实验报告

一、实验名称

Learning Switch

二、实验目的

学习switch转发数据包的机制、以及筛选的三种规则

三、实验内容

Task 2: Basic Switch

在收到一个包的时候，switch会学习这个包的来源src_mac与其对应的端口，在下次发送的时候就可以直接发向那个端口

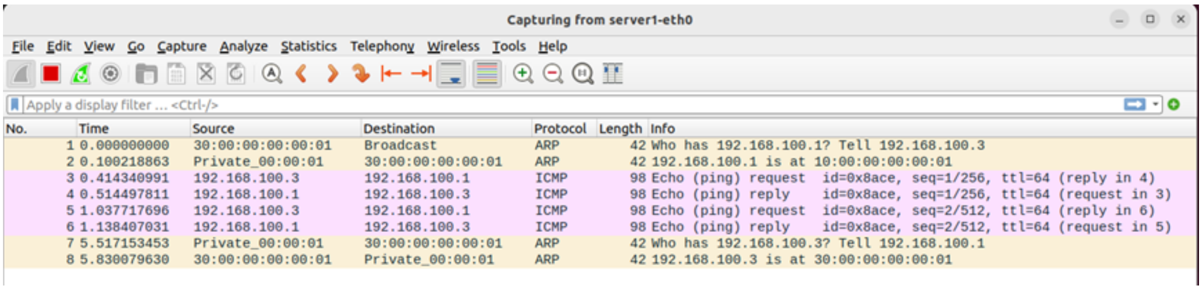
如果不知道一个包的去处dst_mac对应的端口，那么switch会把包发到除发送端外的其他所有端口；如果知道一个包的去处对应的端口，那么switch会发到相应的端口

mininet 测试：

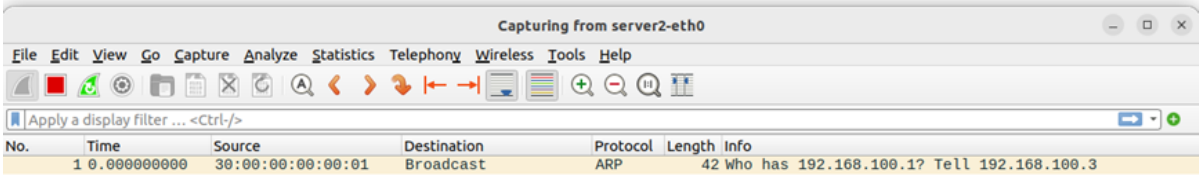
在client的终端中 ping -c 2 192.168.100.1，发送两个“echo”请求到server1的IP地址（192.168.100.1），然后server1回复这两个请求。

switch先是不知道192.168.100.1应该发到哪个端口，所以给server2也发了一个，第二次就知道了，所以第二次只给server1发

在client ping server1的过程中，switch知道了192.168.100.3是哪个端口，在server1回复的时候就直接发给了client


Capturing from server1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
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 0.100218863   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 0.414340991   192.168.100.3       192.168.100.1       ICMP       98 Echo (ping) request  id=0x8ace, seq=1/256, ttl=64 (reply in 4)
    4 0.514497811   192.168.100.1       192.168.100.3       ICMP       98 Echo (ping) reply    id=0x8ace, seq=1/256, ttl=64 (request in 3)
    5 1.037717696   192.168.100.3       192.168.100.1       ICMP       98 Echo (ping) request  id=0x8ace, seq=2/512, ttl=64 (reply in 6)
    6 1.138407031   192.168.100.1       192.168.100.3       ICMP       98 Echo (ping) reply    id=0x8ace, seq=2/512, ttl=64 (request in 5)
    7 5.517153453   Private_00:00:01    30:00:00:00:00:01   ARP        42 Who has 192.168.100.3? Tell 192.168.100.1
    8 5.830079630   30:00:00:00:00:01   Private_00:00:01    ARP        42 192.168.100.3 is at 30:00:00:00:00:01
```


Capturing from server2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
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
```

