

<https://mininet.org/>

python实现自定义网络拓扑结构

1) 根据-topo linear,3来写一个linear.py的脚本

```
from mininet.net import Mininet
from mininet.topo import SingleSwitchTopo

Single3 = SingleSwitchTopo(k=3)
net = Mininet(topo=Single3)
net.start()
net.pingAll()
net.stop()
```

创建linear.py文件到myCustom文件夹下，myCustom文件夹需要与mininet文件夹在同一目录下

2) 根据-topo single,3来写一个single.py的脚本

```
from mininet.net import Mininet
from mininet.topo import SingleSwitchTopo

single3 = SingleSwitchTopo(k=3)
net = Mininet(topo=single3)
net.start()
net.pingAll()
net.stop()
```

3) 根据-topo tree,fanout=2,depth=2来写一个tree.py的脚本

```
from mininet.net import Mininet
from mininet.topolib import TreeTopo

Tree22 = TreeTopo(depth=2, fanout=2)
net = Mininet(topo=Tree22)
net.start()
net.pingAll()
net.stop()
```

4)创建自定义网络拓扑脚本

```

from mininet.net import Mininet
net = Mininet()
# Creating nodes in the network.
c0 = net.addController() //创建一个控制器
h0 = net.addHost('h0') //创建一台主机
s0 = net.addSwitch('s0') //创建一台交换机
h1 = net.addHost('h1')
# Creating links between nodes in network
net.addLink(h0, s0) //创建一条链路，使h0和s0连接起来
net.addLink(h1, s0)
# Configuration of IP addresses in interfaces
h0.setIP('192.168.1.1', 24)
//设置h0主机的IP地址为192.168.1.1，掩码位24位
h1.setIP('192.168.1.2', 24)
net.start() //启动拓扑
net.pingAll() //运行pingall
net.stop() //关闭拓扑

```

addHost()语法可以对主机cpu进行设置，以百分数的形式； addLink()语法可以设置带宽bw、延迟delay、最大队列的大小max_queue_size、损耗率loss

```

from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
net = Mininet(host=CPULimitedHost, link=TCLink)
c0 = net.addController()
s0 = net.addSwitch('s0')
h0 = net.addHost('h0')
h1 = net.addHost('h1', cpu=0.5)
h2 = net.addHost('h1', cpu=0.5)
net.addLink(s0, h0, bw=10, delay='5ms', max_queue_size=1000,
            loss=10, use_htb=True)
net.addLink(s0, h1)
net.addLink(s0, h2)
net.start()
net.pingAll()
net.stop()

```

[mininet实验部分总结-CSDN博客](#)

<code>mininet>nodes</code>	查看全部节点信息
<code>mininet>net</code>	查看链路信息
<code>mininet>dump</code>	查看各节点详细信息
<code>mininet>pingall</code>	测试所有结点是否连通
<code>mininet>pingpair</code>	两个主机将互 ping
<code>mininet>link s1 h2 up/down</code>	启用/禁用s1跟h2之间的链路
<code>mininet>links</code>	报告所有链路状态
<code>mininet>iperf h1 h2</code>	两个节点之间用指定简单的 TCP 测试
<code>mininet>iperfudp 10M h1 h2</code>	两个节点之间用指定简单 udp 进行测试，10M指自己设置的带宽
<code>mininet>time [command]</code>	测量命令所执行的时间
<code>mininet>xterm/gterm s1</code>	打开某结点控制终端
<code>mininet>sh [cmd args]</code>	运行外部 shell 命令
<code>mininet>px/py</code>	执行 python 语句
<code>mininet>source <file></code>	从输入文件读入命令
<code>mininet>exit/quit/EOF</code>	退出 mininet 命令行

mininet采用轻量级的虚拟化技术，使得其模拟的每台主机和交换机都是独立的，所以可以像在真实主机的终端中执行命令一样，在模拟的主机或交换机上执行任何系统命令。在CLI环境中执行的格式为：node command，command 格式和用法同Linux主机，如：

