# CS339 "Learn Mininet" Report

Student: Zhou Haoquan

ID: 519370910059

## Homework 1

To create the topological structure shown in the problem, we first code a `python` file as shown below:

```
#!/usr/bin/env python

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

def myNetwork():

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

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

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='0000000000000001')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='0000000000000001')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='0000000000000001')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)

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

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

    info( '*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])

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

    CLI(net)
    net.stop()

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

Note that we set the IP address of `h1` to be `10.0.0.1`, the IP address of `h2` to be `10.0.0.2` and the IP address of `h3` to be `10.0.0.3`.

Then as the problem suggests, we test the TCP throughput between `h1` and `h2` using the command

```
iperf h1 h2
iperf h1 h3
```

The corresponding result is shown in the following picture:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.57 Mbits/sec', '9.99 Mbits/sec']
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.57 Mbits/sec', '10.0 Mbits/sec']
```

As we seen from the result, the throughput is limited to the bandwidth of the line between `h1`, `h2` and `h3`, which is roughly `9.57~10 Mbits/sec`. This is close to the theoretical value.

## Homework 2

Then we add the packet loss rate to be `5%` to the line between `h1` and `h2` as well as the line between `h1` and `h3`. The we repeat the same step above and get the result as shown in the following picture:

```
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['6.63 Mbits/sec', '6.79 Mbits/sec']
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['7.01 Mbits/sec', '7.45 Mbits/sec']
```

We can see after adding loss to the line, the bandwidth is significantly reduced to about `6.7~7.3 Mbits/sec`. We can say that since we adopt the `TCP` protocol between hosts, then the lost package should be re-sent by the sender, which may result in increasing the time cost. So it is not surprising that the bandwidth will decrease.

## Homework 3

Now we re-configure the topology of the network and add a connection between `h2` as `h3`, which is required by the problem. The code is as follows:

```
#!/usr/bin/env python

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

def myNetwork():

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

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

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='0000000000000001')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='0000000000000001')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='0000000000000001')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(h1, s1)
    net.addLink(h2, s2)
    net.addLink(h3, s3)
    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(s3, s2)

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

```
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])

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

    CLI(net)
    net.stop()

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

Now, we try to test the ping between `h1` and `h2` using the instruction:

```
ping h1 10.0.0.2
```

The result shows that the connection is lost:

```
mininet> h1 ping 10.0.0.2
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
```

This is because the connection between three switches form a loop and the information is stuck in the loop, being transported forever.

To solve this problem, we add the self-defined flow rule to the switches using the instructions:

```
sudo ovs-ofctl addflow s3 "in_port=1 actions=output:2"
sudo ovs-ofctl addflow s3 "in_port=2 actions=output:1"
sudo ovs-ofctl addflow s1 "in_port=1 actions=output:2"
sudo ovs-ofctl addflow s1 "in_port=2 actions=output:1"
```

These instruction make one of the connection in the loop disabled, which should be enough to solve the problem. Then we test the ping from `h1` to `h2`, which results in:

```
mininet> h1 ping 10.0.0.2
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=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.073 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 7166ms
rtt min/avg/max/mdev = 0.054/0.078/0.120/0.019 ms
```

we see that connection between the hosts are recovered.