Computer Networks: lab01

> 任课教师：田臣　　助教：方毓楚， 郑浩等

学院	计算机科学与技术系	专业	计算机科学与技术
学号	201220154	姓名	王紫萁
Email	201220154@smail.nju.edu.cn	开始/完成时间	2022. 3.1/2022.3.4

1. 实验目的

装配实验环境， 熟悉基本工具（`python`，`vscode`，`switchyard`，`mininet`，`wireshark`）的用法。

2. 实验步骤

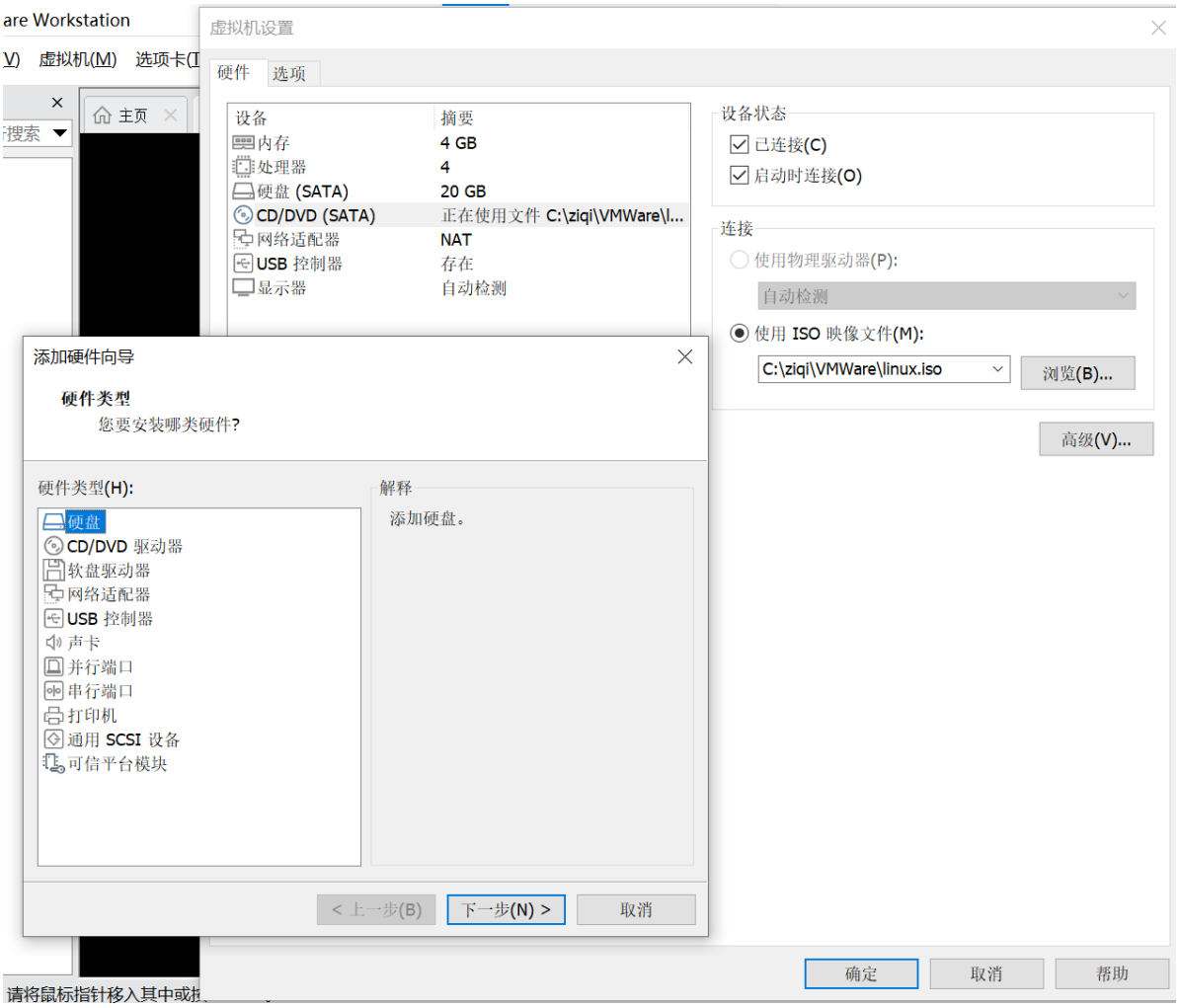
> 为方便起见，我将以`task5`为任务驱动说明实验步骤

2.1. 安装虚拟机

曾设想过在`WSL`上装配环境，但犹豫之下还是想要尽快步进到敲代码的环节，因此选用了VMWare + Ubuntu的方案，在解决虚拟机分辨率较低的问题上得到了较为有效且便捷的解决办法，在此稍作记录

