# Networks and Cloud Infrastructures

Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione DIETI
Laurea Magistrale in Ingegneria Informatica

Prof. Giorgio Ventre

## The Mininet network emulator

# Mininet

- Mininet is a lightweight virtualization/container based emulator that allows to reproduce virtual networks with arbitrary topologies
  - modest hardware requirements, fast startup, hundreds of nodes
  - command line tool, CLI, simple Python API

- Mininet enables SDN development on any laptop or PC, and SDN designs can move seamlessly between Mininet (allowing inexpensive and streamlined development), and the real hardware running at line rate in live deployments

- Python scripts can be used to orchestrate an experiment
  - Network topology definition
  - Events to be triggered in network nodes (e.g. execution of a program)

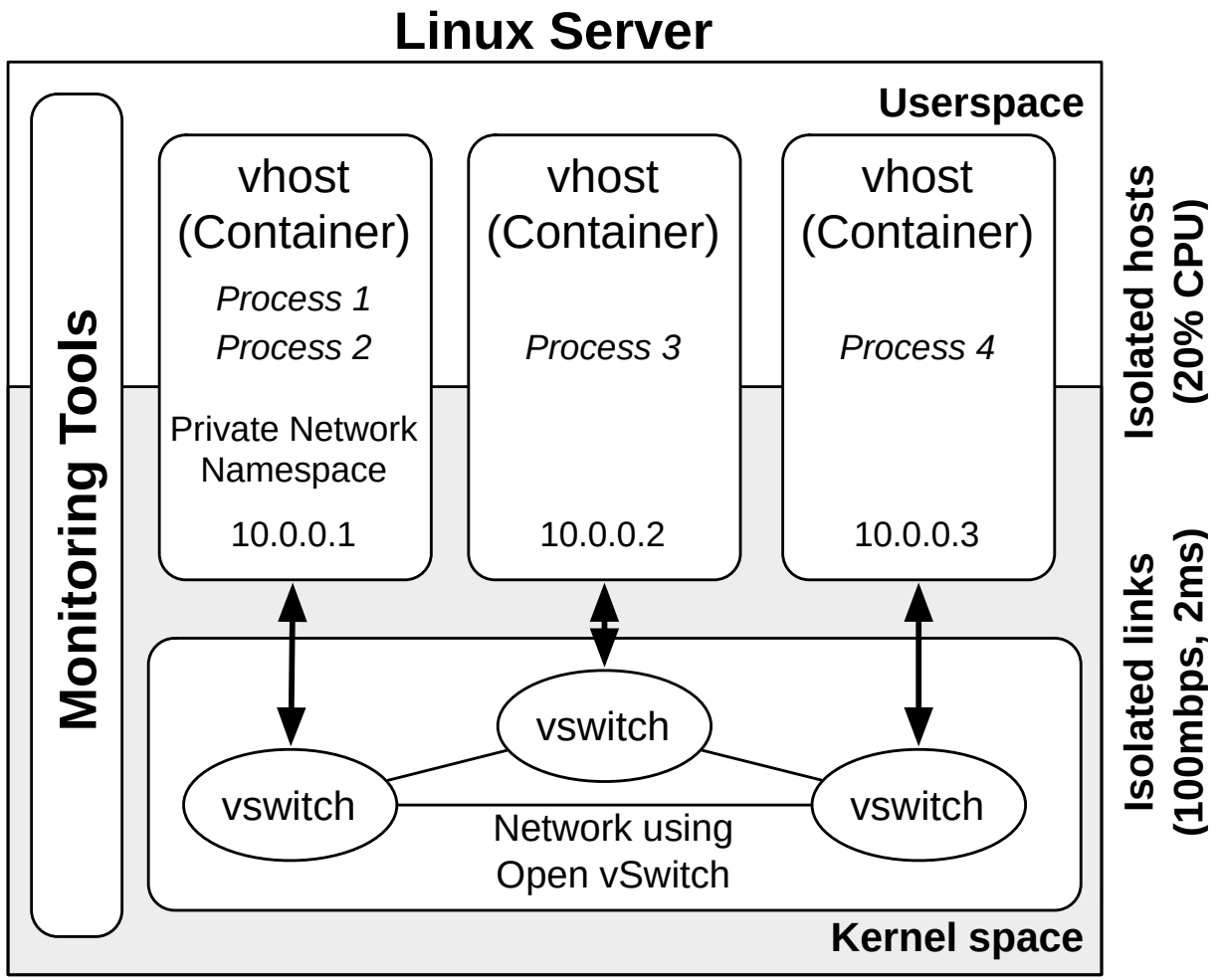
- Mininet VM installation: the easiest way of installing Mininet
  1. Download and install on your PC one of the available hypervisors (e.g., VirtualBox, or KVM)
  2. Download the Mininet pre-installed VM image
  3. Import VM image into selected hypervisor

# Some Mininet and OpenFlow tutorials

- http://mininet.org/walkthrough/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- https://github.com/mininet/mininet/wiki/Videos
- https://github.com/mininet/mininet/wiki/Documentation
- Floodlight Controller
  https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Getting+Started
- RYU Controller Tutorial
  http://sdnhub.org/tutorials/ryu/

