

# 南京大学本科生实验报告

课程名称： 计算机网络

任课教师：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	191220029	姓名	傅小龙
Email	<a href="mailto:1830970417@qq.com">1830970417@qq.com</a>	开始/完成日期	2021/3/24 - 2021/4/1

## 1.实验名称

Lab2: Learning Switch

## 2.实验目的

学习网络中的自学习交换机的工作原理并实现之。

## 3.实验内容

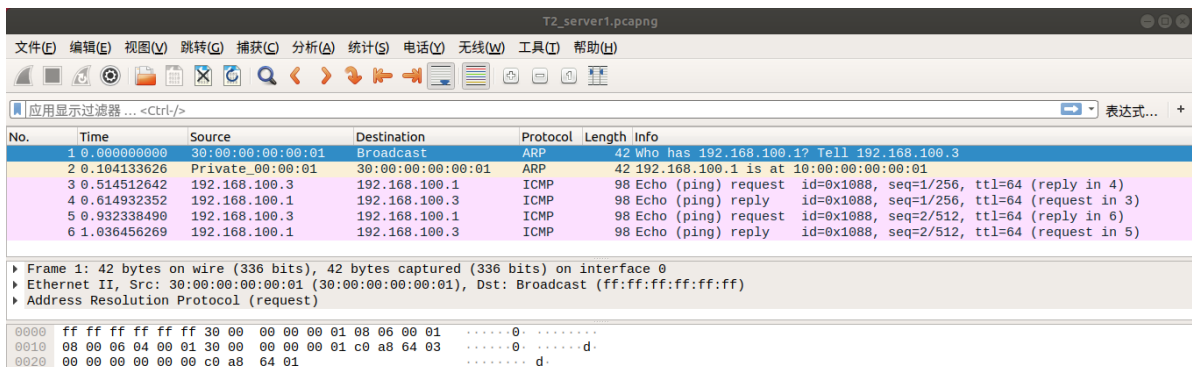
### 3.1 Basic Switch

交换机与集线器的区别在于交换机能够记录网络中结点的MAC地址以及该节点与交换机相连的端口的对应关系。当有数据包发送至交换机时，交换机将源结点的MAC-端口对应地址记录下来。若包的发送目标的MAC在交换机中已经有记录对应的端口，则直接将包由对应的端口发送出去。否则交换机将该包发送至除源节点所属端口外的所有其他结点。

### Deploying

在完成Task2中相关内容后，参照实验手册Task2-Deploying环节对 `server1` 和 `server2` 进行抓包。

下图是wireshark在 `server1` 节点捕获到的数据包：

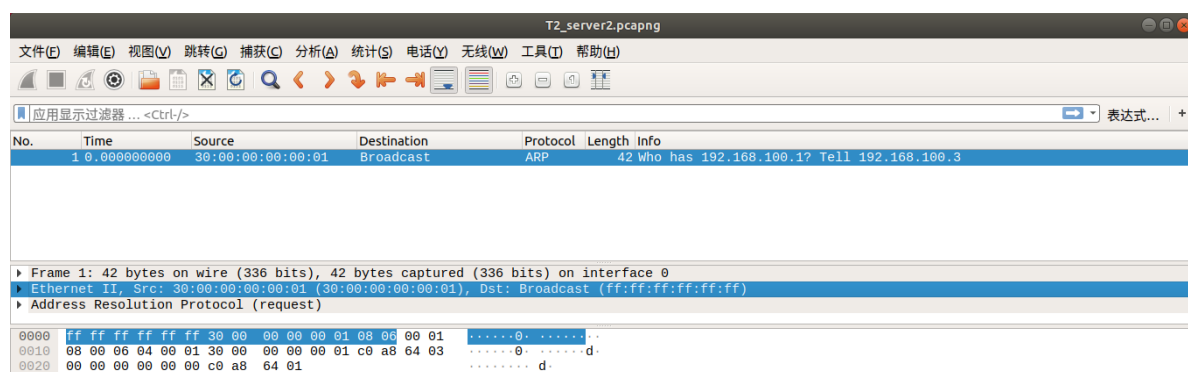


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	who has 192.168.100.1? Tell 192.168.100.3
2	0.104133626	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.514512642	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x1088, seq=1/256, ttl=64 (reply in 4)
4	0.614932352	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x1088, seq=1/256, ttl=64 (request in 3)
5	0.932338490	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x1088, seq=2/512, ttl=64 (reply in 6)
6	1.036456269	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x1088, seq=2/512, ttl=64 (request in 5)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

0000 ff ff ff ff ff 30 00 00 00 01 08 06 00 01 .....0.....  
0010 08 00 06 04 00 01 30 00 00 00 01 c0 a8 64 03 .....0.....d..  
0020 00 00 00 00 00 00 c0 a8 64 01 .....d.....

