

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 助教：方毓楚/于沛文/郑浩/陈伟/李国浩/李睿宸/杨溢...
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：201220096	姓名：钟亚晨
Email: 2991908515@qq.com	开始/完成日期：2022.3.2-2022.3.4

1.实验名称

Switchyard & Mininet

2.实验目的

- ✧ 配置所需要的环境，包括：Linux、Python、Git、Switchyard、Mininet、Wireshark、VSC等；
- ✧ 熟悉VSC配置及使用、熟悉Mininet软件构建网络框架、Wireshark抓包工具、Switchyard框架；
- ✧ 对Mininet构建拓扑、Switchyard制作device、Wireshark抓包及分析进行简单练习；
- ✧ 熟悉阅读英文实验手册及Mininet文档、Switchyard文档、Wireshark第三方博客文档；

3.实验内容

- ✧ 完整阅读教程；
- ✧ 更改Mininet拓扑：删除 `server2` 或者更改为6节点的新拓扑；
- ✧ 更改hub实现逻辑，为其添加包计数，并通过日志输出；
- ✧ 修改hub的testscenario，添加自己新的测试用例；
- ✧ 在Mininet中运行新的hub，并确保其正常工作；
- ✧ 尝试使用Wireshark进行抓取数据包，制造流量并进行分析；

4.实验结果

所有组件正常工作，以下对实验说明中的五个问题进行回答。

均以目录下 `complex_done`（更改拓扑为6节点的新拓扑，同时更新testscenario）为基准进行报告，三节点的 `simple_done` 可很容易进行类比

0x01 show the details of how you build the topology in your report.

建立新的topo主要是更改文件 `start_mininet.py` 的内容：

通过阅读源代码可以了解到：

1. PySwitchTopo类通过继承mininet模块中的Topo类，重写了其构造函数，利用名为 `nodes` 的字典包含所有节点所需要的MAC地址以及IP，之后通过遍历字典调用 `addHost` 和 `addLink` 方法构建拓扑。
2. `main`函数中首先实例化PySwitchTopo对象，之后将其传入Mininet构造函数实例化出 `mininet` 对象，再通过封装好的 `setup_addressing` 和 `disable_ipv6` 来设定各个节点的MAC、IP地址以及禁用IPV6

则构建拓扑的一个容易的方法就是仿照已有节点，通过增加 `nodes` 字典中的键值对来修改/创建新的拓扑。而后的MAC、IP地址配置都会通过已有函数设置好。

```
23
24 nodes = {
25     "server1": {
26         "mac": "10:00:00:00:00:00:{:02x}",
27         "ip": "192.168.100.1/24"
28     },
29     "server2": {
30         "mac": "20:00:00:00:00:00:{:02x}",
31         "ip": "192.168.100.2/24"
32     },
33     "hub": {
34         "mac": "30:00:00:00:00:00:{:02x}",
35     },
36     "client1": {
37         "mac": "40:00:00:00:00:00:{:02x}",
38         "ip": "192.168.100.3/24"
39     },
40     "client2": {
41         "mac": "50:00:00:00:00:00:{:02x}",
42         "ip": "192.168.100.4/24"
43     },
44     "client3": {
45         "mac": "60:00:00:00:00:00:{:02x}",
46         "ip": "192.168.100.5/24"
47     }
48 }
49
```

这里我新增三个节点以构成六节点的不同于初始的新拓扑。

```
1      # server1      client1
2      #           \  /
3      #           hub---client2
4      #           /  \
5      # server2      client3
```

通过mininet的 `nodes`、`net`、`dump` 指令可以容易验证拓扑的成功构建：

```

*** Starting CLI:
mininet> nodes
available nodes are:
client1 client2 client3 hub server1 server2
mininet> net
client1 client1-eth0:hub-eth0
client2 client2-eth0:hub-eth1
client3 client3-eth0:hub-eth2
hub hub-eth0:client1-eth0 hub-eth1:client2-eth0 hub-eth2:client3-eth0 hub-eth3:s
erver1-eth0 hub-eth4:server2-eth0
server1 server1-eth0:hub-eth3
server2 server2-eth0:hub-eth4
mininet> dump
<Host client1: client1-eth0:192.168.100.3 pid=12400>
<Host client2: client2-eth0:192.168.100.4 pid=12402>
<Host client3: client3-eth0:192.168.100.5 pid=12404>
<Host hub: hub-eth0:10.0.0.4,hub-eth1:None,hub-eth2:None,hub-eth3:None,hub-eth4:
None pid=12406>
<Host server1: server1-eth0:192.168.100.1 pid=12408>
<Host server2: server2-eth0:192.168.100.2 pid=12410>
mininet>

```

0x02 Then show the log of your hub when running it in Mininet and how you implement it in your report.