<code>mininet>h1 ifconfig</code>	查看h1节点网络信息
<code>mininet>h1 ping -c 4 h2</code>	实现两主机互连测试
<code>mininet>h1 ifconfig h1-eth0 10.108.126.3 netmask 255.255.255.0</code>	修改虚拟的主机的ip以及mask地址

mininet可视化界面 2.2.0以后版本的mininet支持可视化，在/home/mininet/mininet/examples目录下提供miniedit.py脚本，切换到相应目录下，在终端中执行：

```
sudo ./miniedit.py
```

mn启动参数，格式mn [options]

创建拓扑

(创建完一种类型后可`exit`退出，然后再尝试创建不同类型)

****创建single**

拓扑：（单一（**Single**）**拓扑指整个网络拓扑中交换机有且只有一个，交换机可以下挂一个或多个主机）

```
sudo mn --topo=single,3
```

本例创建了一个交换机、3个主机，3个主机都下挂在一个交换机下

****创建 linear（线性）拓扑：**

（线性（**linear**）**拓扑指交换机连接呈线形排列，且每个交换机所连接主机数目只有一个）

```
sudo mn --topo=linear,3
```

本例创建了3个交换机、3个主机，3个主机分别下挂在一个交换机下

****创建树形（tree）拓扑：**

（树形（**tree**）**拓扑指交换机连接成树形排列，且每个交换机所连接主机一般有多个关联使用参数`depth`及`fanout`）

```
sudo mn --topo=tree,depth=2,fanout=2
```

本例创建了`depth`为2，`fanout`为2的拓扑，表示交换机深度即层数为2，每个交换机下挂2个设备

创建自定义拓扑：（自定义（**custom**）拓扑指python编写文件`file.py`，执行此脚本即可创建定义的拓扑，`-custom`与`-topo`联用）

```
# cd /home/openlab/openlab/mininet/custom
```

```
# sudo mn --custom topo-2sw-2host.py --topo mytopo
```

本例在`custom`目录下存在`topo-2sw-2host.py`文件，调用此文件构建拓扑

自动设置MAC地址和ARP条目

<code>--mac</code>	自动设置MAC地址，MAC地址与IP地址的最后一个字节相同
<code>--arp</code>	为每个主机设置静态ARP表，存储同一网段的主机的mac和IP

设置交换机

```
--switch default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
```

其中，`ovs`，`default`，`ovsk`都为OVS(`openvswitch`)交换机，

`lxbr`=LinuxBridge `user`=UserSwitch `ivs`=IVSSwitch `ovsbr`=OVSBridge

设置控制器

```
--controller=default|none|nox|ovsc|ref|remote|ryu[,param=value...]  
    其中,  ovsc=OVSController none=NullController  
           remote=RemoteController default=DefaultController  
           nox=NOX ryu=Ryu ref=Controller  
--controller=remote,ip=[controller IP] ,port=[controller listening port]  
    设置远程控制器
```

```
sudo mn --controller=remote,ip=10.108.125.9,port=6653 --switch  
ovsk,protocols=OpenFlow13
```

退出并且清理

```
sudo mn -c
```

流表操作

2.1 使用命令查看交换机中的流表

查看流表项:

```
mininet>dpctl dump-flows
```

添加流表项:

在所有的交换机中添加流表, 让从1号端口进入的报文都从2号端口转发 (在操作之前先清空交换机中的流表)

```
mininet>dpctl add-flow in_port=2,actions=output:1  
mininet>dpctl add-flow in_port=1,actions=output:2
```

删除表项:

```
mininet>dpctl del-flows
```

2.2 使用命令查看交换机中的流表

ovs-ofctl是命令行下的交换机管理工具, 也可以在终端中用来管理openflow 流表。

查看交换机中的流表项:

```
sudo ovs-ofctl dump-flows -O openflow13 s1  
# -O参数后面跟协议，s1表示交换机的id。
```

```
ovs-ofctl dump-flows br-sw
```

当然，也可在mininet的命令行窗口使用sh命令来直接调用上述指令：

```
mininet> sh ovs-ofctl dump-flows -O openflow13 s1 # 效果同在终端中执行
```