下图是wireshark在 `server2` 节点捕获到的数据包：



这里输入的 ping 指令要让 client 节点向 server1 发送2次 ping 包.

①由于 client 节点尚不知道 server1 的MAC地址，无法生成符合协议的数据包，因而需要先通过ARP 包广播询问网络中 server1 的MAC地址（根据 ping 指令给出的目标ip地址），这也就产生了在 server1 监听到的第一个数据包以及 server2 监听到的数据包：由 client 发出，广播至整个网络。这里 switch 学习到了 client 的MAC地址与端口号的对应关系。

② server1 在收到该ARP包后，将自己的MAC地址同样通过ARP协议发送给 client，这产生了在 server1 监听到的第二个数据包：由 server1 发出，发送给 client。这里 switch 学习到了 server1 的MAC地址与端口号的对应关系。

③ client 收到 server1 的ARP包后，知道了 server1 的MAC地址，生成ICMP包发送给 server1 进行 ping请求。这产生了在 server1 监听到的第三个数据包。然后 server1 收到 client 的ping请求，向 client 发送ping回复，这产生了在 server1 监听到的第四个数据包。同样地，在 server1 监听到的第五和第六个数据包是另一个ping的请求和回复。由于 switch 已经学习到了 client 和 server1 的MAC-端口对应关系，故在转发这以上四个数据包是都可以进行点对点的发送。

## 3.2 Timeouts

由于交换机所属的网络拓扑可能存在改变，需要实时对交换机保存的MAC-端口对应信息进行更新。这里采取每10s更新一次信息的方法。当交换机收到数据包时，将源节点的MAC-端口对应信息保存或更新其时间，同时遍历交换机内的MAC-端口信息的所有表项，删去超时的表项。

## Testing

下图是运行 myswitch\_to\_testscenario.srpy 测试文件得到的结果：

```
njucs@njucs-VirtualBox: ~/cnLab02/lab-2-191220029
File Edit View Search Terminal Help
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!

(syenv) njucs@njucs-VirtualBox:~/cnLab02/lab-2-191220029$
```

## Deploying

参照实验手册Task2-Deploying环节，在Mininet中运行该网络拓扑并对 server1 和 server2 抓包。

在 client 结点的终端尝试ping server1 结点，采用如下指令

```
ping -c 1 192.168.100.1
```

在输入第一条ping指令后立刻再输入相同的ping指令，之后等待>10s时间，再次输入该ping指令。最后得到的抓包结果如下：

server1 抓包结果：

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	0.101659925	Private:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.512323961	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0xid27, seq=1/256, ttl=64 (reply in 4)
4	0.612965173	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0xid27, seq=1/256, ttl=64 (request in 3)
5	3.988060685	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0xid28, seq=1/256, ttl=64 (reply in 6)
6	4.086033225	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0xid28, seq=1/256, ttl=64 (request in 5)
7	7.742598942	Private:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	6.086281584	30:00:00:00:00:01	Private:00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
9	21.013120275	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0xid29, seq=1/256, ttl=64 (reply in 10)
10	21.117635432	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0xid29, seq=1/256, ttl=64 (request in 9)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

0000 ff ff ff ff ff 30 00 00 00 01 00 00 00 01 .....0.....  
0010 00 00 00 04 00 01 30 00 00 00 01 c0 a8 64 03 .....0.....d..  
0020 00 00 00 00 00 00 c0 a8 64 01 .....d..

server2 抓包结果：

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	21.010107587	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0xid29, seq=1/256, ttl=64 (no response found!)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

0000 ff ff ff ff ff 30 00 00 00 01 00 00 00 01 .....0.....  
0010 00 00 00 04 00 01 30 00 00 00 01 c0 a8 64 03 .....0.....d..  
0020 00 00 00 00 00 00 c0 a8 64 01 .....d..

