

SDN Lab 3

Helpful resource : <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>

In this exercise, you will be learning how to build custom topologies using Mininet Python API and how certain parameters like bandwidth, delay, loss and queue size can be set individually for different links in the topology. You'll also learn how to do performance testing of these custom topologies using ping and iperf.

After the overview, you will be asked to create and submit your own custom topology based on the most common **3-tier Datacenter architecture** i.e., core, aggregation and edge. More details on creating and submitting the code will be provided later on in the instructions. So, make sure that you follow each step carefully.

Overview

The network you'll use in this exercise includes hosts and switches connected in a linear topology, as shown in the figure below.

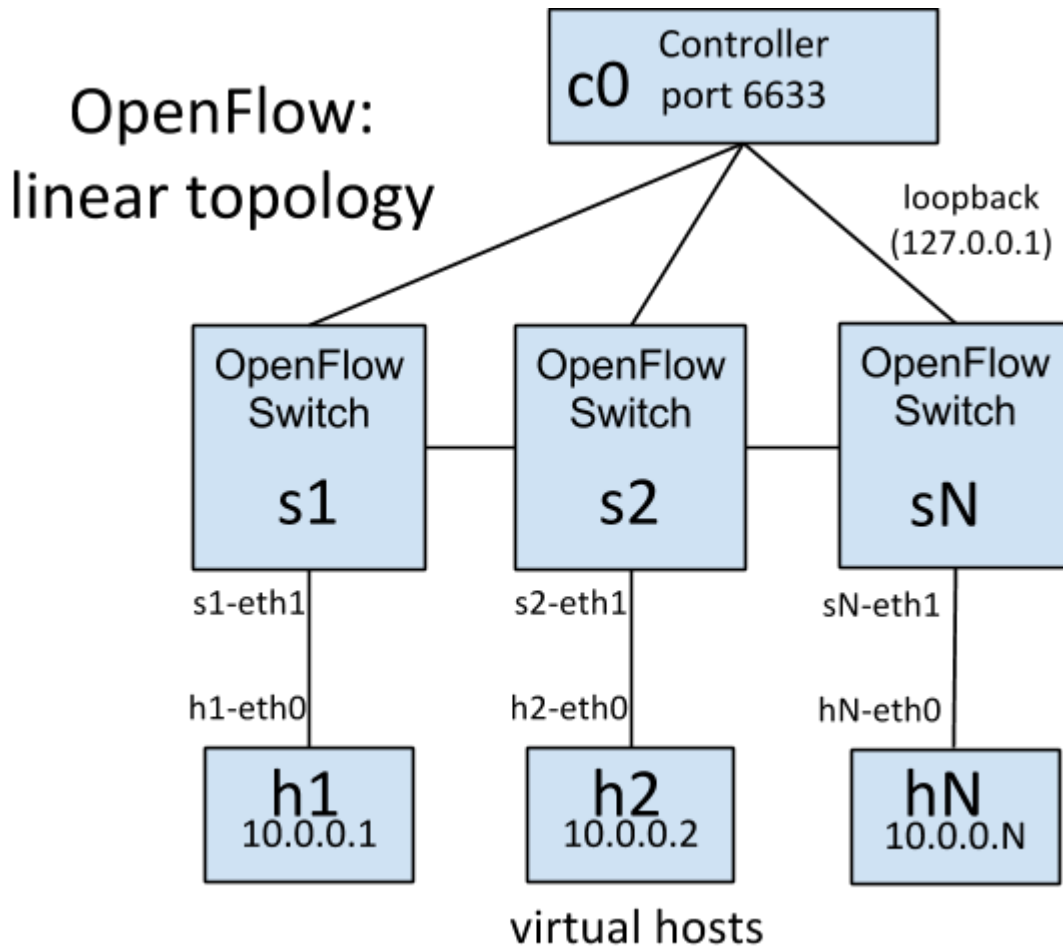


Figure 1: hosts and switches connected in a linear topology

Creating Topology

Mininet supports *parametrized topologies*. With a few lines of Python code, you can create a flexible topology which can be configured based on the parameters you pass into it, and reused for multiple experiments.

