实验报告

计算机网络-Lab2

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的

模拟交换机，实现其基本功能，timeout 的性质，以及 LRU、Least Traffic Volume 替换算法。

二、 实验内容

1. Preparation

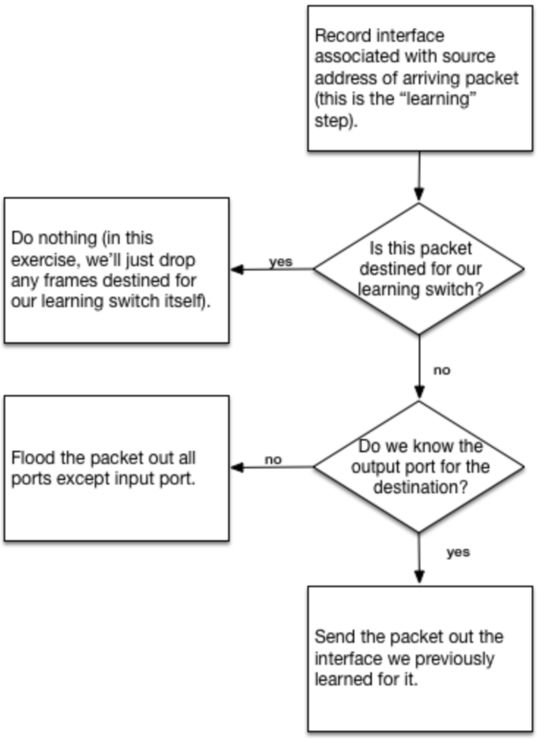
我们已经非常熟悉的 git clone、launch.json、venv……当然，手册还是得看看的。

Finally, your project will look like:

```
.
├── myswitch.py
├── myswitch_lru.py
├── myswitch_to.py
├── myswitch_traffic.py
├── start_mininet.py
├── ...
└── testcases
    ├── myswitch_lru_testscenario.srpy
    ├── myswitch_to_testscenario.srpy
    ├── myswitch_traffic_testscenario.srpy
    └── test_submit.py
```

2. Basic Switch

这一步，我们要实现基本的自学习功能。当一个以太网帧到达交换机，它应遵循以下的流程图（不介意我盗搬个图吧）。



转发表采用字典实现。

```
# tab saving the mac address and the interface name
mem = {}
```

当有包传过来，不管三七二十一，我们更新一下表。

```
# any time we receive a packet, we update the table
mem[packet[Ethernet].src] = fromIface
```

发出时，如果找到对应端口，则从该端口发出；否则直接广播。

```
elif eth.dst in mem:
    log_info(f"Send packet {packet} to {mem[eth.dst]}")
    net.send_packet(mem[eth.dst], packet)
else:
    # broadcast
    for intf in my_interfaces:
        if fromIface!= intf.name:
            log_info (f"Flooding packet {packet} to {intf.name}")
            net.send_packet(intf, packet)
```

我们按照手册中的教程做个测试。

To examine whether your switch is behaving correctly, you can do the following:

1. Open terminals on client, server1 and server2 (`xterm client`, `xterm server1` and `xterm server2` from the Mininet prompt)
2. In the server1 and server2 terminal, run `wireshark`. Wireshark is a program that allows you to "snoop" on network traffic arriving on a network interface. We'll use this to verify that we see packets arriving at server1 and server2 from client.
3. In the terminal on the client node, type `ping -c 2 192.168.100.1`. This command will send two "echo" requests to the server1 node. The server1 node should respond to each of them if your switch is working correctly. You should see at the two echo request and echo replies in Wireshark running on server1, and you will probably see a couple other packets (e.g., ARP, or Address Resolution Protocol, packets).
4. If you run Wireshark on server2, you should **not** see the echo request and reply packets (but you will see the ARP packets, since they are sent with broadcast destination addresses).

server1和server2均收到了ARP广播。这是client用于确认对方的MAC，因为dst为server1，所以server2再无后续反应。

No.	Time	Source	Destination
1	0.000000000	30:00:00:00:00:01	Broadcast

No.	Time	Source	Destination	Protocol	Le
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	
2	0.101438572	Private_00:00:01	30:00:00:00:00:01	ARP	

