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**

学院	计算机科学与技术系	专业（方向）	计算机科学与技术系
学号	181860109	姓名	吴润泽
Email	181860109@smail.nju.edu.cn	开始/完成日期	2020/3/11-2020/3/13

1. 实验名称：Learning Switch

2. 实验目的

1. 学习交换机的自学习原理
2. 理解转发表条目替换的三种机制
3. 掌握 *test scenario* 的编写和测试
4. 掌握配合 *mininet* 和 *wireshark* 验证机制正确性

3. 实验过程

topo结构更改说明

为了满足实验中转发项为5的要求，更改了给定的topo结构，在每个任务的抓包中均有使用，故作此说明。更改结构如下图所示：

```
#       node1  node7 node4
#          \   |   /
#    node6-switch1-node5
#          /   |   \
#     node2  node8  node3
```

启动mininet进行pingall测试，测试通过。

```
mininet> pingall
*** Ping: testing ping reachability
node1 -> node2 node3 node4 node5 node6 node7 node8 X
node2 -> node1 node3 node4 node5 node6 node7 node8 X
node3 -> node1 node2 node4 node5 node6 node7 node8 X
node4 -> node1 node2 node3 node5 node6 node7 node8 X
node5 -> node1 node2 node3 node4 node6 node7 node8 X
node6 -> node1 node2 node3 node4 node5 node7 node8 X
node7 -> node1 node2 node3 node4 node5 node6 node8 X
node8 -> node1 node2 node3 node4 node5 node6 node7 X
switch -> X X X X X X X X
*** Results: 22% dropped (56/72 received)
```

Task 2 自学习交换机基础实现

a. 实现原理

用字典存储mac地址以及对应端口。当packet到达switch时，检查源mac地址是否在字典若不在则加入字典中；检查目的mac地址是否在字典中，若存在则向字典中其对应端口发送，否则向除了源端口外所有端口发送。

b. 代码编写

```
    forward_table=dict()
    while True:
        ...#接收包的逻辑省略
        if packet[0].dst in mymacs:
            log_debug("Packet intended for me")
        else:

src_mac,dst_mac=str(packet[Ethernet].src),str(packet[Ethernet].dst)
            if src_mac not in forward_table.keys():#记录源mac对应端口
                forward_table[src_mac]=input_port
            if dst_mac in forward_table.keys():
                net.send_packet(forward_table.get(dst_mac),packet)
            else:
                #Flooding packet
```

c. 实现测试

I. test scenario测试

测试文档说明

```
    # test case 1: add "30:00:00:00:00:02 : eth1"
    testpkt = mk_pkt("30:00:00:00:00:02", "ff:ff:ff:ff:ff:ff",
"3.3.3.3",...)
    s.expect(PacketInputEvent("eth1", testpkt,...)
    s.expect(PacketOutputEvent("eth0", testpkt, "eth2", testpkt,...)
    # test case 2: match "30:00:00:00:00:02 : eth1"   add
"20:00:00:00:00:01 : eth0"
    reqpkt = mk_pkt("20:00:00:00:00:01", "30:00:00:00:00:02",
'2.2.2.2','3.3.3.3')
    s.expect(PacketInputEvent("eth0", reqpkt,...)
    s.expect(PacketOutputEvent("eth1", reqpkt,...)
    # test case 3: match "20:00:00:00:00:01 : eth0"   add
"40:00:00:00:00:02 eth2"
    resppkt = mk_pkt("40:00:00:00:00:02", "20:00:00:00:00:01",
'4.4.4.4', '2.2.2.2')
    s.expect(PacketInputEvent("eth2", resppkt,...)
    s.expect(PacketOutputEvent("eth0", resppkt,...)
```

1. [eth1]发送[h3]，此时转发表记录为{h3:eth1}，因目的地址为本地广播，因此[eth0,eth2]均收到；
2. [eth0]发送[h2]，此时转发表记录为{h2:eth0, h3:eth1}，目的地址[h3]在记录中，发送到[eth1]即可；
3. [eth1]发送[h4]，同case 2，目的地址[h2]在记录中，发送到[eth0]即可。

