

南京大学计算机网络实验报告

任课教师:田臣

实验二 Learning Switch

计算机科学与技术系

181860055 刘国涛

邮箱: 181860055@smail.nju.edu.cn

2020年3月15日-2020年3月20日

实验目的

- 学习Switch的工作原理
- 实现Switch的三种机制Timeout/LRU/LFU

实验内容

Task1: Preparation

按手册将准备工作完成即可，此处略

Task2: Basic Switch

实现方式：

- 使用一个字典存储MAC地址到端口的映射规则
- 在每次接受到一个packet后，将src和input port加入字典中
- 从字典中查找dst对应的output port，若存在，则直接从output port发送packet

代码示意如下：

```
1  mac2port_table = {}
2  while True:
3      #recv a packet
4
5      mac2port_table[packet[0].src] = input_port
6      output_port = mac2port_table.get(packet[0].dst, False)
7
8      #send packet
9      if packet[0].dst in mymacs:
10         #intend for me
11     elif output_port:
12         #send directly
13         net.send_packet(output_port, packet)
14     else:
15         #flood packet
```

Task3: Timeouts

实现方式：

- 延续Task2中用字典存储映射的方式，对value的数据结构修改为 `mac2port_table_item` 的结构，包含了两个成员变量端口名称 `port` 和时间戳 `timestamp`
- 在每次收到一个packet时，以当前时间和输入端口创建一个 `mac2port_table_item` 并存入字典 `mac2port_table` 中
- 从字典中查找是否存在目的地址的映射规则，并与当前时间比较，检查该规则是否TimeOut

代码示意如下：

首先是 `mac2port_table_item` 的定义

```
1 class mac2port_table_item:
2     timestamp = 0
3     port = ""
4     def __init__(self,timestamp,port):
5         self.timestamp = timestamp
6         self.port = port
7     def __repr__(self):
8         return "<mac2port table item>time=%d,port=%s"%
9         (self.timestamp,self.port)
```

然后是主要逻辑：

```
1 mac2port_table = {}
2 while True:
3     #recv packet
4
5     nowTime = int(time.time())
6     #learning
7     mac2port_table[packet[0].src] =
8     mac2port_table_item(nowTime,input_port)
```

```

9         #get output port and judge port timeout
10
11         output_port_item=mac2port_table.get(packet[0].dst,mac2por
12         t_table_item(0,""))
13
14         # 10s timeout
15         if output_port_item.timestamp and nowTime-
16         output_port_item.timestamp>10:
17             del mac2port_table[packet[0].dst]
18             output_port_item = mac2port_table_item(0,"")
19
20         #send packet

```

Task4: Least Recently Used

LRU替换规则适合使用OrderedDict实现

实现方式:

- 使用一个OrderedDict存储MAC地址到端口的映射规则
- 当接受到一个包时, 将src对应的端口从 `mac2port_table_lru` 弹出, 然后重新插入, 使得该规则成为**最近使用**
- 根据包的dst在 `mac2port_table_lru` 中查找对应的端口, 如果存在则将其更新为**最近使用**, 否则判断表是否已满, 然后插入到 `mac2port_table_lru` 中, 并把表中最后一个元素弹出

代码示意如下:

先实现一个 `mac2port_table_lru` 类

```

1 class mac2port_table_lru(OrderedDict):
2     def __init__(self,capacity):
3         self.capacity = capacity
4         self.table = OrderedDict()
5
6     def get(self,key):
7         if key in self.table: #get and update
8             value = self.table.pop(key)
9             self.table[key] = value

```

```

10         else:
11             value = None
12         return value
13
14     def set(self, key, value):
15         if key in self.table:
16             value = self.table.pop(key)
17             self.table[key] = value
18         else:
19             if len(self.table) == self.capacity:
20                 self.table.popitem(last = False)
21                 self.table[key] = value
22             else:
23                 self.table[key] = value
24

```

然后在循环中实现主要逻辑：

```

1  #recv packet
2
3  mac2port_table.set(packet[0].src, input_port)
4
5  output_port = mac2port_table.get(packet[0].dst)
6
7  #send packet
8