后面就是client连续发送的俩数据报，每次发送server1均作了回应。

3	0.521109412	192.168.100.3	192.168.100.1	ICMP
4	0.623066283	192.168.100.1	192.168.100.3	ICMP
5	1.044422363	192.168.100.3	192.168.100.1	ICMP
6	1.144881402	192.168.100.1	192.168.100.3	ICMP

关注switch。一开始，我们的交换机的转发表空空如也，然而，在client广播后，它得知了client(192.168.100.3)的存在，并把它的MAC和端口eth2联系起来。

```
16:12:00 2022/03/10     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
16:12:00 2022/03/10     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
```

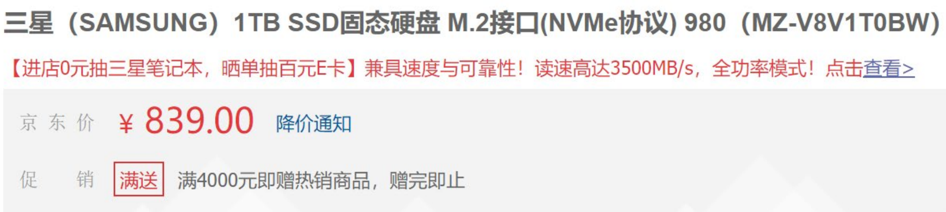
此后，当server1回应时，它便可以定向发送。

```
16:12:00 2022/03/10     INFO Send 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
```

其他的发送记录我们便不再一一阐述了。

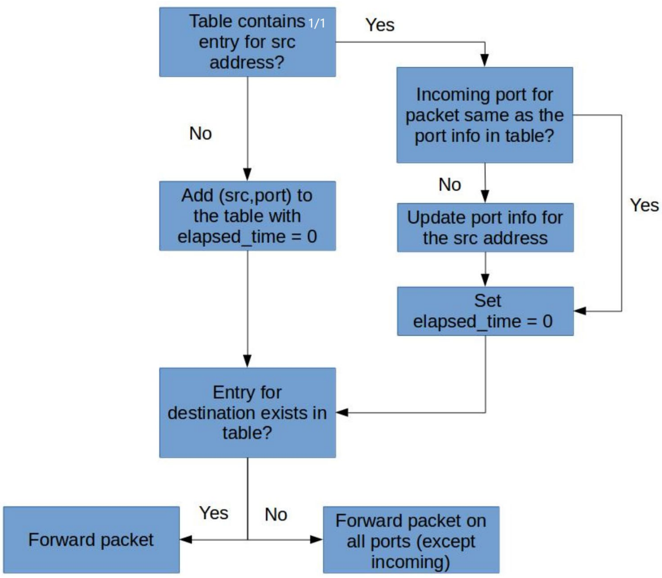
3. Timeouts

众所周知，交换机的容量是有限的，往里边不断地插 980 显然也不是啥经济的事儿。



所以，我们得把一些“衰老”（一段时间内，我们没有收到该 MAC 发出的数据）的记录删除，为年轻的新鲜血液让路。

照例，上流程图。



这一次，我们的表中还需要记这条表项生成的时间（1970 纪元后经过的浮点秒数）。

```
# any time we receive a packet, we update the table
mem[packet[Ethernet].src] = (fromIface, time)
```

为了避免频繁地遍历转发表，采用延迟删除，即，我们在收到消息时，更新整个转发表（而不是在每一秒都检查一次）。

```python
def update_mem() -> None:
    ''' timeout case '''
    nonlocal mem
    keys = list(mem.keys())
    for key in keys:
        if time.time() - mem[key][1] > 10:
            del mem[key]
```

编写自己的 testcase。我们的 testcase 流程是这样的：

- 从 eth1 发送广播，使交换机记住 eth1 对应 MAC；
- 从其他某节点向 eth1 发送，此时，帧只应从 eth1 发出；
- 10s 后，重复此行为，帧应被广播出去。