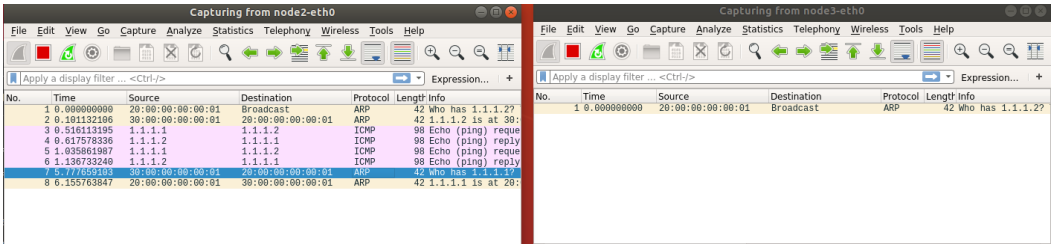
测试结果

运行$swyard\ -t\ mytests\ myswitch.py$，结果如下：

```
(wrz) njucs@njucs-VirtualBox:~/switchyard/lab_2$ swyard -t mytests  myswitch.py
18:41:01 2020/03/13     INFO Starting test scenario mytests

Results for test scenario hub tests: 6 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 send
    out on 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 send
    out on eth0


All tests passed!
```

II. 抓包测试

在mininet中启动编写好的topo结构，node1向node2发起ping，$ping\ node1\ -c\ 2\ node2$，得到node2，node3的抓包结构如图所示。



结果分析

1. 转发表中添加node1的mac地址以及到达的端口，node1不清楚node2的mac地址，将ARP请求帧广播到本地网络上的所有主机，因此node2和node3均能收到该ARP帧；
2. node2收到ARP帧，并向node1发送ARP应答，并且转发表中添加node2的mac地址以及到达的端口；
3. 之后由于转发表中含有两者的mac地址以及对应端口，两节点进行两次询问和回复的ICMP数据包；
4. 过了一段时间，node2又向node1发送ARP请求，询问1.1.1.1（node1）的mac地址，推测可能是由于node2的ip与mac的映射表项生存期已过，于是重新发起ARP请求。

Task 3 超时删除机制

a. 实现原理

1. 自学习部分与Task2原理相同，在添加映射后，并为其添加当前时间的时间戳，即字典键值对变为 $< mac, (port, time) >$；
2. 每当有帧到达，则检测字典中的每一项是否超过保留时间，若超过则将其删除即可。

b. 代码编写

```
    forward_table,max_time=dict(),10.0
    while True:
        ...#接收包的逻辑省略
        now_time=time.time()
        for key,value in list(forward_table.items()):
            if now_time-value[1]>=max_time:#超过存活时间，则丢弃
                forward_table.pop(key)
        if packet[0].dst in mymacs:
            log_debug("Packet intended for me")
        else:

src_mac,dst_mac=str(packet[Ethernet].src),str(packet[Ethernet].dst)
```

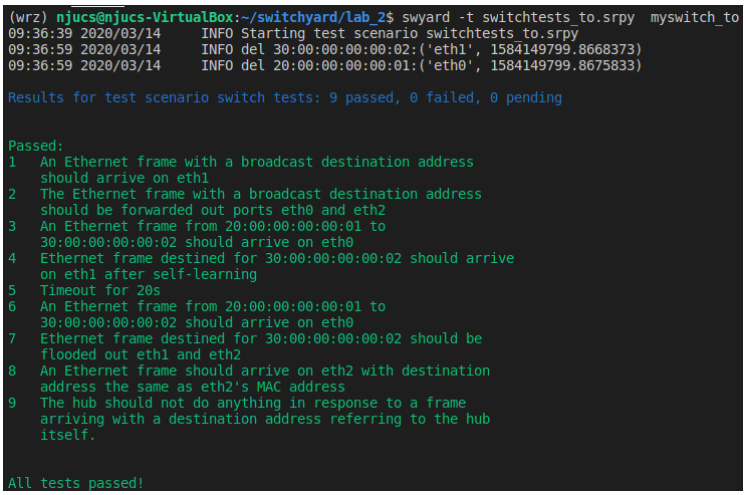
```
            forward_table[src_mac]=(input_port,now_time)#每到达一帧则更新时间戳
            if dst_mac in forward_table.keys():
                net.send_packet(forward_table.get(dst_mac)[0],packet)
            else:
                # flooding packet
```