```

Task5: Least Traffic Volume

本质就是实现一个LFU替换，利用字典结合列表可实现LFU算法

实现方式：

- 使用一个字典map存储MAC地址(key)到端口(value)的映射，然后另外用一个字典freq_map存储访问频率(key)到字典map的结点(value) `node: {key: MAC, value: port}` 的映射
- 在每次调用get方法时更新结点的访问次数，并更新其在freq_map中的位置
- 在调用set方法时，如果表已满，则删除freq_map中访问次数最少的链表中的一个结点，并插入新的结点

代码示意如下:

首先实现 `LFUNode` 和 `mac2port_table_traffic` 类

```
1  class LFUNode():
2      def __init__(self, key, value):
3          self.freq = 0
4          self.key = key
5          self.value = value
6
7  class mac2port_table_traffic():
8      def __init__(self, capacity):
9          self.capacity = capacity
10         self.map = {}
11         self.freq_map = {}
12
13     def get(self, key):
14         if key in self.map: #get and update
15             node = self.map.get(key)
16             freq = node.freq
17             self.freq_map[freq].remove(node)
18             if len(self.freq_map[freq]) == 0:
19                 del self.freq_map[freq]
20
21             freq += 1
22             print(node)
23             node.freq = freq
24             if freq not in self.freq_map:
25                 self.freq_map[freq] = []
26             self.freq_map[freq].append(node)
27
28         else:
29             return None
30         return node.value
31
32     def set(self, key, value):
33
34         if key in self.map:
35             node = self.map.get(key)
36             node.value = value
37
38         else:
```

```

39         if len(self.map) == self.capacity:
40             min_freq = min(self.freq_map)
41             node = self.freq_map[min_freq].pop()
42             del self.map[node.key]
43             node = LFUNode(key, value)
44             node.freq = 0
45             self.map[key] = node
46             if node.freq not in self.freq_map:
47                 self.freq_map[node.freq] = []
48                 self.freq_map[node.freq].append(node)
49
50     def print(self):
51         print("\nmy table({}) is {}".format(len(self.map)))
52         for key,value in self.freq_map.items():
53             for v in value:
54                 print("MAC:{} port:{} freq:
55 {}").format(v.key,v.value,v.freq))
56         print("")

```

循环中的主要逻辑：

```

1  #recv packet
2
3  mac2port_table.set(packet[0].src,input_port)
4
5  output_port = mac2port_table.get(packet[0].dst)
6
7  #send packet
8

```

实验结果

Task 2

Testing: 无要求

Deploying:

1、打开xterm和wireshark

```
*** Starting 0 switches

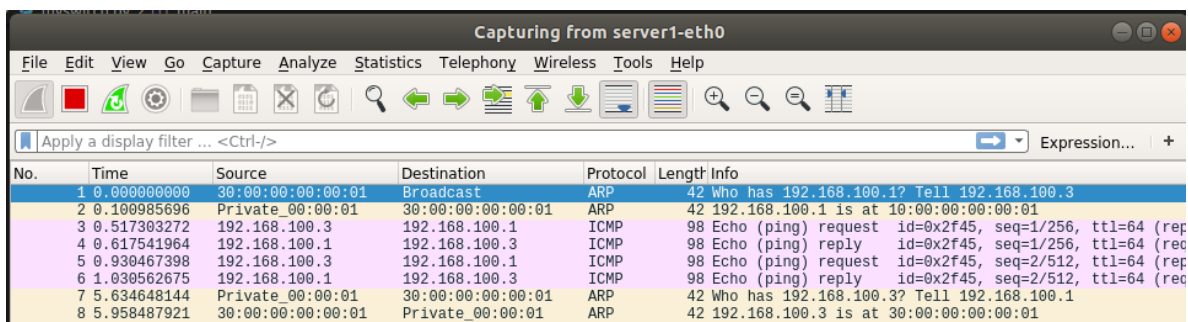
*** Starting CLI:
mininet> xterm switch
mininet> server2 wireshark &
mininet> server1 wireshark &
mininet> client wireshark &
mininet> xterm client
mininet> 
```