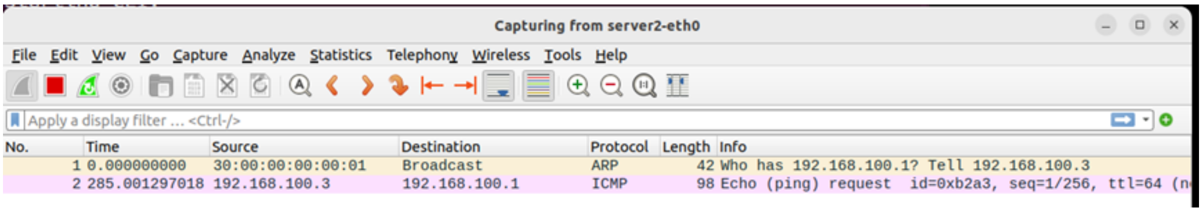
Task 3: Timeouts

每次加入规则时顺带记录加入时间

加入时间与当前时间相差十秒以上的mac和端口对应规则将被删除

mininet测试：

先用client ping 一次server1，server1回复，然后等十秒以上，再ping一次，这时关于server1的记忆已经超时清除了，switch会把client的消息广播出去，这时server2收到了client的消息



Capturing from server2-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
   2 285.001297018 192.168.100.3       192.168.100.1   ICMP         98 Echo (ping) request  id=0xb2a3, seq=1/256, ttl=64 (n
```

Task 4: Least Recently Used

规则有最大条数限制（这里为5），如果容量已满，加入新规则时，将最久未使用的规则（这里视为队尾）删除。

对于一个包，发送端和接收端都要作为最新使用的规则来更新（从队列中移除重新加入）

Mininet测试：

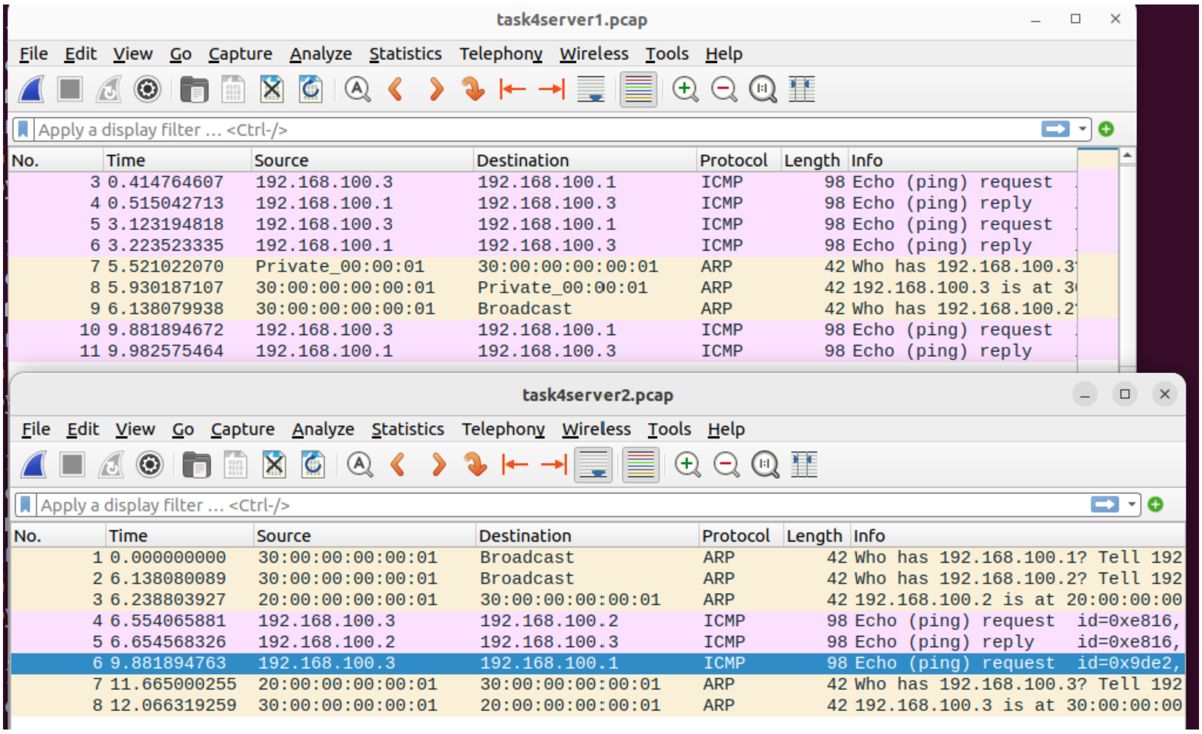
最大条数改为2

先用client来ping server1 一次，这时switch也发给了server2，学习了server1和client，

Client再ping server1 一次，这时switch已经学习了server1和client，没有发给server2，

Client ping server2，这时switch学习了client和server2，忘了server1

Client ping server1 这时switch忘了server1，会把消息也发给server2



task4server1.pcap

