逄愛君 / Lecturer

馮才昇 李峻宇 劉厚辰/ T.A.s

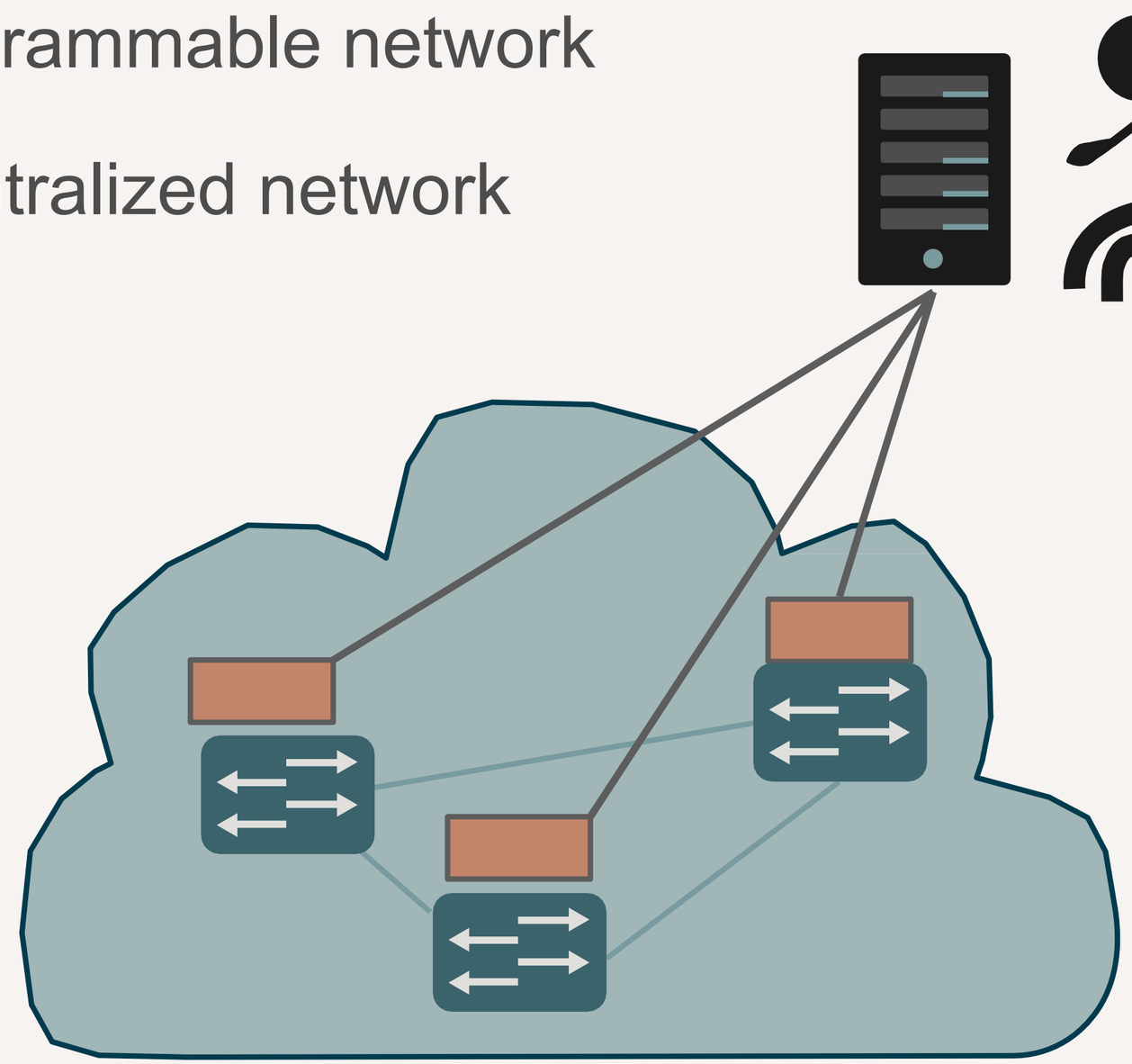
SIMULATING SDN WITH MININET

Outline

- SDN Overview
- Mininet Overview
- Ryu Overview
- Lab Announcement
- Mininet VM & Ryu tutorial
- OpenFlow Protocol and Open vSwitch
- Reference

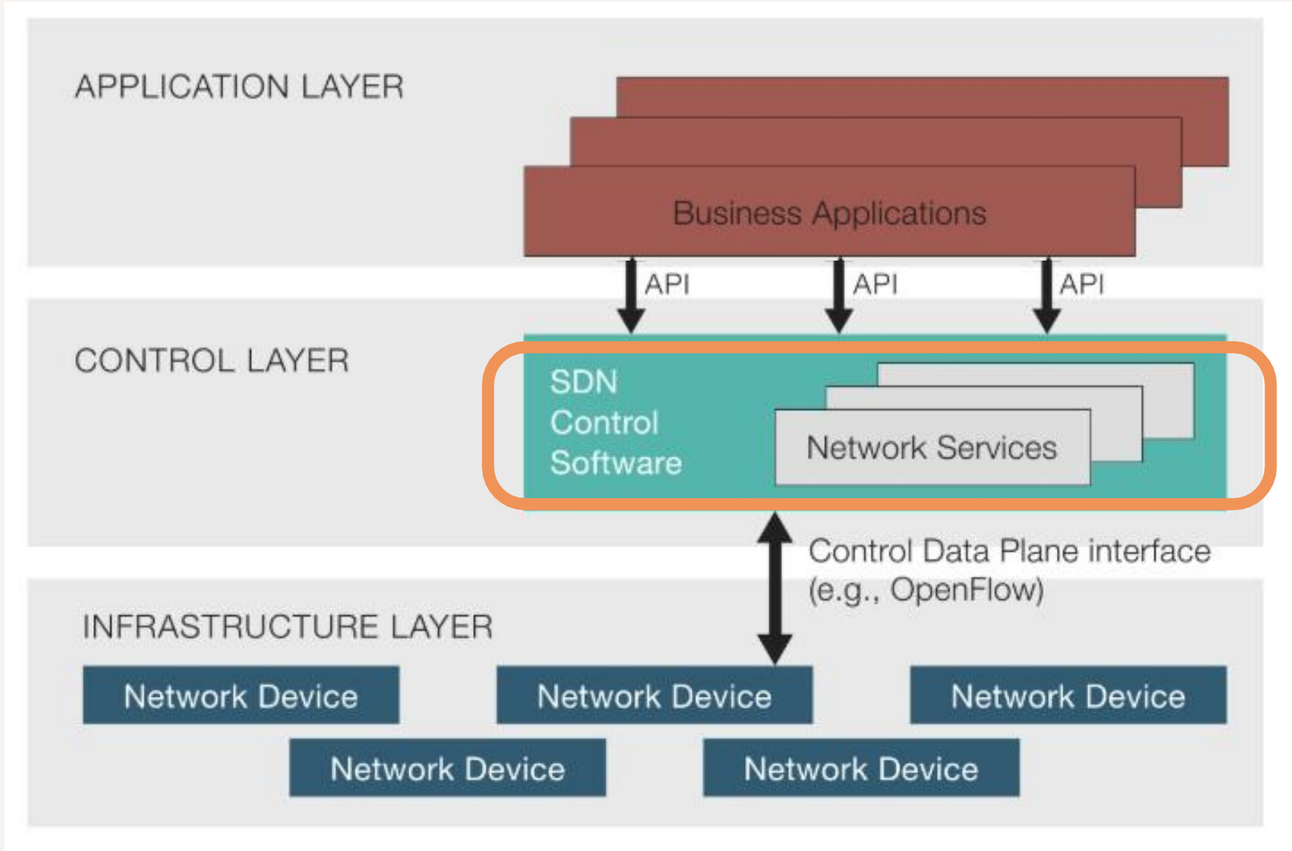
SDN Overview

- Decoupling of control and data planes
- Directly Programmable network
- Logically centralized network



