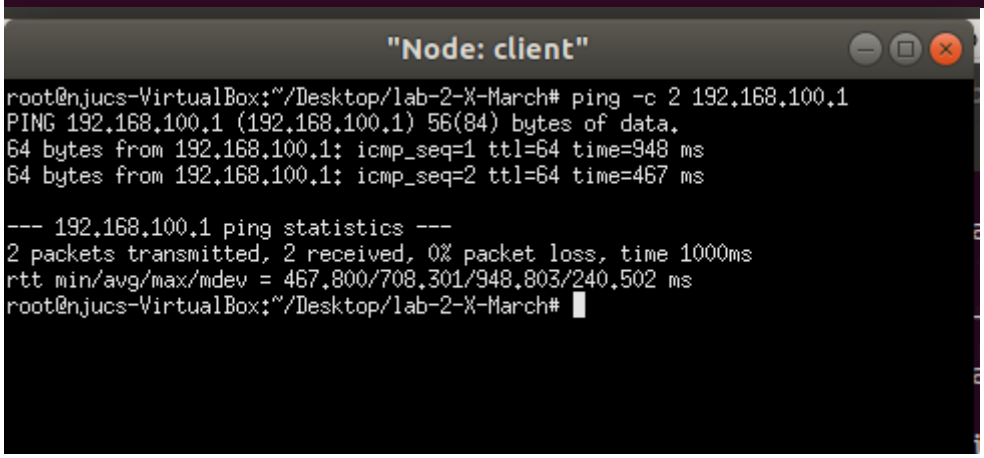
Task2

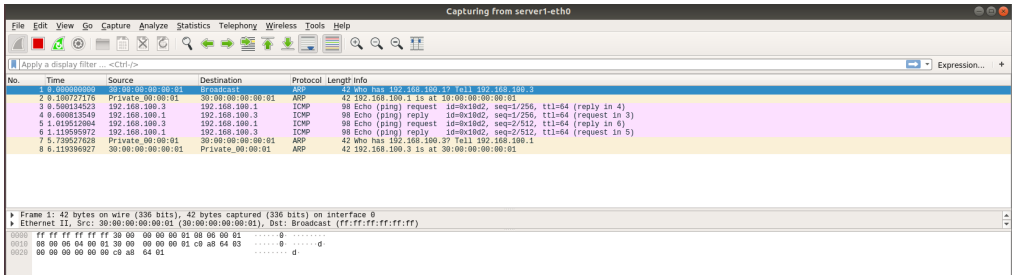
- **首先，根据指导文件中的信息**
  - An Ethernet learning switch is a device that has a set of interfaces ("ports") with links connected to other switches, and to end hosts. When Ethernet frames arrive on any port/interface, the switch sends the frame on an appropriate output port if the switch knows that the host is reachable through that port, or floods the frame out all ports if it does not know where the host is.
  - **由定义可知，我们需要记录ports（interfaces）对应的end host，以便之后有些数据包可以直接输出到对应的端口，也就是说我们需要一个字典来记录每一个输入的数据包的source address和input port数据，发出数据包的时候需要对switch的端口数据进行遍历，如果找到对应的输出端口直接输出，否则选择flooding（向除了输入端口的端口进行输出）。其中需要考虑的特殊情况时发送给switch自己的数据包。**
  - **根据switchyard说明文件中对于packet结构的介绍，提取src，dst等信息**
  - ```
    >>> p[0] # access by index
    <switchyard.lib.packet.ethernet.Ethernet object at 0x104474248>
    >>> p[0].src
    EthAddr('00:00:00:00:00:00')
    >>> p[0].dst
    EthAddr('00:00:00:00:00:00')
    >>> p[0].dst = "ab:cd:ef:00:11:22"
    >>> str(p[0])
    'Ethernet 00:00:00:00:00:00->ab:cd:ef:00:11:22 IP'
    >>> p[0].dst = EthAddr("00:11:22:33:44:55")
    >>> str(p[0])
    'Ethernet 00:00:00:00:00:00->00:11:22:33:44:55 IP'
    >>> p[0].ethertype
    <EtherType.IP: 2048>
    >>> p[0].ethertype = EtherType.ARP
    >>> print (p)
    Ethernet 00:00:00:00:00:00->00:00:00:00:00:00 ARP | IPv4 0.0.0.0->0.0.0.0 ICMP | ICMP EchoRequest 0 0 (0 data bytes)
    >> p[0].ethertype = EtherType.IPv4 # set it back to sensible value
    ```
- **代码解析如下：（部分代码省略）**
  - ```
    while True:
            ......
            log_debug (f"In {net.name} received packet {packet} on
    {fromIface}")
            eth = packet.get_header(Ethernet)
            #add the entry
            port_information[eth.src]=fromIface
            #Non-Ethernet
            if eth is None:
                log_info("Received a non-Ethernet packet?!")
                return
            #intended for me
    ```

