

Lab2实验报告

191220138 杨飞洋

实验目的

了解网络中交换机switch的工作原理，了解switch自学习的机制，并学习采用不同的替换算法进行优化。

实验过程

Basic Switch

基本原理

实现一个基本的switch，switch有一个表，里面记录了端口所连接的mac地址。

如果接收到的数据包中源地址在表中找不到，则将源地址和所对应的端口添加到表中，如果已在表中，则需要更新端口。

对于目的地址，如果目的地址在表中，则直接从表中对应的端口把数据包发送出去，否则从switch中flood。

python中有字典 `dict` 这一数据结构非常适合用来做这个表，由于需要频繁地查找mac地址是否在表中，并查找其对应的端口，我选择把 `key` 值设置为mac地址，`value` 值设置为端口 `port`

代码逻辑

在 `while` 循环外先初始化表，命名为 `lst`

```
lst = {}
```

在循环体内部，首先要根据 `fromIface` 找到对应的端口，再对源地址 `eth.src` 在表中进行更新

```
for intf in my_interfaces:
    if(fromIface == intf.name):
        lst[eth.src] = intf
        break
```

`lst[eth.src] = intf` 这句赋值语句性能很优越，如果 `eth.src` 不在字典中，就添加一项，如果在，就对其`value`值进行修改。

最后对目的地址 `eth.dst` 进行判断，如果在表中，就从对应端口输出，否则flood

```

if eth.dst in lst.keys():
    intf = lst[eth.dst]
    net.send_packet(intf, packet)
else:
    for intf in my_interfaces:
        if fromIface != intf.name:
            log_info (f"Flooding packet {packet} to {intf.name}")
            net.send_packet(intf, packet)

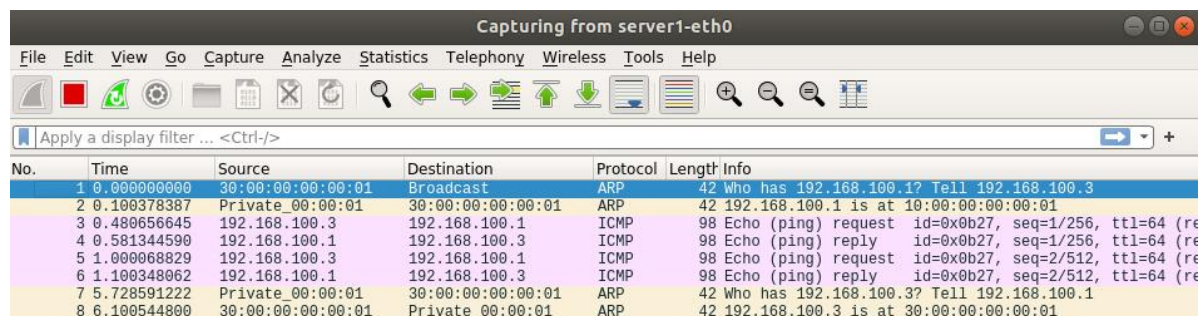
```

测试结果

在client的terminal中键入 `ping -c 2 192.168.100.1`，这个命令会给server1发送两个回响请求，即如果server1收到了这个包，会给client发送回响消息。

用wireshark监听server1和server2的抓包过程，结果如下：

server1:



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	0.100378387	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	0.480656645	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0b27, seq=1/256, ttl=64 (re
4	0.581344590	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0b27, seq=1/256, ttl=64 (re
5	1.000068829	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0b27, seq=2/512, ttl=64 (re
6	1.100348062	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0b27, seq=2/512, ttl=64 (re
7	5.728591222	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	6.100544800	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01

```

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface server1-eth0, id 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)

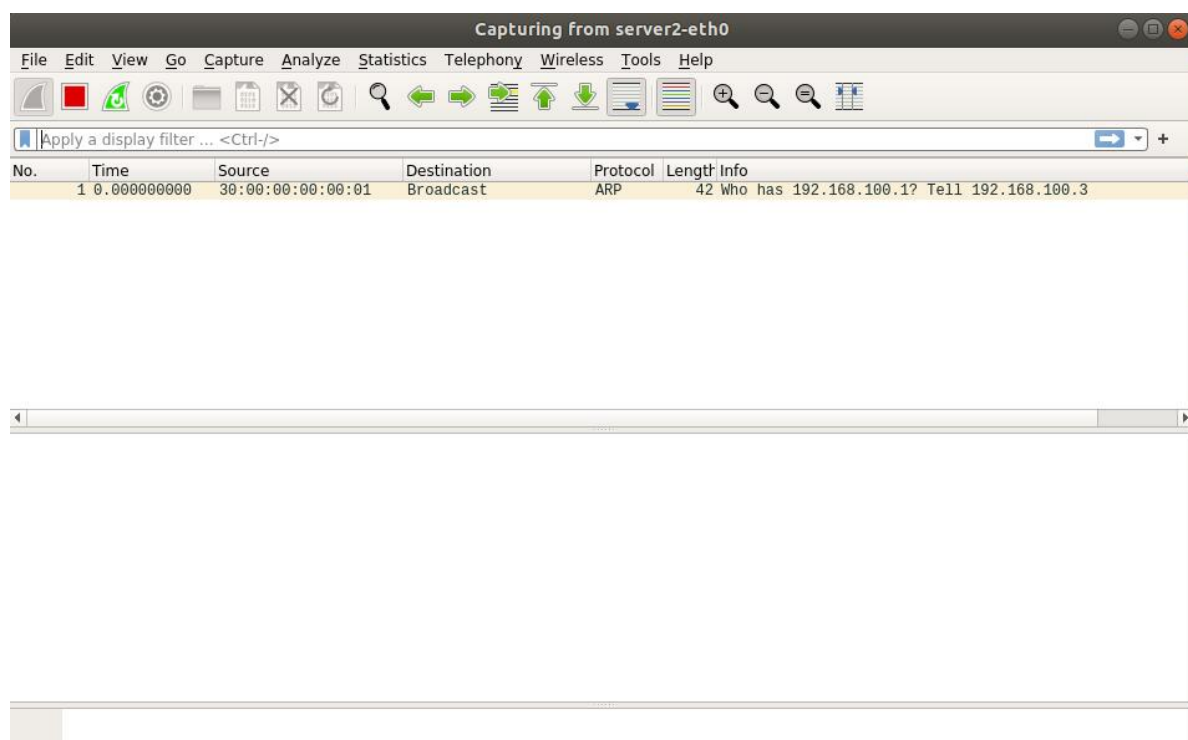
```

```

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.
0020 00 00 00 00 00 00 c0 a8 64 01 .....d.

```

server2:



可见server1接收到了两个来自client的协议为ICMP的数据包，并且发送回给了client两个协议为ICMP的数据包。

在switch第一次接收到来自client的包时，表还是空的，会学习client对应的端口，并把包flood出去，server1接收到，发送回响，此时switch学习server1对应的端口，由于表中已有client的端口，不用flood，直接从对应的端口输出。后面两个的数据传送过程，switch都无需再学习，表项已存在。

由于目的地址和server2的mac地址不一致，则server2始终都不会收到包，只有监听到一个协议为ARP的广播报文，这是最初有client发出的，同样server1也会接收到。

Timeouts

基本原理

将表中一些长期没有被访问的mac地址删除，时间限制设置为10s。

对原先的数据结构做出微小改动，原来是 `lst = {mac_addr : port}`，现在将时间加入，改为 `lst = {mac_addr : (port,time)}`

python提供 `time` 模块，可以用其中的 `time()` 函数获取当前时间

代码逻辑

首先获取当前的时间：

```
t = time.time()
```

找到当前 `fromIface` 对应的 `port`，再对表中 `eth.src` 项进行更新

```
for intf in my_interfaces:
    if(fromIface == intf.name):
        lst[eth.src] = (intf,t)
        break
```

再对表中长期没有用的项进行删除:

```
for mac_addr in list(lst.keys()):
    if(t - lst[mac_addr][1]>10):
        del lst[mac_addr]
```

注意要加 `list`, 否则不允许对 `iterator` 进行删除。

判断 `eth.dst` 是否在表中, 如果在, 从对应端口输出。

```
if eth.dst in lst.keys():
    intf = lst[eth.dst][0]
    net.send_packet(intf,packet)
```

