实验二

一．实验目的

如图 1.1 所示，图为一个 FatTree 拓扑，现在需要实现以下目标：

（1）使用 Mininet 和 MinEdit 来模拟 FatTree，完成 FatTree 拓扑结构的建立。

（2）在带有控制器的 FatTree 网络，由于网络中存在环，需要使用生成树协议；并证明链路的健壮性。

（3）在不带有控制器的 FatTree 网络，需要手动添加相关的流表项，并证明带宽的相对均匀性。

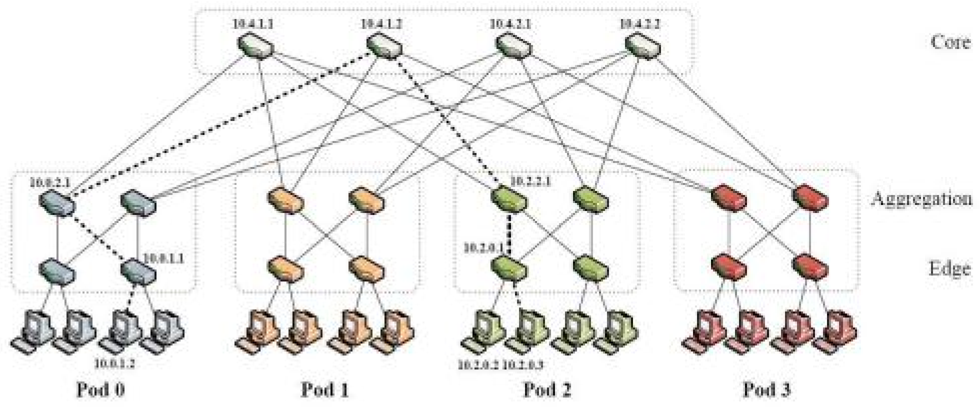


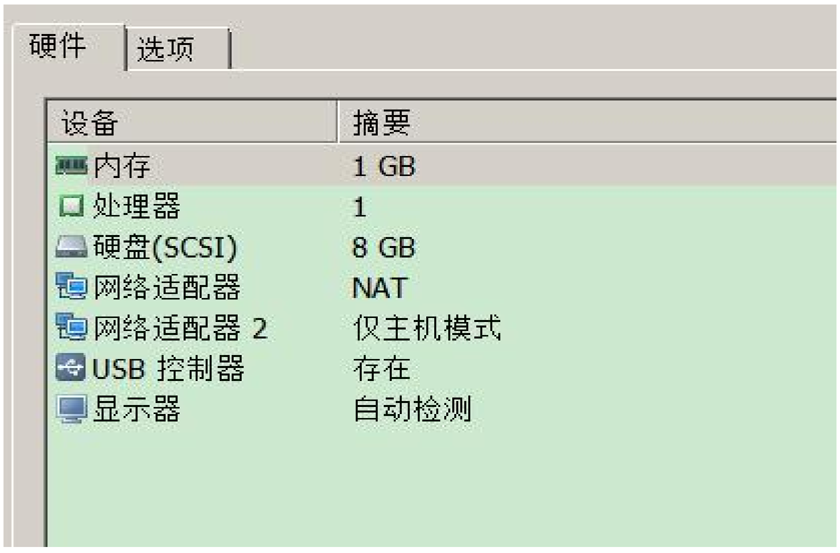
图 1.1

二．实验环境

（1）安装 mininet virtual machine

① 从 http://222.195.68.57/CS05112/pyretic_0.2.0_32bit.ova 下载 ova 镜像，然后将其导入到 VMWare 中

② 虚拟机的配置如图 2.1:



设备	摘要
内存	1 GB
处理器	1
硬盘(SCSI)	8 GB
网络适配器	NAT
网络适配器 2	仅主机模式
USB 控制器	存在
显示器	自动检测

图 2.1 虚拟机配置情况图

（2）在 windows 上安装 Xming 和 putty 软件
（3）如果 Ubuntu 上没有 wireshark 的话，要安装 wireshark
（4）如果 Ubuntu 上没有安装 pox 的话，就要从 http://github.com/noxrepo/pox 上下载 pox
（5）进入到 pyretic/pyretic/examples 文件下，执行以下两条命令：

① wget http://222.195.68.57/CS05112/mininet/pyretic_hub.py

② wget http://222.195.68.57/CS05112/mininet/pyretic_switch.py

（6）配置 miniedit.py

① 因为我在运行 miniedit.py 的时候出错了，提示 no display name and no $DISPLAY environment variable，出现这个问题是和 Matplotlib 有关的

② 使用 whereis matplotlibrc 命令找到 matplotlibrc 文件所在的位置，然后在文件的末尾加上了： backend:Agg 之后就能正常运行 miniedit.py 了

三．带控制器的网络

（1）网络拓扑的建立

我们使用 miniedit 来构建拓扑结构。miniedit 是一个可视化界面，它可以以拖拽图标的形式轻松的完成拓扑结构的建立。它的界面如图 3.1 所示。

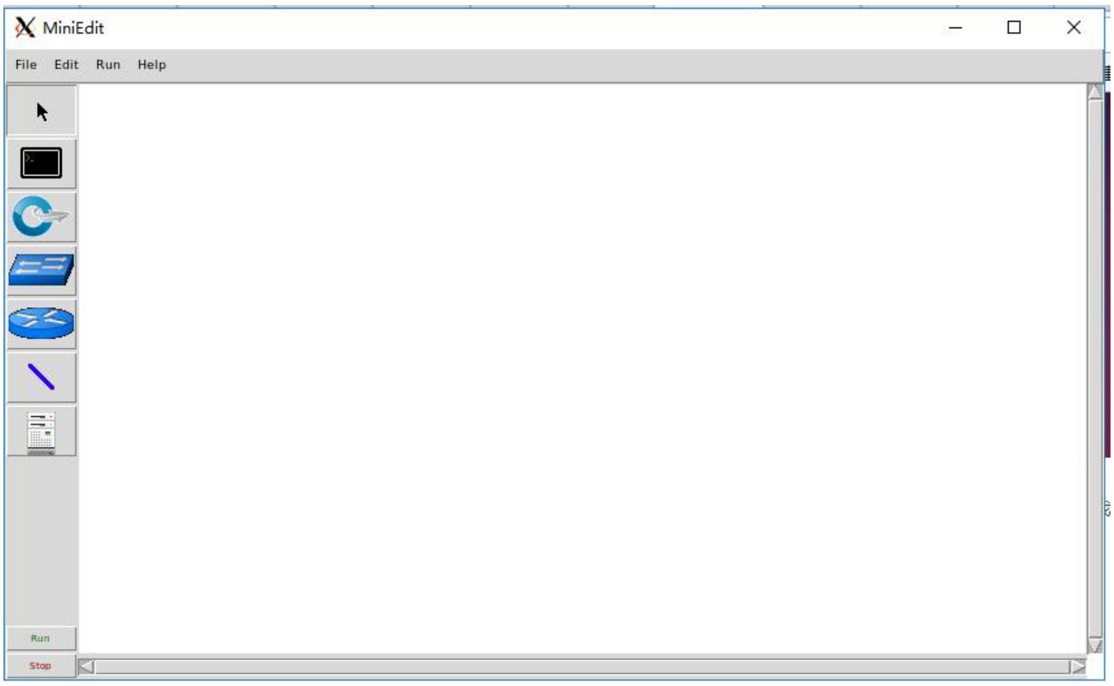


图 3.1 miniedit 界面

打开 miniedit 之后我们我们就可以构建如图 3.2 所示的拓扑图。在构建玩拓扑图后将代码保存到了 myToPo1.py 中

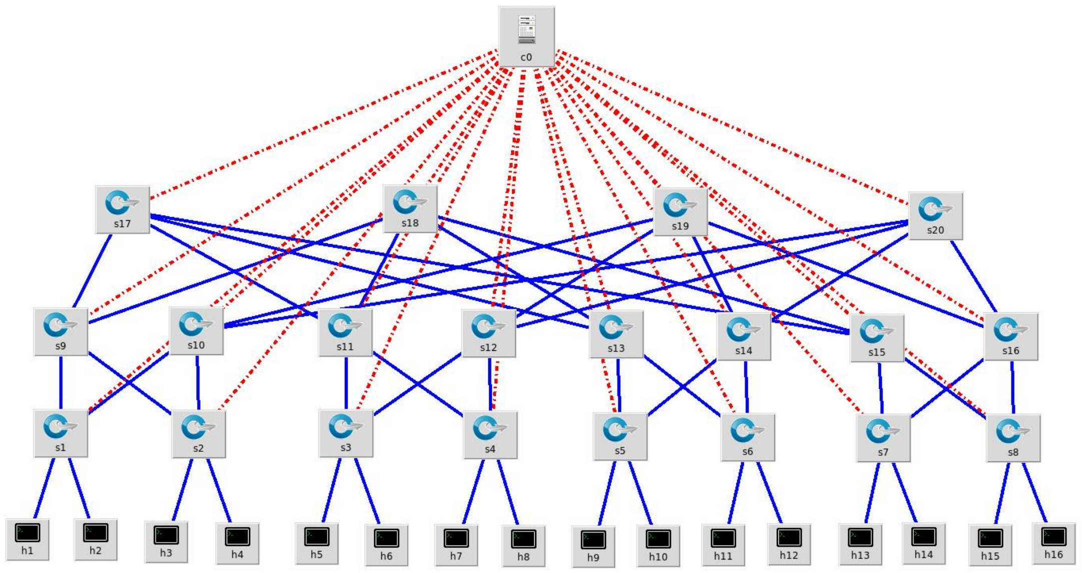
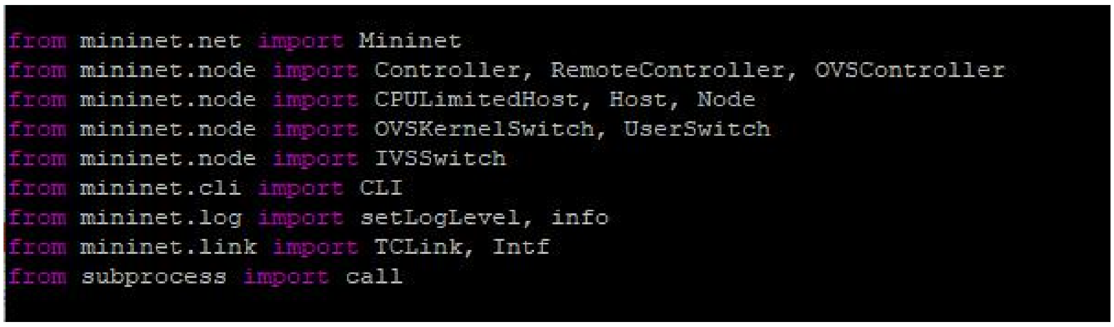


图 3.2 拓扑结构图

（2）拓扑代码及其解释

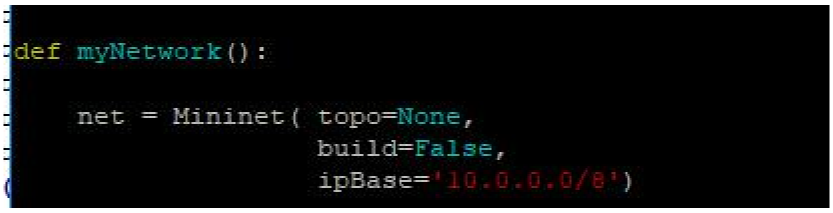
代码保存在 myToPo1.py 中

① 引入包



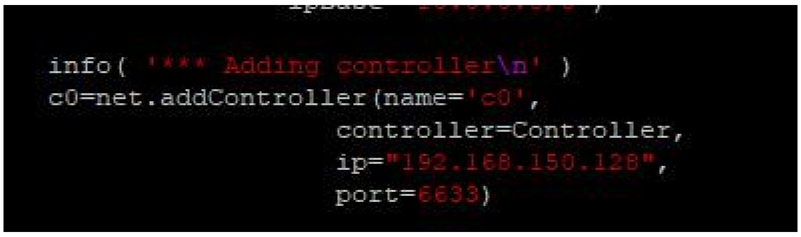
```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
```