#### ①第一次ping:

在 server1 抓到的前4个包与在 server2 抓到的第1个包的含义在本报告 3.1节 中已给出解释，这里不再重复说明。

此时 switch 已经学习到了 client 和 server1 的MAC-端口对应信息。

#### ②第二次ping:

由于与第一次ping时间间隔<10s， switch 在①的学习成果尚未被抛弃，故ping请求和ping回复包都能够实现点对点的转发。所以wireshark仅在 server1 抓到这两个包。

#### ③第三次ping:

由于与第二次ping时间间隔>10s， switch 在①的学习成果已被抛弃，故ping请求包需要switch向其他节点广播，所以wireshark不仅在 server1 捕获了该包，在 server2 也捕获了该包，且 server2 捕获的请求包并没有对应的回复包。这时 switch 又学习到了 client 的MAC-端口对应信息，故ping回复包能够进行点对点转发。所以wireshark仅能在 server1 抓到回复包。

## 3.3 Least Recently Used

由于交换机的存储空间有限，当MAC-端口表无法存放下新的表项时，需要剔除某个表项来腾出空间。这里采用LRU算法，牺牲最近最少使用过的表项来存储新的信息。

参照实验手册Lab2-Task4中的flowchart图表，采用LRU算法的交换机能够记录MAC-PORT-AGE信息。在收到数据包时，若源节点的信息已经在switch中登记，如MAC-端口不匹配，则更新端口信息但不重置其AGE，将各表项的AGE时间项加1后转发该包。若源节点信息在switch中没有登记，则记录相关信息，若空间不足，采用LRU算法牺牲AGE项最大的表项，然后再将除源节点对应表项的AGE全部增加1。

## Testing

下图是运行 myswitch\_lru\_testscenario.srpy 测试文件得到的结果：

```
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!

(syenv) njucs@njucs-VirtualBox:~/cnLab02/lab-2-191220029$
```

## Deploying

参照实验手册Task2-Deploying环节，在Mininet中运行该网络拓扑并对 server1 和 server2 抓包。

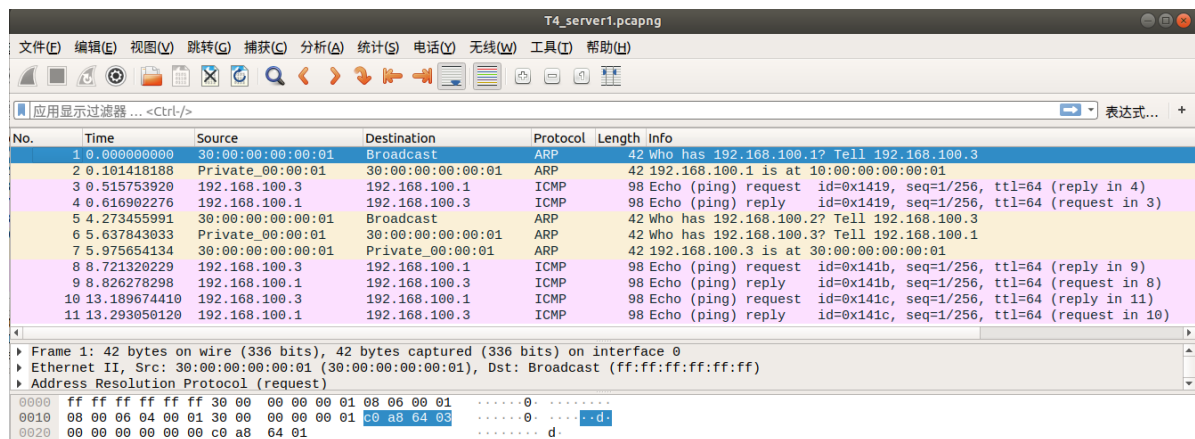
这里我们将 switch 的转发表大小暂时修改为 2，这样就可以在不修改网络拓扑的情况下检验这里实现的 LRU 替换算法的正确性。

在 client 结点的终端分别尝试ping server1 和 server2 结点，采用如下指令

```
ping -c 1 192.168.100.1
ping -c 1 192.168.100.2
```