SDN Overview

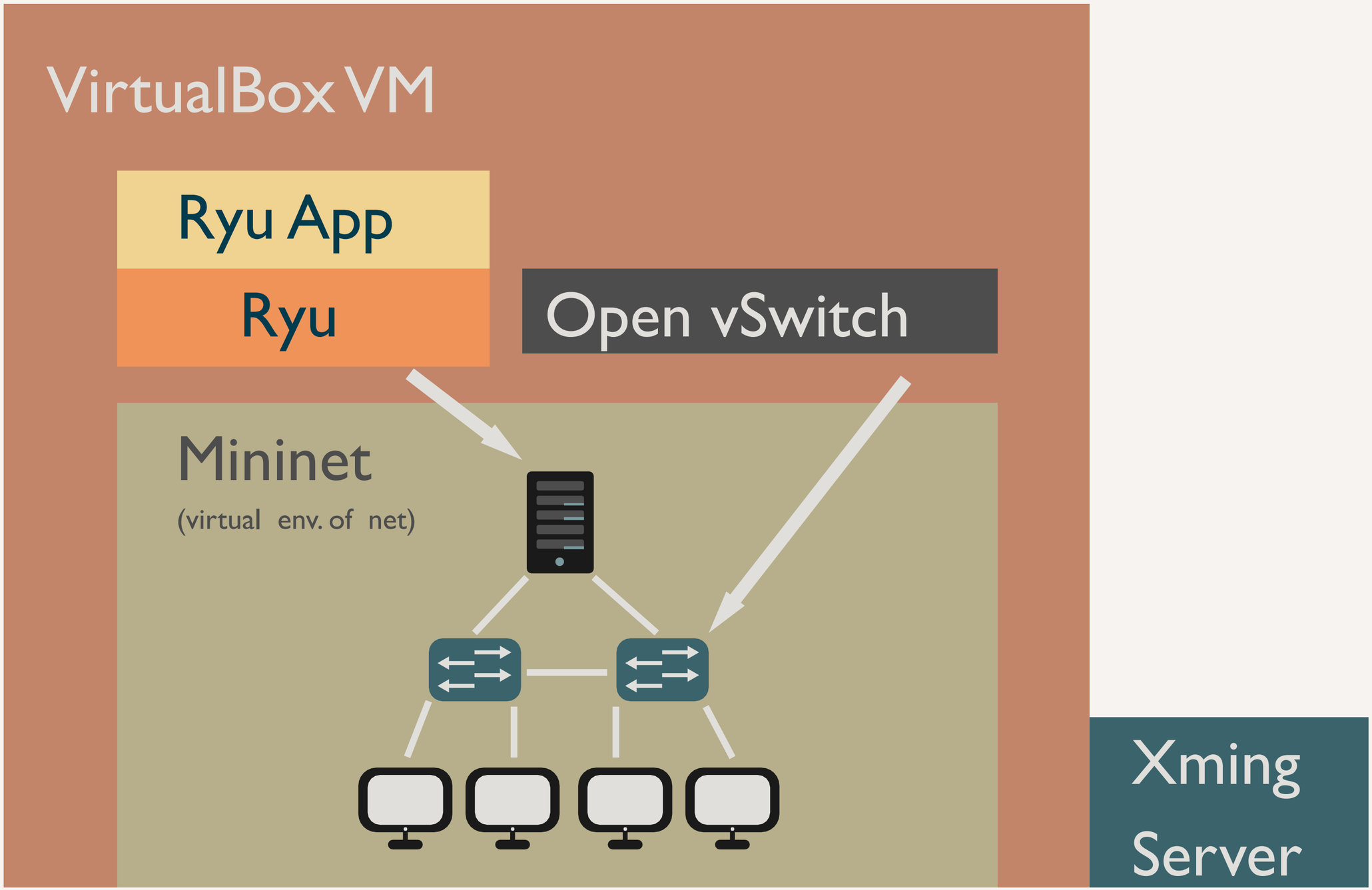
- Under SDN architecture, Ryu framework is in the control layer.
- It helps developers to communicate between the controller and underlying Open vSwitch.



Ryu Framework

Mininet and Open vSwitch

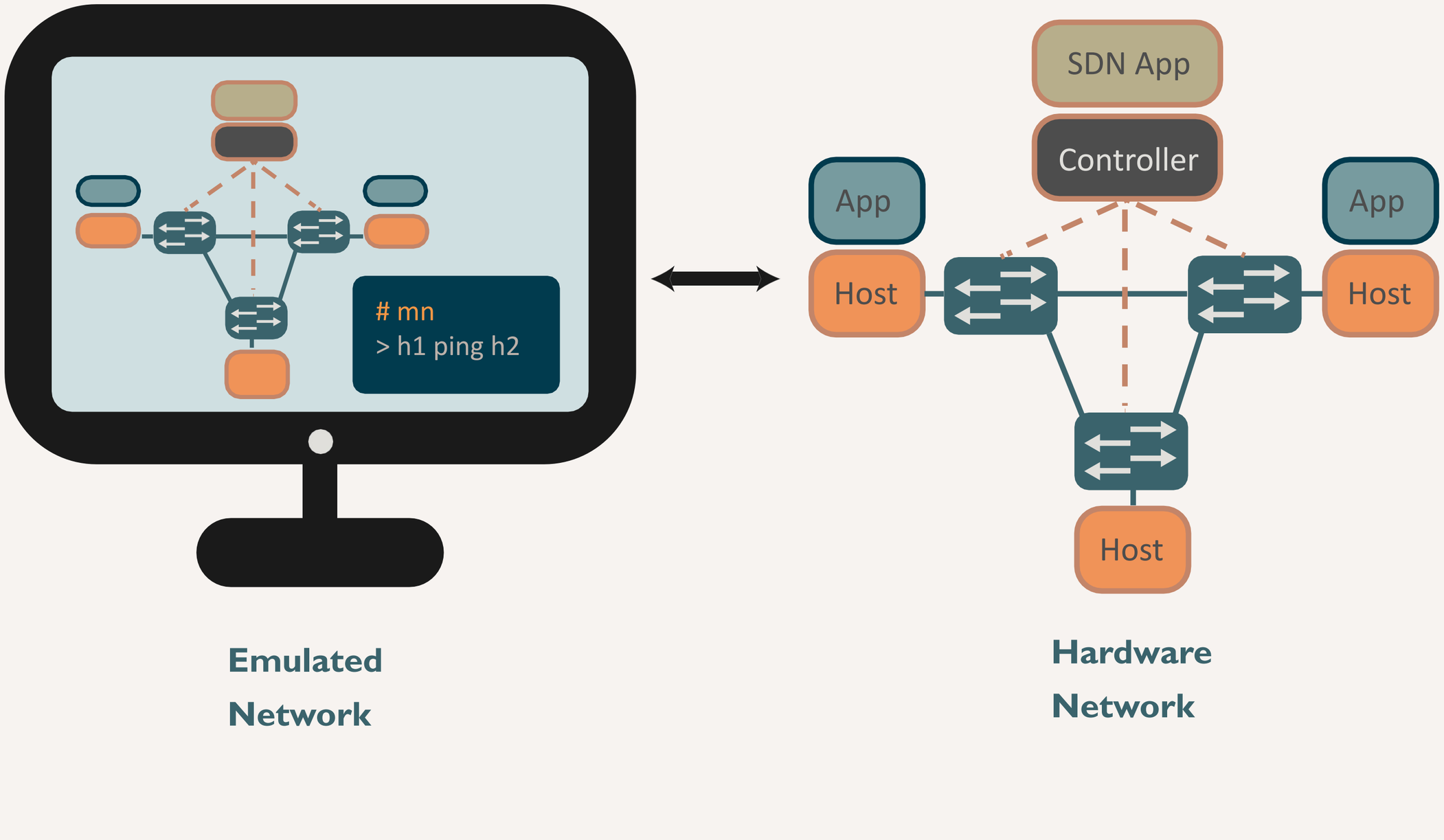
Mininet and Ryu Overview



Mininet

- A network emulator that create virtual network topology running real kernel, switch or application.
- Includes a CLI (Command Line Interface), for debugging or running network-wide tests.
- Supports arbitrary custom topologies, and includes a basic set of parametrized topologies(Linear, Tree).
- Provides a straightforward and extensible Python API for network creation and experimentation.

