南京大学本科生实验报告

- 课程名称：**计算机网络**　　任课教师：田臣/李文中
  助教：

学院	计算机科学与技术系	专业（方向）	计算机科学与技术
学号	201220102	姓名	武雅琛
Email	201220102@smail.nju.edu.cn	开始/完成日期	2022.3.21

1. 实验名称

learning switch

2. 实验目的

- **进一步熟悉Switchyard框架。**

  通过自行学习Switchyard相关**API的实现和接口**进一步熟悉lab提供的Switchyard框架。

- **了解和学习以太网交换机的学习和工作原理。**

  根据实验手册学习以太网交换机的逻辑：**端口映射和数据帧转发。**对计算机网络的二层结构由更深入的理解。

- **了解并且在逻辑上实现三种更新交换机转发表的方式。**

  交换机为了适应拓扑网络的不断变化和提高工作效率会对**转发表的表项进行适当的删除和更新。**这里要了解和实现三种方式：表项超时，LRU（最近最少使用）以及流量检测的方式。

3. 实验内容

task 1:Preparation

task 2:Basic Switch

- 这一步要求将一个仅能在接受帧后将帧的内容溢出到各个端口的集线器代码转换为具有学习能力的交换机代码。
- 任务主要有以下几个部分：
  - **创建转发表**：可以存储MAC地址和端口号。每收到一个数据帧，都要记录源MAC地址和源端口的映射关系，不断插入新的表项。

MAC_address	Interface
aa:bb:cc:dd:ee:ff	eth-0
...	...

    - **转发数据帧**：能够在转发表里找到对应的转发端口或者广播数据帧。最好可以在**$O_{(1)}$的时间**内用MAC地址寻访到端口号。
- 具体实现
    - 这里想到了`python`提供的工具`dict`类来实现，这是因为dict通过`hash`实现，可以通过`key`在$O_{(1)}$的时间寻访到`value`的内容。

```
forward_table = dict()
forward_table[MAC_address] = Interface
```

task 3:Timeouts

- 这一步要求交换机在继承了上一步的功能后具有检测表项超时逻辑上删除表项的功能。
- 我们沿用task2中完成的交换机逻辑但丰富了表项的内容来**记录最近一次接受数据帧时的时间戳**（通过python内置库函数time.time()获取）：
    - 在逻辑上转发表如下：

MAC_address	Interface	timestamp
aa:bb:cc:dd:ee:ff	eth-0	12345.12345
...	...	...

    - 在实现上我选择用`list`**取代了**`dict`**中**`value`**的内容**来满足我可以在字典中存储更多信息的需求。

```
forward_table = dict()
forward_table[MAC_address] = [Interface, timestamp]
```

- 此外还要考虑如何检测表项超时：
    - 比较容易想到的是如果可以有**并发**，可以管理另一个进程在进行超时监督，一旦超时移出转发表。这种方法作为想法容易想到，也最接近超时机制的本意。但**实现起来却比较困难。**
    - 所以，我实现了一种**伪更新**的效果：只有在查询转发表的时候才检验是否超时。如果不曾超时，从对应的端口转发出去，否则删除超时的表项，将数据帧从所有端口溢出。 **（这样意味着有些表项只要没有被查询即使超时了也不会立即被移除，但是在上层看到的效果是和逻辑上的超时机制相同的）**

task 4:Least Recently Used

- 这一步要求依据LRU替换原则维护一个固定大小的交换机转发表以提高对资源利用的效率。
- 依然在task2的原型上做修改，增加了作为LRU判断依据的表项age：即最后一次访问表项内容距当前的经过的时间。
    - 逻辑上转发表结构：

MAC_address	Interface	Age
aa:bb:cc:dd:ee:ff	eth-0	0
ee:ff:gg:hh:ii:jj:kk	eth-1	1

    - 具体实现

```
forward_table = dict()
forward_table[MAC_address] = [Interface, Age = 0]
```

- 此外要实现更新转发表的操作，除了要插入新的表项还要考虑**Age的自增**问题。

  我这里是每接受一个新的数据帧都会调用函数更新转发表将表项的Age自增一次，但未必每次都会成功插入表项。

  虽然能实现相同的效果，但我觉得如此不妥，有些多余，只需要在**需要插入新表项的时候将Age自增一次**即可。

  plus:*后来后悔了，纠正了这个问题，现在的逻辑是只有插入表项时会更新*age

```
def Update_forwardtable(forward_table, src_mac, fromIface):
```

- 在更新转发表时，发现表满要依据LRU规则删去一个表项，即表中Age最大的表项（最近最少用），遍历即可。

```
def Delete_lru(table):
```

task 5 Least Traffic Volumn

- 这一步采用流量检测的方式实现表项更新。
- 依然在task2的原型上做修改，增加了作为LTV判断依据的表项traffic：即为在转发表中表项的访问次数。
  - 逻辑上转发表结构：