具体的代码就不在这展示了，只贴一下测试结果：

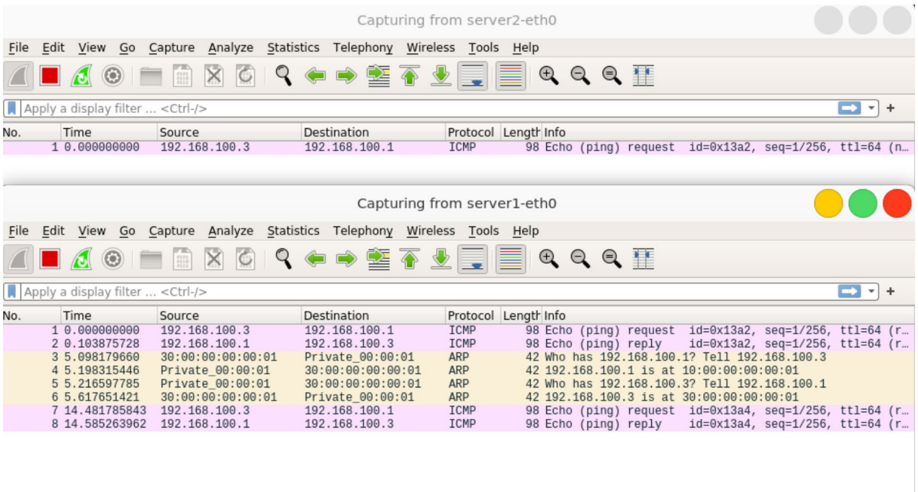
```
7   Ethernet frame destined for 30:00:00:00:00:02 should
    arriveon eth1 and eth2 after 11 seconds
        Expected event: send_packet(s) Ethernet
        20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4
        192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0
        data bytes) out eth1 and 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) out eth2


All tests passed!
```

接着，我们用课程官方测试用例的通过来添上一笔！（吐槽一下，手册说好的 10s，你让我等 20s，是不是不太厚道？）

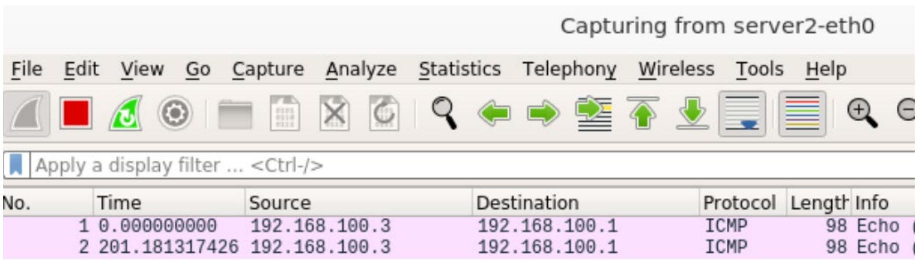
```
5   Timeout for 20s
6   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive on eth0
7   Ethernet frame destined for 30:00:00:00:00:02 should be
    flooded out eth1 and eth2
8   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC 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!
```

