Sharif University of Technology

Computer Engineering Department

Software-Defined Networking

Ali Movaghar

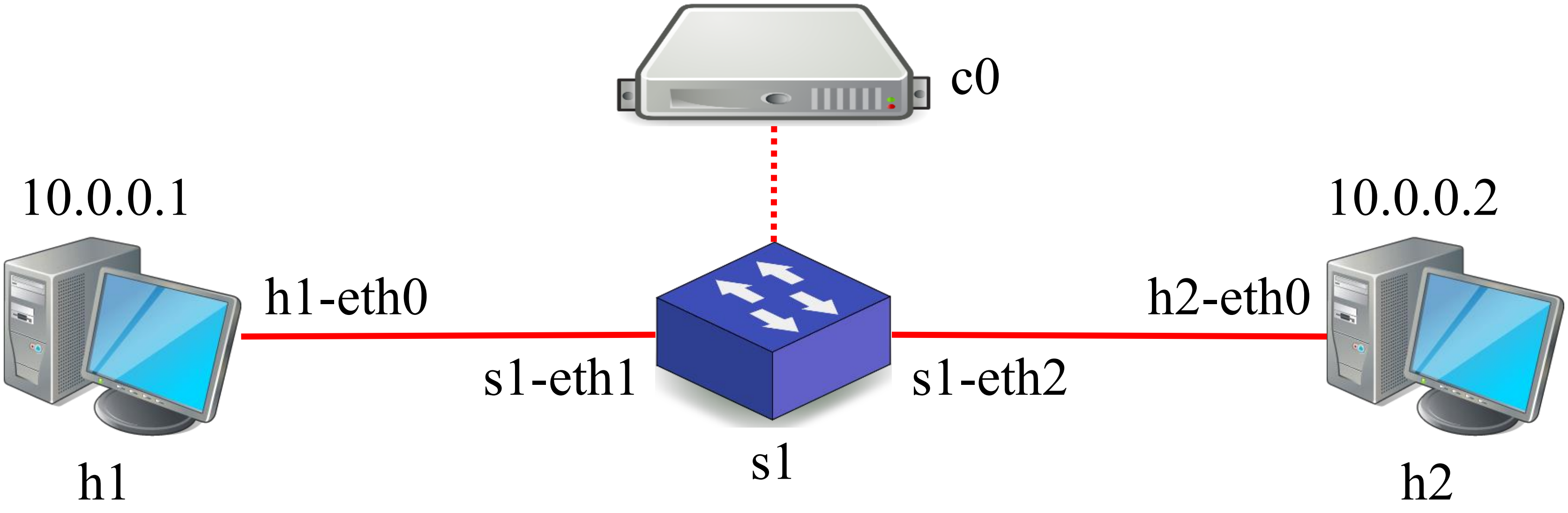
Mohammad Hosseini

Mininet and OVS

TA: Iman Rahmati & Farbod Shahinfar

Starting Mininet and creating a network

```
$ sudo mn
```



Mininet’s default minimal topology

- Mininet’s command prompt:

```
mininet>
```

- Exit Mininet:

```
mininet> exit
```

Help

- Show Mininet’s help

```
$ mn -h
```

- Show Mininet’s CLI commands

```
mininet> help
```

- Show Mininet’s CLI commands

```
mininet> help [command]
```

- If Mininet crashes, clean it up:

```
$ sudo mn -c
```

Network Information

- List all network nodes

```
Mininet> nodes
```

- Show information of all nodes

```
mininet> dump
```

- Show all network links

```
mininet> links
```

- Show network connections of all nodes

```
mininet> net
```

Testing Network

- ping between all hosts and return connectivity results

```
mininet> pingall
```

- ping between all hosts and return timing results

```
mininet> pingallfull
```

- Measure the TCP throughput between two hosts:

```
mininet> iperf [node1] [node2]
```

The command runs an iperf TCP server on the first virtual host and an iperf client on the second virtual host, and then measures the bandwidth.