# Mininet: emulated hosts and switches

- Mininet combines lightweight virtualization (*containers*) with software switches to emulate networks in a Linux-based system
- A Mininet network consists of:
  - ISOLATED HOSTS: a group of user-level processes moved into a network namespace that provides exclusive ownership of interfaces, ports and routing tables
  - EMULATED LINKS: Linux Traffic Control (tc) enforces the data rate of each link to shape traffic to a configured rate - each emulated host has its own virtual Ethernet interface(s)
  - EMULATED SWITCHES: the default Linux Bridge or the Open vSwitch running in kernel mode are used to switch packets across interfaces
- Emulated hosts share the same Linux kernel and the file system of the host in which they run



# Prerequisites

- Install Virualbox, VMWare or other, and create an Ubuntu 22.04 VM
- Install the following packages on the VM using the provided command line:

```
$ sudo apt install git
$ sudo apt install python3-pip
$ sudo pip3 install pandas
$ pip install ryu
$ sudo apt install d-itg
$ sudo apt install nload
```

- Clone Mininet Repository and Install:

```
$ git clone https://github.com/mininet/mininet
$ cd mininet
$ git tag
$ git checkout -b mininet-2.3.0 2.3.0
$ cd ..
$ sudo PYTHON=python3 mininet/util/install.sh -nv
$ sudo mn --switch ovsbr --test pingall # Test Mininet installation
```

# Mininet: network topologies

- Start mininet with a minimal topology (1 switch and 2 connected hosts)

```
$ sudo mn
```

is equivalent to:

```
$ sudo mn --topo minimal
```

- Start mininet with 1 switch and n connected hosts

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

- Start mininet with a linear topology (n switches in a row and 1 host connected to each switch)

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

- Start mininet with a tree topology with depth n and fanout m

```
$ sudo mn --topo tree,depth=n,fanout=m
```

- Start mininet with a custom topology *mytopo* defined in a Python script (mytopo.py)

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

```
class MyTopo( Topo ):
   def build( self, ...):
def myTest( net ):
...
topos = { 'mytopo': MyTopo }
tests = { 'mytest': myTest }
```

- Stop mininet processes

```
$ sudo mn -c
```

# Mininet: controller option

- Start a minimal topology with the default internal controller

```
$ sudo mn
```

- Start a minimal topology without a controller

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

- Start a minimal topology using the reference OpenFlow controller

```
$ sudo mn --controller ref
```

- Start a minimal topology using an external controller (e.g. Ryu, Floodlight, etc.)

```
$ sudo mn --controller remote,ip=[IP_ADDDR],port=[listening_port]
```

- Start a minimal topology using an external controller on 127.0.0.1:6653

```
$ sudo mn --controller remote
```

# Mininet: other options

- Start mininet by assigning MAC addresses sequentially to hosts

```
$ sudo mn --mac
```

  - E.g. host h1 gets MAC 00:00:00:00:00:01, etc.

- Start mininet and show an xterm for every host and switch

```
$ sudo mn -x
```

- Start mininet and run test function iperf when the whole node is up

```
$ sudo mn --test iperf
```

or

```
# iperf -s
```

```
# iperf -c ip_address
```

- Start a minimal topology and exits: computes the time to bring the network up

```
$ sudo mn --test none
```

- Use Open vSwitch for network nodes

```
$ sudo mn --switch ovsk
```

- Force Open vSwitch to use OpenFlow protocol version OpenFlow1.3

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

# Mininet: other options

- Start mininet by assigning specific parameters to all links

```
$ sudo mn --link tc,bw=[bandwidth],delay=[delay_in_millisecond]
```

  - Assigns a given bandwidth and delay to links
    - Example (10.00Mbit 10ms delay):

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

      - Open a terminal on both hosts and run iperf

```
mininet> xterm h1
mininet> xterm h2
```

# Mininet: common commands from the CLI

- Display nodes

```
mininet> nodes
```

- Display links

```
mininet> net
```

- Dump information about all nodes

```
mininet> dump
```

- Execute a method through invoking mininet API

```
mininet> py [mininet_name_space].[method]
```

# Mininet: interaction 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
```

- Verify the connectivity by pinging from host h1 to host h2

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

- Verify the connectivity between all pairs of hosts