② 定义一个 mininet 网络



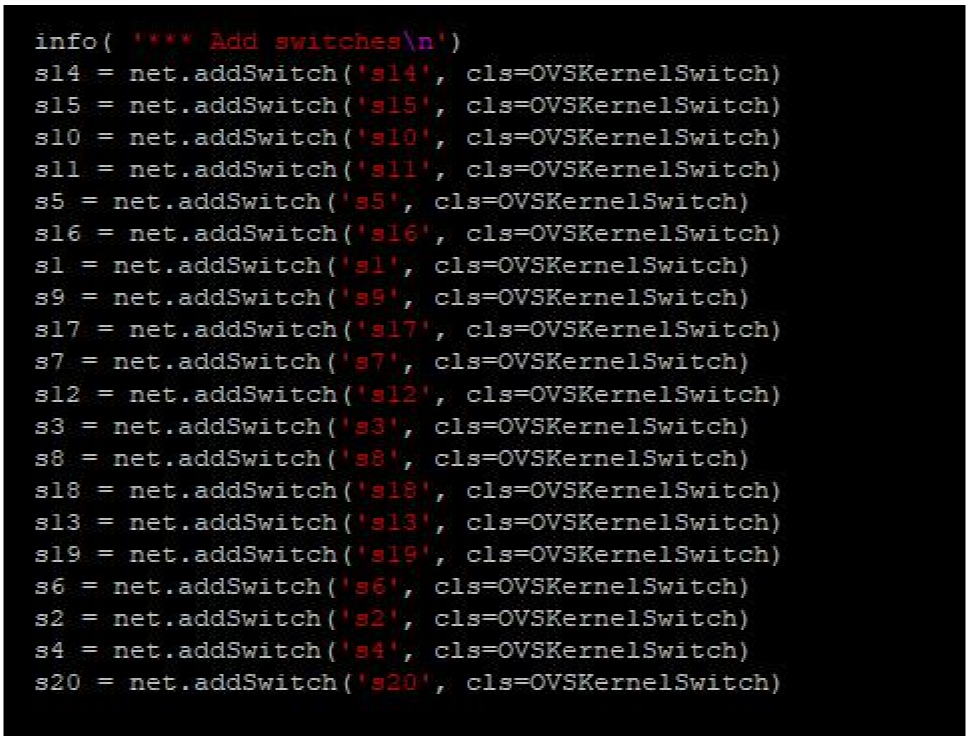
```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')
```

③ 加入一个控制器

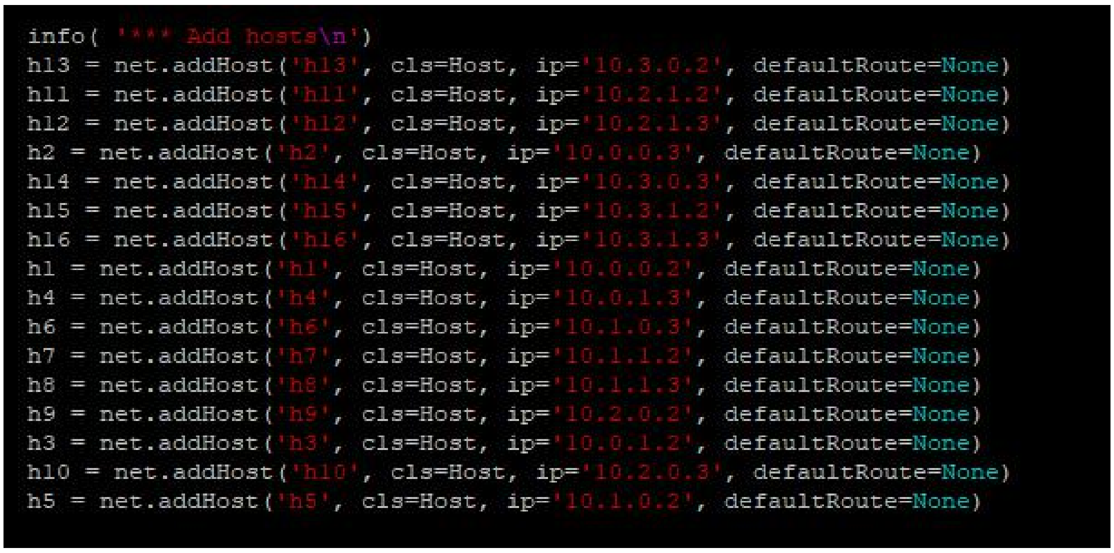


```
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=Controller,
                      ip="192.168.150.128",
                      port=6633)
```

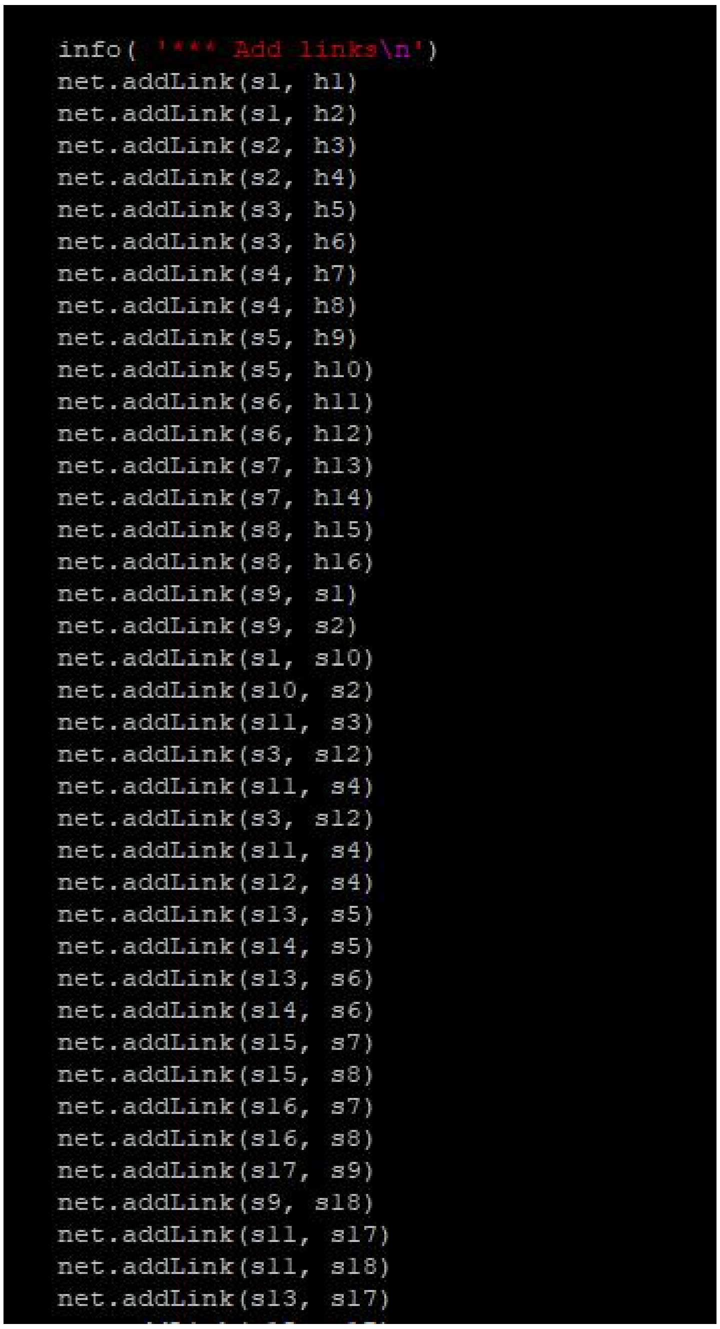
④ 加入 20 个交换机


```
info( '*** Add switches\n')
s14 = net.addSwitch('s14', cls=OVSKernelSwitch)
s15 = net.addSwitch('s15', cls=OVSKernelSwitch)
s10 = net.addSwitch('s10', cls=OVSKernelSwitch)
s11 = net.addSwitch('s11', cls=OVSKernelSwitch)
s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
s16 = net.addSwitch('s16', cls=OVSKernelSwitch)
s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
s17 = net.addSwitch('s17', cls=OVSKernelSwitch)
s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
s12 = net.addSwitch('s12', cls=OVSKernelSwitch)
s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
s18 = net.addSwitch('s18', cls=OVSKernelSwitch)
s13 = net.addSwitch('s13', cls=OVSKernelSwitch)
s19 = net.addSwitch('s19', cls=OVSKernelSwitch)
s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
s20 = net.addSwitch('s20', cls=OVSKernelSwitch)
```

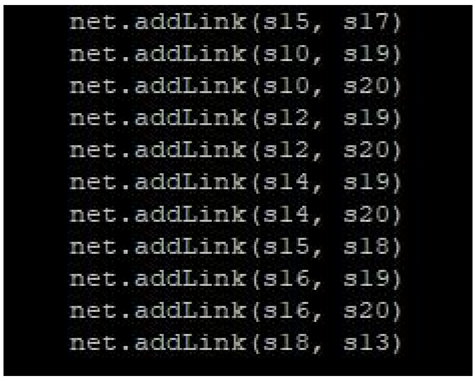
⑤ 加入 16 个节点


```
info( '*** Add hosts\n')
h13 = net.addHost('h13', cls=Host, ip='10.3.0.2', defaultRoute=None)
h11 = net.addHost('h11', cls=Host, ip='10.2.1.2', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.2.1.3', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.3', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.3.0.3', defaultRoute=None)
h15 = net.addHost('h15', cls=Host, ip='10.3.1.2', defaultRoute=None)
h16 = net.addHost('h16', cls=Host, ip='10.3.1.3', defaultRoute=None)
h1 = net.addHost('h1', cls=Host, ip='10.0.0.2', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.1.3', defaultRoute=None)
h6 = net.addHost('h6', cls=Host, ip='10.1.0.3', defaultRoute=None)
h7 = net.addHost('h7', cls=Host, ip='10.1.1.2', defaultRoute=None)
h8 = net.addHost('h8', cls=Host, ip='10.1.1.3', defaultRoute=None)
h9 = net.addHost('h9', cls=Host, ip='10.2.0.2', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.1.2', defaultRoute=None)
h10 = net.addHost('h10', cls=Host, ip='10.2.0.3', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.1.0.2', defaultRoute=None)
```

⑥ 在交换机和节点间添加链路

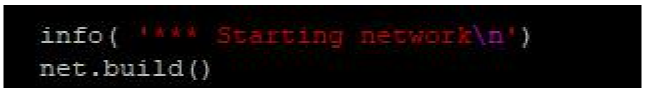


```
info( '*** Add links\n')
net.addLink(s1, h1)
net.addLink(s1, h2)
net.addLink(s2, h3)
net.addLink(s2, h4)
net.addLink(s3, h5)
net.addLink(s3, h6)
net.addLink(s4, h7)
net.addLink(s4, h8)
net.addLink(s5, h9)
net.addLink(s5, h10)
net.addLink(s6, h11)
net.addLink(s6, h12)
net.addLink(s7, h13)
net.addLink(s7, h14)
net.addLink(s8, h15)
net.addLink(s8, h16)
net.addLink(s9, s1)
net.addLink(s9, s2)
net.addLink(s1, s10)
net.addLink(s10, s2)
net.addLink(s11, s3)
net.addLink(s3, s12)
net.addLink(s11, s4)
net.addLink(s3, s12)
net.addLink(s11, s4)
net.addLink(s12, s4)
net.addLink(s13, s5)
net.addLink(s14, s5)
net.addLink(s13, s6)
net.addLink(s14, s6)
net.addLink(s15, s7)
net.addLink(s15, s8)
net.addLink(s16, s7)
net.addLink(s16, s8)
net.addLink(s17, s9)
net.addLink(s9, s18)
net.addLink(s11, s17)
net.addLink(s11, s18)
net.addLink(s13, s17)
```



