Cloud and Datacenter Networking

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

Prof. Roberto Canonico

Learning OpenFlow with Mininet

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

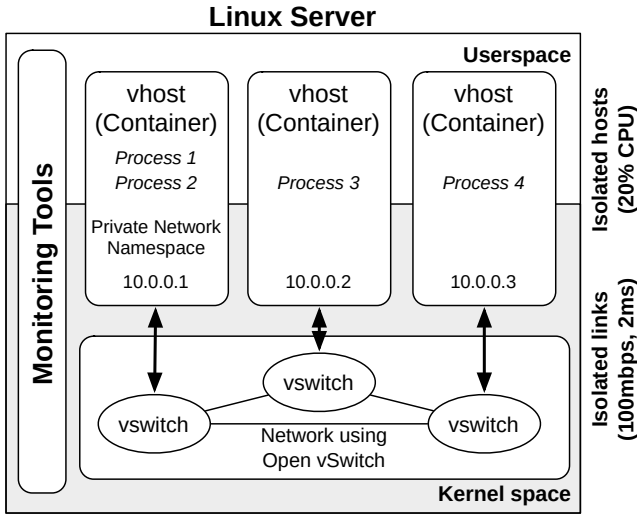
- Mininet enables SDN development on any laptop or PC, and SDN designs can move seamlessly between Mininet (allowing inexpensive and streamlined development), and the real hardware running at line rate in live deployments

- 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 hypervisors (e.g., VirtualBox, or KVM)
  2. Download the Mininet pre-installed VM image
  3. Import VM image into selected hypervisor

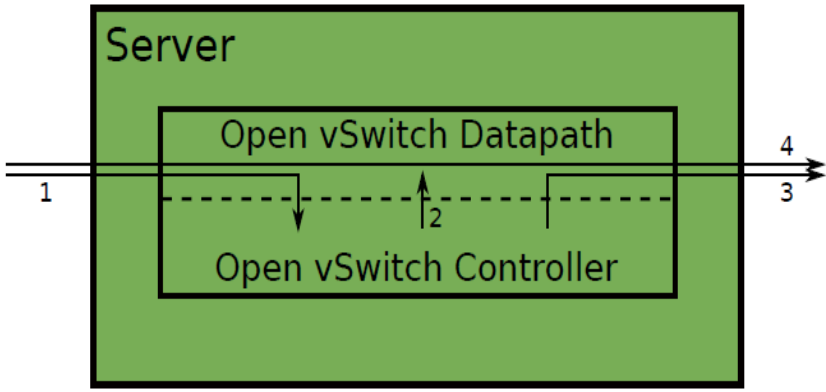
Mininet: emulated hosts and switches

- 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



Open vSwitch as an OpenFlow software switch

- 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 returns the packet to the datapath
  - Subsequent packets are handled directly by the datapath



Open vSwitch and OpenFlow compatibility matrix

		OpenFlow					
		OF1.0	OF1.1	OF1.2	OF1.3	OF1.4	OF1.5
Open vSwitch	1.9 and earlier	yes	—	—	—	—	—
	1.10, 1.11	yes	—	(*)	(*)	—	—
	2.0, 2.1	yes	(*)	(*)	(*)	—	—
	2.2	yes	(*)	(*)	(*)	(%)	(*)
	2.3, 2.4	yes	yes	yes	yes	(*)	(*)
	2.5, 2.6, 2.7	yes	yes	yes	yes	(*)	(*)
	2.8, 2.9, 2.10, 2.11	yes	yes	yes	yes	yes	(*)
	2.12	yes	yes	yes	yes	yes	yes

(*) Supported, with one or more missing features.
(%) Experimental, unsafe implementation.