2、client ping server1

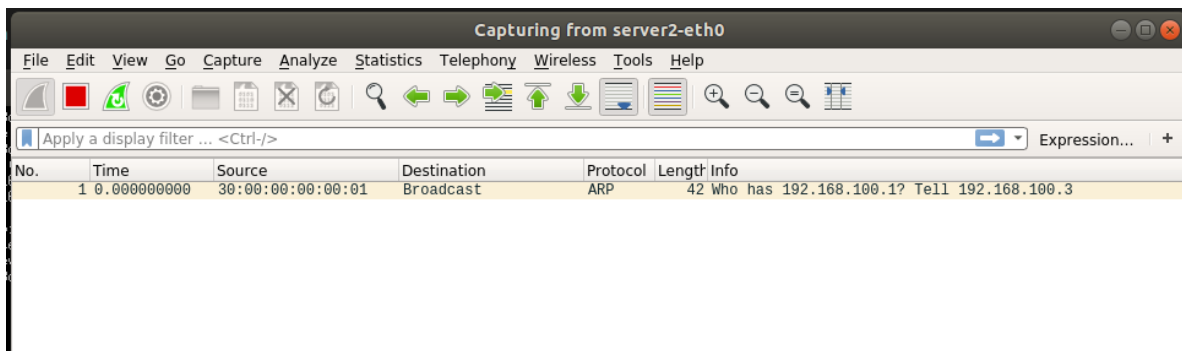
```
"Node: client"
root@njucs-VirtualBox:~/switchyard# ping -c 2 server1
ping: server1: Name or service not known
root@njucs-VirtualBox:~/switchyard# ping -c 2 192.168.100.1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=1028 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=435 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 435.299/731.871/1028.444/296.573 ms, pipe 2
root@njucs-VirtualBox:~/switchyard# 
```

3、然后查看server1和server2的wireshark



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	0.100985696	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.517303272	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x2f45, seq=1/256, ttl=64 (req)
4	0.617541964	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x2f45, seq=1/256, ttl=64 (rep)
5	0.930467398	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x2f45, seq=2/512, ttl=64 (req)
6	1.030562675	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x2f45, seq=2/512, ttl=64 (rep)
7	5.634648144	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	5.958487921	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01



实验现象：server1收到了client发送的包，而server2只收到了一个ARP包

解释：

1. 在client向server1发包前，先发送了ARP广播询问server1的MAC地址，此时switch学习了client与对应端口的规则
2. server1通过广播回复client的询问，此使switch学习了server1与对应端口的规则，并且在switch转发时，直接根据第一步学习的规则向client对应的端口发送该回复，因此server2不会收到这个包
3. 后续的由于ping发送和回复的包也都因为switch中存在client和server1的MAC地址和对应端口，所以server2收不到这些包

Task 3

Testing:

```

*** Done
(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab_2/testcase/switchtests to.srpy lab_2/myswitch_to.py
17:52:22 2020/03/20 INFO Starting test scenario lab_2/testcase/switchtests to.srpy
17:52:22 2020/03/20 INFO In switch tests received 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)
) on eth1
17:52:22 2020/03/20 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
17:52:22 2020/03/20 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
17:52:22 2020/03/20 INFO In switch tests received 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 EchoRequest 0 0 (0 data bytes)
on eth0
17:52:22 2020/03/20 INFO send packet to 30:00:00:00:00:02 by port eth1
17:52:42 2020/03/20 INFO In switch tests received 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 EchoRequest 0 0 (0 data bytes)
on eth0
17:52:42 2020/03/20 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 EchoRequest 0 0 (0 data bytes) to eth1
17:52:42 2020/03/20 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 EchoRequest 0 0 (0 data bytes) to eth2
17:52:42 2020/03/20 INFO In switch tests received packet Ethernet 20:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes)
on eth2
17:52:42 2020/03/20 INFO 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.

All tests passed!

(syenv) njucs@njucs-VirtualBox:~/switchyard$

```

Deploying:

- 1、打开xterm和wireshark

```

*** Starting CLI:
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=911 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=432 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 432.724/672.247/911.771/239.524 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=438 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=497 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 438.254/467.766/497.278/29.512 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=431 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=64 time=489 ms

--- 192.168.100.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 431.509/460.603/489.697/29.094 ms
mininet>