```
mininet> pingall
```

- Measure end-to-end bandwidth between two hosts with iperf
  - Server endpoint

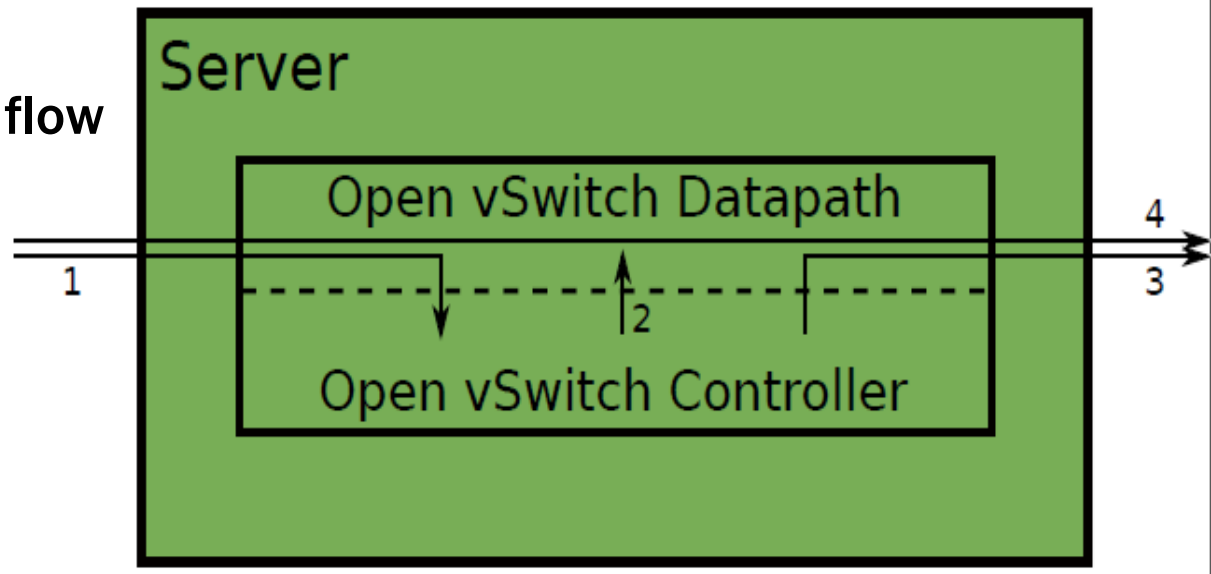
```
mininet> iperf -s -u -p [port_num] &
```

  - Client endpoint

```
mininet> iperf -c [IP] -u -t [duration] -b [bandwidth] -p [port_num]
```

# Open vSwitch

- Mininet is often used to instantiate networks of Open vSwitch switches
- Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license
- Open vSwitch design choices:
  - Flexible Controller computation in User space
  - Fast Datapath packet handling in Kernel space
- The 1st packet of a flow is sent to the controller
- The controller programs the datapath's actions for a flow
  - Usually one, but may be a list
- Actions include:
  - Forward to a port or ports
  - Mirror
  - Encapsulate and forward to controller
  - Drop
- And it returns the packet to the datapath
  - Subsequent packets are handled directly by the datapath
- An Open vSwitch can be remotely controlled by an Openflow SDN controller



# Open vSwitch commands: ovs-vsctl

Ovs:-vsctl:utility for querying and configuring ovs-vswitchd

```
$ sudo ovs-vsctl --help #
usage: ovs-vsctl [OPTIONS] COMMAND [ARG...]

Open vSwitch commands:
  init                        initialize database, if not yet initialized
  show                        print overview of database contents
  emer-reset                  reset configuration to clean state

Bridge commands:
  add-br BRIDGE               create a new bridge named BRIDGE
  add-br BRIDGE PARENT VLAN   create new fake BRIDGE in PARENT on VLAN
  del-br BRIDGE               delete BRIDGE and all of its ports
  list-br                     print the names of all the bridges
...
Port commands (a bond is considered to be a single port):
  list-ports BRIDGE           print the names of all the ports on BRIDGE
  add-port BRIDGE PORT        add network device PORT to BRIDGE
...
  del-port [BRIDGE] PORT      delete PORT (which may be bonded) from BRIDGE
```

# Open vSwitch commands: ovs-vsctl (cont.)

```
...
Interface commands (a bond consists of multiple interfaces):
  list-ifaces BRIDGE            print the names of all interfaces on BRIDGE
  iface-to-br IFACE             print name of bridge that contains IFACE

Controller commands:
  get-controller BRIDGE       print the controllers for BRIDGE
  del-controller BRIDGE       delete the controllers for BRIDGE
  [--inactivity-probe=MSECS]
  set-controller BRIDGE TARGET...  set the controllers for BRIDGE
  get-fail-mode BRIDGE        print the fail-mode for BRIDGE
  del-fail-mode BRIDGE        delete the fail-mode for BRIDGE
  set-fail-mode BRIDGE MODE   set the fail-mode for BRIDGE to MODE

Manager commands:
  get-manager                 print the managers
  del-manager                 delete the managers

...
```

# Open vSwitch commands: ovs-ofctl

Ovs:-ofctl: command line tool for monitoring and administering OpenFlow switches

```
$ sudo ovs-ofctl --help
ovs-ofctl: OpenFlow switch management utility
usage: ovs-ofctl [OPTIONS] COMMAND [ARG...]

For OpenFlow switches:
   show SWITCH                  show OpenFlow information
   dump-desc SWITCH             print switch description
   dump-tables SWITCH           print table stats
...
   dump-ports SWITCH [PORT]     print port statistics
   dump-ports-desc SWITCH       print port descriptions
   dump-flows SWITCH            print all flow entries
   dump-flows SWITCH FLOW       print matching FLOWs
   dump-aggregate SWITCH        print aggregate flow statistics
   dump-aggregate SWITCH FLOW   print aggregate stats for FLOWs
   add-flow SWITCH FLOW         add flow described by FLOW
   add-flows SWITCH FILE        add flows from FILE
   mod-flows SWITCH FLOW        modify actions of matching FLOWs
   del-flows SWITCH [FLOW]      delete matching FLOWs
```

# Experiment #1: internal controller

- Start Mininet and create a simple topology with 1 switch and 2 hosts

```
$ sudo mn --topo single,2 --mac --switch ovsk
```

- **By default, Mininet creates an internal controller that implements a simple learning switch functionality**
- Hosts are named h1 and h2
- Open xterms for hosts h1 and h2 from Mininet prompt

