

南京大学本科生实验报告

181860097 计算机科学与技术系 王明骁

Email: 1164151483@qq.com

开始日期: 2020年3月20日 结束日期: 2020年3月23日

lab2

实验名称

Lab 2: Learning Switch

实验目的

- 学习交换机的基本原理
- 掌握交换机对于在有限的内存中删除表项的几种方法（超时删除，lru算法，最小流量）

task1

1. 实验内容

task1为准备阶段，只需要按照实验手册上的说明，将相关文件copy到lab_2文件夹并做好命名即可

task2

1. 实验内容

task2要求实现一个具有学习功能的交换机，即每次收到数据包时记录下对应的端口号和mac地址，在每次发送数据包之前首先去表中寻找是否学习过该mac地址在哪个端口，如果学习过则直接向对应的端口发送，否则就进行洪泛

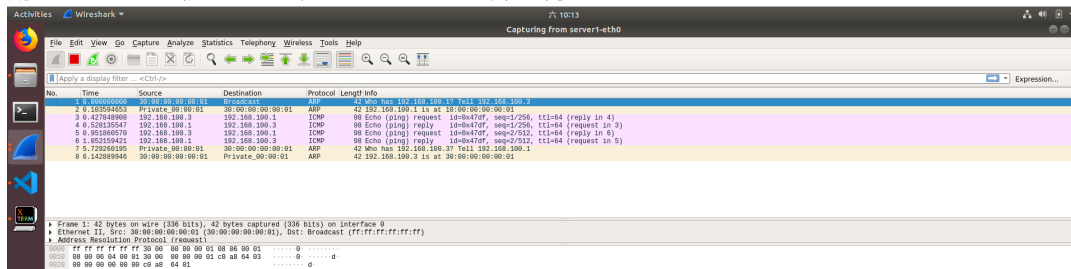
```
flag = 0
for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        break

log_debug ("In {} received packet {} on {}".format(net.name, packet, input_port))
if packet[0].dst in mymacs:
    log_debug ("Packet intended for me")
else:
    if flag == 1:
        net.send_packet(destintf, packet)
    else:
        for intf in my_interfaces:
            if input_port != intf.name:
                log_debug ("Flooding packet {} to {}".format(packet, intf.name))
                net.send_packet(intf.name, packet)
        temp = learning(packet[0].src, input_port)
        table.append(temp)
```

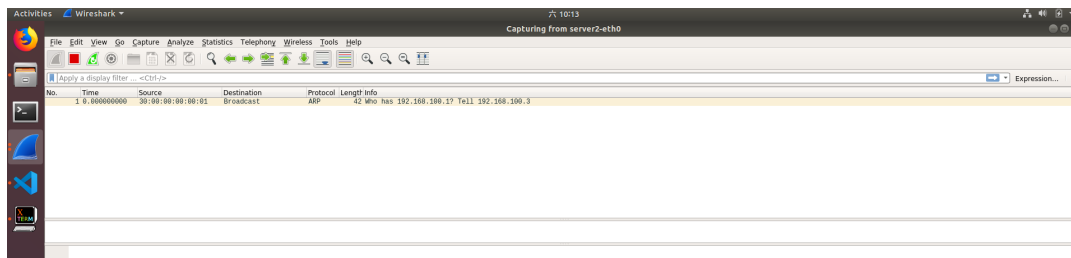
具体实现为，首先在table中是否有对应于目标mac地址的表项，如果有则记录下destintf，并直接向destintf发送，否则进行洪泛

2. 实验结果

用wireshark监视server1和server2的流量并且构造从client到server1的流量



可以看到server1中有两对发送和回复的包



而server2中只有arp包，因此可以知道改交换机学习功能是成功的，在最开始广播arp的时候就知道了server1对应的端口，因此接下来两个包都不再采取广播形式，而是直接向server1所对应的端口发送数据包

3. 核心代码

定义learning类用来存放每一个表项，其中每个表项包含其mac地址和对应的端口

```
class learning(object):
    def __init__(self,mac,interface):
        self.mac = mac
        self.interface = interface
```

