高级计算机网络第二次实验报告

殷康麒 SA16011112

1. 目标

如下图图 1.1 所示，图为一个 FatTree 拓扑。现在需要实现以下的目标：

- 使用 MiniNet 和 Miniedit 完成拓扑结构的建立
- 带控制器的 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。
- 建立不带控制器的 FatTree 的网络，需要手动添加相关的流表项，并证明带宽的相对均匀性。

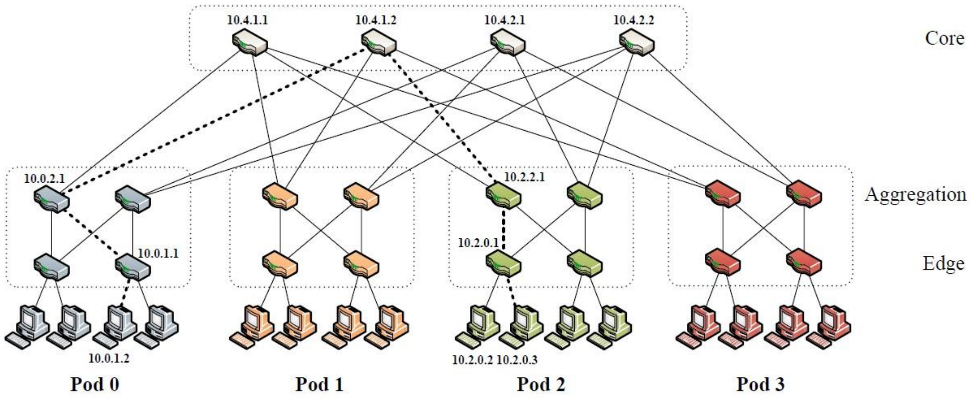


图 1.1

2. 相关环境配置

相关环境的配置在相关的指导 PPT 上已经有详细的步骤，在此不再赘述。需要从百度云盘上下载两个镜像，同时需要安装好 Putty 和 Xterm Server，保证后续试验的正常进行。