```

2、令client ping server1，五秒后再次执行client ping server1，等待超过十秒后再一次令client ping server1

3、观察server1 server2 的wireshark

Capturing from server1-eth0						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	0.100812010	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.411439808	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3130, seq=1/256, ttl=64 (rep
4	0.512379378	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3130, seq=1/256, ttl=64 (req
5	0.935706041	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3130, seq=2/512, ttl=64 (rep
6	1.036360307	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3130, seq=2/512, ttl=64 (req
7	5.622775705	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	5.935211691	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
9	9.194699229	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3132, seq=1/256, ttl=64 (rep
10	9.294905895	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3132, seq=1/256, ttl=64 (req
11	10.241919171	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3132, seq=2/512, ttl=64 (rep
12	10.342576691	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3132, seq=2/512, ttl=64 (req
13	39.441070016	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3136, seq=1/256, ttl=64 (rep
14	39.541660351	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3136, seq=1/256, ttl=64 (req
15	40.491663938	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3136, seq=2/512, ttl=64 (rep
16	40.591744644	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x3136, seq=2/512, ttl=64 (rep
17	44.683774332	30:00:00:00:00:01	Private_00:00:01	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
18	44.783933105	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
19	44.790315373	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
20	45.198879330	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	39.442130138	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x3136, seq=1/256, ttl=64 (no

实验现象：在mininet中一共构造了三次流量，都是由client向server1 ping，从wireshark显示的时间上可以看到，server2为收到前两次ping产生的包（除ARP外），而收到了第三次产生的包

解释：

1. 在第一次广播时，switch就学习了client和server1的MAC和端口对应规则，因此在前十秒产生的client和server1之间的流量都是直接发送，不会被server2接收
2. 在十秒过后，也就是第三次ping的时候，switch接受到了一个由client发来的包，发现学习到的规则已经TimeOut了，所以进行泛洪，server2也接受到了这个包
3. 之后的包又由于规则进行了更新，所以server2接受不到这些包

Task 4

Testing:

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, eth2, eth3 and eth4
- 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 30:00:00:00:00:02 should arrive on eth2
- 6 Ethernet frame destined for 30:00:00:00:00:02 should arrive on eth1 after self-learning
- 7 An Ethernet frame from 30:00:00:00:00:04 to 20:00:00:00:00:01 should arrive on eth3
- 8 Ethernet frame destined to 20:00:00:00:00:01 should arrive on eth0 after self-learning
- 9 An Ethernet frame from 20:00:00:00:00:01 to 30:00:00:00:00:04 should arrive on eth0
- 10 Ethernet frame destined to 20:00:00:00:00:01 should arrive on eth3 after self-learning
- 11 An Ethernet frame from 40:00:00:00:00:05 to 20:00:00:00:00:01 should arrive on eth4
- 12 Ethernet frame destined to 20:00:00:00:00:01 should arrive on eth0 after self-learning
- 13 An Ethernet frame from 30:00:00:00:00:05 to 20:00:00:00:00:01 should arrive on eth4
- 14 Ethernet frame destined to 20:00:00:00:00:01 should arrive on eth0 after self-learning
- 15 An Ethernet frame from 20:00:00:00:00:05 to 30:00:00:00:00:02 should arrive on eth4
- 16 Ethernet frame destined to 30:00:00:00:00:02 should be flooded to eth0, eth1, eth2 and eth3
- 17 An Ethernet frame should arrive on eth2 with destination address the same as eth2's MAC address
- 18 The hub should not do anything in response to a frame arriving with a destination address referring to the hub itself.