c. 实现测试

I. test scenario测试

给定测试文件测试结果

运行$swyard\ -t\ switchtests_to.srpy\ myswitch_to.py$，结果如下：



测试文档说明

```
    # test case 1: add "30:00:00:00:00:02 : eth1"
    testpkt = mk_pkt("30:00:00:00:00:02", "ff:ff:ff:ff:ff:ff",
"3.3.3.3",...)
    s.expect(PacketInputEvent("eth1", testpkt, display=Ethernet),...)
    s.expect(PacketOutputEvent("eth0", testpkt, "eth2", testpkt,...)
    s.expect(PacketInputTimeoutEvent(15.0),"wait for 15.0 s")
    # test case 2: match "30:00:00:00:00:02 : eth1"   add
"20:00:00:00:00:01 : eth0"
    reqpkt = mk_pkt("20:00:00:00:00:01", "30:00:00:00:00:02",
'2.2.2.2','3.3.3.3')
    s.expect(PacketInputEvent("eth0", reqpkt,...)
    s.expect(PacketOutputEvent("eth1", reqpkt, "eth2",reqpkt,...)
    # test case 3: match "20:00:00:00:00:01 : eth0"   add
"40:00:00:00:00:02 eth2"
    resppkt = mk_pkt("40:00:00:00:00:02", "20:00:00:00:00:01",
'4.4.4.4', '2.2.2.2')
    s.expect(PacketInputEvent("eth2", resppkt,...)
    s.expect(PacketOutputEvent("eth0", resppkt,...)
```

1. [eth1]发送[h3]，转发表记录{h3:(eth1,t1)}，因目的地址为本地广播，因此[eth0,eth2]均收到；
2. 此时不会有报的发出，所有端口等待15.0s，则转发表记录超时，{h3:(eth1,t1)}表项被删去；
3. [eth0]发送[h2]，转发表记录{h2:(eth0,t2)}，目的地址[h3]不在记录中，进行泛洪，[eth1,eth2]均收到；

4. [eth1]发送[h4]，转发表记录{h4:(eth0,t3), h2:(eth0,t2)}，目的地址[h2]在记录中，发送到[eth0]即可。

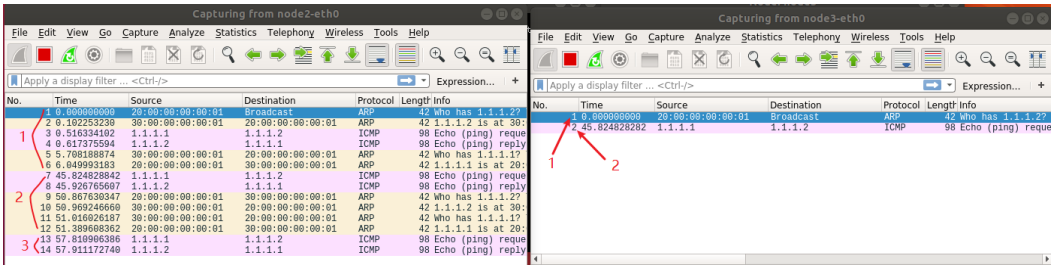
测试结果

运行$swyard\ -t\ mytests_to.py\ myswitch_to$，结果如下：

```
(wrz) njucs@njucs-VirtualBox:~/switchyard/lab_2$ swyard -t switchtests_to.srpy  myswitch_to
09:36:39 2020/03/14     INFO Starting test scenario switchtests_to.srpy
09:36:59 2020/03/14     INFO del 30:00:00:00:00:02:('eth1', 1584149799.8668373)
09:36:59 2020/03/14     INFO del 20:00:00:00:00:01:('eth0', 1584149799.8675833)

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!
```

II. 抓包测试

在mininet中运行已有topo结构，node1向node2发送3次ping请求，第一次与第二次间隔超过10s，得到node2和node3的抓包结果如下：



结果分析

1. 第一次ping与普通交换机情况相同，首先发送ARP帧请求获取目的mac地址，然后进行问答；
2. 等待超过10s，进行第二次通讯，此时node1网卡中仍记录node2的ip和mac地址的映射，故不需要再次发送ARP帧请求。交换机记录的node1和node2表项被删除，故进行泛洪，node3也收到ICMP数据包。
3. 在第二次通讯后，同样node1和node2网卡，互相发送ARP请求，确定ip和mac地址的映射。
4. 第三次通讯，由于交换机含有对应表项，因此仅在node1和node2间通讯，node3不再收到。