3. 带控制器的网络

在本部分中，分别介绍网络拓扑的相关的建立，相关代码的解释，以及其对应的仿真结果及分析，以及最后人为破坏部分链路之后网络的状况。

3.1. 网络拓扑的建立

如下图 3.1 所示，使用 Miniedit 直接建立图 1.1 的拓扑结构并完成连线。建立完成之后输出拓扑文件 TopoFinal.py 文件。

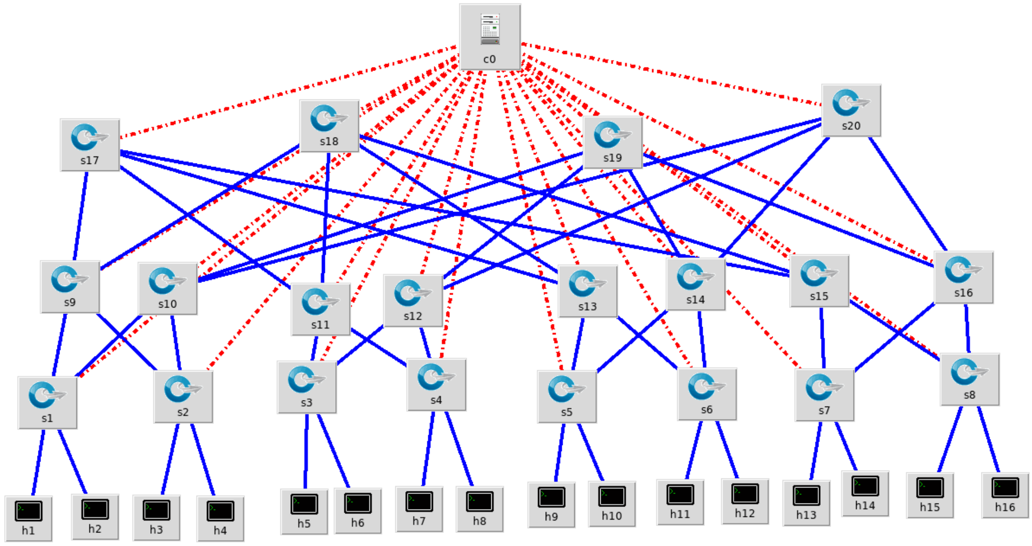


图 3.1

3.2. 拓扑的代码以解释

下表为带控制器的网络拓扑代码，完整的代码见该目录下的 mininet1.py 文件。

```
from mininet.net import Mininet                    //导入需要的包
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

def myNetwork():

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

    info( '*** Adding controller\n' )                //添加控制器
```

```
c0=net.addController('c0',
                     controller=RemoteController,
                     ip="192.168.1.137",
                     port=6653)

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

info( '*** Add hosts\n')               //添加主机
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)
h15 = net.addHost('h15', cls=Host, ip='10.3.1.2', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.2.1.3', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.3.0.3', defaultRoute=None)
