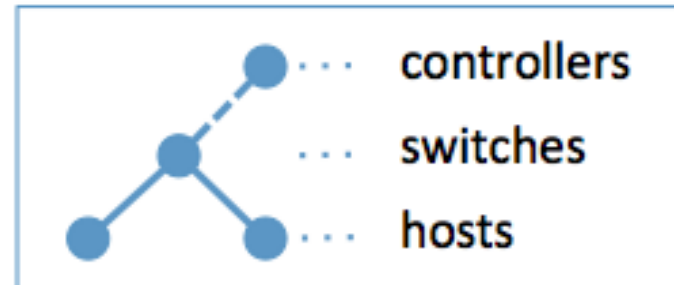


Mininet


```
> sudo mn
```




What is Mininet?

- ⦿ A **virtual network environment** that can run on a single PC
- ⦿ Runs real kernel, switch, and application code on a single machine
 - Command-line, UI, Python interfaces
- ⦿ Many **OpenFlow features** are built-in
 - Useful: developing, deploying, and sharing

Why Use Mininet?

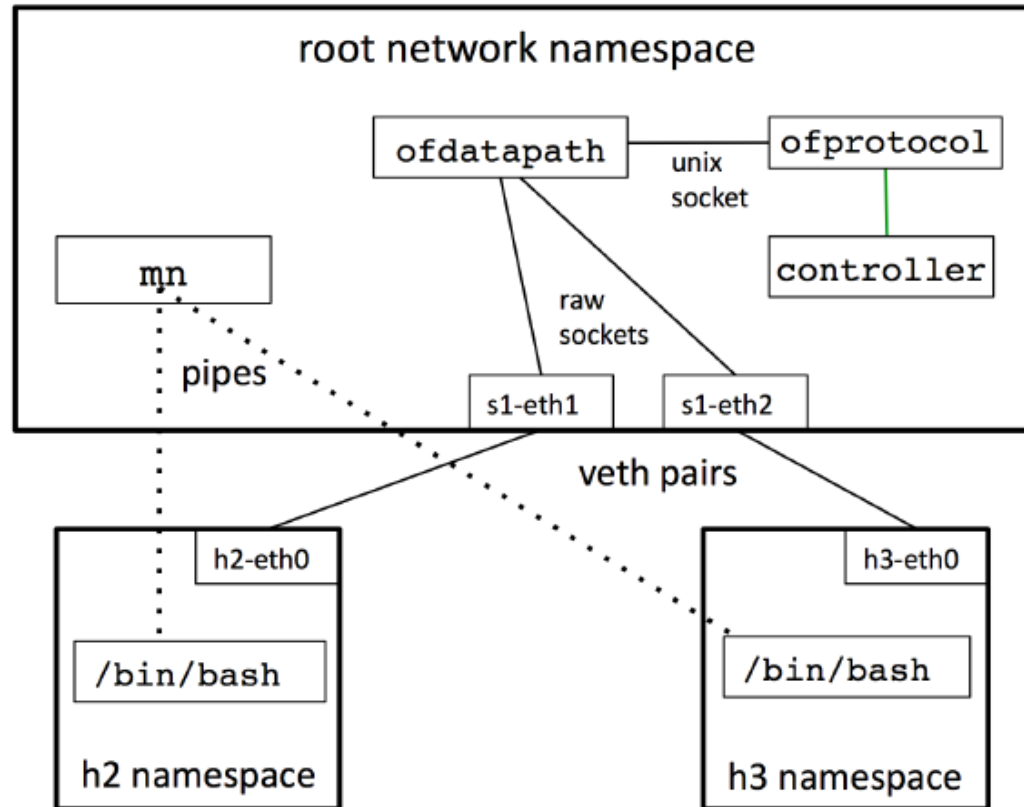
- ⦿ Fast
 - ⦿ Possible to create custom topologies
 - ⦿ Can run real programs (anything that can run on Linux can run on a Mininet host)
 - ⦿ Programmable OpenFlow switches
 - ⦿ Easy to use
 - ⦿ Open source
- 