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: network topologies

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

```
$ sudo mn
```

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 }
```

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
```

- 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 paramters to all links

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

  - Assigns a given bandwidth and delay to links

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: 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
```

- 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]
```

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# wireshark &
```

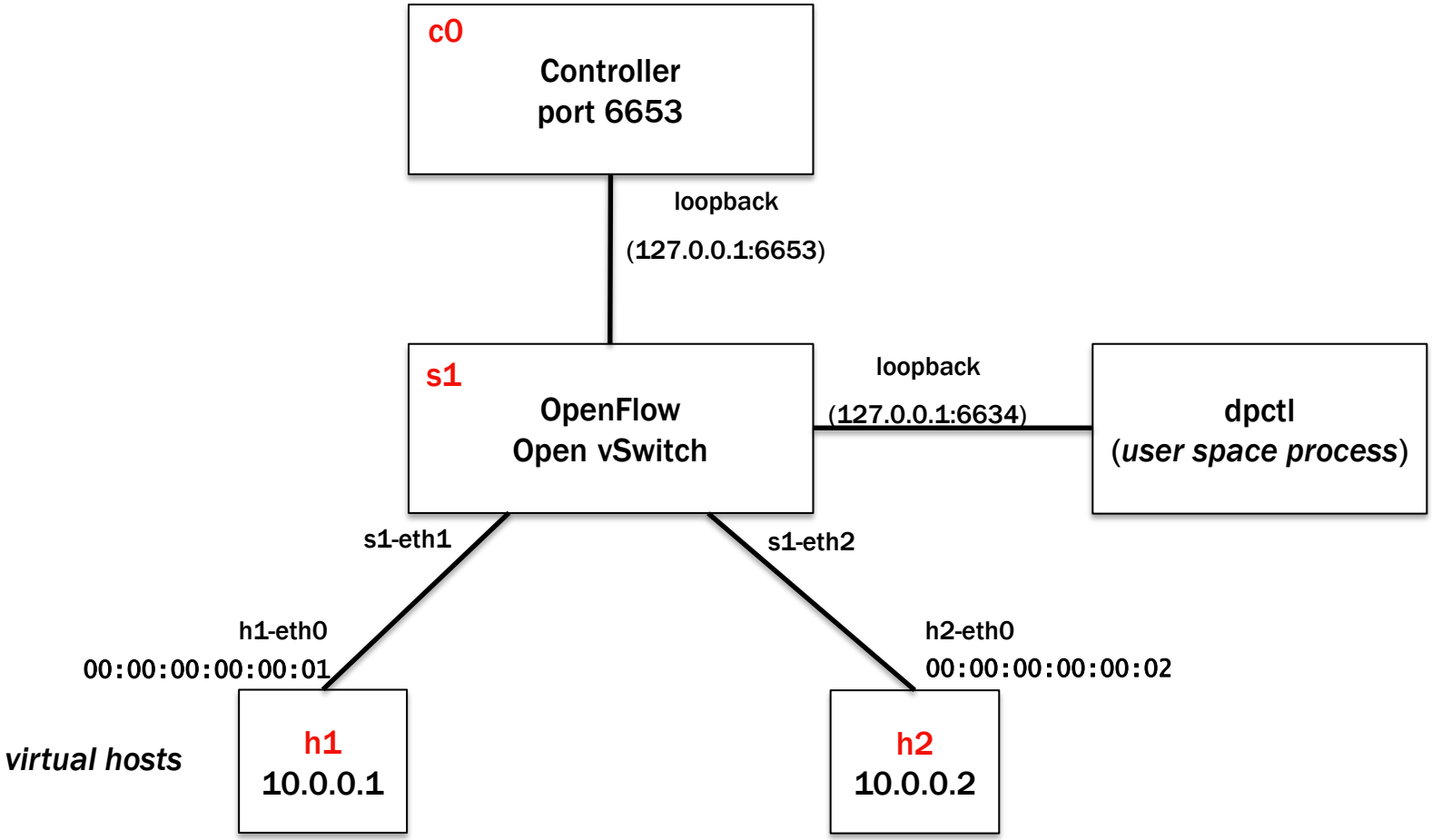
- Exec a simple web server (listenong on port 8000) from xterm on h2

```
h2# python –m SimpleHTTPServer &
```

- Let h1 ping h2 from Mininet prompt

```
mininet> h1 ping h2
```

Experiment #1: components



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`)

Open vSwitch CLI commands (1)

- `sudo ovs-vsctl show`

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

- `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
```

```
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"
```

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 -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 a 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

