南京大学

LAB 2：Learning Switch

课程名称：计算机网络

姓名：孙文博

学号：201830210

学院：计算机科学与技术系

Email：201830210@smail.nju.edu.cn

任课教师：李文中

实验时间：2022.3.7 - 2022.3.24

一、 实验目的

以太网交换机是第 2 层设备，它使用数据包交换来接收、处理帧并将其转发到网络中的其他设备（终端主机、其他交换机）。 交换机具有一组接口（端口），通过这些接口（端口）发送/接收以太网帧。当以太网帧到达任何端口时，交换机会处理帧的报头以获取有关目标主机的信息。如果交换机知道主机可以通过它的一个端口到达，它就会从相应的输出端口发出帧；如果它不知道主机在哪里，它会将帧从除输入端口之外的所有端口中溢出。

本次实验中我们将模拟交换机的逻辑，理想的交换机可以识别处理发给自身的帧以及目标地址为广播地址 FF:FF:FF:FF:FF:FF 的帧。我们将分别采用三种方式实现模拟，分别是：

- 定期从转发表中删除过期条目（超过十秒未更新）；
- 从转发表中删除最近最少使用（LRU）条目。对于此功能，假设我们的转发表一次只能容纳 5 个条目，如果有新条目出现并且表已满，我们将删除与以太网帧目标地址匹配时间最长的条目。
- 删除流量最小的条目。对于此功能，同样假设我们的表一次只能容纳 5 个条目，条目的流量是交换机累计接收到的帧数，即 Destination MAC address == MAC address of entry。

二、 实验内容

1. 基础交换机

以太网学习交换机是一种设备，它具有一组接口（端口），通过链路连接到其他交换机和终端主机。在课内知识学习中我们知道，交换机是“自学习的”，即当以太网帧到达任何端口/接口时，如果交换机知道可以通过该端口访问主机，则交换机会在适当的输出端口上发送帧，或者如果它不知道主机在哪里，则将帧从所有端口转发。

模拟一个基础交换机我们需要维护一个转发表 My_table，它可以通过 Python 中字典的结构实现，如下：

```
# switch table
my_table = {} # a dict
cnt = 0 # the number of element
```

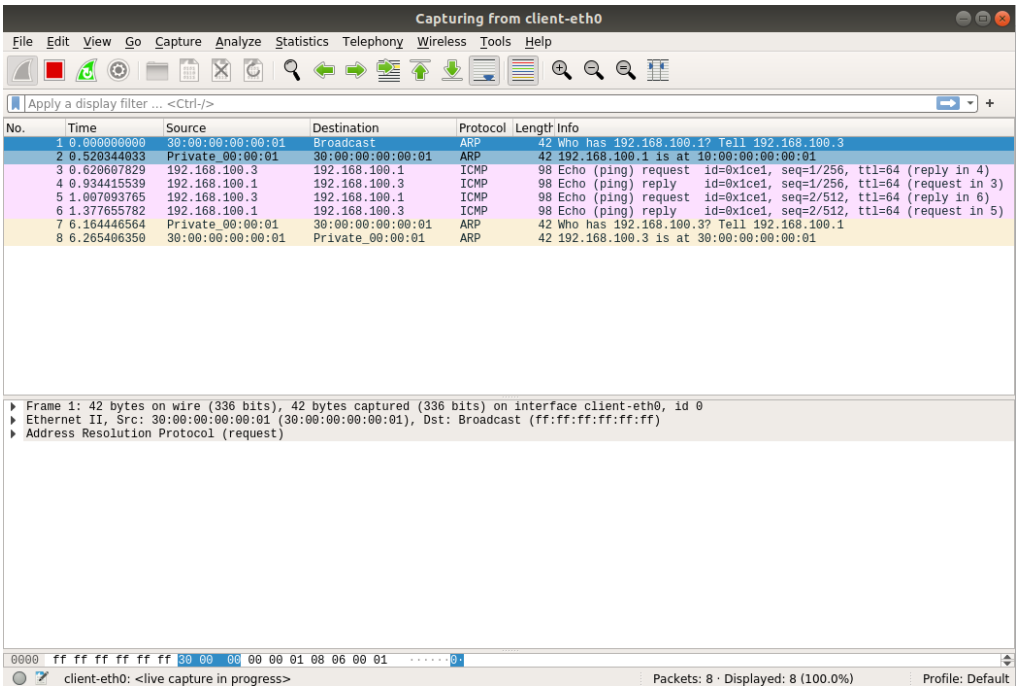
每当有帧来到交换机时，它包含两个 MAC 地址——源地址 src 和目标地址 dst，我们先分析 src 地址是否在转发表中，如存在在更新其对应接口，否则添加到转发表中来，cnt++；再根据 dst 地址查表，若字典中存在 dst 地址则直接发到对应端口，否则泛洪转发。对应代码如下：