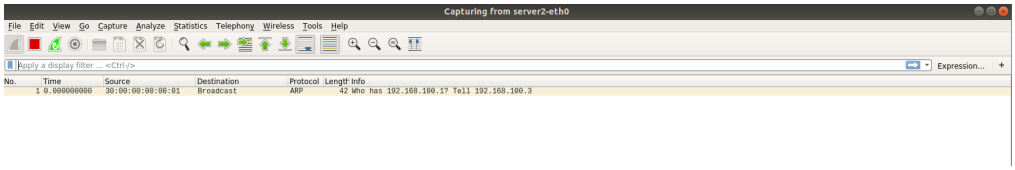
```
        if eth.dst in mymacs:
            log_info("Received a packet intended for me")
        #send directly
        elif eth.dst in port_information.keys:#check port
            net.send_packet(get_outport, packet)
            log_info(f"Send packet{packet} to {eth.dst} by port
{get_outport}")
        #flooding
        else:
            for intf in my_interfaces:
                if fromIface!= intf.name:
                    log_info (f"Flooding packet {packet} to {intf.name}")
                    net.send_packet(intf, packet)

    net.shutdown()
```

- 实验结果及说明如下：
  - 首先打开交换机，打开server1，server2，client的wireshark进行抓包，在client终端输入ping -c 2 192.168.100.1，
    - 

    - 

  - 查看server1和server2的抓包情况：
    - server1：
      

    - server2：
      

  - 对结果进行分析：
    - 一开始，由client向server1发送一个数据包，switch接收到数据包后记录client的port，mac数据，但是table中没有server1的对应数据，所以进行广播，这时候，server1和server2都接收到广播信号（这就是为什么双方都接收到一个ARP广播询问），但是只有

server1做出回复，server1向client发出一个reply信号，此时switch记录下server1对应的mac，port信息，并且由于此时table中已经包含了client的mac，port信号，所以直接向对应端口发送数据包，client接收到server1的reply信号之后进行之后的ping请求等活动，此时由于table中已经包含了server1和client的相关信息，所以不需要再进行广播，ping通后都可以采用direct send的模式进行数据包的传递，所以server2只会接收到一开始的一个广播信号，之后不会再收到数据包

Task3

- **这一步中我们需要实现timeout，所以首先要新建一个时间戳用来记录时间，之后记录到字典中的时候不仅需要进路port信息，还要记录时间信息，这里我们选择使用一个列表来实现，timeout设置为10s，时间记录为int型**
  - **代码实现和说明如下：**
    - ```
      import time

      while True:
              ......

              now=int(time.time())

              ......#same as task2

              if eth.dst in port_information.keys() and now-
      port_information[eth.dst][0]>10:
                  del port_information[eth.dst]

              ......

              elif eth.dst in port_information.keys():#check port
                  net.send_packet(port_information[eth.dst][1], packet)
                  log_info(f"Send packet{packet} to {eth.dst} by port
      {port_information[eth.dst][1]}")

              ......