h13 = net.addHost('h13', cls=Host, ip='10.3.0.2', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.3', 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)
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)
```

```
info( '*** Add links\n')                                  //将链路链接起来
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(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)

info( '*** Starting network\n')                    //启动网络
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()

info( '*** Starting switches\n')                   //将交换机添加到网络中
net.get('s15').start([c0])
net.get('s2').start([c0])
net.get('s17').start([c0])
net.get('s5').start([c0])
net.get('s18').start([c0])
net.get('s6').start([c0])
net.get('s7').start([c0])
net.get('s14').start([c0])
net.get('s20').start([c0])
net.get('s9').start([c0])
net.get('s8').start([c0])
net.get('s10').start([c0])
net.get('s4').start([c0])
net.get('s12').start([c0])
net.get('s13').start([c0])
net.get('s19').start([c0])
net.get('s16').start([c0])
net.get('s1').start([c0])
net.get('s3').start([c0])
net.get('s11').start([c0])

info( '*** Post configure switches and hosts\n')          //将交换机连接到控制器
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')

    info('*** Enable spanning tree\n')

    #net.pingAll()

    CLI(net)
    net.stop()

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

3.3. 拓扑的测试

3.3.1. 正常测试

首先使用 pingall 命令测试所有主机的联通状况，如下图 3.2 所示。

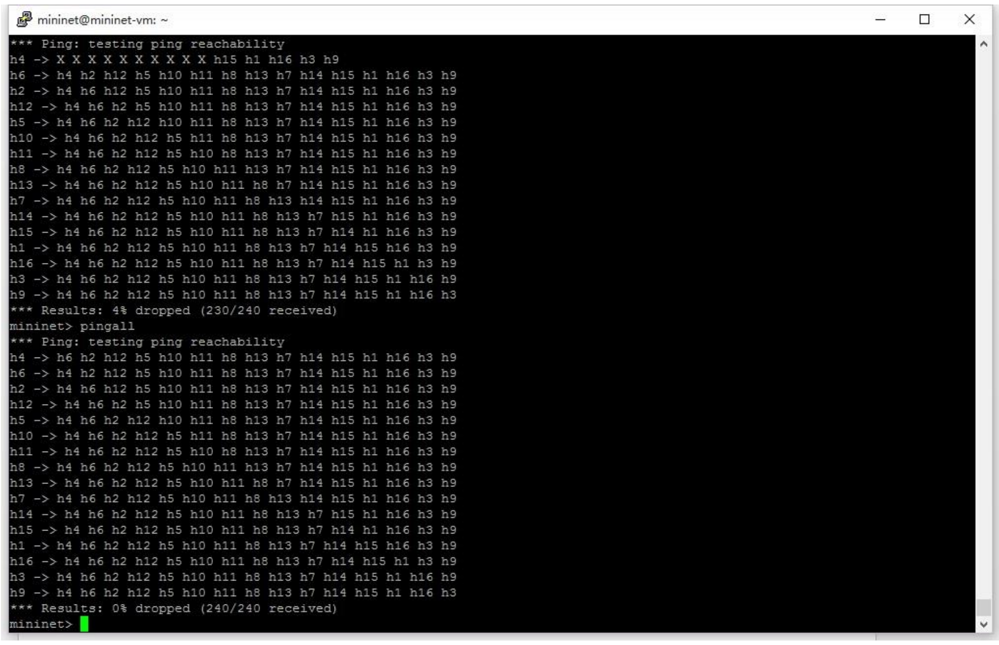
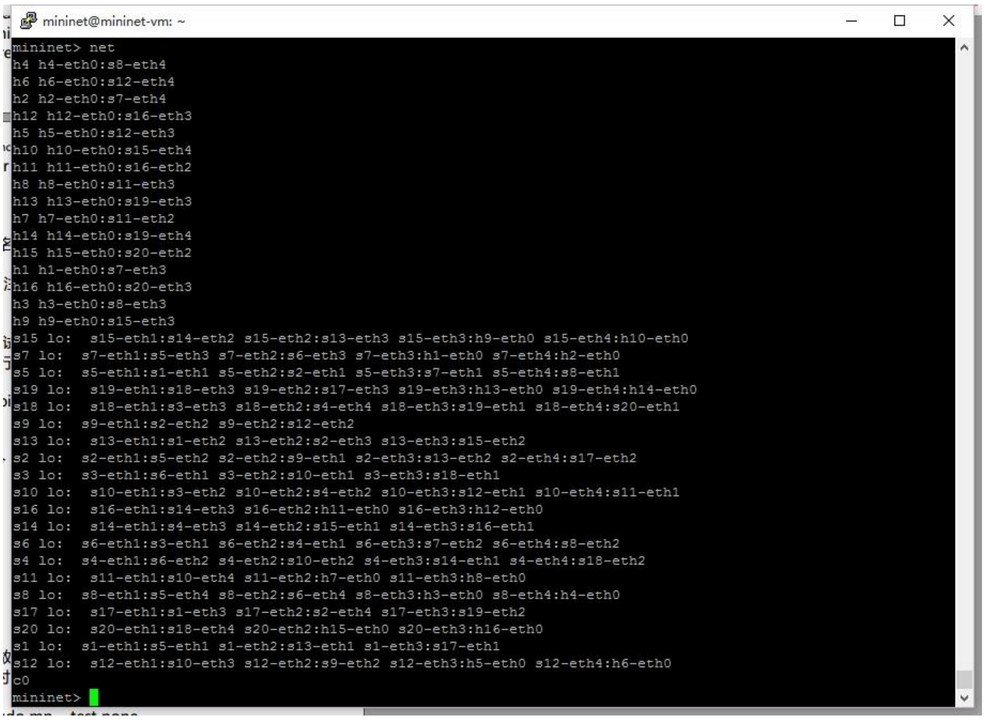


图 3.2

可以观察到在 pingall 的过程中，第一次 ping 的过程中会出现开始部分主机 ping 不通的情况，但是在后续 ping 的过程中全部正常。猜测可能是网络刚启动的时候交换机还没有全部正确连接到控制器的原因。

再使用 net 命令观察网络中所有链路的连接情况，如下图 3.3 所示，说明网络中所有的主机连接正常。

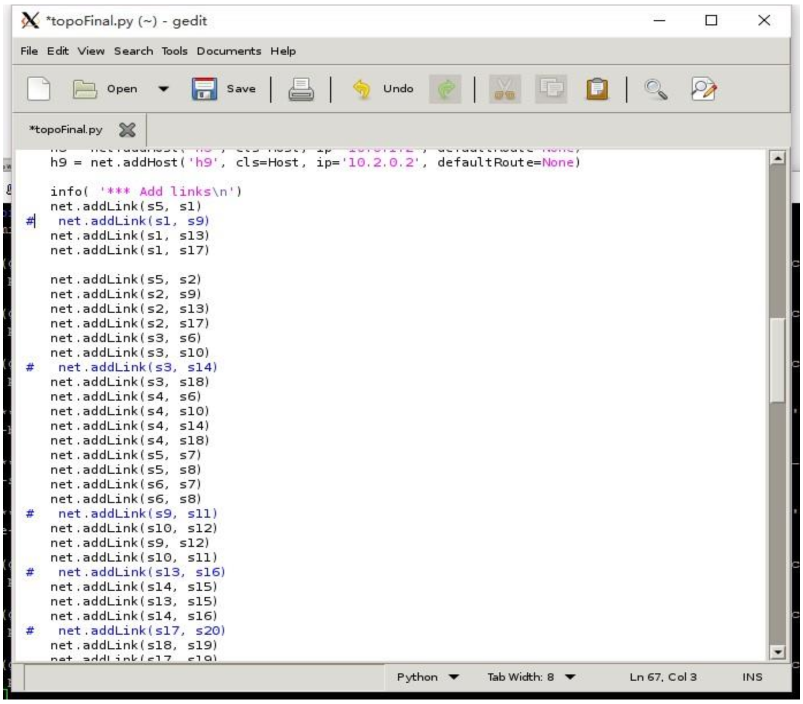


图

3.3

3.3.2. 破坏部分链路情况

破坏部分链路，然后观察网络的情况。如下图 3.4 所示，注释掉其中的五条链路。



图

3.4

然后再次重新建立拓扑，使用 pingall 命令检查链路联通状况，如下图 3.5 所示。

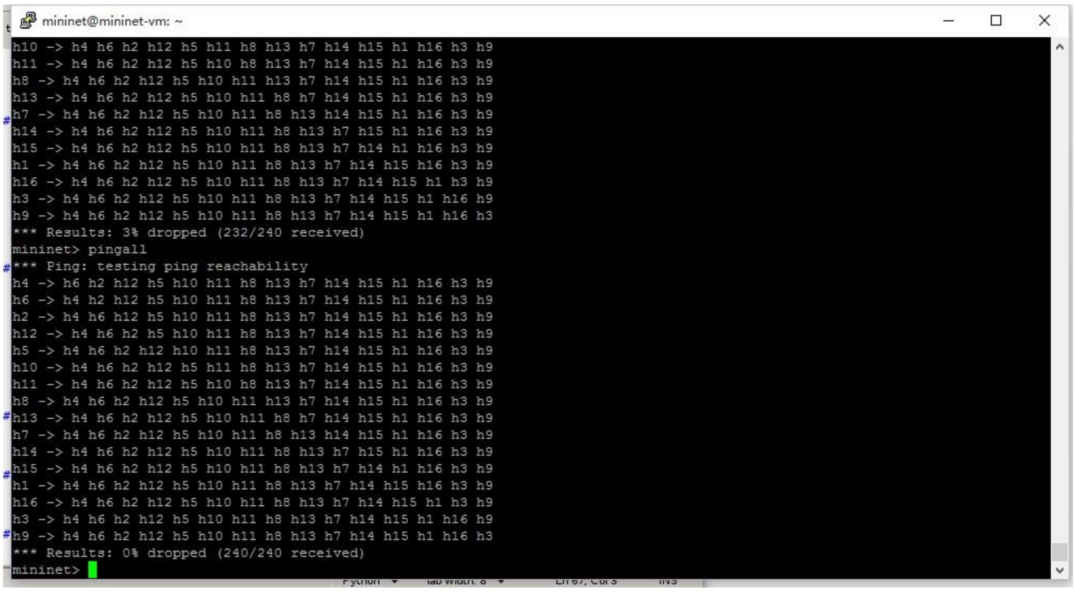


图 3.5

由此可见，在破坏部分链路的情况下，网络仍然可以正常连通。

4. 不带控制器的网络

在本节中将主要介绍不带控制器的网络拓扑，同时解释相应部分的关键代码。

然后是正常的实验结果，包括 pingall 测试和 net 测试。最后是通过 wireshark 和 iperfudp 命令证明网络链路带宽的相对均衡性。

4.1. 网络拓扑的建立

同样，和 3.1 中的步骤一样，使用 MiniEdit 建立拓扑并导出相应的 topoSinge11.py 文件，拓扑如下图 4.1 所示。

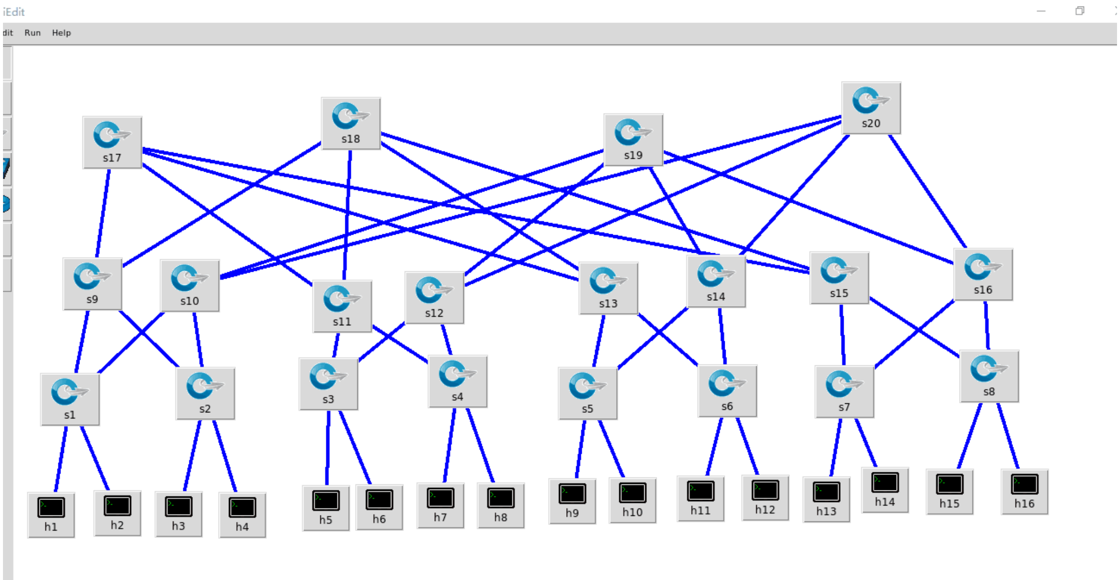


图 4.1

4.2. 部分关键代码及解释

下表将列出部分关键的代码及其相关的解释，完整的代码见目录下的 mininet2.py 文件所示。

