

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	专业（方向）	计算机科学与技术
学号	201220062	姓名	黄子睿
Email	201220062@smail.nju.edu.cn	开始/完成日期	3/14-3/15

1. 实验名称：

Lab2 Learning Switch

2. 实验目的：

学习理解 learning switch 的功能与特点，并通过 lab 1 中提供的工具与平台，用代码实现 learning switch 的功能。

3. 实验内容：

Task 1 Basic Switch

与 hub 相比，switch 内部记录了相关的地址与端口映射信息，构成了一张转发表（forwarding table）。当有 packet 从一个端口传入时，switch 首先在转发表中查询诸条目，如果该 packet 的源地址未在表中，则记录该地址与端口，如果端口发生变化，则更新表中记录。随后 switch 将对比其目标地址与转发表中的记录，如果有匹配，则直接从对应的端口转发出去；否则广播（flood）该 packet。

引入转发表的机制后，switch 与普通的 hub 相比，在重复通信的情形下转发的效率将高于 hub，因为 switch 的转发更加有针对性，命中率也更高。

通过在 myswitch.py 中增加字典，并略微修改转发逻辑，容易用脚本代码增加转发表，从而实现 switch 的基础功能。

下面给出代码的简略分析：

首先增加一个字典 macdict 作为转发表的模拟，具体如下所示：

```
1. macdict = {} # macdict is a dictionary of mac_address:port_name
2.           # It works as a forwarding table.
```

然后修改转发逻辑，首先记录源地址的端口信息，然后在转发时先查询目标地址的端口在转发表中是否有记录，具体代码如下所示：

```
1. # Record the source address.
```

```

2. log_info (f"Receive packet {packet} from {fromIface}, record it in ma
   cdict")
3. macdict[eth.src] = fromIface

```

查表，观察是否由目标地址的端口记录，若有则定向转发，否则洪泛

```

1. # Search macdict first. If eth.dst is found, forward frame exactly; o
   therwise flood.
2. if eth.dst in macdict:
3.     for intf in my_interfaces:
4.         if intf.name == macdict[eth.dst]:
5.             log_info (f"Forwarding packet {packet} to {intf.name}")
6.             net.send_packet(intf, packet)
7.             break
8. else:
9.     for intf in my_interfaces:
10.        if fromIface!= intf.name:
11.            log_info (f"Flooding packet {packet} to {intf.name}")
12.            net.send_packet(intf, packet)

```

下面给出这一阶段的验证：

首先我编写了在 lab 1 的基础上编写了针对基础功能的 testcase，检查转发表是否正确运行，testcase 代码参见 ./mytestcases/myswitch_testscenario.py，下面给出样例简述：

设计如下网络结构，eth0-2 分别是 switch 的三个接口，对应三个地址：

```

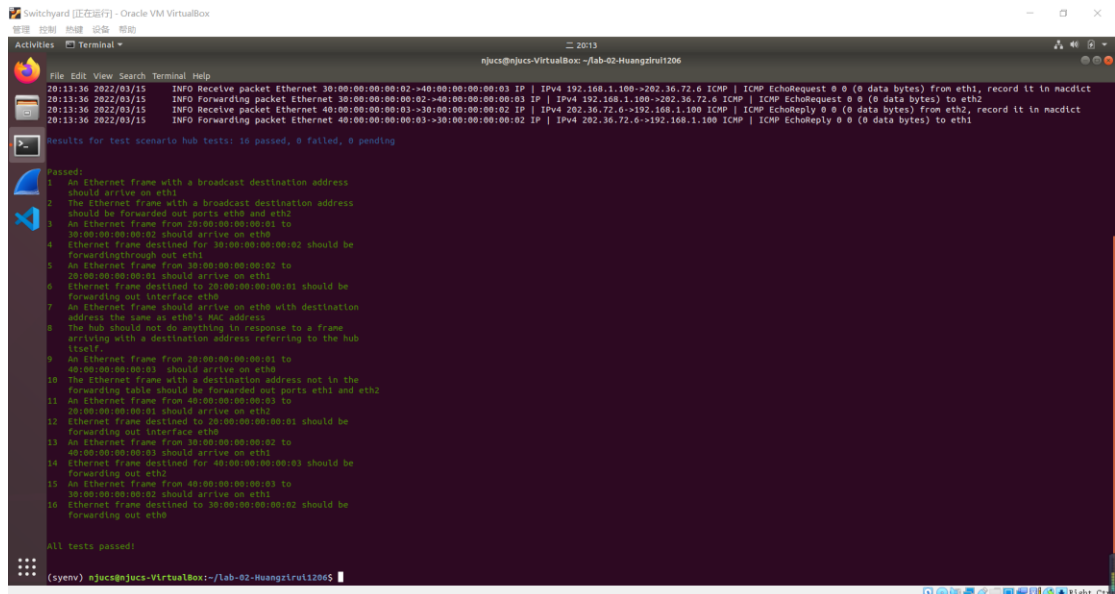
      20::~01 / 192.168.1.100
      eth0
    /      \
   /        \
eth1 -----eth2 40::~03
30::~02      202.36.72.6
172.16.42.2
...