Network Emulator

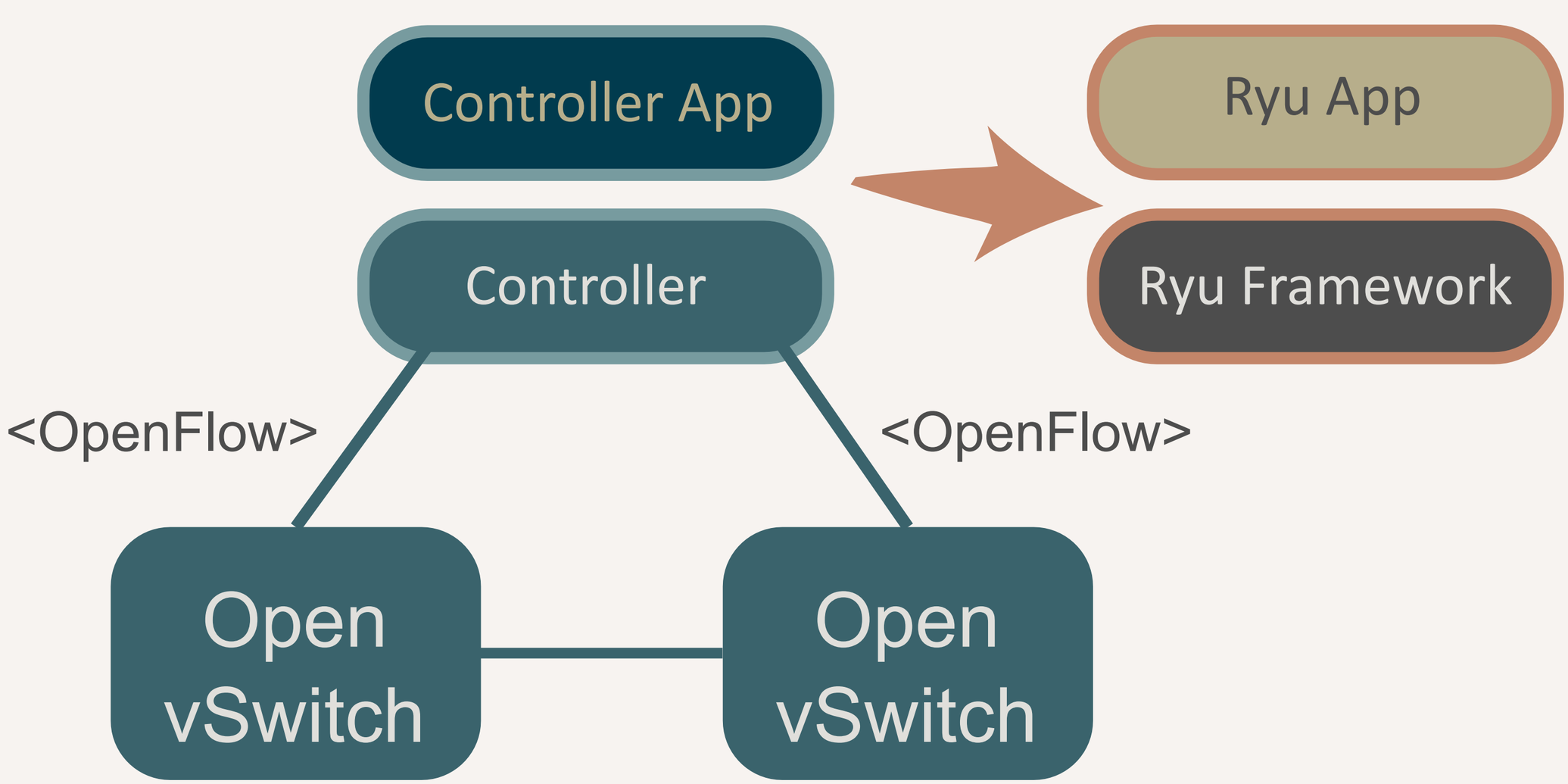


Emulated Network

Hardware Network

Ryu Overview

- Ryu can help developers to communicate between the controller and Open vSwitch by using OpenFlow protocol.

