Lab 2: Learn Mininet

计算机网络 CS339

李子龙 518070910095

2021 年 9 月 30 日

目录

1 第一题 1
1.1 题目 1
1.2 源代码 1
1.3 测试结果 3
2 第二题 3
2.1 题目 3
2.2 源代码变更 3
2.3 测试结果 4
3 第三题 5
3.1 题目 5
3.2 源代码变更（一） 5
3.3 测试结果（一） 5
3.4 源代码变更（二） 6
3.5 测试结果（二） 6

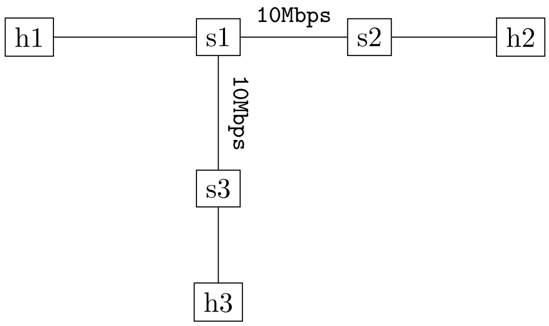
1 第一题

1.1 题目

Simulate the following topology in Mininet. Set the link bandwidth for (s1,s2) and (s1,s3) as 10Mbps. Use Iperf to test the TCP throughput between every host pair.

第一题限制了交换机之间的带宽为 10 Mbps。

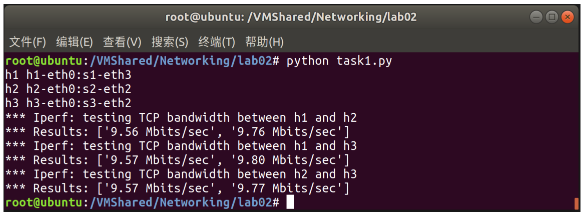
1.2 源代码



Listing 1: task1.py

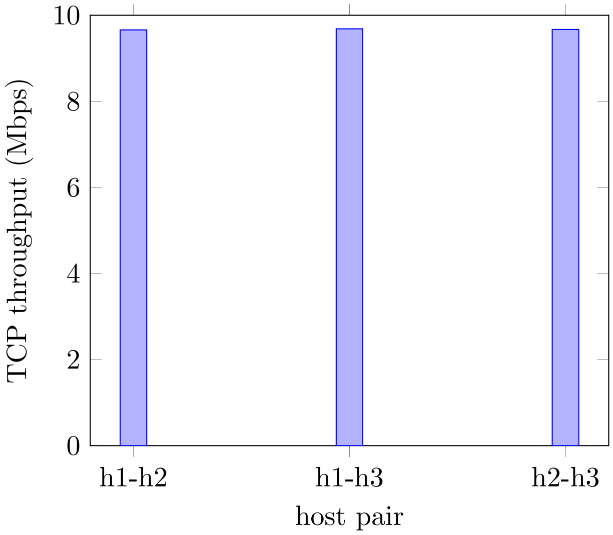
```
# 1. Simulate the following topology in Mininet. Set the link bandwidth for (s1,s2) and (s1,s3) as 10Mbps. Use Iperf to test the TCP throughput between every host pair.
#
# h1--s1--s2--h2
#      |
#     s3
#      |
#     h3

from mininet.link import TCLink
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import lg, info
from mininet.util import dumpNodeConnections

class NetworkTopo(Topo):
    "Topology of task 1."

    def build(self):
        # Create switchs and hosts
        h1, h2, h3 = [self.addHost(h) for h in (’h1’,’h2’,’h3’)]
        s1, s2, s3 = [self.addSwitch(s) for s in (’s1’,’s2’,’s3’)]

        # Wire up switches with constriants
        self.addLink(s1, s2, bw=10)
        self.addLink(s1, s3, bw=10)

        self.addLink(h1, s1)
        self.addLink(h3, s3)
        self.addLink(h2, s2)

def perfTest():
    "Use Iperf to test the TCP throughput between every host pair."
    topo = NetworkTopo()
    # The constructor of TCLink is required
    # to get the constraints from topo.
    net = Mininet(topo=topo,link=TCLink,autoStaticArp=True)
    net.start()
    dumpNodeConnections(net.hosts)
    h1, h2, h3 = net.getNodeByName(’h1’,’h2’,’h3’)
    net.iperf((h1,h2))
    net.iperf((h1,h3))
    net.iperf((h2,h3))
    net.stop()

if __name__ == "__main__":
    # lg.setLogLevel( 'info' )
    perfTest()
```