Task 4 最近最少使用机制

a. 实现原理

1. 转发表换用列表来方便进行索引，每当到达一个包，如果源地址在表中，则仅更新到达端口；
2. 如果源地址不在表中，则将其放在表头，如果表项满了，则删除表尾；
3. 如果目的地址在表中，则将其移动到表头；否则进行泛洪操作。

b. 代码编写

```
forward_table,max_len=list(),5
while True:
    ...#接收包的逻辑省略
        for i in range(len(forward_table)):
```

```
                if forward_table[i][0]==src_mac:
                    src_flag,forward_table[i][1]=True,input_port
                    break
            if src_flag==False:
                if len(forward_table)==max_len:
                    forward_table.pop()
                forward_table.insert(0,[src_mac,input_port])
            for i in range(len(forward_table)):
                if forward_table[i][0]==dst_mac:
                    dst_flag,dst_pair=True,forward_table[i]
                    forward_table.remove(forward_table[i])
                    forward_table.insert(0,dst_pair)
                    net.send_packet(dst_pair[1],packet)
                    break
            if dst_flag==False:
                # flooding packet
```

c. 实现测试

I. test scenario测试

给定测试文件测试结果

运行$swyard\ -t\ switchtests_lru.srpy\ myswitch_lru.py$，结果如下：

```
(wrz) njucs@njucs-VirtualBox:~/switchyard/lab_2$ swyard -t switchtests_lru.srpy myswitch_lru.py
11:03:49 2020/03/14     INFO Starting test scenario switchtests_lru.srpy

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!
```

测试文档说明

```
    case = [(1, 4), (2, 1), (3, 1), (4, 1), (5, 1), (6, 7), (4, 2), (4,
5)]
    #case为发包的情况
    for i in range(8):#开设8个端口
        s.add_interface('eth' + str(i), '90:00:00:00:00:0' + str(i))
    except_table = [[], [1], [1, 2], [1, 3, 2], [1, 4, 3, 2], [1, 5, 4,
3, 2],
                    [6, 1, 5, 4, 3], [6, 1, 5, 4, 3], [5, 6, 1, 4, 3]]
    #except_table为预期交换机中应有的表项
```

为了使得交换机中表项达到5，开设了8个端口（eth0-eth7），7个节点（mac1-mac2）；

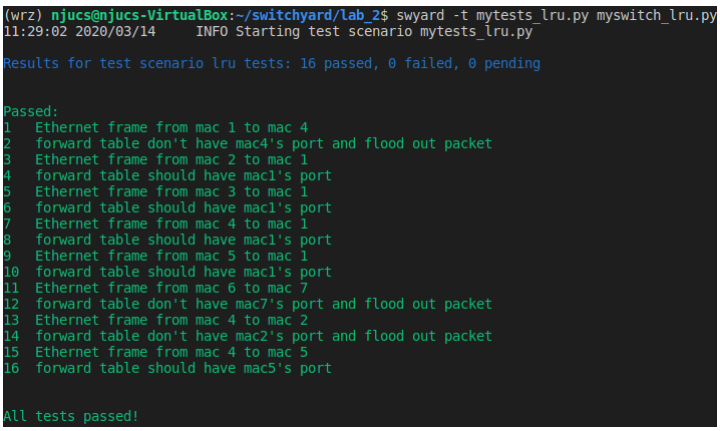
case与except_table一一对应，根据$Least\ Recently\ Used$的机制来确定当前转发表中应有的表项。当发送第i个包时，若except_table中第i项含有包的目的地址，则直接发送到对应端口，否则进行泛洪。由于代码逻辑几乎相同，故仅给出一例展示（曾尝试使用循环结构编写，但遇到一些问题，怀疑是测试文件的运行逻辑不同导致）。

下图为倒数第2次发包，倒数第3次发包时，转发表中mac2的表项被删除，本次发送进行泛洪。

```
    # 4 to 2
    mypkt = mk_pkt(
        str(4) + '0:00:00:00:00:00',str(2) + '0:00:00:00:00:00',
        str(4) + '.0.0.0',str(2) + '.0.0.0')
    s.expect(PacketInputEvent('eth' + str(4), mypkt, display=Ethernet),
             "Ethernet frame from mac {} to mac {}".format(4, 2))
    s.expect(
        PacketOutputEvent('eth0',mypkt,'eth1',mypkt,
                          'eth2',mypkt,'eth3',mypkt,
                          'eth5',mypkt,'eth6',mypkt,
                          'eth7',mypkt,display=Ethernet),
        "forward table don't have mac2's port and flood out packet")
```