其他流表操作

附常用OVS操作

1. 添加网桥: `ovs-vsctl add-br 交换机名`
2. 删除网桥: `ovs-vsctl del-br 交换机名`
3. 添加端口: `ovs-vsctl add-port 交换机名 端口名 (网卡名)`
4. 删除端口: `ovs-vsctl del-port 交换机名 端口名 (网卡名)`
5. 连接控制器: `ovs-vsctl set-controller 交换机名 tcp:IP地址:端口号`
6. 断开控制器: `ovs-vsctl del-controller 交换机名`
7. 列出所有网桥: `ovs-vsctl list-br`
8. 列出网桥中的所有端口: `ovs-vsctl list-ports 交换机名`
9. 列出所有挂接到网卡的网桥: `ovs-vsctl port-to-br 端口名 (网卡名)`
10. 查看open vswitch的网络状态: `ovs-vsctl show`
11. 查看 Open vSwitch 中的端口信息
(交换机对应的 `dpid`, 以及每个端口的 OpenFlow 端口编号, 端口名称, 当前状态等等):
`ovs-ofctl show 交换机名`
12. 修改dpid: `ovs-vsctl set bridge 交换机名 other_config:datapath-id=新DPID`
13. 修改端口号: `ovs-vsctl set Interface 端口名 ofport_request=新端口号`
14. 查看交换机中的所有 Table: `ovs-ofctl dump-tables ovs-switch`
15. 查看交换机中的所有流表项: `ovs-ofctl dump-flows ovs-switch`
16. 删除编号为100 的端口上的所有流表项:
`ovs-ofctl del-flows ovs-switch "in_port=100"`
17. 添加流表项 (以“添加新的 OpenFlow 条目, 修改从端口 p0 收到的数据包的源地址为 9.181.137.1”为例):
`ovs-ofctl add-flow ovs-switch "priority=1 idle_timeout=0,in_port=100,actions=mod_nw_src:9.181.137.1,normal"`
(更多说明请查阅:《基于 Open vSwitch 的 OpenFlow 实践》)
18. 查看 OVS 的版本信息: `ovs-appctl -version`
19. 查看 OVS 支持的 OpenFlow 协议的版本: `ovs-ofctl -version`
1. 列出br-int网桥的接口
`ovs-ofctl -O OpenFlow13 show br-int`
2. 列出br-int网桥的接口
`ovs-ofctl dump-ports -O OpenFlow13 br-int`
3. 列出br-int网桥的某个接口的详细信息
`ovs-ofctl dump-ports -O OpenFlow13 br-int 1`
4. 查看 Open vSwitch 中的端口信息
`ovs-ofctl show -O OpenFlow13 br-int`
5. 获得网络接口的 OpenFlow 编号
`ovs-vsctl get Interface tap8f178fef-10 ofport`
6. 查看网桥下的流表
`ovs-ofctl dump-flows -O OpenFlow13 br-int`
7. 查看ovs下的 datapath 的信息

```
ovs-dpctl show
```

8. 根据流量显示在流表中的走向

```
ovs-appctl ofproto/trace br-int in_port=2 | grep "Rule|action"
```

9. ovs设置控制器

```
ovs-vsctl set-controller br0 tcp:1.2.3.4:663
```

10. ovs添加流表

```
ovs-ofctl add-flow br0 in_port=1,actions=output:2
```

11. 删除网桥中所有的流表

```
ovs-ofctl del-flows br0
```

12. 删除根据匹配项删除网桥中的流表

```
ovs-ofctl del-flows br0 "in_port=1"
```

Mininet模拟多数据中心带宽实验

【SDN系列学习课程-OpenFlow-Ryu-Mininet】 https://www.bilibili.com/video/BV1ft4y1a7ip/?p=17&share_source=copy_web&vd_source=00de136a1bd6eae86f7ccd0c05e6332d

[SDN-Mininet模拟多数据中心带宽实验_fattree.py做流量监控-CSDN博客](#)