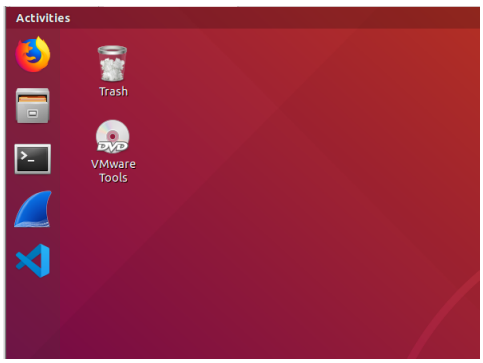
1. 安装光盘驱动

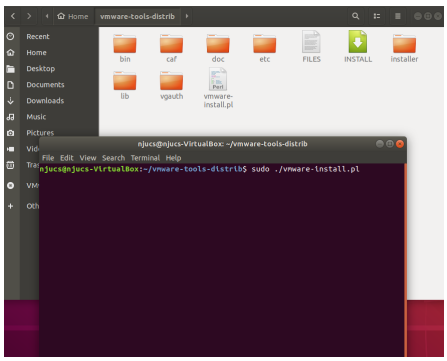
实验提供的虚拟机并没有光盘驱动程序，体现在在虚拟机上右键"安装`VMTools`"选项为不可选，于是我们需要 右键→ 虚拟机设置 →硬件→ 添加→CD/DVD驱动器→使用ISO镜像文件→并选中`VMware`安装路径下的`linux.iso`文件。



2. **安装VMTools**

导入成功后会在桌面出现光盘映像，双击并解压缩后在目录中管理员运行 `vmware-install.pl` 文件，除第一个选项为 `yes` 外其余选项均为默认即可





2.2. 搭建舒适的coding环境

1. **创建虚拟环境venv**

在当前目录下创建名为 `syenv` 的虚拟环境

```
$ python3 -m venv syenv
```

2. **配置python解释器（settings.json），执行设置（task.json），调试设置（ launch.json）**

在本课程的vscode配置文档中有详细的启动与调试文件配置的说明，所以在此不做赘述。

需要注意以下几个步骤：

1. 导入虚拟环境之后需要根据提示安装 `pylint`， `switchyard`， `mininet` 模块。后两个的指令如下：

   ```
   $ pip3 install mininet
   $ python3 -m pip install git+https://gitee.com/pavinberg/switchyard.git
   ```

   由于每次新建的虚拟环境时崭新的，并没有这两个模块的支持，因此需要在每一次配置完虚拟环境都安装以下。如果忘记安装也会因为报错找不到模块而提醒你某个模块的缺失

2. 将无关文件和文件夹记录到 `.gitignore` 中

2.3. step1. 使用mininet构建自己的topology

现阶段使用 `mininet` 时的几个关键步骤：

- 在框架代码中初始化节点（包括节点名称，类型，IP，MAC地址等信息），并构建链路，赋予主机与对应集线器的关联。

```
# node defination
nodes = {
    "server1": {
        "mac": "10:00:00:00:00:{:02x}",
        "ip": "192.168.100.1/24"
    },
    "server2": {
        "mac": "20:00:00:00:00:{:02x}",
        "ip": "192.168.110.1/24"
    },
    "client1": {
        "mac": "30:00:00:00:00:01",
        "ip": "192.168.100.2/24"
    },
    "client2":{
        "mac": "30:00:00:00:00:02",
        "ip": "192.168.110.2/24"
    },
    "hub1": {
        "mac": "40:00:00:00:00:{:02x}",
    },
    "hub2":{
        "mac":"50:00:00:00:00:{:02x}"
    }
}


class PySwitchTopo(Topo):

    def __init__(self, args):
        # Add default members to class.
```

```
        super(PySwitchTopo, self).__init__()

        # Host and link configuration
        #
        #   client1             client2
        #   -       \           /
        #           hub1---- hub2
        #          /            \
        #   server1             server2
        #

        nodeconfig = {"cpu": -1}

        for node in nodes.keys():
            self.addHost(node, **nodeconfig)

        #link the nodes
        # all links are 10Mb/s, 100 millisecond prop delay
        self.addLink("client1", "hub1", bw=10, delay="100ms")
        self.addLink("server1", "hub1", bw=10, delay="100ms")
        self.addLink("hub1", "hub2", bw=10, delay="100ms")
        self.addLink("client2", "hub2", bw=10, delay="100ms")
        self.addLink("server2", "hub2", bw=10, delay="100ms")
```