```

1. 30: 00: 00: 00: 00: 02 广播
2. 20: 00: 00: 00: 00: 01 向 30: 00: 00: 00: 00: 02 发送一个包
3. 30: 00: 00: 00: 00: 02 发送一个地址为该 switch 的包

关键步骤：

第二步中，由于 20: ~ :01 与 30: ~ :02 的对应接口都会被记录在转发表中，因此不用广播 packet，直接将包发往目标即可。下面是运行结果：



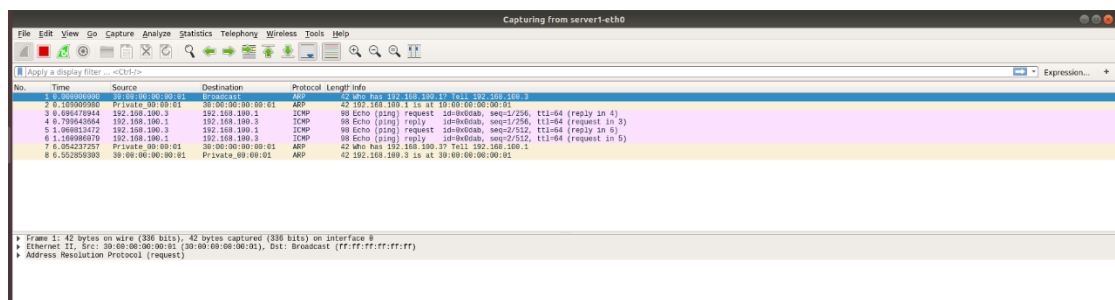
然后运行 **wireshark** 来具体观察 **switch** 的行为：

Mininet 初始化了一个简单的星形网络，server1, server2, client 三个终端通过 switch 相连。首先输入如下指令，打开实验界面：

1. mininet> xterm switch #运行 switch
2. mininet> server1 wireshark & #打开 wireshark 对 server1 与 server2 抓包
3. mininet> server2 wireshark &
4. mininet> xterm client #打开 client 界面

然后在 client 的窗口下，输入 `ping -c 2 192.168.100.1`，由 client 发送一个地址为 server 的 ICMP 包，观察 server1 与 server2 的 wireshark 运行结果：

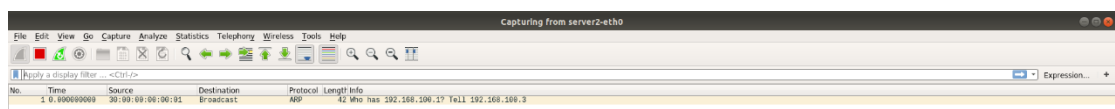
下面是 server1 的 wireshark 窗口：



对于这个传输结果，下面进行一下分析：

首先，client(mac:30:00:00:00:00:1/ip:192.168.100.3)发送了一个 ARP 包进行广播，用以询问地址。得到了 server1 (mac:10:00:00:00:00:01) 的相应。随后两者互相转发了两个 ICMP 包（对应指令中的 ping -c 2），最后再由 server1 应答 client 的 ARP 包结束这次转发过程。

而下面是 server2 的界面：



可以看见，在第一次的广播之后，server2 就没有收到任何消息了，因为这时候 switch

已经记录了 server1 与 client 的地址与接口，不再进行广播。

Task 2: Timeouts

在 step 1 中实现的 learning switch 默认拥有近乎无穷大的转发表。这在实际生活中是不可能存在的。因此当转发表满时，再向其中加入一条新的记录，就需要考虑替换算法。这里增加计时功能，如果一条记录一段时间没有转发查询记录，则将它从表中删去。代码中默认时间设置为 10s。

代码实现简述：

利用 python 自带的<time>库文件，引用其中的 time.time()记录每条记录如表的时间戳，在有一个新的包需要转发时，首先更新此时的转发表，即比较当前时间戳与记录时间戳的差值，大于十秒则将记录移除转发表。

```
1. import time #首先引入 python 中的<time>库
2. ... ..
3. # Check whether forwarding table entries are out of time
4. cur_time = time.time()
5. for addr in list(macdict):
6.     if cur_time - macdict[addr][1] >= 10:
7.         log_info(f"The entry \"{addr}:{macdict[addr]}\" is evicted At
           time {cur_time}")
8.         del macdict[addr]
```

在 ./mytests/myswitch_to_testscenario.py 中，我为这一阶段设计了相关测试样例。

具体样例简述：

网络具体形式与 Task 1 中结构相同，前两个测试用例也相同，从第三步开始：

3. 停止六秒钟

4. 由 40: 00: 00: 00: 00: 03 从 eth2 向 20: 00: 00: 00: 00: 01 发送一个包，此时 40: ~ : 03 刚刚加入表中，时间为 0，20: ~ : 01 此时被更新为 0，而 30: ~ : 02 的时间记录则为 6 秒。