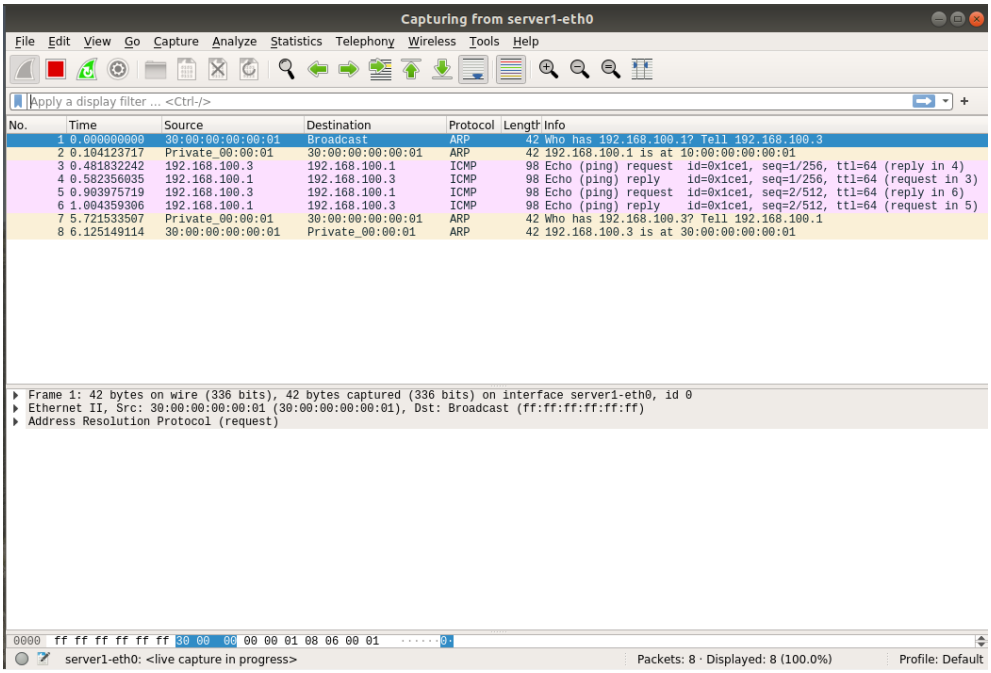
```
# add an information
flag = True
for item in my_table:
    if item == eth.src:
        flag = False
        break
if(flag):
    my_table[eth.src] = fromIface
    cnt = cnt + 1
    log_info("Add an information.")
```