MAC_address	Interface	Traffic
aa:bb:cc:dd:ee:ff	eth-0	0
ee:ff:gg:hh:ii:jj:kk	eth-1	1

  - 具体实现

```
forward_table = dict()
forward_table[MAC_address] = [Interface, Traffic = 0]
```

- 这一步要注意traffic在转发时自增即可。

4. 实验结果

task 2:Basic Switch

Deploying

打开wireshark，启动交换机代码。在Mininet中输入以下指令

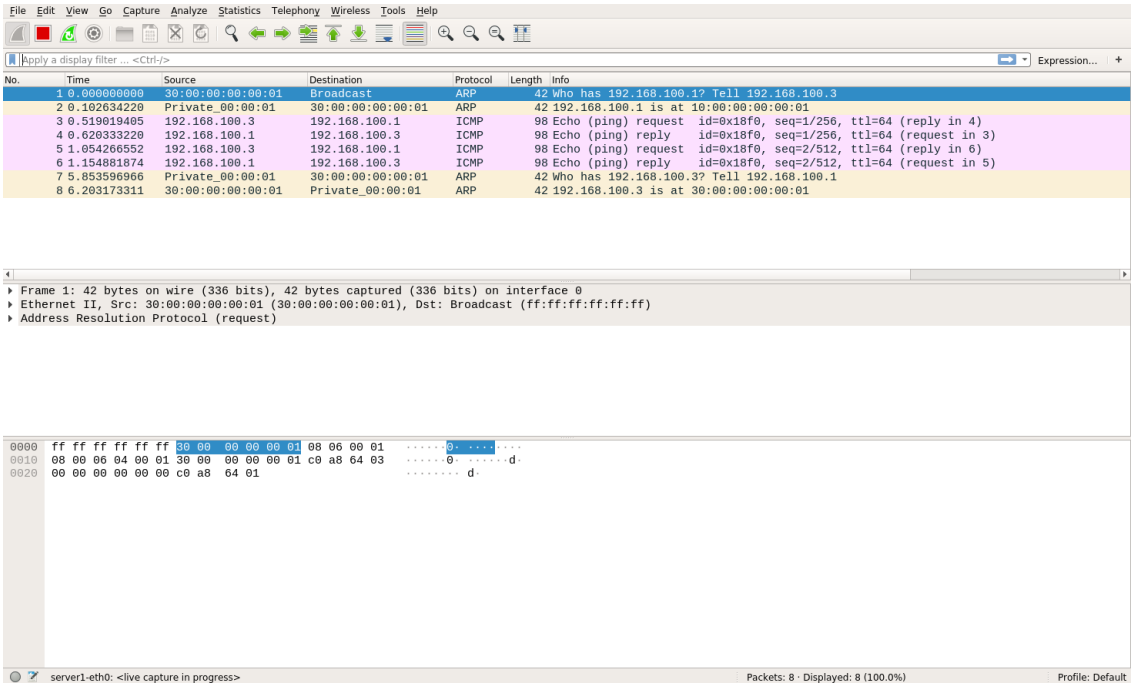
```
Mininet> client ping -c 2 server1
```