每次收到一个数据包时，在表中寻找是否记录过他的mac地址对应的端口，如果记录过，则记下对应的端口并做个标记（flag），在之后的发送中如果之前记录过则直接向对应的端口发送数据，否则进行洪泛，同时，在每收到一个数据包时就对表项进行增加

```
flag = 0
for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        break

log_debug ("In {} received packet {} on {}".format(net.name, packet, input_port))
if packet[0].dst in mymacs:
    log_debug ("Packet intended for me")
else:
    if flag == 1:
        net.send_packet(destintf, packet)
    else:
        for intf in my_interfaces:
            if input_port != intf.name:
                log_debug ("Flooding packet {} to {}".format(packet, intf.name))
                net.send_packet(intf.name, packet)
        temp = learning(packet[0].src,input_port)
        table.append(temp)
```

task3

1. 实验内容

task3要求根据超时原则进行表项的删除，设置为超过10秒的表项进行删除，同时不对表项的数量上限做限制

```

        nowtime = time.time()
        for index in table:
            if nowtime - index.time > 10:
                table.remove(index)
            if index.mac == packet[0].src:
                table.remove(index)

        You, an hour ago • finish task3

        temp = learning(packet[0].src,input_port,nowtime)
        table.append(temp)

        flag = 0
        for index in table:
            if index.mac == packet[0].dst:
                flag = 1
                destintf = index.interface
                break
        if flag == 1:
            net.send_packet(destintf, packet)
        else:
            for intf in my_interfaces:
                if input_port != intf.name:
                    log_debug ("Flooding packet {} to {}".format(packet, intf.name))
                    net.send_packet(intf.name, packet)

```

具体实现为：通过调用time.time()得到当前的时间戳，然后对表中内容进行更新，具体为，遍历表中内容，如果之前添加的时间戳和现在的时间戳之差大于10秒则删除，同时如果表中的mac地址与收到的包的源地址相同则删除，这是为了对表中已有的表项进行更新，删除后再添加新的表项进去与直接对表项中的数据做修改是等价的，为了与没有找到与收到的包的源地址相同的情况下的行为统一，这里采用的是删除再添加的方法。之后的查询逻辑与basic switch相同，在此不再赘述

2. 实验结果

运行提供的测试样例，可以看到正确通过测试

```

(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab 2/switchtests to.srpy lab_2/myswitch_to.py
16:04:19 2020/03/21      INFO Starting test scenario lab_2/switchtests_to.srpy

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.

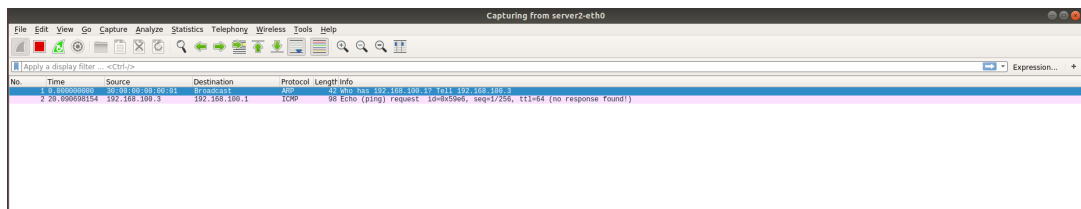
All tests passed!

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

```

接下来是自己对该switch进行测试，采取用wireshark监视每个server的办法，首先是构造了两个从client到server1的流量，然后等待十多秒后，再次构造两个从client到server1的流量，在wireshark中抓到的包如下：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
2	0.164268046	Private 00:00:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 30:00:00:00:00:01
3	0.516141430	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
4	0.617292753	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
5	1.102020710	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
6	1.130464382	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
7	5.075101442	Private 00:00:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	6.207484820	30:00:00:00:00:01	Private 00:00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
9	20.102070705	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
10	20.192270601	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
11	21.143210991	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3
12	21.24443269	192.168.100.1	192.168.100.3	ICMP	60	Echo (ping) request 192.168.100.1 to 192.168.100.3