Alternatives

- ⦿ **Real system:** Pain to configure
 - ⦿ **Networked VMs:** Scalability
 - ⦿ **Simulator:** No path to hardware deployment
- 

The Mininet VM in a Nutshell

Virtual Machine

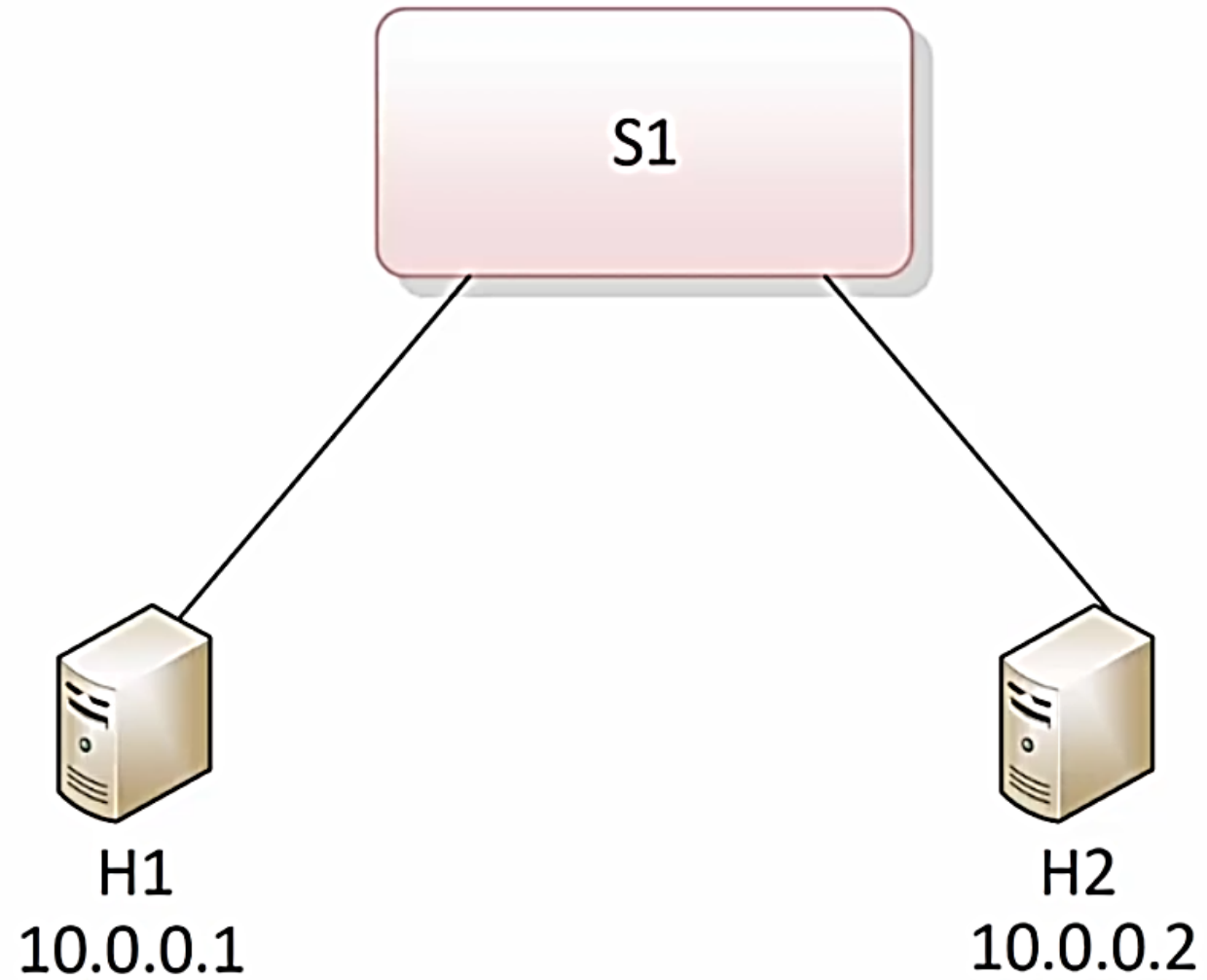


- Launch mininet process
- Per host
 - Bash process
 - Network namespace
- Create veth pairs and assign to namespaces
- Create OpenFlow switch to connect hosts
- Create OpenFlow controller

What are Linux Network Namespaces?

- Multiple isolated networking environments running on a single physical host or VM
- Each network namespace has its own interfaces, routing tables and forwarding tables
- Processes can be dedicated to one network namespace
- Used in OpenStack, Mininet, Docker, more...

Example



Root Namespace



Demo: basic network setup in Linux

```
sudo bash
```

```
# Create host namespaces
```

```
ip netns add h1
```

```
ip netns add h2
```

```
# Create switch
```

```
ovs-vsctl add-br s1
```

```
# Create links
```

```