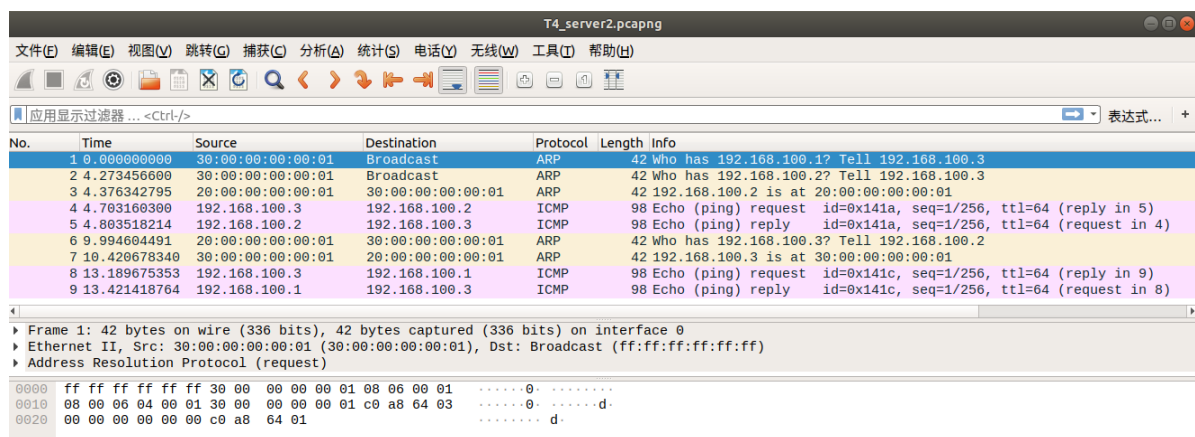
首先各ping一次 server1和 server2 结点，再ping两次 server1 结点。得到的抓包结果如下：

server1 抓包结果：



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	0.101418188	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	0.515753920	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x1419, seq=1/256, ttl=64 (reply in 4)
4	0.616902276	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x1419, seq=1/256, ttl=64 (request in 3)
5	4.273455991	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
6	5.637843033	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
7	5.975654134	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
8	8.721320229	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x141b, seq=1/256, ttl=64 (reply in 9)
9	8.826278298	192.168.100.3	192.168.100.3	ICMP	98	Echo (ping) reply id=0x141b, seq=1/256, ttl=64 (request in 8)
10	13.189674410	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x141c, seq=1/256, ttl=64 (reply in 11)
11	13.293050120	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x141c, seq=1/256, ttl=64 (request in 10)

server2 抓包结果：



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	4.273456600	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
3	4.376342795	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.2 is at 20:00:00:00:00:01
4	4.763160300	192.168.100.3	192.168.100.2	ICMP	98	Echo (ping) request id=0x141a, seq=1/256, ttl=64 (reply in 5)
5	4.803518214	192.168.100.2	192.168.100.3	ICMP	98	Echo (ping) reply id=0x141a, seq=1/256, ttl=64 (request in 4)
6	9.994604491	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.2
7	10.420678340	30:00:00:00:00:01	20:00:00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
8	13.189675353	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x141c, seq=1/256, ttl=64 (reply in 9)
9	13.421418764	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x141c, seq=1/256, ttl=64 (request in 8)

①在ping server1后，wireshark在 server1 捕获的前4个包以及在 server2 捕获到的第一个包与本报告3.1节中的对应分析已给出解释，这里不再赘述。此时 switch 已经学习到了 c1ient 和 server1 结点的 MAC-PORT 对应信息。

②在ping server2 后，由于 c1ient 并不知道 server2 的MAC地址，需要先通过ARP协议广播问询 server2 ip对应的MAC地址。故wireshark在 server1 捕获到了No.5-ARP包，在 server2 捕获到了No.2-ARP包， server2 收到该包后将自己的MAC地址用ARP协议再发回 server1，故wireshark在 server2 捕获到了No.3-ARP包。 switch 在转发该ARP回复包时学习到了 server2 的 MAC-PORT 对应信息，但是此时交换机内无空间存放该信息，需要通过LRU替换算法牺牲一个表项。由于 switch 对 c1ient 的 MAC-PORT 信息的最近使用为 server2 对 c1ient MAC地址问询的回复（即wireshark在 server2 捕获到的第3个包）， server1 信息的最近使用为 server1 对 c1ient 发来的ping包的回应（即wireshark在 server1 捕获到的No.4包）。故 server1 的信息为LRU项，应该被移除。

③再次ping server1 之前， server1 先发出ARP包问询 c1ient 的MAC地址（即wireshark在 server1 捕获到的No.6包）。 switch 又重新学到了 server1 的 MAC-PORT 信息，移除LRU项（LRU项 server2 的信息，分析过程同②，不再赘述）后储存之。之后 c1ient 发出回复（即wireshark在 server1 捕获到的No.7包），使得 server1 的 AGE 项重置，此时的LRU项为 c1ient 的相关信息。