1.3 测试结果


root@ubuntu: /VMShared/Networking/lab02
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab02# python task1.py
h1 h1-eth0:s1-eth3
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.56 Mbits/sec', '9.76 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.57 Mbits/sec', '9.80 Mbits/sec']
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['9.57 Mbits/sec', '9.77 Mbits/sec']
root@ubuntu:/VMShared/Networking/lab02#
```

Pair	Throughput(Mbps)
h1-h2	5.94
h1-h3	7.58
h2-h3	2.46



所有的对的吞吐量都被降低到了 10 Mbps 以下。

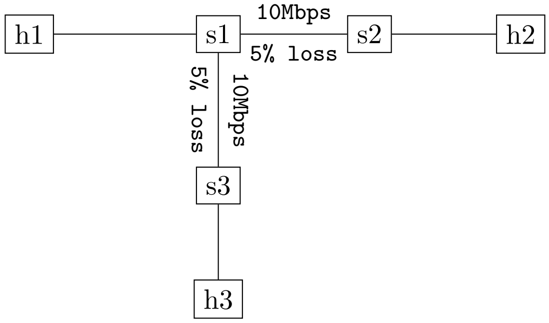
2 第二题

2.1 题目

Now let us set the packet loss rate of the link (s1,s2) and (s1,s3) as 5%. Use Iperf to test the TCP throughput again.

交换机之间的带宽限制为 10 Mbps，丢包率为 5%。

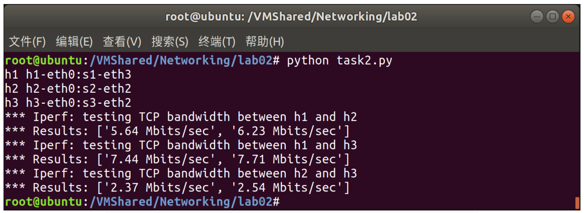
2.2 源代码变更



Listing 2: task2.py

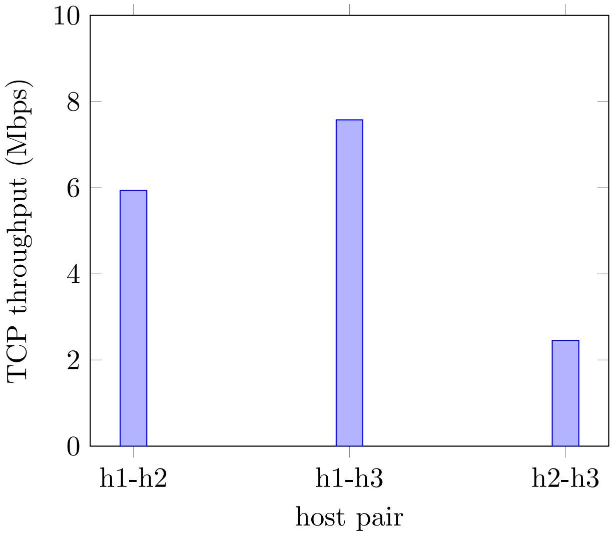
```
        # Wire up switches with constriants
        self.addLink(s1, s2, bw=10, loss=5)
        self.addLink(s1, s3, bw=10, loss=5)
```

2.3 测试结果


root@ubuntu: /VMShared/Networking/lab02
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab02# python task2.py
h1 h1-eth0:s1-eth3
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['5.64 Mbits/sec', '6.23 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['7.44 Mbits/sec', '7.71 Mbits/sec']
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['2.37 Mbits/sec', '2.54 Mbits/sec']
root@ubuntu:/VMShared/Networking/lab02#
```