ip link add h1-eth0 type veth peer name s1-eth1
```

```
ip link add h2-eth0 type veth peer name s1-eth2
```

```
ip link show
```

```
# Move host ports into namespaces
```

```
ip link set h1-eth0 netns h1
```

```
ip link set h2-eth0 netns h2
```

```
ip netns exec h1 ip link show
```

```
ip netns exec h2 ip link show
```

```
# Connect switch ports to OVS
```

```
ovs-vsctl add-port s1 s1-eth1
```

```
ovs-vsctl add-port s1 s1-eth2
```

```
ovs-vsctl show
```

```
# Set up OpenFlow controller
```

```
ovs-vsctl set-controller s1 tcp:127.0.0.1
```

```
ovs-controller ptcp: &
```

```
ovs-vsctl show
```

```
# Configure network
```

```
ip netns exec h1 ifconfig h1-eth0 10.1
```

```
ip netns exec h1 ifconfig lo up
```

```
ip netns exec h2 ifconfig h2-eth0 10.2
```

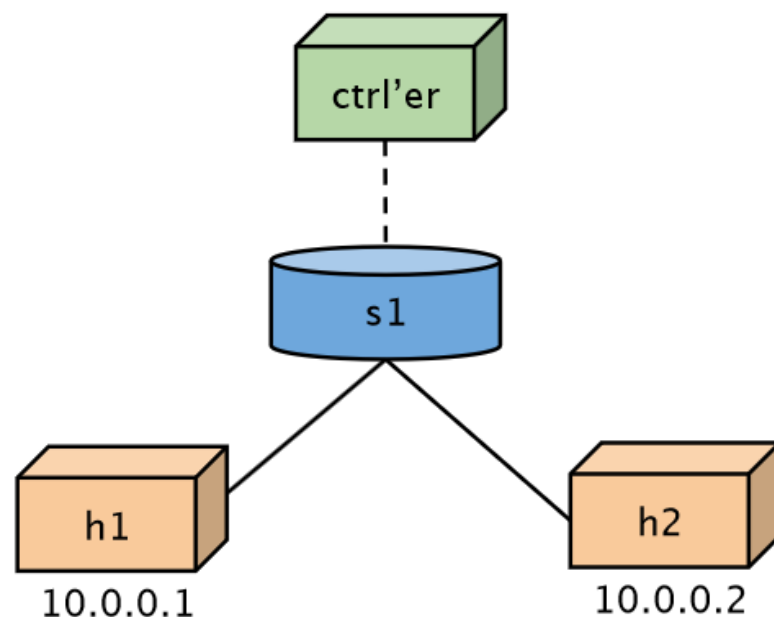
```
ip netns exec h1 ifconfig lo up
```

```
ifconfig s1-eth1 up
```

```
ifconfig s1-eth2 up
```

```
# Test network
```

```
ip netns exec h1 ping -c1 10.2
```



