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（方向）	**计算机科学与技术**
学号	**191220154**	姓名	**张涵之**
Email	**1683762615@qq.com**	开始/完成日期	**2021/3/19 – 2021/3/21**

1. 实验名称：Lab 1: Switchyard & Mininet

2. 实验目的：

a) Setting up the environment;
b) Getting familiar with the tools: Switchyard, Mininet, Wireshark, Git, etc.;
c) Acquiring a general impression on the virtual network structures, learning how to construct nodes, write testcases and capture files.

3. 实验内容

a) Step 1: Modify the Mininet topology
(I choose to delete server2 in the topology);
b) Step 2: Modify the logic of a device
Count how many packets pass through a hub in and out, log the statistical result every time one packet is received;
c) Step 3: Modify the test scenario of a device
(I choose to create a test case using new_packet with different arguments);
d) Step 4: Run your device in Mininet
Run the new hub in the new topology and make sure it works;
e) Step 5: Capture using Wireshark
Save the capture file and describe the details of it capture file;

4. 实验结果

a) Step 1: Deleted server2 in the topology

```
(syenv) njucs@njucs-VirtualBox:~/switchyard$ sudo python examples/
[sudo] password for njucs:
*** Creating network
*** Adding hosts:
client hub server1
*** Adding switches:
```

```
mininet> nodes
available nodes are:
client hub server1
mininet> links
client-eth0<->hub-eth0 (OK OK)
server1-eth0<->hub-eth1 (OK OK)
```

b) Step 2: Counting packets and logging statistics
(See more results of log information in Step 4)

```
(syenv) njucs@njucs-VirtualBox:~/switchyard/examples$ swyard -t myhub_testscenario.py
17:16:20 2021/03/20     INFO Starting test scenario myhub_testscenario.py
17:16:20 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:f
EchoRequest 0 0 (0 data bytes) to eth0
17:16:20 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:f
EchoRequest 0 0 (0 data bytes) to eth2
17:16:20 2021/03/20     INFO in:1 out:2
17:16:20 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:0
hoRequest 0 0 (0 data bytes) to eth1
17:16:20 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:0
hoRequest 0 0 (0 data bytes) to eth2
17:16:20 2021/03/20     INFO in:2 out:4
17:16:20 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:0
hoReply 0 0 (0 data bytes) to eth0
17:16:20 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:0
hoReply 0 0 (0 data bytes) to eth2
17:16:20 2021/03/20     INFO in:3 out:6
17:16:20 2021/03/20     INFO Received a packet intended for me
17:16:20 2021/03/20     INFO in:4 out:6

Results for test scenario hub tests: 8 passed, 0 failed, 0 pending
```

c) Step 3: Created new test case and passed successfully

```
(syenv) njucs@njucs-VirtualBox:~/switchyard$ swyard -t lab-1-RainTreeCrow/testcases/myhub_testscenario.py lab-1-
21:49:29 2021/03/20     INFO Starting test scenario lab-1-RainTreeCrow/testcases/myhub_testscenario.py
21:49:29 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2
EchoRequest 0 0 (0 data bytes) to eth0
21:49:29 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2
EchoRequest 0 0 (0 data bytes) to eth2
21:49:29 2021/03/20     INFO in:1 out:2
21:49:29 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.1
hoRequest 0 0 (0 data bytes) to eth1
21:49:29 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.1
hoRequest 0 0 (0 data bytes) to eth2
21:49:29 2021/03/20     INFO in:2 out:4
21:49:29 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2
hoReply 0 0 (0 data bytes) to eth0
21:49:29 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2
hoReply 0 0 (0 data bytes) to eth2
21:49:29 2021/03/20     INFO in:3 out:6
21:49:29 2021/03/20     INFO Received a packet intended for me
21:49:29 2021/03/20     INFO in:4 out:6
21:49:30 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 192.168.1.1
P EchoRequest 0 0 (0 data bytes) to eth1
21:49:30 2021/03/20     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 192.168.1.1
P EchoRequest 0 0 (0 data bytes) to eth2
21:49:30 2021/03/20     INFO in:5 out:8

Results for test scenario hub tests: 10 passed, 0 failed, 0 pending
```

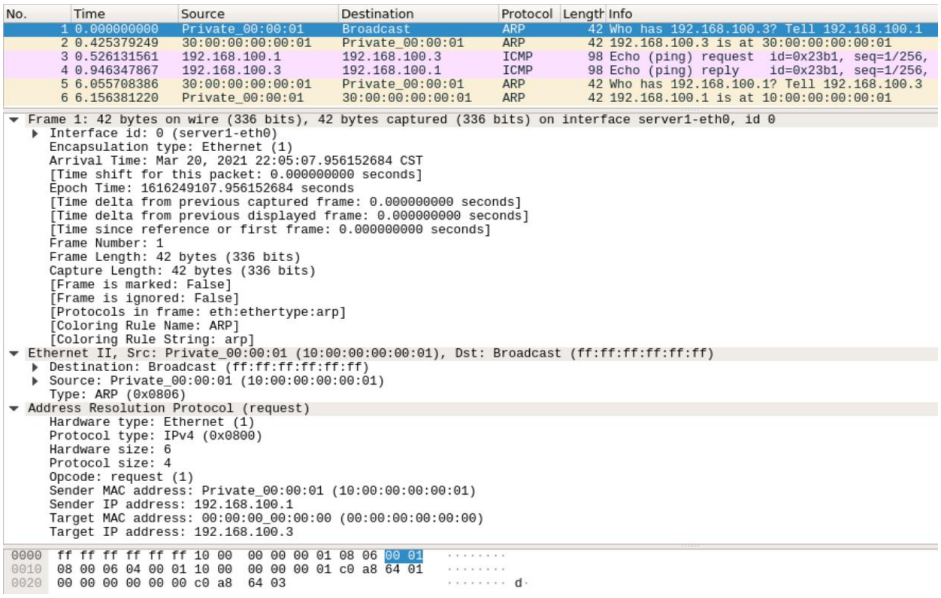
```
8   The hub should not do anything in response to a frame
    arriving with a destination address referring to the hub
    itself.
9   An Ethernet frame with a broadcast destination address
    should arrive on eth0
10  The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth1 and eth2


All tests passed!
```