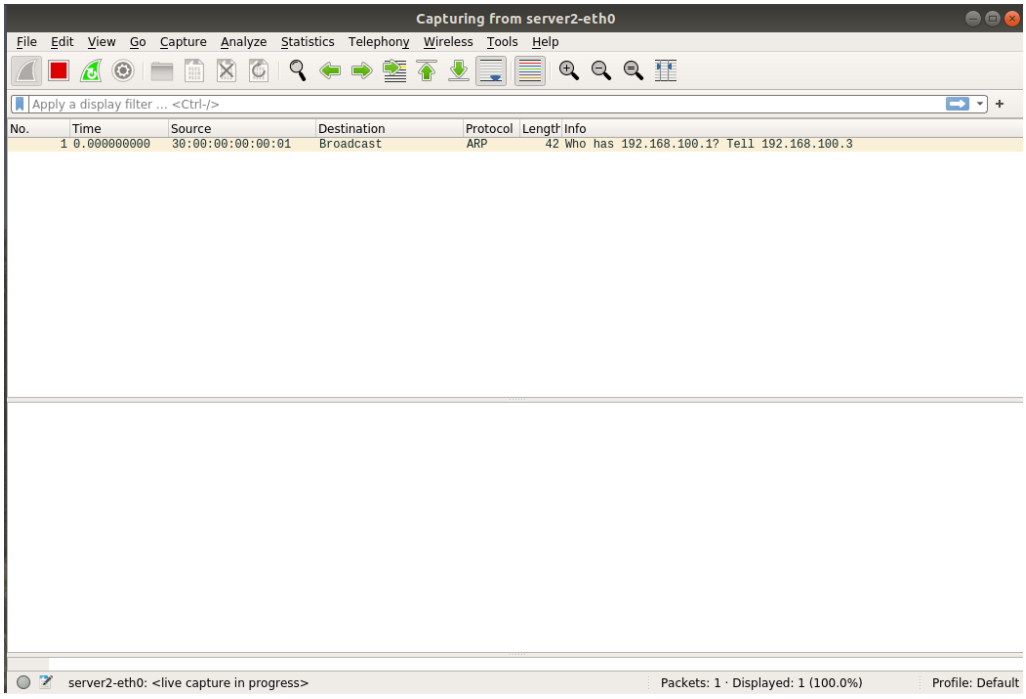
```
        # search the table
        else:
            flag = False
            for item in my_table:
                if item == eth.dst: # find the dest's address
                    flag = True
                    for intf in my_interfaces: # find the interface
                        if intf.name == my_table[item]:
                            log_info (f"Send the packet {packet} to {intf.name}")
                            net.send_packet(intf, packet)

            if(flag == False): # not find, flood the packet except input port
                for intf in my_interfaces:
                    if intf.name != fromIface:
                        log_info (f"Flooding packet {packet} to {intf.name}")
                        net.send_packet(intf, packet)
```

我们在 xterm 终端上模拟 switch 交换机，在 mininet 中模拟主机之间的数据交换，输入“client ping -c 2 192.168.100.1”，此命令将从 client 向 server1 节点发送两个“echo”请求，同时用 wireshark 软件捕获我们的结果：

```
*** Starting CLI:
mininet> client wireshark &
mininet> server1 wireshark &
mininet> server2 wireshark &
mininet> xterm switch
mininet> client ping -c 2 192.168.100.1
```