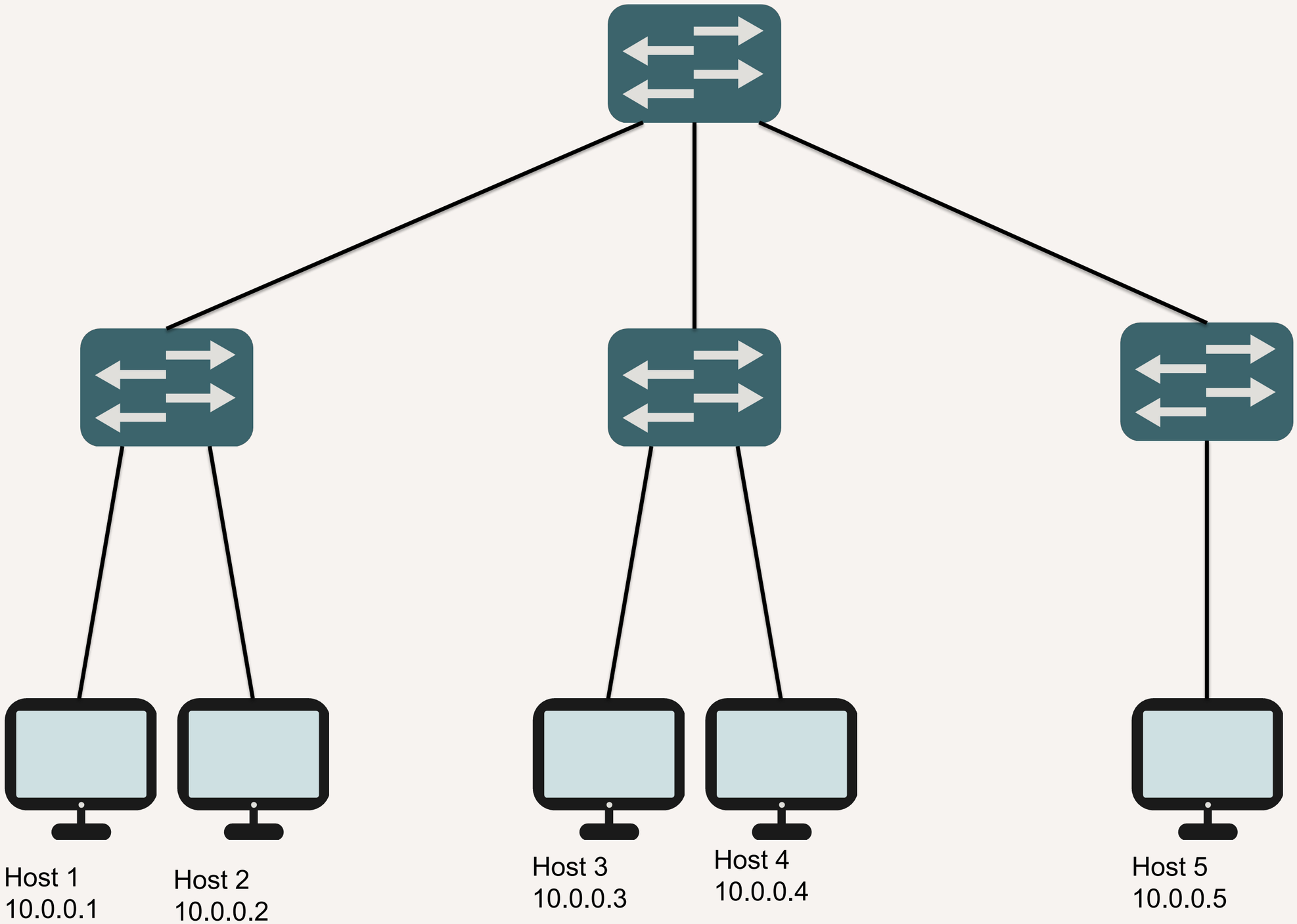


Lab Announcement

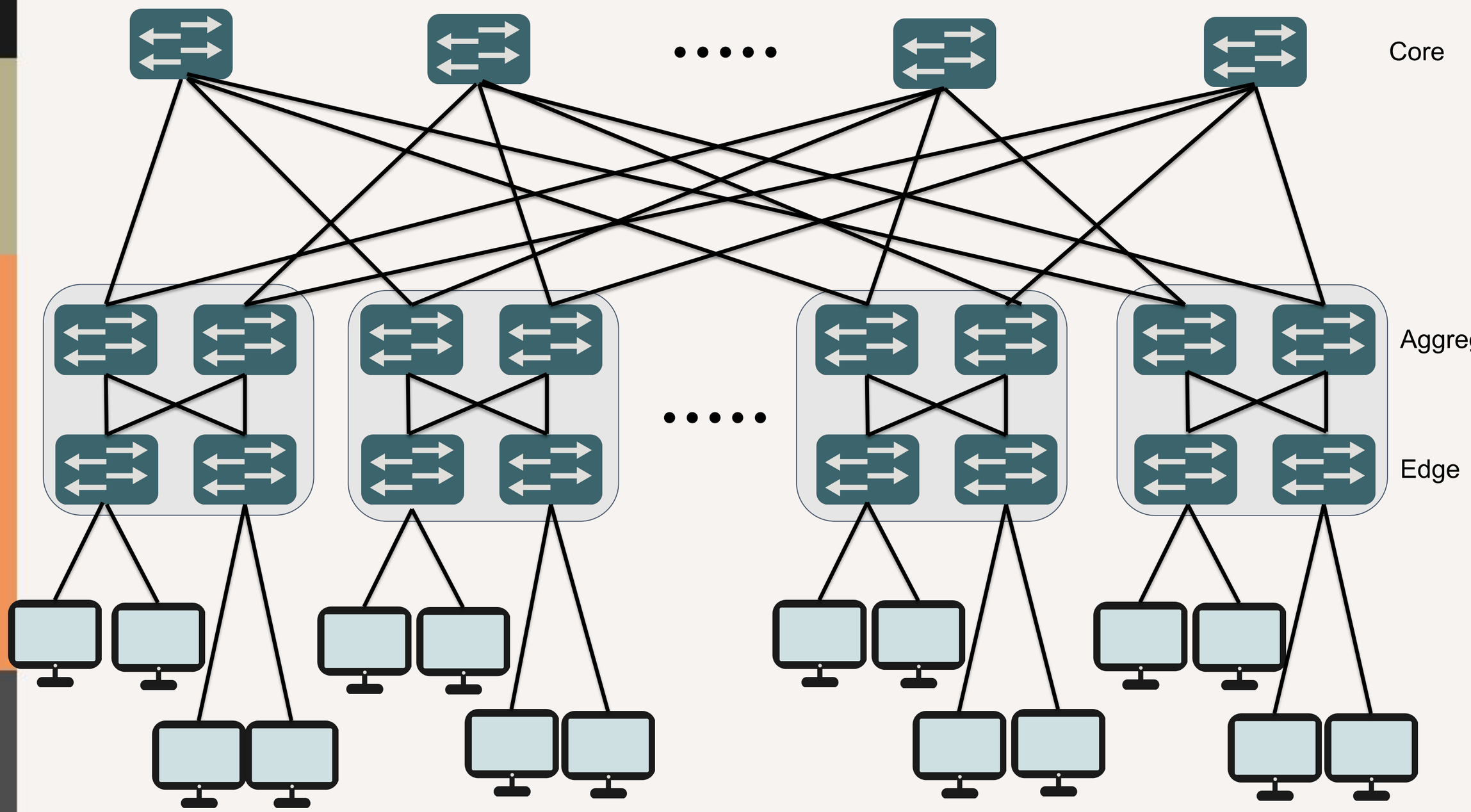
Specification

- Create your network topology by Mininet
  - A tree topology
  - A data center-like topology
- Write your SDN application by Ryu
  - Simple network slicing
  - Network monitor
  - Congestion detection
  - Congestion reaction, drop or re-routing
- Report

Mininet – Tree Topology



Mininet – Data Center-Like Topology



Mininet –Data Center-Like Topology

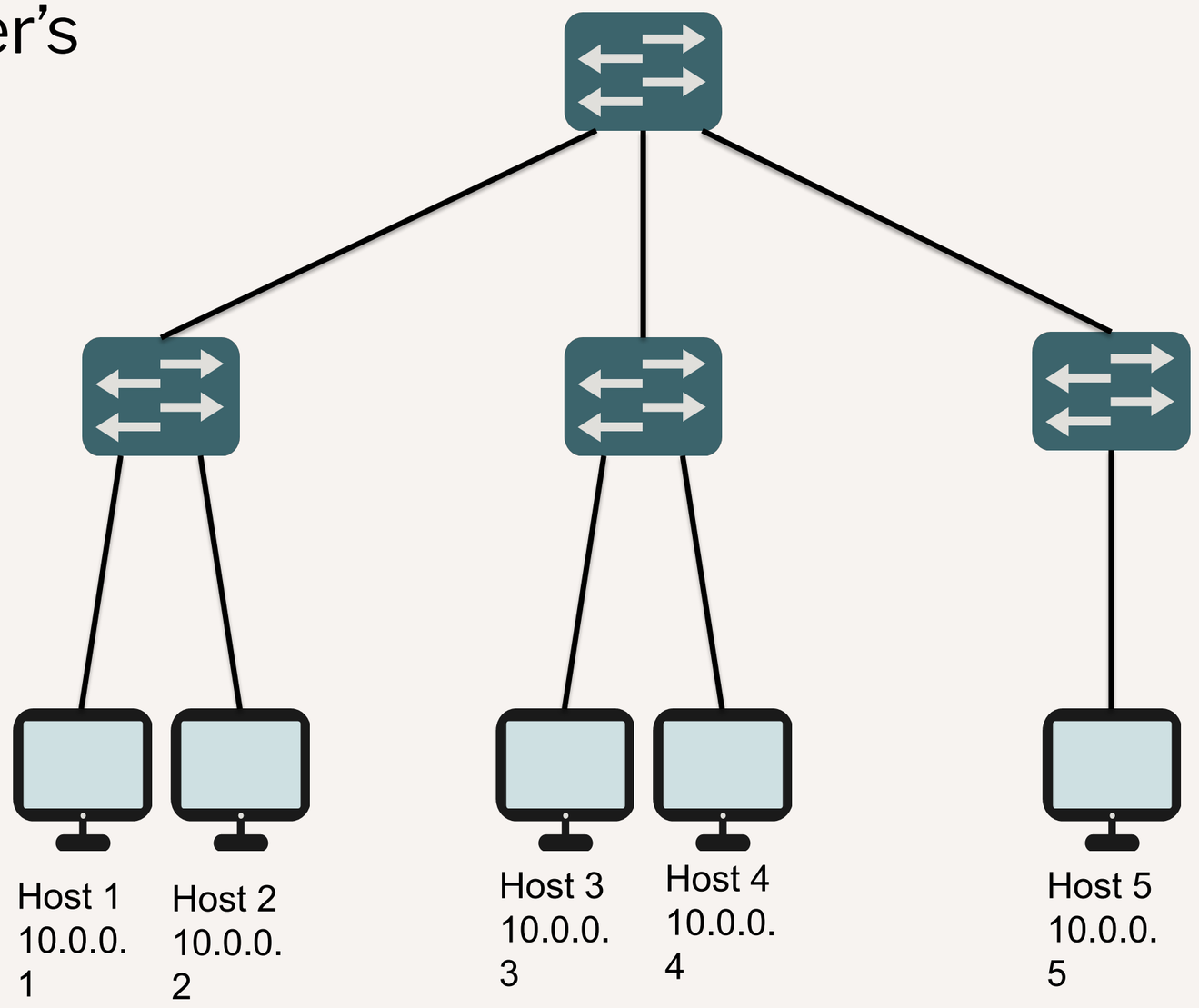
- Your mininet program should be able to receive input which specifies the number of core and aggregation switches.
- There are loops in the topology, which will cause **broadcast storm**. How to solve it?

Ryu application – Simple Network Slicing

- Network slicing is a way to share infrastructure, it overlays multiple virtual network on top of a shared network.
- Controller have the overall view of the physical network, and can install rules to “slice” the network.
- Hosts or switches in different will not know each other and therefore can not access to each other.