本次实验构建了一个以 `hub1` 和 `hub2` 为中心的纺锤形网络，其中 `client1` ， `server1` 在同一子网， `client2` ， `server2` 在同一子网，这一点在实验的后半部分比较重要。

2.4. step2. 使用switchyard构建自己的网络设备

每个网络设备都放在一个自己的专有文件下并且 `switchyard` 框架会搜索该文件中的 `main` 函数作为该部件执行的入口，我们在框架代码的基础上，改造 `myhub` 使得该集线器可以记录输入和输出的 `packet` 的个数并在每一次处理完后将 `in/out` 信息打印出来。

阅读 `myhub.py` 我们不难理解 `hub` 的工作原理： `hub` 记录了接口的 `list` 以及对应的 `mac` 地址，通过 `recv_packet()` 方法获取网络中的某些报文信息，对于目的地是自己接口的包他会将他保留，而其它的包会将他们全部分发到除包入口接口以外的其它所有接口上去。于是，我们在hub中定义两个变量 `cntIn = cntOut = 0` 在有包到来时 `cntIn += 1` ，有包 `flood` 出去时 `cntOut += 1` ，并在每一次循环结束后通过 `log_info()` 方法打印 `cntIn` 和 `cntOut` 信息即可。 (log的输出见2.6节)

值得注意的是 `egress packet` 在本次实验中指穿过出接口的包的个数，对于一个 `ingress` 的包应当在每个发出接口处部署计数而不是只对该包的发出计数一次。

2.5. step3. 编写testscenario测试样例

在 `testcase/` 目录下的 `myhub_testscenario.py` 编写给网络发送报文的样例，本次实验新加的样例对应主机在接入网络时请求 `DHCP` 赋予 `IP` 地址的情况，在该种情况下，报文的源、目的地址如下：

```
# test4: my test case: a new device asking for an IP assignment
    reqpkt = new_packet(
        "20:00:00:00:00:01",    #srcmac
        "ff:ff:ff:ff:ff:ff",    #dstmac
        "0.0.0.0",              #srcIP
        "255.255.255.255",      #dstIP
    )
```

一个新生接入设备需要广播自己的"未出生"事实以期有 `DHCP server` 给他赋予一个响亮的名字，于是集线器的任务自然就是将该报文发送到所有他能发送的地方（即它的所有端口）。因此当报文从eth0子网进入到接口时，集线器的 `packetOutPutEvent()` 应当将该包发送给 `eth1` ， `eth2` 相连的子网接口中。

2.6. step4. 运行测试用例，检查环境是否配置正确以及网络设备是否实现正确

我们将 `task.json` 和 `launch.json` 的 `testscenario` 路径更改为 `testcase/myhub_testscenario.py` ，于是构建和调试便能正常执行，同时myhub也同预想的一样正确输出了每一次入包和出包的数量信息（step2）

```
> Executing task: syenv/bin/python -m switchyard.swyard -t /home/njucs/netlab/lab-01-wayne-ziqi/testcases/myhub_testscenario.py /home/njucs/n
etlab/lab-01-wayne-ziqi/myhub.py <

21:38:43 2022/03/04     INFO Starting test scenario /home/njucs/netlab/lab-01-wayne-ziqi/testcases/myhub_testscenario.py
21:38:43 2022/03/04     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
21:38:43 2022/03/04     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
21:38:43 2022/03/04     INFO in: 1, out: 2
21:38:43 2022/03/04     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 E
choRequest 0 0 (0 data bytes) to eth1
21:38:43 2022/03/04     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 E
choRequest 0 0 (0 data bytes) to eth2
21:38:43 2022/03/04     INFO in: 2, out: 4
21:38:43 2022/03/04     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 E
choReply 0 0 (0 data bytes) to eth0
21:38:43 2022/03/04     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 E
choReply 0 0 (0 data bytes) to eth2
21:38:43 2022/03/04     INFO in: 3, out: 6
21:38:43 2022/03/04     INFO Received a packet intended for me
21:38:43 2022/03/04     INFO in: 4, out: 6
source /home/njucs/netlab/lab-01-wayne-ziqi/syenv/bin/activate
21:38:44 2022/03/04     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 0.0.0.0->255.255.255.255 ICMP | ICMP Ech
oRequest 0 0 (0 data bytes) to eth0
21:38:44 2022/03/04     INFO Flooding packet Ethernet 20:00:00:00:00:01->ff:ff:ff:ff:ff:ff IP | IPv4 0.0.0.0->255.255.255.255 ICMP | ICMP Ech
oRequest 0 0 (0 data bytes) to eth1
21:38:44 2022/03/04     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 should arrive on eth2 with ip request
    address
10  the packet should be broadcast to all interfaces


All tests passed!


Terminal will be reused by tasks, press any key to close it.
```