Testing a Simple Mininet Setup

- ① Try setting up a simple topology with three hosts connected to a single switch:
 - `sudo mn --test pingall --topo single,3`
- ① This setup uses a default switch controller and switch
 - Mininet also allows you to use custom remote controllers (and custom switches)

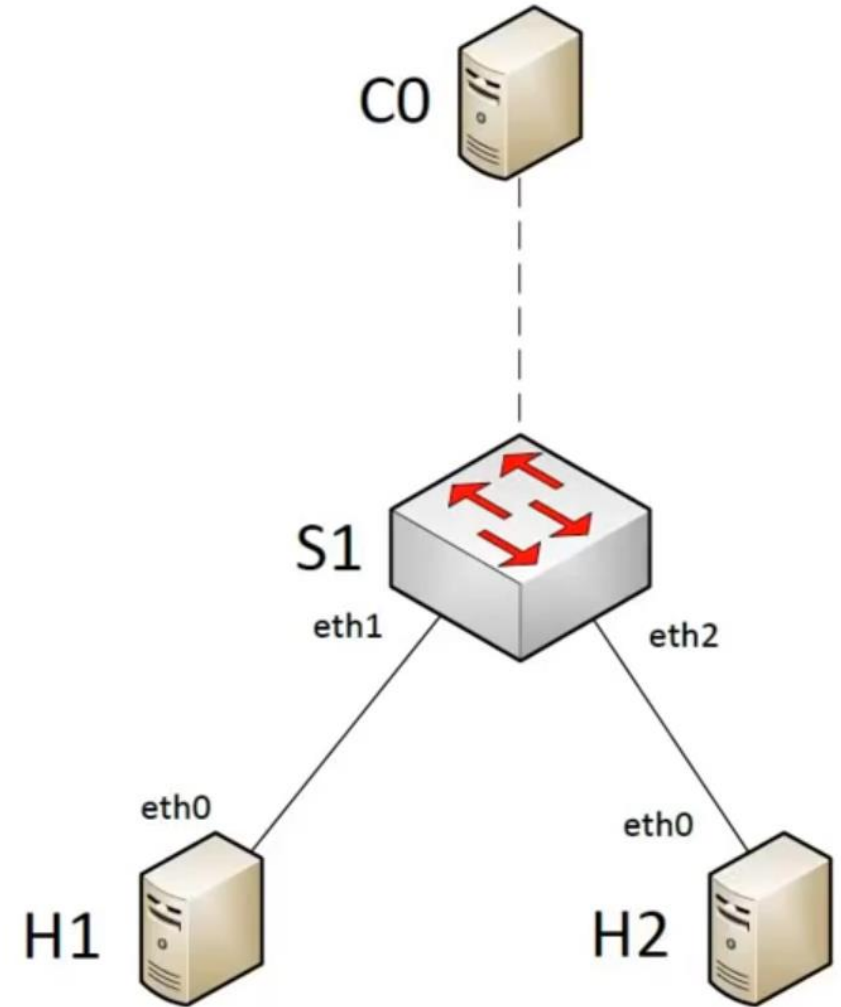
Basic Mininet Command Line

- ⦿ **--topo** – defines a topology via command line upon mininet start-up.
- ⦿ **--switch** – defines the switch to be used. By default the OVSF software switch is used.
- ⦿ **--controller** – defines the controller to be used. If unspecified default controller is used with a default hub behavior.