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

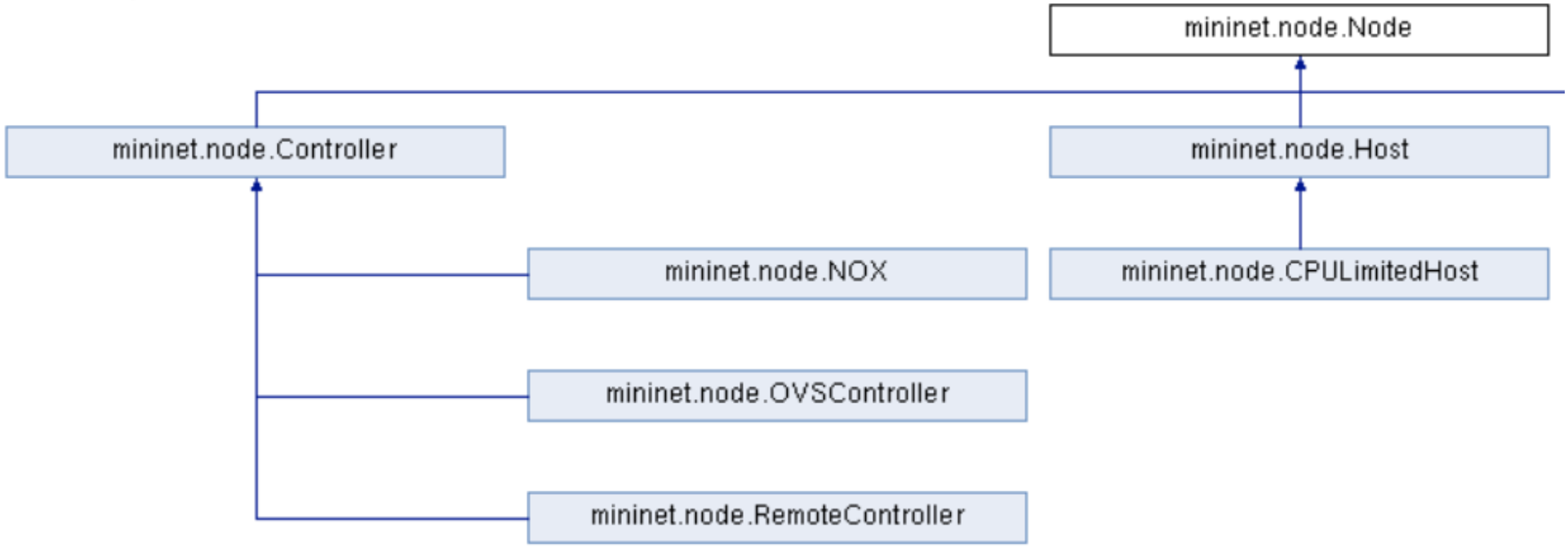
```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.0.0.1/8' )
h2.setIP( '10.0.0.2/8' )
```

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

Node class and subclasses (1/2)