- Test throughput by UDP links

```
mininet> iperfudp [bw] [node1] [node2]
mininet> iperf 1.5G h1 h2
```

- Bring link(s) between nodes up or down

```
mininet> link [node1] [node2] [up/down]
```

Working with Hosts

- Run a command on a host

```
mininet> [hostname] [command]
mininet> h1 ifconfig
mininet> h1 ping h2
```

- Open a terminal for a host

```
mininet> xterm [hostname] …
mininet> xterm h1
mininet> xterm h1 h2
```

“xterm” terminal emulator must be installed on your system

- "sh" is used for commmands that need to be run from system shell instead of mininet prompt

```
mininet> sh [command]
mininet> sh ping google.com
```

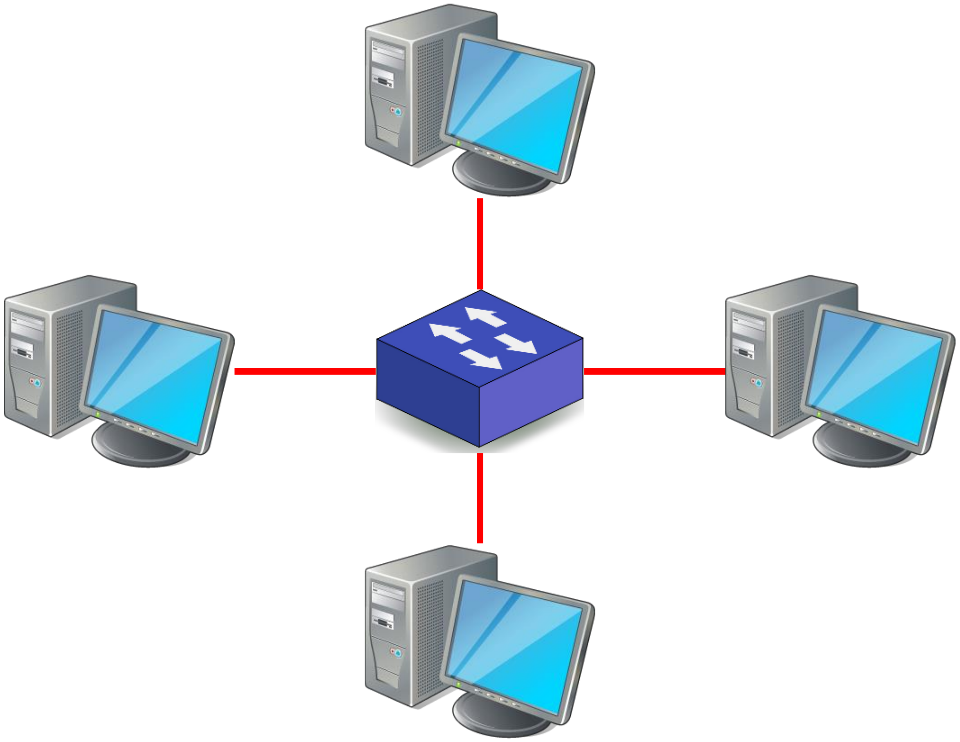
Topologies

- Specify network topology by --topo:

```
$ sudo mn --topo [topology_name],[topology_parameters]
```