```
mininet> xterm h1 h2
```

- Open wireshark from xterm on h1

```
h1# sudo wireshark
```

- Let h1 ping h2 from Mininet prompt

```
mininet> h1 ping h2
```

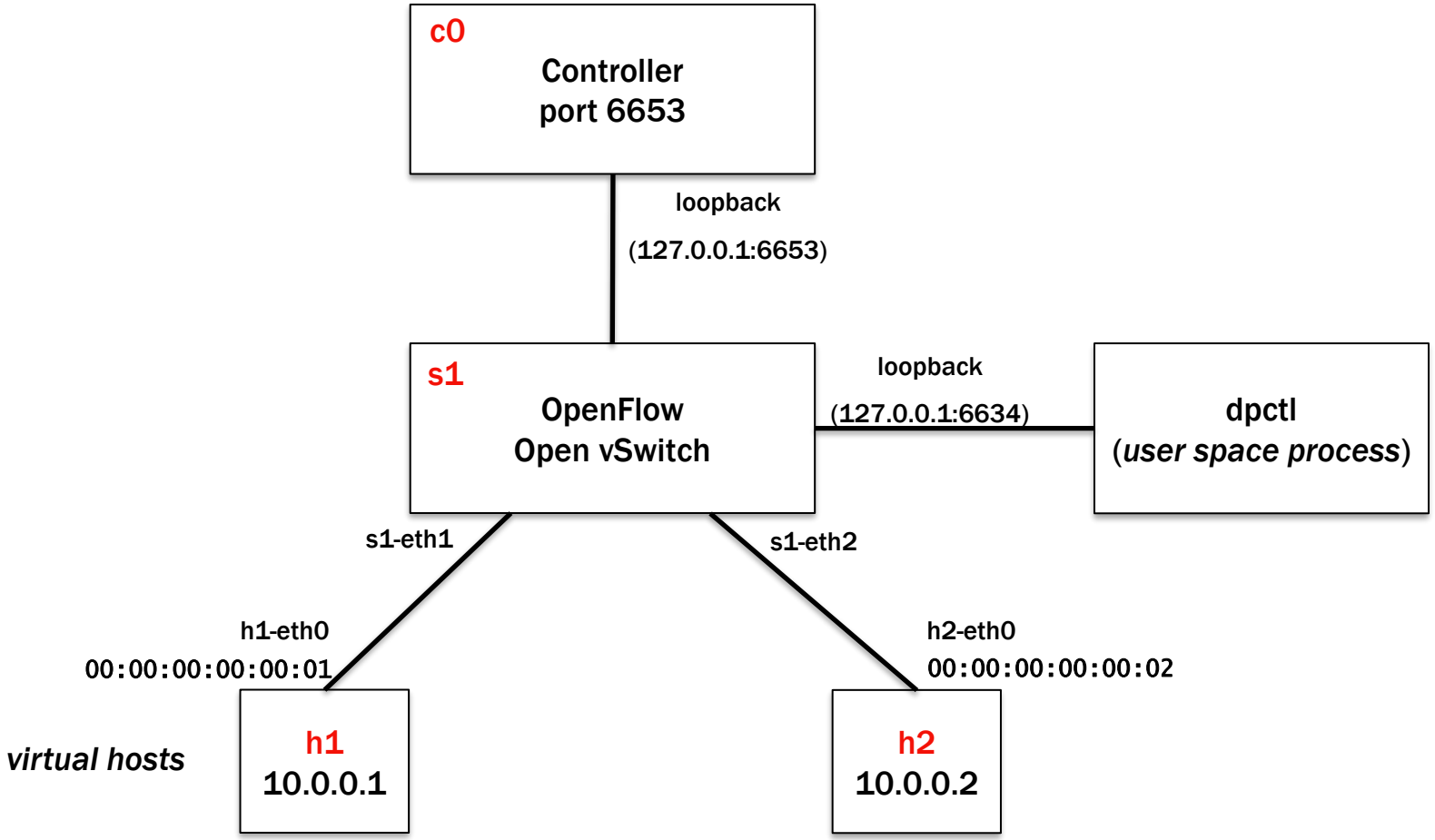
- Exec a simple web server (listening on port 80) from xterm on h2

```
h2# python3 -m http.server 80
```

- Let h1 ping h2 from Mininet prompt

```
mininet> h1 wget h2
```

# Experiment #1: components



# Open vSwitch CLI commands (1)

- `sudo ovs-vsctl show`

Lists all instances of Open vSwitch (e.g. s1, s2, ...)

```
8786cd90-73f6-43c9-bafe-72d80af2a23a
    Bridge s1
        Controller "ptcp:6654"
        fail_mode: secure
        Port s1
            Interface s1
                type: internal
        Port s1-eth1
            Interface s1-eth1
        Port s1-eth2
            Interface s1-eth2
    ovs_version: "2.13.0"
```

- `sudo ovs-ofctl show s1`

Lists all ports of an Open vSwitch switch

```
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(s1-eth1): addr:2a:de:31:4d:51:b9
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:7e:76:4f:50:13:c6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:42:c4:0e:3a:9e:45
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

# Open vSwitch CLI commands (2)

- `sudo ovs-ofctl dump-flows s1 [-O OpenFlow13]`

Lists all entries in a switch Flow Tables; by default talks OpenFlow 1.0

- `sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2 [-O OpenFlow13]`

Adds a flow entry into switch s1

# Experiment #2: no controller

```
$ sudo mn --topo single,2 --mac --switch ovsk --controller none
```

```
$ sudo ovs-ofctl show s1 -O OpenFlow13
$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