在switch的`xterm`看到运行日志：

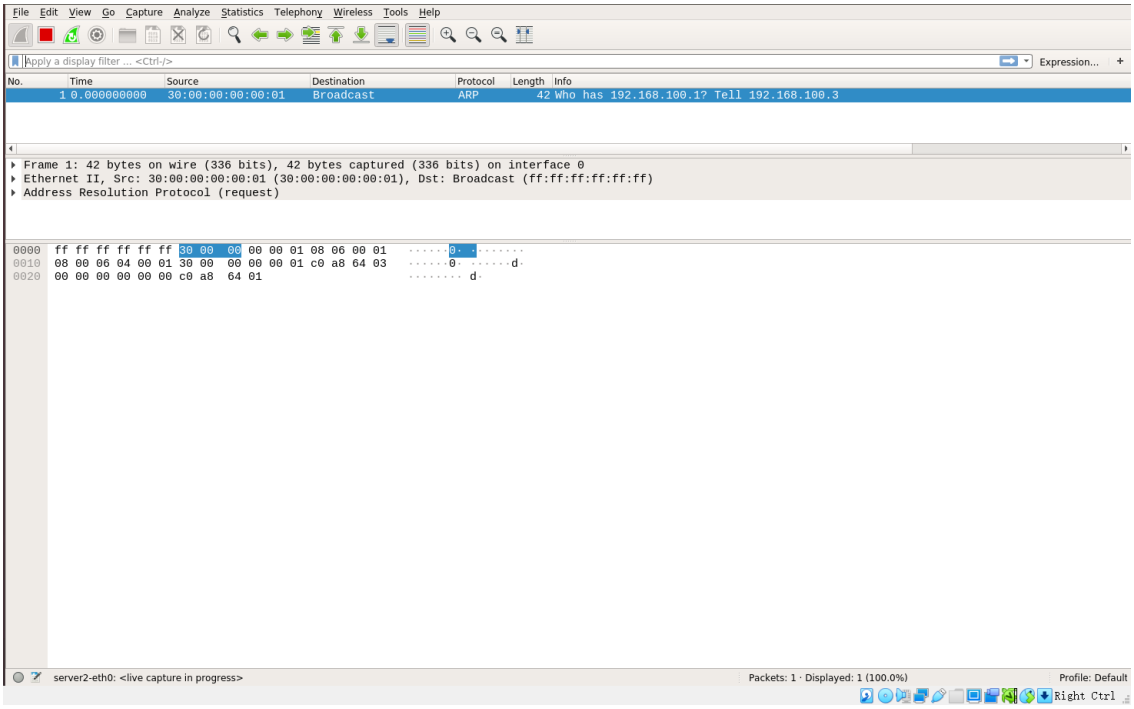
```
root@njucs-VirtualBox:~/lab-02-Wumaomaomao# source ../switchyard/syenv/bin/acti
vate
(syenv) root@njucs-VirtualBox:~/lab-02-Wumaomaomao# swyard myswitch.py
23:04:09 2022/03/21     INFO Saving iptables state and installing switchyard rul
es
23:04:09 2022/03/21     INFO Using network devices: switch-eth2 switch-eth0 swit
ch-eth1
23:04:14 2022/03/21     INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.1
00.1 to switch-eth0
23:04:14 2022/03/21     INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.1
00.1 to switch-eth1
23:04:14 2022/03/21     INFO Fowarding packet Ethernet 10:00:00:00:00:01->30:00:
00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 30:00:00:00:00:01:192.168.
100.3 to switch-eth2
23:04:14 2022/03/21     INFO Fowarding packet Ethernet 30:00:00:00:00:01->10:00:
00:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoRequest 6384
1 (56 data bytes) to switch-eth0
23:04:15 2022/03/21     INFO Fowarding packet Ethernet 10:00:00:00:00:01->30:00:
00:00:00:01 IP | IPv4 192.168.100.1->192.168.100.3 ICMP | ICMP EchoReply 6384 1
(56 data bytes) to switch-eth2
23:04:15 2022/03/21     INFO Fowarding packet Ethernet 30:00:00:00:00:01->10:00:
00:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoRequest 6384
2 (56 data bytes) to switch-eth0
23:04:15 2022/03/21     INFO Fowarding packet Ethernet 10:00:00:00:00:01->30:00:
00:00:00:01 IP | IPv4 192.168.100.1->192.168.100.3 ICMP | ICMP EchoReply 6384 2
(56 data bytes) to switch-eth2
23:04:20 2022/03/21     INFO Fowarding packet Ethernet 10:00:00:00:00:01->30:00:
00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.
100.3 to switch-eth2
23:04:20 2022/03/21     INFO Fowarding packet Ethernet 30:00:00:00:00:01->10:00:
00:00:00:01 ARP | Arp 30:00:00:00:00:01:192.168.100.3 10:00:00:00:00:01:192.168.
100.1 to switch-eth0
```

在server1和server2的wireshark上可以看到捕获结果：

- **server1**:可以看到client和server1之间进行了两次ICMP通信，说明经过交换机的转发**client和server1之间建立了双向通信。**



- **server 2**:可以看到只有在client第一次发送ARP包的时候因为**交换机表为空**没有寻找到对应server1MAC地址的端口在所有端口**溢出数据帧被server2捕获。**经过学习后数据帧直接转发到连接server1的端口上。