          net.shutdown()
      ```

- **testcase结果如下：**
      ```

-

```
njucs@njucs-VirtualBox: ~/Desktop/lab-2-X-March
File  Edit  View  Search  Terminal  Help
(syenv) njucs@njucs-VirtualBox:~/Desktop/lab-2-X-March$ swyard -t testcases/myswitch_to_testscenario.srpy myswitch_to.py
10:24:45 2021/03/31     INFO Starting test scenario testcases/myswitch_to_testscenario.srpy
10:24:45 2021/03/31     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255.255 ICMP | ICMP
EchoRequest 0 0 (0 data bytes) to eth0
10:24:45 2021/03/31     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255.255 ICMP | ICMP
EchoRequest 0 0 (0 data bytes) to eth2
10:24:45 2021/03/31     INFO Send packetEthernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoReq
uest 0 0 (0 data bytes) to 30:00:00:00:00:02 by port eth1
10:25:05 2021/03/31     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP Ec
hoRequest 0 0 (0 data bytes) to eth1
10:25:05 2021/03/31     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP Ec
hoRequest 0 0 (0 data bytes) to eth2
10:25:05 2021/03/31     INFO Received a packet intended for me

Results for test scenario switch tests: 9 passed, 0 failed, 0 pending


Passed:
1   An Ethernet frame with a broadcast destination address
    should arrive on eth1
2   The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0 and eth2
3   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive on eth0
4   Ethernet frame destined for 30:00:00:00:00:02 should arrive
    on eth1 after self-learning
5   Timeout for 20s
6   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive on eth0
7   Ethernet frame destined for 30:00:00:00:00:02 should be
    flooded out eth1 and eth2
8   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
9   The hub should not do anything in response to a frame
    arriving with a destination address referring to the hub
    itself.
```

- **mininet运行结果如下：**
  - 首先像task2一样输入指令，其中ping指令输入三次，前两次间隔十秒以内，后两次间隔十秒以上，指令如下：
    -

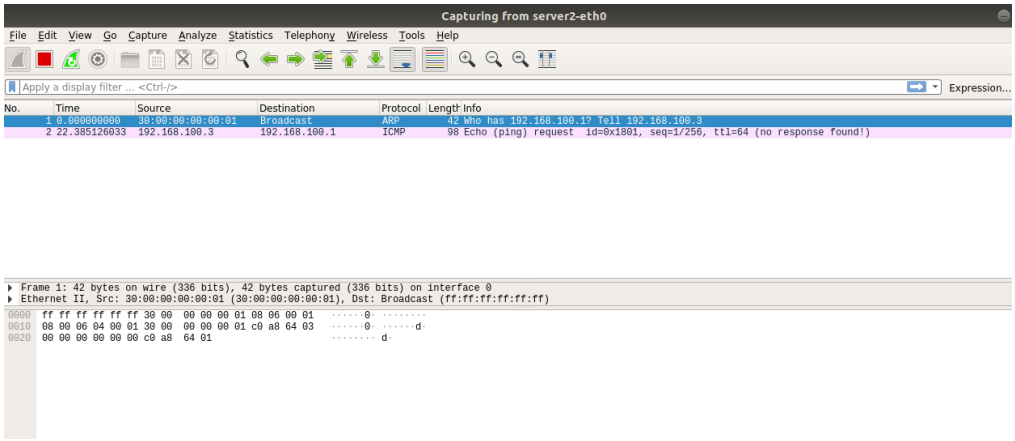
```
mininet> xterm switch
mininet> server1 wireshark &
mininet> server2 wireshark &
mininet> client ping -c 2 server1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=1020 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=535 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 535.275/778.105/1020.935/242.830 ms, pipe 2
mininet> client ping -c 2 server1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=491 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=531 ms

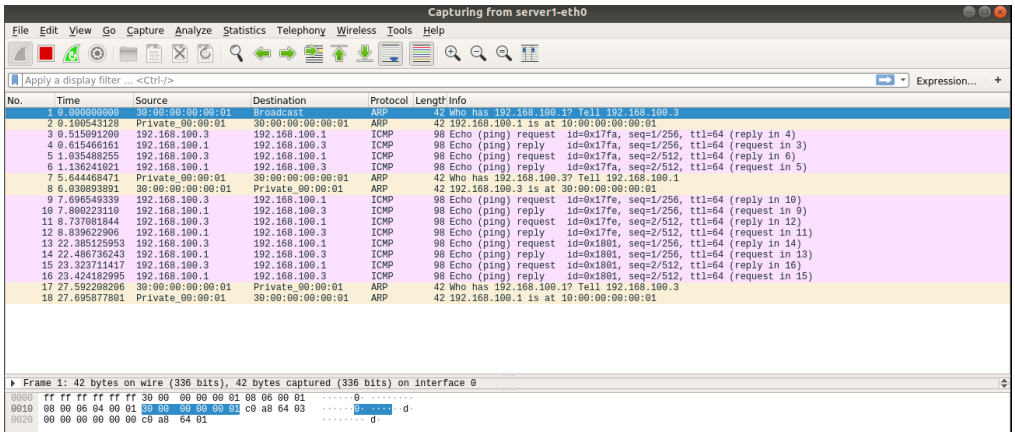
--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 491.514/511.337/531.161/19.836 ms
mininet> client ping -c 2 server1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=565 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=504 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 504.349/534.999/565.650/30.659 ms
mininet>
```