Pair	Throughput(Mbps)
h1-h2	9.66
h1-h3	9.69
h2-h3	9.67

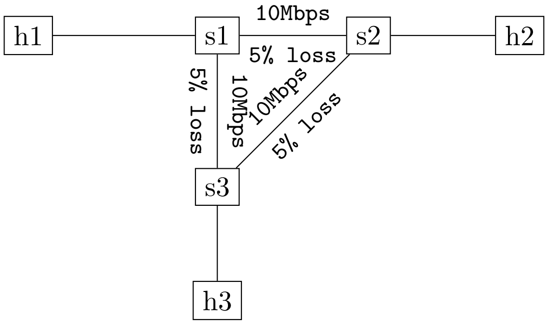


所有对的吞吐量大幅下降，其中 h2 和 h3 之间的吞吐量最低，因为会经过两个丢包链路，所以会丢失更多的包。

3 第三题

3.1 题目

Let us add another link between s2 and s3. Try pinging h2 from h1. What would happen? How would you solve the problem? (Hint: Use ovs-ofctl command to add flow rules.)



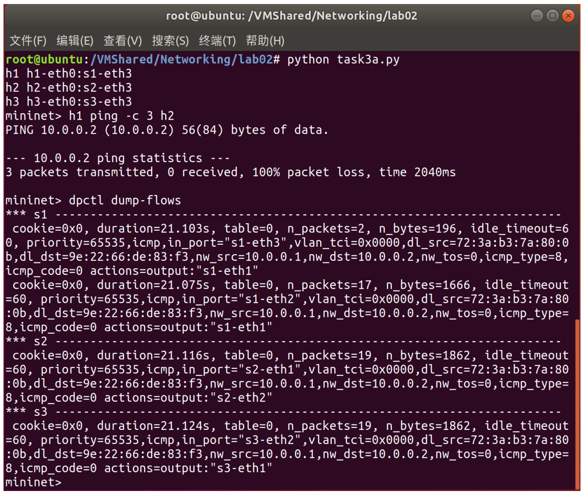
3.2 源代码变更（一）

Listing 3: task3a.py

```
        # New link between s2, s3
        self.addLink(s2, s3, bw=10, loss=5)

        self.addLink(h1, s1)
        self.addLink(h3, s3)
        self.addLink(h2, s2)

def pingTest():
    topo = NetworkTopo()
    net = Mininet(topo=topo,link=TCLink,autoStaticArp=True)
    net.start()
    dumpNodeConnections(net.hosts)
    # h1, h2 = net.getNodeByName('h1','h2')
    # net.ping([h2, h1])
    CLI(net)            # debug interface
    net.stop()

if __name__ == "__main__":
    # lg.setLogLevel( 'info' )
    pingTest()
```

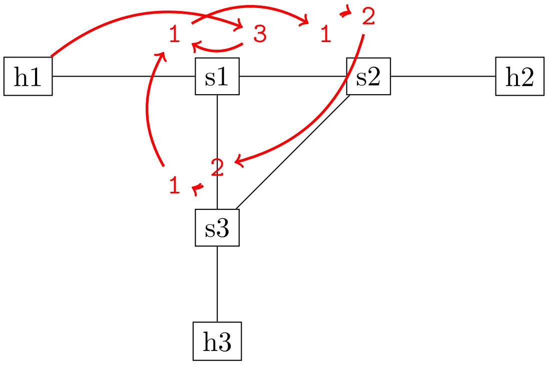
3.3 测试结果（一）

测试显示 h1 到 h2 的包全部丢失。图中还显示了流表信息。右图展示了一个从 h1 发出的包会导致端口间转发循环的一种情况，在这种默认的配置下，会导致找不到到达 h2 的通路。



root@ubuntu: /VMShared/Networking/lab02
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab02# python task3a.py
h1 h1-eth0:s1-eth3
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth3
mininet> h1 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2040ms

