

Lab1——Mininet

(一) 搭建 Fat Tree 网络拓扑 (k=2)

```
test@sdnexp:~/Desktop$ sudo python FatTree.py
*** Creating network
*** Adding hosts:
H1 H2
*** Adding switches:
AS1 AS2 CS1 ES1 ES2
*** Adding links:
(AS1, ES1) (AS2, ES2) (CS1, AS1) (CS1, AS2) (ES1, H1) (ES2, H2)
*** Configuring hosts
H1 H2
*** Starting controller

*** Starting 5 switches
AS1 AS2 CS1 ES1 ES2 ...
*** Starting CLI:
mininet> links
AS1-eth2<->ES1-eth1 (OK OK)
AS2-eth2<->ES2-eth1 (OK OK)
CS1-eth1<->AS1-eth1 (OK OK)
CS1-eth2<->AS2-eth1 (OK OK)
ES1-eth2<->H1-eth0 (OK OK)
ES2-eth2<->H2-eth0 (OK OK)
mininet> pingall
*** Ping: testing ping reachability
H1 -> X
H2 -> X
*** Results: 100% dropped (0/2 received)
```

搭建完拓扑后，两主机无法 ping 通。

运行命令：H1 ping -c3 H2。

```
mininet> H1 ping -c3 H2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2045ms
pipe 3
```

同时，使用 wireshark 抓包，分析与 H1 相连的交换机两端口的数据包。

ES1 与 H1 相连的端口捕获了 H1 发送的 3 个 ARP 请求报文：

*ES1-eth2						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-/> Expression... +						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	b6:8b:a1:38:d8:bf	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
2	1.018152102	b6:8b:a1:38:d8:bf	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
3	2.045304730	b6:8b:a1:38:d8:bf	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1

ES1 与 AS1 相连的端口没有捕获到任何报文：



这说明交换机在收到主机发来的数据包后并没有将其转发出去，因而两主机无法 ping 通。

在终端输入 `sudo ovs-vsctl show`，查看交换机的基本信息：

```
Bridge "ES1"
  fail_mode: secure
  Port "ES1"
    Interface "ES1"
      type: internal
  Port "ES1-eth1"
    Interface "ES1-eth1"
  Port "ES1-eth2"
    Interface "ES1-eth2"
```

查阅资料后发现：fail_mode 是故障模式，意思是 SDN 控制器故障时，交换机未连接控制器时的模式（本实验中禁用了控制器）。在 secure 状态下，交换机按照原来流表继续转发，而无法进行 mac 自学习，故交换机 ES1 在收到主机 H1 发来的数据包后无法对其进行相应处理。

参考链接：[OpenvSwitch 系列之五 网桥特性功能配置](#)

解决方法：对每个交换机执行 `sudo ovs-vsctl del-fail-mode xx`

执行完相应指令后，两主机成功 ping 通：

```
mininet> pingall
*** Ping: testing ping reachability
H1 -> H2
H2 -> H1
*** Results: 0% dropped (2/2 received)
```