实战测试中，我们先让 client 向 server1 发送一个包，短时间内重复一次（在这之前，已经 pingall，各个主机都存有其他主机的 MAC）。

可以看到，server2 仅收到第一次的包。原因在于，第一次，转发表为空，交换机做了广播。

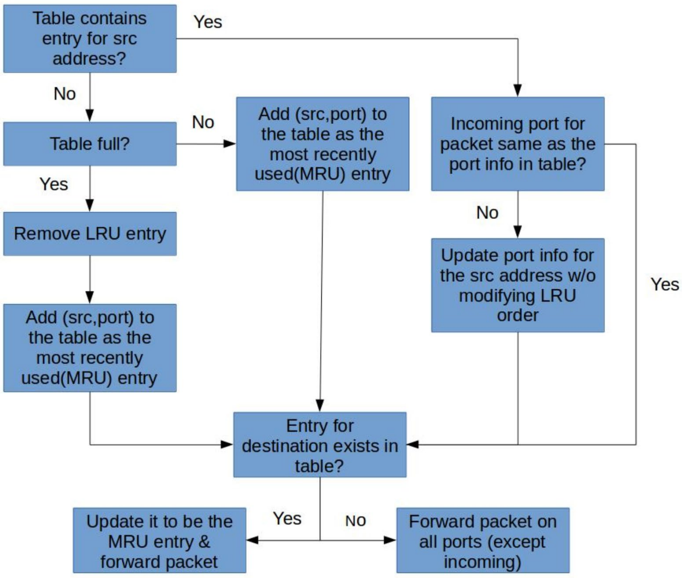


等待一会儿（大于老化期），我们重复一次。关注 server2，可以推断，交换机忘记了过期的记录，再次广播。



4. Least Recently Used

国际惯例，先上流程图。



LRU 算法，从实现上说，需要我们为每个表项设置一个 LRU 位（当然，不止 1 位），我们对其不断更新；这对于硬件是易于实现且高效的，但众所周知，Python 是软的，而且还不是一般的慢，这就要求我们从算法上做点优化。

> ⓘ You may notice that it is not efficient to update the age of all other rules every time you use a table rule. Because the time complexity is O(N) for each table query operation, where N is the number of rules in the table. Besides, when you need to evict a table entry, you have to iterate all the table rules to find the item to be evicted. Therefore, more efficient implementation of LRU algorithm is encouraged in this lab.

那我们从 LRU 本质入手，不妨设置一个队列，操作如下：

- 最新访问的，我们挪到队列尾；
- 队列满时，我们删去队列头（最长时间不访问）；
- 这样，只有查找这一动作的时间复杂度会达到 O(N)，其余都可以在 O(1)内完成。

关键代码如下：

表项的更新

```python
def update_mem(mac_addr, new_port) -> None:
    nonlocal mem, mem_capacity
    for addr, port in mem:
        # if the mac address is already in the table, update the port
        if addr == mac_addr:
            mem.remove((addr, port))
            mem.append((mac_addr, port))
            break
    else:
        mem.append((mac_addr, new_port))
        # if the table is full, remove the oldest entry
        if len(mem) > mem_capacity:
            mem.popleft()
```

查找

```python
else:
    for addr, port in mem:
        # if the mac address is already in the table, send the packet
        if addr == eth.dst:
            log_info(f"Send packet {packet} to {port}")
            net.send_packet(port, packet)
            break
    else:
        broadcast(fromIface, packet)
```

自己做一个简单测试，仅用于检查表项是否正常删除：

```python
# test case 3: 4 more packets with different mac addresses
for i in range(4):
    hw_src = '40:00:00:00:00:0' + str(i)
    hw_dst = '50:00:00:00:00:0' + str(i)
    ip_src = '192.168.127.' + str(i)
    ip_dst = '192.168.128.' + str(i)
    reqpkt = new_packet(hw_src, hw_dst, ip_src, ip_dst)
    s.expect(PacketInputEvent("eth0", reqpkt, display=Ethernet), (''))
    s.expect(PacketOutputEvent("eth1", reqpkt, "eth2", reqpkt, display=Et

# test case 4: the entry whose mac is 30:00:00:00:00:01 should be erased
reqpkt = new_packet(
    "20:00:00:00:00:01",
    "30:00:00:00:00:01",
    '192.168.1.100',
    '172.16.42.2'
)
s.expect(
```

```
4   Ethernet frame destined for 30:00:00:00:00:01 should
    arriveon eth1 after self-learning
5
6
7
8
9
10
11
12
13  An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:01 should arrive on eth2
14  the entry whose mac is 30:00:00:00:00:01 should have been
    erased


All tests passed!
```