```
No.  Time          Source              Destination         Protocol Length Info
   3 0.414764607   192.168.100.3       192.168.100.1       ICMP         98 Echo (ping) request
   4 0.515042713   192.168.100.1       192.168.100.3       ICMP         98 Echo (ping) reply
   5 3.123194818   192.168.100.3       192.168.100.1       ICMP         98 Echo (ping) request
   6 3.223523335   192.168.100.1       192.168.100.3       ICMP         98 Echo (ping) reply
   7 5.521022070   Private_00:00:01    30:00:00:00:00:01   ARP          42 Who has 192.168.100.3
   8 5.930187107   30:00:00:00:00:01   Private_00:00:01    ARP          42 192.168.100.3 is at 3
   9 6.138079938   30:00:00:00:00:01   Broadcast           ARP          42 Who has 192.168.100.2
  10 9.881894672   192.168.100.3       192.168.100.1       ICMP         98 Echo (ping) request
  11 9.982575464   192.168.100.1       192.168.100.3       ICMP         98 Echo (ping) reply
```

task4server2.pcap

```
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
   2 6.138080089   30:00:00:00:00:01   Broadcast           ARP          42 Who has 192.168.100.2? Tell 192
   3 6.238803927   20:00:00:00:00:01   30:00:00:00:00:01   ARP          42 192.168.100.2 is at 20:00:00:00
   4 6.554065881   192.168.100.3       192.168.100.2       ICMP         98 Echo (ping) request  id=0xe816,
   5 6.654568326   192.168.100.2       192.168.100.3       ICMP         98 Echo (ping) reply    id=0xe816,
   6 9.881894763   192.168.100.3       192.168.100.1       ICMP         98 Echo (ping) request  id=0x9de2,
   7 11.665000255  20:00:00:00:00:01   30:00:00:00:00:01   ARP          42 Who has 192.168.100.3? Tell 192