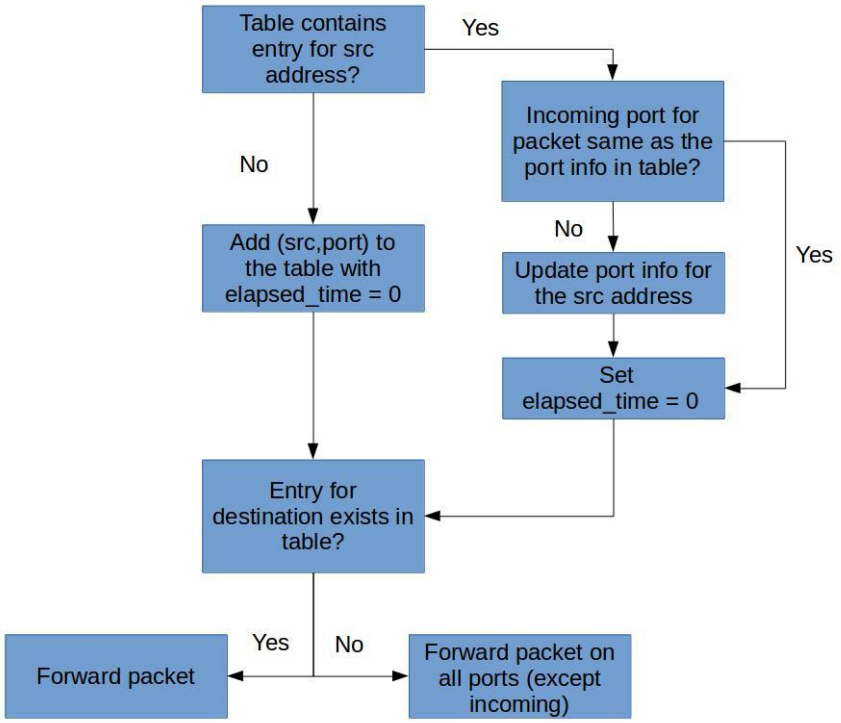




可以发现，server1 节点对 client 的每一个请求做出响应，一共两个回显请求和回显回复，另外还有几个其他数据包（例如 ARP 等地址解析协议数据包）。而在 server2 上运行的 Wireshark，没有回显请求和回复数据包（除了 ARP 数据包，因为它们是使用广播目标地

址发送的），这正符合我们的预期，因为交换机转发表中已有 client 和 server1 对应端口，不会再向 server2 转发。

2. 基于超时的改进



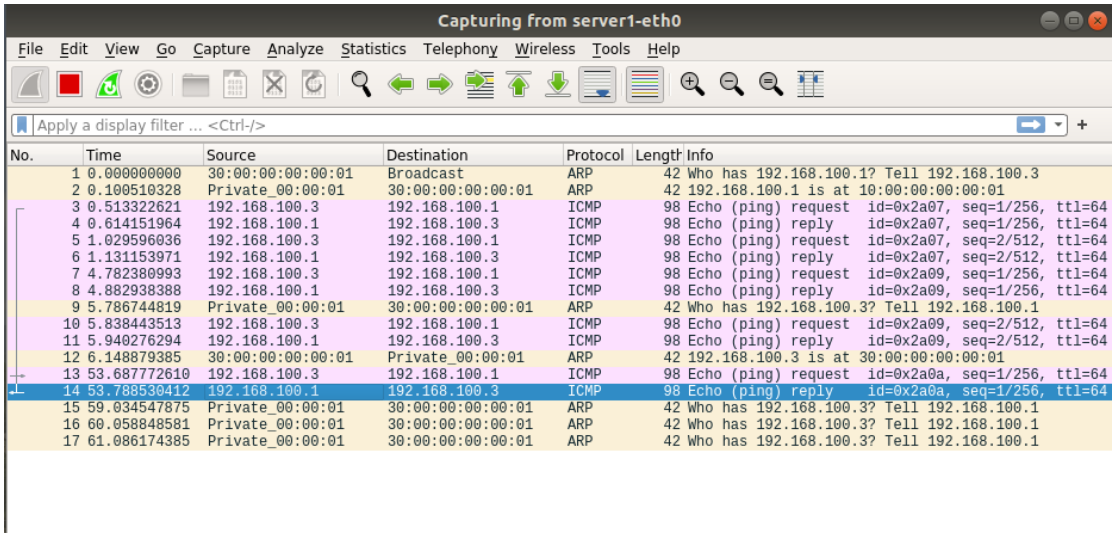
基于超时交换机的逻辑如上图，每条信息都会增加一个时间戳属性，以判断当前信息存在了多长时间，若超过 10s，则自动从转发表中删除。这里我们用到了 python 中的 time 模块，它可以返回当前时间的时间戳，我们单独开一个 start_time 字典记录每条信息的开始时间，代码如下：

