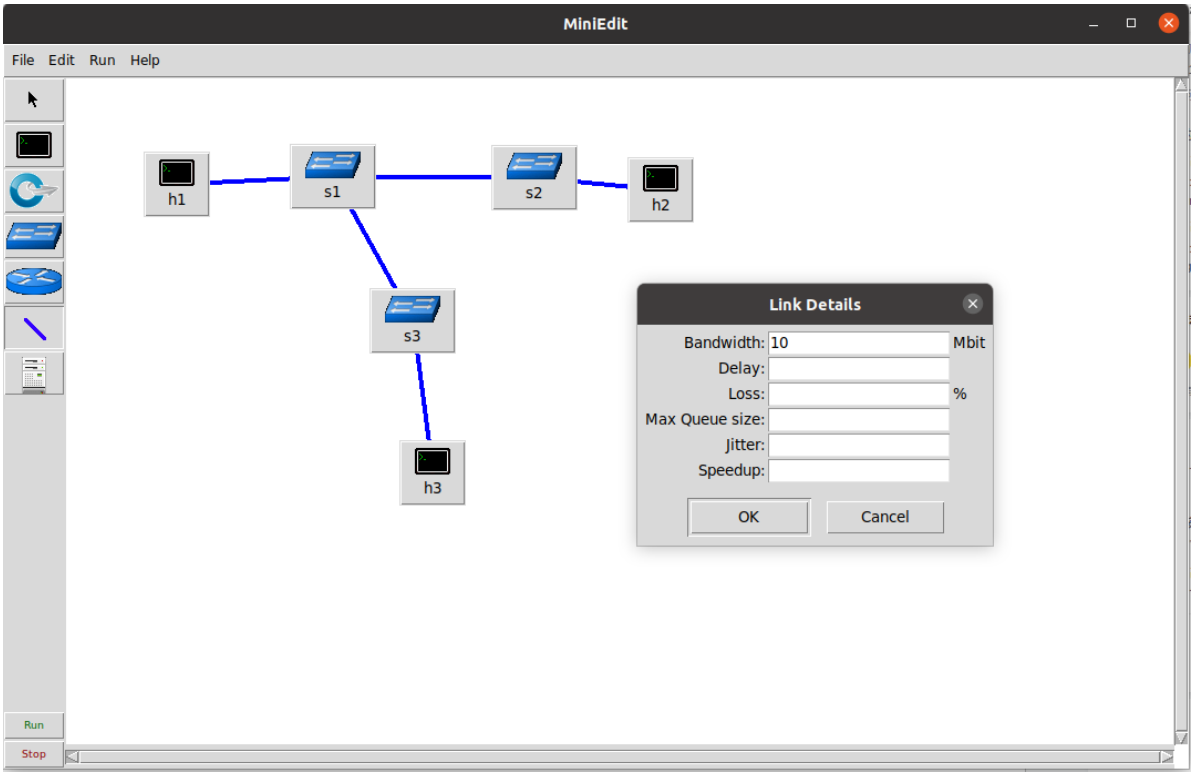
Lab2: Play with Mininet

Xin Xu 519021910726

the TCP Throughput

At first, we use miniedit to draw the topology as follows and configure the bandwidth between switches s1 and s2 addition to s1 and s3:



After saving the topology to a .py file, we use iperf command to test throughput between everypair in CLI:

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.30 Mbits/sec', '9.84 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.00 Mbits/sec', '9.62 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['8.98 Mbits/sec', '9.43 Mbits/sec']
```

the TCP Throughput with Packet loss

Add the packet loss constrain to .py file as follows:

```
info( '*** Add links\n')
net.addLink(h1, s1)
net.addLink(s2, h2)
net.addLink(s3, h3)
s1s2 = {'bw':10,'loss':5}
net.addLink(s1, s2, cls=TCLink , **s1s2)
s1s3 = {'bw':10,'loss':5}
net.addLink(s1, s3, cls=TCLink , **s1s3)
```

Run this .py file and test the throughput between switches s1 and s2 addition to s1 and s3 again:

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['6.14 Mbits/sec', '6.47 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['5.94 Mbits/sec', '6.30 Mbits/sec']
mininet> iperg h2 h3
*** Unknown command: iperg h2 h3
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['1.38 Mbits/sec', '1.45 Mbits/sec']
```

Add Link

Add the link between switches s2 and s3 in .py file as follows:

```
info( '*** Add links\n')
net.addLink(h1, s1)
net.addLink(s2, h2)
net.addLink(s3, h3)
s1s2 = {'bw':10,'loss':5}
net.addLink(s1, s2, cls=TCLink , **s1s2)
s1s3 = {'bw':10,'loss':5}
net.addLink(s1, s3, cls=TCLink , **s1s3)
net.addLink(s2, s3)
```

And try pinging h2 from h1:

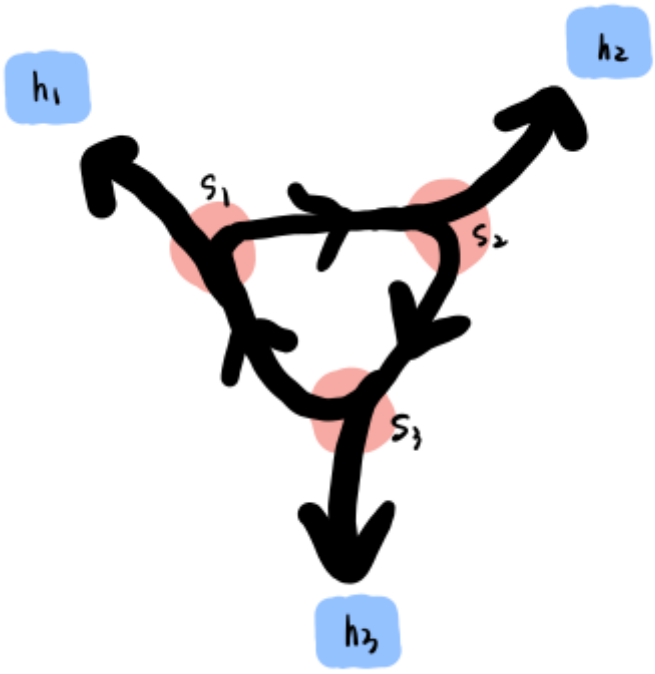
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 6152ms
pipe 4
```

We can see that there is no connection between h1 and h2. Try pingall command:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

It's false to ping all the pairs. We will try to use ovs-ofctl command to re-specify data packets flow within a switch.

At first, I try to specify data flows as follows: there is a cycle flow inside the three switches: s1, s2 and s3, transmitting the data along the cycle without a break until data packets reach the destination:



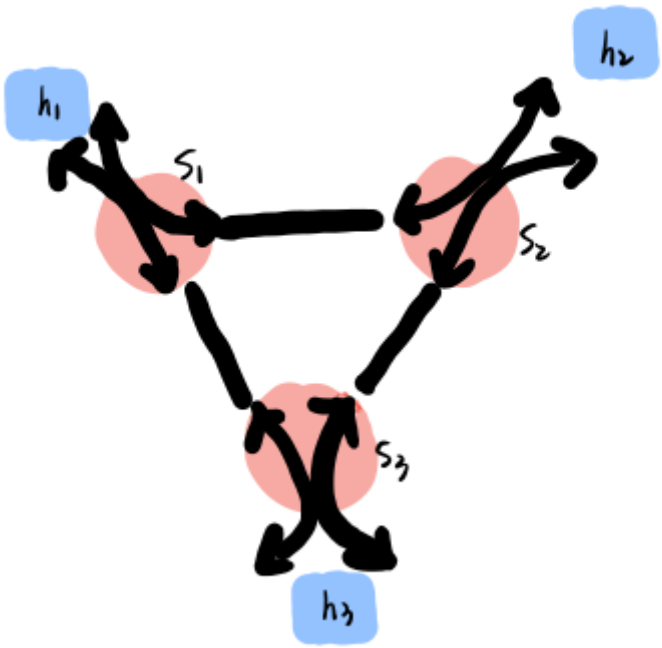
The commands to add these data flows are as follows:

```
mininet> sh ovs-ofctl add-flow s1 in_port=1,actions=output:2
mininet> sh ovs-ofctl add-flow s2 in_port=1,actions=output:3
mininet> sh ovs-ofctl add-flow s3 in_port=1,actions=output:2
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> sh ovs-ofctl add-flow s2 in_port=2,actions=flood
mininet> sh ovs-ofctl add-flow s3 in_port=3,actions=flood
mininet> sh ovs-ofctl add-flow s1 in_port=3,actions=flood
```

The output of this module is:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.606 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.646 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.648 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.703 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.705 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.707 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.764 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.766 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.768 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.806 ms (DUP!)
```

There is a question of duplicate data packets (because the data packets will turn around within the cycle even though hosts already have received them), so I figure another data flow module: the host port can transmit data packets forward all other ports inside a switch and every data packet transmitted into a switch will be submitted to the host port.



The commands to specify this module are as follows:

```
mininet> sh ovs-ofctl add-flow s1 in_port=2,actions=output:1
mininet> sh ovs-ofctl add-flow s1 in_port=3,actions=output:1
mininet> sh ovs-ofctl add-flow s1 in_port=1,actions=all
mininet> sh ovs-ofctl add-flow s2 in_port=2,actions=output:1
mininet> sh ovs-ofctl add-flow s2 in_port=3,actions=output:1
mininet> sh ovs-ofctl add-flow s2 in_port=1,actions=all
mininet> sh ovs-ofctl add-flow s3 in_port=2,actions=output:1
mininet> sh ovs-ofctl add-flow s3 in_port=3,actions=output:1
mininet> sh ovs-ofctl add-flow s3 in_port=1,actions=all
```

The output of this module is:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.515 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.125 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.120/0.253/0.515/0.185 ms
```

We can see that there are no duplicates along the data flow.

Add these modifications into .py file.

Reflection

Questions

- I don't understand why there is no connection in the original topology of the last question. In my opinion, there are actual links between switches and hosts. Additionally, I use ovs-ofctl dump-flows command to verify there actually are data flows in a switch. So, I'm puzzled why there is no connection in the original topology.

Tips

- It's helpful to use sudo mn -c command to clear redundant topology in order to avoid strange mistakes.
- There may be a failure the first time to run pingall command even though there is actual a connection within this pairs. One Solution is to pingall again. The reason is that the first time to ping a pair, the routing table may don't have the message of the way to transmit data (for example, the switch don't know which port to submit the input data), so it may return a failure response at the first time, get the IPs of the pair and register this pair's transmitting message into the routing table. So, the next time the switch can find the information needed in its routing table and the transmission can succeed.