   8 12.066319259  30:00:00:00:00:01   20:00:00:00:00:01   ARP          42 192.168.100.3 is at 30:00:00:00
```

Task 5: Least Traffic Volume

规则有最大条数限制（这里为5），如果容量已满，加入新规则时，将最少使用的规则（这里视为堆顶）删除。

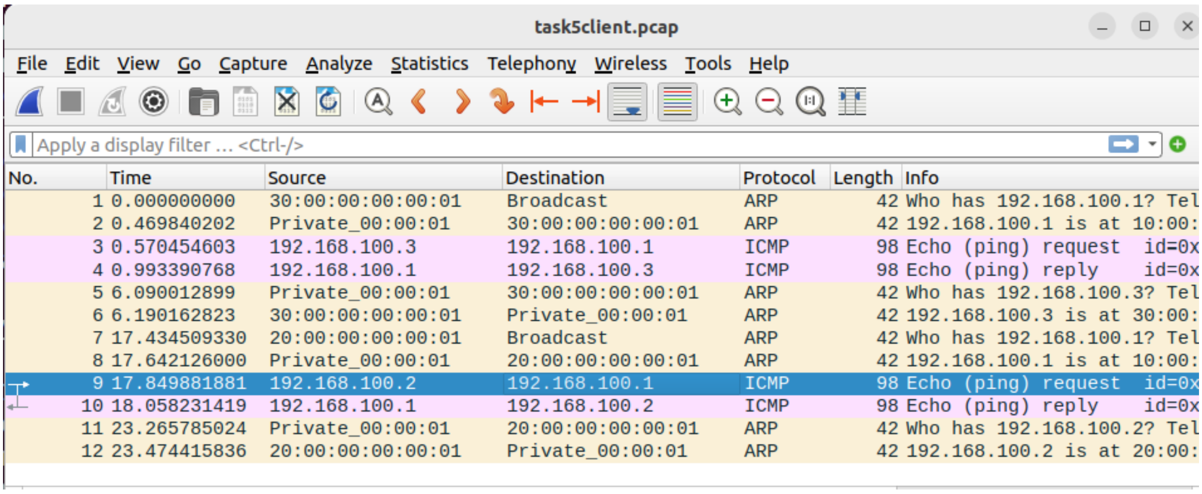
对于一个包，接收端需要更新它的使用次数+1

mininet测试：

最大条数改为2

client ping server1 ： client -> server1 ， switch学习了client，不知道server1，到处转发；server1 -> client，switch学习了server1，client使用次数+1

server2 ping server1 ： server2 -> server1，switch学习了server2，忘了server1，到处转发（这样client也收到了）； server1->server2，switch学习了server1，忘了server2，到处转发（这样client又收到了）

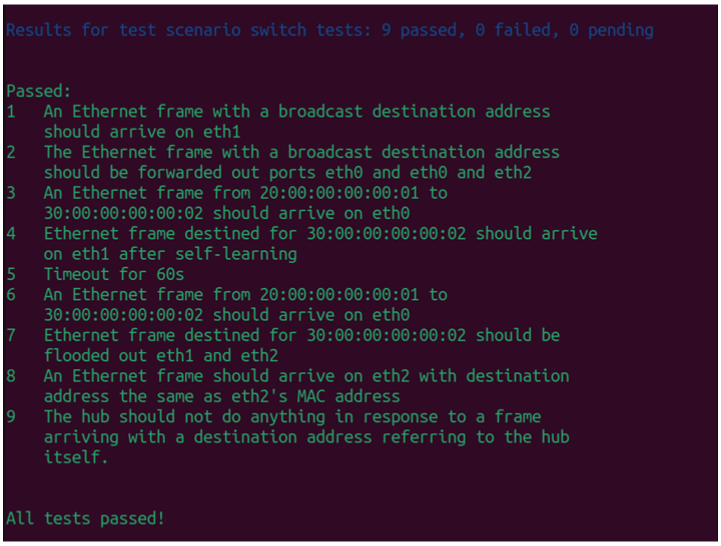

task5client.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

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? Tel
2	0.469840202	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:
3	0.570454603	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x
4	0.993390768	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x
5	6.090012899	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tel
6	6.190162823	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:
7	17.434509330	20:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tel
8	17.642126000	Private_00:00:01	20:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:
9	17.849881881	192.168.100.2	192.168.100.1	ICMP	98	Echo (ping) request id=0x
10	18.058231419	192.168.100.1	192.168.100.2	ICMP	98	Echo (ping) reply id=0x
11	23.265785024	Private_00:00:01	20:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tel
12	23.474415836	20:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 20:00:

四、实验结果

通过了给出的三个测试用例

Task 3: Timeouts

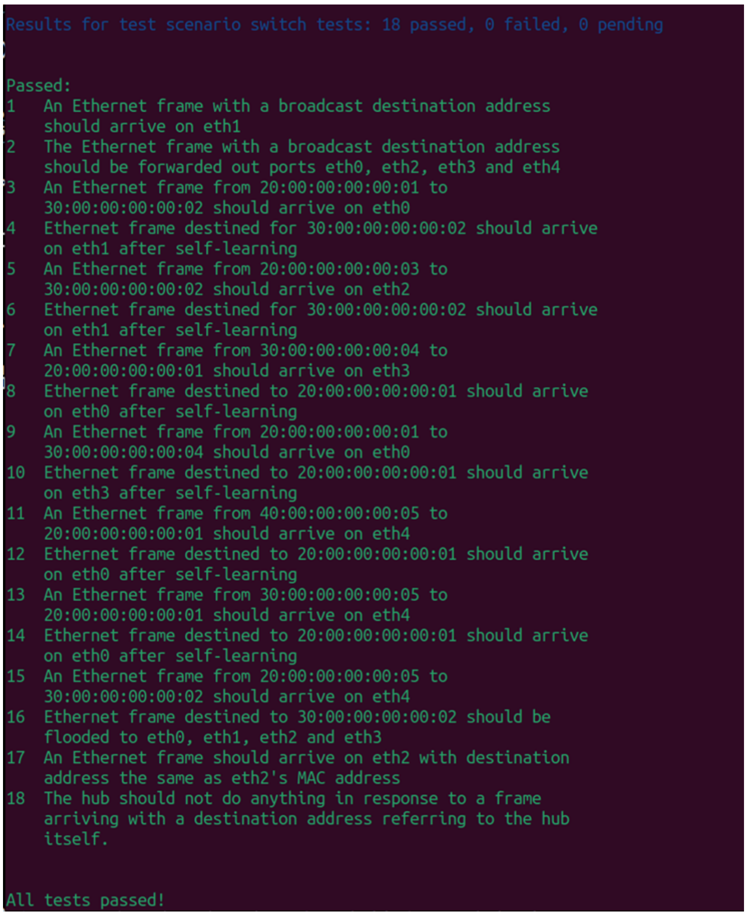