5. 停止六分钟

6. 发送一个 40: ~ : 03 到 30: ~ : 02 的包，此时 30: ~ : 02 已经超时出表，因此将会出现广播的情况。

下面两张分别是通过自己设计的测试用例的截图与通过实验自带测试用例的截图：

```
Switchyard [正在运行] - Oracle VM VirtualBox
Activities Terminal
njlucs@njlucs-VirtualBox: ~/lab-02-Huanggrui12065

20:49:38 2022/03/15 INFO Receive packet Ethernet 20:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 192.168.1.100->202.36.72.6 ICMP | ICMP EchoReply 0 0 (0 data bytes) from eth0, record it in macdict
20:49:38 2022/03/15 INFO Forwarding packet Ethernet 20:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 192.168.1.100->202.36.72.6 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth2
20:49:44 2022/03/15 INFO Receive packet Ethernet 40:00:00:00:00:03->30:00:00:00:00:02 IP | IPv4 202.36.72.6->192.168.1.100 ICMP | ICMP EchoRequest 0 0 (0 data bytes) from eth2, record it in macdict
20:49:44 2022/03/15 INFO Flooding packet Ethernet 40:00:00:00:00:03->30:00:00:00:00:02 IP | IPv4 202.36.72.6->192.168.1.100 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
20:49:44 2022/03/15 INFO Flooding packet Ethernet 40:00:00:00:00:03->30:00:00:00:00:02 IP | IPv4 202.36.72.6->192.168.1.100 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
20:49:44 2022/03/15 INFO Receive packet Ethernet 30:00:00:00:00:02->40:00:00:00:00:03 IP | IPv4 192.168.1.100->202.36.72.6 ICMP | ICMP EchoReply 0 0 (0 data bytes) from eth1, record it in macdict
20:49:44 2022/03/15 INFO Forwarding packet Ethernet 30:00:00:00:00:02->40:00:00:00:00:03 IP | IPv4 192.168.1.100->202.36.72.6 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth2

Results for test scenario hub tests: 16 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 be forwarding through out eth1
5 An Ethernet frame from 30:00:00:00:00:02 to 20:00:00:00:00:01 should arrive on eth1
6 Ethernet frame destined to 20:00:00:00:00:01 should be forwarding out interface eth0
7 Sleep for 6 seconds.
8 An Ethernet frame from 40:00:00:00:00:03 to 30:00:00:00:00:01 should arrive on eth2
9 Ethernet frame destined to 20:00:00:00:00:01 should be forwarding out interface eth0
10 An Ethernet frame from 20:00:00:00:00:01 to 40:00:00:00:00:03 should arrive on eth0
11 Ethernet frame destined to 40:00:00:00:00:03 should be forwarding out interface eth2
12 Sleep for 6 seconds.
13 An Ethernet frame from 40:00:00:00:00:03 to 30:00:00:00:00:02 should arrive on eth2
14 Ethernet frame destined for 30:00:00:00:00:02 should be flooding out eth0 and eth1
15 An Ethernet frame from 30:00:00:00:00:02 to 40:00:00:00:00:03 should arrive on eth1
16 Ethernet frame destined to 40:00:00:00:00:03 should be forwarding out interface eth0

All tests passed!
(syew) njlucs@njlucs-VirtualBox:~/lab-02-Huanggrui12065
```

(通过自己设计的测试用例)

```
Switchyard [正在运行] - Oracle VM VirtualBox
Activities Terminal
njlucs@njlucs-VirtualBox: ~/lab-02-Huanggrui12065

10 Ethernet frame destined to 40:00:00:00:00:03 should be forwarding out interface eth0

All tests passed!

(syew) njlucs@njlucs-VirtualBox:~/lab-02-Huanggrui12065$ suyard -t testcases/myswitch to testscenario.srpy myschitch_to.py
20:51:48 2022/03/15 INFO Starting test scenario testcases/myswitch to testscenario.srpy
20:51:48 2022/03/15 INFO Receive 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) from eth1, record it in macdict
20:51:48 2022/03/15 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
20:51:48 2022/03/15 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
20:51:48 2022/03/15 INFO Receive 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) from eth0, record it in macdict
20:51:48 2022/03/15 INFO Forwarding 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
20:52:08 2022/03/15 INFO Receive 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) from eth0, record it in macdict
20:52:08 2022/03/15 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
20:52:08 2022/03/15 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
20:52:08 2022/03/15 INFO Receive 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) from eth2, record it in macdict
20:52:08 2022/03/15 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!
(syew) njlucs@njlucs-VirtualBox:~/lab-02-Huanggrui12065
```

(通过助教设计的测试用例)

然后打开 mininet 进行分析验证:

简述一下实验思路:

首先通过 client ping server1, 将在转发表中记录 client 与 server1 的信息。等待此时再用 server2 ping server1, 将不会再 client 中看到相关信息, 而 server2 也加入表中。等待一段时间 (至少 10 秒), 再向发送目标为 server1 的包, 理应会出现洪泛的情况, 因为 server1 已经出表了, 需要洪泛以查找 server1。但是最后洪泛的过程在 wireshark 中无法直接观察到, 因为 ping 会在发送 ICMP 包之前发送一个广播的 ARP 包, 询问目标地址的方位, 等到目标信息进行答复之后, 目标地址对应的接口已经重新记录在表中。为了解决这一问题, 我选择在代码中增加

下面是具体实验过程:

1.打开 switch, client, server2 的 xterm 转发终端, 并在 switch 窗口中运行 myswitch_to.py, 再打开 wireshark 观察 client, server1 与 server2:

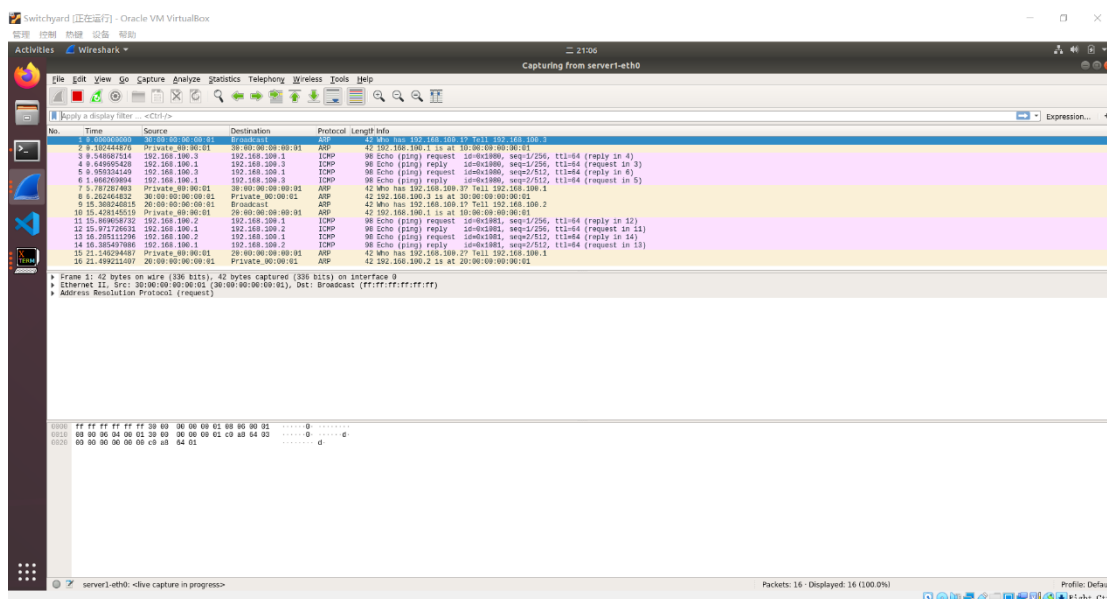
2.在 client 中输入 ping -c 2 192.168.100.1,运行结果同 Task 1, 这里不再附图。

3.在 server2 中输入 ping -c 192.168.100.1, 运行结果如下:

Server1 wireshark 截图:

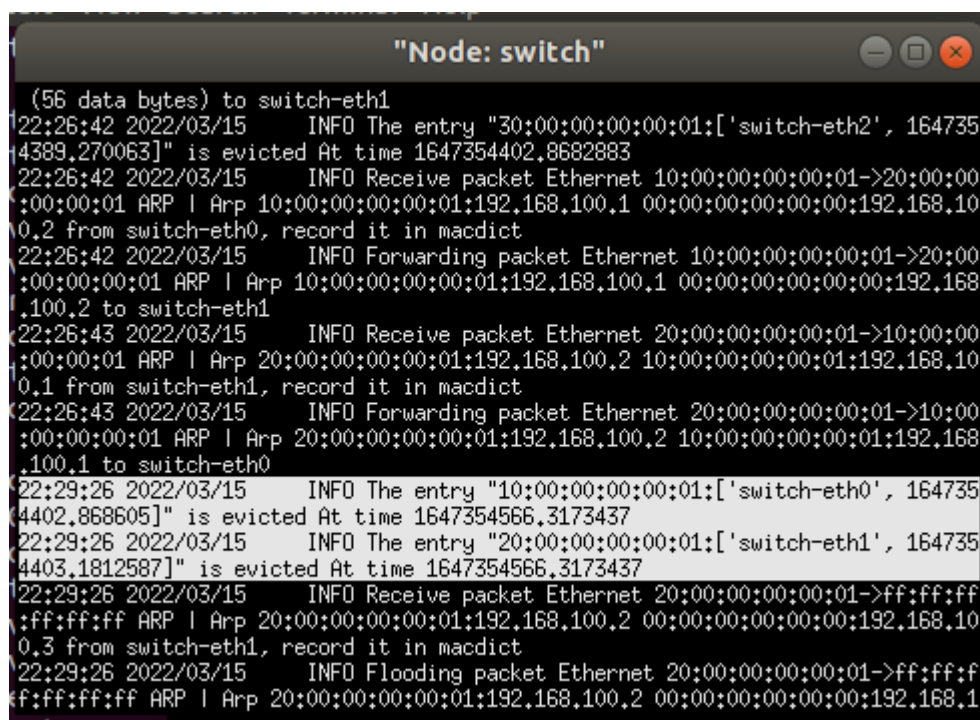
可以看到 server1 有两次转发应答经历, 分别是与 client 与 server2。