```
mininet> h1 ping h2
```

All ports of switch shown, but no flows installed.
Ping fails because ARP cannot go through

```
$ sudo ovs-ofctl add-flow s1 -O OpenFlow13
in_port=1,actions=output:2
$ sudo ovs-ofctl add-flow s1 -O OpenFlow13
in_port=2,actions=output:1
mininet> h1 ping h2
```

```
mininet> h1 ping h2
```

Ping works now!

# OpenFlow rules set by an external controller

- For the topology created with

```
$ sudo mn --topo single,2 --mac --switch ovsk --controller remote,port=6653
```

- if the OpenFlow controller is running (e,g, Floodlight with the Forwarding module enabled)

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

- produces a similar output (actual flow rules depend on the controller logic):

```
cookie=0x20000000000000, duration=1.343s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, idle_age=1,
priority=1,arp,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
cookie=0x20000000000000, duration=6.361s, table=0, n_packets=1, n_bytes=42, idle_timeout=5, idle_age=1,
priority=1,arp,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x20000000000000, duration=6.356s, table=0, n_packets=6, n_bytes=588, idle_timeout=5, idle_age=0,
priority=1,ip,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2
actions=output:2
cookie=0x20000000000000, duration=6.354s, table=0, n_packets=6, n_bytes=588, idle_timeout=5,
idle_age=0,priority=1,ip,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,
nw_dst=10.0.0.1 actions=output:1
```

# Running Mininet scenarios described in Python scripts

- Mininet installation makes all Mininet classes available to the Python interpreter so that they can be instantiated from a regular Python script
- Instead of running the `mn` command and issuing commands from the CLI, a Mininet scenario may be executed by passing a Python script to the regular Python interpreter
- The script needs super-user rights, hence it must be run with sudo

```
$ sudo python test1.py
```

- The Python script needs to import all the relevant classes, create a custom topology (as a class derived from the mininet.topo.Topo base class), instantiate a Mininet network object and perform all the required actions
- Before terminating the script, some cleanup methods need to be invoked
- Before presenting some sample scripts, a first look at the hierarchy of classes created by Mininet is necessary
- The Python script may build the emulated scenarios by working at different semantic levels

# Node and link classes

- mininet.node.Node
  - A virtual network node, which is simply in a network namespace
  - From the generic Node class three classes are derived: Host, Switch and Controller
- mininet.link.Link
  - A basic link, which is represented as a pair of nodes

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch('s1', inNamespace=False)
c0 = Controller('c0', inNamespace=False)
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.0.0.1/8' )
h2.setIP( '10.0.0.2/8' )
```