Deploying:

- 1、首先在 `start_mininet.py` 中更改拓扑结构，本次测试需要6个点

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

- 2、打开wireshark，并构建流量：client依次对server1,server2,server3,server4,server5,server1 ping

```

mininet> server1 wireshark &
mininet> server2 wireshark &
mininet> client ping -c 1 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=612 ms

--- 192.168.100.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 612.349/612.349/612.349/0.000 ms
mininet> client ping -c 1 server2
PING 192.168.100.2 (192.168.100.2) 56(84) bytes of data.
64 bytes from 192.168.100.2: icmp_seq=1 ttl=64 time=590 ms

--- 192.168.100.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 590.144/590.144/590.144/0.000 ms
mininet> client ping -c 1 server3
PING 192.168.100.3 (192.168.100.3) 56(84) bytes of data.
64 bytes from 192.168.100.3: icmp_seq=1 ttl=64 time=490 ms

--- 192.168.100.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 490.194/490.194/490.194/0.000 ms
mininet> client ping -c 1 server4
PING 192.168.100.4 (192.168.100.4) 56(84) bytes of data.
64 bytes from 192.168.100.4: icmp_seq=1 ttl=64 time=863 ms

--- 192.168.100.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 863.138/863.138/863.138/0.000 ms
mininet> client ping -c 1 server5
PING 192.168.100.5 (192.168.100.5) 56(84) bytes of data.
64 bytes from 192.168.100.5: icmp_seq=1 ttl=64 time=938 ms

--- 192.168.100.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 938.887/938.887/938.887/0.000 ms
mininet> client ping -c 1 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=576 ms

--- 192.168.100.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 576.497/576.497/576.497/0.000 ms
mininet>