测试结果

运行$swyard\ -t\ mytests_lru.py\ myswitch_lru.py$。，结果如下：



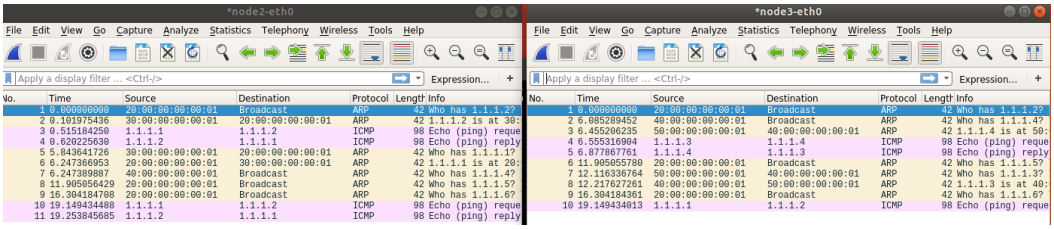
II. 抓包测试

1. 在mininet中依次进行node1 ping node2，node3 ping node4，node1 ping node5，node1 ping node6，node1 ping node2。
2. 在ARP请求过程中，虽然对于所有节点都进行了发送，即每个已经在交换表中的节点都根据 LRU 机制被放置在表头。但由于之后源和目的进行ICMP通讯，因此对表

项顺序无影响。

3. 查看node2，node3的抓包情况。

结果分析



1. node1 ping node2 此时交换机中无表项：
    - 首先node1发起ARP请求，询问node2的mac地址，因此node2，node3均收到了ARP帧对应Node2，3的抓包结果 $No.1$ ，交换机记录了node1的mac和端口映射；
    - 随后node2进行ARP应答，对应了Node2抓包结果的 $No.2$ ，交换机记录node2映射；
    - 随后进行请求和应答操作，对应了Node2抓包结果的 $No.3$ $No.4$ ；
    - 最后node2网卡刷新ip地址与mac地址的映射，对应了Node2抓包结果 $No.5$ $No.6$ 。
2. node3 ping node4 交换机表项为[node1,node2]：
    - 过程与1相同，ARP请求，对应Node2抓包结果的 $No.7$ ，Node3抓包结果的 $No.2$ ；
    - ARP应答对应Node3抓包结果 $No.3$ ，通讯过程对应Node3抓包结果 $No.4$ $No.5$ ；
    - node3网卡刷新ip地址与mac地址的映射过程，对应了Node3抓包结果 $No.7$ $No.8$ 。
    - 交换机在ARP请求应答过程中，添加了node3和node4映射。
3. node1 ping node5 交换机表项为[node3,node4,node1,node2]：
    - 过程与1相同，ARP请求，对应Node2抓包结果的 $No.8$ ，Node3抓包结果的 $No.6$ ；
    - 换机在ARP请求应答过程中，添加了node5映射，并根据 LRU ，node1移动至表头。
4. node1 ping node6 交换机表项为[node1,node5,node3,node4,node2]：
    - 过程与3相同，ARP请求，对应Node2抓包结果的 $No.9$ ，Node3抓包结果的 $No.9$ ；
    - 换机在ARP请求应答过程中，添加了node6映射，并根据 LRU ，node2在表尾因此被删除，node1为ping回复的目的地址移动至表头。
5. node1 ping node2 交换机表项为[node1,node6,node5,node3,node4]：
    - 此时node1记录了node2的mac地址，而交换机中不含有node2的mac地址与发送端口的映射，因此进行泛洪操作，因此node2，node3均收到包 $No.10$ ，随后node2进行回复对应抓包结果 $No.11$ 。