For example, here is a simple network topology (based on Figure 1) which consists of a specified number of hosts (h_1 through h_N) connected to their individual switches (s_1 through s_N):

Linear Topology (without Performance Settings)

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

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import irange, dumpNodeConnections
from mininet.log import setLogLevel

class LinearTopo(Topo):
    "Linear topology of k switches, with one host per switch."

    def __init__(self, k=2, **opts):
        """Init.
        k: number of switches (and hosts)
        hconf: host configuration options
        lconf: link configuration options"""

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

        self.k = k

        lastSwitch = None
        for i in irange(1, k):
            host = self.addHost('h%s' % i)
            switch = self.addSwitch('s%s' % i)
            self.addLink( host, switch)
            if lastSwitch:
                self.addLink( lastSwitch, switch)
            lastSwitch = switch
```

```

        self.addLink( switch, lastSwitch)
        lastSwitch = switch

def simpleTest():
    "Create and test a simple network"
    topo = LinearTopo(k=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')

    simpleTest()

```

Figure 1. LinearTopo.py

The important classes, methods, functions and variables in the above code include:

- Topo: the base class for Mininet topologies
- addSwitch(): adds a switch to a topology and returns the switch name
- addHost(): adds a host to a topology and returns the host name
- addLink(): adds a bidirectional link to a topology (and returns a link key, but this is not important). Links in Mininet are bidirectional unless noted otherwise.
- Mininet: main class to create and manage a network
- start(): starts your network
- pingAll(): tests connectivity by trying to have all nodes ping each other
- stop(): stops your network
- net.hosts: all the hosts in a network
- dumpNodeConnections(): dumps connections to/from a set of nodes.
- setLogLevel('info' | 'debug' | 'output'): set Mininet's default output level; 'info' is recommended as it provides useful information.

Additional example code may be found in [mininet/examples](http://mininet.org/examples).

Setting Performance Parameters

In addition to basic behavioral networking, Mininet provides performance limiting and isolation features, through the `CPULimitedHost` and `TCLink` classes.

There are multiple ways that these classes may be used, but one simple way is to specify them as the default host and link classes/constructors to `Mininet()`, and then to specify the appropriate parameters in the topology.

Linear Topology (with Performance Settings)

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

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import irange, dumpNodeConnections
from mininet.log import setLogLevel

class LinearTopo(Topo):
    "Linear topology of k switches, with one host per switch."

    def __init__(self, k=2, **opts):
        """Init.
        k: number of switches (and hosts)
        hconf: host configuration options
        lconf: link configuration options"""

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

        self.k = k

        lastSwitch = None
        for i in irange(1, k):
            host = self.addHost('h%s' % i, cpu=.5/k)
            switch = self.addSwitch('s%s' % i)
            # 10 Mbps, 5ms delay, 1% loss, 1000 packet queue
```

```

        self.addLink( host, switch, bw=10, delay='5ms', loss=1,
max_queue_size=1000, use_htb=True)
        if lastSwitch:
            self.addLink(switch, lastSwitch, bw=10, delay='5ms', loss=1,
max_queue_size=1000, use_htb=True)
            lastSwitch = switch

def perfTest():
    "Create network and run simple performance test"
    topo = LinearTopo(k=4)
    net = Mininet(topo=topo,
                  host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))
    net.stop()

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

    perfTest()

```

Some important methods and parameters:

`self.addHost(name, cpu=f)`: This allows you to specify a fraction of overall system CPU resources which will be allocated to the virtual host.

`self.addLink(node1, node2, bw=10, delay='5ms', max_queue_size=1000, loss=1, use_htb=True)`: adds a bidirectional link with bandwidth, delay and loss characteristics, with a maximum queue size of 1000 packets using the Hierarchical Token Bucket rate limiter and netem delay/loss emulator. The parameter `bw` is expressed as a number in Mb/s; `delay` is expressed as a string with units in place (e.g. '5ms', '100us', '1s'); `loss` is expressed as a percentage (between 0 and 100); and `max_queue_size` is expressed in packets.