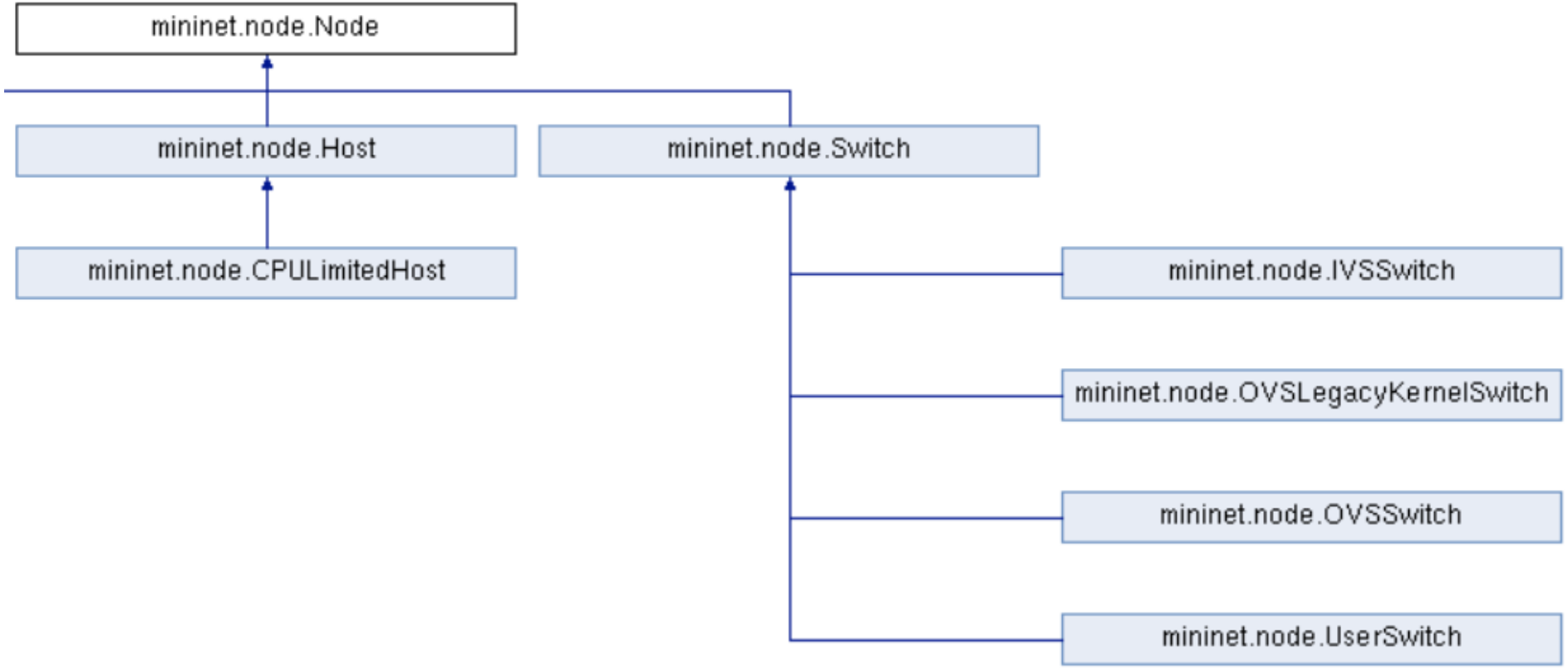
- Node generic class
- 3 subclasses: **Controller, Host,** Switch