通过阅读 `myhub.py` 对hub的实现源码，我们可以很明显的通过已有的log_info/log_debug来实现所需要的功能：

```

5
6 import switchyard
7 from switchyard.lib.userlib import *
8
9
10 def main(net: switchyard.llnetbase.LLNetBase):
11     my_interfaces = net.interfaces()
12     mymacs = [intf.ethaddr for intf in my_interfaces]
13
14     in_num = 0
15     out_num = 0
16     while True:
17         try:
18             _, fromIface, packet = net.recv_packet()
19         except NoPackets:
20             continue
21         except Shutdown:
22             break
23
24         log_debug(f"In {net.name} received packet {packet} on {fromIface}")
25         eth = packet.get_header(Ethernet)
26         if eth is None:
27             in_num = in_num + 1
28             log_info(f"in: {in_num} out: {out_num}")
29             log_info("Received a non-Ethernet packet?!")
30             return
31         if eth.dst in mymacs:
32             log_info("Received a packet intended for me")
33             in_num = in_num + 1
34         else:
35             in_num = in_num + 1
36             for intf in my_interfaces:
37                 if fromIface != intf.name:
38                     out_num = out_num + 1
39                     log_info(f"Flooding packet {packet} to {intf.name}")
40                     net.send_packet(intf, packet)
41             log_info(f"in: {in_num} out: {out_num}")
42
43     net.shutdown()
44

```

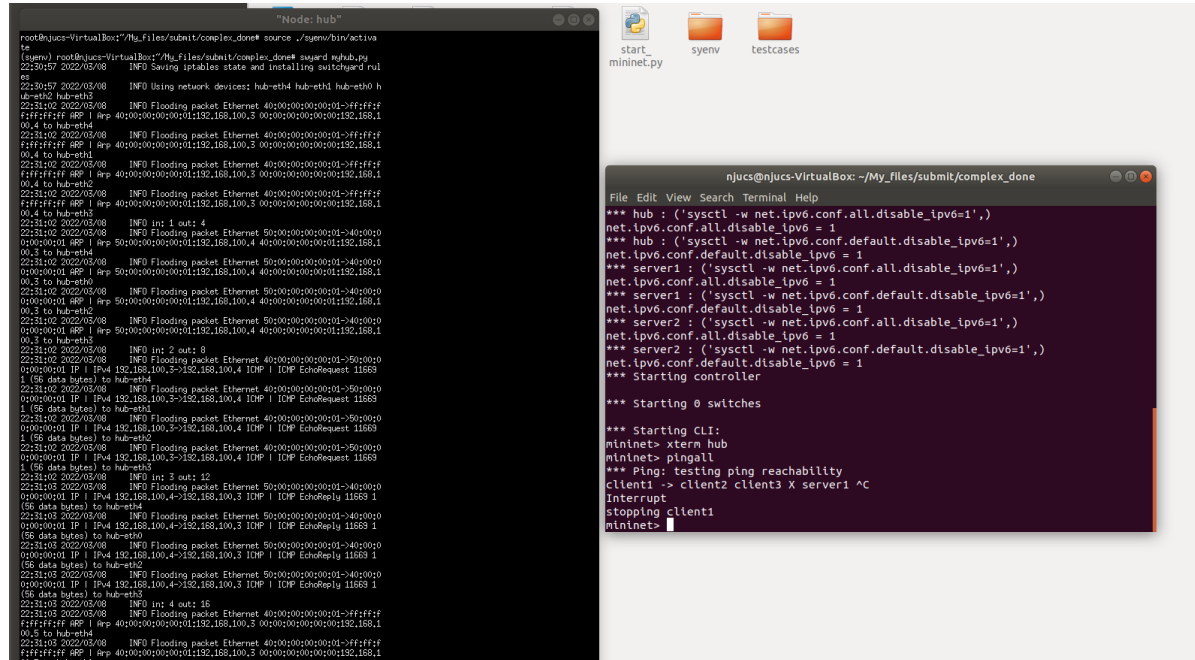
在第15、16行，我们定义了两个变量 `in_num` 和 `out_num` 并且在开始时置为0，分别用于累计每一次接收、发送包的数量；