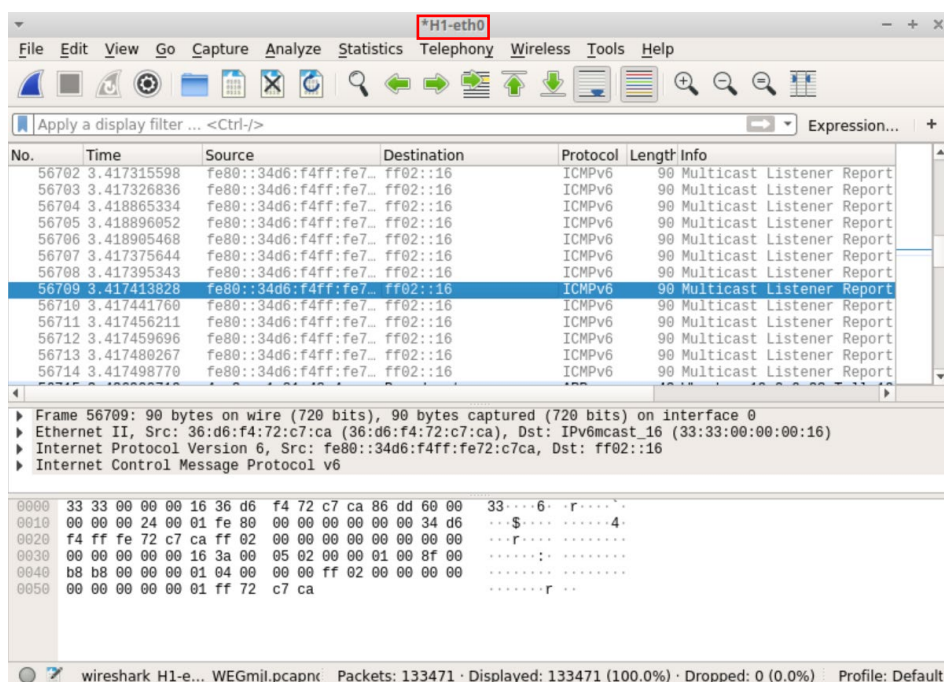
(二) 搭建 Fat Tree 网络拓扑 (k=4)

```
test@sdnexp:~/Desktop$ sudo python FatTree.py
*** Creating network
*** Adding hosts:
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
*** Adding switches:
AS1 AS2 AS3 AS4 AS5 AS6 AS7 AS8 CS1 CS2 CS3 CS4 ES1 ES2 ES3 ES4 ES5 ES6 ES7 ES8
*** Adding links:
(AS1, ES1) (AS1, ES2) (AS2, ES1) (AS2, ES2) (AS3, ES3) (AS3, ES4) (AS4, ES3) (AS4, ES4)
(AS5, ES5) (AS5, ES6) (AS6, ES5) (AS6, ES6) (AS7, ES7) (AS7, ES8) (AS8, ES7) (AS8, ES8)
(CS1, AS1) (CS1, AS3) (CS1, AS5) (CS1, AS7) (CS2, AS1) (CS2, AS3) (CS2, AS5) (CS2, AS7)
(CS3, AS2) (CS3, AS4) (CS3, AS6) (CS3, AS8) (CS4, AS2) (CS4, AS4) (CS4, AS6) (CS4, AS8)
(ES1, H1) (ES1, H2) (ES2, H3) (ES2, H4) (ES3, H5) (ES3, H6) (ES4, H7) (ES4, H8) (ES5, H9)
(ES5, H10) (ES6, H11) (ES6, H12) (ES7, H13) (ES7, H14) (ES8, H15) (ES8, H16)
*** Configuring hosts
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
*** Starting controller
*** Starting 20 switches
AS1 AS2 AS3 AS4 AS5 AS6 AS7 AS8 CS1 CS2 CS3 CS4 ES1 ES2 ES3 ES4 ES5 ES6 ES7 ES8 ...

*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
H1 -> X X X X X X X X X X X X X X
H2 -> X X X X X X X X X X X X X X
H3 -> X X X X X X X X X X X X X X
H4 -> X X X X X X X X X X X X X X
H5 -> X X X X X X X X X X X X X X
H6 -> X X X X X X X X X X X X X X
H7 -> X X X X X X X X X X X X X X
H8 -> X X X X X X X X X X X X X X
H9 -> X X X X X X X X X X X X X X
H10 -> X X X X X X X X X X X X X X
H11 -> X X X X X X X X X X X X X X
H12 -> X X X X X X X X X X X X X X
H13 -> X X X X X X X X X X X X X X
H14 -> X X X X X X X X X X X X X X
H15 -> X X X X X X X X X X X X X X
H16 -> X X X X X X X X X X X X X X
*** Results: 100% dropped (0/240 received)
```

按照(一)中方法删除每个交换机的 fail_mode 后,主机间仍无法 ping 通。

同样,运行命令 H1 ping -c3 H2, 并使用 wireshark 对主机 H1 进行抓包:



The screenshot shows a Wireshark capture on interface H1-eth0. The display filter is set to 'arp'. The packet list shows three ARP requests from 4e:3e:e1:81:48:4e to the broadcast address. The packet details pane shows the selected packet as an ARP request. The packet bytes pane shows the raw data of the ARP request.

No.	Time	Source	Destination	Protocol	Length	Info
56715	3.436900719	4e:3e:e1:81:48:4e	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
74899	4.462707876	4e:3e:e1:81:48:4e	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
89764	5.528908878	4e:3e:e1:81:48:4e	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1

可以发现，主机 H1 发出的 ARP 请求报文被“淹没”在了大量的 ICMPv6 报文中（总共有 13.3 万多个）。

使用 wireshark 对主机 H2 进行抓包：

The screenshot shows a Wireshark capture on interface H2-eth0. The display filter is set to 'arp'. The packet list shows a large number of ICMPv6 Multicast Listener Reports from fe80::2c27:82ff:fe80::ff02::16 to ff02::16. The packet details pane shows the selected packet as an ICMPv6 Multicast Listener Report. The packet bytes pane shows the raw data of the ICMPv6 Multicast Listener Report.