如果不在则flood, 具体代码不再贴了。

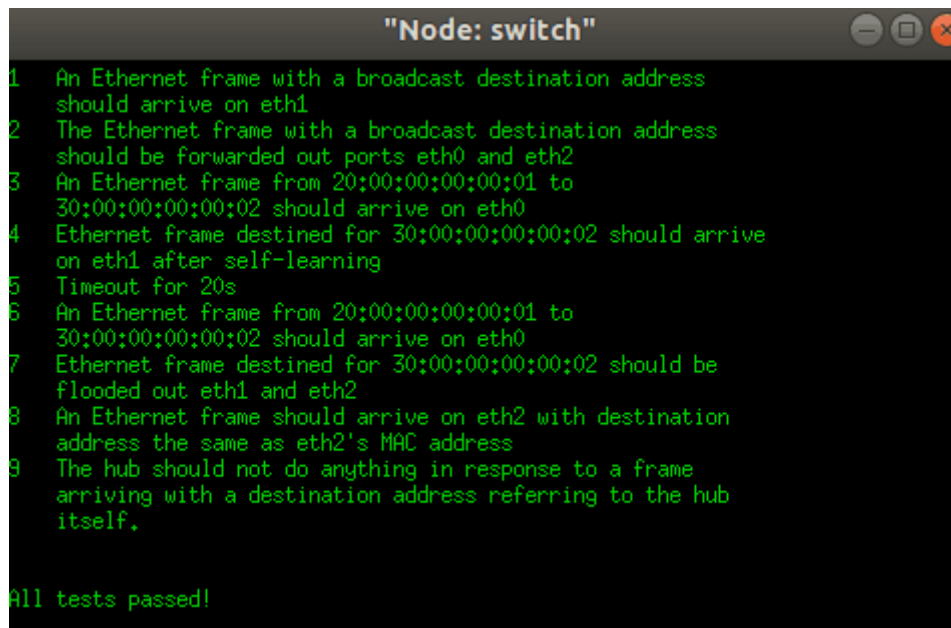
测试结果

在mininet中 `xterm switch` 打开switch

在switch中, 键入 `source /home/rafael/switchyard/syenv/bin/activate` 打开虚拟环境

利用testcases中的测试文件进行测试 `swyard -t testcases/myswitch_to_testscenario.srpy`
`myswitch_to.py`

结果如下:



```
"Node: switch"
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!
```

Least Recently Used

基本原理

表中最多存在5个表项，当需要进行替换时，淘汰掉最长不进行数据传输的节点。

对timeout的数据结构再改动下，改为 `lst = {mac_addr : [port,age]}`，因为要随时改动age，tuple型不支持，改为list

淘汰掉age最大的。

代码逻辑

首先根据 `fromIface` 找到对应的 `port`

```
port = my_interfaces[0]

for intf in my_interfaces:
    if intf.name == fromIface:
        port = intf
        break
```

如果 `eth.src` 在表中，检查一下端口和之前的是否一样，如果不一样进行改动。如果不在表中且表满，找到表中age最大的表项删除，加入新表项，其余age+1，如果表未满，则加入，其余age+1。

找age最大的表项可以用mac和lambda函数轻松解决

```
for mac_addr in lst.keys():
    lst[mac_addr][1] += 1
if eth.src in lst:
    if port != lst[eth.src][0]:
        lst[eth.src][0] = port
else:
    if len(lst) == 5:
        max_addr = max(lst, key = lambda k:lst[k][1])
        del lst[max_addr]
    lst[eth.src] = [port,0]
```

`lst[eth.src] = [port,0]` 可以对表项进行更新和改动，十分高效。

最后处理目的地址 `eth.dst`，如果在表中从对应的端口输出数据包，并且更新其age为0

```
if eth.dst in lst.keys():
    intf = lst[eth.dst][0]
    lst[eth.dst][1] = 0
    net.send_packet(intf,packet)
```

如果不在表中则flood出去。

测试结果

和timeout类似的测试方式，结果如下：

```
"Node: switch"

on eth3 after self-learning
11 An Ethernet frame from 40:00:00:00:05 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 30:00:00:00:05 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 20:00:00:00:05 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
les itself.

All tests passed!
```

Least Traffic Volume

基本原理

和LRU一样，还是最多存在5个表项，淘汰的规则做出改变，替换掉经历过数据传输最少的节点。

数据结构沿用LRU的，为 `lst = {mac_addr : [port, traffic_volume]}`，淘汰掉traffic最小的。

代码逻辑

首先根据 `fromIface` 找到对应的 `port`

```
port = my_interfaces[0]

for intf in my_interfaces:
    if intf.name == fromIface:
        port = intf
        break
```

如果 `eth.src` 在表中，检查一下端口和之前的是否一样，如果不一样进行改动

```
if eth.src in lst:
    if port != lst[eth.src][0]:
        lst[eth.src][0] = port
```

如果 `eth.src` 不在表中，如果表不满，则加入，`traffic_volume=1`；如果表满，则找到 `traffic_volume` 最小的表项进行删除，并加入新表项。

```
if(len(lst) == 5):
    min_addr = min(lst, key = lambda k: lst[k][1])
    del lst[min_addr]

lst[eth.src] = [port, 0]
```

