SDN Hands-on

Kotaro Kataoka

Outline: Let's Play with RYU & Mininet

- Completing installing of Mininet + Ryu
- Frequently used commands about OVS
- Running Learning Switch
- SDN App Development in a nutshell
  - Let's go deeper in the learning switch code
  - Let's implement Port Mirroring

Installing Mininet and Upgrading Open vSwitch

- Mininet Installation
  - https://github.com/mininet/mininet/wiki/Mininet-VM-Images
- Launch Mininet VM (version 2.x.x) and check IP Address
  - mininet@mininet-vm:~$ ifconfig
- Loging from VM host using SSH (-X for X11)
  - $ ssh -X mininet@Mininet_VM_IP_ADDRESS
- Upgrade Open vSwitch
  - http://techandtrains.com/2013/10/24/another-way-to-upgrade-open-vswitch-in-mininet/

Installating SDN Controller to Mininet VM

- Installing Floodlight

  sudo apt-get install build-essential default-jdk ant python-dev eclipse

  Download floodlight @http://www.projectfloodlight.org/download/

  Please install Version 1.1 for developers (No VM Appliance) or Floodlight master(latest version) https://github.com/floodlight/floodlight

- Installing RYU using pip

  % pip install ryu

- Installing from the source code

  % git clone git://github.com/osrg/ryu.git

  % cd ryu; python ./setup.py install

Launching ryu-manager

$ ryu-manager --version

- Got Errors? Let's check required package is installed in Mininet-VM and re-install Ryu
  - pkg_resources.VersionConflict: (six 1.5.2 (/usr/lib/python2.7/dist-packages), Requirement.parse('six>=1.7.0'))

  $ sudo pip install -U six

  - pkg_resources.DistributionNotFound: pbr>=0.6,!=0.7,<1.0

  $ easy_install --upgrade pip

  $ sudo python ./setup.py install

$ ryu-manager --version

Launching Mininet with 3 hosts, 1 switch, 1 controller

- sudo mn --topo single,3 --mac --switch ovsk --controller remote -x
  - Topology of one switch and three hosts
  - Automatically sets the MAC address of the host
  - Open vSwitch as SDN switch
  - Uses external OpenFlow controller
  - xterm for generated nodes (hosts/switch/controller)

- Did 5 Windows come up?

Showing OVS-Configuration

```
root@mininet-vm:~# ovs-vsctl show
1077578e-f495-46a1-a96b-441223e7cc22
    Bridge "s1"
        Controller "tcp:127.0.0.1:6633"
        Controller "ptcp:6634"
        fail_mode: secure
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth3"
            Interface "s1-eth3"
    ovs_version: "2.0.2"
```

- Open vSwitch is abstracted as “Bridge”. “s1” is the name of switch

Showing port and name on s1

```
root@mininet-vm:~# ovs-dpctl show
system@ovs-system:
        lookups: hit:127 missed:57 lost:0
        flows: 0
        port 0: ovs-system (internal)
        port 1: s1-eth3
        port 2: s1-eth1
        port 3: s1-eth2
        port 4: s1 (internal)
```

Using OVS with OpenFlow 1.x

- Configuring OpenFlow 1.3 on s1
  - Default version is 1.0

$ ovs-vsctl set bridge s1 protocols=OpenFlow1x

- And then check the empty flow table

$ ovs-ofctl -O OpenFlow13 dump-flows s1

- Datapath ID (DPID) of s1?

$ ovs-vsctl get bridge s1 datapath-id

What happen if hosts attempt to communicate now?

- Ping fails because controller does not run any program

mininet> pingall

*** Ping: testing ping reachability

h1 -> X X

h2 -> X X

h3 -> X X

*** Results: 100% dropped (0/6 received)

Lanching Learning Switch (OF1.x)

root@mininet-vm:~# ryu-manager --verbose ryu.app.simple_switch_1x

- Then let’s do something again

mininet> pingall

What did happen to Controller?