此时ping server1，由于 switch 都存有两结点的信息，能够点对点的转发两个ping包（即wireshark在 server1 捕获到的No.8-9包）。

④最后一次ping server1 之前， server2 又发出了ARP包问询 c1ient 的MAC地址（即wireshark在 server2 捕获到的No.6包）。此时 switch 学到了 server2 的 MAC-PORT 信息，牺牲LRU项（LRU项为 server1 的相关信息，分析过程同②，不再赘述）并存储之。 c1ient 随后给出回复（即wireshark在 server2 捕获到的No.7包）， switch 根据转发表点对点转发。



此时ping server1，由于 switch 的转发表内没有 server1 的相关信息，switch 将该ping请求包 flood 至所有其他节点，对应了wireshark在 server1 和 server2 都捕获了该包（server1-No.10, server2-No.8）。当 server1 再发回ping回复包时，switch 又学习到了 server1 的 MAC-PORT 信息，需要牺牲LRU项存储，而被牺牲的表项恰好是 client 的信息，故 switch 在转发该回复包时只能再次 flood，故wireshark在 server1 和 server2 都捕获了该包（server1-No.11, server2-No.9）。LRU 替换算法实现的正确性在这一步被验证。

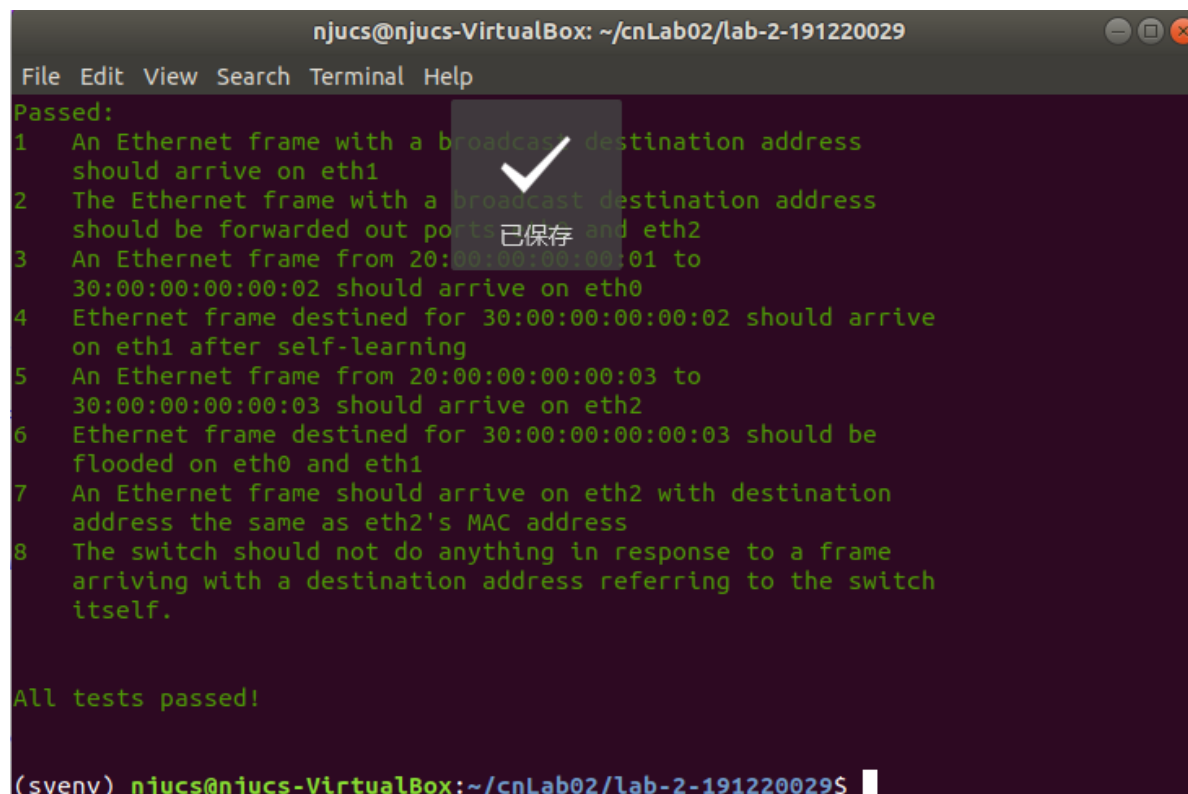
### 3.4: Least Traffic Volume

由于交换机的存储空间有限，当MAC-端口表无法存放下新的表项时，需要剔除某个表项来腾出空间。这里采用LTV算法，牺牲最近流量最少的表项来存储新的信息。

参照实验手册Lab2-Task5中的flowchart图表，采用LRU算法的交换机能够记录MAC-PORT-Traffic Volumn信息。在收到数据包时，若源节点的信息已经在switch中登记，如MAC-端口不匹配，则更新端口信息但不重置其流量，之后转发该包。若源节点信息在switch中没有登记，则记录相关信息，若空间不足，采用LTV算法牺牲AGE项最大的表项。若转发地址已在转发表中，将其对应信息的流量加1，否则 flood该包。

### Testing

下图是运行 myswitch\_lru\_testscenario.srpy 测试文件得到的结果：



```
njucs@njucs-VirtualBox: ~/cnLab02/lab-2-191220029
File Edit View Search Terminal Help
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!

(syenv) njucs@njucs-VirtualBox:~/cnLab02/lab-2-191220029$
```

### Deploying

参照实验手册Task2-Deploying环节，在Mininet中运行该网络拓扑并对 server1 和 server2 抓包。

这里我们将 switch 的转发表大小暂时修改为 2，这样就可以在不修改网络拓扑的情况下检验这里实现的 LTV 替换算法的正确性。

采用和本报告 3.3 节相同的测试方法：在 client 终端首先各ping一次 server1 和 server2 结点，再 ping两次 server1 结点。对 server1 和 server2 抓包。

server1 抓包结果：

T5\_server1.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(W) 无线(W) 工具(T) 帮助(H)

应用显示过滤器... <Ctrl-/> 表达式...

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	0.100471619	Private_00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.512274865	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0d9f, seq=1/256, ttl=64 (reply in 4)
4	0.612541534	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0d9f, seq=1/256, ttl=64 (request in 3)
5	2.804334009	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