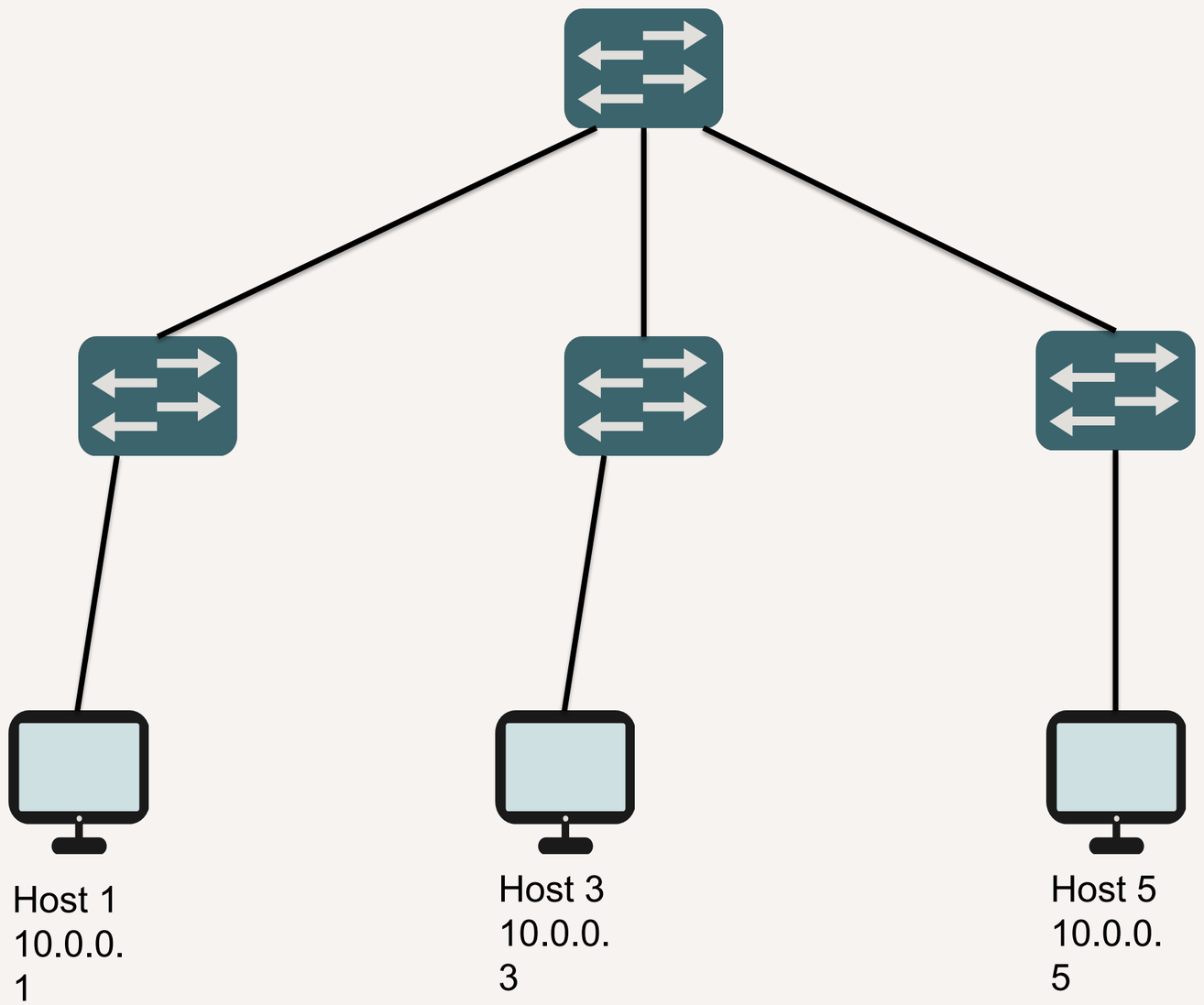
Ryu application – Simple Network Slicing

Topology from controller's view



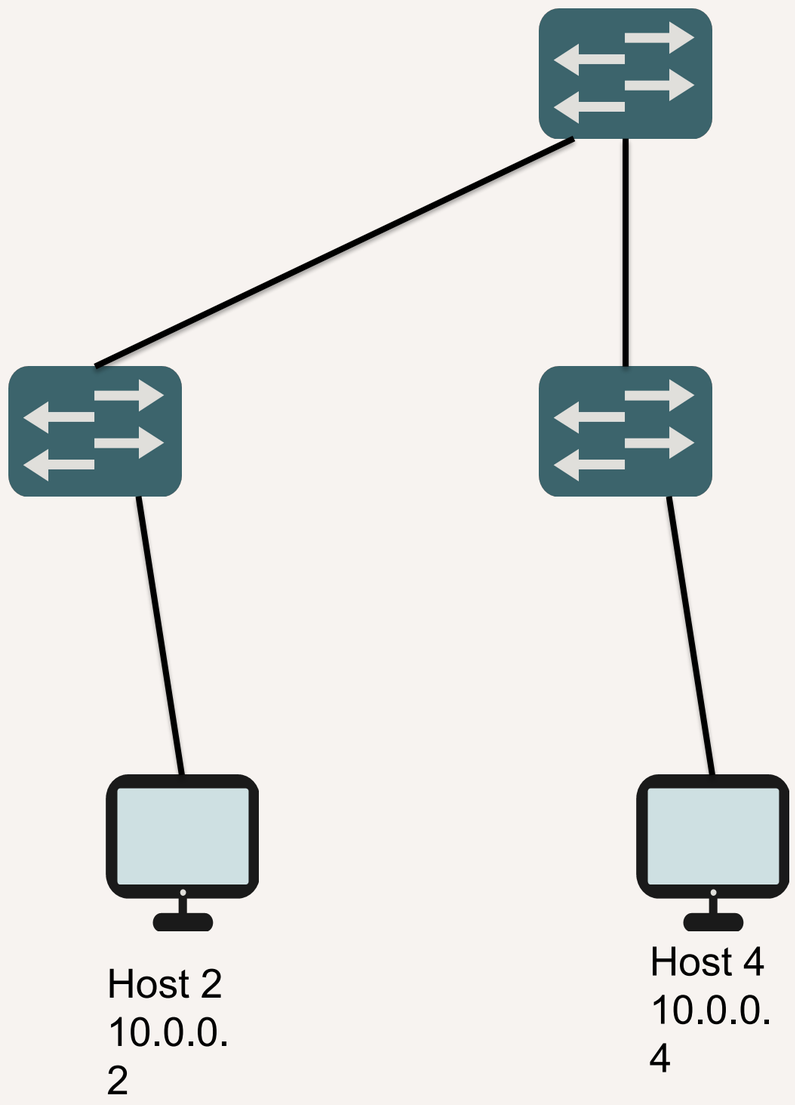
Ryu application – Simple Network Slicing

Network 1



Ryu application – Simple Network Slicing

Network 2



Ryu application – Simple Network Slicing

- To simplify your work, you can slice the network by IP address only(e.g. class 1 contains 10.0.0.<odd number>, class 2 contains 10.0.0.<even number>)
- In this task, you can use destination MAC address to transfer packets and flood packet to unknown hosts(learning switch).

Ryu application – Network Monitor

You need to periodically send request to switch and receive port and flow statistical information from switches.