  - 得到抓包数据如下：
    - server2



- server1



  - 由上面的抓包数据可以看到，由于第一次的时候，先进行广播，获得了发送数据包的端口信息，五秒以内信息没有被删除，所以可以继续直接发送，server2只会收到第一次广播的信号，但是十秒之后再进行ping指令的时候，由于timeout的功能，我们已经删除了对应的entry，所以执行的是flooding的发送方式，server2也会接收到第三次client发送出的数据包，所以我们实现的timeout功能正确的实现了

Task4

- **根据lru规则，我们需要记录最近使用的端口信息也就是说我们不能像之前那样直接用字典来存储信息，而要对信息进行排序的处理，所以需要引入ordereddict，之后对于每次需要添加新的端口信息的时候**
  - **按序添加信息**
  - **进行table长度的判断，如果table已满删去最早添加的信息（last=False）**
  - **如果信息已经存在，删去原本的信息，重新添加这样就可以实现按最近使用的顺序对数据进行排列**
- **代码书写和说明如下：**

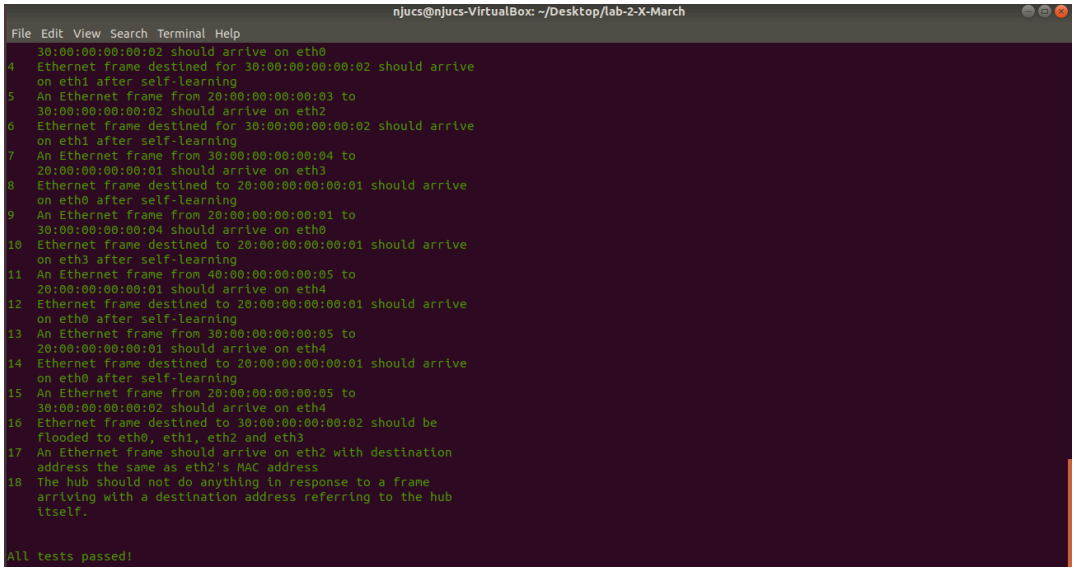
```
from collections import OrderedDict


def main(net: switchyard.llnetbase.LLNetBase):

    ......

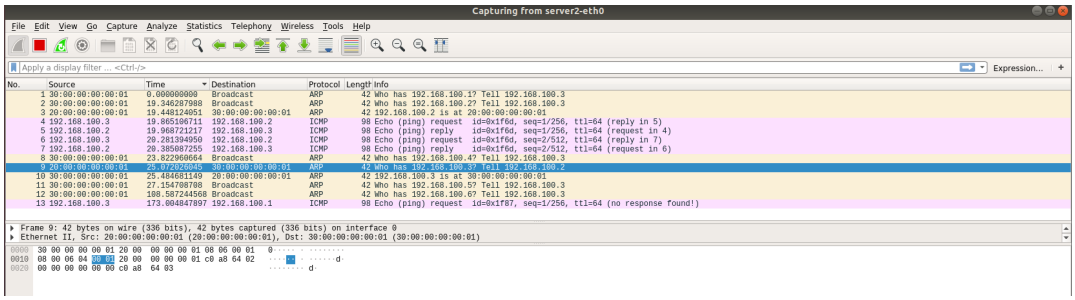
    port_information=OrderedDict();
    while True:

        ......
#add
        if eth.src in port_information.keys():
            port_information.pop(eth.src)
        if len(port_information)==5:
            port_information.popitem(last=False)
        port_information[eth.src]=fromIface
#send
        ......
```