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
	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()
#...
```

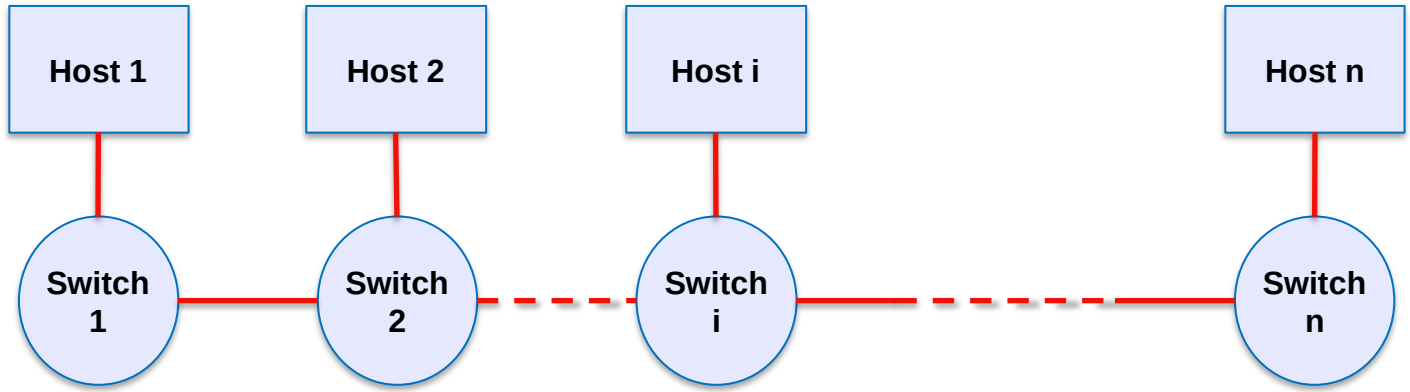
Custom test function

Third Mininet script (3/3)

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

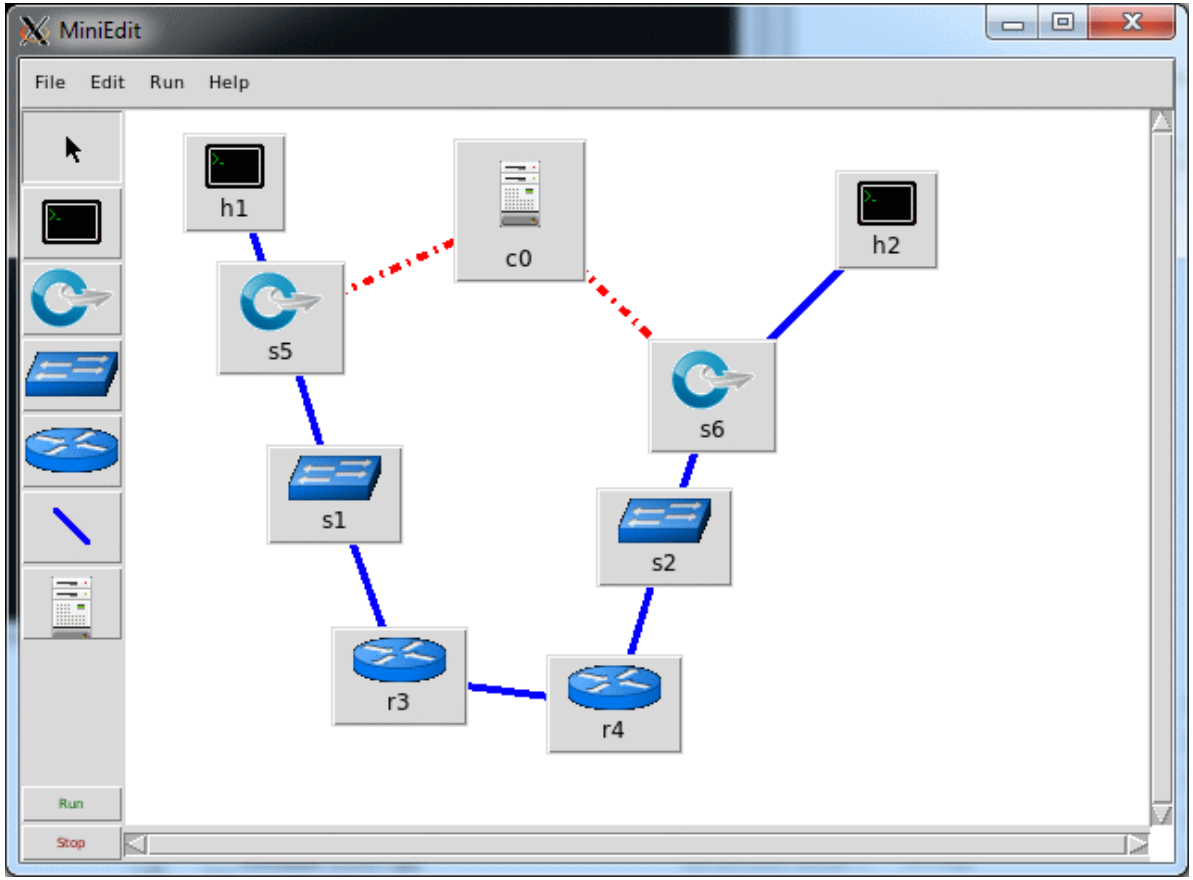
Script startup

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



Starting Ryu

- Ryu is a Python based controller
- Most controller platforms, including Ryu, expose some native features to allow these key features:
  - Ability to listen to asynchronous events (e.g., PACKET_IN, FLOW_REMOVED) and to observe events using ryu.controller.handler.set_ev_cls decorator.
  - Ability to parse incoming packets (e.g., ARP, ICMP, TCP) and fabricate packets to send out into the network
  - Ability to create and send an OpenFlow/SDN message (e.g., PACKET_OUT, FLOW_MOD, STATS_REQUEST) to the programmable dataplane.
- With RYU you can achieve all of those by invoking set of applications to handle network events, parse any switch request and react to network changes by installing new flows, if required
- A basic controller functionality is implemented in the `simple_switch_13.py` module:

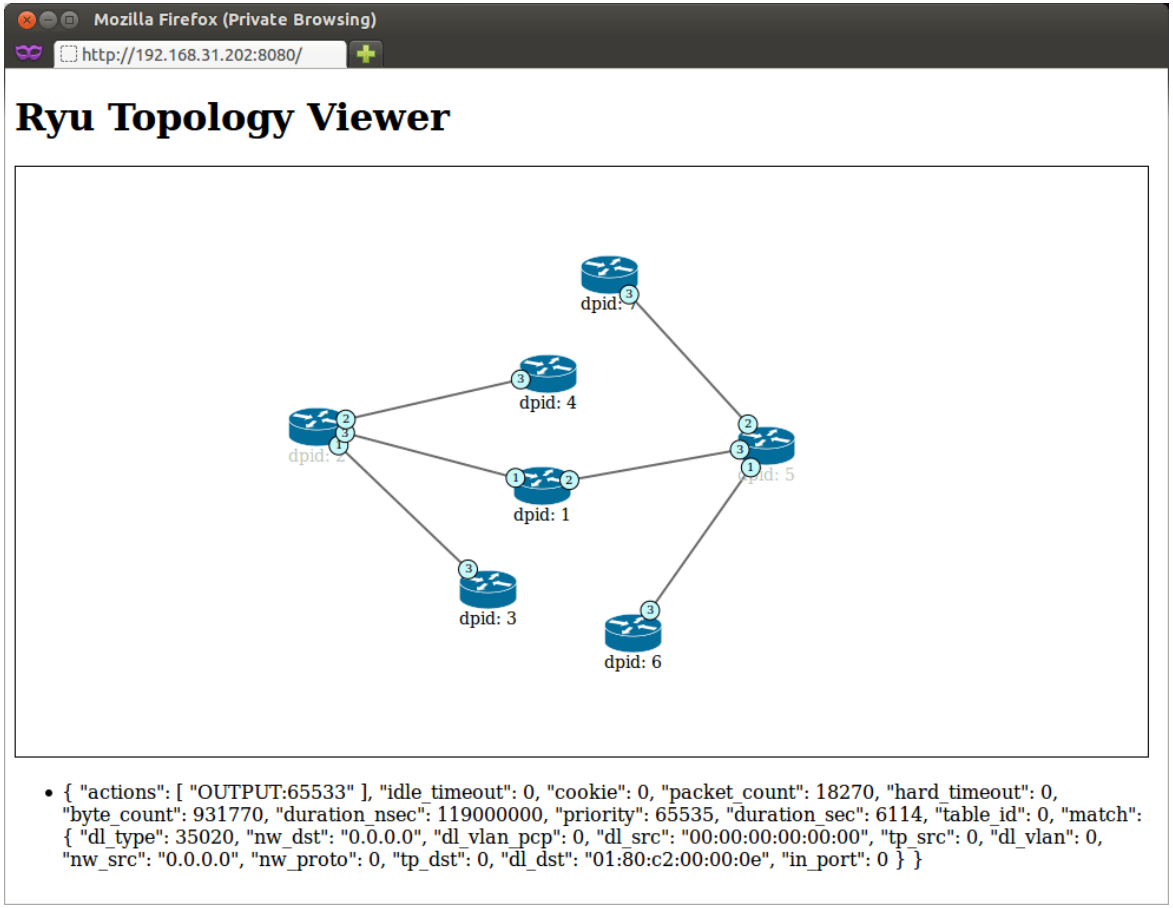
```
$ ryu-manager --verbose ryu/app/simple_switch_13.py
```

Starting Ryu with a GUI module

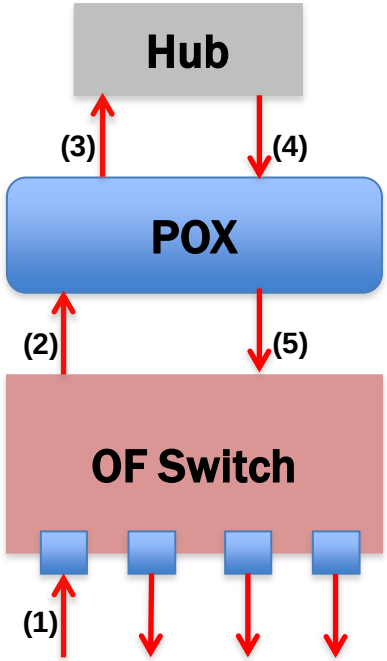
- Ryu provides a minimal web-based GUI that shows a graphical view of the network topology
- To start the GUI:

```
$ ryu-manager --verbose --observe-links
    ryu/app/gui_topology/gui_topology
    ryu/app/simple_switch_13.py
```

- The GUI is accessible with a browser on port 8080 of the host running the controller `http://controller_IP:8080`
- The GUI shows flow rules in each of the switches

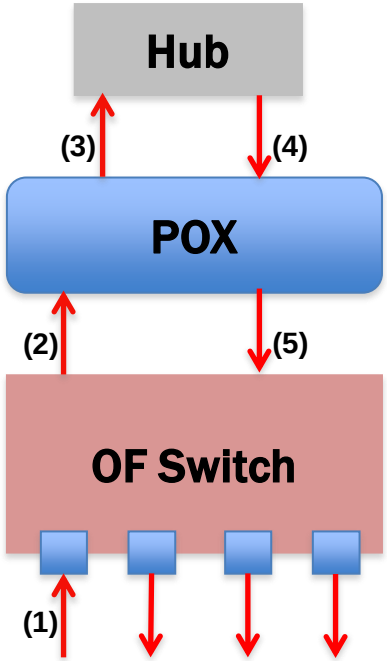


Controller logic: case #1 - hub



- App logic:
  - On init, register the appropriate packet_in handlers
  - On packet_in:
    - Extract full packet
    - Generate packet_out msg with data of the received packet
    - Set action = FLOOD
    - Send packet_out msg to the switch that generated the packet_in

Controller logic in POX: case #1 – hub