在第27行，在接收到非因特网包的情况下，我们收到了一个包，将 `in_num` 增加1，并且将信息通过 `log_info` 进行打印，而后函数返回，hub崩溃；

在第33行，在接收到一个发送给hub的包时，我将 `in_num` 增加1；

在第35、38行，hub接收到了一个并不是发送给hub，但是经过hub的包，此时首先将 `in_num` 增加一，表示hub从入端口接收到了一个包，而后在for循环中，当端口不是入端口时，发送包，并且将 `out_num` 增加一，表示hub多向外发送一个包；

在第41行，即一次循环结束，我们通过 `log_info` 打印信息，将hub接收/发送包的数量进行报告。



在hub执行了其逻辑代码后效果如下，使用 `pingall` 制造流量，则每个包进入hub后会向除入端口外的其它端口进行广播，这里hub连接了五个hosts，因而一个入端口每次接收到一个数据包，而其余的四个出端口则一共发送出四个数据包，并且以 `in:<in_num> out:<out_num>` 的形式进行了log显示。

0x03 Then show the details of your test cases in your report.

由于这里使用了全新的拓扑，因而原有的testcase需要进行更改，一下进行了一些简单的更改：

- 1.通过修改/增加 `add_interface` 调用次数增加了端口数量；
- 2.对于已有的类似广播等的测试用例，在接收方、发送方将新增的三个端口及包添加到了测试函数当中

另外，通过已有的 `new_packet` 生成了新的包，该包为一个传送给hub的广播，hub在接收到这个广播后应当什么也不做：

```

107 # test case mine: the hub received a packet which be boardcast
108 # do nothing
109 mypkt1 = new_packet(
110     "ff:ff:ff:ff:ff:ff",
111     "10:00:00:00:00:03",
112     '192.168.1.100',
113     '172.16.42.2'
114 )
115 s.expect(
116     PacketInputEvent("eth2", mypkt1, display=Ethernet),
117     ("received a boardcast packet")
118 )
119 s.expect(
120     PacketInputTimeoutEvent(1.0),
121     ("hub do nothing")
122 )