```
from mininet.net import Mininet                    //导入需要的包
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

def myNetwork():

    net = Mininet( topo=None,
```

```
        build=False,
        ipBase='10.0.0.0/8')

info( '*** Adding controller\n' )                    //添加控制机
c0=net.addController(name='c0',
          controller=Controller,
          protocol='tcp',
          port=6633)

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

info( '*** Add hosts\n')                             //添加主机
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)
h6 = net.addHost('h6', cls=Host, ip='10.1.0.3', defaultRoute=None)
h15 = net.addHost('h15', cls=Host, ip='10.3.1.2', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.2.1.3', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.3.0.3', defaultRoute=None)
h13 = net.addHost('h13', cls=Host, ip='10.3.0.2', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.3', 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)
```

```
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)

info( '*** Add links\n')                    //将所有链路连接起来
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(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)

info( '*** Starting network\n')          //开启网络
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
   controller.start()

info( '*** Starting switches\n')     //将交换机添加到网络中去，没有连接任何交换机
net.get('s15').start([c0])
net.get('s20').start([c0])
net.get('s2').start([c0])
net.get('s3').start([c0])
net.get('s5').start([c0])
net.get('s17').start([c0])
net.get('s6').start([c0])
net.get('s13').start([c0])
net.get('s11').start([c0])
net.get('s18').start([c0])
net.get('s8').start([c0])
net.get('s19').start([c0])
net.get('s1').start([c0])
net.get('s7').start([c0])
net.get('s12').start([c0])
net.get('s10').start([c0])
net.get('s16').start([c0])
net.get('s9').start([c0])
net.get('s14').start([c0])
net.get('s4').start([c0])

info( '*** Post configure switches and hosts\n')  //给每个交换机分配 IP 地址
s15.cmd('ifconfig s15 10.3.2.1')
s20.cmd('ifconfig s20 10.4.2.2')
s2.cmd('ifconfig s2 10.0.1.1')
s3.cmd('ifconfig s3 10.1.0.1')
s5.cmd('ifconfig s5 10.2.0.1')
s17.cmd('ifconfig s17 10.4.1.1')
s6.cmd('ifconfig s6 10.2.1.1')
s13.cmd('ifconfig s13 10.2.2.1')
s11.cmd('ifconfig s11 10.1.2.1')
s18.cmd('ifconfig s18 10.4.1.2')
s8.cmd('ifconfig s8 10.3.1.1')
```