task 3:Timeouts

Testing

- 只进行了一次20s的超时检测，并不复杂，故不赘述。

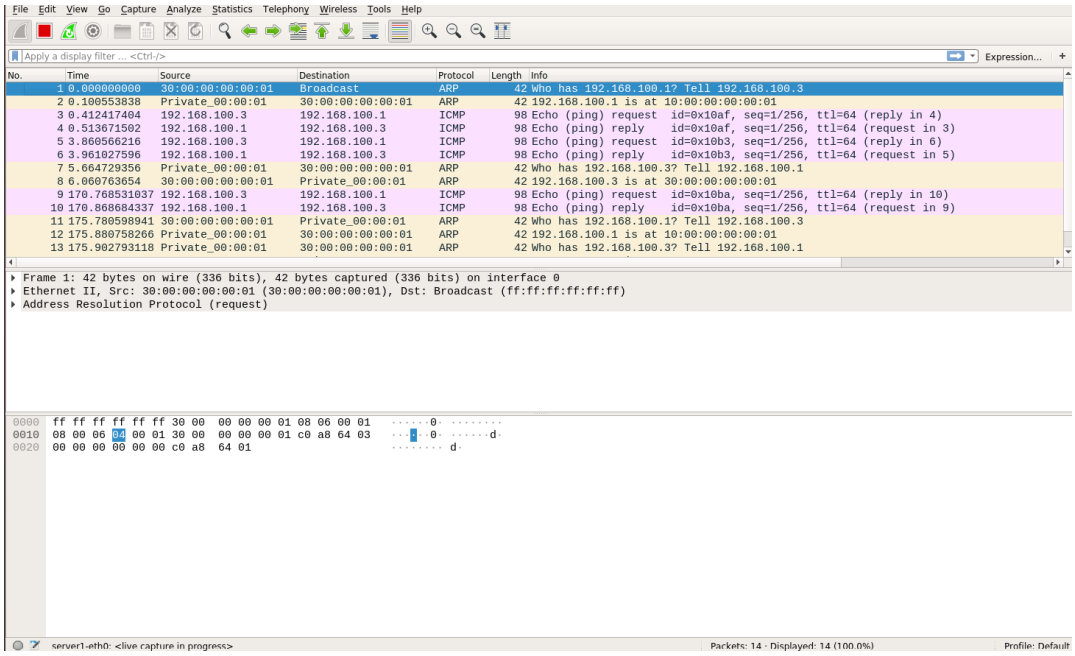
```
(syenv) njucs@njucs-VirtualBox:~/lab-02-Wumaomaomao$ swyard -t testcases/myswitch_to_testscenario.srpy myswitch_to.py
19:19:55 2022/03/22     INFO Starting test scenario testcases/myswitch_to_testscenario.srpy
19:19:55 2022/03/22     INFO timestamp:1647947995.2538276
19:19:55 2022/03/22     INFO 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
19:19:55 2022/03/22     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
19:19:55 2022/03/22     INFO timestamp:1647947995.255
19:19:55 2022/03/22     INFO Fowarding 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
19:20:15 2022/03/22     INFO timestamp:1647948015.2791407
19:20:15 2022/03/22     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->1
72.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
19:20:15 2022/03/22     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->1
72.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth2
19:20:15 2022/03/22     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.


All tests passed!
```

Deploying

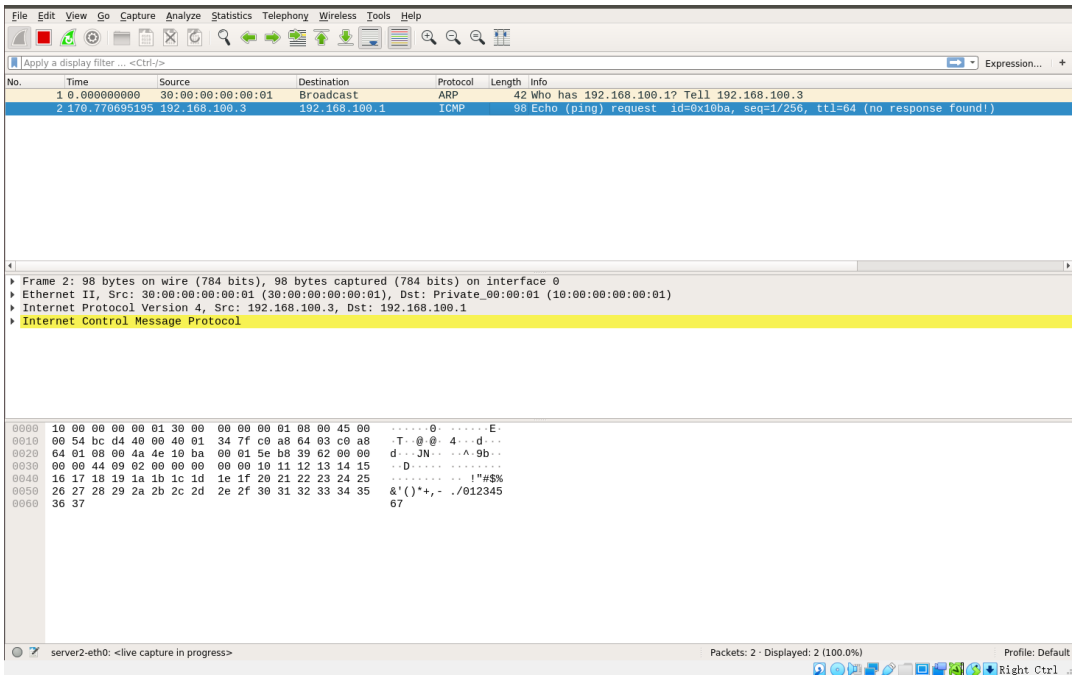
- 创建默认的Mininet拓扑网络。
- 输入以下命令共三次：**第一次，第一次后10s内（验证交换机学习端口后可以转发数据帧），第二次后10s后（验证10s后表项再次访问启动超时机制）。**

```
Mininet> client ping -c 1 server1
```

