ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Mininet basics: hands-on session

Chiara Grasselli

LAB. OF NETWORK PROGRAMMABILITY AND AUTOMATION - PROGRAMMABLE NETWORKING (A.Y. 2024/2025)

Starting Mininet

- You can
  - Run the **default network topology**, which consists of 2 hosts, 1 switch, and 1 controller (the default controller)
- Or
  - Run a **custom network topology** with the default controller or specifying an external controller

- You can execute the Mininet default topology by typing on your Terminal: **sudo mn** (requires root privileges).
- Use **sudo mn –c** to perform cleanup after leaving the Mininet CLI or if errors occur.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Basics operations

Mininet has its own Command-Line Interface (CLI) - **mininet>** - with which you can:

- see network elements, by typing command **nodes**
- see network topology, by typing commands **net** and **links**
- test connectivity between hosts, by typing command **pingall**
- open a Terminal for specific nodes of the network, by typing command **xterm <node_name1> <node_name2>** (etc.)
- check networking configuration, by typing Linux iproute2 commands

  e.g., **<node_name> ip address**

... other useful operation (enter **?** to see the complete list)

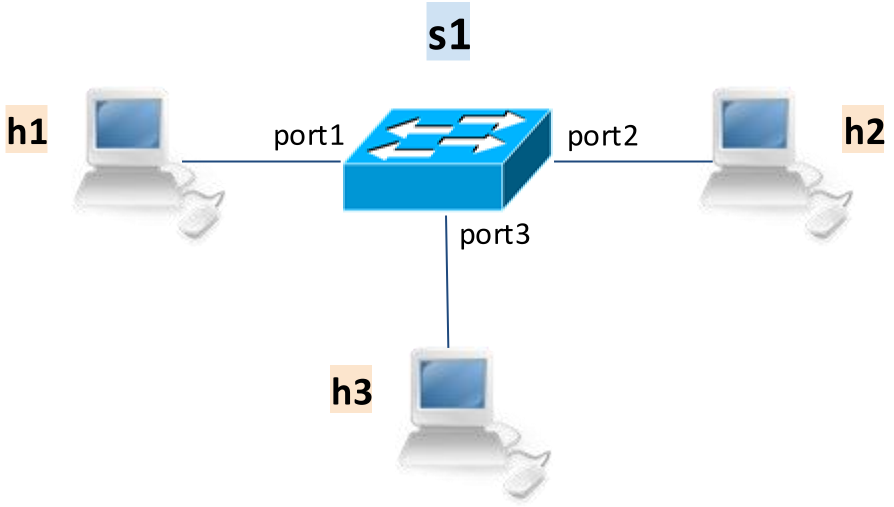
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Predefined parametrized topologies

Use the option **--topo** with the command **sudo mn** to start Mininet with a different built-in network topology, passing parameters to change topology size and type.

For instance:

- **--topo single,3**



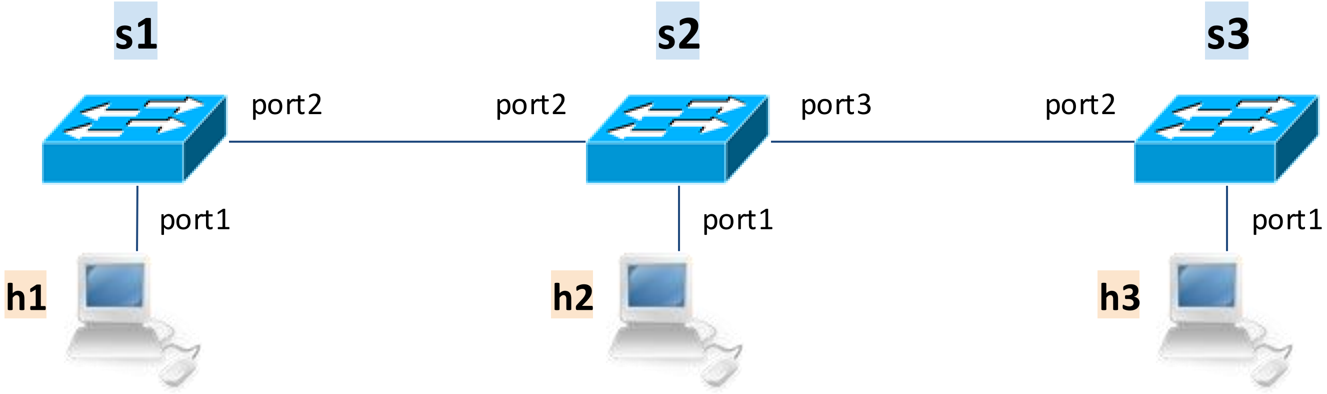
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Predefined parametrized topologies