同时也可以看到, 每一次 ping 在发送 ICMP 文件之前首先会有一个 ARP 包的广播问询。对应地址应答之后, ping 双方的地址与接口都已经被记录了。因此即使某个端口被移除转发表, 在广播问询之后这个端口就会被重新记录下来, 因此通过 log_out 的方式查看某个记录是否被移除。



可以证明这个时候 server1,server2, client 三者的地址与接口都是在转发表中的。

4. 经过一段时间 (超过十秒) 后, 再 server2 ping client, 程序将首先检查有无超时的记录, 若有将其删除。在 xterm switch 窗口中观察到有记录被移除转发表, 如下图所示:



由此证明，myswitch_to.py 正确运行。（注意最后选中的记录，表示 server1 与 server2 因为超时被移出转发表）

Task 3: Least Recently Used

还是以 Task 1 中的 learning switch 为范本，这个任务中需要以 LRU 的规则从转发表中移除最近一段时间里最不经常用的记录。为了实现这一功能，需要在基础版本的功能上改进 macdict（即代码中的转发表）的实现，通过构造包含一个字典与一个队列的类，实现了 $O(1)$ 时间复杂度， $O(n)$ 空间复杂度的 LRU 算法。

下面首先介绍一下 LRU 在代码中的实现：

```
1. class LRUCache:
2.     def __init__(self, size=5):
3.         self.size = size
4.         self.dict = {}
5.         self.list = []
6.
7.     def set(self, key, value):
8.         if key in self.dict:
9.             if self.dict[key] != value:
10.                 self.list.remove(key)
11.             else : #If the entry is in the dict, do nothing.
12.                 return
13.         elif len(self.dict) == self.size:
14.             lru_key = self.list.pop()
15.             # For deploying
16.             log_info(f"According to the LRU rule, {lru_key} is evicted out forwarding table.")
17.             self.dict.pop(lru_key)
18.             self.list.insert(0, key)
19.             self.dict[key] = value
20.
21.     def get(self, key):
22.         if key in self.dict:
23.             self.list.remove(key)
24.             self.list.insert(0, key)
25.             return self.dict[key]
26.         else:
27.             return None
```

1. 通过引入 list 作为转发表 LRU 优先度的记录，实现了 $O(1)$ 时间复杂度的 LRU 算法实现

编写关于这一任务的测试用例（具体参见./mytestcases/myswitch_lru_testscenario.py），需要改进之前的网络模型，使之更为复杂。具体形式如下文注释所示

```
1.     ....
```


2.	mac 地址	端口	代号
3.	20: ~ :01	eth0	a
4.	30: ~ :02	eth1	b
5.	20: ~ :03	eth2	c
6.	30: ~ :04	eth3	d
7.	40: ~ :05	eth4	e
8.	20: ~ :05	eth0	f
9.	'''		