Trying Out Different Mininet Topologies

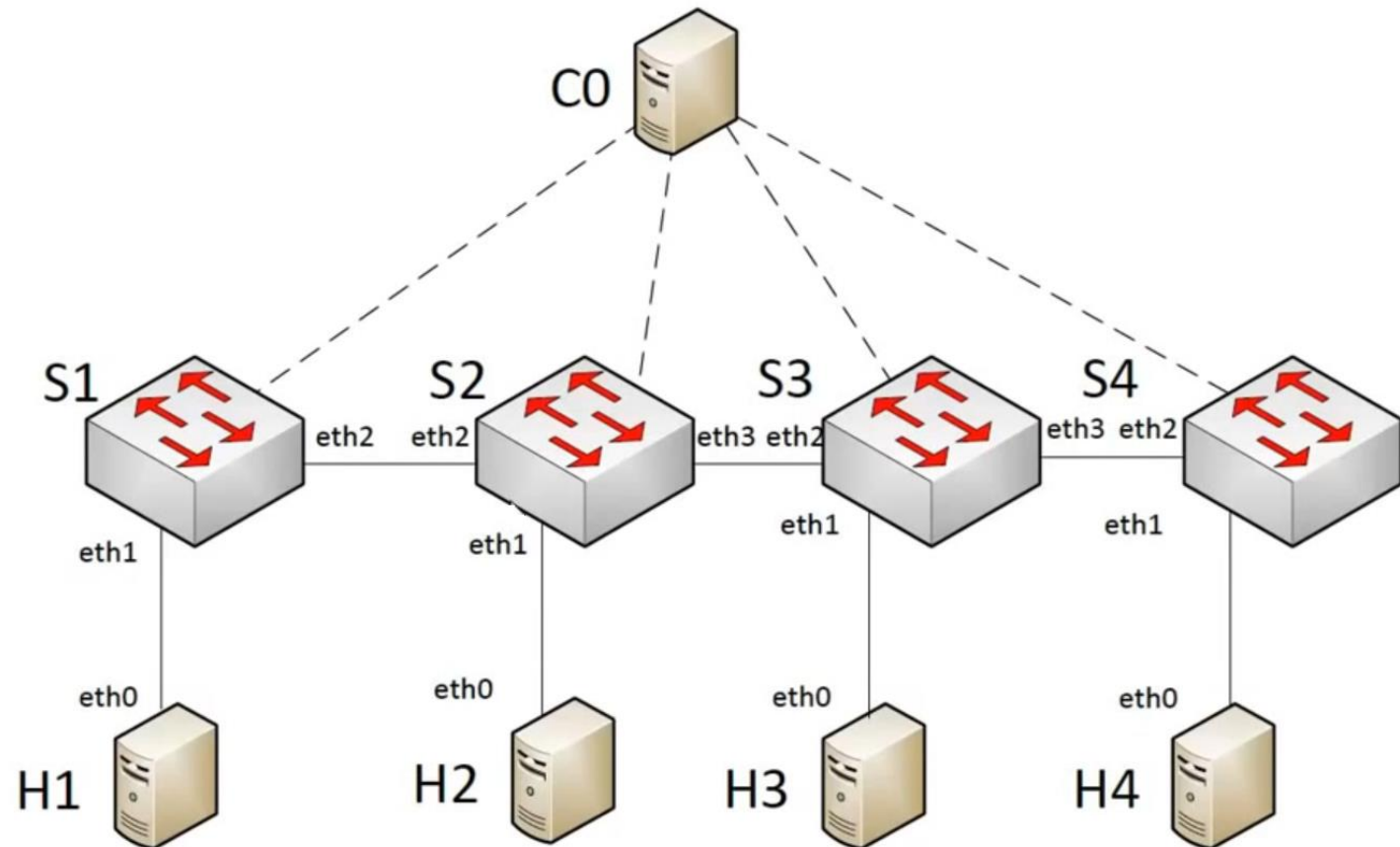
- ⦿ Minimal network with two hosts, one (1) switch
 - `sudo mn -topo minimal`
- ⦿ Example with 4 hosts and 4 switches
 - `sudo mn --topo linear,4`
- ⦿ Example with 4 hosts all connected to one switch.
 - `sudo mn --topo single,4`
- ⦿ Tree topology with defined depth and fan-out.
 - `sudo mn --topo tree,depth=2,fanout=2`