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:03 00:00:00:00:00:01 3

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:01 00:00:00:00:00:03 1

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:03 00:00:00:00:00:02 3

EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn

packet in 1 00:00:00:00:00:02 00:00:00:00:00:03 2

Let’s take a look of Flow Table

root@mininet-vm:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=331.815s, table=0, **n_packets**=3, **n_bytes**=238, priority=1,**in_port**=3,**dl_dst**=00:00:00:00:00:02 **actions**=output:2
cookie=0x0, duration=331.834s, table=0, n_packets=373014, n_bytes=24618948, priority=1,in_port=2,dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x0, duration=331.825s, table=0, n_packets=3, n_bytes=238, priority=1,in_port=3,dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x0, duration=331.814s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=2,dl_dst=00:00:00:00:00:03 actions=output:3
cookie=0x0, duration=331.824s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=1,dl_dst=00:00:00:00:00:03 actions=output:3
cookie=0x0, duration=331.831s, table=0, n_packets=373036, n_bytes=16322899304, priority=1,in_port=1,dl_dst=00:00:00:00:00:02 actions=output:2
cookie=0x0, duration=395.824s, table=0, n_packets=9, n_bytes=546, priority=0 actions=CONTROLLER:65535

Developing SDN Application using Ryu in a Nutshell

Thanks to
http://sdnhub.org/tutorials/ryu/

https://github.com/osrg/ryu-book/blob/master/ja/source/switching_hub.rst

Key Components of Ryu

- app/ – Contains set of applications that run on-top of the controller.
- base/ – Contains the base class for RYU applications. The RyuApp class in the app_manager.py file is inherited when creating a new application.
- controller/ – Contains the required set of files to handle OpenFlow functions (e.g., packets from switches, generating flows, handling network events, gathering statistics etc).
- lib/ – Contains set of packet libraries to parse different protocol headers and a library for OFConfig. In addition, it includes parsers for Netflow and sFlow too.
- ofproto/ – Contains the OpenFlow protocol specific information and related parsers to support different versions of OF protocol (1.0, 1.2, 1.3, 1.4)
- topology/: Contains code that performs topology discovery related to OpenFlow switches and handles associated information (e.g., ports, links etc). Internally uses LLDP protocol.

What to implement?

- 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.

Let’s take a look of simple_switch13.py

Importing Libraries

```
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
```

Event Handlers

Packet Headers

Definition	Explanation
ryu.controller.handler.HANDSHAKE_DISPATCHER	Exchange of HELLO message
ryu.controller.handler.CONFIG_DISPATCHER	Waiting to receive SwitchFeatures message
ryu.controller.handler.MAIN_DISPATCHER	Normal status
ryu.controller.handler.DEAD_DISPATCHER	Disconnection of connection

Initiating a new class

```
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 = {}
```

- Speficy OpenFlow version to be 1.3
- Create Learning Table
  - Dictionary for “mac_to_port”

Install table-miss flow entry during controller configures OVS

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
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)
```

During the configuration event of OVS…

Create a vacant match to let the switch send packet_in to the controller if no flow entry matches in the flow table. Injected as a flow rule.

add_flow ()

```
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)
```

“flow_mod” message is sent from Controller to Switch

Unknown Packet Comes!

```
@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]

    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)
```

Every time packet_in message comes, then this handler is called.

From where did the packet come?

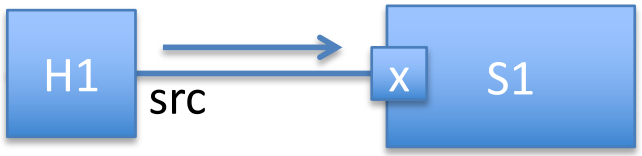
We want the content of Ethernet Header

Logging on Controller Contole

Core Part as Learning Switch

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

On S1, H1 is reachable via Port X




```
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)]
```

Deciding outgoing port of the packet
One port to DST or Flooding