```
from pox.core import core
import pox.openflow.libopenflow_01 as of

# Object spawned for each switch
class L2Hub (object):
    def __init__ (self, connection):
        # Keep track of the connection to the switch so that we can
        # send it messages!
        self.connection = connection

        # This binds all our event listener
        connection.addListeners(self)

    # Handles packet in messages from the switch.
    def _handle_PacketIn (self, event):
        packet = event.parsed # This is the parsed packet data.

        packet_in = event.ofp # The actual ofp_packet_in message.

        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id

        # Add an action to send to the specified port
        action = of.ofp_action_output(port = of.OFPP_FLOOD)
        msg.actions.append(action)

        # Send message to switch
        self.connection.send(msg)


def launch ():
    def start_switch (event):
        L2Hub(event.connection)

    core.openflow.addListenerByName("ConnectionUp", start_switch)
```

Controller logic: case #2 - learning switch

- App logic:
  - On init, create a dict to store MAC to switch port mapping

```
self.mac_to_port = {}
```

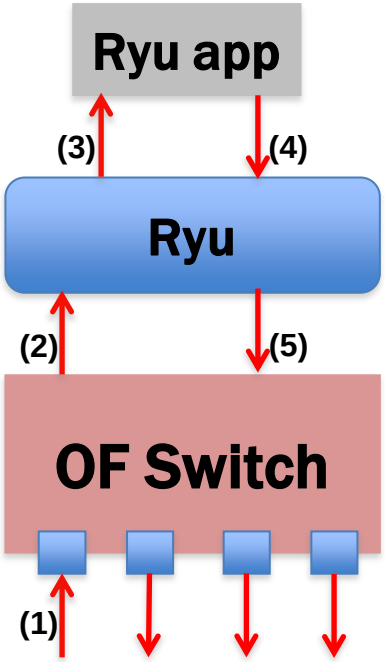
  - On packet_in:
    - Parse packet to reveal src and dst MAC addr
    - Map src_mac to the incoming port