2.7. step5. 综合switchyard，mininet，wireshark抓包

2.7.1. 载入网络以及DIY的设备myhub

终于来到了激动人心的环节（），我们的集线器测试通过后就考虑将他应用到构建好的 `topology` 中。首先激活虚拟环境

```
$ source ./syenv/bin/activate
```

打开运行 `start_mininet.py` 部署我们的网络（前提是已经装配好了2.2.节的 `mininet` 并能成功运行）

```
$ sudo python3 start_mininet.py
```

成功的输出如下：

```
client1 client2 client3 hub server1 server2
server1 server1-eth0 10:00:00:00:00:01
server2 server2-eth0 20:00:00:00:00:01
client1 client1-eth0 30:00:00:00:00:01
client2 client2-eth0 30:00:00:00:00:02
client3 client3-eth0 30:00:00:00:00:03
hub hub-eth0 40:00:00:00:00:01
hub hub-eth1 40:00:00:00:00:02
hub hub-eth2 40:00:00:00:00:03
hub hub-eth3 40:00:00:00:00:04
hub hub-eth4 40:00:00:00:00:05
*** client1 : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** client1 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** client2 : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** client2 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** client3 : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** client3 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** hub : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
```

在 `xterm` 中打开需要自定义的设备 `hub1`

```
$ xterm hub1
```

并再次激活环境， `xterm` 窗口中载入我们自己的设备（前提是已经装配好了2.2.节的 `switchyard` 并能成功运行），在正式运行之前应当确保在 `swyard` 测试环境下通过（ `-t` 参数）

```
$ swyard myhub.py
```

`hub2` 同理重复一遍，

`pingall` 测试，将会把我们部署的所有可达的链路都测试一遍

```
$ mininet> pingall
```

测试成功,

打开 `wireshark` 执行抓包（我们以定义好的 `client1` 为节点）

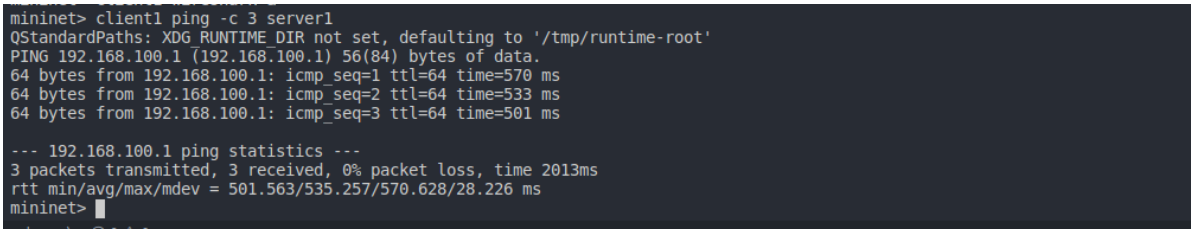
```
$ mininet> client1 wireshark &
```

2.7.2. wireshark抓包

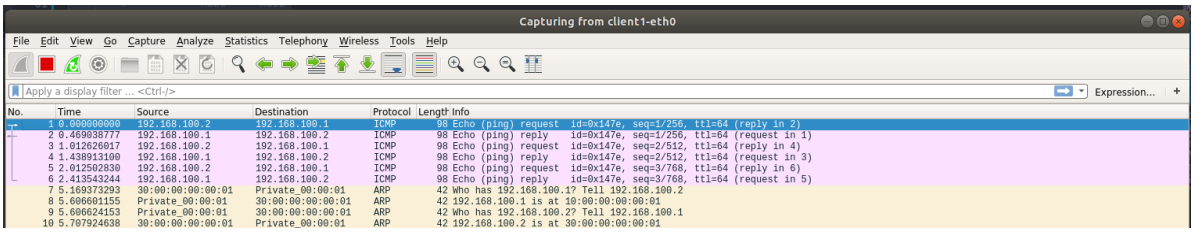
首先向 `server1` `ping` 过去3个包

```
$ mininet> client1 ping -c 3 server1
```

log输出如下



`wireshark` 状态如下：



从上往下分析我们有：

NO.	行为
7	ARP协议，client1向server1发送告知IP地址的请求
8	ARP协议，server1告知client1自身的MAC地址
9	ARP协议，server1向client1发送告知IP地址的请求
10	ARP协议，将client1的mac地址与IP地址关联
1，3，5	ICMP协议，client1发送3个报文， 向server1请求并将数据包发出
2，4，6	server1向client1分别就1，3，5进行答复确认收到数据包