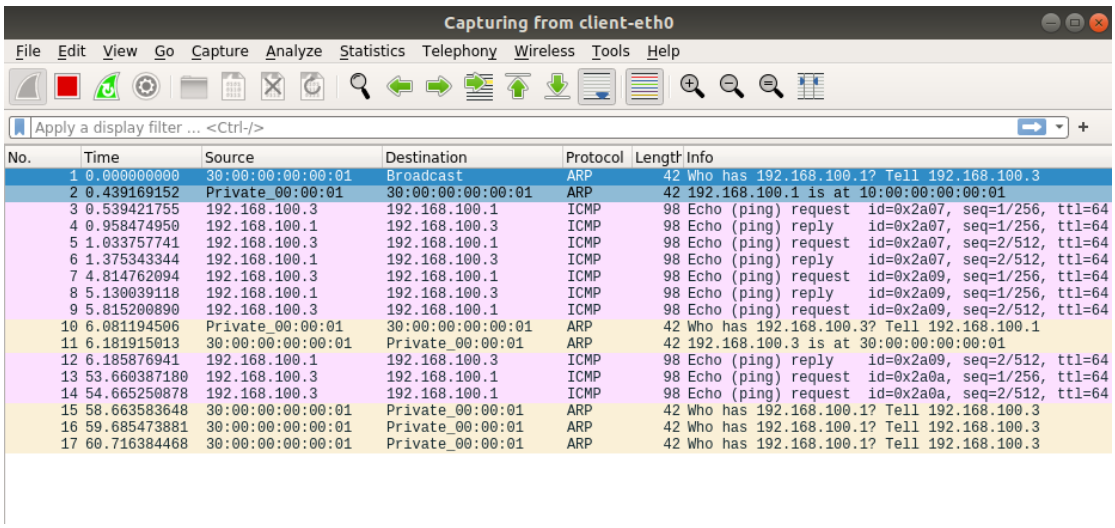
```
# delete overdue information
for item in start_time:
    if time.time()-start_time[item]>10:
        del my_table[item]
        log_info("delete an information.")
```

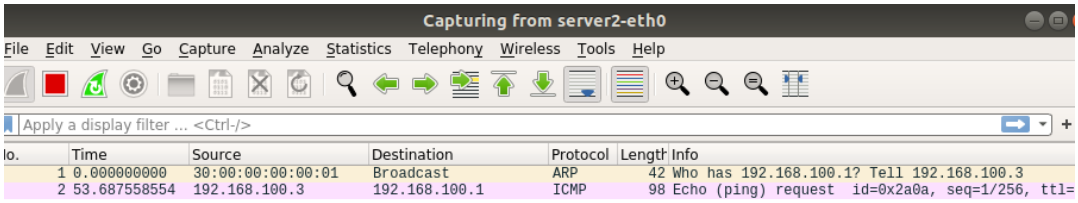
执行 testcase 中的一系列测试样例，得到的结果如下：

```
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
```