- 在server1以及server2的wireshark中捕获数据帧
  - server1:进行了三次ICMP通信。说明经过交换机client和server1建立了双向通信，但尚且不能确定交换机是否有转发机制和超时机制。



  - server2:可以看到捕获到两个数据帧：第一个是client在第一次ping的时候发出的ARP包，彼时**交换机表为空**，将数据帧溢出,第二个是client在第三次ping时发出的ICMP包，这是因为**第三次ping操作和第二次间隔超过了十秒**，使得交换机对server1的MAC地址学习已经超时，重新广播了数据帧。而其他数据包都没有被server2捕获，说明交换机将**其他数据包直接转发到了目的端口。**



task 4:Least Recently Used

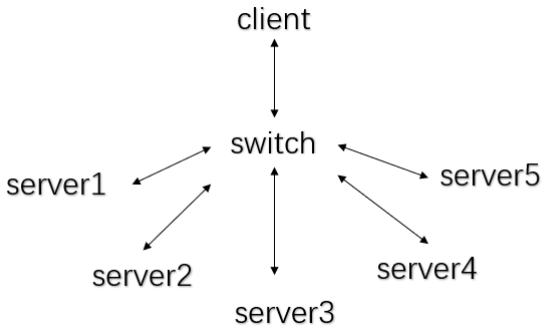
Testing

- 分析提示信息是先用一个广播的链路帧让交换机**学习了MAC地址**`30:00:00:00:00:02`**的端口**`eth-1`，然后验证了确实可以转发到`eth-1`后涌入了**6条数据帧来自5个不同的MAC地址**，根据LRU原则**冲刷了最早的一条表项**。最后再次向`30:00:00:00:00:02`发送一个链路帧，发现交换机查询转发表并未找到转发端口，故将链路帧直接在各个端口溢出。

```
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!
```

Deploying

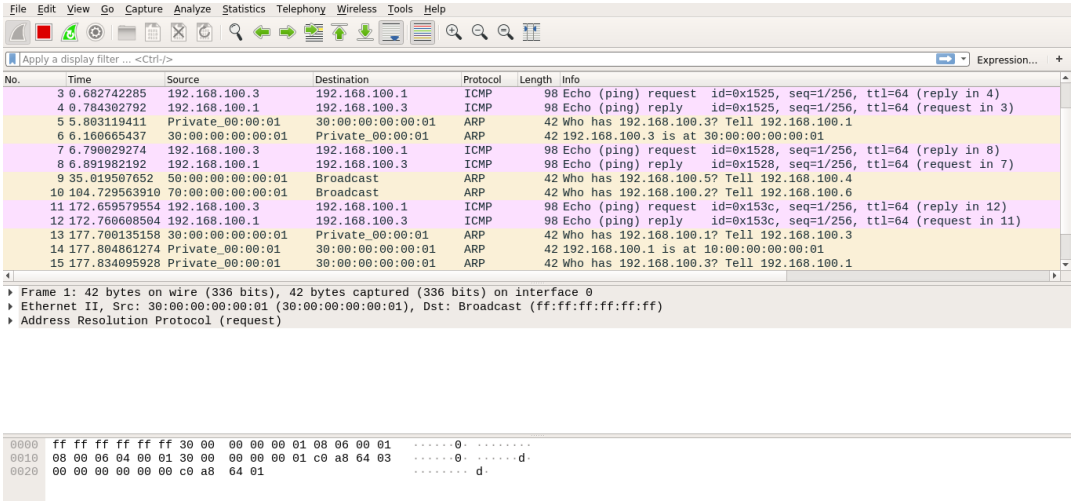
- 为了方便这一部分使用合适的Mininet网络部署交换机，我修改了`start_mininet.py`的逻辑，增加了`server3`,`server4`,`server5`三个节点和交换机连接。



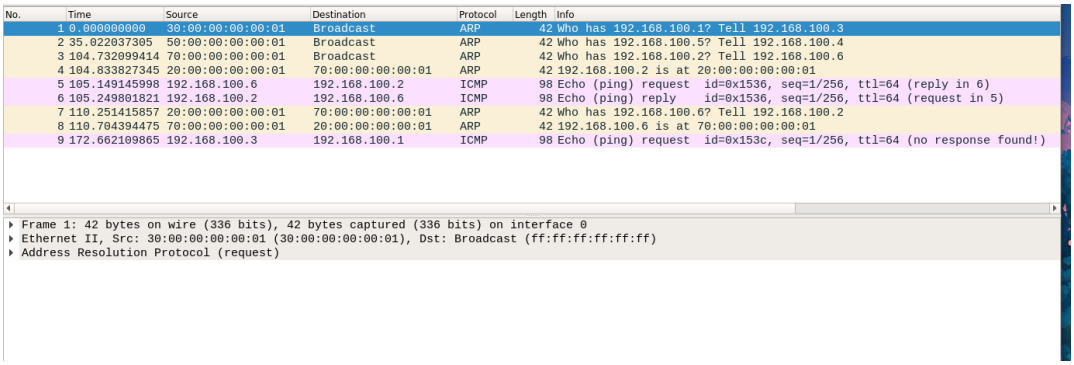
- 依次输入以下命令

```
Mininet> client ping -c 1 server1
Mininet> client ping -c 1 server1
Mininet> server3 ping -c 1 server4
Mininet> server5 ping -c 1 server2
Mininet> client ping -c 1 server1
```