```
    s19.cmd('ifconfig s19 10.4.2.1')
    s1.cmd('ifconfig s1 10.0.0.1')
    s7.cmd('ifconfig s7 10.3.0.1')
    s12.cmd('ifconfig s12 10.1.3.1')
    s10.cmd('ifconfig s10 10.0.3.1')
    s16.cmd('ifconfig s16 10.3.3.1')
    s9.cmd('ifconfig s9 10.0.2.1')
    s14.cmd('ifconfig s14 10.2.3.1')
    s4.cmd('ifconfig s4 10.1.1.1')

    CLI(net)
    net.stop()

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

4.3. 拓扑的测试

同样，与前面的实验类似，使用 pingall 命令查看网络中的拓扑，如下图 4.2 所示。

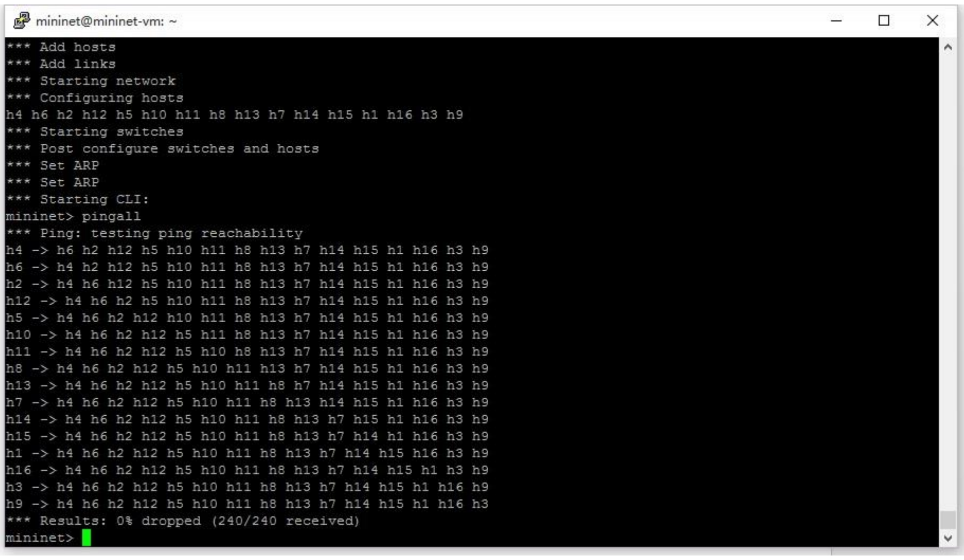
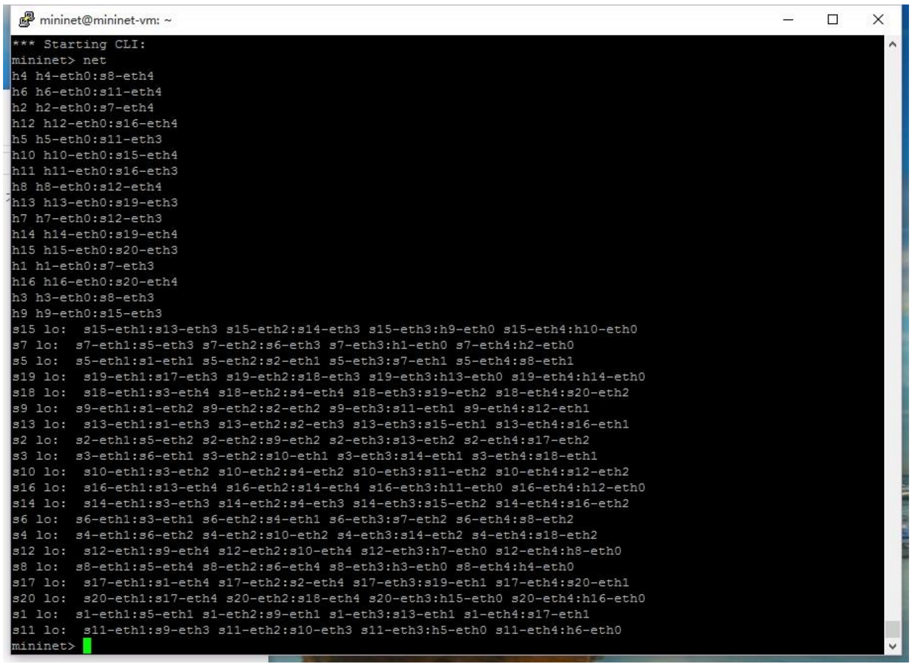