```

3、查看switch日志，在高亮一行（ping server5以后）显示了即将被替换掉的表项

```
{EthAddr('10:00:00:00:00:01'),'switch-eth0'}
```



```
"Node: switch"
21:08:33 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.4 to switch-eth4
21:08:33 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->20:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 20:00:00:00:00:00:192.168.100.2 on switch-eth5
21:08:33 2020/03/23 INFO 20:00:00:00:00:01 output_port:switch-eth1
21:08:33 2020/03/23 INFO send packet to 20:00:00:00:00:01 by port switch-eth1
21:08:33 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 40:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 40:00:00:00:00:01:192.168.100.4 60:00:00:00:00:00:192.168.100.6 on switch-eth3
21:08:33 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:33 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:33 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->40:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.4 ICMP | ICMP EchoRequest 14684 1 (56 data bytes) on switch-eth5
21:08:33 2020/03/23 INFO 40:00:00:00:00:01 output_port:switch-eth3
21:08:33 2020/03/23 INFO send packet to 40:00:00:00:00:01 by port switch-eth3
21:08:34 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 40:00:00:00:00:01->60:00:00:00:00:01 IP | IPv4 192.168.100.4->192.168.100.6 ICMP | ICMP EchoReply 14684 1 (56 data bytes) on switch-eth3
21:08:34 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:34 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:36 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->30:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.3 on switch-eth5
21:08:36 2020/03/23 INFO 30:00:00:00:00:01 output_port:switch-eth2
21:08:36 2020/03/23 INFO send packet to 30:00:00:00:00:01 by port switch-eth2
21:08:36 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 30:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.5 on switch-eth2
21:08:36 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:36 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:36 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.3 60:00:00:00:00:00:192.168.100.6 on switch-eth2
21:08:36 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:36 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:36 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:36 2020/03/23 INFO ff:ff:ff:ff:ff:ff output_port:None
21:08:36 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.5 to switch-eth5
21:08:36 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.5 to switch-eth1
21:08:36 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.5 to switch-eth2
21:08:36 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.5 to switch-eth0
21:08:36 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.5 to switch-eth4
21:08:36 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->30:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 30:00:00:00:00:00:192.168.100.3 on switch-eth5
21:08:36 2020/03/23 INFO 30:00:00:00:00:01 output_port:switch-eth2
21:08:36 2020/03/23 INFO send packet to 30:00:00:00:00:01 by port switch-eth2
21:08:36 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 60:00:00:00:00:00:192.168.100.5 on switch-eth4
21:08:36 2020/03/23 INFO OrderedList{((EthAddr('20:00:00:00:00:01'), 'switch-eth1'), (EthAddr('40:00:00:00:00:01'), 'switch-eth3'), (EthAddr('60:00:00:00:00:01'), 'switch-eth5'), (EthAddr('30:00:00:00:00:01'), 'switch-eth2')))}
21:08:36 2020/03/23 INFO ##### delete : (EthAddr('10:00:00:00:00:01'), 'switch-eth0')#####
21:08:36 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:36 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:37 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->50:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.5 ICMP | ICMP EchoRequest 14689 1 (56 data bytes) on switch-eth5
21:08:37 2020/03/23 INFO send packet to 50:00:00:00:00:01 by port switch-eth4
21:08:37 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 50:00:00:00:00:01->60:00:00:00:00:01 IP | IPv4 192.168.100.5->192.168.100.6 ICMP | ICMP EchoReply 14689 1 (56 data bytes) on switch-eth4
21:08:37 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:39 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 40:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 40:00:00:00:00:01:192.168.100.4 00:00:00:00:00:00:192.168.100.6 on switch-eth3
21:08:39 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:39 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:39 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->40:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 40:00:00:00:00:00:192.168.100.4 on switch-eth5
21:08:39 2020/03/23 INFO 40:00:00:00:00:01 output_port:switch-eth3
21:08:39 2020/03/23 INFO send packet to 40:00:00:00:00:01 by port switch-eth3
21:08:41 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.1 ICMP | ICMP EchoRequest 14632 1 (56 data bytes) on switch-eth5
21:08:41 2020/03/23 INFO 10:00:00:00:00:01 output_port:None
21:08:41 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.1 ICMP | ICMP EchoRequest 14632 1 (56 data bytes) to switch-eth3
21:08:41 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.1 ICMP | ICMP EchoRequest 14632 1 (56 data bytes) to switch-eth1
21:08:41 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.1 ICMP | ICMP EchoRequest 14632 1 (56 data bytes) to switch-eth2
21:08:41 2020/03/23 INFO Flooding packet Ethernet 60:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.100.6->192.168.100.1 ICMP | ICMP EchoRequest 14632 1 (56 data bytes) to switch-eth4
21:08:41 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 10:00:00:00:00:01->60:00:00:00:00:01 IP | IPv4 192.168.100.1->192.168.100.6 ICMP | ICMP EchoReply 14632 1 (56 data bytes) on switch-eth0
21:08:41 2020/03/23 INFO OrderedList{((EthAddr('20:00:00:00:00:01'), 'switch-eth1'), (EthAddr('40:00:00:00:00:01'), 'switch-eth3'), (EthAddr('60:00:00:00:00:01'), 'switch-eth5'), (EthAddr('30:00:00:00:00:01'), 'switch-eth2')))}
21:08:41 2020/03/23 INFO ##### delete : (EthAddr('20:00:00:00:00:01'), 'switch-eth1')#####
21:08:41 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:41 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:42 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 50:00:00:00:00:01->60:00:00:00:00:01 ARP | Arp 50:00:00:00:00:01:192.168.100.5 00:00:00:00:00:00:192.168.100.6 on switch-eth4
21:08:42 2020/03/23 INFO 60:00:00:00:00:01 output_port:switch-eth5
21:08:42 2020/03/23 INFO send packet to 60:00:00:00:00:01 by port switch-eth5
21:08:42 2020/03/23 INFO In njucs-VirtualBox received packet Ethernet 60:00:00:00:00:01->50:00:00:00:00:01 ARP | Arp 60:00:00:00:00:01:192.168.100.6 50:00:00:00:00:00:192.168.100.5 on switch-eth5
21:08:42 2020/03/23 INFO 50:00:00:00:00:01 output_port:switch-eth4
21:08:42 2020/03/23 INFO send packet to 50:00:00:00:00:01 by port switch-eth4
```