  - 意在学习了server1对应的端口之后检验交换机的转发功能。然后用新的表项冲刷转换表，最后交换机将链路帧溢出到各个端口。
  - 最后一条命令也对应了FAQ中提到的一种情况，在执行前，转发表的顺序大致为：server1,server3,server4,server5,server2，而最后一条命令发送的第一个请求帧是client->server1，所以会将server1对应的表项冲刷后广播帧。
- wireshark
  - server1：可以收到经交换机转发后的client的ICMP包。



  - server2：这里比较重要，首先server2**捕获了第一次client向server1发送的ARP包**（见第一条），而第二次client向server1发送包的时候交换机**已经学习了端口**，server2便不再受到来自于client的报文。后来在server1的表项被**冲刷之后**，server2**再次可以收到**发向server1的报文了（见最后一条）。（而且还捕获了server3发送给server4的ARP包（第二条））
  -



task 5:Least Traffic Volumn

Testing

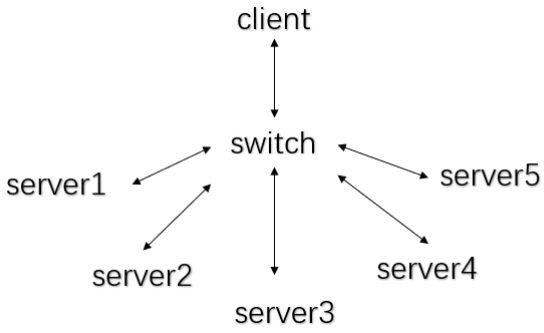
- 可以看到这个过程中转发可以让表项流量增加而广播不会，这是符合流程图要求的。但没有彰显出替换规则的实现，这将在Delploying中展开讨论。

```
(syenv) njucs@njucs-VirtualBox:~/lab-02-Wumaomaomao$ swyard -t testcases/myswitch_traffic_testscenario.srpy my
switch_traffic.py
09:45:28 2022/03/24     INFO Starting test scenario testcases/myswitch_traffic_testscenario.srpy
09:45:28 2022/03/24     INFO (mac:30:00:00:00:00:02,port:eth1,traffic:0)
09:45:28 2022/03/24     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
09:45:28 2022/03/24     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
09:45:28 2022/03/24     INFO (mac:30:00:00:00:00:02,port:eth1,traffic:0)
09:45:28 2022/03/24     INFO (mac:20:00:00:00:00:01,port:eth0,traffic:0)
09:45:28 2022/03/24     INFO Fowarding 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
09:45:28 2022/03/24     INFO (mac:30:00:00:00:00:02,port:eth1,traffic:1)
09:45:28 2022/03/24     INFO (mac:20:00:00:00:00:01,port:eth0,traffic:0)
09:45:28 2022/03/24     INFO (mac:20:00:00:00:00:03,port:eth2,traffic:0)
09:45:28 2022/03/24     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
09:45:28 2022/03/24     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
09:45:28 2022/03/24     INFO Received a packet intended for me
09:45:28 2022/03/24     INFO (mac:30:00:00:00:00:02,port:eth1,traffic:1)
09:45:28 2022/03/24     INFO (mac:20:00:00:00:00:01,port:eth2,traffic:0)
09:45:28 2022/03/24     INFO (mac:20:00:00:00:00:03,port:eth2,traffic:0)
```

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


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


ll tests passed!
```

Delpolying

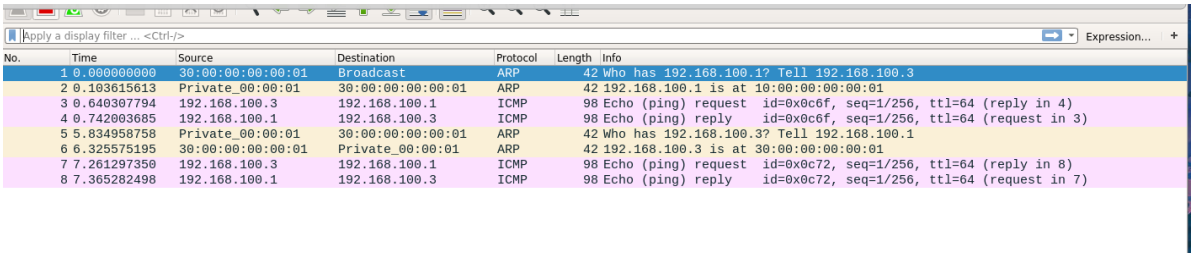
- 沿用了task4对Mininet的修改。



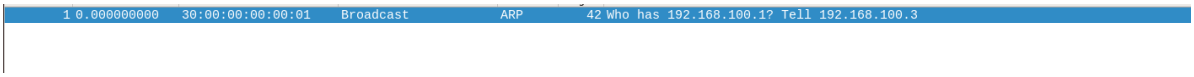
先输入