You may find it useful to create a Python dictionary to make it easy to pass the same parameters into multiple method calls, for example:

```
linkopts = dict(bw=10, delay='5ms', loss=1, max_queue_size=1000,
use_htb=True)
```

```
    linkopts = dict(bw=10, delay='5ms', loss=1, max_queue_size=1000,
use_htb=True)
'''
    alternately: linkopts = {'bw':10, 'delay':'5ms', 'loss':1,
'max_queue_size':1000, 'use_htb':True}
'''

    self.addLink(node1, node2, **linkopts)
```

Setting Performance Parameters

In addition to basic behavioral networking, Mininet provides performance limiting and isolation features, through the `CPULimitedHost` and `TCLink` classes.

There are multiple ways that these classes may be used, but one simple way is to specify them as the default host and link classes/constructors to `Mininet()`, and then to specify the appropriate parameters in the topology. (You could also specify custom classes in the topology itself, or create custom node and link constructors and/or subclasses.)

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        for h in range(n):
            # Each host gets 50%/n of system CPU
            host = self.addHost('h%s' % (h + 1),
                                cpu=.5/n)
            # 10 Mbps, 5ms delay, 10% loss, 1000 packet queue
            self.addLink(host, switch,
                          bw=10, delay='5ms', loss=10, max_queue_size=1000, use_htb=True)

def perfTest():
    "Create network and run simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo=topo,
                  host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
```

```

dumpNodeConnections(net.hosts)
print "Testing network connectivity"
net.pingAll()
print "Testing bandwidth between h1 and h4"
h1, h4 = net.get('h1', 'h4')
net.iperf((h1, h4))
net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    perfTest()

```

Important methods and parameters:

`self.addHost(name, cpu=f):`

This allows you to specify a fraction of overall system CPU resources which will be allocated to the virtual host.

`self.addLink(node1, node2, bw=10, delay='5ms', max_queue_size=1000, loss=10, use_htb=True):` adds a bidirectional link with bandwidth, delay and loss characteristics, with a maximum queue size of 1000 packets using the Hierarchical Token Bucket rate limiter and netem delay/loss emulator. The parameter `bw` is expressed as a number in Mbit; `delay` is expressed as a string with units in place (e.g. '5ms', '100us', '1s'); `loss` is expressed as a percentage (between 0 and 100); and `max_queue_size` is expressed in packets.

Running in Mininet

To run the custom topology you have created above, follow the instructions below:

- Create a `LinearTopo.py` script on your Mininet VM and copy the contents of Linear Topology (without Performance Settings), listed above in it.
- ***Place your code under the home/mininet directory.***
- (1) `mininet@mininet-vm:~/mininet$ sudo python CustomTopo.py`
- (2) Make the script executable
`$ chmod u+x LinearTopo.py`
- Execute the script
`$ sudo ./LinearTopo.py`

Output

```
*** Creating network
```

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s1, s2) (s2, s3) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 4 switches
s1 s2 s3 s4
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (0/12 lost)
*** Stopping 4 hosts
h1 h2 h3 h4
*** Stopping 4 switches
s1 ...s2 ....s3 ....s4 ...
*** Stopping 1 controllers
c0
*** Done
```


Assignment

Background

Data center networks typically have a tree-like topology. End-hosts connect to top-of-rack switches, which form the leaves (edges) of the tree; one or more core switches form the root; and one or more layers of aggregation switches form the middle of the tree. In a basic tree topology, each switch (except the core switch) has a single parent switch. Additional switches and links may be added to construct more complex tree topologies (e.g., fat tree) in an effort to improve fault tolerance or increase inter-rack bandwidth.