```

而后通过 `swyard -t ./testcases/myhub_testscenario.py myhub.py` 指令，进行测试，能够通过所有的测试用例：

```

nJucs@nJucs-VirtualBox: ~/My_files/submit/complex_done
21:01:20 2022/03/03 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
21:01:20 2022/03/03 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 eth3
21:01:20 2022/03/03 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 eth4
21:01:20 2022/03/03 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 eth5
21:01:20 2022/03/03 INFO In: 1 out: 5
21:01:20 2022/03/03 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
21:01:20 2022/03/03 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
21:01:20 2022/03/03 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 eth3
21:01:20 2022/03/03 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 eth4
21:01:20 2022/03/03 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 eth5
21:01:20 2022/03/03 INFO In: 1 out: 5
21:01:20 2022/03/03 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
21:01:20 2022/03/03 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
21:01:20 2022/03/03 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 eth3
21:01:20 2022/03/03 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 eth4
21:01:20 2022/03/03 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 eth5
21:01:20 2022/03/03 INFO In: 1 out: 5
21:01:20 2022/03/03 INFO Received a packet intended for me
21:01:20 2022/03/03 INFO In: 1 out: 0
21:01:21 2022/03/03 INFO Received a packet intended for me
21:01:21 2022/03/03 INFO In: 1 out: 0

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

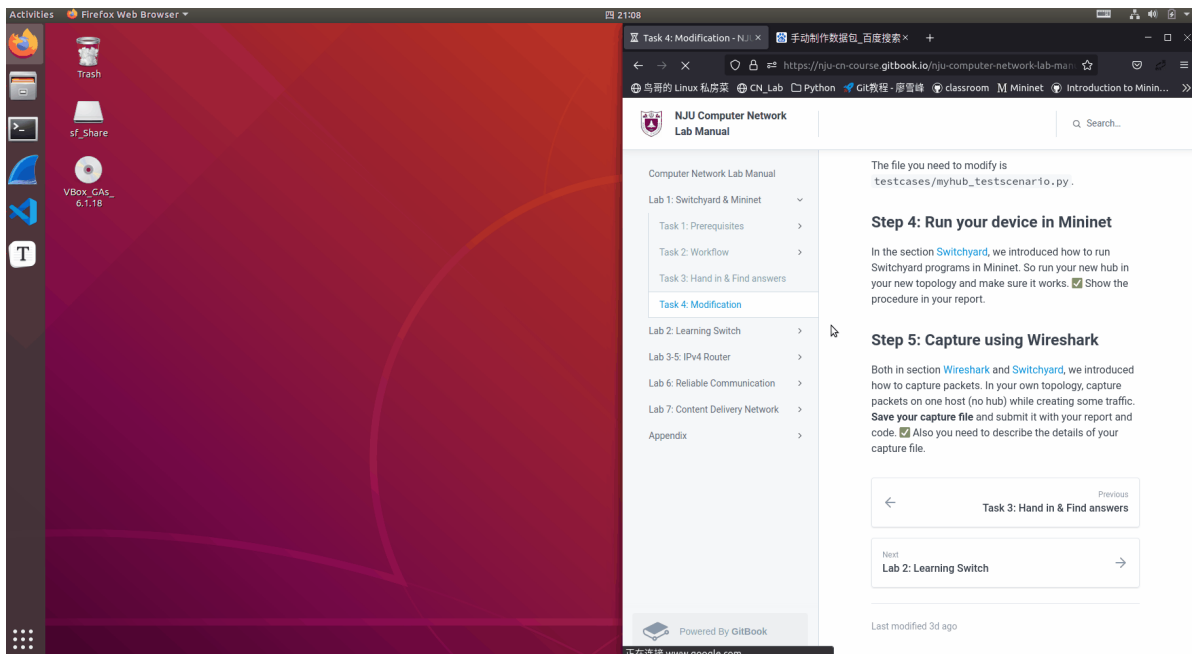
Passed:
1 An Ethernet frame with a broadcast destination address
  should arrive on eth1
2 The Ethernet frame with a broadcast destination address
  should be forwarded out ports eth0, eth2, eth3, eth4, eth5
3 An Ethernet frame from 20:00:00:00:00:01 to
  30:00:00:00:00:02 should arrive on eth0
4 Ethernet frame destined for 30:00:00:00:00:02 should be
  flooded out eth0, eth2, eth3, eth4, eth5
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 out eth0, eth2, eth3, eth4, eth5
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 received a boardcast packet
10 hub do nothing

All tests passed!
(syenv) nJucs@nJucs-VirtualBox:~/My_files/submit/complex_done$

```

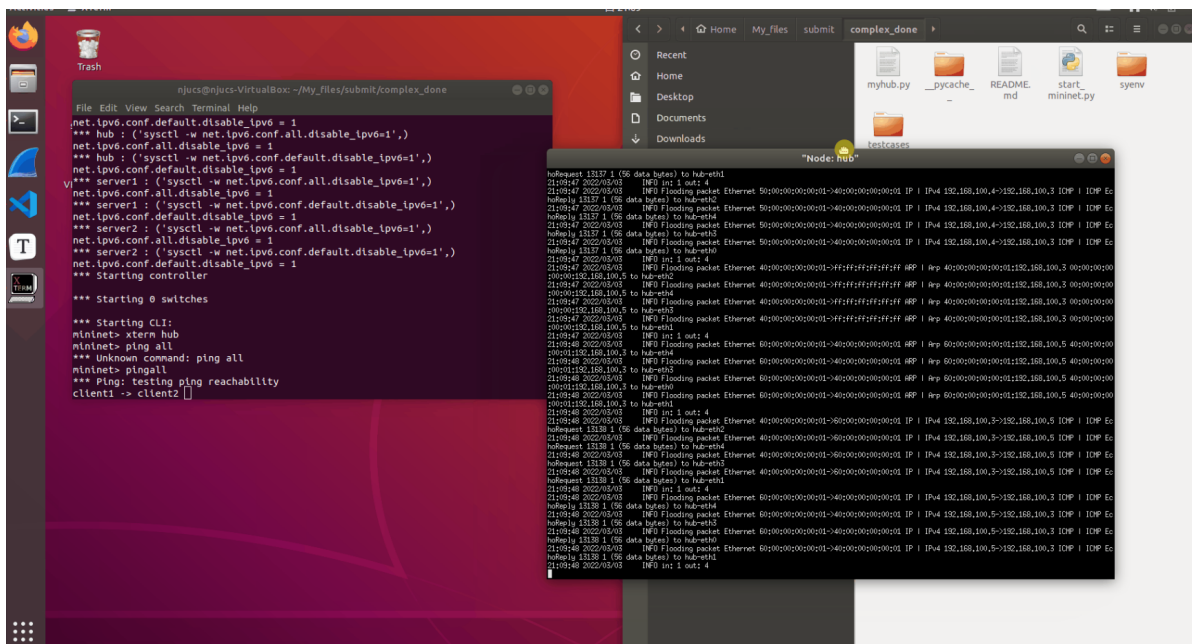
0x04 ✓ Run your new hub in your new topology and make it works. Show the procedure in your report.