- Topology: **single**

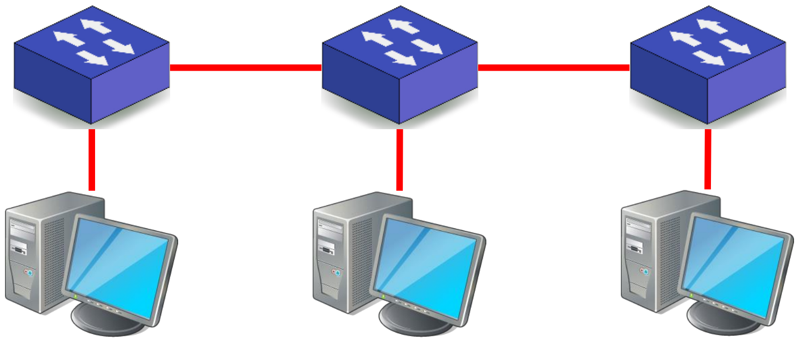
```
$ sudo mn --topo single,4
```



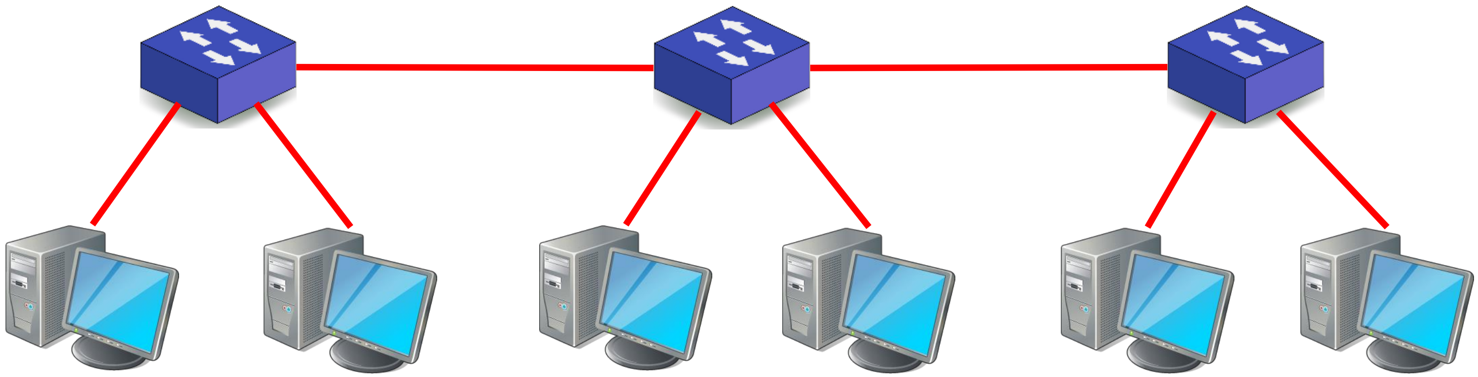
Topologies

- Topology: **linear**

```
$ sudo mn --topo linear,3
```



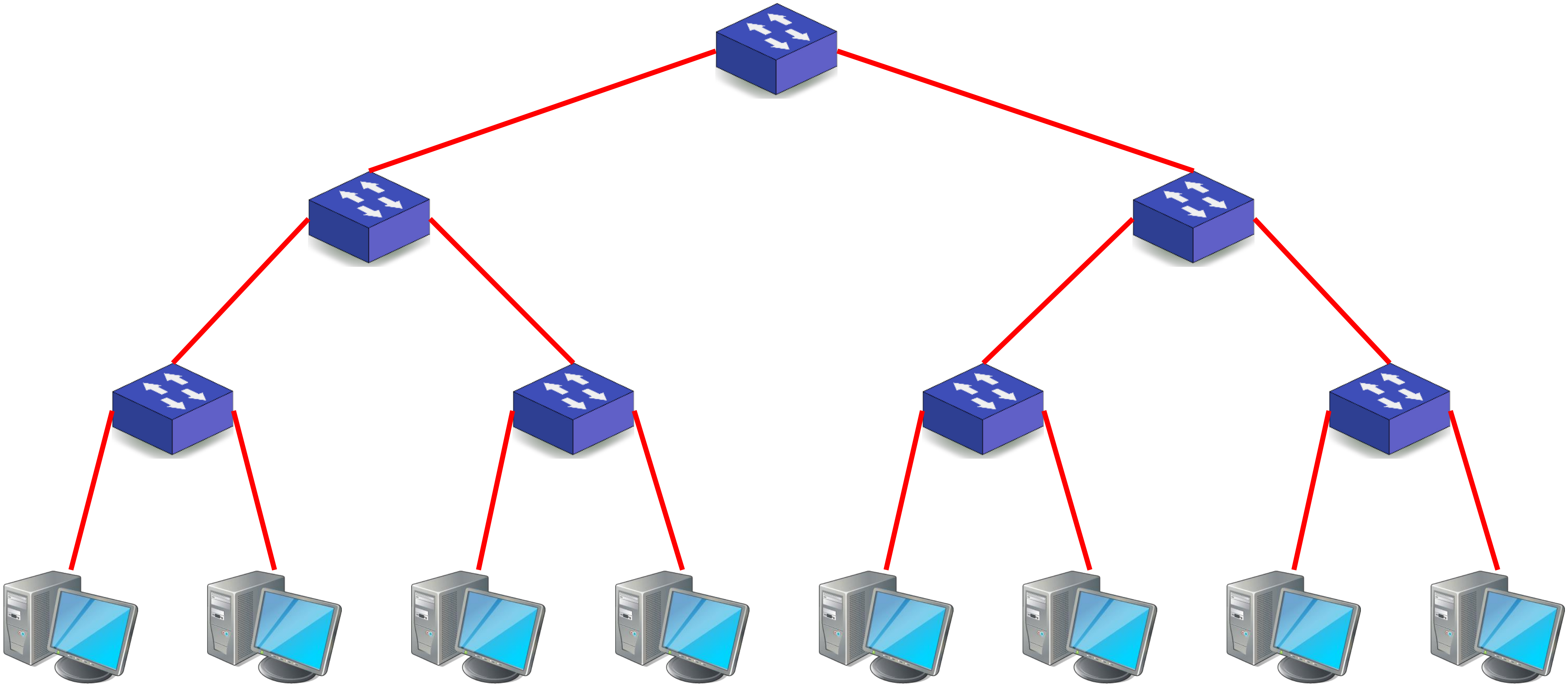
```
$ sudo mn --topo linear,3,2
```



Topologies

- Topology: **tree**

```
$ sudo mn --topo tree,3,2
$ sudo mn --topo tree,depth=3,fanout=2
```



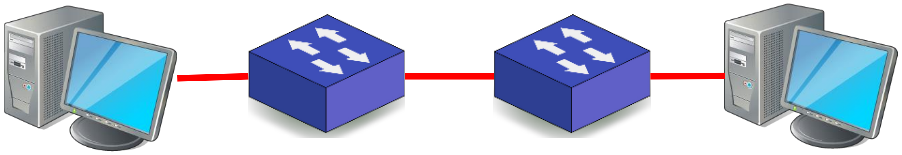
Custom Topology

mycustomtopo.py

```
from mininet.topo import Topo

class MyTopo( Topo ):
    def build( self ):
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )
        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```



Adding the 'topos' dictionary with a key/value pair to generate our newly defined topology enables one to pass in '--topo=mytopo' from the command line.

```
$ sudo mn --custom mycustomtopo.py --topo mytopo
```

Link Settings

- Links and their characteristics can be specified by the parameter “--link”

```
$ sudo mn --link tc,bw=100
```

Link bandwidth: 100 Mbits/sec

```
$ sudo mn --link tc,bw=100,delay=10ms
```

```
$ sudo mn --link tc,bw=100,delay=10ms,loss=25
```

Packet loss rate: 25%

```
$ sudo mn --link tc,max_queue_size=1000
```

Switch and Controller Settings

- Switches and their characteristics can be specified by the parameter "--switch"

```
$ sudo mn --switch ovs,protocols=OpenFlow13
```

- The controller can be specified by the parameter "--controller"

```
$ sudo mn --controller none
```

```
$ sudo mn --controller remote
$ sudo mn --controller remote,ip=127.0.0.1
$ sudo mn --controller remote,ip=127.0.0.1,port=6633
$ sudo mn --controller remote,ip=127.0.0.1,port=6653
```

Open vSwitch

- **ovs-vsctl** is a utility that comes with Open vSwitch and enables us to monitor and configure Open vSwitch instances.
- Various parameters such as switches, their ports, flow table settings, OpenFlow version, fail-mode, queue settings, and controller settings can be configured by this tool which is based on OVSDB protocol.
- Show the name of Open vSwitch instances

```
$ sudo ovs-vsctl list-br
```

- Show information of switch instances

```
$ sudo ovs-vsctl show
```

- Show the controllers of a switch

```
$ sudo ovs-vsctl get-controller [switch_name]
```

- Get/Set a parameter such as OpenFlow version