可以看到，在最开始构造的流量中，server1能正确接收到包，而server2只有ARP包，因此可以知道该交换器的学习功能正确实现，而在time为20构造的流量中，可以看到server2也收到了广播的一个包，因此可以知道交换器确实在表中删除了超时的条目（否则他仍会只向server1对应的端口发送）

3. 核心代码

表项的内容与之前相比增加了时间戳：

```
class learning(object):
    def __init__(self,mac,interface,time):
        self.mac = mac
        self.interface = interface
        self.time = time
```

时间记录用来记录每个表项被添加的时间

删除超时表项和查找以及发送的核心代码:

```

nowtime = time.time()
for index in table:
    if nowtime - index.time > 10:
        table.remove(index)
    if index.mac == packet[0].src:
        table.remove(index)

    You, an hour ago • finish task3

temp = learning(packet[0].src,input_port,nowtime)
table.append(temp)

flag = 0
for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        break
if flag == 1:
    net.send_packet(destintf, packet)
else:
    for intf in my_interfaces:
        if input_port != intf.name:
            log_debug ("Flooding packet {} to {}".format(packet, intf.name))
            net.send_packet(intf.name, packet)

```

其原理在实验内容部分已经阐述，在此不再赘述

task4

1. 实验内容

task4要求利用lru算法对表进行删除，即统计每个表项的使用频率，当表满（超过5个）时，选择使用频率最低的一项删除，具体实现为：

```

for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        index.age = 0
    else:
        index.age = index.age + 1
log_debug ("In {} received packet {} on {}".format(net.name, packet, input_port))
if packet[0].dst in mymacs:
    log_debug ("Packet intended for me")
else:
    if flag == 1:
        net.send_packet(destintf, packet)
    else:
        for intf in my_interfaces:
            if input_port != intf.name:
                log_debug ("Flooding packet {} to {}".format(packet, intf.name))
                net.send_packet(intf.name, packet)
temp = learning(packet[0].src,input_port,0)
    You, a few seconds ago • Uncommitted changes

for index in table:
    if index.mac == packet[0].src:
        table.remove(index)
if len(table) >= 5:
    table.remove(max(table,key = lambda x: x.age))

table.append(temp)

```

其学习和在表中查找的功能与基础交换机实现一样，额外增加的是，在每次使用到一个表项时，用age记录下其未被访问的次数，因此每收到一个包，未使用过的表项则计数+1，同时如果该表项被使用，则将age清零，这样删除时删除age最大的一个即是最近最少使用的一项，同时对于已有的表项，在新的内容进来时将其删除再添加以达到更新的效果（可以同时更新端口和使用频率）当表中的数量大于等于5，并且还要向表中添加数据时，删除age最大的一项，这里使用lambda函数和max来查找表中age最大的一项

2. 实验结果

首先运行提供的测试样例，可以看到通过测试

Switchyard 副本 [正在运行] - Oracle VM VirtualBox

管理 控制 视图 热键 设备 帮助

Activities Visual Studio Code 六 11:01

myswitch_lru.py - switchyard - Visual Studio Code

EXPLORER

OPEN EDITORS

SWITCHYARD

- clientapp_udpstack.py
- hubtests.py
- myhub.py
- README.rst
- readpkt.py
- sendpkt.py
- server_udpstack.py
- sniff.py
- start_mininet.py
- sydump.py
- udpstack_tests.py
- udpstack.py
- lab_1
- lab_2
 - __pycache__
 - myswitch_lru.py M
 - myswitch_to.py
 - myswitch_traffic.py
 - myswitch.py
 - mytests_lru.py
 - mytests_to.py
 - mytests_traffic.py
 - mytests.py
 - start_mininet.py
 - switchtests_lru.py U
 - switchtests_traffic.py U
 - switchtests.py U
- switchyard
 - lib
 - sim
 - __init__.py
 - hostfirewall.py
 - importcode.py
 - l1netbase.py
 - l1netreal.py
 - l1nettest.py
 - outputfmt.py

