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	计算机科学与技术系	专业	计算机科学与技术
学号	185220001	姓名	磯田智明
Email	185220001@smail.nju.edu.cn	开始/完成日期	3.13/3.20

1. 实验名称

Switchyard & Mininet

2. 实验目的

了解Linux，Python，Git的基本使用方式

安装并配置VSCode，Mininet，Wireshark，Switchyard，并且掌握其基础用法

3. 实验内容

Task 1: Prerequisites

了解Linux，Python，Git的基本使用方式

Task 2: Workflow

安装并配置VSCode，Mininet，Wireshark，Switchyard，并且掌握其基础用法

Task 3: Hand in & Find answers

Task 4: Modification

- 修改mininet拓扑逻辑
- 为myhub中收发数据包计数
- 使用框架代码中的new_pkt函数创建新的包
- 运行mininet
- 用wireshark抓包

4. 实验结果

Step 1: Modify the Mininet topology

在本节中，删除了 `start_mininet.py` 拓扑逻辑中的 `server2`

```
...
nodes = {
    "server1": {
        "mac": "10:00:00:00:00:{:02x}",
        "ip": "192.168.100.1/24"
    },
    # "server2": {
    #     "mac": "20:00:00:00:00:{:02x}",
    #     "ip": "192.168.100.2/24"
    # },
...
```

```
mininet> nodes
available nodes are:
client hub server1
```

Step 2: Modify the logic of a device

在本节中，需要统计 `myhub.py` 中接收到的包和转发出去的包有多少个，所以需要自己定义两个变量 `ingress_pkt_num, egress_pkt_num` 来计数，并且在适当的位置使它们的值加一

何时修改 ingress_pkt_num 的值

显见，只有通过异常处理的recv_packet才会导致 `in+=1`

```
try:
    _, fromIface, packet = net.recv_packet()
except NoPackets:
    continue
except Shutdown:
    break

ingress_pkt_num += 1
```

何时修改egress_pkt_num的值

从框架代码可以看出从至少应该在此之后，才会从其他端口转发包。

```
eth = packet.get_header(Ethernet)
```

只要清楚接下来代码所对应的三个if条件之后都在做什么即可明确应该在哪一步修改 `egress_pkt_num` 的值

- **第一个if条件**

```
if eth is None:
```

第一个条件非常简单，如果这个包的类型不对则不需要处理

- **第二个if条件**

```
if eth.dst in mymacs:
```

第二个条件是判断eth.dst属不属于mymacs，这就要弄清楚mymacs中所包含的是什么内容。mymacs的定义如下：

```
my_interfaces = net.interfaces()
mymacs = [intf.ethaddr for intf in my_interfaces]
```

可以看出mymacs是hub中所涉及的所有端口的物理地址的list。所以第二个条件判断的是包的目的地址是不是指向hub的物理地址

所以综上所述 `egress_pkt_num` 的值应该在else中修改，并在每种情况下输出 `ingress_pkt_num, egress_pkt_num` 的值，即：

```
if eth is None:
    log_info("Received a non-Ethernet packet?!")
    log_info(f"in:{ingress_pkt_num} out:{egress_pkt_num}")
    return
if eth.dst in mymacs:
    log_info("Received a packet intended for me")
    log_info(f"in:{ingress_pkt_num} out:{egress_pkt_num}")
else:
    for intf in my_interfaces:
        if fromIface!= intf.name:
            egress_pkt_num += 1
            log_info (f"Flooding packet {packet} to {intf.name}")
            log_info(f"in:{ingress_pkt_num} out:{egress_pkt_num}")
            net.send_packet(intf, packet)
```

Step 3: Modify the test scenario of a device

本节是在 `myhub_testscenario.py` 中使用 `new_packet` 函数创建自己的示例，创建的示例具体如下：

```
mypkt = new_packet(
    "20:00:00:00:00:01",
    "ff:ff:ff:ff:ff:ff",
    "192.168.1.100",
    "255.255.255.255"
)
s.expect(
    PacketInputEvent("eth2", mypkt, display=Ethernet),
    ("An Ethernet frame with a broadcast destination address should arrive on
eth2")
)
s.expect(
    PacketOutputEvent("eth0", mypkt, "eth1", mypkt, display=Ethernet),
    ("The Ethernet frame with a broadcast destination address should be
forwarded out ports eth0 and eth1")
)
```

`PacketInputEvent` 是从20:00:00:00:00:01物理地址广播，将会到达eth2端口

`PacketOutputEvent` 从eth0和eth1端口转发mypkt

Step 4: Run your device in Mininet

Running in the Test Environment

使用指令测试

```
$ swyard -t examples/myhub_testscenario.py examples/myhub.py
```

```
(syenv) njucs@njucs-VirtualBox:~/sy/lab-1-vectormoon$ swyard -t testcases/myhub_testscenario.py myhub.py
22:17:30 2021/03/19     INFO Starting test scenario testcases/myhub_testscenario.py
22:17:30 2021/03/19     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
22:17:30 2021/03/19     INFO in:1 out:1
22:17:30 2021/03/19     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
22:17:30 2021/03/19     INFO in:1 out:2
22:17:30 2021/03/19     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
22:17:30 2021/03/19     INFO in:2 out:3
22:17:30 2021/03/19     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
22:17:30 2021/03/19     INFO in:2 out:4
22:17:30 2021/03/19     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth0
22:17:30 2021/03/19     INFO in:3 out:5
22:17:30 2021/03/19     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth2
22:17:30 2021/03/19     INFO in:3 out:6
22:17:30 2021/03/19     INFO Received a packet intended for me
22:17:30 2021/03/19     INFO in:4 out:6
22:17:31 2021/03/19     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 192.168.1.100->255.255.255.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
22:17:31 2021/03/19     INFO in:5 out:7
22:17:31 2021/03/19     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 192.168.1.100->255.255.255.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
22:17:31 2021/03/19     INFO in:5 out:8

Results for test scenario hub tests: 10 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
    flooded out eth1 and eth2
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
    flooded outeth0 and eth2
7   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
8   The hub should not do anything in response to a frame
    arriving with a destination address referring to the hub
    itself.
9   An Ethernet frame with a broadcast destination address
    should arrive on eth2
10  The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0 and eth1


All tests passed!
```