No.	Time	Source	Destination	Protocol	Length	Info
49968	3.823836287	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49969	3.823839986	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49970	3.823878304	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49971	3.823890903	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49972	3.823919092	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49973	3.823939381	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49974	3.823952140	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49975	3.823997156	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49976	3.824029152	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49977	3.824045064	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49978	3.824058451	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49979	3.824074391	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report
49980	3.824083576	fe80::2c27:82ff:fe80::ff02::16	ff02::16	ICMPv6	90	Multicast Listener Report

Frame 49974: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
 Ethernet II, Src: 2e:27:82:89:77:c8 (2e:27:82:89:77:c8), Dst: IPv6mcast_16 (33:33:00:00:00:16)
 Internet Protocol Version 6, Src: fe80::2c27:82ff:fe80::ff02::16, Dst: ff02::16
 Internet Control Message Protocol v6

0000 33 33 00 00 00 16 2e 27 82 89 77 c8 86 dd 60 00 33w.....
 0010 00 00 00 24 00 01 fe 80 00 00 00 00 00 2c 27\$.....
 0020 82 ff fe 89 77 c8 ff 02 00 00 00 00 00 00 00w.....
 0030 00 00 00 00 16 3a 00 05 02 00 00 01 00 8f 00:.....
 0040 49 97 00 00 00 01 04 00 00 00 ff 02 00 00 00 00 I.....
 0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

可以发现，主机 H2 也被大量的 ICMPv6 报文“淹没”，甚至连 ARP 应答报文都没有发出。

对比 k=4 和 k=2 两种网络拓扑，最大的不同应该是 k=4 的拓扑中出现了环路。这时，用来探测网络连接、更新路由等信息的数据包就可能形成广播风暴，让网络处于极度拥塞的状态。

解决方法：为每一个交换机开启生成树协议，避免数据包在一个环上来回转

发 (sudo ovs-vsctl set bridge xx stp_enable=true)。

执行完相应指令后，等待一段时间，所有主机都能成功 ping 通：

```
mininet> pingall
*** Ping: testing ping reachability
H1 -> H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H2 -> H1 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H3 -> H1 H2 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H4 -> H1 H2 H3 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H5 -> H1 H2 H3 H4 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H6 -> H1 H2 H3 H4 H5 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16
H7 -> H1 H2 H3 H4 H5 H6 H8 H9 H10 H11 H12 H13 H14 H15 H16
H8 -> H1 H2 H3 H4 H5 H6 H7 H9 H10 H11 H12 H13 H14 H15 H16
H9 -> H1 H2 H3 H4 H5 H6 H7 H8 H10 H11 H12 H13 H14 H15 H16
H10 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H11 H12 H13 H14 H15 H16
H11 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H12 H13 H14 H15 H16
H12 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H13 H14 H15 H16
H13 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H14 H15 H16
H14 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H15 H16
H15 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H16
H16 -> H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15
*** Results: 0% dropped (240/240 received)
```

(三) 分析数据包的路径

以 H1 ping -c3 H8 为例：

由于主机 H1 与交换机 ES1 直接相连，故先查询 ES1 的 MAC 表：

```
test@sdnexp:~$ sudo ovs-appctl fdb/show ES1
port  VLAN  MAC                               Age
  4      0  00:00:00:00:00:02                249
  2      0  00:00:00:00:00:0f                249
  2      0  00:00:00:00:00:0d                249
  3      0  00:00:00:00:00:01                233
  2      0  00:00:00:00:00:0e                233
  2      0  00:00:00:00:00:0b                 69
  2      0  00:00:00:00:00:04                 53
  2      0  00:00:00:00:00:08                 53
  2      0  00:00:00:00:00:09                 53
```

发现出口为 port 2，即转发到交换机 AS2，其 MAC 表为：

```
test@sdnexp:~$ sudo ovs-appctl fdb/show AS2
port  VLAN  MAC                               Age
  3      0  00:00:00:00:00:01                265
  1      0  00:00:00:00:00:0e                265
  1      0  00:00:00:00:00:0b                101
  4      0  00:00:00:00:00:04                 85
  1      0  00:00:00:00:00:08                 85
  1      0  00:00:00:00:00:09                 85
```

发现出口为 port 1，即转发到交换机 CS3，其 MAC 表为：

```
test@sdnexp:~$ sudo ovs-appctl fdb/show CS3
port VLAN MAC Age
 4 0 00:00:00:00:00:0e 294
 3 0 00:00:00:00:00:0b 130
 2 0 00:00:00:00:00:08 114
 3 0 00:00:00:00:00:09 114
```