```
Results for test scenario switch tests: 9 passed, 0 failed, 0 pending


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 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 60s
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!
```

Task 4: Least Recently Used

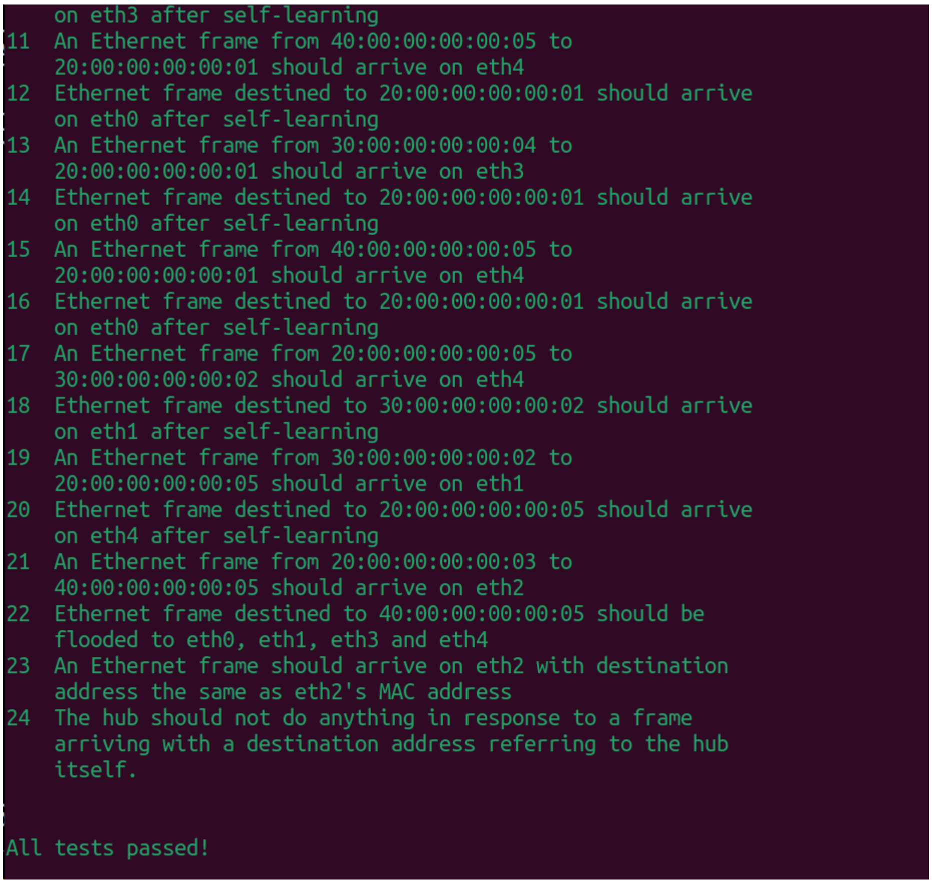


```
Results for test scenario switch tests: 18 passed, 0 failed, 0 pending


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!
```

Task 5: Least Traffic Volume



```
    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 30:00:00:00:00:04 to
    20:00:00:00:00:01 should arrive on eth3
14  Ethernet frame destined to 20:00:00:00:00:01 should arrive
    on eth0 after self-learning
15  An Ethernet frame from 40:00:00:00:00:05 to
    20:00:00:00:00:01 should arrive on eth4
16  Ethernet frame destined to 20:00:00:00:00:01 should arrive
    on eth0 after self-learning
17  An Ethernet frame from 20:00:00:00:00:05 to
    30:00:00:00:00:02 should arrive on eth4
18  Ethernet frame destined to 30:00:00:00:00:02 should arrive
    on eth1 after self-learning
19  An Ethernet frame from 30:00:00:00:00:02 to
    20:00:00:00:00:05 should arrive on eth1
20  Ethernet frame destined to 20:00:00:00:00:05 should arrive
    on eth4 after self-learning
21  An Ethernet frame from 20:00:00:00:00:03 to
    40:00:00:00:00:05 should arrive on eth2
22  Ethernet frame destined to 40:00:00:00:00:05 should be
    flooded to eth0, eth1, eth3 and eth4
23  An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
24  The hub should not do anything in response to a frame
    arriving with a destination address referring to the hub
    itself.