4、查看server1 server2的wireshark

Capturing from server1-eth0					
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help					
Apply a display filter ... <Ctrl-/> Expression...					
No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	192.168.100.6	192.168.100.1	ICMP	98 Echo (ping) request id=0x3953, seq=1/256, ttl=64 (reply in 3)
2	0.100249988	192.168.100.1	192.168.100.6	ICMP	98 Echo (ping) reply id=0x3953, seq=1/256, ttl=64 (request in 2)
3	2.93765590	192.168.100.6	192.168.100.2	ICMP	98 Echo (ping) request id=0x3957, seq=1/256, ttl=64 (no response found!)
4	5.125688376	60:00:00:00:00:01	Private_00:00:01	ARP	42 Who has 192.168.100.1? Tell 192.168.100.6
5	5.179675419	Private_00:00:01	60:00:00:00:00:01	ARP	42 Who has 192.168.100.6? Tell 192.168.100.1
6	5.226480138	Private_00:00:01	60:00:00:00:00:01	ARP	42 192.168.100.1 is at 10:00:00:00:00:01
7	5.545888612	60:00:00:00:00:01	Private_00:00:01	ARP	42 192.168.100.6 is at 60:00:00:00:00:01
8	6.298181819	192.168.100.6	192.168.100.3	ICMP	98 Echo (ping) request id=0x395a, seq=1/256, ttl=64 (no response found!)
9	8.597937000	60:00:00:00:00:01	Broadcast	ARP	42 Who has 192.168.100.4? Tell 192.168.100.6
10	11.840538445	60:00:00:00:00:01	Broadcast	ARP	42 Who has 192.168.100.5? Tell 192.168.100.6
11	16.600390559	192.168.100.6	192.168.100.1	ICMP	98 Echo (ping) request id=0x3964, seq=1/256, ttl=64 (reply in 3)
12	16.701689772	192.168.100.1	192.168.100.6	ICMP	98 Echo (ping) reply id=0x3964, seq=1/256, ttl=64 (request in 2)

Capturing from server2-eth0					
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help					
Apply a display filter ... <Ctrl-/> Expression...					
No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	192.168.100.6	192.168.100.1	ICMP	98 Echo (ping) request id=0x3953, seq=1/256, ttl=64 (no response found!)
2	2.931671075	192.168.100.6	192.168.100.2	ICMP	98 Echo (ping) request id=0x3957, seq=1/256, ttl=64 (reply in 3)
3	3.032451236	192.168.100.2	192.168.100.6	ICMP	98 Echo (ping) reply id=0x3957, seq=1/256, ttl=64 (request in 2)
4	6.298060812	192.168.100.6	192.168.100.3	ICMP	98 Echo (ping) request id=0x395a, seq=1/256, ttl=64 (no response found!)
5	8.177523604	60:00:00:00:00:01	20:00:00:00:00:01	ARP	42 Who has 192.168.100.2? Tell 192.168.100.6
6	8.256447764	20:00:00:00:00:01	60:00:00:00:00:01	ARP	42 Who has 192.168.100.6? Tell 192.168.100.2
7	8.277779202	20:00:00:00:00:01	60:00:00:00:00:01	ARP	42 192.168.100.2 is at 20:00:00:00:00:01
8	8.597966438	60:00:00:00:00:01	Broadcast	ARP	42 Who has 192.168.100.4? Tell 192.168.100.6
9	8.598913104	60:00:00:00:00:01	20:00:00:00:00:01	ARP	42 192.168.100.6 is at 60:00:00:00:00:01
10	11.841502503	60:00:00:00:00:01	Broadcast	ARP	42 Who has 192.168.100.5? Tell 192.168.100.6
11	16.598397436	192.168.100.6	192.168.100.1	ICMP	98 Echo (ping) request id=0x3964, seq=1/256, ttl=64 (no response found!)

实验现象：在最开始泛洪时server2接收到了client发给server1的包，而结束时，server2再次接收到了client发给server1的包

解释：

1. 在最开始时，switch的表里为空，学习client的规则后，对接收到的包进行泛洪

2. 之后client依次向server2,3,4发包，switch共学习到了server1,2,3,4,client的规则，此时表已满
3. 在client向server5 ping的时候，根据LRU，server1的规则被替换
4. 最后client向server1 ping的时候，表里无对应的规则，因此switch将包进行泛洪，server2也就再次收到了这个包