In this assignment, your task is to create a simple tree topology. You will assume each level i.e., core, aggregation, edge and host to be composed of a single layer of switches/hosts with a configurable fanout value (k). For example, a simple tree network having a single layer per each level and a fanout of 2 looks like:

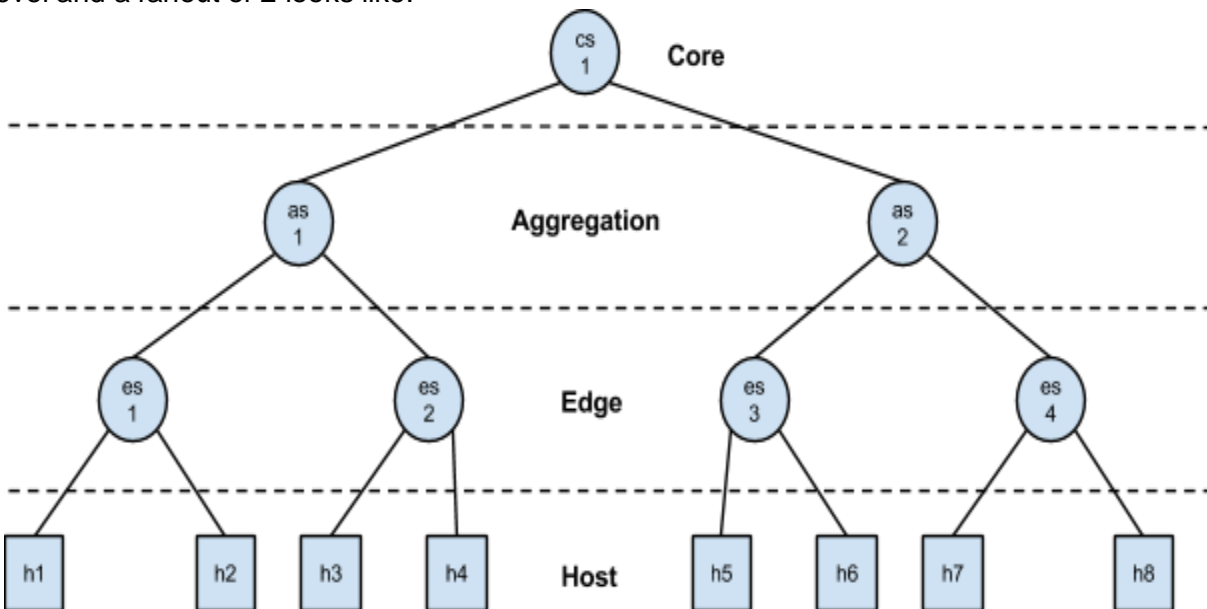


Figure 2: Simple Tree Topology with Fanout 2

To start this exercise,
`CustomTopo.py`: a skeleton class which you will update with the logic for creating the datacenter topology described above.

CustomTopo.py

The skeleton class takes following arguments as input:

- `linkopts1`: for specifying performance parameters for the links between core and aggregation switches.
- `linkopts2`: for specifying performance parameters for the links between

- linkopts3: for specifying performance parameters for the links between edge switches and host
- Fanout: to specify fanout value i.e., number of childs per node.

Submitting your Code

Any plagiarism is prohibited. (ex: from Internet or any other students)

Example

```
shiny@ubuntu:~/mininet$ ls
```

```
shiny@ubuntu: ~/mininet
h1 h2 h3 h4 h5 h6 h7 h8
*** Done
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ 
shiny@ubuntu:~/mininet$ sudo python PA3.py
```

```
#Function to create topo object and Mininet objects

def perfTest():
    "Create network and run simple performance test"
    """--- core to aggregation switches"""
    linkopts1 = {'bw':50, 'delay':'5ms'}
    """--- aggregation to edge switches"""
    linkopts2 = {'bw':30, 'delay':'10ms'}
    """--- edge switches to hosts"""
    linkopts3 = {'bw':10, 'delay':'15ms'}
    topo = CustomTopo(linkopts1 , linkopts3 , fanout= 2)
    net = Mininet(topo=topo, host=CPUimitedHost, link=TCLink)
    net.start()
    net.stop()

topos = { 'custom': ( lambda: CustomTopo() ) }

#Main function
if __name__ == "__main__":
    setLogLevel('info')
    perfTest()
```

Step 3: You'll see the test like this pic.