mininet> dpctl dump-flows
*** s1 -----------------------------------------------------------------------
 cookie=0x0, duration=21.103s, table=0, n_packets=2, n_bytes=196, idle_timeout=6
0, priority=65535,icmp,in_port="s1-eth3",vlan_tci=0x0000,dl_src=72:3a:b3:7a:80:0
b,dl_dst=9e:22:66:de:83:f3,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,
icmp_code=0 actions=output:"s1-eth1"
 cookie=0x0, duration=21.075s, table=0, n_packets=17, n_bytes=1666, idle_timeout
=60, priority=65535,icmp,in_port="s1-eth2",vlan_tci=0x0000,dl_src=72:3a:b3:7a:80
:0b,dl_dst=9e:22:66:de:83:f3,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=
8,icmp_code=0 actions=output:"s1-eth1"
*** s2 -----------------------------------------------------------------------
 cookie=0x0, duration=21.116s, table=0, n_packets=19, n_bytes=1862, idle_timeout
=60, priority=65535,icmp,in_port="s2-eth1",vlan_tci=0x0000,dl_src=72:3a:b3:7a:80
:0b,dl_dst=9e:22:66:de:83:f3,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=
8,icmp_code=0 actions=output:"s2-eth2"
*** s3 -----------------------------------------------------------------------
 cookie=0x0, duration=21.124s, table=0, n_packets=19, n_bytes=1862, idle_timeout
=60, priority=65535,icmp,in_port="s3-eth2",vlan_tci=0x0000,dl_src=72:3a:b3:7a:80
:0b,dl_dst=9e:22:66:de:83:f3,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=
8,icmp_code=0 actions=output:"s3-eth1"
mininet>
```



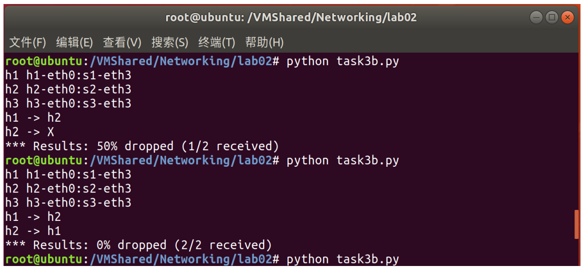
3.4 源代码变更（二）

Listing 4: task3b.py

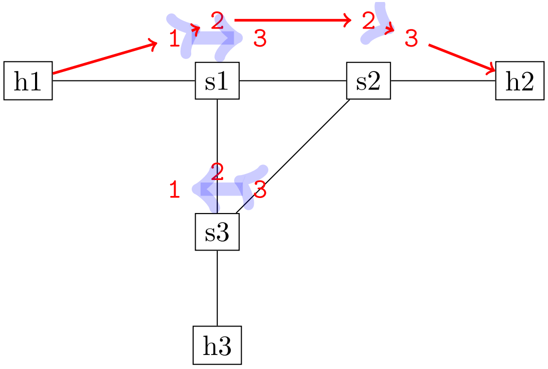
```
        # New link between s2, s3
        self.addLink(s2, s3, bw=10, loss=5)

        self.addLink(h1, s1)
        self.addLink(h3, s3)
        self.addLink(h2, s2)

def pingTest():
    topo = NetworkTopo()
    net = Mininet(topo=topo,link=TCLink,autoStaticArp=True)
    net.start()
    dumpNodeConnections(net.hosts)
    # add flow rules
    run('sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2,3')
    run('sudo ovs-ofctl add-flow s2 in_port=2,actions=output:3')
    run('sudo ovs-ofctl add-flow s3 in_port=3,actions=output:1,2')
    h1, h2 = net.getNodeByName('h1','h2')
    net.ping([h1,h2])
    net.stop()

if __name__ == "__main__":
    # lg.setLogLevel( 'info' )
    pingTest()
```

3.5 测试结果（二）

通过手动添加流表规则，


root@ubuntu: /VMShared/Networking/lab02
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/VMShared/Networking/lab02# python task3b.py
h1 h1-eth0:s1-eth3
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth3
h1 -> h2
h2 -> X
*** Results: 50% dropped (1/2 received)
root@ubuntu:/VMShared/Networking/lab02# python task3b.py
h1 h1-eth0:s1-eth3
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth3
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
root@ubuntu:/VMShared/Networking/lab02# python task3b.py
```



s1p1→s1p2,s1p3

s2p2→s2p3

s3p3→s3p1,s3p2

就能够实现 h1 到 h2 的通信。