```
# 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)
    # 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
```

Add a flow rule to the destination host at the OVS if it has been known.

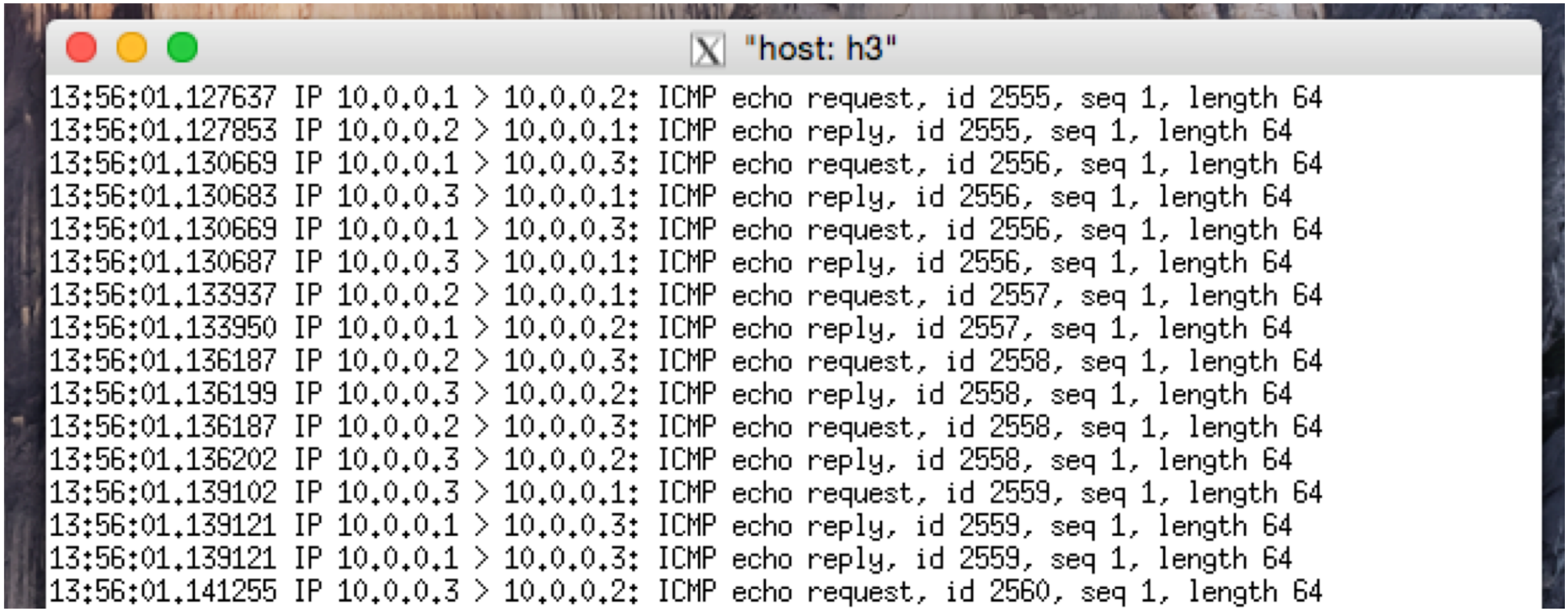
```
out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```

Packet Out!

A Small Extension

Enabling port mirroring on the learning switch

- One port on OVS copies all incoming packets to Mirroring Port (ex. Port 3 on S1)
- You may hard-code the mirroring port



Question

- How to modify the sample code?

- Think by your self first!

Answer

```
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)]
```

Add 3 lines here!!

```
if in_port != 3:
    if out_port != ofproto.OFPP_FLOOD:
        actions = [parser.OFPActionOutput(out_port), parser.OFPActionOutput(3)]

# 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)
    # 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)
```

Running Your Code

- Stop the running controller (Ctrl-C on Controller)