```
self.mac_to_port[dpid][src_mac] = in_port
```

    - Lookup dst_mac in mac_to_port dict to find out_port
    - If found, create flow_mod and send packet else, flood like hub

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

Controller logic in Ryu: case #2 - learning switch (1/3)



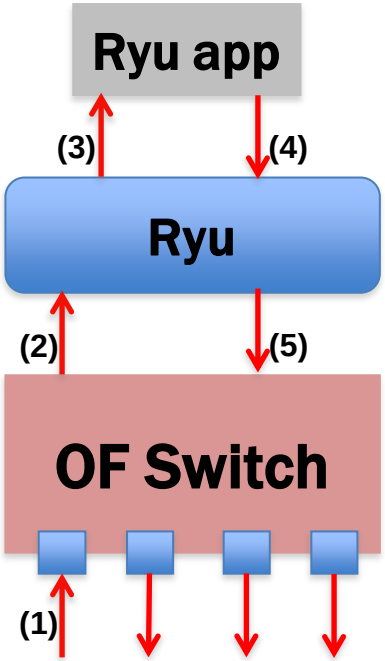
ryu/app/simple_switch_13.py

```
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

class ExampleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ExampleSwitch13, self).__init__(*args, **kwargs)
        # initialize mac address table.
        self.mac_to_port = {}
        def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        # construct flow_mod message and send it.
        inst=[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)
        ...
```

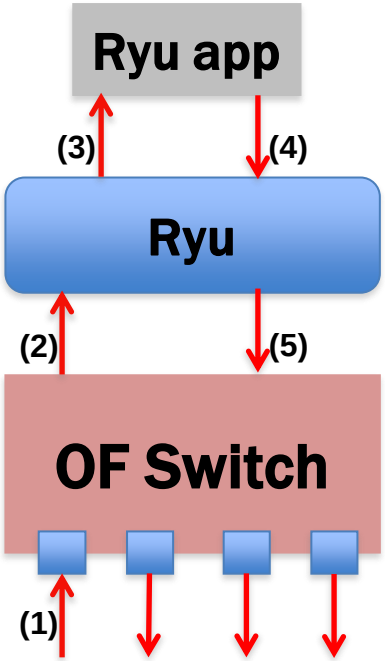
Controller logic in Ryu: case #2 - learning switch (2/3)



ryu/app/simple_switch_13.py

```
...
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    # get Datapath ID to identify OpenFlow switches.
    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})
    # analyse the received packets using the packet library.
    pkt = packet.Packet(msg.data)
    eth_pkt = pkt.get_protocol(ethernet.ethernet)
    dst = eth_pkt.dst
    src = eth_pkt.src
    # get the received port number from packet_in message.
    in_port = msg.match['in_port']
    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 the destination mac address is already learned,
    # decide which port to output the packet, otherwise FLOOD.
    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD
    ...
```

Controller logic in Ryu: case #2 - learning switch (3/3)



ryu/app/simple_switch_13.py

```
        ...
        @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
        def _packet_in_handler(self, ev):
            ...
            ...
            ...
            ...
            # construct action list.
            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)
                self.add_flow(datapath, 1, match, actions)
            # construct packet_out message and send it.
            out = parser.OFPPacketOut(datapath=datapath,
                                      buffer_id=ofproto.OFP_NO_BUFFER,
                                      in_port=in_port, actions=actions,
                                      data=msg.data)
            datapath.send_msg(out)
```

Handling of events in Ryu

- The line

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
```

- before the `_packet_in_handler` function definition is a Python *decorator*
- Decorators are sort of wrapper functions (defined elsewhere in the code) that are executed when a function is invoked
- The `set_ev_cls` decorator in Ryu is used to register a handler function as associated to a specific event
- In particular, the decorator swown above is used to associate the function `_packet_in_handler` to the reception of OpenFlow PACKET_IN messages
- Each Ryu application has a first-in/first-out queue for handling events by preserving their order
- A Ryu application is single-threaded
- In Ryu, an OpenFlow message of type `<name>` is associated to an event that is instance of the class `ryu.controller.opf_event.EventOF<name>`