Task 5

Testing:

```
(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab 2/testcase/switchtests traffic.srpy lab 2/myswitch.traffic.py
17:38:30 2020/03/24 INFO Starting test scenario lab 2/testcase/switchtests traffic.srpy
17:38:30 2020/03/24 INFO In switch tests received packet Ethernet 30:00:00:00:02->ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes)
) on eth1
17:38:30 2020/03/24 INFO ff:ff:ff:ff:ff:ff output port:None
17:38:30 2020/03/24 INFO Flooding packet Ethernet 30:00:00:00:02->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
17:38:30 2020/03/24 INFO Flooding packet Ethernet 30:00:00:00:02->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
17:38:30 2020/03/24 INFO In switch tests received packet Ethernet 20:00:00:00:01->30:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes)
on eth0
17:38:30 2020/03/24 INFO 30:00:00:00:02 output port:eth1
17:38:30 2020/03/24 INFO send packet to 30:00:00:00:02 by port eth1
17:38:30 2020/03/24 INFO In switch tests received packet Ethernet 20:00:00:00:03->30:00:00:00:03 IP | IPv4 172.16.42.3->172.16.42.3 ICMP | ICMP EchoRequest 0 0 (0 data bytes) on
eth2
17:38:30 2020/03/24 INFO 30:00:00:00:03 output port:None
17:38:30 2020/03/24 INFO Flooding packet Ethernet 20:00:00:00:03->30:00:00:00:03 IP | IPv4 172.16.42.3->172.16.42.3 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
17:38:30 2020/03/24 INFO Flooding packet Ethernet 20:00:00:00:03->30:00:00:00:03 IP | IPv4 172.16.42.3->172.16.42.3 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
17:38:30 2020/03/24 INFO In switch tests received packet Ethernet 20:00:00:00:01->10:00:00:00:03 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes)
on eth2
17:38:30 2020/03/24 INFO 10:00:00:00:03 output port:None
17:38:30 2020/03/24 INFO 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 20:00:00:00:01 to
  30:00:00:00:02 should arrive on eth0
4 Ethernet frame destined for 30:00:00:00:02 should arrive
  on eth1 after self-learning
5 An Ethernet frame from 20:00:00:00:03 to
  30:00:00:00:03 should arrive on eth2
6 Ethernet frame destined for 30:00:00:00:03 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!
```

Deploying:

1、首先构造一些流量：

```
1 client ping -c 10 server1
2 client ping -c 5 server2
3 client ping -c 2 server3
4 client ping -c 1 server4
```

此时switch的表如下图中第一个箭头，此时表已满，表中server4即将被替换

```

my table(5) is :
MAC:10:00:00:00:00:01 port:switch-eth0 freq:11
MAC:20:00:00:00:00:01 port:switch-eth1 freq:6
MAC:30:00:00:00:00:01 port:switch-eth2 freq:3
MAC:60:00:00:00:00:01 port:switch-eth5 freq:26
MAC:40:00:00:00:00:01 port:switch-eth3 freq:2

21:46:43 2020/03/24      INFO ff:ff:ff:ff:ff:ff output_port:None
21:46:43 2020/03/24      INFO Flooding packet Ethernet 60:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.1
00.5 to switch-eth0
21:46:43 2020/03/24      INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.1
00.5 to switch-eth1
21:46:43 2020/03/24      INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.1
00.5 to switch-eth2
21:46:43 2020/03/24      INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.1
00.5 to switch-eth3
21:46:43 2020/03/24      INFO Flooding packet Ethernet 60:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 60:00:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.1
00.5 to switch-eth4
21:46:43 2020/03/24      INFO In njucs-VirtualBox received packet Ethernet 50:00:
00:00:00:01->60:00:00:00:00:01 ARP | Arp 50:00:00:00:00:01:192.168.100.5 60:00:0
0:00:00:01:192.168.100.6 on switch-eth4
60:00:00:00:00:01 switch-eth5 26

my table(5) is :
MAC:10:00:00:00:00:01 port:switch-eth0 freq:11
MAC:20:00:00:00:00:01 port:switch-eth1 freq:6
MAC:30:00:00:00:00:01 port:switch-eth2 freq:3
MAC:50:00:00:00:00:01 port:switch-eth4 freq:0
MAC:60:00:00:00:00:01 port:switch-eth5 freq:27

```

2、再构造一个流量

```
1 client ping -c 1 server5
```

此时如上图所示，由于表中无server5，所以client发送的ARP包被泛洪。接着server5的应答被直接发送到client，而server5被加入到switch的表中，如上图的第二个箭头，server4被替换

总结与感想

本次实验学习了switch的原理，进一步了解了链路层中的优化