- Minimal network with two hosts, one (1) switch
 - `sudo mn -topo minimal`

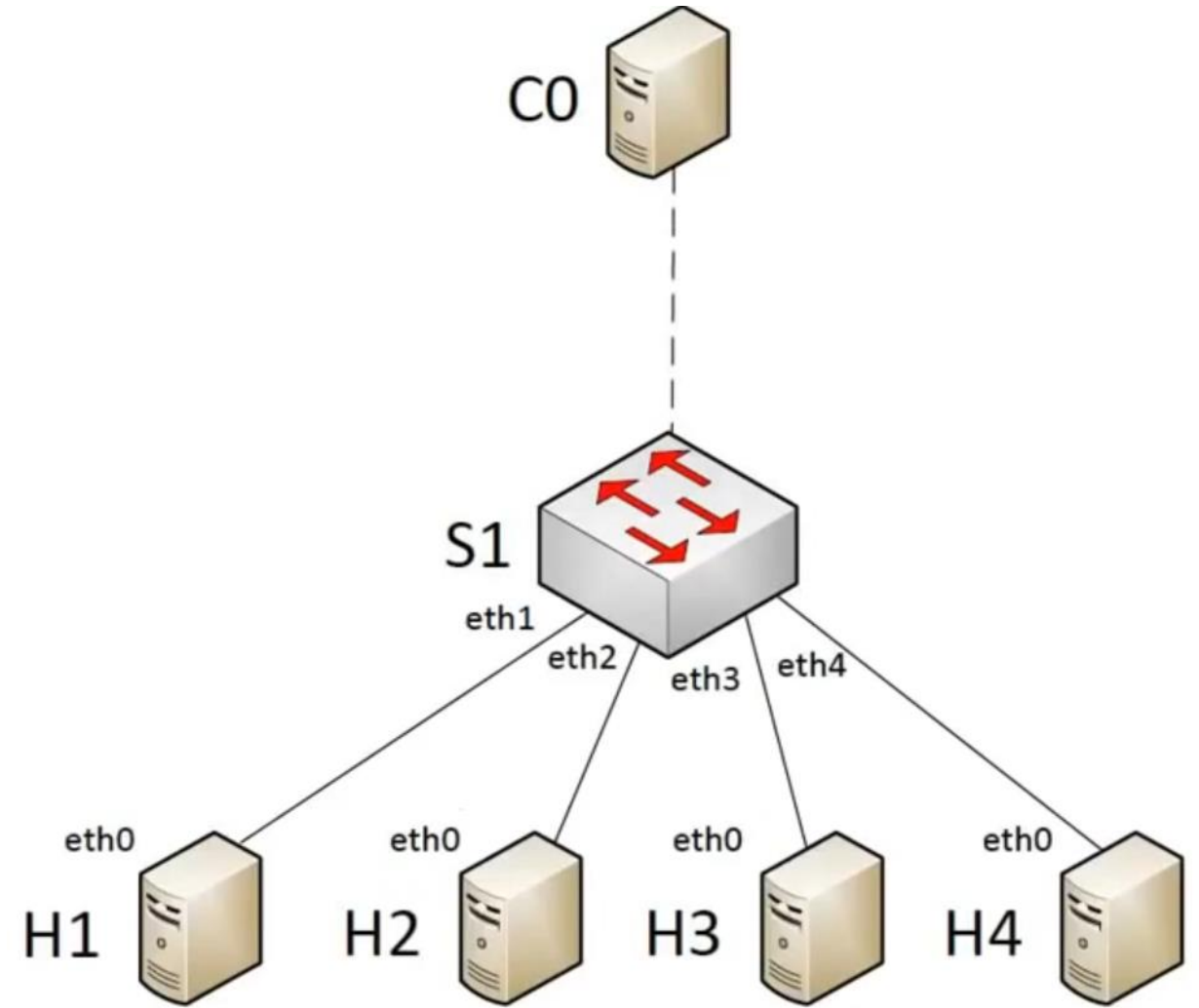


Example with 4 hosts and 4 switches

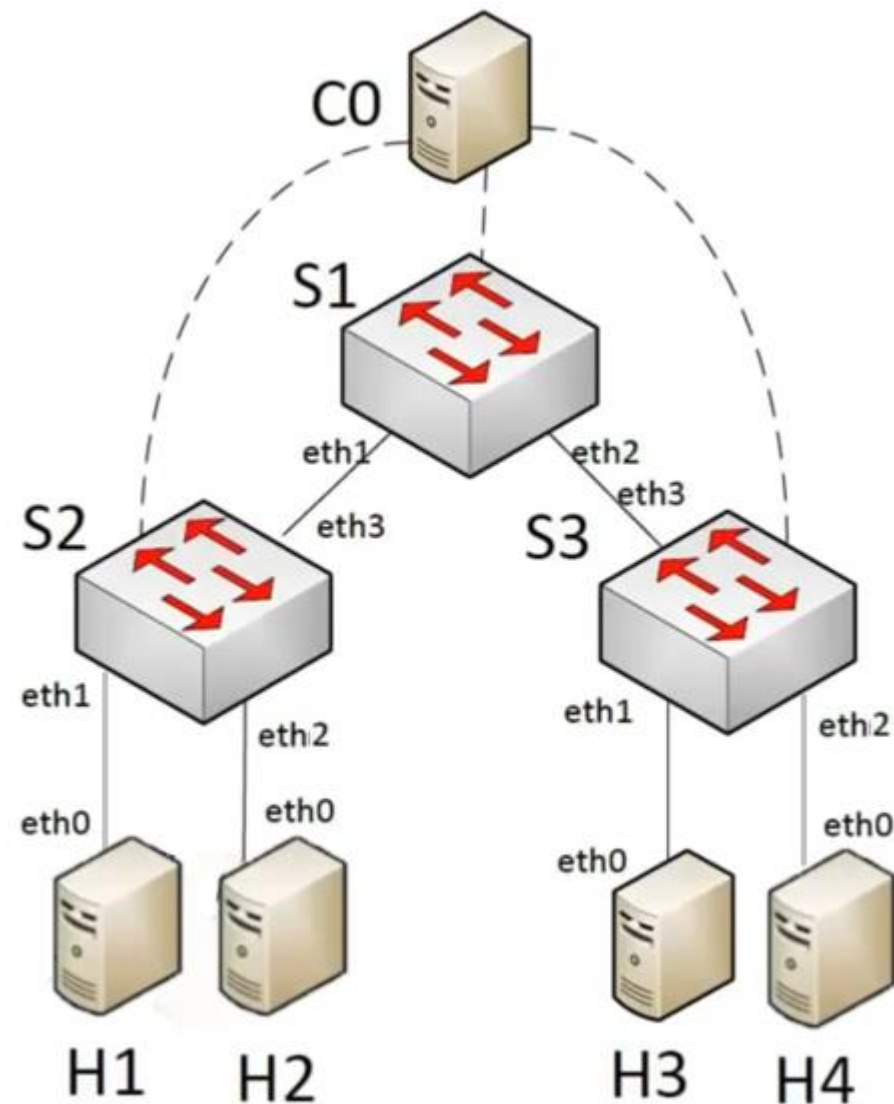
- `sudo mn --topo linear,4`



- Example with 4 hosts all connected to one switch.
 - `sudo mn --topo single,4`



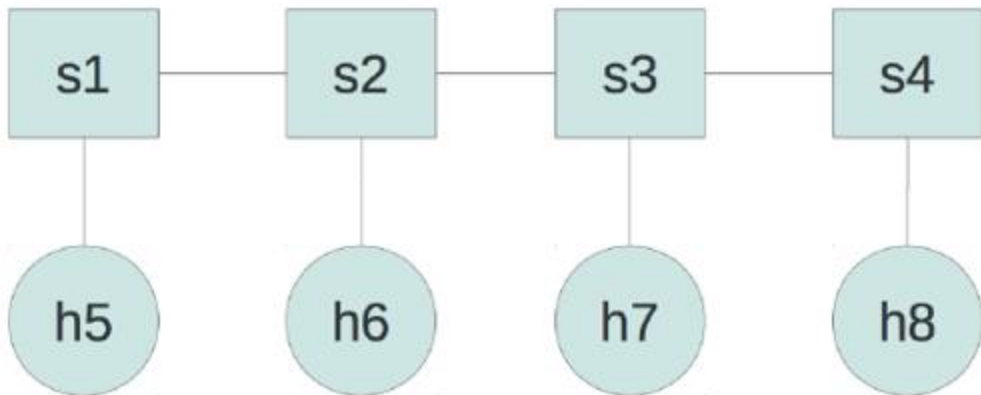
- Tree topology with defined depth and fan-out.
 - `sudo mn --topo tree,depth=2,fanout=2`



How mn Works: mn executes Python

- “mn” is a launch script that executes Python
- Consider: “—topo linear,4”

```
from mininet.net import Mininet  
  
from mininet.topo import LinearTopo  
  
Linear = LinearTopo(k=4)  
  
net = Mininet(topo=Linear)  
  
net.start()  
net.pingAll()  
net.stop()
```



Writing Your Own Mininet Topologies

- Example: Two hosts, one switch
- mininet.cli.CLI(net)** before `net.stop()` will escape to interactive CLI before script terminates
- addLink** allows you to specify: Bandwidth (bw) in Mbps, Delay (delay), Maximum Queue Size (max_queue_size), Loss (loss) in percentage