All tests passed!
```

五、核心代码

myswitch_to.py:

```
import switchyard
from switchyard.lib.userlib import *
from time import time

def main(net: switchyard.llnetbase.LLNetBase):
    my_interfaces = net.interfaces()
    mymacs = [intf.ethaddr for intf in my_interfaces]

    '''
        Your switch may have a table like:
        MAC Address          Interface      Timestamp
        ab:cd:ef:fe:cd:ba   interface-0    123456.123456
    '''

    mac_table = {}


    while True:
        try:
            _, fromIface, packet = net.recv_packet()
        except NoPackets:
            continue
        except Shutdown:
            break

        log_debug (f"In {net.name} received packet {packet} on {fromIface}")
        eth = packet.get_header(Ethernet)
        if eth is None:
            log_info("Received a non-Ethernet packet?!")
            return

        # record interface associated with source address of arriving packet
        mac_table[eth.src] = [fromIface, time()]

        # delete entries older than 10 seconds
        for key in list(mac_table):
            if time() - mac_table[key][1] > 10:
                del mac_table[key]

        if eth.dst in mymacs:
            log_info("Received a packet intended for me")
        else:
            #search for the output port for the destination
            if eth.dst in mac_table: # if know
                log_info(f"Sending packet on {mac_table[eth.dst][0]}")
                net.send_packet(mac_table[eth.dst][0], packet)
            else:
                #if dont know, flood
                for intf in my_interfaces:
                    if fromIface!= intf.name:
                        log_info (f"Flooding packet {packet} to {intf.name}")
                        net.send_packet(intf, packet)

    net.shutdown()
```

myswitch_lru.py:

```
import switchyard
from switchyard.lib.userlib import *
from collections import deque

def main(net: switchyard.llnetbase.LLNetBase):
    my_interfaces = net.interfaces()
    mymacs = [intf.ethaddr for intf in my_interfaces]

    mac_table = deque(maxlen=5)

    while True:
        try:
            _, fromIface, packet = net.recv_packet()
        except NoPackets:
            continue
        except Shutdown:
            break

        log_debug (f"In {net.name} received packet {packet} on {fromIface}")
        eth = packet.get_header(Ethernet)
        if eth is None:
            log_info("Received a non-Ethernet packet?!")
            return


        # record interface associated with source address of arriving packet
        for pair in mac_table:
            if pair[0] == eth.src:
                mac_table.remove(pair)
                break
        mac_table.append([eth.src, fromIface])


        if eth.dst in mymacs:
            log_info("Received a packet intended for me")
        else:
            #search for the output port for the destination
            for pair in mac_table:
                if pair[0] == eth.dst: # if know update and send
                    mac_table.remove(pair)
                    mac_table.append(pair)
                    log_info(f"Sending packet {packet} to {pair[1]}")
                    net.send_packet(pair[1], packet)
                    break

            else:   #if dont know, flood
```

```
                for intf in my_interfaces:
                    if fromIface!= intf.name:
                        log_info (f"Flooding packet {packet} to {intf.name}")
                        net.send_packet(intf, packet)

    net.shutdown()
```

myswitch_traffic.py:

```
import switchyard
from switchyard.lib.userlib import *

import heapq

def main(net: switchyard.llnetbase.LLNetBase):
    my_interfaces = net.interfaces()
    mymacs = [intf.ethaddr for intf in my_interfaces]
    max_size = 5
    mac_table = [] # [traffic, mac, interface]

    while True:
        try:
            _, fromIface, packet = net.recv_packet()
        except NoPackets:
            continue
        except Shutdown:
            break

        log_debug (f"In {net.name} received packet {packet} on {fromIface}")
        eth = packet.get_header(Ethernet)
        if eth is None:
            log_info("Received a non-Ethernet packet?!")
            return


        # record interface associated with source address of arriving packet

        for tuple in mac_table:
            if tuple[1] == eth.src:
                break
        else:
            if len(mac_table) >= max_size:
                heapq.heappop(mac_table)
            heapq.heappush(mac_table, [0, eth.src, fromIface])


        if eth.dst in mymacs:
            log_info("Received a packet intended for me")
        else:
            #search for the output port for the destination
            for tuple in mac_table:
                if tuple[1] == eth.dst:
                    tuple[0] += 1
                    log_info(f"Sending packet {packet} to {tuple[2]}")
                    net.send_packet(tuple[2], packet)
                    break

            else:   #if dont know, flood
                for intf in my_interfaces:
                    if fromIface!= intf.name:
                        log_info (f"Flooding packet {packet} to {intf.name}")
                        net.send_packet(intf, packet)

    net.shutdown()
```

六、总结与感想