这里使用gif图片进行过程演示，并且利用 `pingall` 来制造流量以及使用测试用例来进行单轮测试。（该演示中hub记录的是每次收发包的数量而非累计数量）



若无法在Markdown文件中正常查看gif文件，则可在目录./CN_Lab1实验报告/test1.gif中找到文件对整个运行过程进行查看。

另外附上图片演示的过程：




通过xterm将 hub 的终端转发出来，由于这里是六节点的拓扑，hub连接了其余五个节点，则 pingall后每次接收包都为接收1，其余4个端口发送包，和图片所演示的in/out一致，说明hub是正常工作的。

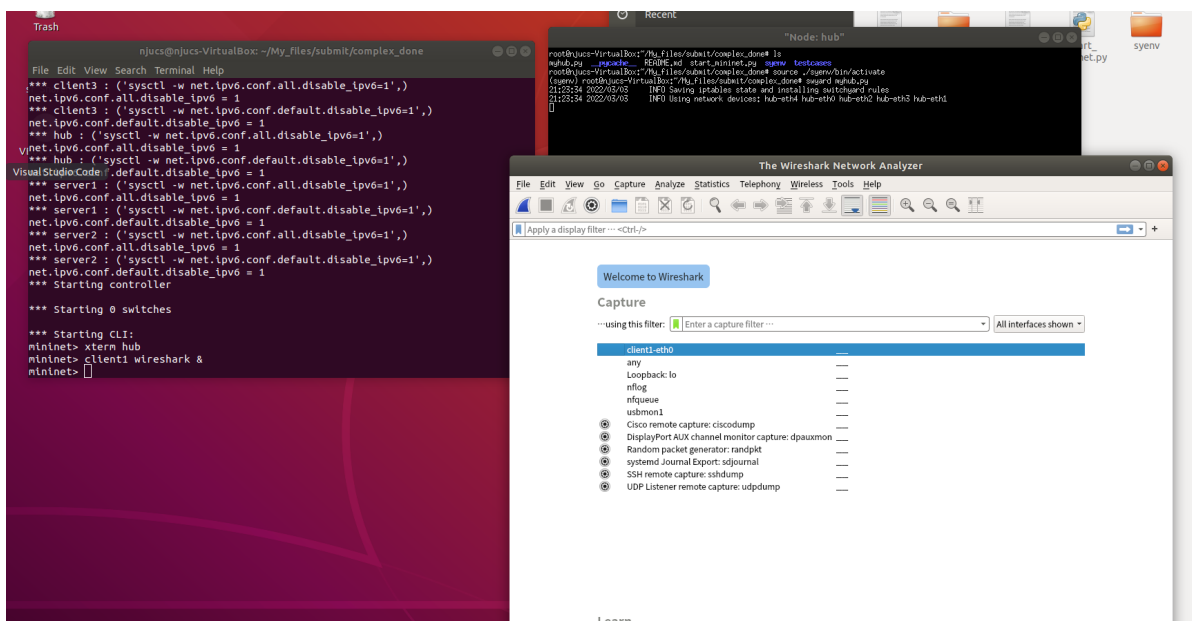

```
njucs@njucs-VirtualBox: ~/My_files/submit/complex_done
File Edit View Search Terminal Help
*** server1 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** server2 : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** server2 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** Starting controller

*** Starting 0 switches

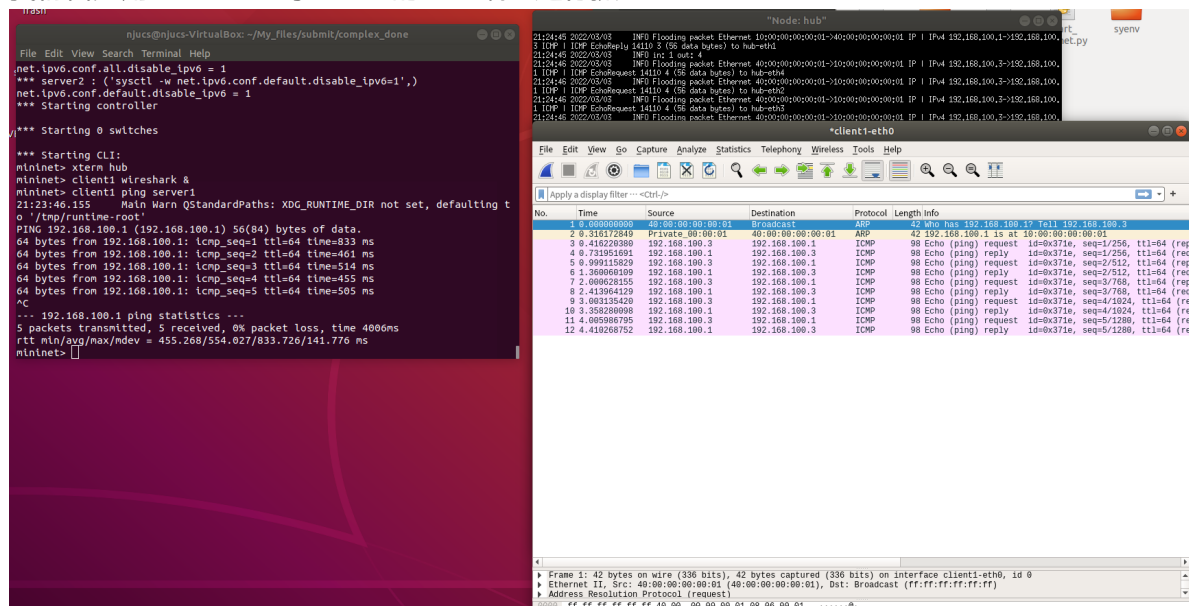
*** Starting CLI:
mininet> xterm hub
mininet> ping all
*** Unknown command: ping all
mininet> pingall
*** Ping: testing ping reachability
client1 -> client2 client3 X server1 server2
client2 -> client1 client3 X server1 server2
client3 -> client1 client2 X server1 server2
hub -> X X X X X
server1 -> client1 client2 client3 X server2
server2 -> client1 client2 client3 X server1
*** Results: 33% dropped (20/30 received)
mininet>
```

并且在pingall后，可以查看结果，发现其余5个节点相互均可以通信，两两通信共20次，故而所实现的hub是可以正常工作的。

0x05  Save your capture file and submit it with your report and code. Also you need to describe the details of your capture file.



我们首先用wireshark对client1的eth0端口进行抓包



之后我们在client1上运行指令ping server1，由client1向server1发送数据包。在很短时间后我们使用Ctrl+C终止了ping的过程。我们获得了如下数据包：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	0.316172849	Private 00:00:00:01	40:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.416220380	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x371e, seq=1/256, ttl=64 (req)
4	0.731951691	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x371e, seq=1/256, ttl=64 (rep)
5	0.999115829	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x371e, seq=2/512, ttl=64 (req)
6	1.360069109	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x371e, seq=2/512, ttl=64 (rep)
7	2.000628155	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x371e, seq=3/768, ttl=64 (req)
8	2.413964129	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x371e, seq=3/768, ttl=64 (rep)
9	3.003135420	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x371e, seq=4/1024, ttl=64 (req)
10	3.358280098	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x371e, seq=4/1024, ttl=64 (rep)
11	4.005986795	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x371e, seq=5/1280, ttl=64 (req)
12	4.410268752	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x371e, seq=5/1280, ttl=64 (rep)