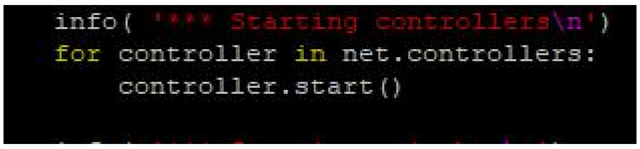
```
net.addLink(s15, s17)
net.addLink(s10, s19)
net.addLink(s10, s20)
net.addLink(s12, s19)
net.addLink(s12, s20)
net.addLink(s14, s19)
net.addLink(s14, s20)
net.addLink(s15, s18)
net.addLink(s16, s19)
net.addLink(s16, s20)
net.addLink(s18, s13)
```

⑦ 开启网络



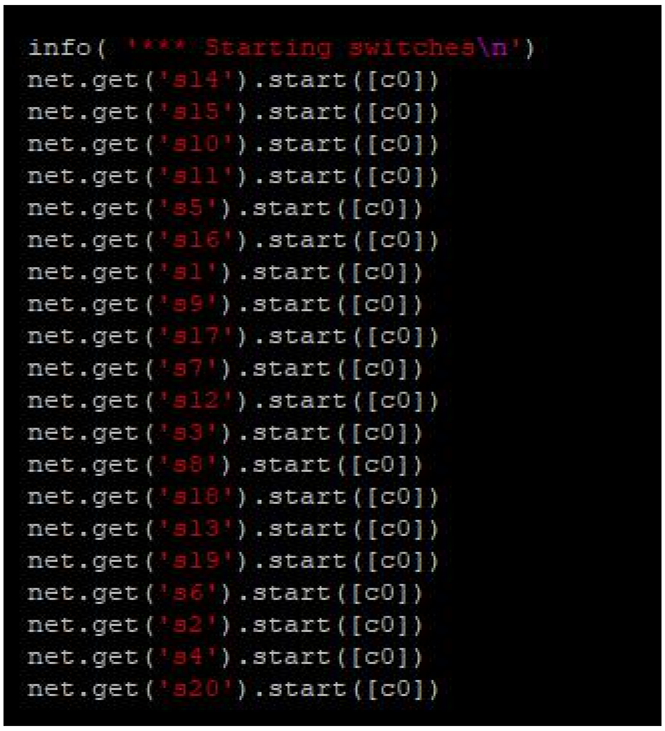
```
info( '*** Starting network\n')
net.build()
```

⑧ 开启控制器



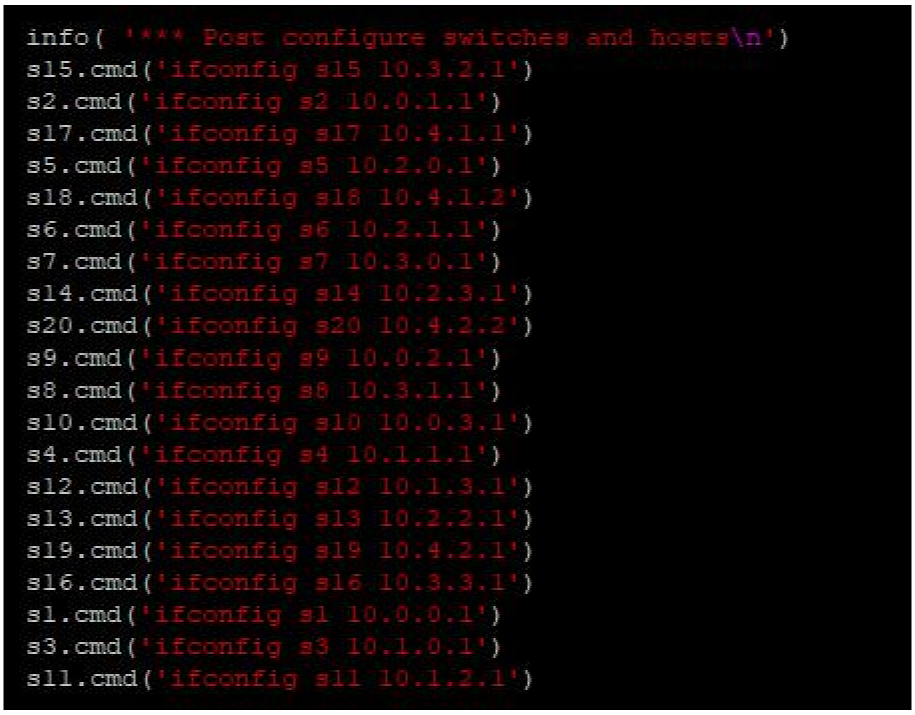
```
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()
```

⑨ 开启交换机



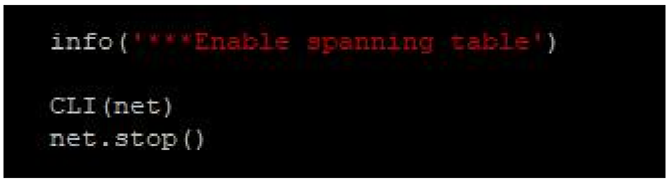
```
info( '*** Starting switches\n')
net.get('s14').start([c0])
net.get('s15').start([c0])
net.get('s10').start([c0])
net.get('s11').start([c0])
net.get('s5').start([c0])
net.get('s16').start([c0])
net.get('s1').start([c0])
net.get('s9').start([c0])
net.get('s17').start([c0])
net.get('s7').start([c0])
net.get('s12').start([c0])
net.get('s3').start([c0])
net.get('s8').start([c0])
net.get('s18').start([c0])
net.get('s13').start([c0])
net.get('s19').start([c0])
net.get('s6').start([c0])
net.get('s2').start([c0])
net.get('s4').start([c0])
net.get('s20').start([c0])
```

⑩ 配置交换机



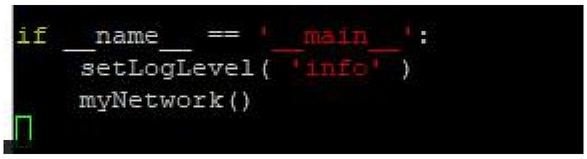
```
info( '*** Post configure switches and hosts\n')
s15.cmd('ifconfig s15 10.3.2.1')
s2.cmd('ifconfig s2 10.0.1.1')
s17.cmd('ifconfig s17 10.4.1.1')
s5.cmd('ifconfig s5 10.2.0.1')
s18.cmd('ifconfig s18 10.4.1.2')
s6.cmd('ifconfig s6 10.2.1.1')
s7.cmd('ifconfig s7 10.3.0.1')
s14.cmd('ifconfig s14 10.2.3.1')
s20.cmd('ifconfig s20 10.4.2.2')
s9.cmd('ifconfig s9 10.0.2.1')
s8.cmd('ifconfig s8 10.3.1.1')
s10.cmd('ifconfig s10 10.0.3.1')
s4.cmd('ifconfig s4 10.1.1.1')
s12.cmd('ifconfig s12 10.1.3.1')
s13.cmd('ifconfig s13 10.2.2.1')
s19.cmd('ifconfig s19 10.4.2.1')
s16.cmd('ifconfig s16 10.3.3.1')
s1.cmd('ifconfig s1 10.0.0.1')
s3.cmd('ifconfig s3 10.1.0.1')
s11.cmd('ifconfig s11 10.1.2.1')
```

⑪ 结束



```
    info('***Enable spanning table')

    CLI(net)
    net.stop()
```

⑫ 主函数



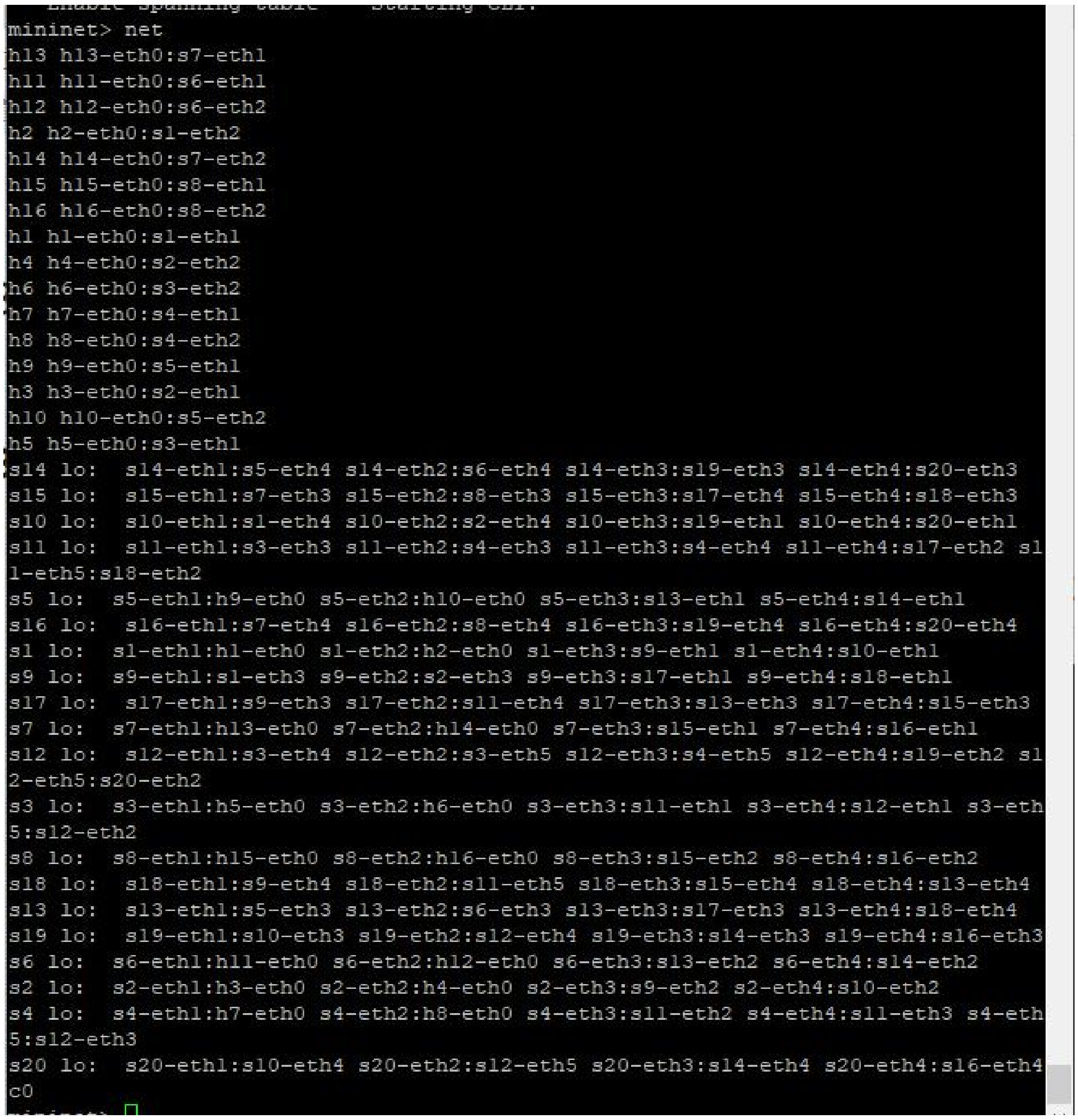
```
if __name__ == '__main__':
     setLogLevel( 'info' )
     myNetwork()
```