6	5.832069725	Private_00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
7	6.250698796	30:00:00:00:00:01	Private_00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
8	7.816111904	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0da1, seq=1/256, ttl=64 (reply in 9)
9	7.916980196	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0da1, seq=1/256, ttl=64 (request in 8)
10	10.416348746	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0da2, seq=1/256, ttl=64 (reply in 11)
11	10.517386266	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0da2, seq=1/256, ttl=64 (request in 10)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
 Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 Address Resolution Protocol (request)  
 0000 ff ff ff ff ff 30 00 00 00 00 01 08 06 00 01 .....0.....  
 0010 08 00 06 04 00 01 30 00 00 00 01 c0 a8 64 03 .....0.....d..  
 0020 00 00 00 00 00 c0 a8 64 01 .....d.....

server2 抓包结果:

T5\_server2.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(W) 无线(W) 工具(T) 帮助(H)

应用显示过滤器... <Ctrl-/> 表达式...

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	2.804332661	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.2? Tell 192.168.100.3
3	2.908024797	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.2 is at 20:00:00:00:00:01
4	3.333925311	192.168.100.3	192.168.100.2	ICMP	98	Echo (ping) request id=0x0da0, seq=1/256, ttl=64 (reply in 5)
5	3.434246169	192.168.100.2	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0da0, seq=1/256, ttl=64 (request in 4)
6	8.643454164	20:00:00:00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.2
7	9.064610967	30:00:00:00:00:01	20:00:00:00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01
8	10.417313842	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0da2, seq=1/256, ttl=64 (no response found!)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
 Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 Address Resolution Protocol (request)  
 0000 ff ff ff ff ff 30 00 00 00 00 01 08 06 00 01 .....0.....  
 0010 08 00 06 04 00 01 30 00 00 00 01 c0 a8 64 03 .....0.....d..  
 0020 00 00 00 00 00 c0 a8 64 01 .....d.....