首先由client1发送一个广播，寻问IP地址为192.168.100.1的设备也就是server1的MAC地址，并且由匹配的设备进行返回，可以看到这两个数据包都是使用的ARP协议。

之后client1获取到了server1的MAC地址，建立起连接后由client1和server1之间相互发送request/reply数据包，ping过程我们在Shell中看到的一条一条信息就是因为这些数据包的相互传输所形成的。可以看到ping产生的流量总是从IP地址192.168.100.3 (client1) -> 192.168.100.1 (server1)，再从192.168.100.1 (server1) -> 192.168.100.3 (client1) 由request和reply交替进行，均使用ICMP因特网控制报文协议来报告主机是否可达。



在下方我们可以看到传输数据的为48 bytes的具体内容，以及使用ICMP因特网控制报文协议所追加的报文头，其中指定了类型为8，即节点发送回显答复消息响应ICMP回显消息，而后校验和为0xe44a，校验状态表述该数据包传输完好。


```
Address: Private_00:00:01 (10:00:00:00:00:01)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ..0 .... = IG bit: Individual address (unicast)
▼ Source: 40:00:00:00:00:01 (40:00:00:00:00:01)
Address: 40:00:00:00:00:01 (40:00:00:00:00:01)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ..0 .... = IG bit: Individual address (unicast)
Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.100.3, Dst: 192.168.100.1
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
▼ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
0000 00.. = Differentiated Services Codepoint: Default (0)
.... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
Total Length: 84
Identification: 0x7e6b (32363)
▼ Flags: 0x40, Don't fragment
0... .... = Reserved bit: Not set
.1... .... = Don't fragment: Set
..0. .... = More fragments: Not set
Fragment Offset: 0
Time to Live: 64
Protocol: ICMP (1)
Header Checksum: 0x72e8 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.100.3
Destination Address: 192.168.100.1
▼ Internet Control Message Protocol
0000 10 00 00 00 00 00 01 40 00 00 00 00 01 08 00 45 00 .....@.....F.
```

另外还可见其使用了IPv4，即IP协议第四版，可通过wireshark看见其中各个字段的内容

```
▼ Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface client1-eth0, id 0
▼ Interface id: 0 (client1-eth0)
Interface name: client1-eth0
Encapsulation type: Ethernet (1)
Arrival Time: Mar 3, 2022 21:24:43.820968439 CST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1646313883.820968439 seconds
[Time delta from previous captured frame: 0.100047531 seconds]
[Time delta from previous displayed frame: 0.100047531 seconds]
[Time since reference or first frame: 0.416220380 seconds]
Frame Number: 3
Frame Length: 98 bytes (784 bits)
Capture Length: 98 bytes (784 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:icmp:data]
[Coloring Rule Name: ICMP]
[Coloring Rule String: icmp || icmpv6]
▼ Ethernet II, Src: 40:00:00:00:00:01 (40:00:00:00:00:01), Dst: Private_00:00:01 (10:00:00:00:00:01)
▼ Destination: Private_00:00:01 (10:00:00:00:00:01)
Address: Private_00:00:01 (10:00:00:00:00:01)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ..0 .... = IG bit: Individual address (unicast)
▼ Source: 40:00:00:00:00:01 (40:00:00:00:00:01)
Address: 40:00:00:00:00:01 (40:00:00:00:00:01)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ..0 .... = IG bit: Individual address (unicast)
Type: IPv4 (0x0800)
0000 10 00 00 00 00 00 01 40 00 00 00 00 01 08 00 45 00 .....@.....E.
```

以及一些互联网帧的其它信息

5.核心代码

```
1 # start_mininet.py中更改拓扑为六节点新拓扑的代码
2 nodes = {
3     "server1": {
4         "mac": "10:00:00:00:00:{:02x}",
5         "ip": "192.168.100.1/24"
6     },
7     "server2": {
8         "mac": "20:00:00:00:00:{:02x}",
9         "ip": "192.168.100.2/24"
10    },
11    "hub": {
12        "mac": "30:00:00:00:00:{:02x}",
13    },
14    "client1": {
15        "mac": "40:00:00:00:00:{:02x}",
```

```

16         "ip": "192.168.100.3/24"
17     },
18     "client2": {
19         "mac": "50:00:00:00:00:{:02x}",
20         "ip": "192.168.100.4/24"
21     },
22     "client3": {
23         "mac": "60:00:00:00:00:{:02x}",
24         "ip": "192.168.100.5/24"
25     }
26 }