```
$ sudo ovs-vsctl get bridge [switch_name] protocols
$ sudo ovs-vsctl set bridge [switch_name] protocols=OpenFlow13
```

Open vSwitch

- **ovs-ofctl** program is a command line tool for monitoring and administering OpenFlow switches.
- It can show and modify the current state of an OpenFlow switch, including features, configuration, and table entries.
- It works with any OpenFlow switch, not just Open vSwitch.

- Show switch capabilities and its ports

```
$ sudo ovs-ofctl show [switch_name]
```

- Show flow table entries

```
$ sudo ovs-ofctl dump-flows [switch_name]
```

- Show table statistics

```
$ sudo ovs-ofctl dump-tables [switch_name]
```

- For remote switches (the control port can be obtained by ovs-vsctl

```
$ ovs-ofctl dump-flows tcp:127.0.0.1:6634
```

Open vSwitch

- ovs-ofctl examples:

```
$ sudo ovs-ofctl del-flows s1
$ sudo ovs-ofctl add-flow s1 priority=0,action=normal
$ sudo ovs-ofctl add-flow s1 priority=10,action=drop

$ sudo ovs-ofctl add-flow s1
priority=500,in_port=1,actions=output:2

$ sudo ovs-ofctl add-flow s1
in_port=1,dl_dst=00:00:00:00:00:02,actions=output:2

$ sudo ovs-ofctl add-flow s1
dl_type=0x806,nw_proto=1,actions=flood

$ sudo ovs-ofctl add-flow s1
nw_src=10.0.0.0/24,nw_dst:10.0.0.0/24,actions=normal
```

Open vSwitch

- ovs-ofctl examples:

```
$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
$ sudo ovs-ofctl --protocols=OpenFlow13 dump-flows s1
```

Mininet scripting

- ❖ You can make use of Mininet's python library and write Mininet scripts to automate your experiments.
- ❖ Many example scripts can be found in the example directory of Mininet.

- ➢ To run a script:

```
$ sudo python yourscript.py
```

- ➢ or if you have installed the python2 version of Mininet:

```
$ sudo python2 yourscript.py
```

Mininet modules

- Useful Mininet modules:

```
from mininet.net import Mininet
from mininet.node import RemoteController, OVSKernelSwitch
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel, info
```

Creating a network

- ➢ Creating a Mininet network:

```
net = Mininet()
```

- ➢ Adding a remote controller:

```
c0 = net.addController(' c0 ', controller=RemoteController, ip=
'127.0.0.1 ')
```

- ➢ Adding a switch:

```
s1 = net.addSwitch( 's1' )
s1 = net.addSwitch( 's1', switch=OVSKernelSwitch,
protocols='OpenFlow13' )
```

- ➢ Adding a host:

```
h1 = net.addHost( 'h1' )
h1 = net.addHost( 'h1', mac='00:00:00:00:00:01', ip='10.0.0.1' )
```

Creating a network

- Adding a link:

```
net.addLink( h1, s1 )
net.addLink( h1, s1, cls=TCLink, delay= '10ms' )
net.addLink( h1, s1, cls=TCLink, delay= '10ms' , bw=100, loss=0,
max_queue_size=100)

# or
net = Mininet(link=TCLink)
net.addLink( h1, s1, delay= '10ms' )
```

Running a network

- Starting a network:

```
net.start()
```

- Openning Mininet’s CLI:

```
CLI( net )
```

- Stopping the network:

```
net.stop()
```

- Mininet’s built-in tests:

```
net.pingAll()

net.iperf( ( h1, h2 ), l4Type='UDP' )
```

Working with hosts

- Running a command in a host:

```
h1.cmd('ping -c1 10.0.0.2')

result = h1.cmd('ping -c1 10.0.0.2')
print(result)
```

```
print(h1.IP())
print(h1.MAC())

h1.setIP('10.0.0.101')
h1.setMAC('00:00:00:00:00:A')
```

Other useful functions

- Suspending execution (in seconds):

```
import time

time.sleep(10)
```

- To run a command in your OS shell:

```
import os

cmd = 'mkdir results'
os.system(cmd)
```

- To pass arguments to the script from command-line:

```
import sys

print(str(sys.argv[1]))
for arg_i in range (1, len(sys.argv)):
    print("arg_%d: %s" % (arg_i, str(sys.argv[arg_i])) )
```