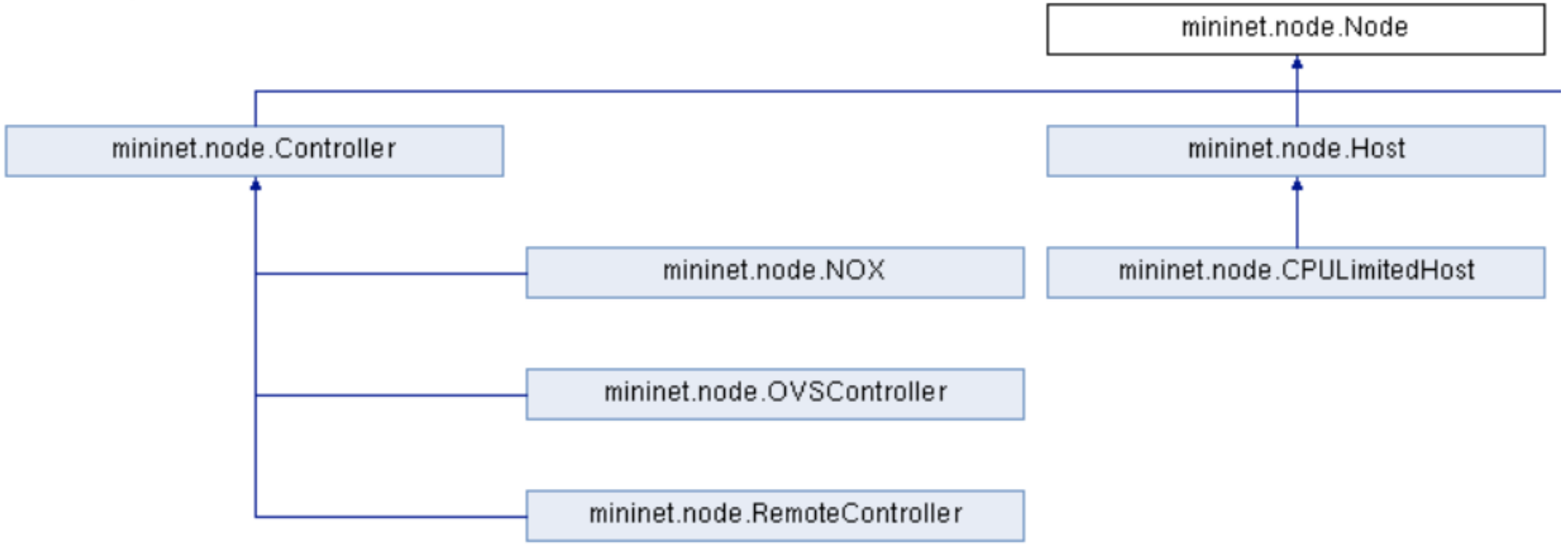
| Class | Method | Description |
|---|---|---|
| Node | MAC/setMAC | Return/Assign MAC address of a node or specific interface |
| | IP/setIP | Return/Assign IP address of a node or specific interface |
| | cmd | Send a command, wait for output, and return it |
| | terminate | Send kill signal to Node and clean up after it |
| Link | Link | Create a link to another node, make two new interfaces |

```
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

# Node class and subclasses (1/2)

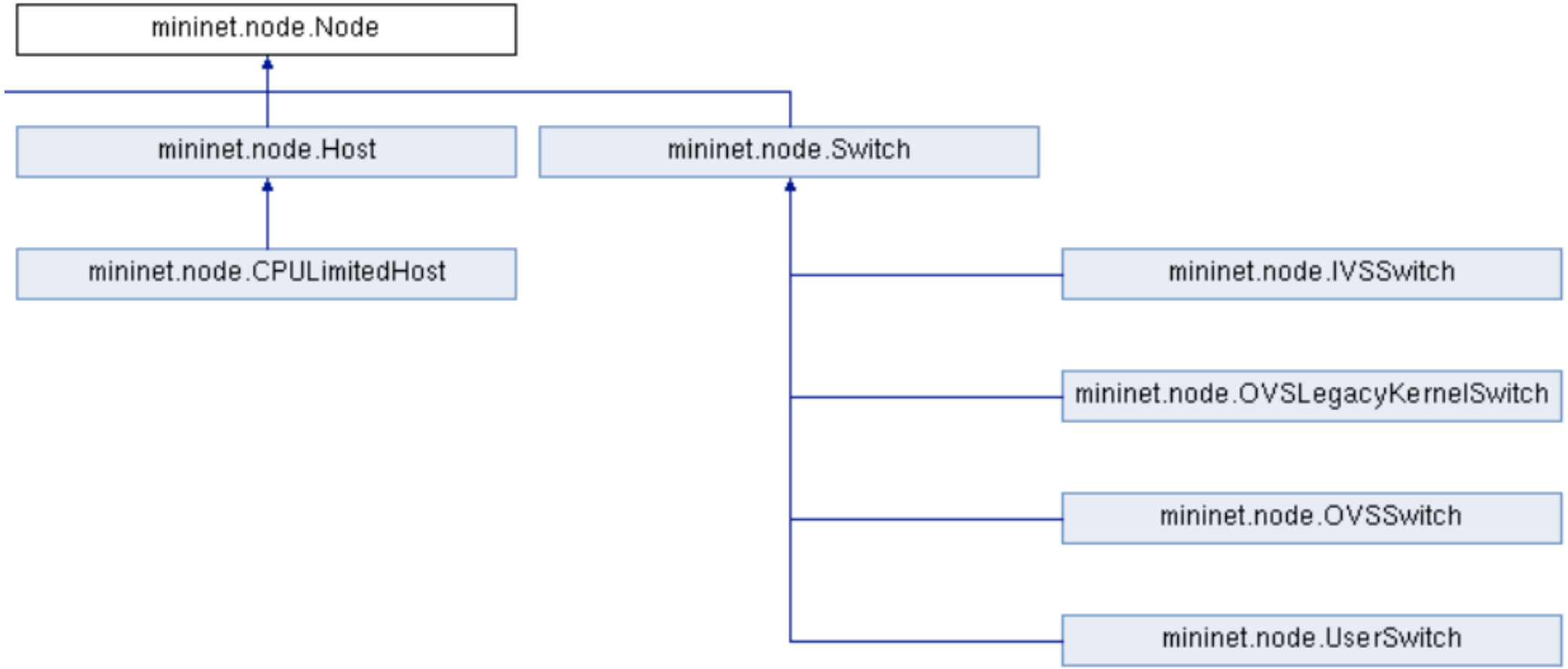
- Node generic class
- 3 subclasses: **Controller, Host**, Switch

Inheritance diagram for mininet.node.Node:



# Node class and subclasses (2/2)

- Node generic class
- 3 subclasses: Controller, **Host, Switch**



# Mininet class

- mininet.net.Mininet
  - Network emulation with hosts spawned in network namespaces

| Class | Method | Description |
|---|---|---|
| Mininet | addHost | Add a host to network |
| | addSwitch | Add a switch to network |
| | addLink | Link two nodes into together |
| | addController | Add a controller to network |
| | getNodeByName | Return node(s) with given name(s) |
| | start | Start controller and switches |
| | stop | Stop the controller, switches and hosts |
| | ping | Ping between all specified hosts and return all data |

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
```

```
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

# Topo class

- mininet.topo.Topo

| Class | Method | Description |
|---|---|---|
| Topo | Methods similar to net | E.g., addHost, addSwitch, addLink, |
| | addNode | Add node to graph |
| | addPort | Generate port mapping for new edge |
| | switches | Return all switches |
| | Hosts/nodes/switches/links | Return all hosts/nodes/switches/links |
| | isSwitch | Return true if node is a switch, return false otherwise |

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1):
        hosts = [ self.addHost( 'h%d' % i )
            for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )
```

```
net = Mininet( topo=SingleSwitchTopo(3) )
net.start()
CLI( net )
net.stop()
```