（3）拓扑测试

① 正常测试

1) 执行 Sudo ./myToPo1.py 命令，进入 mininet 模拟

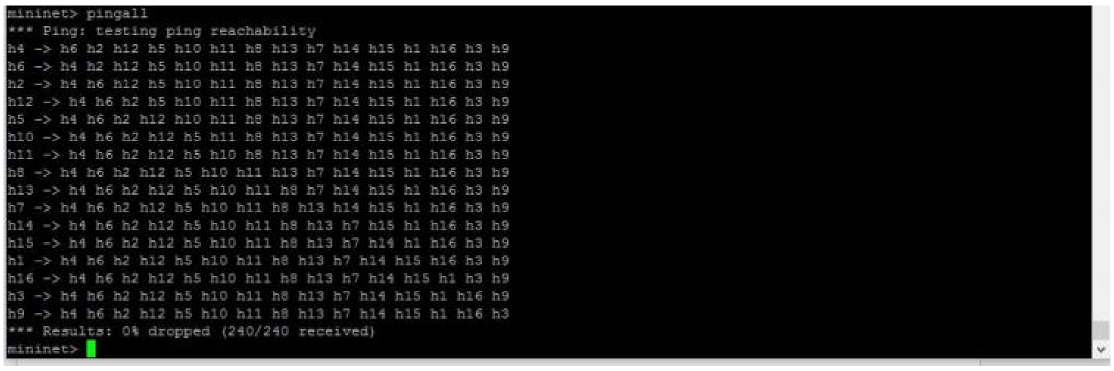
2) 执行 net 命令，查看网络结构，如图 3.3。



```
mininet> net
h13 h13-eth0:s7-eth1
h11 h11-eth0:s6-eth1
h12 h12-eth0:s6-eth2
h2 h2-eth0:s1-eth2
h14 h14-eth0:s7-eth2
h15 h15-eth0:s8-eth1
h16 h16-eth0:s8-eth2
h1 h1-eth0:s1-eth1
h4 h4-eth0:s2-eth2
h6 h6-eth0:s3-eth2
h7 h7-eth0:s4-eth1
h8 h8-eth0:s4-eth2
h9 h9-eth0:s5-eth1
h3 h3-eth0:s2-eth1
h10 h10-eth0:s5-eth2
h5 h5-eth0:s3-eth1
s14 lo:  s14-eth1:s5-eth4 s14-eth2:s6-eth4 s14-eth3:s19-eth3 s14-eth4:s20-eth3
s15 lo:  s15-eth1:s7-eth3 s15-eth2:s8-eth3 s15-eth3:s17-eth4 s15-eth4:s18-eth3
s10 lo:  s10-eth1:s1-eth4 s10-eth2:s2-eth4 s10-eth3:s19-eth1 s10-eth4:s20-eth1
s11 lo:  s11-eth1:s3-eth3 s11-eth2:s4-eth3 s11-eth3:s4-eth4 s11-eth4:s17-eth2 s1
1-eth5:s18-eth2
s5 lo:  s5-eth1:h9-eth0 s5-eth2:h10-eth0 s5-eth3:s13-eth1 s5-eth4:s14-eth1
s16 lo:  s16-eth1:s7-eth4 s16-eth2:s8-eth4 s16-eth3:s19-eth4 s16-eth4:s20-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s9-eth1 s1-eth4:s10-eth1
s9 lo:  s9-eth1:s1-eth3 s9-eth2:s2-eth3 s9-eth3:s17-eth1 s9-eth4:s18-eth1
s17 lo:  s17-eth1:s9-eth3 s17-eth2:s11-eth4 s17-eth3:s13-eth3 s17-eth4:s15-eth3
s7 lo:  s7-eth1:h13-eth0 s7-eth2:h14-eth0 s7-eth3:s15-eth1 s7-eth4:s16-eth1
s12 lo:  s12-eth1:s3-eth4 s12-eth2:s3-eth5 s12-eth3:s4-eth5 s12-eth4:s19-eth2 s1
2-eth5:s20-eth2
s3 lo:  s3-eth1:h5-eth0 s3-eth2:h6-eth0 s3-eth3:s11-eth1 s3-eth4:s12-eth1 s3-eth
5:s12-eth2
s8 lo:  s8-eth1:h15-eth0 s8-eth2:h16-eth0 s8-eth3:s15-eth2 s8-eth4:s16-eth2
s18 lo:  s18-eth1:s9-eth4 s18-eth2:s11-eth5 s18-eth3:s15-eth4 s18-eth4:s13-eth4
s13 lo:  s13-eth1:s5-eth3 s13-eth2:s6-eth3 s13-eth3:s17-eth3 s13-eth4:s18-eth4
s19 lo:  s19-eth1:s10-eth3 s19-eth2:s12-eth4 s19-eth3:s14-eth3 s19-eth4:s16-eth3
s6 lo:  s6-eth1:h11-eth0 s6-eth2:h12-eth0 s6-eth3:s13-eth2 s6-eth4:s14-eth2
s2 lo:  s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:s9-eth2 s2-eth4:s10-eth2
s4 lo:  s4-eth1:h7-eth0 s4-eth2:h8-eth0 s4-eth3:s11-eth2 s4-eth4:s11-eth3 s4-eth
5:s12-eth3
s20 lo:  s20-eth1:s10-eth4 s20-eth2:s12-eth5 s20-eth3:s14-eth4 s20-eth4:s16-eth4
c0
```

图 3.3 网络结构

3) 执行 pingall 指令测试所有主机的连通状况，结构如图 3.4。



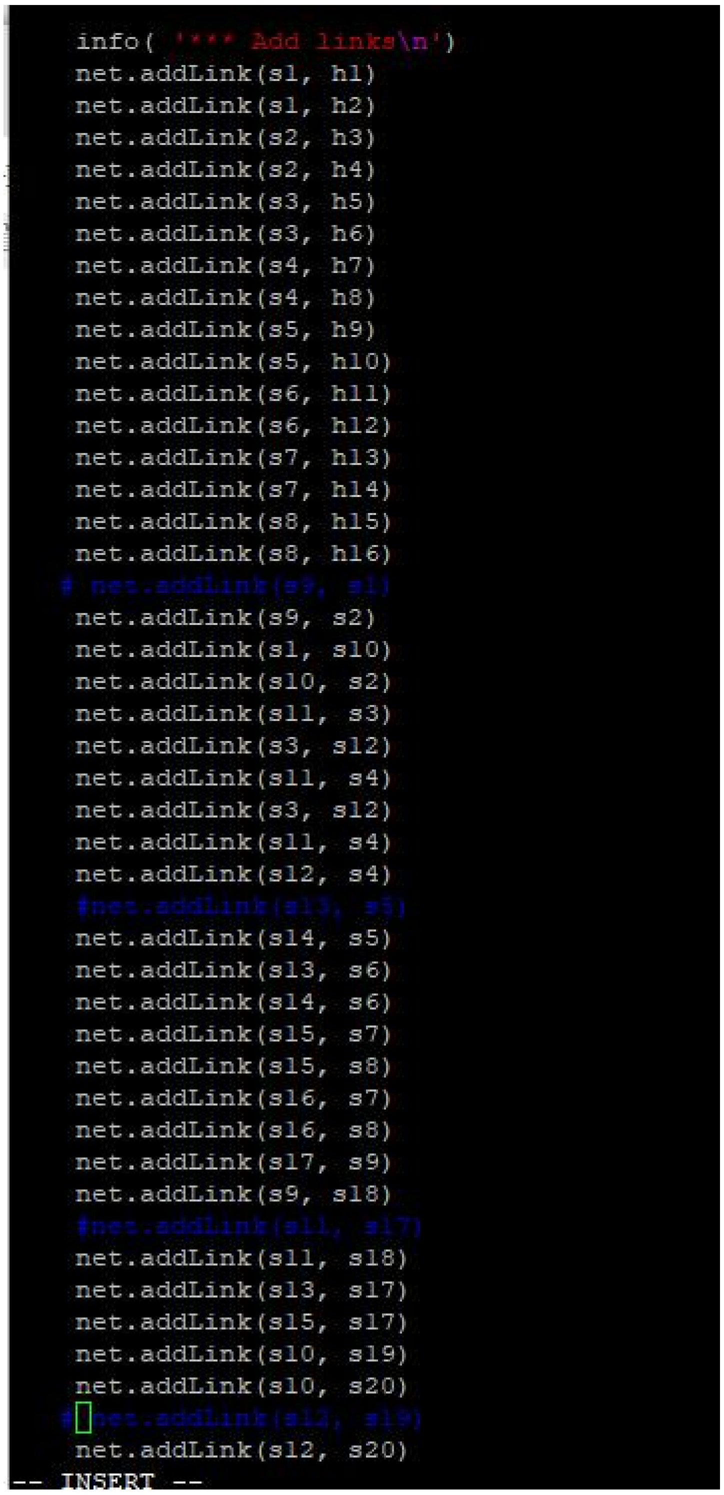
```
mininet> pingall
*** Ping: testing ping reachability
h4 -> h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h6 -> h4 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h2 -> h4 h6 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h12 -> h4 h6 h2 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h5 -> h4 h6 h2 h12 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h10 -> h4 h6 h2 h12 h5 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h11 -> h4 h6 h2 h12 h5 h10 h8 h13 h7 h14 h15 h1 h16 h3 h9
h8 -> h4 h6 h2 h12 h5 h10 h11 h13 h7 h14 h15 h1 h16 h3 h9
h13 -> h4 h6 h2 h12 h5 h10 h11 h8 h7 h14 h15 h1 h16 h3 h9
h7 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h14 h15 h1 h16 h3 h9
h14 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h15 h1 h16 h3 h9
h15 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h1 h16 h3 h9
h1 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h16 h3 h9
h16 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h3 h9
h3 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h9
h9 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3
*** Results: 0% dropped (240/240 received)
mininet> 
```

图 3.4 主机连通状况

4)　从步骤三和步骤四的结果可以看出，网络被正常构建，并且所有主机之间能正确连通。

②　异常测试

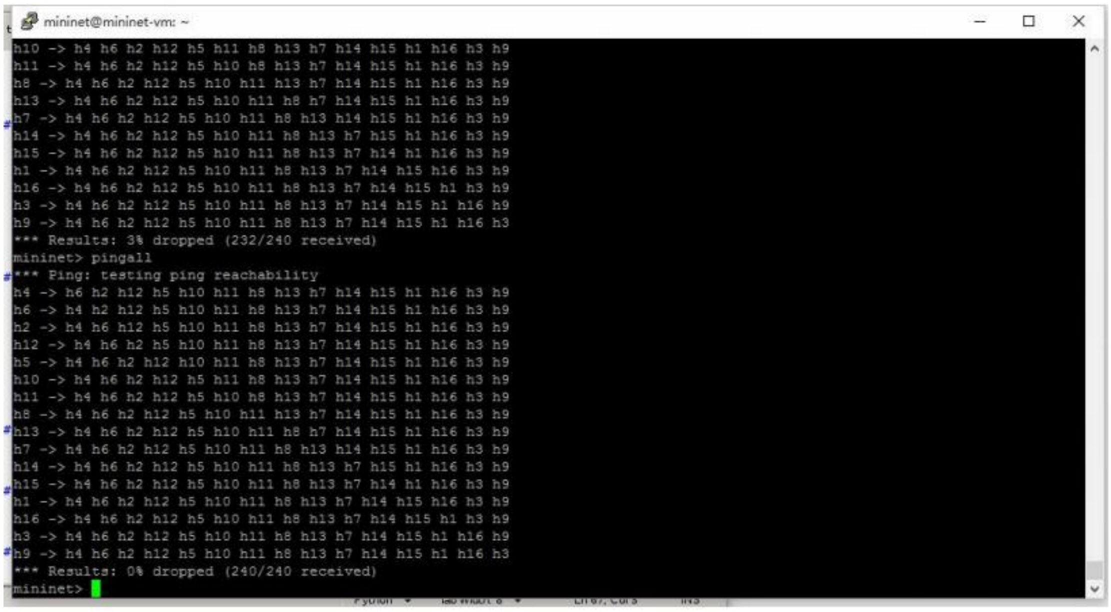
我们对网络结构中的部分链路进行破坏，然后观察网络结构和连通情况。如图 3.5 所示，我们注释掉其中的四条链路：(s9, s1), (s13, s5), (s11, s17), (s12, s19)。



```
    info( '*** Add links\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s2, h3)
    net.addLink(s2, h4)
    net.addLink(s3, h5)
    net.addLink(s3, h6)
    net.addLink(s4, h7)
    net.addLink(s4, h8)
    net.addLink(s5, h9)
    net.addLink(s5, h10)
    net.addLink(s6, h11)
    net.addLink(s6, h12)
    net.addLink(s7, h13)
    net.addLink(s7, h14)
    net.addLink(s8, h15)
    net.addLink(s8, h16)
   # net.addLink(s9, s1)
    net.addLink(s9, s2)
    net.addLink(s1, s10)
    net.addLink(s10, s2)
    net.addLink(s11, s3)
    net.addLink(s3, s12)
    net.addLink(s11, s4)
    net.addLink(s3, s12)
    net.addLink(s11, s4)
    net.addLink(s12, s4)
    #net.addLink(s13, s5)
    net.addLink(s14, s5)
    net.addLink(s13, s6)
    net.addLink(s14, s6)
    net.addLink(s15, s7)
    net.addLink(s15, s8)
    net.addLink(s16, s7)
    net.addLink(s16, s8)
    net.addLink(s17, s9)
    net.addLink(s9, s18)
    #net.addLink(s11, s17)
    net.addLink(s11, s18)
    net.addLink(s13, s17)
    net.addLink(s15, s17)
    net.addLink(s10, s19)
    net.addLink(s10, s20)
   # net.addLink(s12, s19)
    net.addLink(s12, s20)
-- INSERT --
```