①在ping server1后, wireshark在 server1 捕获的前4个包以及在 server2 捕获到的第一个包与本报告3.1节中的对应分析已给出解释, 这里不再赘述。此时 switch 已经学习到了 client 和 server1 结点的 MAC-PORT 对应信息:

MAC	INTERFACE	TRAFFIC-VOLUMN
30:00:00:00:00:01(client)	switch-eth2	2
10:00:00:00:00:01(server1)	switch-eth0	1

②在ping server2 后, 由于 client 并不知道 server2 的MAC地址, 需要先通过ARP协议广播询问 server2 ip对应的MAC地址。故wireshark在 server1 捕获到了No.5-ARP包, 在 server2 捕获到了No.2-ARP包, server2 收到该包后将自己的MAC地址用ARP协议再发回 server1, 故wireshark在 server2 捕获到了No.3-ARP包。switch 在转发该ARP回复包时学习到了 server2 的 MAC-PORT 对应信息, 但是此时交换机内无空间存放该信息, 需要通过LTV替换算法牺牲一个表项。根据①给出的转发表, server1 的信息为LVT项, 应该被移除。在完成ping的请求和回复的转发后, switch 的转发表如下:

MAC	INTERFACE	TRAFFIC-VOLUMN
30:00:00:00:00:01(client)	switch-eth2	4
20:00:00:00:00:01(server2)	switch-eth1	1

③再次ping server1 之前, server1 先发出ARP包询问 client 的MAC地址 (即wireshark在 server1 捕获到的No.6包)。switch 又重新学到了 server1 的 MAC-PORT 信息, 移除LTV项 server2 的相关信息后储存之。之后 client 发出回复 (即wireshark在 server1 捕获到的No.7包)。

此时ping server1, 由于 switch 都存有两结点的信息, 能够点对点的转发两个ping包 (即wireshark在 server1 捕获到的No.8-9包)。

完成ping操作后 switch 的转发表如下:



MAC	INTERFACE	TRAFFIC-VOLUMN
30:00:00:00:00:01(client)	switch-eth2	6
10:00:00:00:00:01(server1)	switch-eth0	2

④最后一次ping server1之前，server2又发出了ARP包询问client的MAC地址（即wireshark在server2捕获到的No.6包）。此时switch学到了server2的MAC-PORT信息，牺牲LTV项server1的相关信息并存储之。client随后给出回复（即wireshark在server2捕获到的No.7包），switch根据转发表点对点转发。

此时的转发表如下

MAC	INTERFACE	TRAFFIC-VOLUMN
30:00:00:00:00:01(client)	switch-eth2	7
20:00:00:00:00:01(server2)	switch-eth1	1

此时ping server1，由于switch的转发表内没有server1的相关信息，switch将该ping请求包flood至所有其他节点，对应了wireshark在server1和server2都捕获了该包（server1-No.10, server2-No.8）。当server1再发回ping回复包时，switch又学习到了server1的MAC-PORT信息，需要牺牲LTV项存储，此时转发表没有变化，故被牺牲的表项是server2的信息，client的信息仍然留在转发表中，在转发回复包时不需要flood。故wireshark只在server1捕获了该包（server1-No.11）。LTV替换算法实现的正确性在这一步被验证。

## 4.核心代码

### 4.1 Basic Switch

使用dictionary(字典)数据结构存储学习到的MAC地址-接口信息。每次收到数据包时学习或更新表项。发送非广播包时查找字典内的MAC关键字，若查找到，则直接根据表项信息转发，否则flood。

```
table = {} #dictionary: MAC - port
#...
eth = packet.get_header(Ethernet)
#...
    table[eth.src] = fromIface

    if eth is None:
        #...
    if eth.dst in mymacs:
        #...
    else:
        if eth.dst in table.keys() and eth.dst != "ff:ff:ff:ff:ff:ff": #if
eth.dst is already learnt the dst's port and dst is not broadcast
            net.send_packet(table[eth.dst], packet) #then send out directly
            log_info (f"sending packet {packet} 'point to point' to
{intf.name}")
        else:
            #...
#...
```

## 4.2 Timeouts

调用time库。使用 dictionary(字典) 数据结构存储学习到的MAC地址-[接口-时间]信息。在每次收到需要转发的数据包时，更新当前时间以及学习或更新表项。然后遍历所有表项，将超时表项的关键字用 List 容器(代码中的变量名为ele)存储，完成遍历后根据ele[]将字典table中超时的表项删除。由于不支持在遍历字典的同时删除字典中的元素，故采用上述办法删去超时表项。

```
import time
#...
table = {} #dictionary: MAC - [port, time]
if eth is None:
    #...
    if eth.dst in mymacs:
        #...
    else:
        curtime = time.time() #get current time

        table[eth.src] = [fromIface, curtime]
        ele = []
        for MAC in table.keys():
            if curtime >= table[MAC][1] + 10:
                ele.append(MAC) #record timeout elements
        for i in ele:
            del table[i] #delete timeout elements
        if eth.dst in table.keys() and eth.dst != "ff:ff:ff:ff:ff:ff": #if
eth.dst is already learnt the dst's port and dst is not broadcast
            net.send_packet(table[eth.dst][0], packet) #then send out
directly
            log_info (f"sending packet {packet} 'point to point' to
{intf.name}")
        else:
            #...
```