组表：

os.popen(cmd):

这种调用方式是通过管道的方式来实现，函数返回一个file-like的对象，里面的内容是脚本输出的内容（可简单理解为echo输出的内容）。使用os.popen调用test.sh的情况：

python调用Shell脚本，有两种方法：os.system(cmd)或os.popen(cmd),前者返回值是脚本的退出状态码，后者的返回值是脚本执行过程中的输出内容。实际使用时视需求情况而选择。

明显地，像调用“ls”这样的shell命令，应该使用popen的方法来获得内容

两者的区别是：

os.system(cmd)的返回值只会有0(成功),1,2

os.popen(cmd)会把执行的cmd的输出作为值返回。

例如：

```
#!/usr/bin/python
import os
import time
if '__main__' == __name__:
    os.system('ovs-ofctl add-group s2 group_id=1,type=all,bucket=output:2,bucket=output:3,bucket=output:4')
    os.system('ovs-ofctl add-group s2 group_id=2,type=all,bucket=output:1,bucket=output:3,bucket=output:4')
    os.system('ovs-ofctl add-group s2 group_id=3,type=all,bucket=output:2,bucket=output:1,bucket=output:4')
    os.system('ovs-ofctl add-group s2 group_id=4,type=all,bucket=output:2,bucket=output:3,bucket=output:1')

    os.system('ovs-ofctl add-group s5 group_id=1,type=all,bucket=output:2,bucket=output:3,bucket=output:4')
    os.system('ovs-ofctl add-group s5 group_id=2,type=all,bucket=output:1,bucket=output:3,bucket=output:4')
    os.system('ovs-ofctl add-group s5 group_id=3,type=all,bucket=output:2,bucket=output:1,bucket=output:4')
    os.system('ovs-ofctl add-group s5 group_id=4,type=all,bucket=output:2,bucket=output:3,bucket=output:1')

    os.system('ovs-ofctl add-flow s1 priority=999,in_port=1,actions=output:2')
    os.system('ovs-ofctl add-flow s1 priority=999,in_port=2,actions=output:1')

    os.system('ovs-ofctl add-flow s2 priority=200,dl_src=00:00:00:00:00:02,actions=group:1')
    os.system('ovs-ofctl add-flow s2 priority=200,dl_src=00:00:00:00:00:01,actions=group:2')
    os.system('ovs-ofctl add-flow s2 priority=200,in_port=3,actions=group:3')
    os.system('ovs-ofctl add-flow s2 priority=200,in_port=4,actions=group:4')

def killMn():
    cmd="sudo mn -c"
    text = os.popen( "echo %s | sudo -S %s" % ('password',cmd) ).read()
```

流表

[Quality of Service \(QoS\) — Open vSwitch 3.3.90 documentation](#)

假设您要设置连接到物理以太网端口eth0（1 Gbps设备）和虚拟机接口vif1.0和vif2.0的网桥br0，并且您要将从vif1.0到eth0的流量限制为10 Mbps，将从vif2.0到eth的流量限制至20 Mbps。然后，您可以通过以下方式配置网桥：

```

ovs-vsctl -- \
  add-br br0 -- \
  add-port br0 eth0 -- \
  add-port br0 vif1.0 -- set interface vif1.0 ofport_request=5 -- \
  add-port br0 vif2.0 -- set interface vif2.0 ofport_request=6 -- \
  set port eth0 qos=@newqos -- \
  --id=@newqos create qos type=linux-htb \
    other-config:max-rate=1000000000 \
    queues:123=@vif10queue \
    queues:234=@vif20queue -- \
  --id=@vif10queue create queue other-config:max-rate=10000000 -- \
  --id=@vif20queue create queue other-config:max-rate=20000000

```

使用OpenFlow将数据包从vif1.0和vif2.0引导到为它们保留的队列：

```

ovs-ofctl add-flow br0 in_port=5,actions=set_queue:123,normal
ovs-ofctl add-flow br0 in_port=6,actions=set_queue:234,normal

```

删除网桥：

```

ovs-vsctl del-br br0

```