在 mininet 上模拟主机之间的数据传输，并用 wireshark 捕获结果如下：

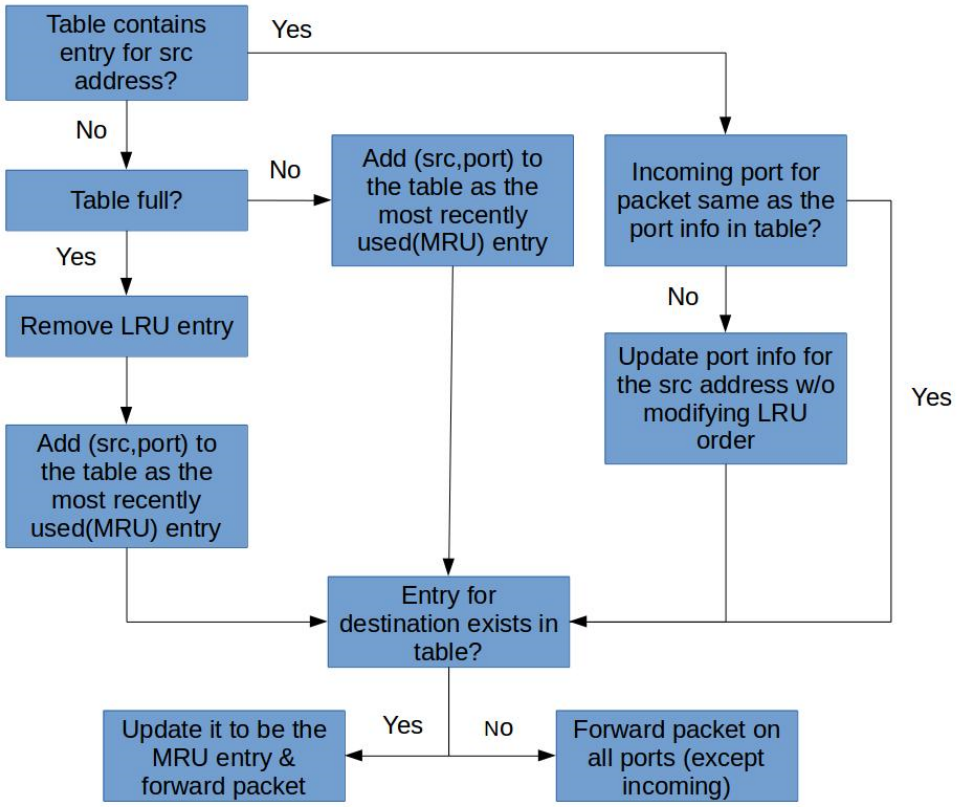






其中我们输入了两次“client ping -c 2 192.168.100.1”命令，并在两条命令间手动间隔 10s,可以看到,转发表之前的内容自动删除，server2 也收到了 client 发给 server1 的数据帧。

3. 基于 LRU 的改进



LRU 机制的流程如上图所示，我们给每条信息增加一个权值，权值的规定根据它最近使用的次序，例如我们的交换机中转发表已有 5 个条目：[h2, h3, h4, h5, h1]，其中 h2 是最长时间未匹配的条目

h1 是最近匹配的条目。此时如果交换机收到数据包(h6, h2)，它将添加一个条目 h6，因为它不在表中。但是，由于表已满（5 个条目），需要删除 LRU 条目，即 h2。 所以转发表变为：[h3, h4, h5, h1, h6]，并且交换机将在除传入端口之外的所有端口上广播传入的数据包，因为它没有关于 h2 的信息了。对应实现的代码如下：

```
# add an information
flag = True
for item in my_table:
    if item == eth.src:
        flag = False
        # updata information
        my_table[item] = fromIface
        log_info("Updata an information.")
        break

if(flag):
    if cnt < 5:
        my_table[eth.src] = fromIface
        cnt = cnt + 1
        LRU[eth.src] = 5
        for item in LRU:
            LRU[item] = LRU[item] - 1
    else:
        my_table[eth.src] = fromIface
        LRU[eth.src] = 5
        for item in LRU:
            LRU[item] = LRU[item] - 1
            if LRU[item] < 0:
                del my_table[item]
                log_info("Delete an information.")

    log_info("Add an information.")
    print(LRU)
```

其中 LRU 是我们维护的字典，记录每条信息对应的权值，分别为 0，1，2，3，4；新增条目时，根据转发表中信息条数进行判断，同时修改所有条目的 LRU 值，从而实现新来的数据放在最前面，删除最久的数据。

```
    # search the table
    else:
        flag = False # is in table
        for item in my_table:
            if item == eth.dst: # find the dest's address
                flag = True
                for i in LRU:
                    if LRU[i] > LRU[item]:
                        LRU[i] = LRU[i] - 1
                LRU[item] = 4
                print(LRU)
```