- Port statistical information
  - send/receive packet count
  - send/receive byte count
  - drop count
  - error count
- Flow statistical information
  - in-port
  - match fields
  - action
  - match packet/byte count

Ryu application – Network Monitor

- You need to print out the statistical information received by your application.
- Below is a example format for printing our, you can use arbitrary format to show the information
- How to define a flow?

```
Flow Statistical Information
datapath         in-port  match src_ip src_port       dst_ip dst_port  protocol action   packets  bytes
---------------- -------- ------------------------------------------------- -------- -------- --------
0000000000000001        2     10.0.0.2    33868     10.0.0.1     5001       TCP  dropped        4      296
0000000000000001        2     10.0.0.2    33872     10.0.0.1     5001       TCP  dropped        3      222
0000000000000001        3     10.0.0.3    58212     10.0.0.1     5001       TCP   port 1      106  1055572
0000000000000001        1     10.0.0.1     5001     10.0.0.3    58212       TCP   port 3       81     5346
Port Statistical Information
datapath         port     rx-pkts  rx-bytes tx-pkts  tx-bytes dropped  error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       97     6454      158  1060678        0        0
0000000000000001        2       22     1616       50     4990        0        0
0000000000000001        3      150  1060100      116     9068        0        0
0000000000000001 fffffffe        0        0        0        0       24        0
```

Ryu application – Congestion Detection

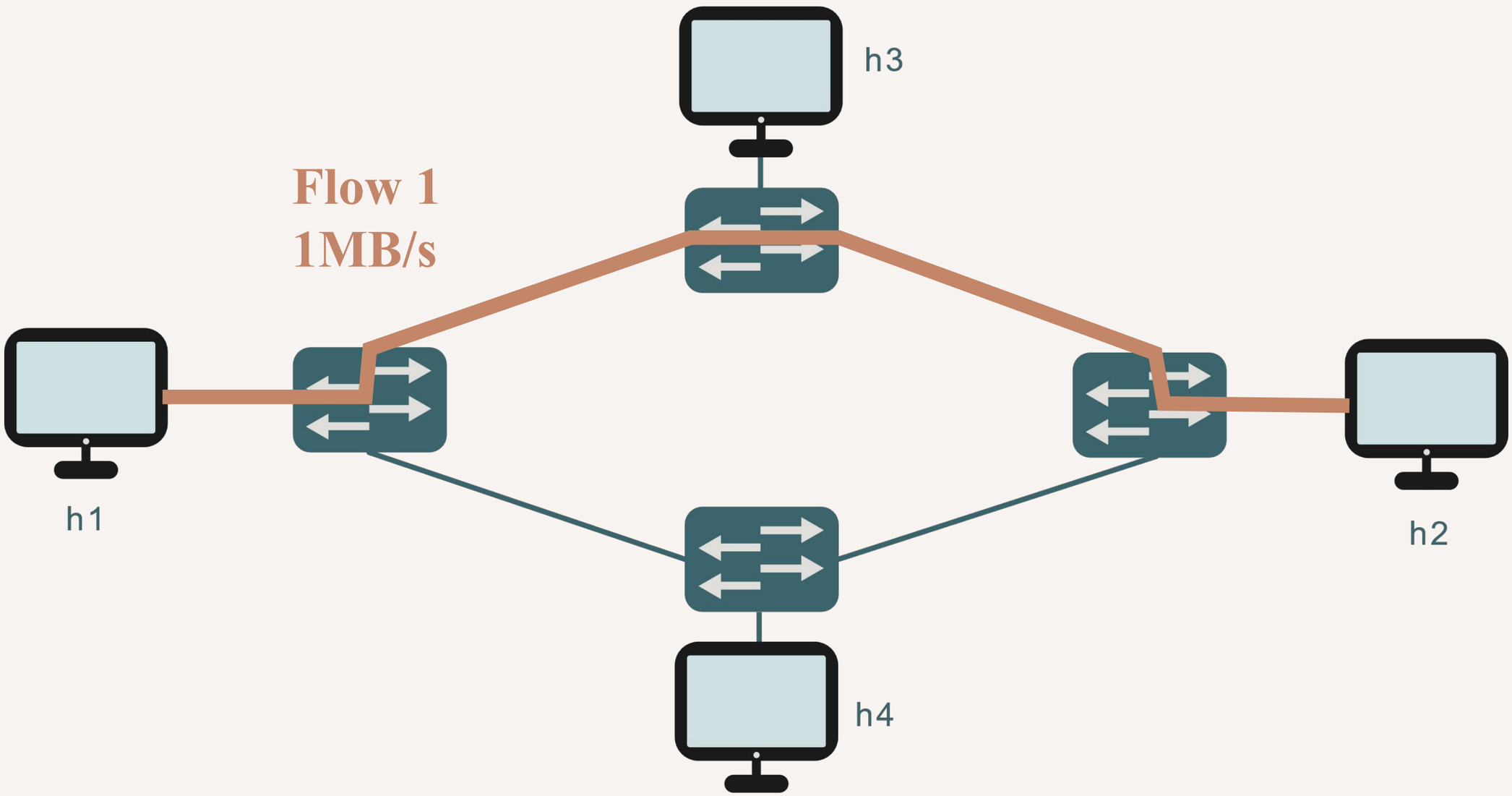
- Base on your network monitor, now you can detection a congestion on a port of a switch
- You can calculate the delta value by the information received this time and previous time
- Once a congestion occurred(e.g. 1MB/s), your application should detect it and print out a alert message

Ryu application – Congestion Reaction

- In addition to detecting a congestion when occured, now you should react to this congestion, you can either
  - Drop the big flow
  - Re-route the big flow
- In this task, you must identify a flow by 5-tuple field
  - source IP address(e.g. 10.0.0.1)
  - source port(e.g. 5001)
  - destination IP address(e.g. 10.0.0.3)
  - destination port(e.g. 33872)
  - Protocol(e.g. TCP, UDP)

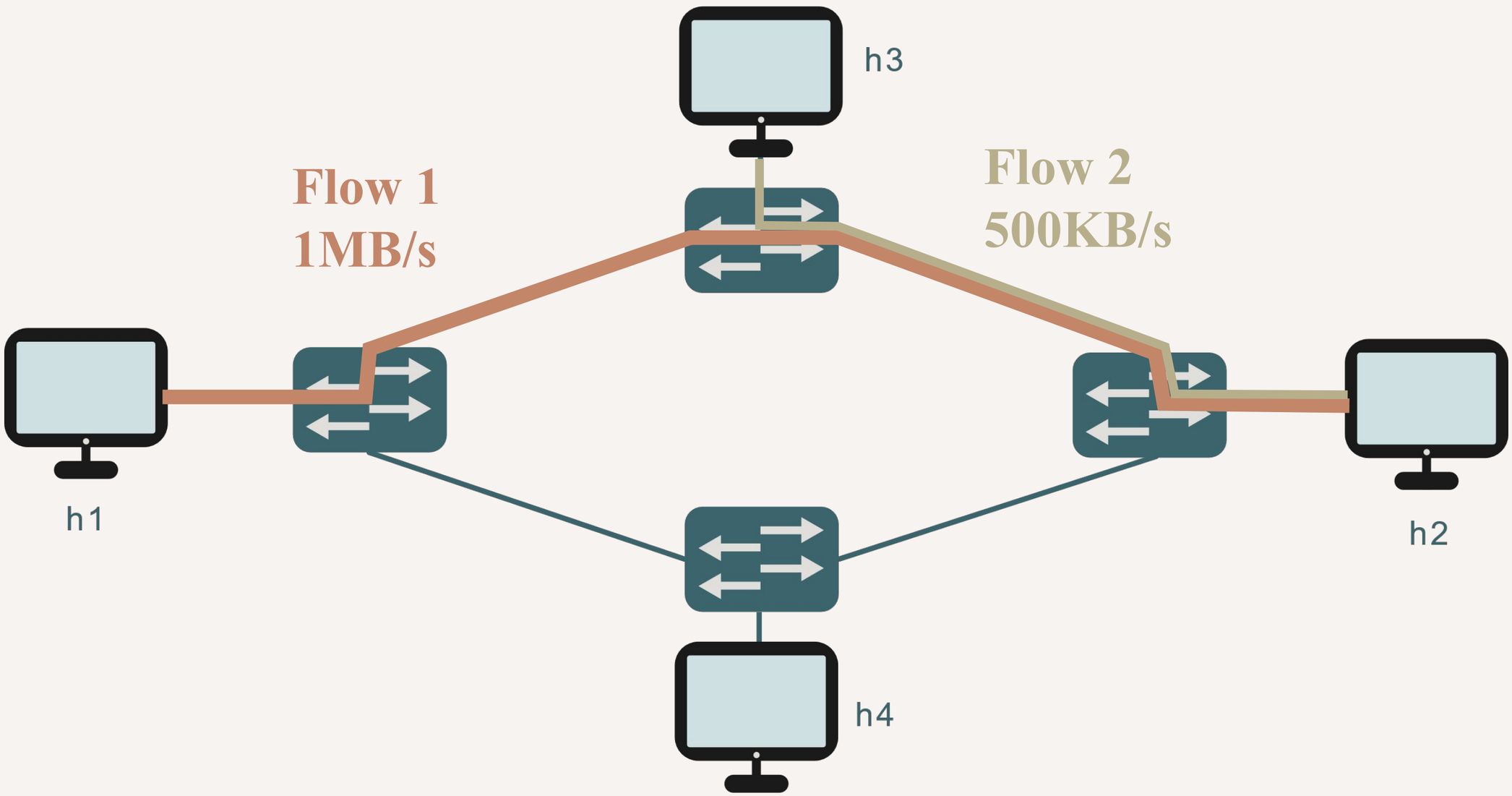
Ryu application – Congestion Example

- This is the path from h1 to h2 originally. We can send packets from h1 to h2.