经由上述分析，LRU机制正常运行。

H4 Task 5 最少使用机制

H5 a. 实现原理

1. 转发表换用列表来方便进行索引，每到达一个包，如果源地址在表中，则更新其端口；

2. 如果源地址不在表中，则将其放在表头，并且其traffic设为0，如果表项满了，则删除表尾；
3. 如果目的地址在表中，同样将其traffic计数加一；否则进行泛洪操作；
4. 对整个列表按照traffic进行降序排列，使得表尾为traffic最低的表项，方便删除。

b. 代码编写

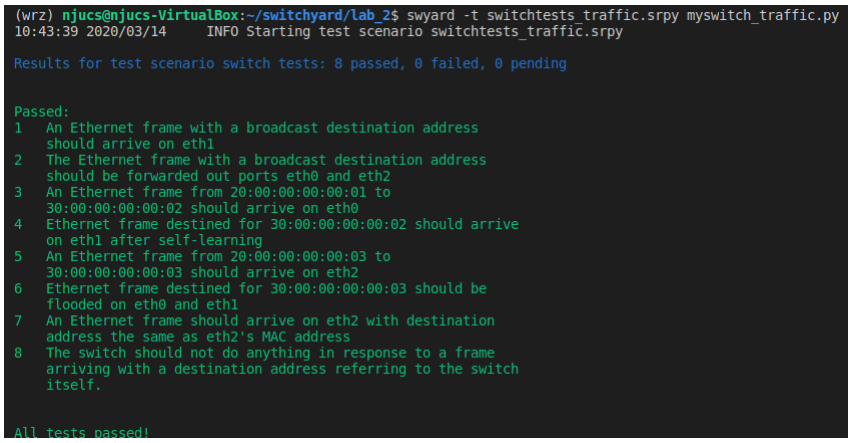
```
    forward_table,max_len=list(),5
    while True:
        ...#接收包的逻辑省略
            for i in range(len(forward_table)):
                if forward_table[i][0]==src_mac:
                    src_flag,forward_table[i][1]=True,input_port
                    break
            if src_flag==False:
                if len(forward_table)==max_len:
                    forward_table.pop()
                forward_table.insert(0,[src_mac,input_port,0])
            for i in range(len(forward_table)):
                if forward_table[i][0]==dst_mac:
                    dst_flag,forward_table[i][2]=True,forward_table[i]
[2]+1
                    net.send_packet(forward_table[i][1],packet)
                    break
            if dst_flag==False:
                # flooding packet
            forward_table.sort(key=lambda x:x[2],reverse=True)
```

c. 实现测试

I. test scenario测试

给定测试文件测试结果

运行$swyard\ -t\ switchtests_traffic.srpy\ myswitch_traffic.py$，结果如下：



测试文档说明

```
    case = [(1, 4), (2, 1), (3, 1), (4, 1), (5, 1), (2, 3), (2, 4), (3,
2),
            (6, 7), (4, 5), (4, 3)]
    #case为发包的情况
    for i in range(8):#开设8个端口
        s.add_interface('eth' + str(i), '90:00:00:00:00:0' + str(i))
    except_table = [[], [1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3,
4, 5],
                    [1, 3, 2, 4, 5], [1, 3, 4, 2, 5], [1, 2, 3, 4, 5],
                    [1, 2, 3, 4, 6], [1, 2, 3, 4, 6], [1, 3, 2, 4, 6]]
    #except_table为预期交换机中应有的表项
```

基本逻辑与Task4的测试文件相同，except_table表项的确定根据$Least\ Traffic\ Volume$原则，并且元素的使用次数按照降序排列。当发送第i个包时，若except_table中第i项含有包的目的地址，则直接发送到对应端口，否则进行泛洪。同样给出一次发包示例。

下图为倒数第2次发包，倒数第3次发包时，traffic为零的mac5从表项删除，因此本次进行泛洪。

```
    # 4 to 5
    mypkt = mk_pkt(
        str(4) + '0:00:00:00:00:00',str(5) + '0:00:00:00:00:00',
        str(4) + '.0.0.0',str(5) + '.0.0.0')
    s.expect(PacketInputEvent('eth' + str(4), mypkt, display=Ethernet),
             "Ethernet frame from mac {} to mac {}".format(4, 5))
    s.expect(
        PacketOutputEvent('eth0',mypkt,'eth1',mypkt,
                          'eth2',mypkt,'eth3',mypkt,
                          'eth5',mypkt,'eth6',mypkt,
                          'eth7',mypkt,display=Ethernet),
        "forward table don't have mac5's port and flood out packet")
```