# First Mininet script (1/2)

```
#!/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 build(self, n=2):
        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)
#...
```

Custom topology class

# First Mininet script (2/2)

```
# ...
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()
```

Custom test function

Script startup

# Second Mininet script (1/2)

```
#!/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, 2% loss, 1000 packet queue
            self.addLink(host, switch, bw=10, delay='5ms',
                         loss=2, max_queue_size=1000,
                         use_htb=True)
#...
```

Custom topology class

# Second Mininet script (2/2)

```
# ...
def perfTest():
    "Create network and run a simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    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()
```

Custom test function

Script startup

# Third Mininet script (1/3)

```
#!/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 LinearTopo(Topo):
    "Linear topology of n switches, with one host per switch."
    def build(self, n=2):
        lastSwitch = None
        for i in range(n):
            # Each host gets 50%/n of system CPU
            host = self.addHost('h%s' % (i+1), cpu=.5/n)
            switch = self.addSwitch('s%s' % (i+1))
            # 10 Mbps, 5ms delay, 2% loss, 1000 packet queue
            self.addLink(host, switch, bw=10, delay='5ms',
                         loss=2, max_queue_size=1000, use_htb=True)
            #...
```

Custom topology class

# Third Mininet script (2/3)

```
            #...
            if lastSwitch:
                self.addLink(switch, lastSwitch, bw=10, delay='5ms',
                             loss=2, max_queue_size=1000, use_htb=True)
            lastSwitch = switch

def perfTest():
    "Create network and run a simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    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()
#...
```