图 4.2

由上图可见，所有主机两两之间都可以 ping 通，在观察图 4.3 中 net 命令结果。



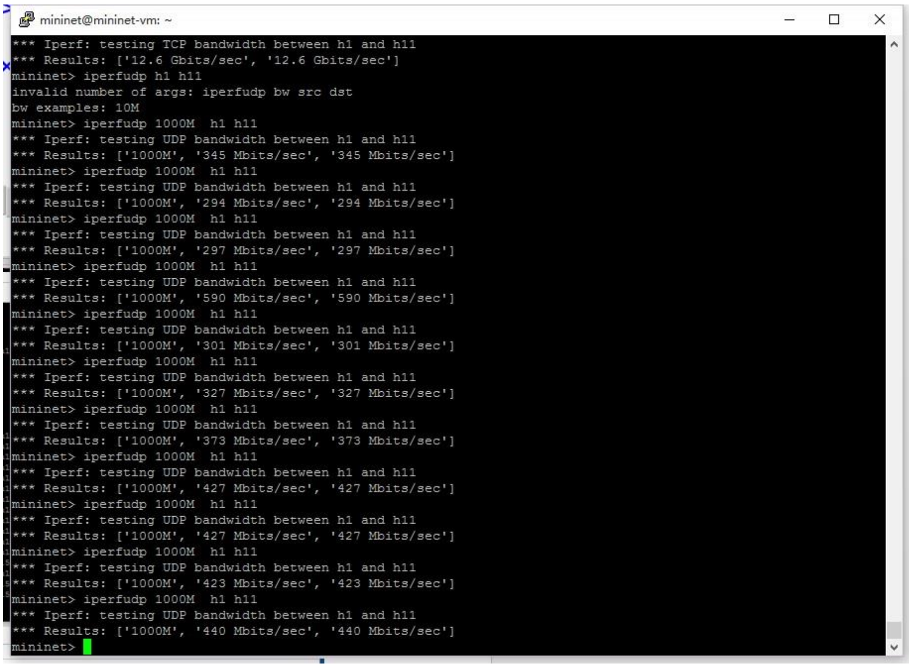
```
mininet@mininet-vm: ~
*** Starting CLI:
mininet> net
h4 h4-eth0:s8-eth4
h6 h6-eth0:s11-eth4
h2 h2-eth0:s7-eth4
h12 h12-eth0:s16-eth4
h5 h5-eth0:s11-eth3
h10 h10-eth0:s15-eth4
h11 h11-eth0:s16-eth3
h8 h8-eth0:s12-eth4
h13 h13-eth0:s19-eth3
h7 h7-eth0:s12-eth3
h14 h14-eth0:s19-eth4
h15 h15-eth0:s20-eth3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

图 4.3

可以看出所有的主机之间链路正常，并和 FatTree 的链路规则一致。

4.4. 链路相对均衡性验证

下面通过实验来证明链路的带宽分布是相对均匀的，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，同时排除掉虚拟机与主机之间通讯的包。



```
mininet@mininet-vm: ~
*** 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

