TOPOLOGY

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel
class CustomTopo(Topo):
    def build(self):
        # Create hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')
        # Create switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        # Add links between switches (linear topology)
        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s3, s4)
        # Add links between hosts and switches
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s2)
        self.addLink(h4, s2)
        self.addLink(h5, s3)
        self.addLink(h6, s3)
```

```
        self.addLink(h7, s4)
        self.addLink(h8, s4)
# Register the custom topology
topos = {'custom': (lambda: CustomTopo())}
```

Breakdown of TOPOLOGY.py (Custom Mininet Topology Script)

Imports

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel
```

✓ These imports bring in the necessary Mininet modules:

- Topo: Used to define the network topology.
- Mininet: The core Mininet network simulation class.
- OVSSwitch: Specifies that Open vSwitch (OVS) will be used.
- RemoteController: Allows connecting to an external (remote) SDN controller.
- CLI: Enables interaction with Mininet's command-line interface.
- setLogLevel: Configures logging verbosity.

Custom Topology Class

```
class CustomTopo(Topo):
    def build(self):
```

✓ **Defines a custom topology** by extending Topo class.

Creating Hosts

```
        # Create hosts
```

```
h1 = self.addHost('h1')
h2 = self.addHost('h2')
h3 = self.addHost('h3')
h4 = self.addHost('h4')
h5 = self.addHost('h5')
h6 = self.addHost('h6')
h7 = self.addHost('h7')
h8 = self.addHost('h8')
```

✓ **8 hosts are created:** h1 to h8.

Creating Switches

```
# Create switches
s1 = self.addSwitch('s1')
s2 = self.addSwitch('s2')
s3 = self.addSwitch('s3')
s4 = self.addSwitch('s4')
```

✓ **4 Open vSwitches (OVS) are created:** s1 to s4.

Connecting Switches (Linear Topology)

```
# Add links between switches (linear topology)
self.addLink(s1, s2)
self.addLink(s2, s3)
self.addLink(s3, s4)
```

✓ **Forms a linear switch topology:**
s1 — s2 — s3 — s4

Connecting Hosts to Switches

```
# Add links between hosts and switches
self.addLink(h1, s1)
```

```
        self.addLink(h2, s1)
        self.addLink(h3, s2)
        self.addLink(h4, s2)
        self.addLink(h5, s3)
        self.addLink(h6, s3)
        self.addLink(h7, s4)
        self.addLink(h8, s4)
```

✓ **Hosts are connected as follows:**

- h1, h2 → s1
- h3, h4 → s2
- h5, h6 → s3
- h7, h8 → s4

✓ This structure **divides the network into separate segments**, with two hosts per switch.

Registering the Topology

```
# Register the custom topology
topos = {'custom': (lambda: CustomTopo())}
```

✓ Registers the topology so that Mininet can **load it with the --topo option**.

Breakdown of Mininet Command to Start the Topology

$ sudo mn --custom TOPOLOGY.py --topo custom --controller=remote,ip=127.0.0.1,port=6633 --switch=ovs,protocols=OpenFlow10

✓ This command **starts Mininet with the custom topology**.

- **Breakdown of the options:**
  - --custom TOPOLOGY.py → Loads the custom topology from the script.
  - --topo custom → Uses the topology named custom (registered in the script).
  - --controller=remote,ip=127.0.0.1,port=6633 →

    - Mininet **does not start its own controller** but connects to a **remote controller** running at 127.0.0.1 on port 6633.
- --switch=ovs,protocols=OpenFlow10 →
    - Uses **Open vSwitch (OVS)** as the switch type.
    - Configures OVS to use **OpenFlow 1.0** (since POX supports OpenFlow 1.0 by default).

Breakdown of POX Controller Command

$ ./pox.py forwarding.l2_learning openflow.discovery host_tracker samples.pretty_log log.level --DEBUG openflow.of_01 --address=127.0.0.1 --port=6633

✓ This command **starts the POX SDN controller**.

🔹 **Breakdown of modules:**

- forwarding.l2_learning → Implements **Layer 2 (L2) learning switch** behavior.
- openflow.discovery → Enables **link discovery** between switches.
- host_tracker → Tracks host MAC addresses and locations.
- samples.pretty_log → Provides better log formatting.
- log.level --DEBUG → Enables detailed **debugging logs**.
- openflow.of_01 → Configures the controller to use **OpenFlow 1.0**.
- --address=127.0.0.1 --port=6633 →
    - Binds the controller to **127.0.0.1:6633** (matching Mininet's controller settings).