```
Mininet> client ping -c 2 server1
```

- server1可以正常通讯



- server2由于交换机的学习机制只捕获到了client第一次发出的ARP包



- 最后看交换机可以看到两个端口已经被学习，并且流量符合预期

- 然后我想要触发流量替换机制，故输入

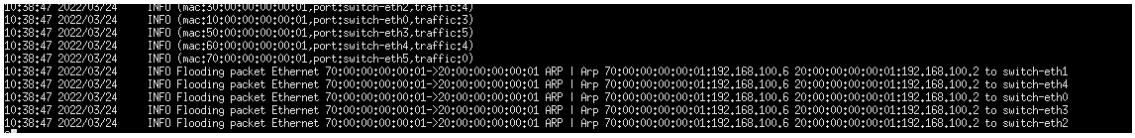
```
Mininet>server3 ping -c 3 server4
```

- 表项现状：可以看到eth3和eth4的流量为5和4

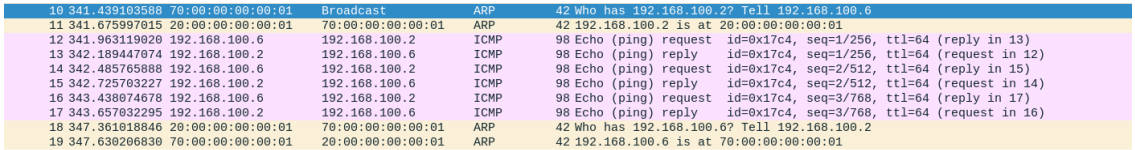
- 再次输入

```
Mininet>server5 ping -c 3 server2
```

- 此时触发了替换机制，但是出现了一种滑稽的现象：如果按照LRU的原则来看，server1的端口会最先被移除，但是我们这次是流量机制。导致剩下的两个端口反复争夺一个表项空间，不幸的是，因为反复争夺，这个表项不断变化，也就无法进行流量积累，~~实现了一次全广播通信~~。



- 这里是server1的抓包结果，从我标注的地方往下正好是server2和server5的通信内容。
- 有点像**局域网网络攻击**，用大量高频率的mac地址攻击交换机，使得交换机不得不溢出传输的数据帧，然后在局域网内架设另一个设备捕获广播数据帧即可监听通信（叹气，这么看来这种替换方法并不安全捏，如果结合超时机制的话可能会好一点）。



5. 核心代码

task 1:Preperation

task 2:Basic Switch

```
# myswitch.py
...
forward_table = dict()    #创建空的转发表
...
```

```
if eth.dst in mymacs:     #接收到一个发向交换机端口的数据帧，丢弃。但是要学习对应源mac地址和端口的映射关系。
    log_info("Received a packet intended for me")
    forward_table[eth.src] = fromIface
else:                     #接收到一个需要转发的数据帧：学习映射关系后查找转发表确定转发端口。
    forward_table[eth.src] = fromIface
            if eth.dst in forward_table:  #在转发表中找到了转发端口
                forward_interface = forward_table[eth.dst]
                intf = net.interface_by_name(forward_interface)
                log_info (f"Fowarding packet {packet} to {intf.name}")
                net.send_packet(intf, packet)
            else:                         #否则数据帧将在其他端口溢出
                for intf in my_interfaces:
                    if fromIface!= intf.name:
                        log_info (f"Flooding packet {packet} to {intf.name}")
                        net.send_packet(intf, packet)
```

task 3:Timeouts

```
# myswitch_to.py
...
forward_table = dict()
...
timestamp = time.time() #每接受一个包，获取一个当前系统的时间戳
...
if eth.dst in mymacs:   #接受到一个发送到交换机端口的数据帧，丢弃。用源MAC地址的映射关系和时间戳填写转发表。
    log_info("Received a packet intended for me")
    forward_table[eth.src] = [fromIface, timestamp]
...
else:
    log_info(f"timestamp:{timestamp}") # 输出日志：当前时间戳。
            forward_table[eth.src] = [fromIface, timestamp] #用源MAC地址的映射关系和时间戳填写转发表
            if eth.dst in forward_table:                    #若目的MAC地址在转发表中
                if forward_table[eth.dst][1] + 10 >= timestamp: #如果对应表项尚未超时，在对应端口转发
                    forward_interface = forward_table[eth.dst][0]
                    intf = net.interface_by_name(forward_interface)
                    log_info (f"Fowarding packet {packet} to {intf.name}")
                    net.send_packet(intf, packet)
                else:                                       #否则，在其他端口溢出帧并在转发表中删除超时表项
                    forward_table.pop(eth.dst)
                    for intf in my_interfaces:
                        if fromIface!= intf.name:
                            log_info (f"Flooding packet {packet} to {intf.name}")
                            net.send_packet(intf, packet)
            else:                                           #不在转发表中，在其他端口溢出帧
                for intf in my_interfaces:
                    if fromIface!= intf.name:
```