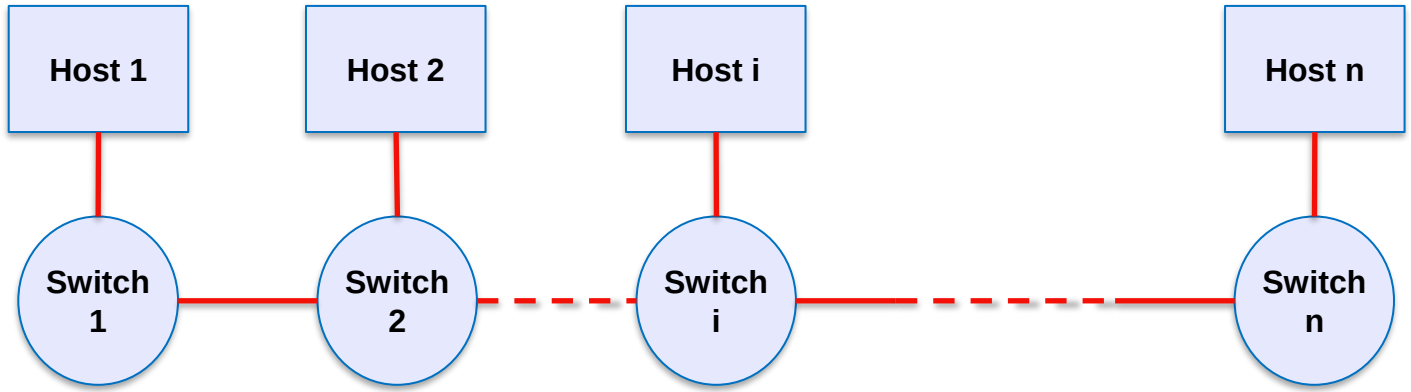
Custom test function

# Third Mininet script (3/3)

```
#...
if __name__ == '__main__':
    setLogLevel('info')
    perfTest()
```

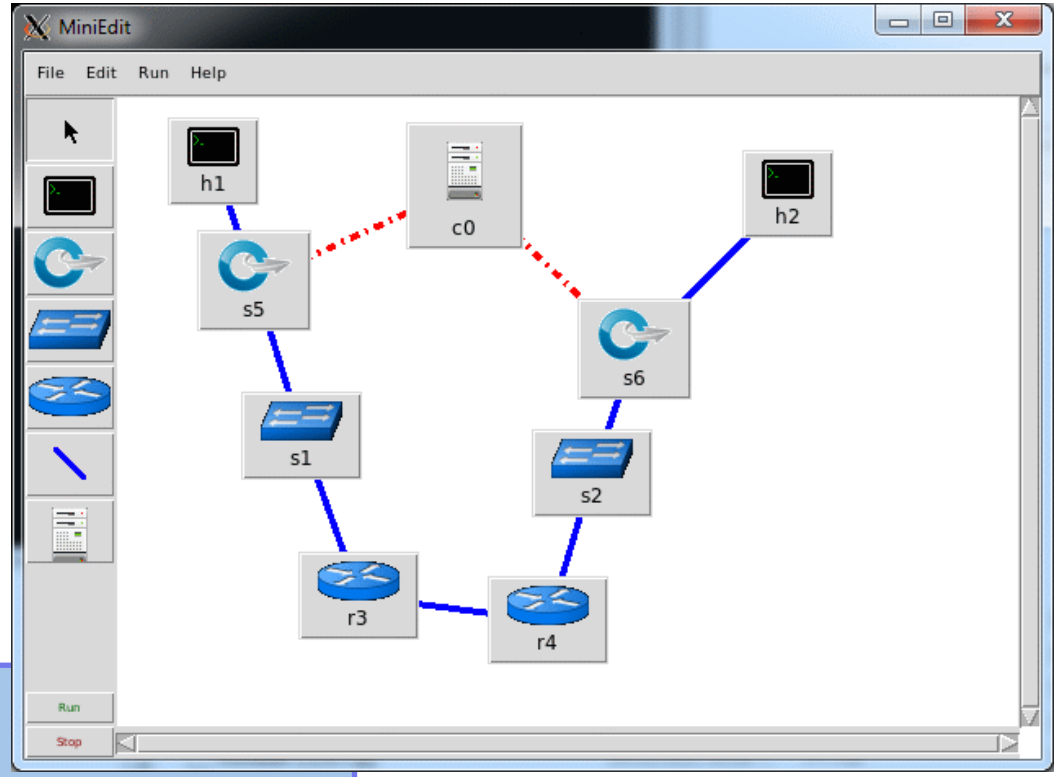
Script startup

**Linear topology**



# Mininet graphical tools

- MiniEdit
  - A GUI application which eases the Mininet topology generation
  - Either save the topology or export as a Mininet python script




```
$ sudo apt install python-tk
$ sudo apt-get install openvswitch-testcontroller
$ sudo ln /usr/bin/ovs-testcontroller /usr/bin/controller
$ sudo python2 ~/mininet/examples/miniedit.py
# create the topology
# export the topology as a .py file
$ sudo python3 my_topology.py
```

# Remote controller with Ryu

- Ryu is a component-based software defined networking framework.
- Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications.
- We use Ryu for running an example of remote controller.

```
$ cd /home/mininet/.local/lib/python3.10/site-packages/ryu/app
$ nano wsgi.py
# Comment the AlreadyHandledResponse class in wsgi.py
$ ryu-manager simple_switch_13.py
```

```
$ sudo mn --controller remote
```

```
$ sudo wireshark
```

# Controller logic: Simple Switch 13

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)

    def switch_features_handler(self, ev):

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
```

# Controller logic: Simple Switch 13

```
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly.  The bug has been fixed in OVS v2.1.0.
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)
```

# Controller logic: Simple Switch 13

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
        return
    dst = eth.dst
    src = eth.src
    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})
    self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

    # learn a mac address to avoid FLOOD next time.
    self.mac_to_port[dpid][src] = in_port

    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(out_port)]
```

```
    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
        # verify if we have a valid buffer_id, if yes avoid to send both
        # flow_mod & packet_out
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(datapath, 1, match, actions)
    data = None

    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data
    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

# Example: topology defined in Python script and remote controller

```
#!/usr/bin/python
import threading
import random
import time
from mininet.log import setLogLevel, info
from mininet.topo import Topo
from mininet.net import Mininet, CLI
from mininet.node import OVSKernelSwitch, Host
from mininet.link import TCLink, Link
from mininet.node import RemoteController #Controller

...
```

# Example: topology defined in Python script and remote controller

```
class Environment(object):
    def __init__(self):
        "Create a network."
        self.net = Mininet(controller=RemoteController, link=TCLink)
        info("*** Starting controller\n")
        c1 = self.net.addController( 'c1', controller=RemoteController) #Controller
        c1.start()
        info("*** Adding hosts and switches\n")
        self.h1 = self.net.addHost('h1', mac ='00:00:00:00:00:01', ip= '10.0.0.1')
        self.h2 = self.net.addHost('h2', mac ='00:00:00:00:00:02', ip= '10.0.0.2')
        self.cpe1 = self.net.addSwitch('s1', cls=OVSKernelSwitch)
        self.cpe2 = self.net.addSwitch('s2', cls=OVSKernelSwitch)
        self.core1 = self.net.addSwitch('s3', cls=OVSKernelSwitch)
        info("*** Adding links\n")
        self.net.addLink(self.h1, self.cpe1, bw=6, delay='0.0025ms')
        self.path1 = self.net.addLink(self.cpe1, self.core1, bw=3, delay='25ms')
        self.net.addLink(self.core1, self.cpe2, bw=3, delay='25ms')
        self.net.addLink(self.cpe2, self.h2, bw=6, delay='0.0025ms')
        info("*** Starting network\n")
        self.net.build()
        self.net.start()
…
```

# Example: topology defined in Python script and remote controller

```
...
if __name__ == '__main__':

    setLogLevel('info')
    info('starting the environment\n')
    env = Environment()

    info("*** Running CLI\n")
    CLI(env.net)
```

# Running the example

```
$ ryu-manager simple_switch_13.py
```

```
$ sudo python3 my_topo.py
```

```
$ sudo wireshark
```

# Capture OpenFlow packets with wireshark

- If wireshark is not able to decode OF packets, reinstall a newer version

```
sudo apt-get remove wireshark
sudo apt-get -y install libgtk-3-dev libqt4-dev flex bison
wget https://www.wireshark.org/download/src/all-versions/wireshark-1.12.3.tar.bz2
tar xvfj wireshark-1.12.3.tar.bz2
cd wireshark-1.12.3
./configure
make -j4
sudo make install
sudo echo "/usr/local/lib" >> /etc/ld.so.conf
sudo ldconfig
```

- If the controller is running locally, capture packets on `lo` interface (*loopback*) on port TCP/6653 (filter = `tcp port 6653`)