d) Step 4: Running the new hub in Mininet

```
"Node: hub"
17:04:17 2021/03/20     INFO Saving iptables state and installing switchyard rul
es
17:04:17 2021/03/20     INFO Using network devices: hub-eth1 hub-eth2 hub-eth0
17:04:23 2021/03/20     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 hub-eth1
17:04:23 2021/03/20     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 hub-eth2
17:04:23 2021/03/20     INFO in:1 out:2
17:04:23 2021/03/20     INFO Flooding packet Ethernet 10:00:00:00:00:01->30:00:0
0:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 30:00:00:00:00:01:192.168.1
00.3 to hub-eth2
17:04:23 2021/03/20     INFO Flooding packet Ethernet 10:00:00:00:00:01->30:00:0
0:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 30:00:00:00:00:01:192.168.1
00.3 to hub-eth0
17:04:23 2021/03/20     INFO in:2 out:4
17:04:24 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:01->10:00:0
0:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoRequest 6168 1
 (56 data bytes) to hub-eth1
17:04:24 2021/03/20     INFO Flooding packet Ethernet 30:00:00:00:00:01->10:00:0
0:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoRequest 6168 1
 (56 data bytes) to hub-eth2
17:04:24 2021/03/20     INFO in:3 out:6
```

The testing process involves running start_mininet.py, which set up the nodes client, hub and server1, if try to ping at once, the drop rate would be 100%, as the links are not established yet. If ping after running myhub.py in xterm hub window, however, only the links between nodes EXCEPT hub can transfer file successfully, for hub does noy have an IP address and cannot be pinged.

e) Step 5: Capturing files with Wireshark



This is the file captured after running “server1 ping -c client”, From the file I can see the interface id, arrival time, capture length of the frame, as well as the source and destination, the MAC and IP address of the sender and the target, I may as well infer that the packet is sent using broadcast.

5. 核心代码

a) Step 1: Delete server2 by annotating relevant code;
This is the only place where “server2” is mentioned, when the nodes are used the code is often written “for node in nodes”, so I assume deleting it from the array is enough. Further testing shows it IS enough.

```
25  nodes = {
26      "server1": {
27          "mac": "10:00:00:00:00:{:02x}",
28          "ip": "192.168.100.1/24"
29      },
30      #"server2": {
31      #    "mac": "20:00:00:00:00:{:02x}",
32      #    "ip": "192.168.100.2/24"
33      #},
34      "client": {
35          "mac": "30:00:00:00:00:{:02x}",
36          "ip": "192.168.100.3/24"
37      },
```

b) Step 2: Adding counters to count packets and logging statistics;
The statistical result is logged every time a packet is received with the format of each line in:<ingress packet count> out:<egress packet count>.

```
10  def main(net: switchyard.llnetbase.LLNetBase):
11      my_interfaces = net.interfaces()
12      mymacs = [intf.ethaddr for intf in my_interfaces]
13      in_counter = 0
14      out_counter = 0
```

```
16      while True:
17          try:
18              _, fromIface, packet = net.recv_packet()
19              in_counter += 1
20          except NoPackets:
21              continue
22          except Shutdown:
23              break
```

```
34                  if fromIface!= intf.name:
35                      log_info (f"Flooding packet {packet} to {intf.name}")
36                      net.send_packet(intf, packet)
37                      out_counter += 1
38          log_info ("in:{} out:{}".format(in_counter, out_counter))
```

c) Step 3: Creating new test case

From the example testcases I can infer there are three types of cases, they are broadcast, any unicast address and a dest address among the interfaces. I chose to write another testcase using broadcast destination, which arrives on eth0 and was forwarded out through ports eth1 and eth2.

```
96      #test case 4(new): a frame with broadcast destination should get sent out
97      # all ports except ingress
98      testpkt = new_packet(
99          "20:00:00:00:00:01",
100         "ff:ff:ff:ff:ff:ff",
101         "192.168.1.100",
102         "255.255.255.255"
103     )
104     s.expect(
105         PacketInputEvent("eth0", testpkt, display=Ethernet),
106         ("An Ethernet frame with a broadcast destination address "
107          "should arrive on eth0")
108     )
109     s.expect(
110         PacketOutputEvent("eth1", testpkt, "eth2", testpkt, display=Ethernet),
111         ("The Ethernet frame with a broadcast destination address should be "
112          "forwarded out ports eth1 and eth2")
113     )
114     return s
```

6. 总结与感想

a) When trying to understand the structure of the virtual network and how the test cases work, it is useful to read the examples carefully, analyze the similarities and differences between them, and test the guessing through running.

b) Certain phenomenon deserve attention for there are rationales behind them, for example, the drop rate of command “pingall” differs before and after running myhub.py because of the links between nodes. Moreover, the rate itself has to do with the features of hub, that it works on a second level and does not have an IP address to make a packet. When coming across this kind or results during testing, I should pause to think about the logic behind them.