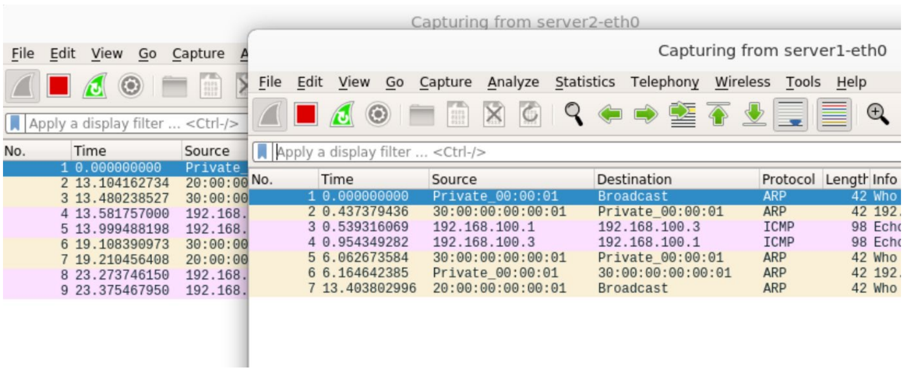
官方测试集，PASS！

```
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!
```

下面，重复之前的流程，是驴子是马总得拉出去遛遛。为了测试方便，我们暂时把容量改为 2。然后，依次进行如下操作。

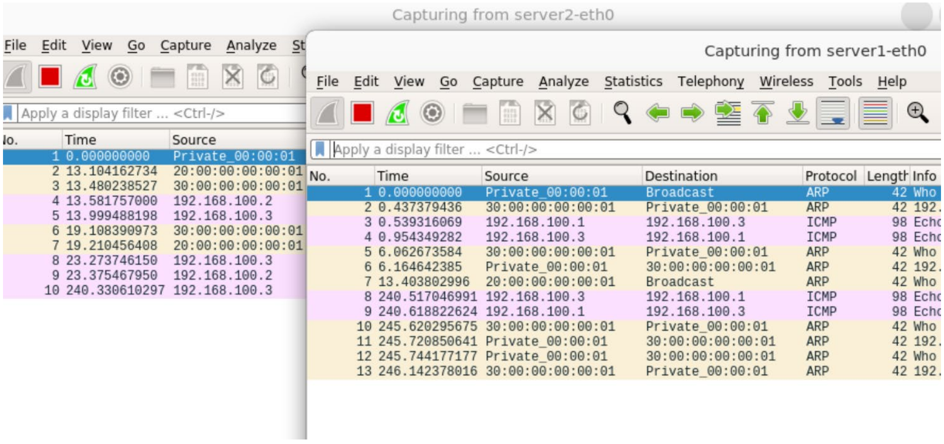
```
1) server1 ping -c 1 client
2) server2 ping -c 1 client
3) client  ping -c 1 server2
```