Use the option **--topo** with the command **sudo mn** to start Mininet with a different built-in network topology, passing parameters to change topology size and type.

For instance:

- **--topo linear,3**



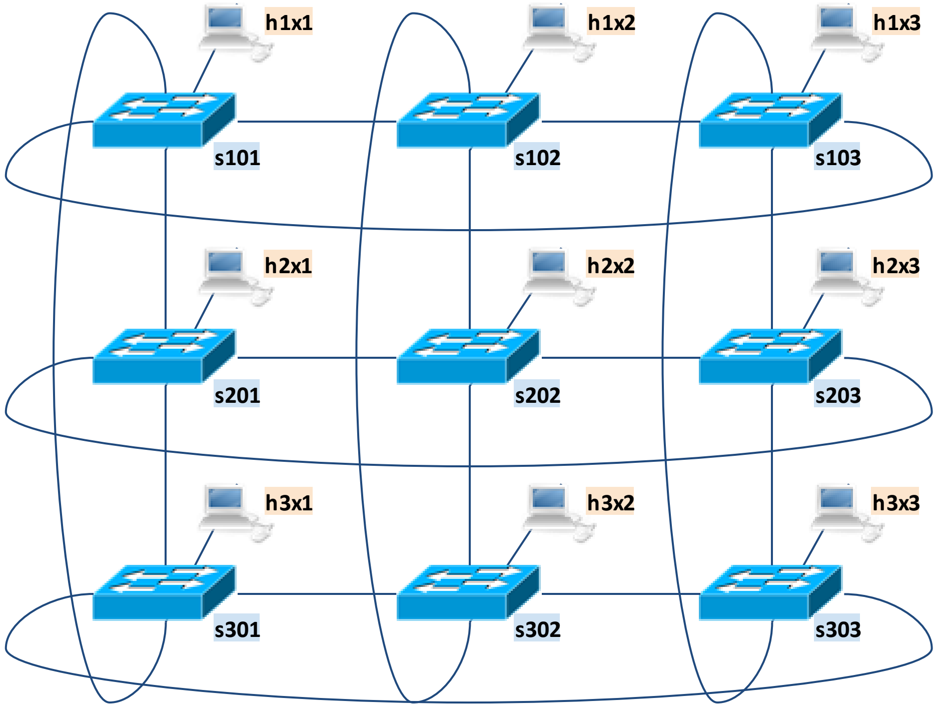
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Predefined parametrized topologies

Use the option **--topo** with the command **sudo mn** to start Mininet with a different built-in network topology, passing parameters to change topology size and type.

For instance:

- **--topo torus,3,3**



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Custom topologies

What if I want to build my own topology?