- Flush the flow table of S1

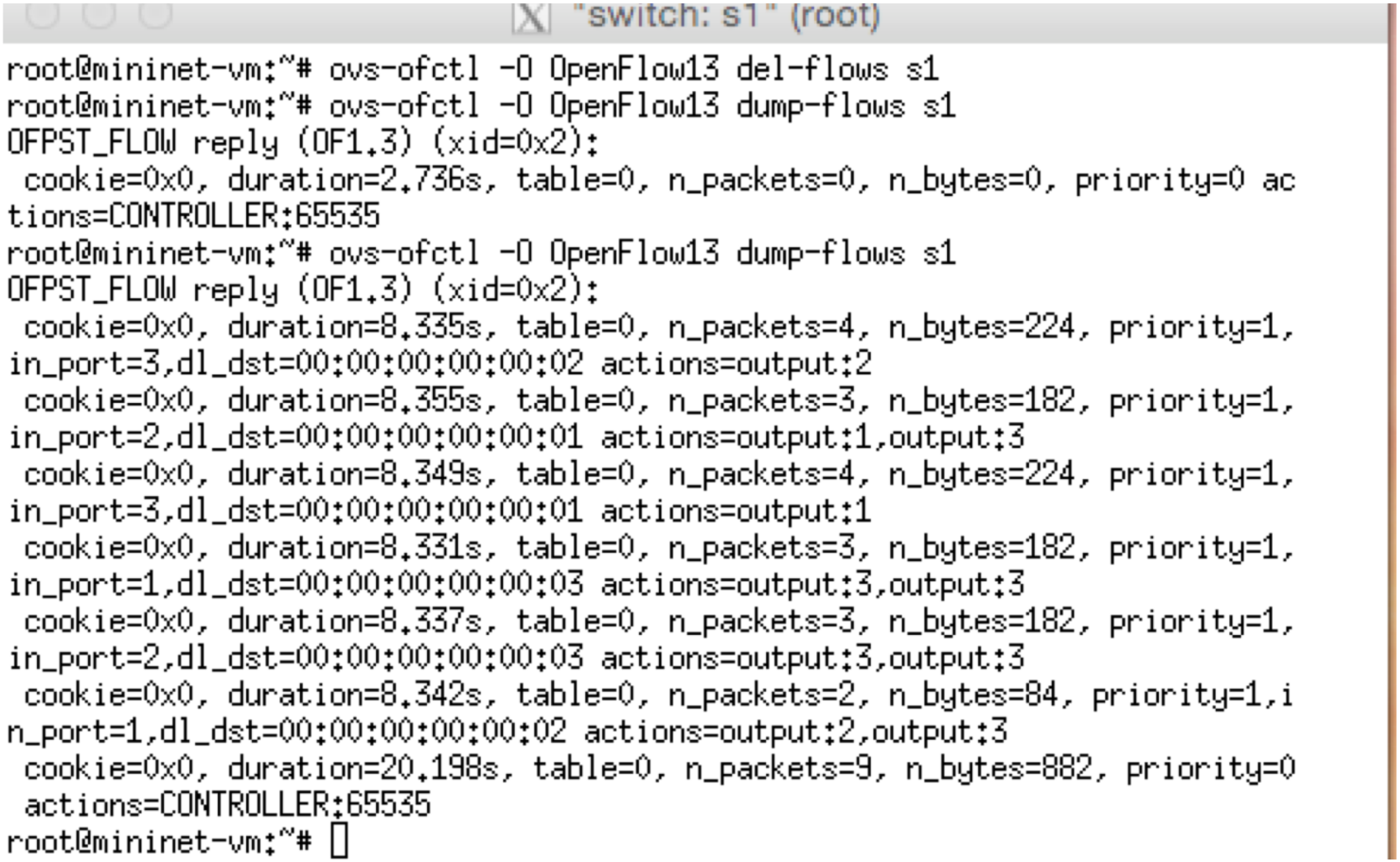
ovs-ofctl -O OpenFlow13 del-flows s1

- Run your code

ryu-manager simple_switch_13-Mirror.py

Oops… Doesn't look very good?

- 2 out_ports for in_port 1 and 2 is correct, but…



Additional Hands-on?

- SDN-based Router
  - Develop network topology with multiple subnets
  - Giving IP address from different subnet to each host
  - Give static IP address to each interface on OVS -> Default gateway
  - Handling ARP messages from hosts
  - Handling IP packets from hosts
  - You may forget updating TTL, returning ICMP error messages

Done