```

```

1  # myhub.py中为hub添加收发包计数功能的代码
2  def main(net: switchyard.llnetbase.LLNetBase):
3      my_interfaces = net.interfaces()
4      mymacs = [intf.ethaddr for intf in my_interfaces]
5
6      in_num = 0
7      out_num = 0
8      while True:
9          try:
10             _, fromIface, packet = net.recv_packet()
11             except NoPackets:
12                 continue
13             except Shutdown:
14                 break
15
16             log_debug (f"In {net.name} received packet {packet} on
17 {fromIface}")
18             eth = packet.get_header(Ethernet)
19             if eth is None:
20                 in_num = in_num + 1
21                 log_info (f"in: {in_num} out: {out_num}")
22                 log_info("Received a non-Ethernet packet?!")
23                 return
24             if eth.dst in mymacs:
25                 log_info("Received a packet intended for me")
26                 in_num = in_num + 1
27             else:
28                 in_num = in_num + 1
29                 for intf in my_interfaces:
30                     if fromIface != intf.name:
31                         out_num = out_num + 1
32                         log_info (f"Flooding packet {packet} to
33 {intf.name}")
34                         net.send_packet(intf, packet)
35                 log_info (f"in: {in_num} out: {out_num}")

```

```

1  # myhub_testscenario.py中新增的测试用例，通过new_packet制造一个发给hub的广
2 播，其效果应该与发给hub的普通数据包一致
3  # test case mine: the hub received a packet which be boardcast

```

```

3      # do nothing
4      mypkt1 = new_packet(
5          "ff:ff:ff:ff:ff:ff",
6          "10:00:00:00:00:03",
7          '192.168.1.100',
8          '172.16.42.2'
9      )
10     s.expect(
11         PacketInputEvent("eth2", mypkt1, display=Ethernet),
12         ("received a boardcast packet")
13     )
14     s.expect(
15         PacketInputTimeoutEvent(1.0),
16         ("hub do nothing")
17     )

```

6.总结与感想

总结:

- 👉 Wireshark为一个抓包工具，抓取Mininet中的数据包；Mininet为一个软件网络框架，用于在软件层面构建网络；Switchyard则是一个基础组件开发框架，提供了组件开发以及测试功能；
- 👉 掌握了VSC中.vscode下配置文件的简单使用和配置；
- 👉 熟练了Python虚拟环境的创建以及使用，不得不说，真的非常方便且优美；
- 👉 熟练掌握了Git和远程仓库的基本命令，可熟练完成推送和拉取；
- 👉 学会了翻墙 (x)

感想:

- 👉 Mininet官方文档、Switchyard官方文档以及Wireshark第三方博客文档非常有用，虽然全英文读起来较为困难，但信息比起中文博客可靠得多（宁愿读英文文档也不愿意浪费时间找中文博客了x
- 👉 刚开始实验时的一大难题就是github上不去，通过一段时间的摸索，现在已能熟练的进行科学上网，github访问很快！
- 👉 先前由于自己摸索已经对Git和github还有Python虚拟环境有一定了解，因此在初始接触到实验时还是有不少熟悉的内容，欢迎来踩博客 (x
- [Python虚拟环境 · pawx2's Studio](#)
- 👉 最后的最后，还是要勉励自己多花时间读英文文档以及计算机网络大黑书增强理论学习和知识储备，很多Mininet、Switchyard的接口以及计算机网络原理方面的东西还是有一定了解才能在应用的时候游刃有余，譬如自己在创建6节点拓扑时分配的IP地址就曾经因为间隔太小而报错，根本还是理论知识匮乏。
- 👉 配置环境的过程还是非常顺利且愉快的 (✓
- 👉 在首次完成实验后将虚拟环境文件夹进行了移动，随后导致了虚拟环境中的 `swyard` 不可用的问题，最后结合 `which` 指令以及其它一些信息，判断出可能是随意移动文件夹导致虚拟环境无法找到对应的 `swyard` 文件，最后通过重新生成配置虚拟环境完美解决问题。(x下次能跑就再也不乱移动文件夹折腾来折腾去了