这部分代码是对转发表中用到的目的地址，将它的 LRU 值更新到最大（值为 4）。执行 testcase 中的一系列测试样例，得到的结果如下：

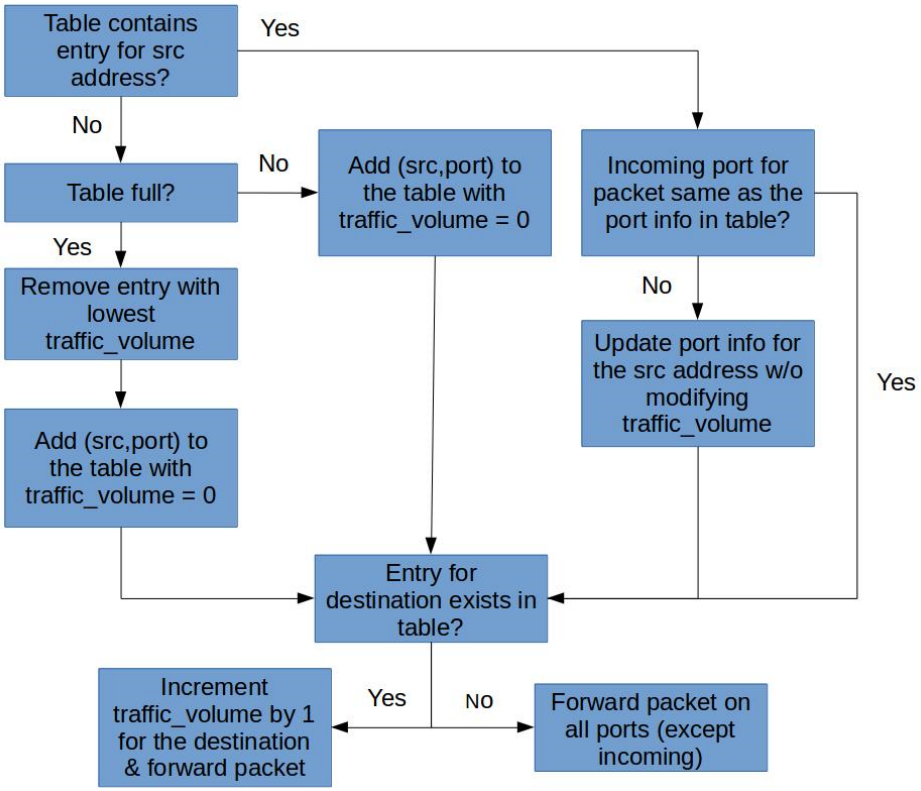
```
Passed:
1   An Ethernet frame with a broadcast destination address
    should arrive on eth1
2   The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0, eth2, eth3 and eth4
3   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive 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
```

在 mininet 中，我们在不改变拓扑结构的情况下观察每次的 LRU 表，从而判断是否与我们的预期相符，LRU 表的变化如下：

```
21:15:42 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 eth4
21:15:42 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 3, EthAddr('20:00:00:00:00:01'): 4}
{EthAddr('30:00:00:00:00:02'): 4, EthAddr('20:00:00:00:00:01'): 3}
21:15:42 2022/03/24     INFO Send the 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
21:15:42 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 3, EthAddr('20:00:00:00:00:01'): 2, EthAddr('20:00:00:00:00:03
'): 4}
{EthAddr('30:00:00:00:00:02'): 4, EthAddr('20:00:00:00:00:01'): 2, EthAddr('20:00:00:00:00:03
'): 3}
21:15:42 2022/03/24     INFO Send the packet Ethernet 20:00:00:00:00:03->30:00:00:00:00:02 IP
 | IPv4 192.168.1.102->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
21:15:42 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 3, EthAddr('20:00:00:00:00:01'): 1, EthAddr('20:00:00:00:00:03
'): 2, EthAddr('30:00:00:00:00:04'): 4}
{EthAddr('30:00:00:00:00:02'): 2, EthAddr('20:00:00:00:00:01'): 4, EthAddr('20:00:00:00:00:03
'): 1, EthAddr('30:00:00:00:00:04'): 3}
21:15:42 2022/03/24     INFO Send the packet Ethernet 30:00:00:00:00:04->20:00:00:00:00:01 IP
 | IPv4 172.16.42.4->192.168.1.100 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
21:15:42 2022/03/24     INFO Updata an information.
{EthAddr('30:00:00:00:00:02'): 2, EthAddr('20:00:00:00:00:01'): 3, EthAddr('20:00:00:00:00:03
'): 1, EthAddr('30:00:00:00:00:04'): 4}
21:15:42 2022/03/24     INFO Send the packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:04 IP
 | IPv4 192.168.1.100->172.16.42.4 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth3
21:15:42 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 1, EthAddr('20:00:00:00:00:01'): 2, EthAddr('20:00:00:00:00:03
'): 0, EthAddr('30:00:00:00:00:04'): 3, EthAddr('40:00:00:00:00:05'): 4}
{EthAddr('30:00:00:00:00:02'): 1, EthAddr('20:00:00:00:00:01'): 4, EthAddr('20:00:00:00:00:03
'): 0, EthAddr('30:00:00:00:00:04'): 2, EthAddr('40:00:00:00:00:05'): 3}
21:15:42 2022/03/24     INFO Send the packet Ethernet 40:00:00:00:00:05->20:00:00:00:00:01 IP
 | IPv4 128.16.42.4->192.168.1.100 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
21:15:42 2022/03/24     INFO Delete an information.
21:15:42 2022/03/24     INFO Add an information.
```