OUTLINE

OUTPUT

TERMINAL

DEBUG CONSOLE

PROBLEMS

```

*** Stopping 3 links
...
*** Stopping 0 switches

*** Stopping 4 hosts
client server1 server2 switch
*** Done
(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab_2/switchtests_lru.srpy lab_2/myswitch_lru.py
11:00:10 2020/03/21 INFO Starting test scenario lab_2/switchtests_lru.srpy

Results for test scenario switch tests: 18 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, 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.

All tests passed!

```

接下来是自己在mininet中测试，由于该网络拓扑只有2个server，因此表的5个数据上限设定让表永远都不会满，故测试时将上限改为2，但提交时仍提交5的版本，测试方法同样是用wireshark监视每个server，首先构造从client到server1的流量，此时交换机表中存的是client和server1，然后构造client到server2，此时由于lru算法，交换机表中存的是client和server2，然后再构造从client到server1的流量，由于此时server1已被删除，因此会进行广播，用wireshark抓到的包如下：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	0.52271914	192.168.100.3	30:00:00:00:00:01	ICMP	98	Echo (ping) request 10-0x5fea, seq=1/256, ttl=64 (reply in 4)
4	0.62368385	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply 10-0x5fea, seq=1/256, ttl=64 (request in 3)
5	2.91228944	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
6	5.54545708	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request 10-0x5ff1, seq=1/256, ttl=64 (reply in 8)
7	6.64968987	Private 98:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
8	6.76334379	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply 10-0x5ff1, seq=1/256, ttl=64 (request in 6)
9	6.622732369	30:00:00:00:00:01	Private_00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	2.91228978	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
3	3.91396473	30:00:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.2 is at 30:00:00:00:00:01
4	3.44078571	192.168.100.3	192.168.100.2	ICMP	98	Echo (ping) request 10-0x5fed, seq=1/256, ttl=64 (reply in 5)
5	5.54545919	192.168.100.2	192.168.100.3	ICMP	98	Echo (ping) reply 10-0x5fed, seq=1/256, ttl=64 (request in 4)
6	6.5591457491	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request 10-0x5ff1, seq=1/256, ttl=64 (no response found)
7	6.713108423	30:00:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.2
8	9.68255759	30:00:00:00:00:01	20:00:00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01

可以看到最开始构造从client到server1的流量对应于0秒的时间戳，然后构造的从client到server2的流量对应于2秒的时间戳，最后构造的从client到server1的流量对应于5秒的时间戳，可以看到server2也收到了这个包，因此可以知道交换机进行了洪泛，因此可以知道表中确实server1对应的表项被删除了，算法正确

3. 核心代码

对于表中每一项，增加了age变量，用来记录未被使用的次数

```
class learning(object):
    def __init__(self,mac,interface,age):
        self.mac = mac
        self.interface = interface
        self.age = age
```

lru算法的具体实现：已在实验内容中阐述

```

for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        index.age = 0
    else:
        index.age = index.age + 1
log_debug ("In {} received packet {} on {}".format(net.name, packet, input_port))
if packet[0].dst in mymacs:
    log_debug ("Packet intended for me")
else:
    if flag == 1:
        net.send_packet(destintf, packet)
    else:
        for intf in my_interfaces:
            if input_port != intf.name:
                log_debug ("Flooding packet {} to {}".format(packet, intf.name))
                net.send_packet(intf.name, packet)
temp = learning(packet[0].src,input_port,0)
    You, a few seconds ago * Uncommitted changes
for index in table:
    if index.mac == packet[0].src:
        table.remove(index)
if len(table) >= 5:
    table.remove(max(table,key = lambda x: x.age))

table.append(temp)