如上图 4.4 所示，选择了图 4.1 拓扑中的 h1 和 h11 主机测试最大 udp 带宽，设置物理链路为 1000M 带宽的网络。

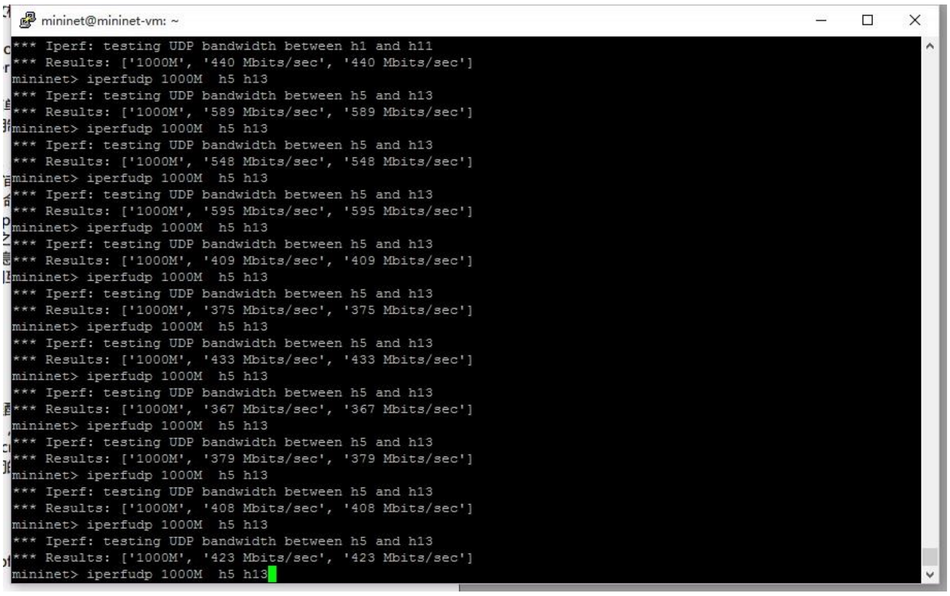


图 4.5

如上图 4.5 所示，选择 h5 和 h13 再做类似的测试。由图 4.1 可以看出，（h1,h11）和（h5,h13）两对主机属于不同的 pod 以及处于 pod 中不同的位置，具有一定的代表性，测试的结果如上图所示。

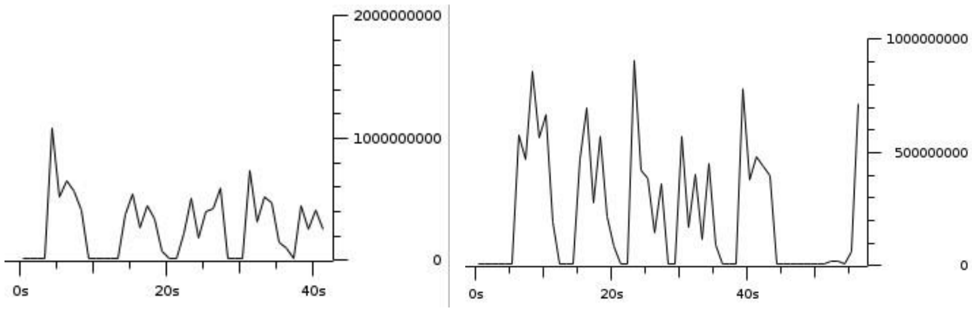


图 4.6

上图 4.6 为在执行 iperf 命令的时候使用 wireshark 抓包后的 IO 图（图中纵坐标尺度稍有不同），图中纵坐标单位为 B/s，左图为（h1，h11）的 IO 图，右图为（h5，h13）的 IO 图。可以由上图可以简单的看出，（h1,h11）和(h5,h13)之间的最大带宽基本相同。所以可以观察出该 FatTree 拓扑还是具有一定的均匀性的。

5. 总结

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也是各有优劣的，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。然而，带有控制器的拓扑在网络管理上面也有一定的灵活性，不需要手动建立流表项，使用起来更加方便灵活。