图 3.5 破坏部分链路后的拓扑图

1) 我们重新执行 sudo ./myToPo1.py
2) 执行 pingall 命令检查链路的连通情况，结果如图 3.6 所示：


mininet@mininet-vm: ~

```
h10 -> h4 h6 h2 h12 h5 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h11 -> h4 h6 h2 h12 h5 h10 h8 h13 h7 h14 h15 h1 h16 h3 h9
h8 -> h4 h6 h2 h12 h5 h10 h11 h13 h7 h14 h15 h1 h16 h3 h9
h13 -> h4 h6 h2 h12 h5 h10 h11 h8 h7 h14 h15 h1 h16 h3 h9
h7 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h14 h15 h1 h16 h3 h9
h14 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h15 h1 h16 h3 h9
h15 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h1 h16 h3 h9
h1 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h16 h3 h9
h16 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h3 h9
h3 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h9
h9 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3
*** Results: 3% dropped (232/240 received)
mininet> pingall
*** Ping: testing ping reachability
h4 -> h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h6 -> h4 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h2 -> h4 h6 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h12 -> h4 h6 h2 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h5 -> h4 h6 h2 h12 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h10 -> h4 h6 h2 h12 h5 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h11 -> h4 h6 h2 h12 h5 h10 h8 h13 h7 h14 h15 h1 h16 h3 h9
h8 -> h4 h6 h2 h12 h5 h10 h11 h13 h7 h14 h15 h1 h16 h3 h9
h13 -> h4 h6 h2 h12 h5 h10 h11 h8 h7 h14 h15 h1 h16 h3 h9
h7 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h14 h15 h1 h16 h3 h9
h14 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h15 h1 h16 h3 h9
h15 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h1 h16 h3 h9
h1 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h16 h3 h9
h16 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h3 h9
h3 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h9
h9 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3
*** Results: 0% dropped (240/240 received)
mininet> 
```

图 3.6 破坏部分链路后网络的连通结果图

3)　从结果图上很容易看出，在破坏部分链路的情况下，网络仍然可以正常连通。

四．不带控制器的网络

（1）网络拓扑的构建

我们依旧使用 miniedit 进行网络拓扑结构的构建。其拓扑结构图如图 4.1 所示。我们将该拓扑结构导出为 myToPo2.py。

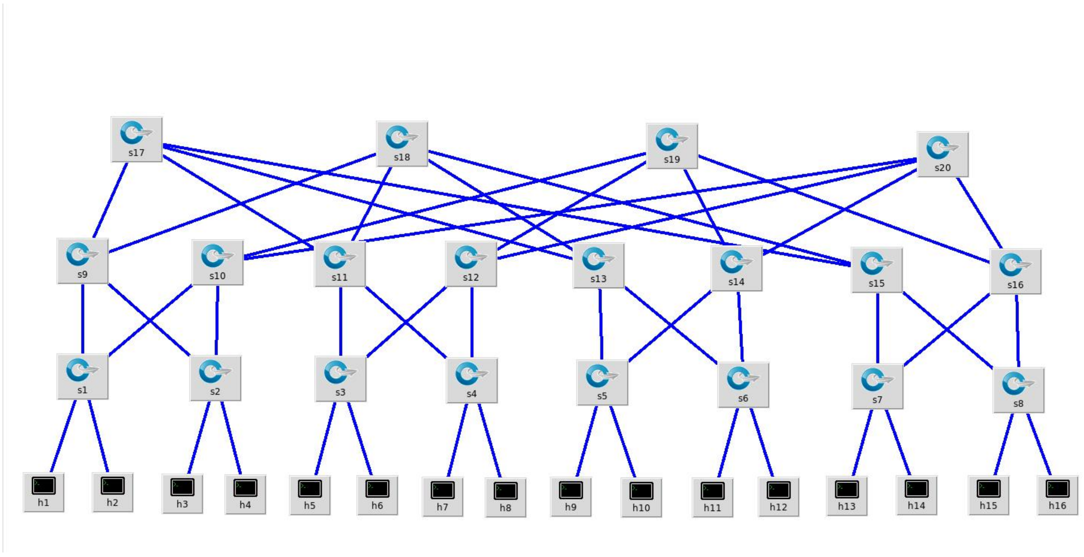
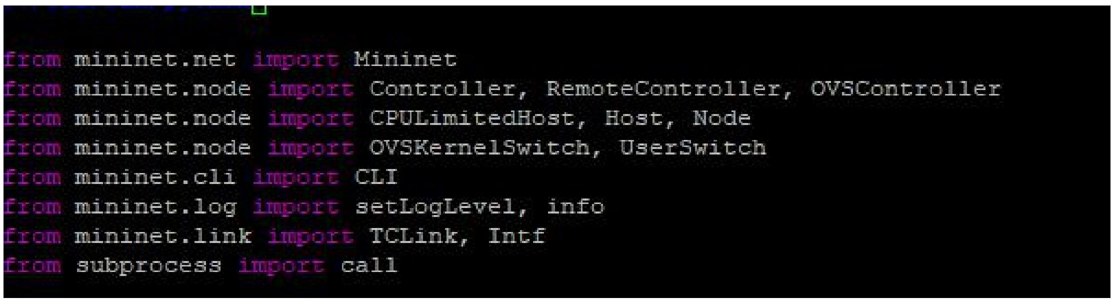


图 4.1 不带控制器的网络拓扑结构图

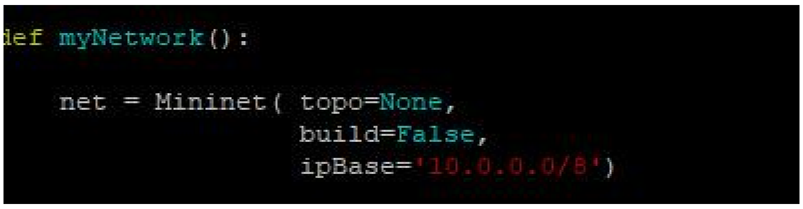
（2）拓扑代码和解释

① 引入包



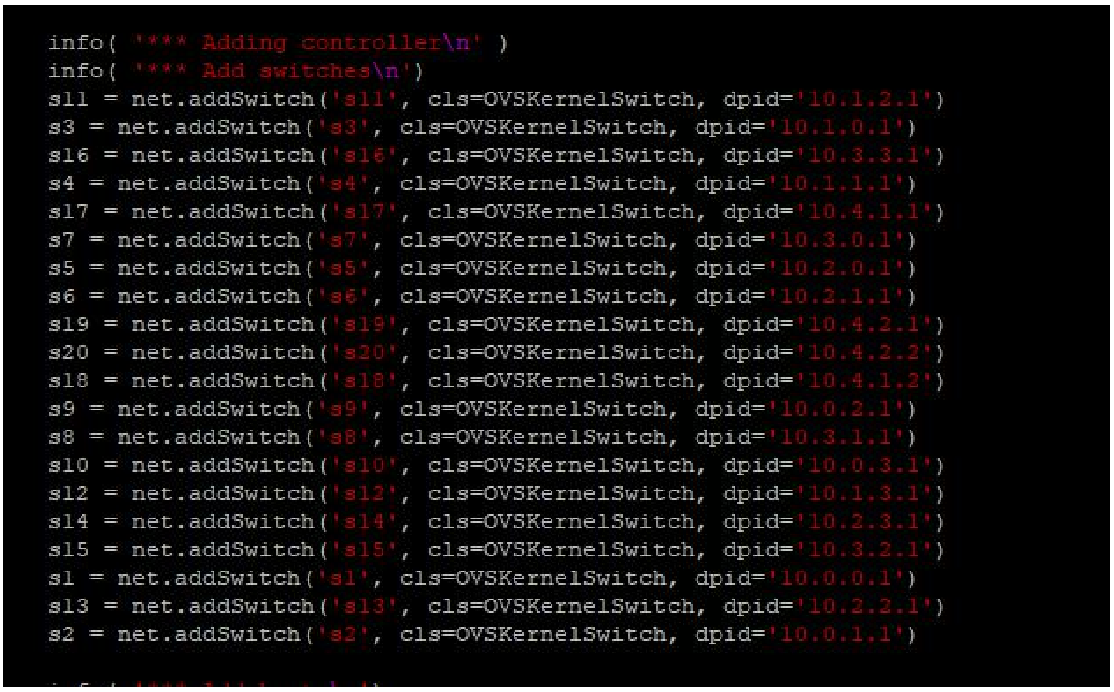
```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
```

② 定义一个 mininet 网络



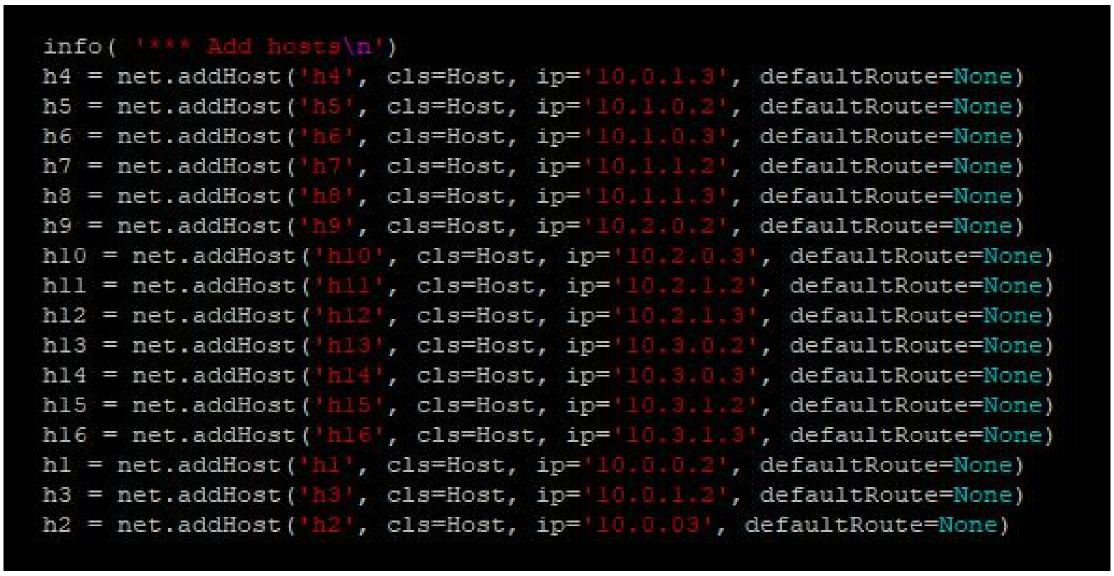
```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')
```