观察 server1 和 2，如下图：



接着

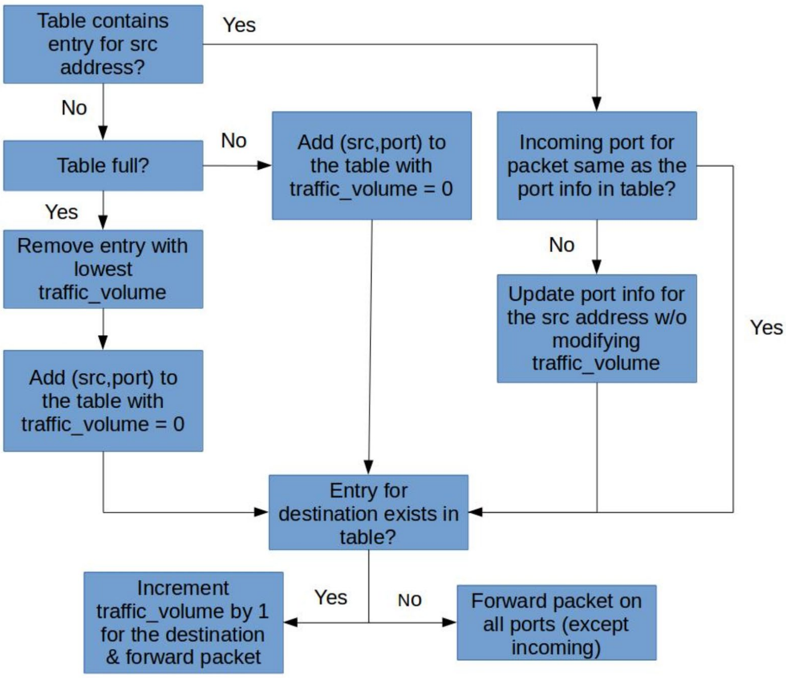
1) client ping -c 1 server1



由于 LRU 的替换机制，交换机找不到 server1 对应的端口，因此再次进行了广播。

5. Least Traffic Volume

不说废话，图片说明一切。



若记录已满，则取出最小的……这不就是个堆嘛？

当交换机接收到一个包，若表中已有记录，则啥都不做；否则添加记录，若表满，则去除流量最小的记录。

```python
def update_mem(mac_addr, new_port) -> None:
    nonlocal mem, mem_capacity
    for vol, addr, port in mem:
        # if the mac address is already in the table, do nothing
        if (addr, port)== (mac_addr, new_port):
            break
    else:
        # if the table is full, remove the entry with the lowest volume
        if len(mem) >= mem_capacity:
            heapq.heappop(mem)
        heapq.heappush(mem, (0, mac_addr, new_port))
```

发出包时，更新记录（流量自增）。由于此时堆的结构可能被破坏了，我们需要重新建堆。

```python
for i in range(len(mem)):
    # if the mac address is already in the table, send the packet
    # and update the entry(vol += 1)
    if mem[i][1] == eth.dst:
        vol, dst_addr, dst_port = mem.pop(i)
        log_info(f"Send packet {packet} to {dst_port}")
        net.send_packet(dst_port, packet)
        mem.append((vol + 1, dst_addr, dst_port))
        heapq.heapify(mem)
        break
else:
    broadcast(fromIface, packet)
```

由堆的性质，易得，当表容量为 N，算法复杂度为 O(N)，仍然优于每次更新时排序的算法。

编写自己的测试脚本。为了测试方便，我们暂时把转发表容量调为 2。

```python
mem = [] # a heap
mem_capacity = 2
```

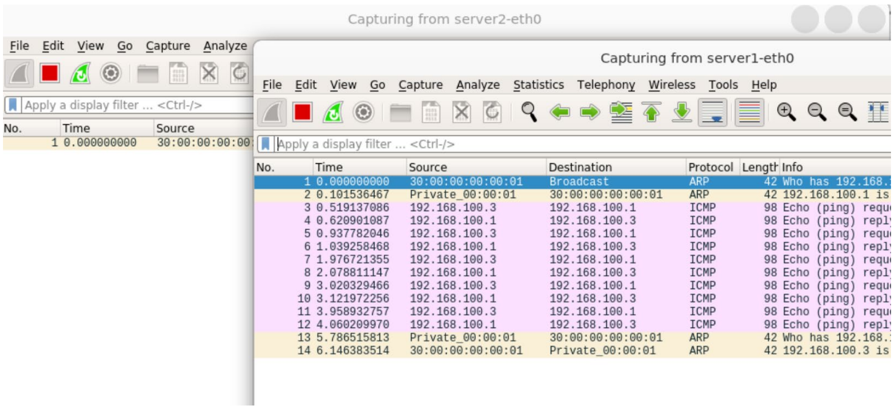
```
7   An Ethernet frame with a broadcast destination address
    should arrive on eth2
8   The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0 and eth1
9   An Ethernet frame destined for 30:00:00:00:00:00 should
    arrive on eth1
10  The Ethernet frame destined for 30:00:00:00:00:00 should be
    forwarded out ports eth0 and eth2


All tests passed!
```

官方测试集。

```
   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!
```

实战演练，开始!