然后设计如下文注释展示的实验过程：

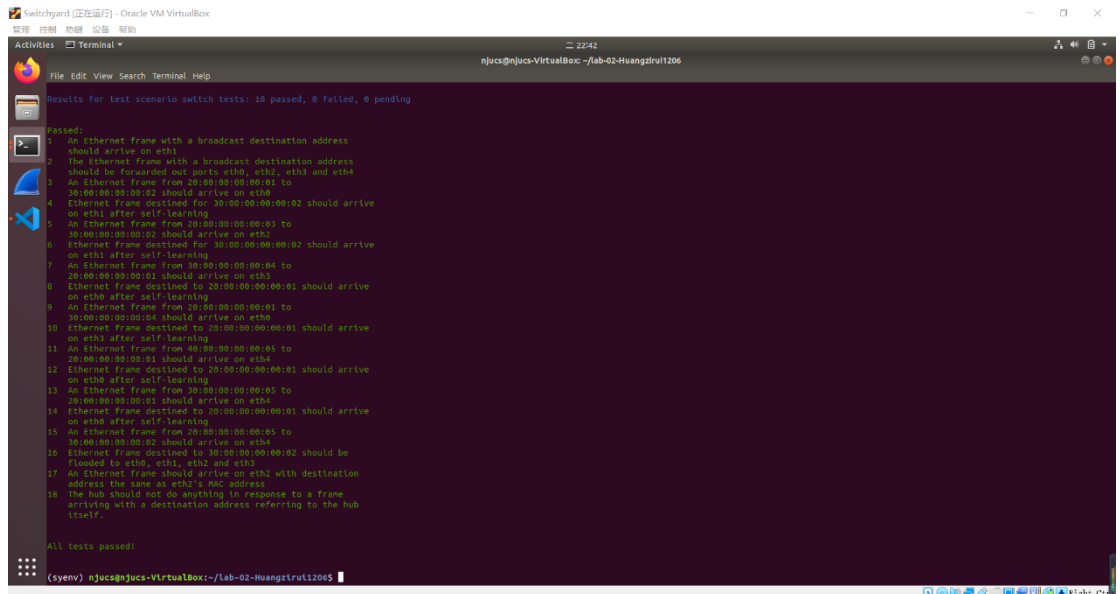
1. '''
2. 括号中表示当前操作结束之后，LRUCache 中 list 元素的先后顺序，即各个地址在转发表中的优先级
3. 1. b broadcasts; (b)
4. 2. a to b (b,a)
5. b to a (a,b)
6. 3. c to b (b,c,a)
7. b to c (c,b,a)
8. 4. c to d (c,b,a) *flood
9. d to c (c,d,b,a)
10. 5. e to b (b,e,c,d,a)
11. b to e (e,b,c,d,a)
12. 6. f to e (e,f,b,c,d)
13. e to f (f,e,b,c,d)
14. 7. b to a (b,f,e,c,d)*a is out of table, flood.
15. a to b (b,a,f,e,c)
16. 8. f to d (f,b,a,e.c)*d is out of table, flood.
17. d to f (f,d,b,a,e)
18. '''

主要观察 7, 8 步中，由于 a,d 当时应当已经移除出转发表，因此可以观察到洪泛。
下面是通过自己编写的用例与助教用例的截图：

```

Switchyard [正在运行] - Oracle VM VirtualBox
nJucs@nJucs-VirtualBox: /lab-02-Huangziru1206
File Edit View Search Terminal Help
12 00:00:00:00:00:04 should arrive on eth2
13 The Ethernet frame with address 30:00:00:00:00:04 is not in
the table, and should be forwarded out ports eth0, eth1,
eth3 and eth4
14 An Ethernet frame from 30:00:00:00:00:04 to
20:00:00:00:00:01 should arrive on eth1
15 Ethernet frame destined to 30:00:00:00:00:03 should be
forwarding out interface eth2
16 An Ethernet frame from 40:00:00:00:00:05 to
30:00:00:00:00:02 should arrive on eth4
17 Ethernet frame destined to 20:00:00:00:00:02 should be
forwarding out interface eth1
18 An Ethernet frame from 30:00:00:00:00:02 to
40:00:00:00:00:05 should arrive on eth0
19 Ethernet frame destined to 40:00:00:00:00:05 should be
forwarding out interface eth4
20 An Ethernet frame from 40:00:00:00:00:05 to
20:00:00:00:00:05 should arrive on eth4
21 Ethernet frame destined to 20:00:00:00:00:05 should be
forwarding out interface eth0
22 An Ethernet frame from 30:00:00:00:00:02 to
20:00:00:00:00:01 should arrive on eth1
23 The Ethernet frame with address 20:00:00:00:00:01 is not in
the table, and should be forwarded out ports eth0, eth2,
eth3 and eth4
24 An Ethernet frame from 20:00:00:00:00:01 to
30:00:00:00:00:02 should arrive on eth0
25 Ethernet frame destined to 30:00:00:00:00:02 should be
forwarding out interface eth1
26 An Ethernet frame from 20:00:00:00:00:05 to
20:00:00:00:00:01 should arrive on eth0
27 The Ethernet frame with address 30:00:00:00:00:04 is not in
the table, and should be forwarded out ports eth1, eth2,
eth3 and eth4
28 An Ethernet frame from 30:00:00:00:00:04 to
20:00:00:00:00:05 should arrive on eth3
29 Ethernet frame destined to 20:00:00:00:00:05 should be
forwarding out interface eth0
All tests passed!
(syenv) nJucs@nJucs-VirtualBox: /lab-02-Huangziru1206

```

通过自己的用例与助教的水例

然后在 mininet 中验证代码的运行。

首先修改 start_mininet.py 的代码，使得网络结构更加复杂。修改后，mininet 中的终端数量增加到 6 个，都通过 switch 连接，具体参考下面的代码：

```
1. # After my revision, the net topology has 6 hosts
2. # So it will become a 6-
   host star topology to test some complicated situations.
3.
4. nodeconfig = {'cpu':-1}
5. self.addHost('server1', **nodeconfig)
6. self.addHost('server2', **nodeconfig)
7. #add more servers
8. self.addHost('server3',**nodeconfig)
9. self.addHost('server4',**nodeconfig)
10. self.addHost('server5',**nodeconfig)
11. self.addHost('switch', **nodeconfig)
12. self.addHost('client', **nodeconfig)
13.
14. for node in ['server1','server2','client','server3','server4',
               'server5']:
15.     # all links are 10Mb/s, 100 millisecond prop delay
16.     self.addLink(node, 'switch', bw=10, delay='100ms')
```