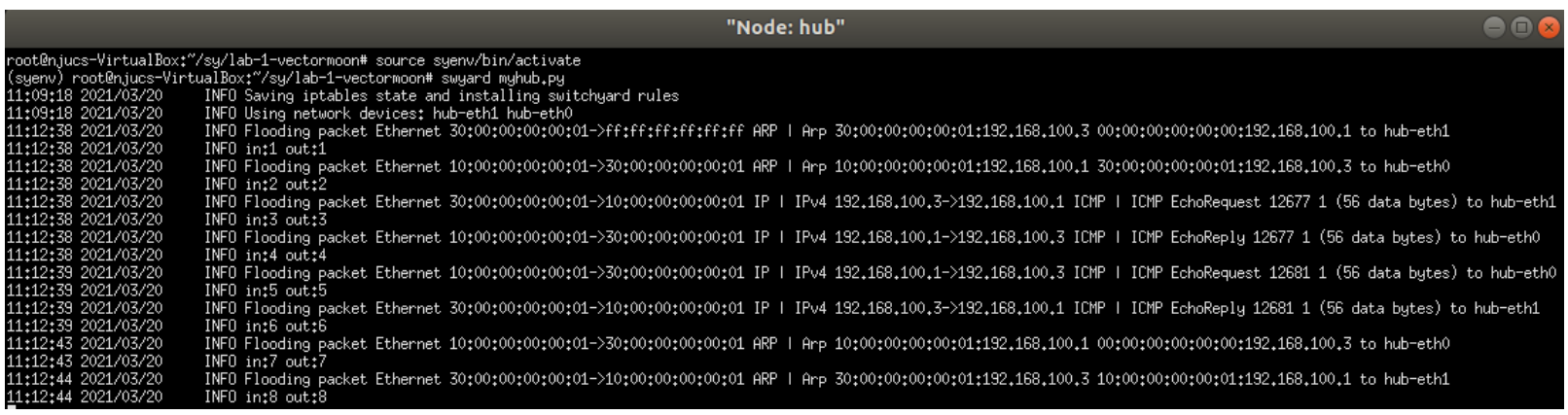
Running in the Mininet

使用如下指令打开Mininet

```
$ sudo python examples/start_mininet.py
mininet> xterm hub
```

之后在mininet中输入pingall指令观察terminal和xterm中有什么变化

在xterm中



在terminal中

```
mininet> pingall
*** Ping: testing ping reachability
client -> X server1
hub -> X X
server1 -> client X
*** Results: 66% dropped (2/6 received)
```

之所以结果会是66% dropped是因为hub只是集线器，在源码的实现中也没有分配ip地址

Step 5: Capture using Wireshark

在mininet依次输入

```
mininet> client wireshark &
mininet> client ping -c1 server1
```

第二行指令会创造一些流量，下面分析wireshark中第一个包所展示的结果：

1. 下图蓝色部分前六个字节0xffffffffffff代表目的地址，因为client不知道server1的物理地址所以要先广播找到server1的物理地址；之后紧接着的六个字节0x300000000000001表示源地址。在之后0x0806表示的是使用的ARP协议。

```
0000  ff 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 00 01 c0 a8 64 03   ......0· ......d·
0020  00 00 00 00 00 00 c0 a8  64 01                     ........ d·
```

上面提及到的源地址和目的地址详情中又涉及到 `LG bit` 和 `IG bit`

```
▼ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
     Address: Broadcast (ff:ff:ff:ff:ff:ff)
     .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
     .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
▼ Source: 30:00:00:00:00:01 (30:00:00:00:00:01)
     Address: 30:00:00:00:00:01 (30:00:00:00:00:01)
     .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
     .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
```

其中IG位对应MAC地址的第8位，用于区分单播地址和组播地址；而LG位对应MAC地址的第7位，它的作用就是做一个特殊标识，如果有关部门或者政府想让这个设备“特殊”一点，可以把这一位置为1，用来区别于其他设备。

2. 下面蓝色部分前两个字节0x0001表示使用硬件类型是以太网，协议类型0x0800为IPv4，紧接着的0x06和0x04分别表示硬件地址长度和协议地址长度；操作类型是0x0001为request操作；剩下的剩余部分，分别对应发送方的mac地址和ip地址以及接收方的mac地址和ip地址

```
▼ Address Resolution Protocol (request)
     Hardware type: Ethernet (1)
     Protocol type: IPv4 (0x0800)
     Hardware size: 6
     Protocol size: 4
     Opcode: request (1)
     Sender MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
     Sender IP address: 192.168.100.3
     Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
     Target IP address: 192.168.100.1
```

```
0000  ff 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 00 01 c0 a8 64 03   ......0· ......d·
0020  00 00 00 00 00 00 c0 a8  64 01                     ........ d·
```