- 运行结果如下
- 

- 在mininet中验证，我们首先需要对拓扑结构进行更改，因为我们的设计中table可以容纳5个entry，所以至少需要六个节点，在start_mininet.py文件中添加server3，server4，server5
  - 添加后新的拓朴结构如下：

```
#new:
# Host and link configuration
#
#
#    server1 server3
#            \ /
#             switch----client
#            / \      \
#    server2 server4  server5
#
```

- 执行client对server1，server2，server3，server4，server5，server1进行依次的ping的请求，然后我们根据server2的抓包数据可以看到：
- 

  - 由于一开始table中逐个加入entry，所以server2除了发给自己的ping请求以外都只接收到了广播信号，但是最后却接收到一个flooding的数据包，因为关于server1的entry已经被删除，所以发生了泛洪。

Task5

- 类似于Task4，只是把替换标准换成频率而已，我们采用两个字典的方式，一个用来存储mac和port信息，一个用来存储mac和freq信息，其中mac地址是key，删除的时候，我们先在freq字典中找到最小项，删除该元素并获得他的key值，再在port字典中删去这一项
- 代码书写和说明如下

```
def main(net: switchyard.llnetbase.LLNetBase):
    ......
    port_information={}
    port_freq={}
    while True:
        ......
        #add the entry
        if eth.src in port_information.keys():
            port_freq[eth.src]+=1
        else:
            #delete
            if len(port_information)==5:
                minfreq=100000
                delkey=eth.src
                for a in port_freq.keys():
                    if port_freq[a]<minfreq:
                        minfreq=port_freq[a]
                        delkey=a
                del port_information[delkey]
                del port_freq[delkey]
            port_freq[eth.src]=0
            port_information[eth.src]=fromIface
        #send
        ......

        #print
        for a in port_information.keys():
            log_info(f"mac:{a} port:{port_information[a]} freq:
{port_freq[a]}")
    net.shutdown()
```