## 4.3 Least Recently Used

使用 dictionary(字典) 数据结构存储学习到的MAC地址-[接口-最近使用时间]信息。相关逻辑在3.3节中已有说明这里不再赘述。

对于LRU表项的查找这里采用遍历整个字典查找最近使用时间值最大的一项，使用 List 容器index暂存待删除表项的[MAC-AGE]信息，完成遍历后将之删除。时间复杂度为O(n).

```
table = {} #dictionary: MAC - [port, age]

#...

if eth is None:
    #...
    if eth.dst in mymacs:
        #...
    else:
        if eth.src in table.keys(): #table contains entry for src address
            if table[eth.src][0] != fromIface: #incoming port for packet not
same as the port info in table
                table[eth.src][0] = fromIface #update port
```

```

        table[eth.src][1] += 1 #age++
    else : #add one new info to table
        while len(table) >= 5: # table is full
            index = [0, -1] # [MAC-MAXAGE]
            for MAC in table.keys():
                if table[MAC][1] > index[1]:
                    index = [MAC, table[MAC][1]]
                log_info(f"checking: {MAC} has age {table[MAC][1]}")
            log_info(f"remove {index[0]} from table")
            del table[index[0]] #delete LRU info
        table[eth.src] = [fromIface, 0] #add new info to table
        log_info(f"add {eth.src} to table")
    for MAC in table.keys(): #age ++ except input port(avoid adding age
to newly created item)
        if MAC != eth.src:
            table[MAC][1] += 1

#...

```

## 4.4 Least Traffic Volume

使用 `dictionary`(字典) 数据结构存储学习到的MAC地址-[接口-流量]信息。相关逻辑在3.4节中已有说明这里不再赘述。

对于流量最少表项的查找这里采用遍历整个字典查找流量值最小的一项，使用 `List` 容器 `index` 暂存待删除表项的[MAC-Traffic Volumn]信息，完成遍历后将之删除。时间复杂度为O(n)。

```

table = {} #dictionary: MAC - [port, traffic volumn]
#...

if eth is None:
    #...
if eth.dst in mymacs:
    #...
else:
    if eth.src in table.keys(): #table contains entry for src address
        if table[eth.src][0] != fromIface: #incoming port for packet not
same as the port info in table
            table[eth.src][0] = fromIface
    else: #add one new info to table
        if len(table)>= 5: #table is full
            index = [0, 10000000] #[MAC-MIN traffic volumn]
            for MAC in table.keys():
                if table[MAC][1] < index[1]:
                    index = [MAC, table[MAC][1]]
                log_info(f"checking: {MAC} has traffic volumn
{table[MAC][1]}")
            log_info(f"remove {index[0]} from table")
            del table[index[0]] #delete LTV info
        log_info(f"add {eth.src} - {fromIface} to table")
        table[eth.src] = [fromIface, 0] #add new info to table

#...

```

## 5. 实验总结与感想

本次实验对交换机的功能以及交换机内部的替换算法进行了模拟。在实验过程中，发现当交换机的转发表容量较小而与交换机的设备或接口较多时，交换机将频繁地使用替换算法来更新转发表，若网络繁忙时，交换机很可能会将大部分包以 flood 的形式转发（如本报告 3.3 节的 Deploying 环节的情况），这也增大了网络压力。替换算法的效率和交换机转发表容量也是网络性能的关键因素。

本报告中实现的替换算法对牺牲项的查找使用了  $O(n)$  时间复杂度的算法。还有时间复杂度更优的实现办法，例如将转发表项按堆结构存储，这样可以将寻找并删除牺牲项的时间复杂度下降到对数级别。

在抓包过程中，各节点产生的 ARP 包也对实验结果产生了影响，导致各 Deploying 环节设计的测试办法与预期实验结果产生了偏差。不过好在也能体现替换算法实现的正确与否。