```
                        log_info (f"Flooding packet {packet} to
{intf.name}")
                        net.send_packet(intf, packet)
```

task 4:Least Recently Used

```
#myswitch_lru.py
#主体代码
...
forward_table = dict()
...
if eth.dst in mymacs:
    log_info("Received a packet intended for me")
    Update_forwardtable(forward_table,eth.src,fromIface) #更新表项
...
else:
    Update_forwardtable(forward_table,eth.src,fromIface) #更新表项
    if eth.dst in forward_table:
        forward_interface = forward_table[eth.dst][0]
        intf = net.interface_by_name(forward_interface)
        log_info (f"Fowarding packet {packet} to {intf.name}")
        net.send_packet(intf, packet)
    else:
        for intf in my_interfaces:
            if fromIface!= intf.name:
                log_info (f"Flooding packet {packet} to {intf.name}")
                net.send_packet(intf, packet)

#Update_forwardtable函数定义
def Update_forwardtable(forward_table, src_mac, dst_mac, fromIface):
    max_capcity = 5
    if src_mac in forward_table:
        if fromIface != forward_table[src_mac][0]:
            forward_table[src_mac][0] = fromIface #检测到拓扑网络变化，更新端口，
但不改变LRU次序
    else: #将插入新表项，则age = age + 1
        for key in forward_table:
            forward_table[key][1] = forward_table[key][1] + 1
        if (len(forward_table) >= max_capcity):   #超过容量，删去LRU表项
            Delete_lru(forward_table)
        forward_table[src_mac] = [fromIface, 0]   #插入新表项

    if dst_mac in forward_table:                  #如果访问已有表项进行转发，则将
已有表项更新为MRU
        for key in forward_table:
            forward_table[key][1] = forward_table[key][1] + 1
        forward_table[dst_mac][1] = 0

    for key in forward_table:
        #输出日志：转发表表项内容
        log_info(f"(mac:{key},port:{forward_table[key][0]},age:
{forward_table[key][1]})")
#Delete_lru(forward_table)
def Delete_lru(table):
    age = 0
    delete_key = 0
    for key in table: #找到age最大的表项，即LRU
```

```
        if table[key][1] >= age:
            age = table[key][1]
            delete_key = key
    table.pop(delete_key)
```

task 5:Least Traffic Volumn

```
...
forward_table = dict()
...
if eth.dst in mymacs:
    log_info("Received a packet intended for me")
    Update_forwardtable(forward_table,eth.src,eth.dst,fromIface) #更新表项
else:
    Update_forwardtable(forward_table,eth.src,eth.dst,fromIface) #更新表项
    if eth.dst in forward_table:
        forward_interface = forward_table[eth.dst][0]
        intf = net.interface_by_name(forward_interface)
        log_info (f"Fowarding packet {packet} to {intf.name}")
        net.send_packet(intf, packet)
    else:
        for intf in my_interfaces:
            if fromIface!= intf.name:
                log_info (f"Flooding packet {packet} to {intf.name}")
                net.send_packet(intf, packet)

#Update_forwardtable函数定义
def Update_forwardtable(forward_table, src_mac, dst_mac, fromIface):
    max_capcity = 5
    if src_mac in forward_table:              #如果源mac地址已经在表项中
        if fromIface != forward_table[src_mac][0]: #拓扑网络发生变化，更新端口但不修改流量记录
                forward_table[src_mac][0] = fromIface
    else:                                     #插入新的表项
        if (len(forward_table) >= max_capcity): #表满，剔除LTV表项
            Delete_low_traffic(forward_table)
        forward_table[src_mac] = [fromIface, 0] #插入新表项，流量为0

    if dst_mac in forward_table:                #如果转发端口在转发表中，记录流量
        forward_table[dst_mac][1] = forward_table[dst_mac][1] + 1
    for key in forward_table:                   #转发表更新后输出日志
        log_info(f"(mac:{key},port:{forward_table[key][0]},traffic:{forward_table[key][1]})")
```

6. 总结与感想

~~不做DDL战士，从我做起，从计网做起QAQ。~~

交换机的转发表更新机制依据不同的替换算法可以得到不同的效果，各自都有优点，也可以有可以被坏人利用的缺陷。想必真正部署的情况下，需要考虑更多的因素，需要更好的结合多种算法的特征，来得到最好的转发效果。