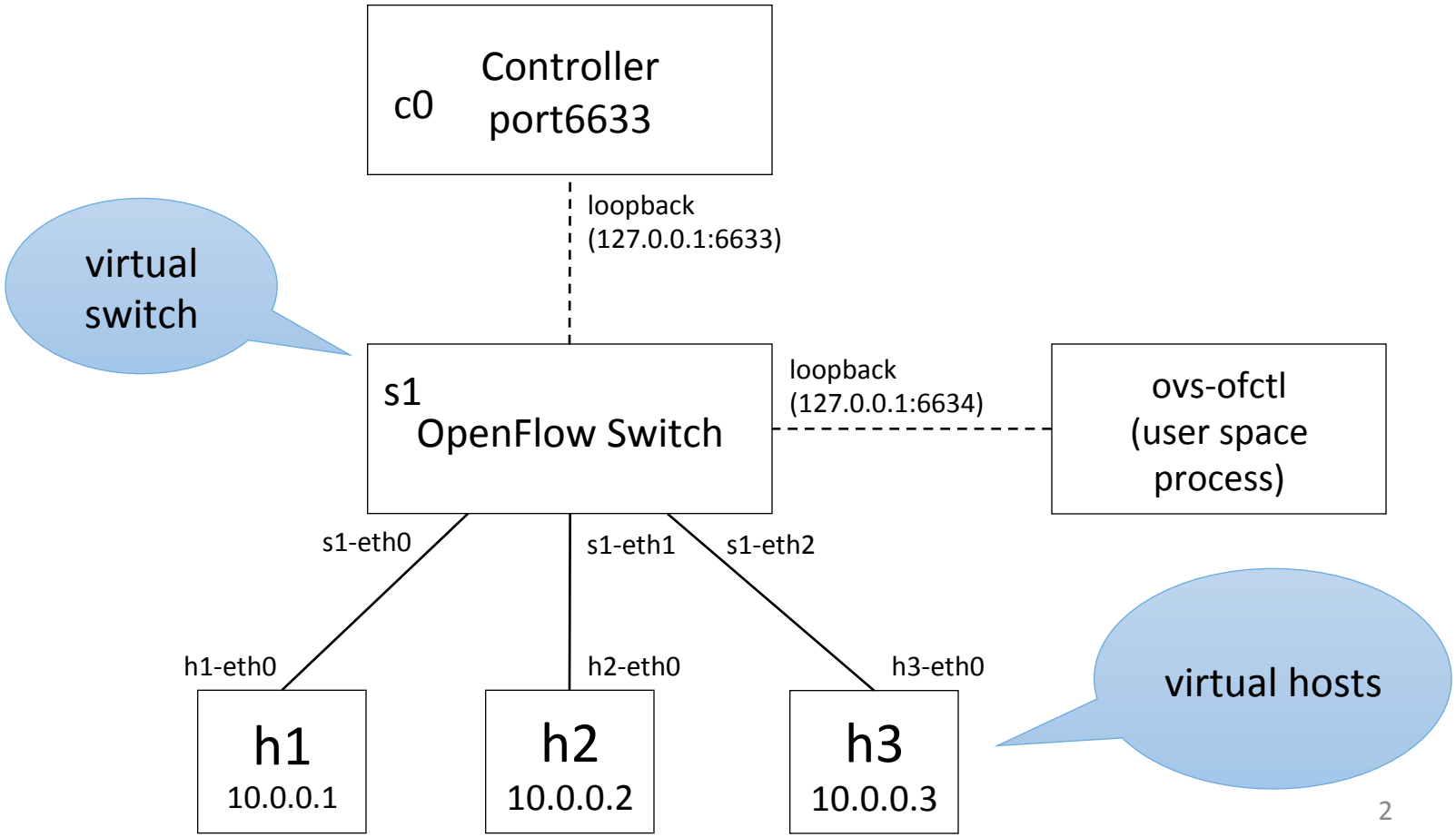
Mininet & OpenFlow

19/05/2017

Setup 1: Mininet-based Single Switch

```
sudo mn --topo single,3 --switch ovsk --controller remote
```



Pox Controller

- POX is an open platform for the rapid development and prototyping of **network control software**

- Pox architecture is ‘‘component based’’

- Ex: ~/pox$ `./pox.py samples.pretty_log forwarding.l2_learning`

- Some stock components:
  - openflow.of_01 *(usually started automatically)*
  - forwarding.hub
  - forwarding.l2_learning
  - forwarding.l2_pairs
  - forwarding.l2_multi
  - openflow.spanning_tree
  - openflow.discovery
  - **misc.of_tutorial** → the component we will customize in this lab
  - …

Pox Controller: an Example

- Open two terminals in the Mininet VM
- In the first terminal execute command 1)
- Then, execute a ping test, does it work?
- In the other terminal execute command 2) and repeat the ping test. Something changed??