You could see the iperf test results are the same with your link settings.

```
shiny@ubuntu:~/mininet$ sudo python PA3.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
A1 A2 C1 E1 E2 E3 E4
*** Adding links:
(30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (A1, E1) (30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (A1, E2) (30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (A2, E3) (
30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (A2, E4) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (C1, A1) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (C1, A2) (10.00
Mbit 15ms delay) (10.00Mbit 15ms delay) (h1, E1) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (h2, E1) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (h3, E2) (10.00M
bit 15ms delay) (10.00Mbit 15ms delay) (h4, E2) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (h5, E3) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (h6, E3) (10.00M
bit 15ms delay) (10.00Mbit 15ms delay) (h7, E4) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (h8, E4)
*** Configuring hosts
h1 (cfs -1/1000000us) h2 (cfs -1/1000000us) h3 (cfs -1/1000000us) h4 (cfs -1/1000000us) h5 (cfs -1/1000000us) h6 (cfs -1/1000000us) h7 (cfs -1/1000000us) h8 (cfs -1/1000000us
)
*** Starting controller
c0
*** Starting 7 switches
A1 A2 C1 E1 E2 E3 E4 ... (50.00Mbit 5ms delay) (30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (50.00Mbit 5ms delay) (30.00Mbit 10ms delay) (30.00Mbit 10ms delay) (50.00
Mbit 5ms delay) (50.00Mbit 5ms delay) (30.00Mbit 10ms delay) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (30.00Mbit 10ms delay) (10.00Mbit 15ms delay) (10.00Mbit 15
ms delay) (30.00Mbit 10ms delay) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (30.00Mbit 10ms delay) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay)
*** Starting CLI:
mininet>
mininet> net
mininet> net
h1 h1-eth0:E1-eth2
h2 h2-eth0:E1-eth3
h3 h3-eth0:E2-eth2
h4 h4-eth0:E2-eth3
h5 h5-eth0:E3-eth2
h6 h6-eth0:E3-eth3
h7 h7-eth0:E4-eth2
h8 h8-eth0:E4-eth3
A1 lo: A1-eth1:C1-eth1 A1-eth2:E1-eth1 A1-eth3:E2-eth1
A2 lo: A2-eth1:C1-eth2 A2-eth2:E3-eth1 A2-eth3:E4-eth1
C1 lo: C1-eth1:A1-eth1 C1-eth2:A2-eth1
E1 lo: E1-eth1:A1-eth2 E1-eth2:h1-eth0 E1-eth3:h2-eth0
E2 lo: E2-eth1:A1-eth3 E2-eth2:h3-eth0 E2-eth3:h4-eth0
E3 lo: E3-eth1:A2-eth2 E3-eth2:h5-eth0 E3-eth3:h6-eth0
E4 lo: E4-eth1:A2-eth3 E4-eth2:h7-eth0 E4-eth3:h8-eth0
c0
mininet>
```

Step 4: mininet>>net (to verify your topo)

mininet>>pingall (to make sure your topo work properly)

```
mininet>
mininet> net
h1 h1-eth0:E1-eth2
h2 h2-eth0:E1-eth3
h3 h3-eth0:E2-eth2
h4 h4-eth0:E2-eth3
h5 h5-eth0:E3-eth2
h6 h6-eth0:E3-eth3
h7 h7-eth0:E4-eth2
h8 h8-eth0:E4-eth3
A1 lo: A1-eth1:C1-eth1 A1-eth2:E1-eth1 A1-eth3:E2-eth1
A2 lo: A2-eth1:C1-eth2 A2-eth2:E3-eth1 A2-eth3:E4-eth1
C1 lo: C1-eth1:A1-eth1 C1-eth2:A2-eth1
E1 lo: E1-eth1:A1-eth2 E1-eth2:h1-eth0 E1-eth3:h2-eth0
E2 lo: E2-eth1:A1-eth3 E2-eth2:h3-eth0 E2-eth3:h4-eth0
E3 lo: E3-eth1:A2-eth2 E3-eth2:h5-eth0 E3-eth3:h6-eth0
E4 lo: E4-eth1:A2-eth3 E4-eth2:h7-eth0 E4-eth3:h8-eth0
c0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

The names of hosts and switch are the same with the pic above.

Some unexpected errors may occur when not closing controller/topogry normally.

You need to use the command “**sudo mn -c**” to clear all the objects to make your environment clean.