发现出口为 port 2，即转发到交换机 AS4，其 MAC 表为：

```
test@sdnexp:~$ sudo ovs-appctl fdb/show AS4
port VLAN MAC Age
 1 0 00:00:00:00:00:0b 173
 4 0 00:00:00:00:00:08 157
 1 0 00:00:00:00:00:09 157
```

发现出口为 port 4，即转发到交换机 ES4，其 MAC 表为：

```
test@sdnexp:~$ sudo ovs-appctl fdb/show ES4
port VLAN MAC Age
 2 0 00:00:00:00:00:0b 195
 4 0 00:00:00:00:00:08 179
 2 0 00:00:00:00:00:09 179
```

发现出口为 port 4，即转发到主机 H8。

通过 wireshark 抓包也可以进行验证：

***ES1-eth3**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x315c,
2	0.001103104	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x315c,
3	0.550737009	da:9f:13:24:bf:b1	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8,
4	1.003739766	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x315c,
5	1.003843289	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x315c,
6	2.015843438	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x315c,
7	2.015945465	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x315c,

***AS2-eth3**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	76:2d:4a:97:fe:2d	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8,
2	0.696408753	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3162,
3	0.697058145	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3162,
4	1.698018987	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3162,
5	1.698041793	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3162,
6	2.002888548	76:2d:4a:97:fe:2d	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8,
7	2.713119416	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3162,
8	2.713416366	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3162,

***CS3-eth1**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316a,
2	0.001327979	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316a,
3	0.651797452	e2:e6:5a:71:90:c0	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8,
4	1.003335928	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316a,
5	1.003414253	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316a,
6	2.009551467	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316a,
7	2.009571482	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316a,

*AS4-eth1						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316f,
2	0.000195109	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316f,
3	1.000147856	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316f,
4	1.000164808	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316f,
5	1.148755902	ea:da:94:17:38:92	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8
6	2.031700971	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x316f,
7	2.031776101	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x316f,

*ES4-eth2						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3175,
2	0.000119081	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3175,
3	1.001006579	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3175,
4	1.001021087	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3175,
5	1.310836970	2e:2c:64:a9:32:90	Spanning-tree-(for-...	STP	52	Conf. Root = 32768/0/1a:4e:54:8
6	2.033339944	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request id=0x3175,
7	2.033393047	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3175,

故其转发路径为 H1→ES1→AS2→CS3→AS4→ES4→H8。

(四) 源代码

```

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel
import os

class FatTree(Topo):

    def __init__(self, k, **opts):

        super(FatTree, self).__init__(**opts)

        CoreSwitches = [self.addSwitch('CS'+str(i+1)) for i in
range((k/2)**2)]
        AggrSwitches = [self.addSwitch('AS'+str(i+1)) for i in
range((k/2)*k)]
        EdgeSwitches = [self.addSwitch('ES'+str(i+1)) for i in
range((k/2)*k)]
        Host = [self.addHost('H'+str(i+1)) for i in range((k**3)/4)]

        # Create Link between Core and Aggregation
        for i in range((k/2)**2):
            for j in range(k):
                self.addLink(CoreSwitches[i],
AggrSwitches[i//(k/2)+j*(k/2)])

```

```

        # Create Link between Aggregation and Edge
        for i in range((k/2)*k):
            for j in range(k/2):
                self.addLink(AggrSwitches[i],
EdgeSwitches[i//(k/2)*(k/2)+j])

        # Create Link between Edge and Host
        for i in range((k/2)*k):
            for j in range(k/2):
                self.addLink(EdgeSwitches[i], Host[i*(k/2)+j])

def run(k=2):
    topo = FatTree(k)
    net = Mininet(topo=topo, controller=None, autoSetMacs=True)

    net.start()

    for i in range((k/2)**2):
        os.system('sudo ovs-vsctl set bridge CS' + str(i+1) + '
stp_enable=true')
        os.system('sudo ovs-vsctl del-fail-mode CS' + str(i+1))
        for i in range((k/2)*k):
            os.system('sudo ovs-vsctl set bridge AS' + str(i+1) + '
stp_enable=true')
            os.system('sudo ovs-vsctl del-fail-mode AS' + str(i+1))
            for i in range((k/2)*k):
                os.system('sudo ovs-vsctl set bridge ES' + str(i+1) + '
stp_enable=true')
                os.system('sudo ovs-vsctl del-fail-mode ES' + str(i+1))

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    run(4)

```