Commands:

1) `~$ sudo mn --topo single,3 --controller remote`
2) `~/pox$ ./pox.py samples.pretty_log forwarding.l2_learning`

Packets in POX

- POX generally works with `ethernet` packets
  - Which often contain `ipv4` packets…
    - (which often contain `tcp` packets…)

- Some of the packet types supported by POX:
  - ethernet, arp, ipv4, icmp, tcp, udp, dhcp, dns…

- Most packets have some sort of header and some sort of a payload
  - The payload is another type of packet

Ethernet packets in POX

- `Class ethernet`
  - defined in ~/pox/pox/lib/packet/ethernet.py
- Attributes:
  - `dst (EthAddr)`
  - `src (EthAddr)`
  - `type (int)`
  - `effective_ethertype (int)`
  - `payload (for example an ipv4 packet…)`
- Constants:
  - `IP_TYPE, ARP_TYPE, VLAN_TYPE, …`
- Example: `packet.src, packet.IP_TYPE`

The Event System

- Event Handling in POX fits into the publish/subscribe paradigm
  - Certain objects publish events and others subscribe to specific events on these objects
- In other words: we'd like a particular piece of code to be called
- Ex: `chef.addListenerByName("SpamReady", spam_ready)`

The object that raises the event

The name of the event

The function handling the event

The Event System

- Ex: object chef raises two events, SpamReady and SpamFinished

```
class HungryPerson (object):
  """ Models a person that loves to eat spam """

  def __init__ (self):
    chef.addListeners(self)

  def _handle_SpamReady (self, event):
    print "I can't wait to eat!"

  def _handle_SpamFinished (self, event):
    print "Spam is finished!  Smelt delicious!"
```

Example: empty controller

- Let's go to the code and see the events ConnectionUp and PacketIn!

- ConnectionUp: fired in response to the establishment of a new control channel with a switch

- PacketIn: Fired when the controller receives an OpenFlow Packet-In message from a switch
  - Attributes:
    - `port` (`int`): number of port the packet came in on
    - `data` (`bytes`): raw packet data
    - `parsed` (`packet subclasses`): packet's parsed version
    - `ofp` (`ofp_packet_in`): OpenFlow message which caused this event

Packet-In message in POX

- The POX object type is `ofp_packet_in`
- Attributes:
  - `in_port (int)`: number of port the packet came in on
  - `data (bytes)`: raw packet data
  - `buffer_id (int)`: ID of the buffer in which the packet is stored at the switch
  - …

Packet-Out message in POX

- The POX object type is `ofp_packet_out`

attribute	type	default	notes
in_port	int	OFPP_NONE	Switch port that the packet arrived on, if resending a packet
data	bytes / ethernet / ofp_packet_in	''	The data to be sent. If you specify an `ofp_packet_in` for this, `in_port`, `buffer_id`, and `data` will all be set correctly – this is the easiest way to resend a packet.
buffer_id	int/None	None	ID of the buffer in which the packet is stored at the switch. If you're not resending a buffer by ID, use None
actions	list of ofp_action_XXXX	[ ]	An action or a list of actions

OpenFlow actions in POX

- `ofp_action_output:` Forward packets out of a port
- `Ex: of.ofp_action_output(port = 4)`

Output port for the packet

Reference to the object that manages the OpenFlow protocol

Possible values for ‘‘port’’:

- **OFPP_IN_PORT**: Send back out the port the packet was received on
- **OFPP_TABLE**: Perform actions specified in flowtable. Note: Only applies to ofp_packet_out messages
- **OFPP_NORMAL**: Process via normal L2/L3 legacy switch configuration (if available – switch dependent)
- **OFPP_FLOOD**: output all openflow ports except the input port and those with flooding disabled
- **OFPP_ALL**: output all openflow ports except the in port
- **OFPP_NONE**: Output to no where
- …

OpenFlow messages in POX

```
""" Instructs the switch to resend a packet that
it had sent to us. "packet_in" is the ofp_packet_in object
the switch had sent to the controller due to a table-miss. """

msg = of.ofp_packet_out()
msg.data = packet_in

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

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

Example: `of_tutorial.py`

- Let's go to the code and see the OpenFlow tutorial!
- You can find the code here:

~/pox/pox/misc/of_tutorial.py

- To start the controller, type in the ~/pox folder:

```
./pox.py misc.of_tutorial
    samples.pretty_log
```

Exercise 2

- Modify the `of_tutorial.py` to implement the behavior of a learning switch using the OpenFlow message `Packet-Out`

IP packets in POX

- `Class ipv4`
  - defined in ~/pox/pox/lib/packet/ipv4.py
- Attributes:
  - `dstip (IPAddr)`
  - `srcip (IPAddr)`
  - `protocol (int)`
  - `payload (for example a TCP packet…)`
- Constants:
  - `TCP_PROTOCOL, UDP_PROTOCOL, …`

TCP packets in POX

- `Class tcp`
  - defined in ~/pox/pox/lib/packet/tcp.py
- Attributes:
  - `dstport (EthAddr)`
  - `srcport (EthAddr)`
  - `SYN (bool)`
  - `FIN (bool)`
  - `ACK (for example an ipv4 packet…)`
  - …

Example

```
# packet is the ethernet packet
if (packet.type == packet.IP_TYPE):

     ipPkt = packet.payload
     if (str(ipPkt.srcip) == "10.0.0.1"):

          if (ipPkt.protocol == ipPkt.TCP_PROTOCOL):

               tcpPkt = ipPkt.payload

               …

     else:
          return False

else:
          return False
```

Exercise 3

- Develop a firewall that allows only
  - ARP packets
  - TCP packets over IP packets, but only if:
    - directed to host 10.0.0.1 (port 80)
    - host 10.0.0.1 is the source

- **Hint**: use the **nc** command to test your firewall
  - Server: `nc -l 80` *# open a socket*
  - Client: nc *<serv IP addr>* 80 *# connect to the server*