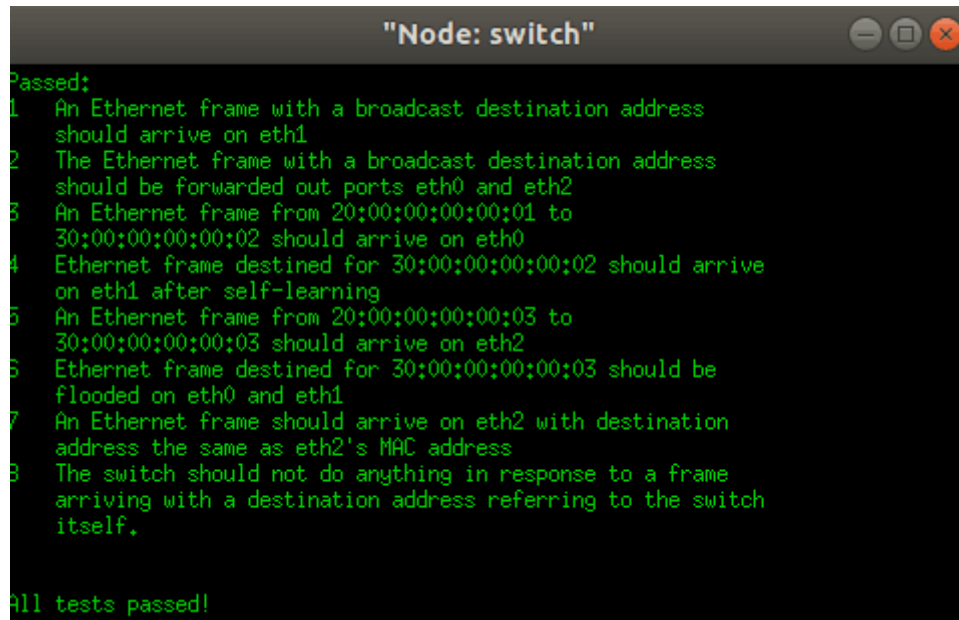
最后处理目的地址 `eth.dst`，如果在表中，则从对应的端口输出，且对应的 `traffic_volume+1`

```
if eth.dst in lst.keys():
    intf = lst[eth.dst][0]
    lst[eth.dst][1] += 1
    net.send_packet(intf, packet)
```

如果目的地址不在表中，则flood出去。

测试结果

和LRU类似的测试方法，结果如下：



```
"Node: switch"
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!
```

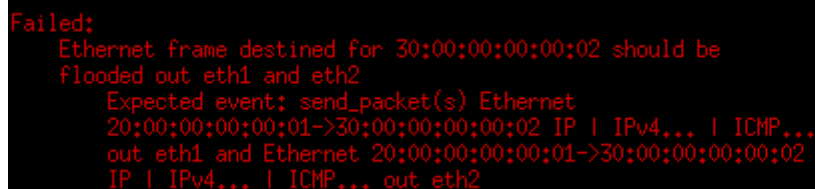
实验中遇到的问题与解决方法

1、

一开始打开几个node的terminal，进行ping操作时，在wireshark上什么都看不到，是因为我另开了几个linux的terminal，每个里面都start_mininet了一下，导致不同步，应该在同一个linux的terminal下，start一个mininet，在其中xterm

2、

问题截图：



```
Failed:
Ethernet frame destined for 30:00:00:00:00:02 should be
flooded out eth1 and eth2
Expected event: send_packet(s) Ethernet
20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4... | ICMP...
out eth1 and Ethernet 20:00:00:00:00:01->30:00:00:00:00:02
IP | IPv4... | ICMP... out eth2
```

报错原因是 `send_packet` 函数传参错误，应该传端口port

3、

问题截图：

```
Expected event:  
  Ethernet frame destined for 30:00:00:00:00:02 should be  
  flooded out eth1 and eth2  
  
Failure observed:  
  RuntimeError: dictionary changed size during iteration
```

原因是在字典进行iteration时是不能删除元素的，类似如下代码：

```
for mac_addr in lst.keys():  
    if(t - lst[mac_addr][1]>10):  
        del lst[mac_addr]
```

需要将其先转换成list型，`for mac_addr in list(lst.keys()):`

实验心得与思考

1

python内置的很多函数和语句性能真的十分优越，非常简练。

2

通过代码模拟加深了对switch自学习工作原理的理解，也学习了几种替换算法。

3

3种算法比较：

timeout算法对内存没有限制，理论上可以容纳很多的节点，硬件上不易实现，不过逻辑简单。

LRU算法，如果一个节点A是周期性传输，期间会有许许多多的其他节点路过，则每次都会将A删除，当A再来时，又要删除其他的，进行加入，会开销比较大。即流量较小的节点会挤占流量大的节点的空间。

traffic算法，问题是最新加入的数据常常会被踢除，因为其起始方法次数少。