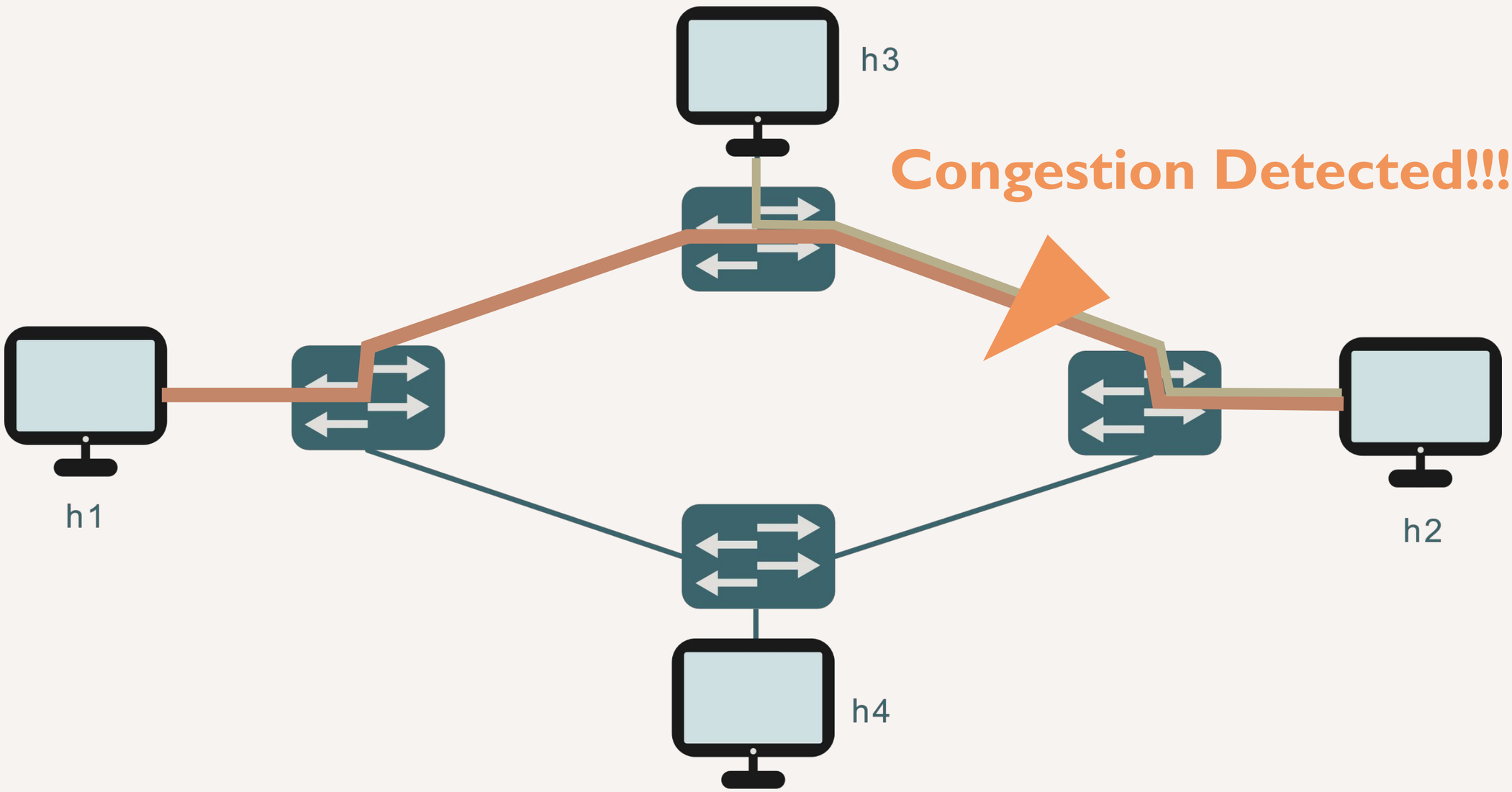
Ryu application – Congestion Example

- However, there may be others packets on the path from h1 to h2.



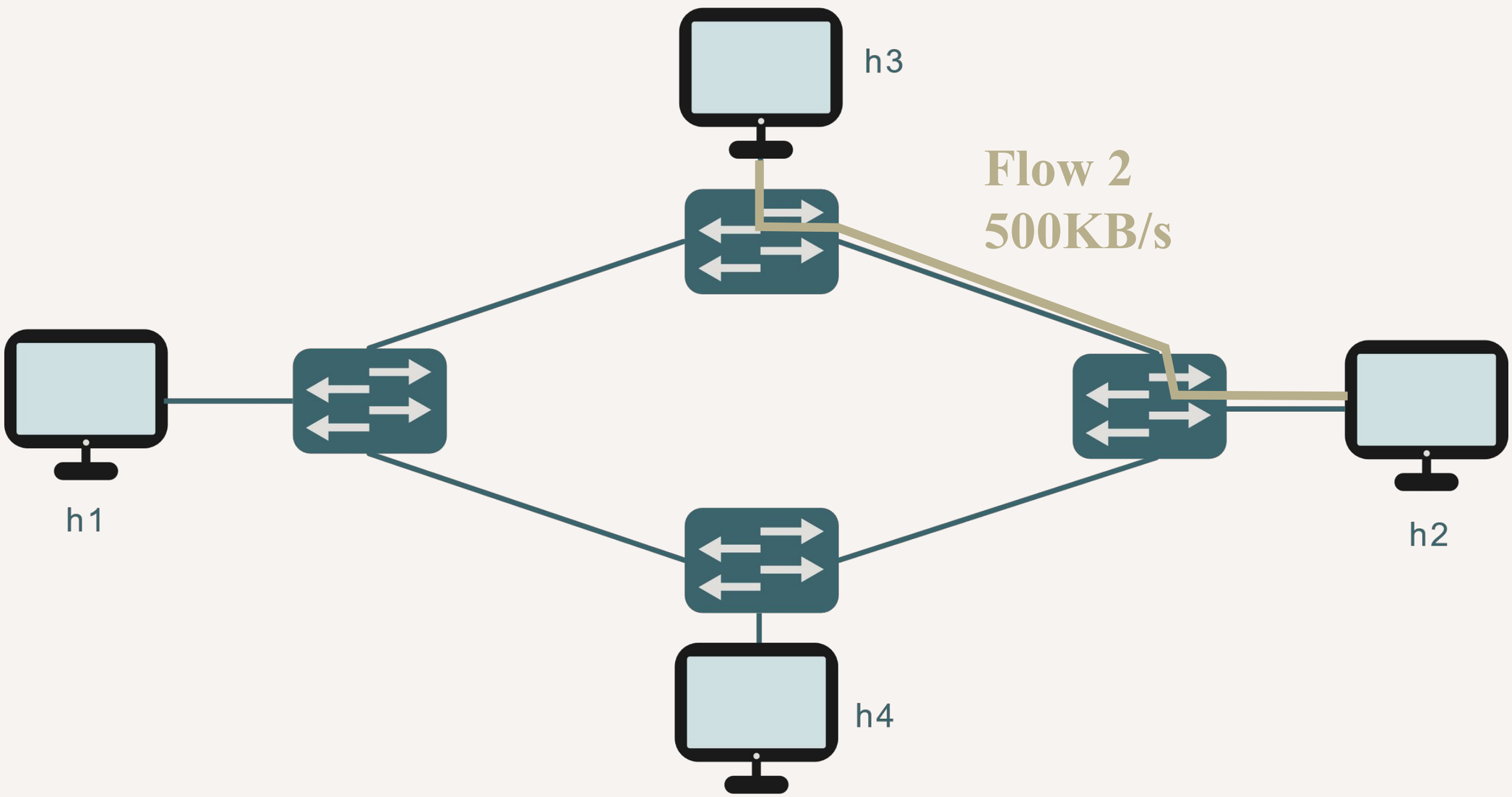
Ryu application – Congestion Example

- While congestion occurs, your application should be able to detect it. (20%)