Describing a custom topology is easy and intuitive with the **Python API**, which allows us to:

- add Hosts
- add Switches
- add Link connection between hosts and switches, and between switches

Run your custom topology by typing:

```
sudo mn --custom <python_code_topology.py> --topo <symbolic_topology_name>
```

Configuring Mininet: other options

Create an emulated network topology and enter Mininet CLI:

sudo mn <options>

- **--controller <type>** : controller type and relevant parameters

  e.g., **--controller remote** to use an external controller on 127.0.0.0:6653

  e.g., **--controller remote,ip=10.0.1.10,port=6633** (self-explanatory)
- **--switch <type>** : software switch type and relevant parameters

  e.g., **--switch ovsk** for OvS

  e.g., **--switch ovsk,protocols=OpenFlow13** for OvS using OpenFlow v1.3
- **--mac** : assign predictable MAC addresses to emulated hosts (sequentially)
- **--arp** : fill the ARP table of emulated hosts with all MAC addresses
- **--link tc**: to manage traffic control

  e.g., **--link tc,bw=<bandwidth>,delay=<delay_in_millisecond>** to assign a given bandwidth and delay to links

Practical session 1: default topology

Run the Mininet default topology with the default controller and play with the Mininet CLI:

1) Run Mininet default topology: **sudo mn**
2) List all available operations: **?**
3) List network nodes: **nodes**
4) Show network topology: **net**
5) Show link connections: **links**
6) Show switch(es) port configuration: **ports**

Practical session 1: node connectivity

1) Check *h1* network interface configuration: **h1 ip address**
2) Check *h1* routing table configuration: **h1 ip route**
3) Open a Terminal on *h1* node: **xterm h1**
   - **Note**: xterm font size may be too small; in such a case, you can perform the following steps:
     1. In your home directory, create **.Xresources** file
     2. Write the following lines:
        ```
        xterm*faceName: DejaVu Sans Mono Book
        xterm*faceSize: 16
        ```
     3. Execute command **xrdb -merge .Xresources**
4) Test connectivity between *h1* and *h2*. Two ways are possible:
   - Via node Terminal: **ping <IP_node>**
   - Via Mininet CLI: **h1 ping h2**

Practical session 2: custom topology

Run the custom topology "*topo-2sw-2host.py*"* with the default controller and play with the Mininet CLI:

- Run Mininet custom topology:

```
sudo mn --custom topo-2sw-2host.py --topo mytopo --link tc,bw=10
```

- List network nodes: **nodes**
- Show network topology: **net**
- Show link connections: **links**
- Show switch(es) port configuration: **ports**

*The topology is available on Virtuale

Practical session 2: node connectivity

1) Check *h1* network interface configuration: **h1 ip address**
2) Check *h1* routing table configuration: **h1 ip route**
3) Open a Terminal on *h1* node: **xterm h1**
4) Test connectivity between *h1* and *h2*, via the node Terminal or the Mininet CLI

Practical session 2: OpenFlow

What about OpenFlow flow entries installed on OvS switches?

1) Check OpenFlow (OF) rules with command:
   sudo ovs-ofctl dump-flows <switch_name>
   - **Note**: type **ovs-ofctl --help** or check the *man* page to see the complete list of commands
2) Make a ping between two nodes and check OF rules again. What happened?

SDN controller

What if I want to build my own controller?

It is more complicated, you need to know the programming language (e.g., Java, Python) and

- use an API provided by the SDN controller (we will see Ryu)
- use OF protocol to manage the switches in your network

- Run custom Ryu controller by typing:

```
ryu-manager <python_code_cont.py>
```

- Run custom topology with custom controller by typing:

```
sudo mn --custom <python_code_topology.py> --topo
<symbolic_topology_name> --controller remote --switch ovsk
```

Practical session 3: Ryu controller

Run Mininet custom topology with custom Ryu controller "*simple_switch_13.py*" and play with OvS switches:

1) Run Ryu controller: **ryu-manager simple_switch_13.py**
2) Run Mininet custom topology: **sudo mn --custom topo-2sw-2host.py --topo mytopo --controller remote --switch ovsk**
3) List OpenFlow rules on one of the switches: e.g., **sudo ovs-ofctl dump-flows s2**
4) Test connectivity between *h1* and *h2,* via the node Terminal or the Mininet CLI
5) List the flow entries on the switch again: has anything changed?

OpenFlow messages

What about OpenFlow messages?