其中打印的字典是 LRU，输出转发表每个 MAC 地址对应的 LRU 值。

4. 基于流量的改进



基于流量的改进方法类似LRU，只是将先后顺序换为经过的流量，转发表满时，删除流量最低的那个信息，我们维护一个 traffic 字典记录每个 MAC 地址的流量，使用 python 中 min(traffic)获取最小流量的 MAC 地址，将其从 My_table 中删除，代码如下：

```
# add an information
flag = True
for item in my_table:
    if item == eth.src:
        flag = False
        # updata information
        my_table[item] = fromIface
        log_info("Updata an information.")
        break

if(flag):
    if cnt < 5:
        my_table[eth.src] = fromIface
        traffic[eth.src] = 0
        cnt = cnt + 1
    else:
        my_table[eth.src] = fromIface
        traffic[eth.src] = 0
        del my_table[min(traffic)]
        log_info("Delete an information.")
    log_info("Add an information.")
    print(traffic)

# search the table
else:
    flag = False # is in table
    for item in my_table:
        if item == eth.dst: # find the dest's address
            flag = True
            traffic[item] = traffic[item]+1
            print(traffic)
```

执行 testcase 中的一系列测试样例，得到的结果如下：

```
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!
```

在此过程中类似于 mininet 上的验证，我们打印了 traffic 字典，可以看到实时流量数据：

```
(testsyenv) njucs@njucs-VirtualBox:~/swtichyard/lab-02-Florentino-73$ swyard -t testcases/mys
witch_traffic_testscenario.srpy myswitch.py
21:28:53 2022/03/24     INFO Starting test scenario testcases/myswitch_traffic_testscenario.s
rpy
21:28:53 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 0}
21:28:53 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
21:28:53 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
21:28:53 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 0, EthAddr('20:00:00:00:00:01'): 0}
{EthAddr('30:00:00:00:00:02'): 1, EthAddr('20:00:00:00:00:01'): 0}
21:28:53 2022/03/24     INFO Send the 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
21:28:53 2022/03/24     INFO Add an information.
{EthAddr('30:00:00:00:00:02'): 1, EthAddr('20:00:00:00:00:01'): 0, EthAddr('20:00:00:00:00:03
'): 0}
21:28:53 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
21:28:53 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
21:28:53 2022/03/24     INFO Updata an information.
21:28:53 2022/03/24     INFO Received a packet intended for me
```

结果是符合预期的，实验成功！

三、 核心代码

本次实验需要修改的三个文件主要代码截图已在上面的过程中展示并说明，具体代码已提交到 classroom 中。

四、 总结与反思

本次实验相较第一次实验难度并没有提升，反而是省去了配置环境等繁琐步骤，实验关键在于理解课本中交换机的相关知识并学会转发表原理，此外还需要对 python 语法有一些基础理解。但是由于时间分配原因，最后卡着 ddl 验收，实验报告也拖到了截止后半小时😭，之后需要好好反思。此外要感谢验收的助教 gg，检查了实验的重点和关键部分，省去不必要的时间浪费，并在创新的地方给了我鼓励和称赞❤️，之后的实验一定合理分配时间，加油！💪