```

task5

1. 实验内容：

task5要求当表满时删除最少流量的表项

```

# search
flag = 0
for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        index.age = index.age + 1

# add
temp = learning(packet[0].src,input_port,1) #send one time
for index in table:    You, 2 days ago * fix lru bug and
    if index.mac == packet[0].src:
        table.remove(index)
if len(table) >= 5:
    table.remove(min(table,key = lambda x: x.age))

table.append(temp)

```

因此采取age记录每个表项的流量，每次遇到dest的mac与表项中的mac相同时，流量计数+1，当表满时，删除age最小的一个，与lru算法类似，采用min函数和lambda函数找到表中age最小的一项并删除，而更新时仍然采用将旧的表项删除并添加新的表项的做法

2. 实验结果

运行测试样例，可以看到正确通过


```
(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab_2/switchtests_traffic.srpy lab_2/myswitch_traffic.py
09:53:05 2020/03/23 INFO Starting test scenario lab_2/switchtests_traffic.srpy

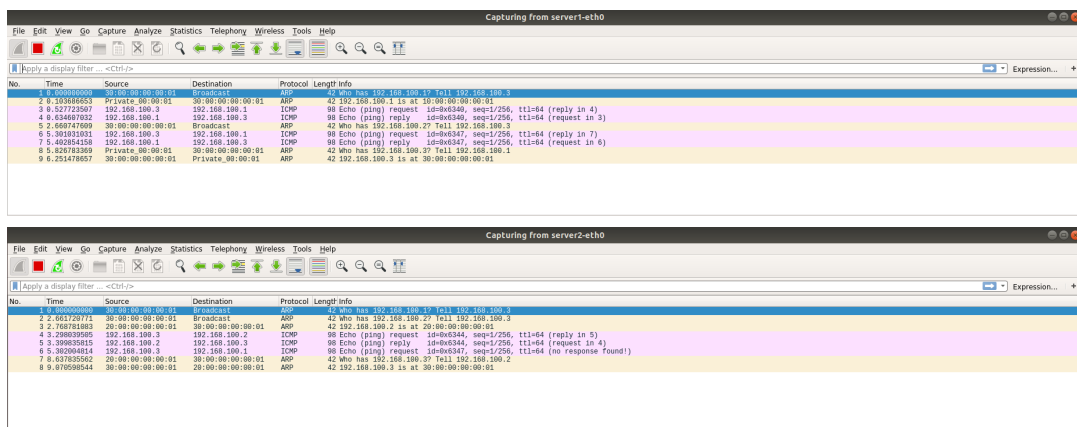
Results for test scenario switch tests: 8 passed, 0 failed, 0 pending

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!

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

在mininet中测试：同样采取用wireshark监视流量的办法，首先构造从client到server1的流量，然后构造从client到server2的流量，此时由于表中server1的流量最小，因此server1被删除，然后构造从client到server1的流量，理论上server2应该也要能收到洪泛的流量，用wireshark抓到的包如下：



可以看到最开始构造的两个包都正确被收到，同时，最后一个从client到server1的包也被server2收到，因此交换机进行了洪泛，故表中server1确实被删除，该算法正确

3. 核心代码

表中每一项增加了age，该age记录了每一项的流量，删除时选择age最小的一项进行删除

```
class learning(object):
    def __init__(self,mac,interface,age):
        self.mac = mac
        self.interface = interface
        self.age = age
```

具体实现最小流量删除的代码：已在实验内容部分阐述逻辑


```

# search
flag = 0
for index in table:
    if index.mac == packet[0].dst:
        flag = 1
        destintf = index.interface
        index.age = index.age + 1

# add
temp = learning(packet[0].src,input_port,1) #send one time
for index in table:
    if index.mac == packet[0].src:
        table.remove(index)
if len(table) >= 5:
    table.remove(min(table,key = lambda x: x.age))

table.append(temp)

```

总结与感想

lab2整体而言不算难，且与课程联系非常紧密，可以让我们马上就实践到课上学到的知识，不得不说设计的非常好！而在本次实验中更是体会到交换机与hub的区别，实践了交换机的代码，对网络的理解加深了！