然后采取以下操作：

Server1 ping server 2, server2 ping server3, server3 ping server 4, server4 ping server5, 这时候应当转发表已经满了（默认大小是 5），再由 server5 ping client 则应当将 server1 的信息从转发表中剔除。

下面是具体的运行过程：

```

*** Starting CLI:
mininet> xterm switch
mininet> server1 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=992 ms

--- 192.168.100.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 992.895/992.895/992.895/0.000 ms
mininet> server2 ping -c 1 server3
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=898 ms

--- 192.168.100.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 898.596/898.596/898.596/0.000 ms
mininet> server3 ping -c 1 server4
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=1059 ms

--- 192.168.100.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1059.527/1059.527/1059.527/0.000 ms
mininet> server4 ping -c 1 server5
PING 192.168.100.6 (192.168.100.6) 56(84) bytes of data.
64 bytes from 192.168.100.6: icmp_seq=1 ttl=64 time=931 ms

--- 192.168.100.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 931.252/931.252/931.252/0.000 ms
mininet> server5 ping -c 1 client
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=1033 ms

--- 192.168.100.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1033.781/1033.781/1033.781/0.000 ms
mininet>

```

```

Node: switch
23:07:16 2022/03/15      INFO Flooding packet Ethernet 70:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:00:00:01:192.168.100.2 00:00:00:00:00:00:192.168.100.2 to switch-eth1
23:07:16 2022/03/15      INFO Flooding packet Ethernet 70:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:00:00:01:192.168.100.4 00:00:00:00:00:00:192.168.100.4 to switch-eth4
23:07:16 2022/03/15      INFO Flooding packet Ethernet 70:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:00:00:01:192.168.100.5 00:00:00:00:00:00:192.168.100.5 to switch-eth2
23:07:16 2022/03/15      INFO Flooding packet Ethernet 70:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:00:00:01:192.168.100.6 00:00:00:00:00:00:192.168.100.6 to switch-eth0
23:07:16 2022/03/15      INFO Flooding packet Ethernet 70:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.3 to switch-eth3
23:07:16 2022/03/15      INFO Receive packet Ethernet 30:00:00:00:01->70:00:00:00:01 ARP | Arp 30:00:00:00:01:192.168.100.3 70:00:00:00:01:192.168.100.6 from switch-eth2, record it in macdict
23:07:16 2022/03/15      INFO According to the LRU rule, 10:00:00:00:01 is evicted out forwarding table.
23:07:16 2022/03/15      INFO Forwarding packet Ethernet 30:00:00:00:01->70:00:00:00:01 ARP | Arp 30:00:00:00:01:192.168.100.3 70:00:00:00:01:192.168.100.6 to switch-eth5
23:07:16 2022/03/15      INFO Receive packet Ethernet 70:00:00:00:01->30:00:00:00:01

```

注意选中的记录，可见确实 server1 的记录被移除了，说明代码运行正确。

Task 4: Least Traffic Volume

再这一任务中，替换规则更改为流量大小的比较，在最近一段时间中访问次数最少的记录将会被移出转发表。仍然采用构造类的方式，设计 LTVCache 以实现这一算法功能，具体代码分析见下：

```
1. class LTVCache:
2.     def __init__(self, size = 5):
3.         self.ndict = {} #ndict 字典记录 address:port_name
4.         self.vdict = {} #vdict 字典记录 address:traffic_volume
5.         self.size = size
6.
7.     def set(self, key, value):
8.         if key in self.ndict:
9.             if self.ndict[key] != value: #如果特定地址的信息发生改变，
                重置其 traffic volume
10.                self.ndict[key] = value
11.                self.vdict[key] = 0
12.         else:
13.             if len(self.ndict) >= self.size:
14.                 min_key = min(self.vdict, key = lambda k:self.vdict[k
                    ]) #调用排序算法，找出 traffic_volume 最小的记录，并将其删除
15.                 log_info(f"According to the LTV rule, {min_key} is ev
                    icted out forwarding table.")
16.                 self.ndict.pop(min_key)
17.                 self.vdict.pop(min_key)
18.                 self.ndict[key] = value
19.                 self.vdict[key] = 0
20.
21.     def get(self, key):
22.         if key in self.ndict:
23.             self.vdict[key] += 1 #增加记录的 traffic_volume
24.             return self.ndict[key]
25.         else:
26.             return None
```