③ 创建 20 个交换机并配置相应的 ip



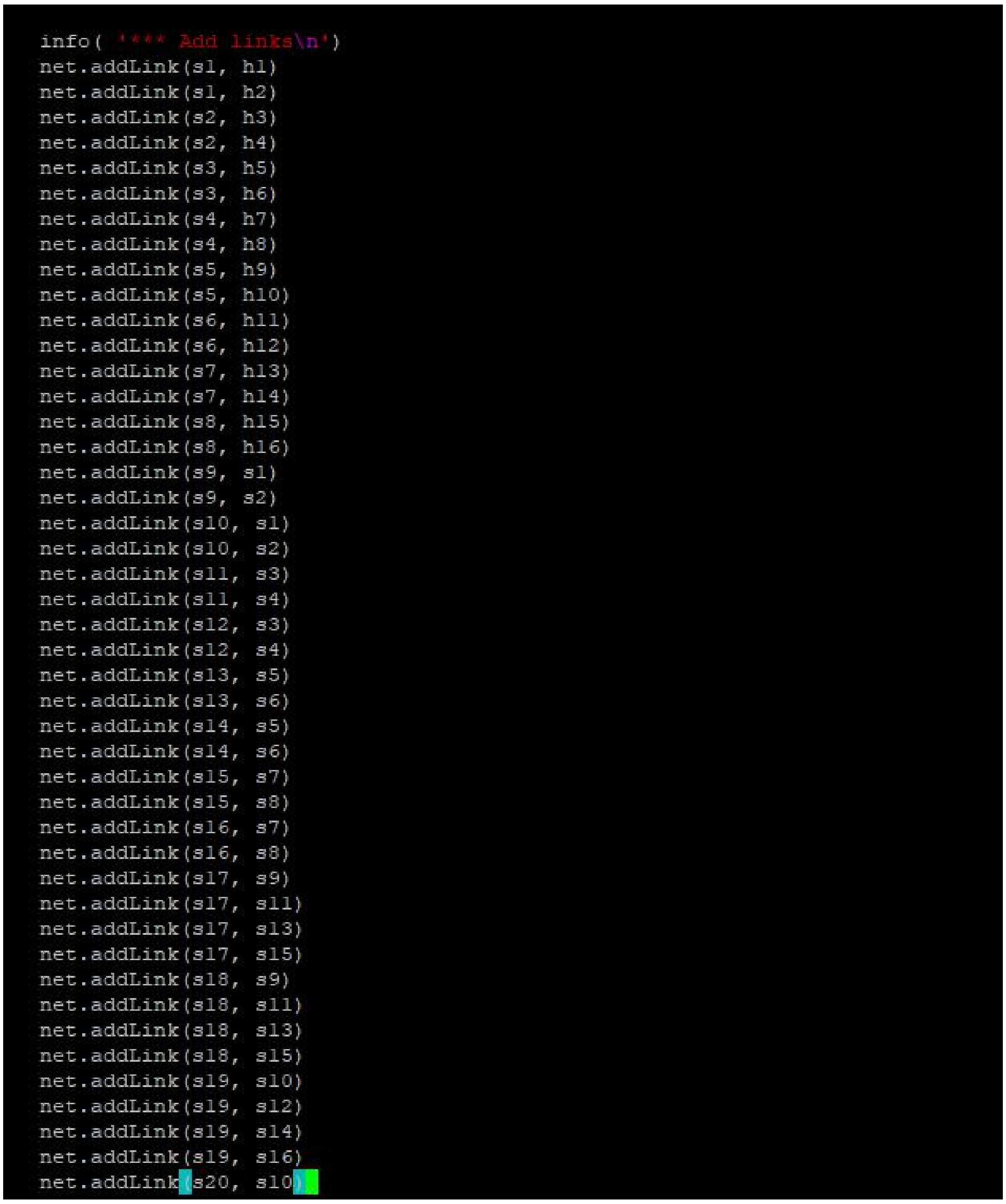
```
    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    s11 = net.addSwitch('s11', cls=OVSKernelSwitch, dpid='10.1.2.1')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='10.1.0.1')
    s16 = net.addSwitch('s16', cls=OVSKernelSwitch, dpid='10.3.3.1')
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch, dpid='10.1.1.1')
    s17 = net.addSwitch('s17', cls=OVSKernelSwitch, dpid='10.4.1.1')
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch, dpid='10.3.0.1')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch, dpid='10.2.0.1')
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch, dpid='10.2.1.1')
    s19 = net.addSwitch('s19', cls=OVSKernelSwitch, dpid='10.4.2.1')
    s20 = net.addSwitch('s20', cls=OVSKernelSwitch, dpid='10.4.2.2')
    s18 = net.addSwitch('s18', cls=OVSKernelSwitch, dpid='10.4.1.2')
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch, dpid='10.0.2.1')
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch, dpid='10.3.1.1')
    s10 = net.addSwitch('s10', cls=OVSKernelSwitch, dpid='10.0.3.1')
    s12 = net.addSwitch('s12', cls=OVSKernelSwitch, dpid='10.1.3.1')
    s14 = net.addSwitch('s14', cls=OVSKernelSwitch, dpid='10.2.3.1')
    s15 = net.addSwitch('s15', cls=OVSKernelSwitch, dpid='10.3.2.1')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='10.0.0.1')
    s13 = net.addSwitch('s13', cls=OVSKernelSwitch, dpid='10.2.2.1')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='10.0.1.1')
```

④ 创建 16 个节点并配置相应的 ip

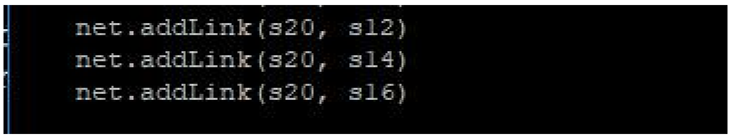


```
info( '*** Add hosts\n')
h4 = net.addHost('h4', cls=Host, ip='10.0.1.3', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.1.0.2', defaultRoute=None)
h6 = net.addHost('h6', cls=Host, ip='10.1.0.3', defaultRoute=None)
h7 = net.addHost('h7', cls=Host, ip='10.1.1.2', defaultRoute=None)
h8 = net.addHost('h8', cls=Host, ip='10.1.1.3', defaultRoute=None)
h9 = net.addHost('h9', cls=Host, ip='10.2.0.2', defaultRoute=None)
h10 = net.addHost('h10', cls=Host, ip='10.2.0.3', defaultRoute=None)
h11 = net.addHost('h11', cls=Host, ip='10.2.1.2', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.2.1.3', defaultRoute=None)
h13 = net.addHost('h13', cls=Host, ip='10.3.0.2', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.3.0.3', defaultRoute=None)
h15 = net.addHost('h15', cls=Host, ip='10.3.1.2', defaultRoute=None)
h16 = net.addHost('h16', cls=Host, ip='10.3.1.3', defaultRoute=None)
h1 = net.addHost('h1', cls=Host, ip='10.0.0.2', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.1.2', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.03', defaultRoute=None)
```

⑤ 添加链路

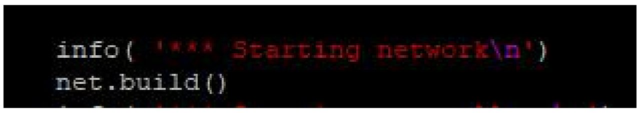


```
info( '*** Add links\n')
net.addLink(s1, h1)
net.addLink(s1, h2)
net.addLink(s2, h3)
net.addLink(s2, h4)
net.addLink(s3, h5)
net.addLink(s3, h6)
net.addLink(s4, h7)
net.addLink(s4, h8)
net.addLink(s5, h9)
net.addLink(s5, h10)
net.addLink(s6, h11)
net.addLink(s6, h12)
net.addLink(s7, h13)
net.addLink(s7, h14)
net.addLink(s8, h15)
net.addLink(s8, h16)
net.addLink(s9, s1)
net.addLink(s9, s2)
net.addLink(s10, s1)
net.addLink(s10, s2)
net.addLink(s11, s3)
net.addLink(s11, s4)
net.addLink(s12, s3)
net.addLink(s12, s4)
net.addLink(s13, s5)
net.addLink(s13, s6)
net.addLink(s14, s5)
net.addLink(s14, s6)
net.addLink(s15, s7)
net.addLink(s15, s8)
net.addLink(s16, s7)
net.addLink(s16, s8)
net.addLink(s17, s9)
net.addLink(s17, s11)
net.addLink(s17, s13)
net.addLink(s17, s15)
net.addLink(s18, s9)
net.addLink(s18, s11)
net.addLink(s18, s13)
net.addLink(s18, s15)
net.addLink(s19, s10)
net.addLink(s19, s12)
net.addLink(s19, s14)
net.addLink(s19, s16)
net.addLink(s20, s10)
```



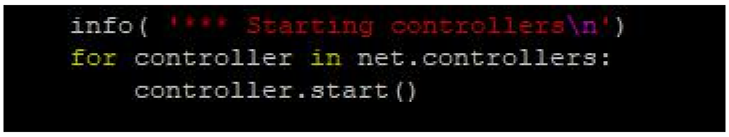
```
net.addLink(s20, s12)
net.addLink(s20, s14)
net.addLink(s20, s16)
```

⑥　启动网络



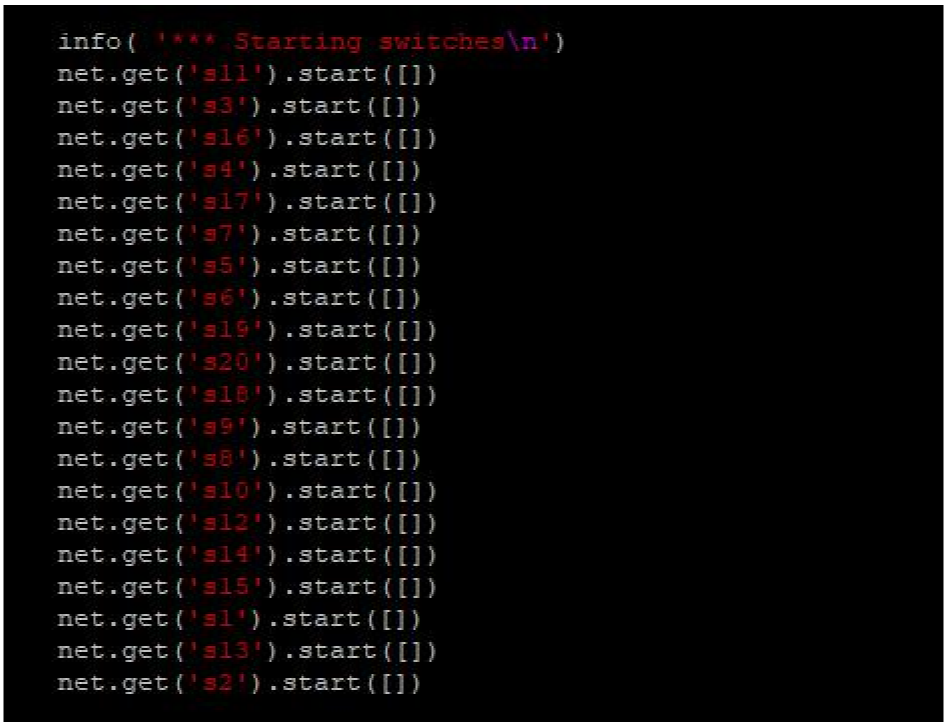
```
info( '*** Starting network\n')
net.build()
```