测试结果

运行$swyard\ -t\ mytests_traffic.py\ myswitch_traffic$，结果如下：

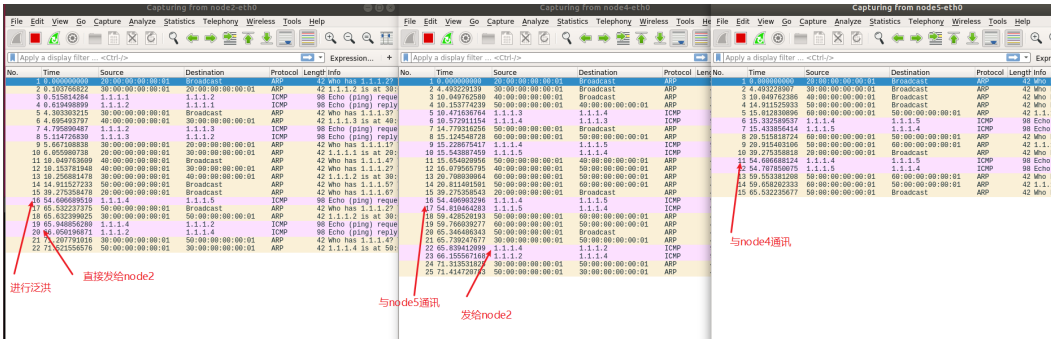
```
(wrz) njucs@njucs-VirtualBox:~/switchyard/lab_2$ swyard -t mytests_traffic.py myswitch_traffic.py
11:03:01 2020/03/14     INFO Starting test scenario mytests_traffic.py

Results for test scenario traffic tests: 20 passed, 0 failed, 0 pending


Passed:
1   Ethernet frame from mac 1 to mac 2
2   forward table don't have mac2's port and flood out packet
3   Ethernet frame from mac 2 to mac 1
4   forward table should have mac1's port
5   Ethernet frame from mac 3 to mac 1
6   forward table should have mac1's port
7   Ethernet frame from mac 4 to mac 1
8   forward table should have mac1's port
9   Ethernet frame from mac 5 to mac 1
10  forward table should have mac1's port
11  Ethernet frame from mac 2 to mac 3
12  forward table should have mac3's port
13  Ethernet frame from mac 4 to mac 3
14  forward table should have mac3's port
15  Ethernet frame from mac 6 to mac 7
16  forward table don't have mac7's port and flood out packet
17  Ethernet frame from mac 4 to mac 5
18  forward table don't have mac5's port and flood out packet
19  Ethernet frame from mac 4 to mac 3
20  forward table should have mac3's port


All tests passed!
```

H6 II. 抓包测试

1. 在mininet中依次进行node1 ping node2，node2 ping node3，node3 ping node4，node4 ping node5，根据 $Least\ Traffic\ Volume$ 原则，可知此时交换表中有node[1,2,3,4,5]5个表项，且由于ARP请求存在，node[1,2,3,4,5]的traffic依次减少。

2. node1 ping node6，交换表中表项已满，故将traffic最低的node5删除，然后添加node6；
3. node4 ping node5，交换表中不含node5，故进行泛洪；
4. node4 ping node2，交换表中有node2，故直接发送；
5. 查看node4、node2、node5的抓包情况，检验是否与预期一致。

结果分析



与预期结果一致，$Least\ Traffic\ Volume$机制工作正常。

4. 总结与感想

本次实验中对于交换机的转发机制机制理解更加明确，对于mininet，wireshark工具的使用以及结果的分析更加熟练，并且学会了编写自己的测试文件。

总体感觉难度不是很大，理解机制后，代码编写时间很短，但是在测试代码和结果分析时遇到障碍，如sleep函数的使用，期望事件的具体含义。在机制测试环节花费的时间最长，希望以后可以更加熟练。

5. 文档结构

```
njucs@njucs-VirtualBox:~/switchyard/lab_2$ tree
.
├── 181860109吴润泽_lab_2.pdf
├── myswitch_lru.py
├── myswitch.py
├── myswitch_to.py
├── myswitch_traffic.py
├── mytests_lru.py
├── mytests.py
├── mytests_to.py
├── mytests_traffic.py
├── start_mininet.py
├── switchtests_lru.srpy
├── switchtests_to.srpy
└── switchtests_traffic.srpy

0 directories, 13 files
```