- **Controller-to-switch**: used to directly manage or inspect the state of the switch
- **Asynchronous**: initiated by the switch, used to update the controller of network events/changes to the state of the switch
- **Symmetric**: initiated by either the switch or the controller and are sent without solicitation

*https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf
*https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

Practical session 4: OF messages

Run the Mininet custom topology with custom controller again and use Wireshark to analyze the messages exchanged between the controller and the switches.

<u>What kind of traffic can we see?</u>

- **Control plane traffic**: management traffic
- **Data plane traffic**: data traffic

Control plane is separated from the data plane.

Practical session 4: OF messages

Be aware that:

- Controller runs locally (*loopback* interface) and listens on **TCP port 6653 or 6633**
- Switches and controller are in the so-called *global namespace*
- We need to intercept both control and data plane traffic

Practical session 4: OF messages

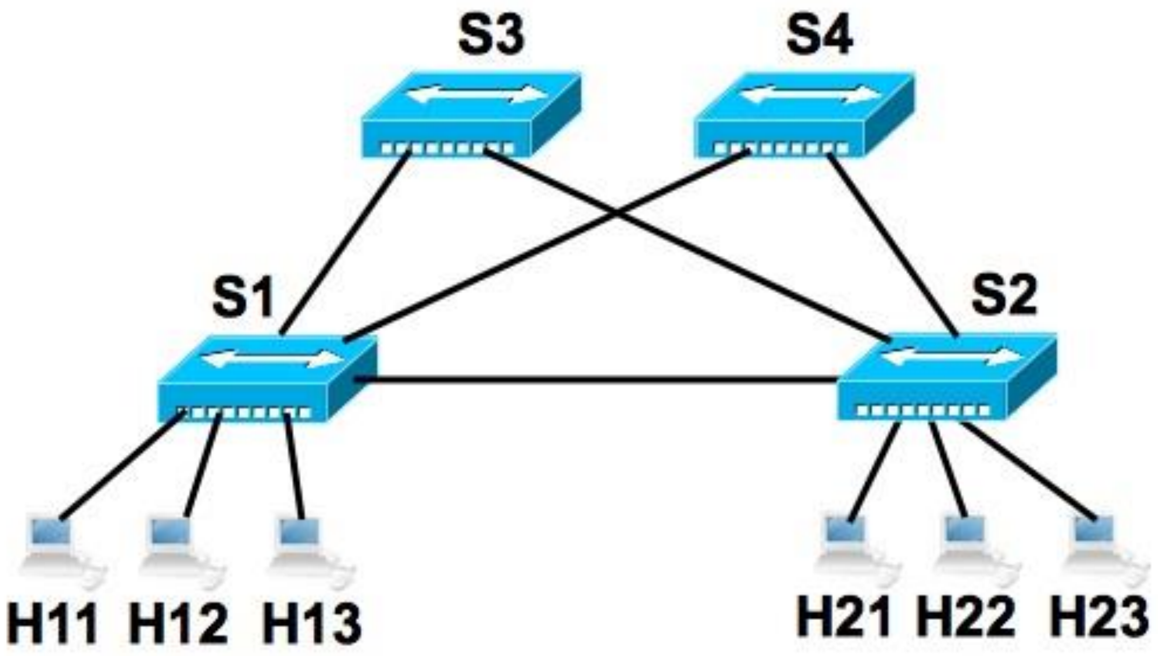
Start 2 Wireshark sessions to capture control plane and data plane traffic.

Investigate control plane traffic:

- OFPT_HELLO
- OFPT_FEATURE_REQUEST/OFPT_FEATURE_REPLY
- OFPT_SET_CONFIG
- OFPT_PORT_STATUS
- OFPT_PACKET_IN/OUT
- OFPT_ECHO

Exercise

Try to implement the following topology starting from the *topo-2sw-2host.py* script:


S3
S4
S1
S2
H11
H12
H13
H21
H22
H23