⑦　启动控制器，但是因为本结构没有用到控制器，所以该语句并没有效果



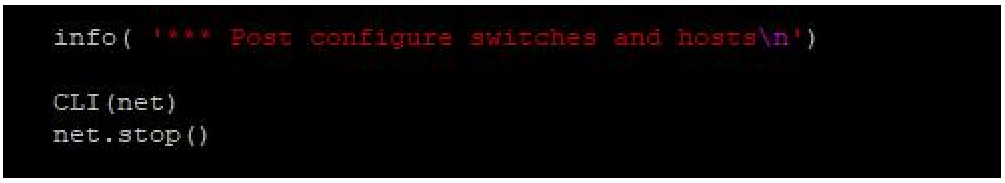
```
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()
```

⑧ 启动交换机



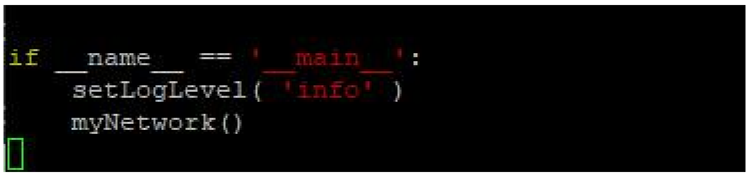
```
info( '*** Starting switches\n')
net.get('s11').start([])
net.get('s3').start([])
net.get('s16').start([])
net.get('s4').start([])
net.get('s17').start([])
net.get('s7').start([])
net.get('s5').start([])
net.get('s6').start([])
net.get('s19').start([])
net.get('s20').start([])
net.get('s18').start([])
net.get('s9').start([])
net.get('s8').start([])
net.get('s10').start([])
net.get('s12').start([])
net.get('s14').start([])
net.get('s15').start([])
net.get('s1').start([])
net.get('s13').start([])
net.get('s2').start([])
```

⑨ 结束网络



```
info( '*** Post configure switches and hosts\n')

CLI(net)
net.stop()
```

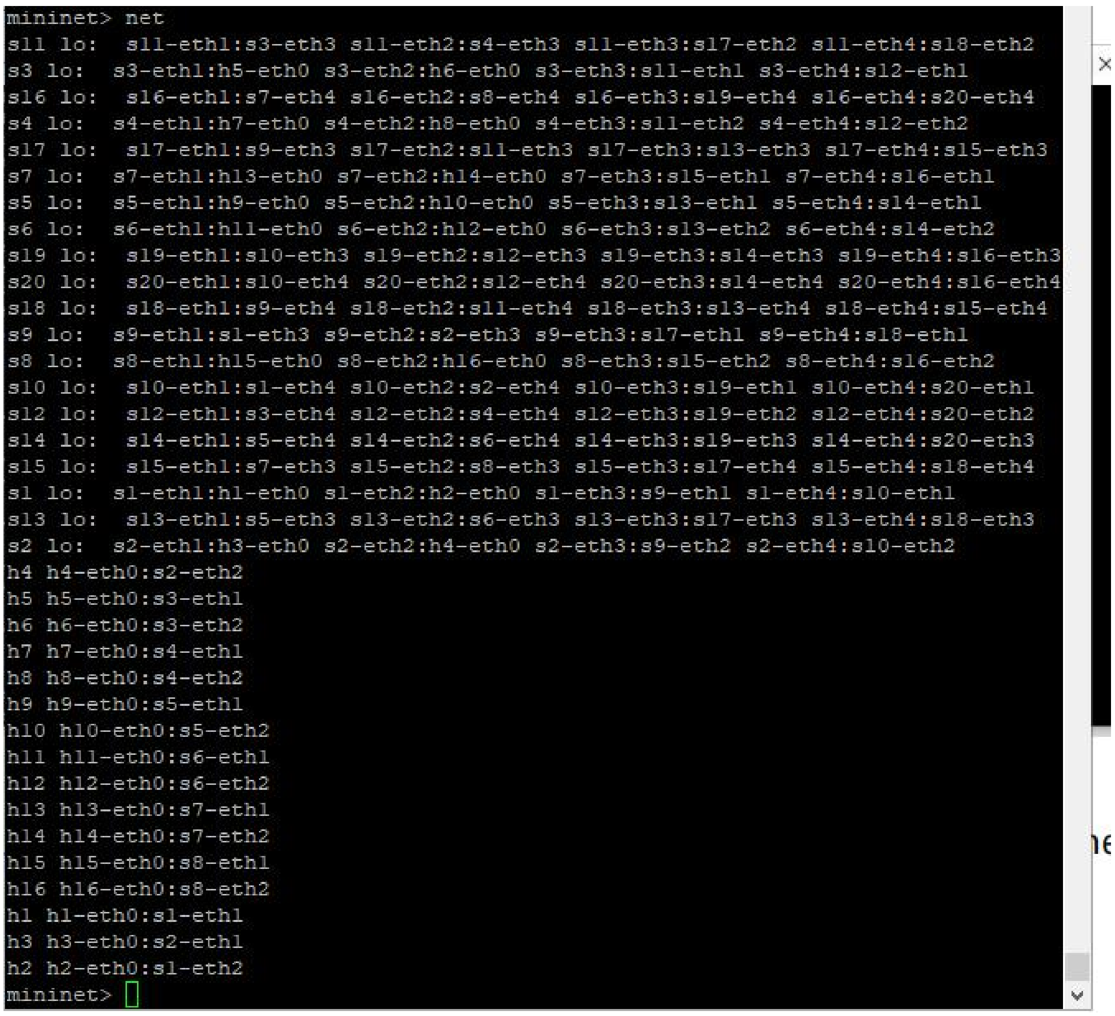
⑩ 主函数



```
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

（3）拓扑测试

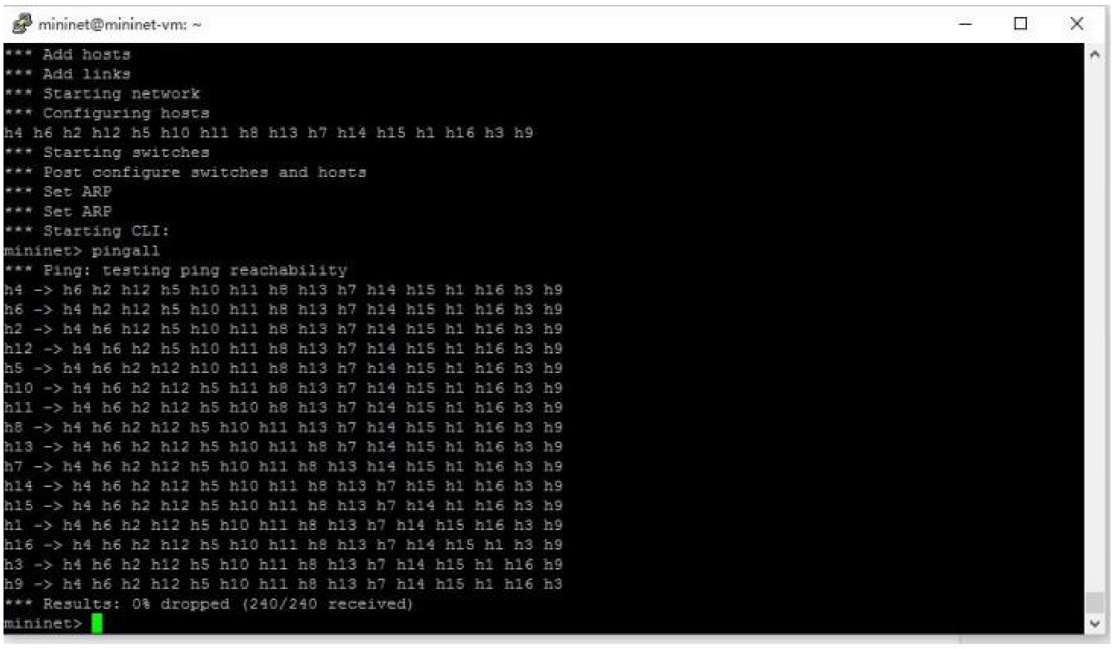
① 使用 net 命令查看当前的网络结构，如图 4.2。