- test结果如下：

```
(syenv) njucs@njucs-VirtualBox:~/Desktop/lab-2-X-March$ swyard -t testcases/myswitch_traffic_testscenario.srpy myswitch_traffic.py
21:05:57 2021/03/31     INFO Starting test scenario testcases/myswitch_traffic_testscenario.srpy
21:05:57 2021/03/31     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
21:05:57 2021/03/31     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth2
21:05:57 2021/03/31     INFO Send packetEthernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to 30:00:00:00:
00:02 by port eth1
21:05:57 2021/03/31     INFO Flooding packet Ethernet 20:00:00:00:00:03->30:00:00:00:00:03 IP | IPv4 172.16.42.3->172.16.42.3 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
21:05:57 2021/03/31     INFO Flooding packet Ethernet 20:00:00:00:00:03->30:00:00:00:00:03 IP | IPv4 172.16.42.3->172.16.42.3 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
21:05:57 2021/03/31     INFO Received a packet intended for me

Results for test scenario switch tests: 8 passed, 0 failed, 0 pending

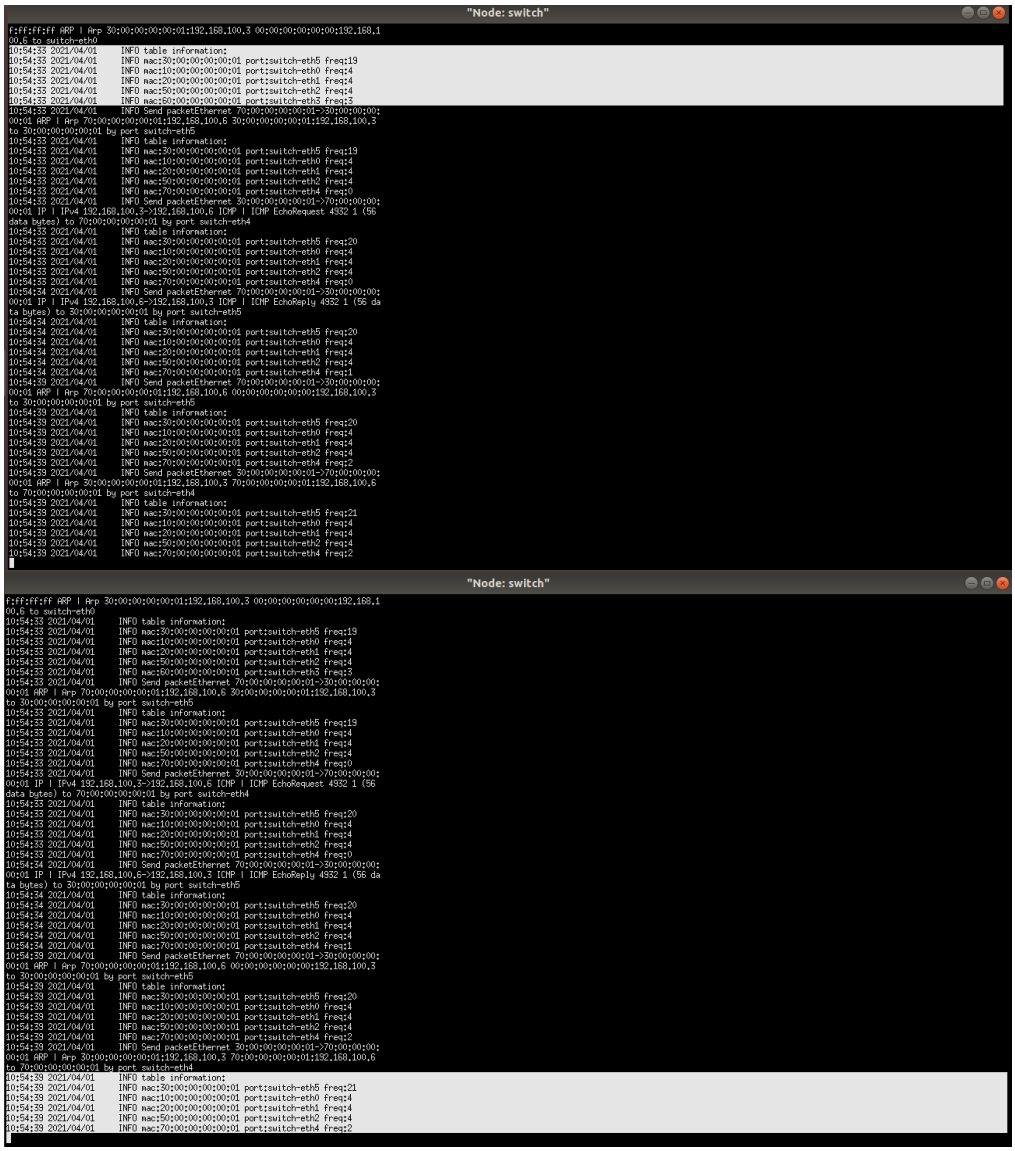

Passed:
1   An Ethernet frame with a broadcast destination address
    should arrive on eth1
2   The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0 and eth2
3   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive on eth0
4   Ethernet frame destined for 30:00:00:00:00:02 should arrive
    on eth1 after self-learning
5   An Ethernet frame from 20:00:00:00:00:03 to
    30:00:00:00:00:03 should arrive on eth2
6   Ethernet frame destined for 30:00:00:00:00:03 should be
    flooded on eth0 and eth1
7   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
8   The switch should not do anything in response to a frame
    arriving with a destination address referring to the switch
    itself.


All tests passed!
```

- mininet运行结果和说明如下：
  - 打开switch，输入以下ping的指令

- ```
  client ping -c 3 server1
  client ping -c 3 server2
  client ping -c 3 server3
  client ping -c 2 server4
  client ping -c 1 server5
  ```

  - 根据我们写的输出信息，在switch的终端中可以看到table的变化

    - 两个高亮处可以看到table中关于server4的entry被删除了，替换成了server5的数据
  ```