由于抓包工具时间精度不高，在刚开始抓包时会先打印出ARP协议的信息（即7，8，9，10）条协议，等建立起IP-MAC映射后才真正发送有效数据包。

2.7.3. 实验时的一次小错误以及反思

```
mininet> client1 ping -c 1 client2
connect: Network is unreachable
mininet>
```

如果 `client1` 向 `client2` 发送数据包请求，但是由于两个主机不在同一子网，而hub并不允许不同子网之间的主机通信因此会报告 `network is unreachable`

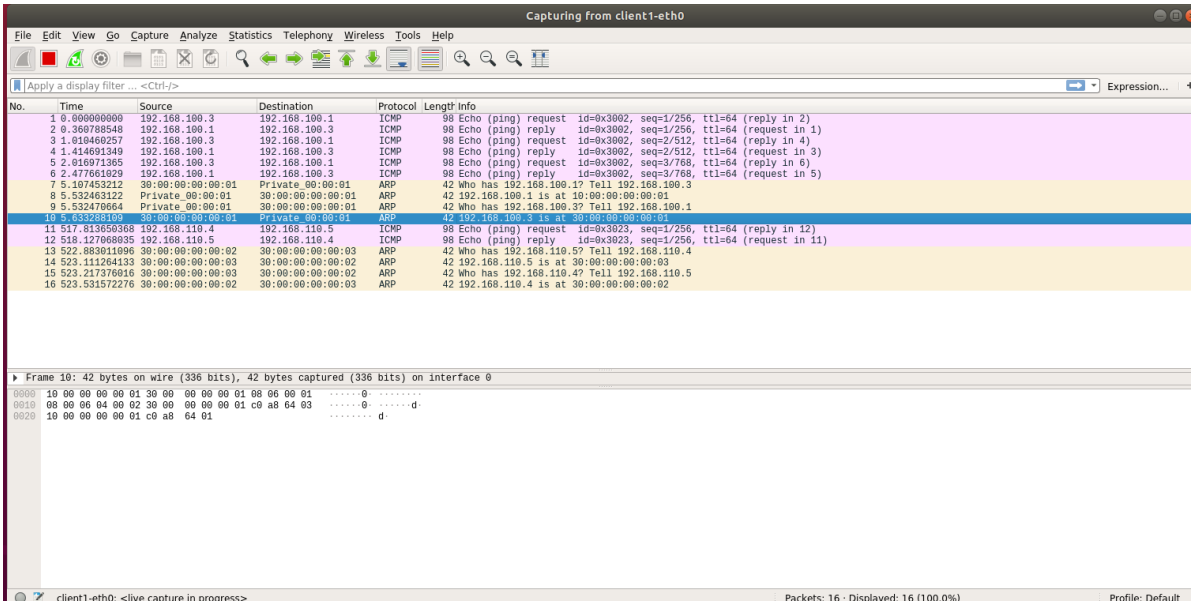
```
mininet> client2 ping -c 1 client3
PING 192.168.110.5 (192.168.110.5) 56(84) bytes of data.
64 bytes from 192.168.110.5: icmp_seq=1 ttl=64 time=545 ms

--- 192.168.110.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 545.203/545.203/545.203/0.000 ms
mininet>
```

于是再次查看我们定义的每个节点，发现 `client2` ， `server2` 在同一子网下，我们将二者 `ping` 通

```
$ mininet> client2 ping -c server2
```

发现在 `client1-eth0` 的 `wireshark` 窗口中也捕捉到了来自 `client2` 和 `server2` 的包，说明集线器将广播性质的包和点对点的包（除到达自身接口的包之外）无差别地发送到其它各个接口，取舍情况完全由连接接口的主机决定默许还是回答



3. 实验收获

- 装配好了实验需要的虚拟机，并找到了解决虚拟机分辨率低的方法。
- 学会了 `python` ， `git` 的基本用法。
- 了解了 `switchyard` 框架并掌握了如何构建自己的 `topology` ，学会了编写测试用例 `.testscenario`
- 在编写测试样例时了解了几个特殊的IP地址：
    - 0.0.0.0： `any` ，表示一台未获取IP地址的接入设备。

- 127.0.0.1： `localhost` ，表示主机自己，常用来测试主机接口是否正常，发包测试时不会发布到网络上。
- 255.255.255.255： `broadcast` ，表示发出的信号可以被任何有能力接受并处理包的设备接受。