然后设计这一任务对应的测试环境，使用 Task 3 构建的网络拓扑结构，并在转发顺序上略加调整，得到这一任务的测试样例，具体见下注释（具体代码参见./mytestcases/myswitch_traffic_testscenario.py）：

```
1. ....
2. 括号中是记录对应地址与其对应的访问次数，其顺序对应此时 LTVCache 中的优先级顺序
3. 1. b broadcasts;    (b0)
4. 2. a to b
5.    b to a
6.    (a1,b1)
```

```

7. 3. c to b
8. b to c
9. (b2,a1,c1)
10. 4. c to d
11. d to c
12. (b2,c2,a1,d0)
13. 5. e to b
14. b to e
15. (b3,c2,a1,e1,d0)
16. 6. b to a
17. a to b
18. (b4,a2,c2,e1,d0)
19. 7. f to b
20. b to f
21. (evict d)
22. (b5,a2,c2,e1,f1)
23. 8. a to d
24. d to a * d is out of table, flood
25. (b5,a3,c2,e1,d0)
26. '''

```

主要需要关注第 8 步对应的测试用例，因为此时 d 按道理应当被移除出转发表，因此会出现洪泛的情况。

下面是通过自己设计的测试用例的截图：

```

Switchyard [正在运行] - Oracle VM VirtualBox
Activities Terminal njucs@njucs-VirtualBox - /lab-02-Huangtrui1206
File Edit View Search Terminal Help
10:00:00:00:00:04 should arrive on eth2
12 The Ethernet frame with address 10:00:00:00:00:04 is not in
the table, and should be forwarded out ports eth0, eth1,
eth3 and eth4
13 An Ethernet frame from 10:00:00:00:00:04 to
20:00:00:00:00:03 should arrive on eth3
14 Ethernet frame destined to 10:00:00:00:00:03 should be
forwarding out interface eth2
15 An Ethernet frame from 10:00:00:00:00:05 to
10:00:00:00:00:02 should arrive on eth4
16 Ethernet frame destined to 10:00:00:00:00:02 should be
forwarding out interface eth1
17 An Ethernet frame from 10:00:00:00:00:02 to
10:00:00:00:00:01 should arrive on eth1
18 Ethernet frame destined to 10:00:00:00:00:03 should be
forwarding out interface eth4
19 An Ethernet frame from 10:00:00:00:00:02 to
20:00:00:00:00:01 should arrive on eth1
20 The Ethernet frame with address 20:00:00:00:00:02 should be
forwarded out ports eth0
21 An Ethernet frame from 20:00:00:00:00:01 to
10:00:00:00:00:02 should arrive on eth0
22 Ethernet frame destined to 10:00:00:00:00:02 should be
forwarding out interface eth1
23 An Ethernet frame from 10:00:00:00:00:02 to
20:00:00:00:00:03 should arrive on eth3
24 Ethernet frame destined to 20:00:00:00:00:05 has no entry in
table, and should be forwarding out interface eth0, eth2,
eth3, eth4
25 An Ethernet frame from 20:00:00:00:00:05 to
10:00:00:00:00:02 should arrive on eth1
26 Ethernet frame destined to 10:00:00:00:00:02 should be
forwarding out interface eth0
27 An Ethernet frame from 20:00:00:00:00:01 to
10:00:00:00:00:04 should arrive on eth0
28 Ethernet frame destined to 10:00:00:00:00:02 has no entry in
the table and should be forwarding out interface eth1, eth3,
eth4, eth0
29 An Ethernet frame from 10:00:00:00:00:04 to
20:00:00:00:00:03 should arrive on eth3
30 Ethernet frame destined to 10:00:00:00:00:01 should be
forwarding out interface eth0
All tests passed!
(syenv) njucs@njucs-VirtualBox:~/lab-02-Huangtrui1206

```

然后需要在 mininet 中测试代码的运行情况：

就按照上面设计的 testcase，令 server1-5 对应 a-e，client 对应 f，则在最后一次 server1 ping server4 时会看到 client 被移除出转发表。(略微调整，将 b broadcast 修改为 server2 ping server1)。

下面是具体的测试结果截图：

按照上述流程执行后，得到如下结果，注意到选中的部分，可以看到 server4 在第

7 步的时候就已经被移除了。

```
23:27:36 2022/03/15 INFO Forwarding packet Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 ARP | Arp 20:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.100.1 to switch-eth0
23:27:42 2022/03/15 INFO Receive packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 from switch-eth2, record it in macdict
23:27:42 2022/03/15 INFO According to the LRU rule, 50:00:00:00:00:01 is evicted out forwarding table.
23:27:42 2022/03/15 INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 to switch-eth0
23:27:42 2022/03/15 INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 to switch-eth1
23:27:42 2022/03/15 INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 to switch-eth3
23:27:42 2022/03/15 INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 to switch-eth5
23:27:42 2022/03/15 INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.100.2 to switch-eth4
23:27:42 2022/03/15 INFO Receive packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:00
```

由此可见，相应的代码实现是正确的。

到此，Lab 2 所有内容就完成了。