```
mininet> net
s11 lo:  s11-eth1:s3-eth3 s11-eth2:s4-eth3 s11-eth3:s17-eth2 s11-eth4:s18-eth2
s3 lo:  s3-eth1:h5-eth0 s3-eth2:h6-eth0 s3-eth3:s11-eth1 s3-eth4:s12-eth1
s16 lo:  s16-eth1:s7-eth4 s16-eth2:s8-eth4 s16-eth3:s19-eth4 s16-eth4:s20-eth4
s4 lo:  s4-eth1:h7-eth0 s4-eth2:h8-eth0 s4-eth3:s11-eth2 s4-eth4:s12-eth2
s17 lo:  s17-eth1:s9-eth3 s17-eth2:s11-eth3 s17-eth3:s13-eth3 s17-eth4:s15-eth3
s7 lo:  s7-eth1:h13-eth0 s7-eth2:h14-eth0 s7-eth3:s15-eth1 s7-eth4:s16-eth1
s5 lo:  s5-eth1:h9-eth0 s5-eth2:h10-eth0 s5-eth3:s13-eth1 s5-eth4:s14-eth1
s6 lo:  s6-eth1:h11-eth0 s6-eth2:h12-eth0 s6-eth3:s13-eth2 s6-eth4:s14-eth2
s19 lo:  s19-eth1:s10-eth3 s19-eth2:s12-eth3 s19-eth3:s14-eth3 s19-eth4:s16-eth3
s20 lo:  s20-eth1:s10-eth4 s20-eth2:s12-eth4 s20-eth3:s14-eth4 s20-eth4:s16-eth4
s18 lo:  s18-eth1:s9-eth4 s18-eth2:s11-eth4 s18-eth3:s13-eth4 s18-eth4:s15-eth4
s9 lo:  s9-eth1:s1-eth3 s9-eth2:s2-eth3 s9-eth3:s17-eth1 s9-eth4:s18-eth1
s8 lo:  s8-eth1:h15-eth0 s8-eth2:h16-eth0 s8-eth3:s15-eth2 s8-eth4:s16-eth2
s10 lo:  s10-eth1:s1-eth4 s10-eth2:s2-eth4 s10-eth3:s19-eth1 s10-eth4:s20-eth1
s12 lo:  s12-eth1:s3-eth4 s12-eth2:s4-eth4 s12-eth3:s19-eth2 s12-eth4:s20-eth2
s14 lo:  s14-eth1:s5-eth4 s14-eth2:s6-eth4 s14-eth3:s19-eth3 s14-eth4:s20-eth3
s15 lo:  s15-eth1:s7-eth3 s15-eth2:s8-eth3 s15-eth3:s17-eth4 s15-eth4:s18-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s9-eth1 s1-eth4:s10-eth1
s13 lo:  s13-eth1:s5-eth3 s13-eth2:s6-eth3 s13-eth3:s17-eth3 s13-eth4:s18-eth3
s2 lo:  s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:s9-eth2 s2-eth4:s10-eth2
h4 h4-eth0:s2-eth2
h5 h5-eth0:s3-eth1
h6 h6-eth0:s3-eth2
h7 h7-eth0:s4-eth1
h8 h8-eth0:s4-eth2
h9 h9-eth0:s5-eth1
h10 h10-eth0:s5-eth2
h11 h11-eth0:s6-eth1
h12 h12-eth0:s6-eth2
h13 h13-eth0:s7-eth1
h14 h14-eth0:s7-eth2
h15 h15-eth0:s8-eth1
h16 h16-eth0:s8-eth2
h1 h1-eth0:s1-eth1
h3 h3-eth0:s2-eth1
h2 h2-eth0:s1-eth2
mininet> 
```

图 4.2 不含有控制器网络的拓扑结构

② 使用 ping 来查看当前网络的连通情况， 其结果如图 4.3



mininet@mininet-vm: ~

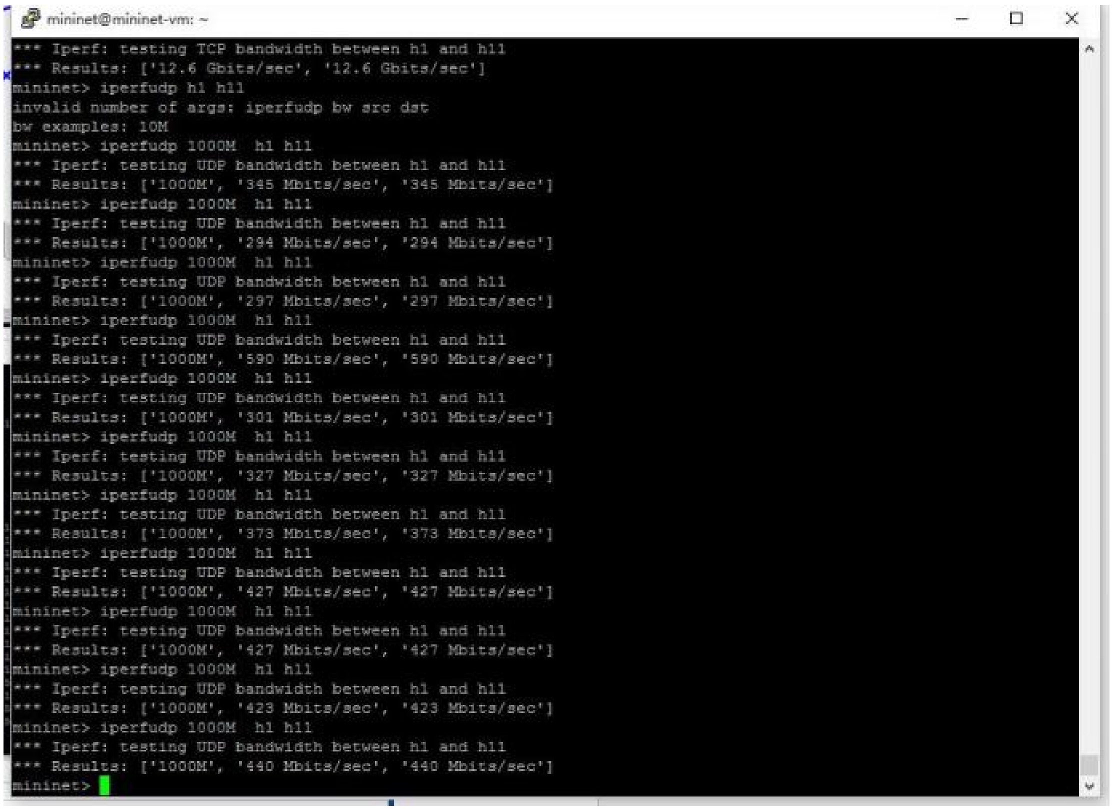
```
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
*** Starting switches
*** Post configure switches and hosts
*** Set ARP
*** Set ARP
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h4 -> h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h6 -> h4 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h2 -> h4 h6 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h12 -> h4 h6 h2 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h5 -> h4 h6 h2 h12 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h10 -> h4 h6 h2 h12 h5 h11 h8 h13 h7 h14 h15 h1 h16 h3 h9
h11 -> h4 h6 h2 h12 h5 h10 h8 h13 h7 h14 h15 h1 h16 h3 h9
h8 -> h4 h6 h2 h12 h5 h10 h11 h13 h7 h14 h15 h1 h16 h3 h9
h13 -> h4 h6 h2 h12 h5 h10 h11 h8 h7 h14 h15 h1 h16 h3 h9
h7 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h14 h15 h1 h16 h3 h9
h14 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h15 h1 h16 h3 h9
h15 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h1 h16 h3 h9
h1 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h16 h3 h9
h16 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h3 h9
h3 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h9
h9 -> h4 h6 h2 h12 h5 h10 h11 h8 h13 h7 h14 h15 h1 h16 h3
*** Results: 0% dropped (240/240 received)
mininet> 
```

图 4.3 不带控制器网络的连通情况

（4）链路均衡性测试

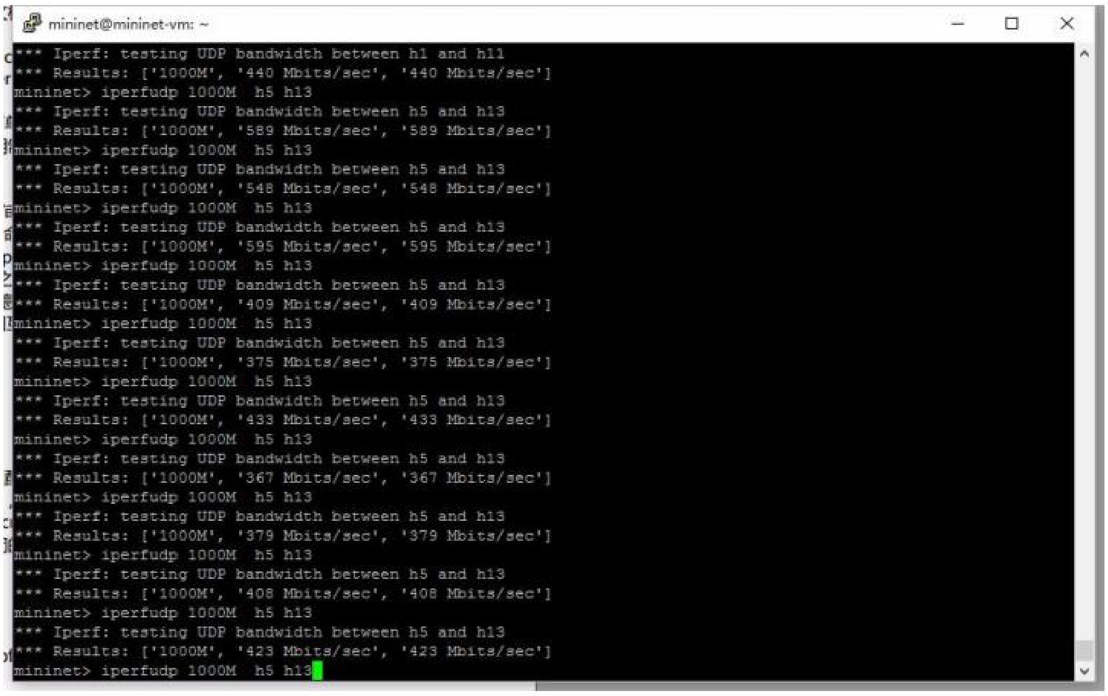
下面通过实验来证明链路的带宽分布式相对均匀的，在本部分中使用 iperfudp 命令来测试网络的带宽，并在此过程中使用 wireshark 抓包。抓包过程中端口设置为 any，同时排除掉虚拟机和主机之间通讯的包。

为了实验更具有代表性，我们选择图 4.1 中的 h1 和 h11, h5 和 h13 来测试最大 udp 带宽，设置物理链路为 1000M 带宽的网络。之所以选择这两对主机节点是因为它们属于不同的 pod，比较有代表性。其结果如图 4.4 和图 4.5。



```
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['12.6 Gbits/sec', '12.6 Gbits/sec']
mininet> iperfudp h1 h11
invalid number of args: iperfudp bw src dst
bw examples: 10M
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '345 Mbits/sec', '345 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '294 Mbits/sec', '294 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '297 Mbits/sec', '297 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '590 Mbits/sec', '590 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '301 Mbits/sec', '301 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '327 Mbits/sec', '327 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '373 Mbits/sec', '373 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '427 Mbits/sec', '427 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '427 Mbits/sec', '427 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '423 Mbits/sec', '423 Mbits/sec']
mininet> iperfudp 1000M  h1 h11
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '440 Mbits/sec', '440 Mbits/sec']
mininet>
```

图 4.4 测试 h1 和 h11 之间的链路带宽



```
*** Iperf: testing UDP bandwidth between h1 and h11
*** Results: ['1000M', '440 Mbits/sec', '440 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '589 Mbits/sec', '589 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '548 Mbits/sec', '548 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '595 Mbits/sec', '595 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '409 Mbits/sec', '409 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '375 Mbits/sec', '375 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '433 Mbits/sec', '433 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '367 Mbits/sec', '367 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '379 Mbits/sec', '379 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '408 Mbits/sec', '408 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
*** Iperf: testing UDP bandwidth between h5 and h13
*** Results: ['1000M', '423 Mbits/sec', '423 Mbits/sec']
mininet> iperfudp 1000M  h5 h13
```

图 4.5 测试 h5 和 h13 之间的链路带宽

我们利用 wireshark 进行抓包，可以得到得到图 4.6 和图 4.7 所示的 IO 图。

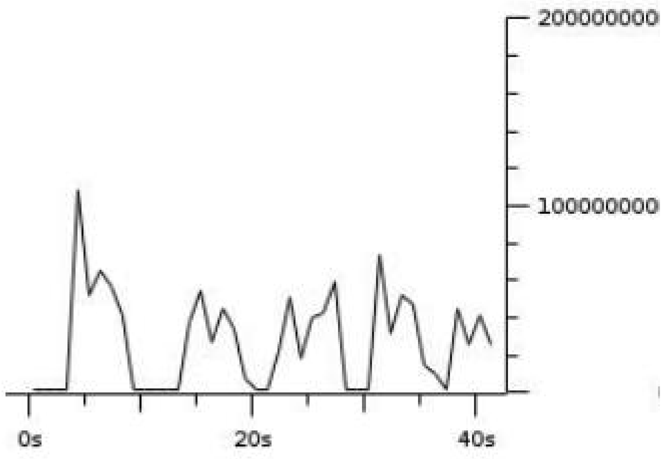


图 4.6 h1 和 h11 的 IO 图

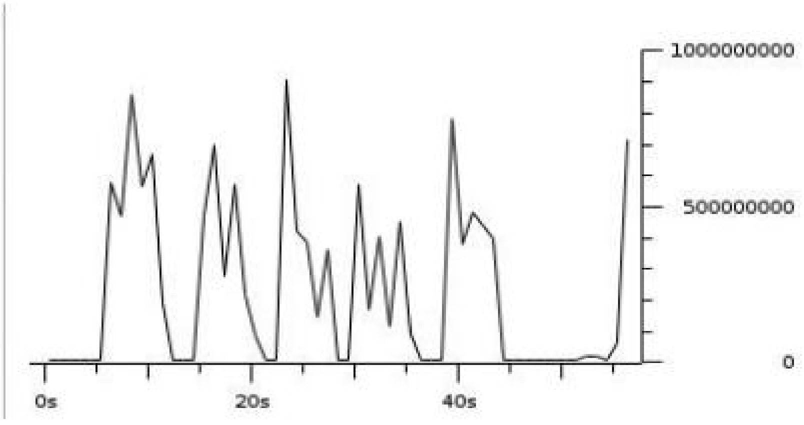


图 4.7 h5 和 h13 的 IO 图

从(h1, h11)和(h5, h13)的 IO 图，我们可以发现这二者的最大带宽基本相同。所以说该 FatTree 具有一定的均匀性

五．总结和体会

1. 通过该实验，熟悉了 mininet 软件，并能够 mininet 来实现简单的网络结构。

2. mininet 中有 mininedit 它使得网络结构的创建变得可视化，我们可以通过简单的拖拽实现网络。

3. 本实验使用了两种控制方式来实现了 FatTree。在实验的过程中，也不难看出两种方式各有优劣，在 pingall 的过程中，不带控制器的拓扑 ping 的响应时间更快一些，因为不需要和控制器之间进行通讯。然后带有控制器的拓扑在网络管理上也有一定的灵活性，不需要手动建立流表项，使用起来更加灵活方便。