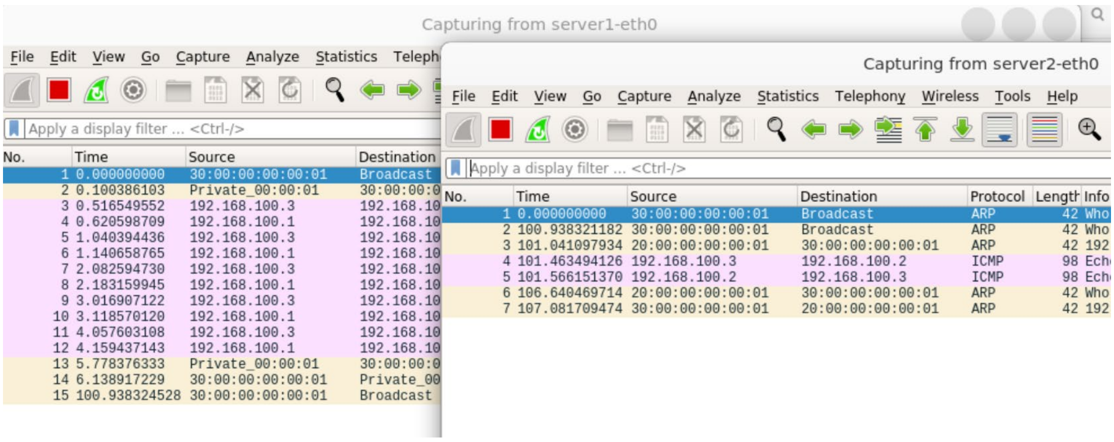
同样，为了方便，我们把表的容量改为 2。并使 client 向 server1 连续发送 5 次。如预期的那样，server2 仅接收到第一次的广播。



此时，client 流量比 server1 多 1。

```
16:03:11 2022/03/13      INFO [(6, EthAddr('10:00:00:00:00:01'), 'switch-eth0'),
(7, EthAddr('30:00:00:00:00:01'), 'switch-eth2')]
```

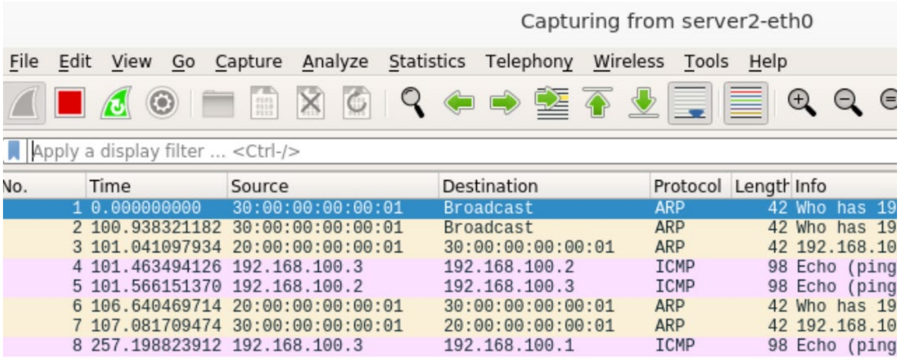
使 client 向 server2 发送。



可以看到，server1 的项被删除了。

```
16:04:51 2022/03/13     INFO [(2, EthAddr('20:00:00:00:00:01'), 'switch-eth1'),
(10, EthAddr('30:00:00:00:00:01'), 'switch-eth2')]
```

此时使 client 去 ping server1，server2 会接收到交换机的广播。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 19
2	100.938321182	30:00:00:00:00:01	Broadcast	ARP	42	Who has 19
3	101.041097934	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.10
4	101.463494126	192.168.100.3	192.168.100.2	ICMP	98	Echo (ping
5	101.566151370	192.168.100.2	192.168.100.3	ICMP	98	Echo (ping
6	106.640469714	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 19
7	107.081709474	30:00:00:00:00:01	20:00:00:00:00:01	ARP	42	192.168.10
8	257.198823912	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping

三、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

四、 总结与感想

- 通过几份练习，至少这个学期对交换机念念不忘了；
- 接触了多种替换算法，了解了不同算法的优劣；
- 锻炼了自己写测试样例的能力（尽管样例不是一般的弱）；
- 更加了解 MAC、端口等知识。