```
from mininet.net import Mininet
from mininet.util import createLink
net = Mininet()
```

```
# Creating nodes in the network.
c0 = net.addController()
h0 = net.addHost('h0')
s0 = net.addSwitch('s0')
h1 = net.addHost('h1')
```

```
# Creating links between nodes in network (2-ways)
net.addLink(h0, s0)
net.addLink(h1, s0)
```

```
# Configuration of IP addresses in interfaces
h0.setIP('192.168.1.1', 24)
h1.setIP('192.168.1.2', 24)
```

```
net.start()
net.pingAll()
net.stop()
```

More Complicated Topology Generation

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

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

```
class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def __init__(self, n=2, **opts):

        # Initialize topology and default options
        Topo.__init__(self, **opts)
        switch = self.addSwitch('s1')

        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)
```

```
def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=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()
```

Mininet Command Line Interface Usage

❖ Mininet Command Line Interface Usage

- *Interact with hosts and switches*

- Start a minimal topology

```
$ sudo mn
```

- Start a minimal topology using a remote controller

```
$ sudo mn --controller=remote,ip=[IP_ADDRESS],port=[listening port]
```

- Start a custom topology

```
$ sudo mn --custom [topo_script_path] --topo=[topo_name]
```

- Display nodes

```
mininet> nodes
```

- Display links

```
mininet> net
```

- Dump information about all nodes

```
mininet> dump
```

Mininet Command Line Interface Usage

❖ Mininet Command Line Interface Usage

■ Interact with hosts and switches

- Check the IP address of a certain node

```
mininet> h1 ifconfig -a
```

- Print the process list from a host process

```
mininet> h1 ps -a
```

■ Test connectivity between hosts

- Verify the connectivity by pinging from host1 to host2

```
mininet> h1 ping -c 1 h2
```

- Verify the connectivity between all hosts

```
mininet> pingall
```

Grading Scheme