Summary

✅ **TOPOLOGY.py** defines:

- 8 hosts, 4 switches in a **linear topology**.
- Hosts are evenly distributed across the switches.

✅ **Mininet Command:**

- Loads the custom topology and connects to a **remote POX controller**.

✅ **POX Controller Command:**

- Runs POX with essential modules, enabling **L2 switching, link discovery, and host tracking**.

Next Steps

🚀 **Start Mininet**:

```
sudo mn --custom TOPOLOGY.py --topo custom --
controller=remote,ip=127.0.0.1,port=6633 --switch=ovs,protocols=OpenFlow10
```

🚀 **Start POX Controller**:

```
./pox.py forwarding.l2_learning openflow.discovery host_tracker
samples.pretty_log log.level --DEBUG openflow.of_01 --address=127.0.0.1 --
port=6633
```

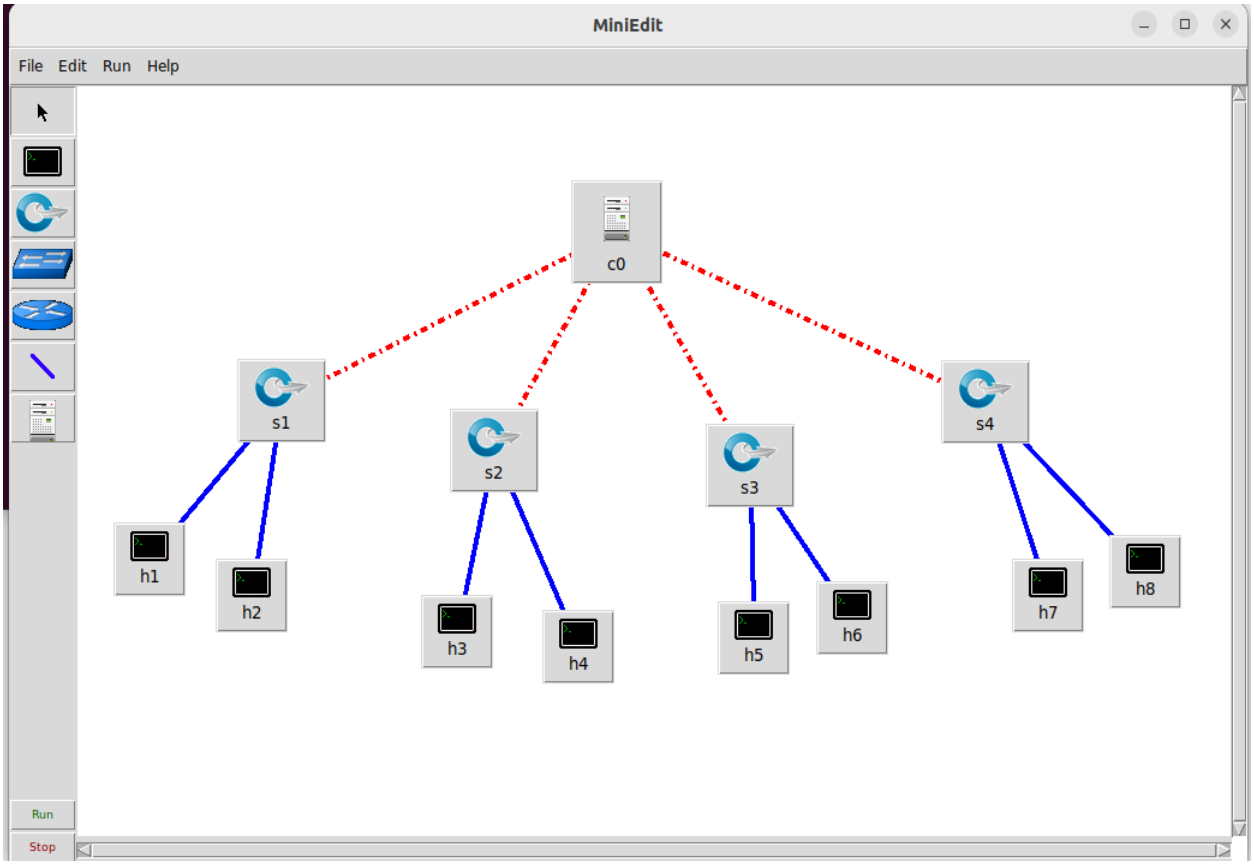
📌 **Use Mininet CLI**:

```
mininet> pingall
```

✓ To check if all hosts can communicate.

📌 **View POX logs**:

✓ To monitor controller activity.


MiniEdit
File
Edit
Run
Help
c0
s1
s2
s3
s4
h1
h2
h3
h4
h5
h6
h7
h8
Run
Stop