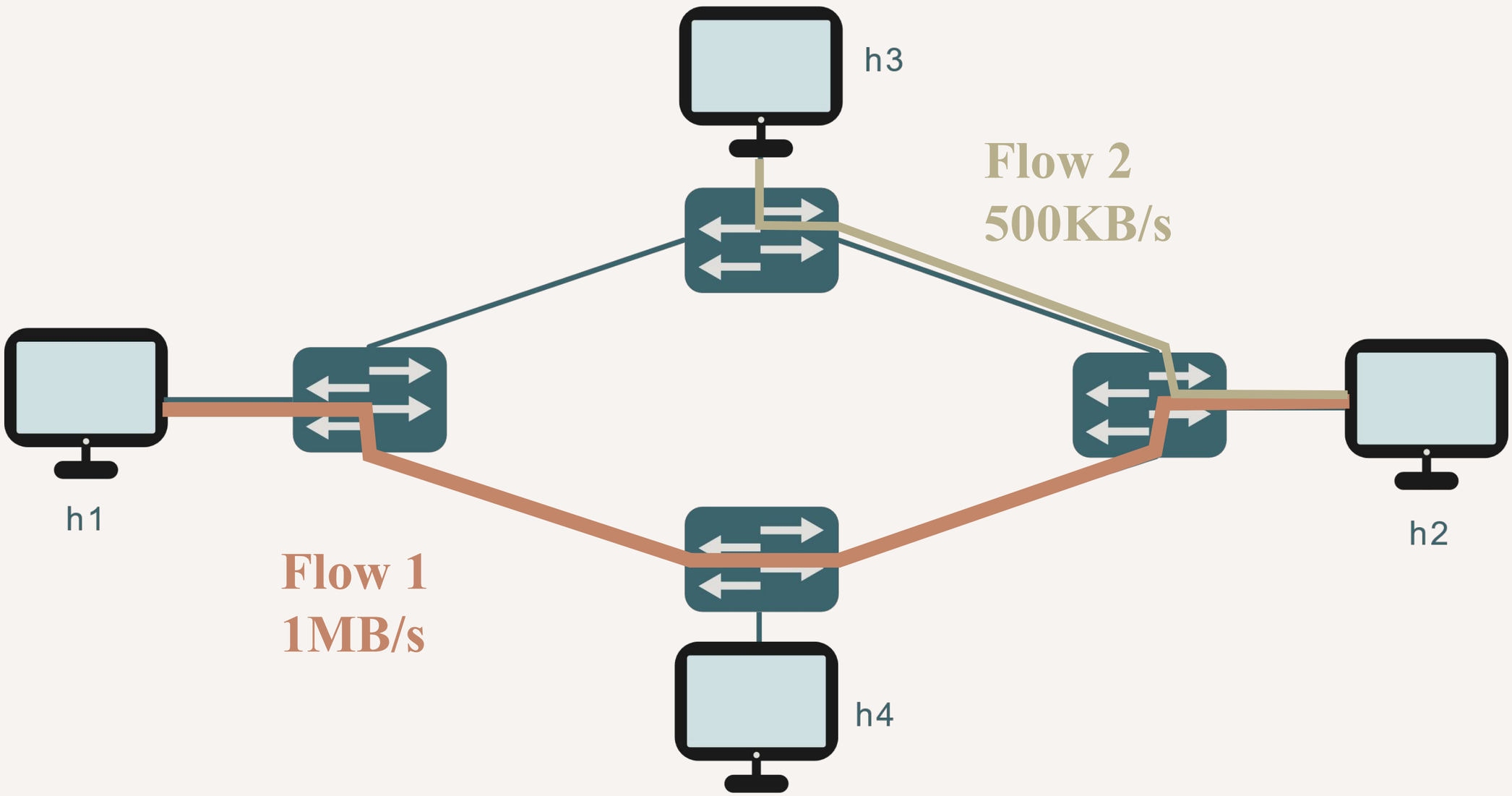
Ryu application – Congestion Example

- Your application can drop the big flow.



Ryu application – Congestion Example

- Your application can re-route the big flow.



Grading Policy

- **Mininet (25%)**
  - A tree topology (15%)
  - A data center like topology (10%)
- **Ryu application (50%)**
  - Simple network slicing (8% + 2%)
  - Network monitor (8% + 2%)
  - Congestion detection (5% + 5%)
  - Congestion reaction - drop (5% + 5%)
  - Congestion reaction - re-route (10%)
- **Report (25%)**

Report

- **Environment (5%)**
- **Explain the pros and cons that there are loops in a network topology. (10%)**
- **Explain the broadcast storm and how you handle it in this lab. Is there any better solution to handle broadcast storms under SDN? If yes, explain how; if no, explain why. (10%)**

Deadline

- **5 / 27 Demo**
- **6 / 3 23:59:59**
  - Submit report and source code to NTU cool
  - Report has to be in pdf format

Mininet

Mininet mn Linux Commands

- This command needs superuser privilege(sudo).
- Basic CLI

Command Line	Description
$ sudo mn	Start a minimal topology
$ exit	Exit mininet
$ sudo mn -c	Exit and free the resource

Mininet CLI Commands

- Display Mininet CLI commands

```
$ mininet> help
```

- Display nodes

```
$ mininet> nodes
```

- Display links

```
$ mininet> net
```

- Dump information about all nodes

```
$ mininet> dump
```

Mininet mn Linux Commands

- -h, --help # Show the help message.
- --controller # Define what controller to use.
- --topo # Tell mininet what topology to use.
- --custom # Read custom topology and node parameter from .py file.
- -x, --xterm # Spawn an xterm window for all nodes inside the network.
- --mac # The mininet will assign static MAC addr. for each host.

Mininet CLI Commands

- Xterm display

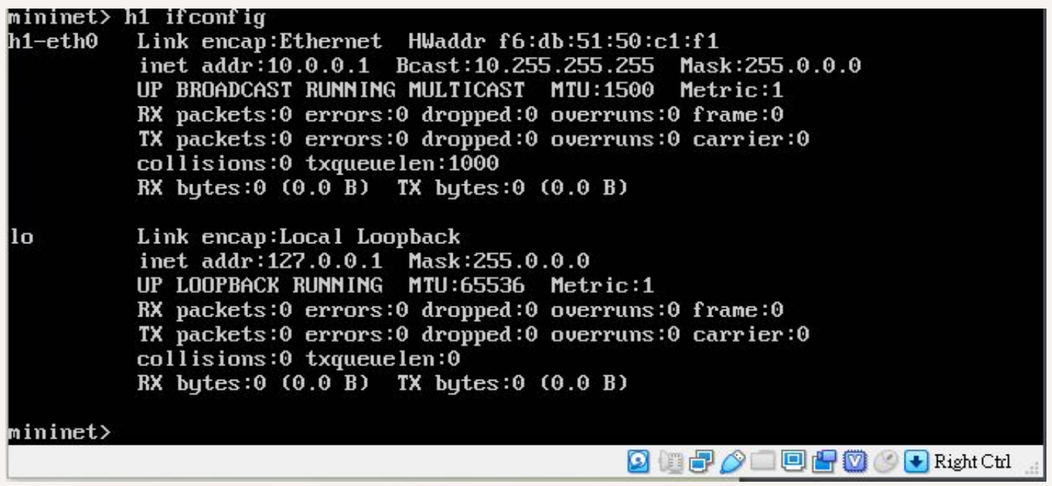
```
$ mininet> xterm h1
```

- Ping between all hosts

```
$ mininet> pingall
```

- <node> <command>
  - Ex1:
  - Ex2:

```
$ mininet> h1 ifconfig
$ mininet> h1 ping h2
```



views on the mininet from host1

Xterm on Mininet CLI

```
$ sudo mn --topo single,3
$ mininet> xterm h1
```

```
[2019-01-22 21:21.16]  ~
[p1974.DESKTOP-HVG9A9V] ▸ ssh mininet@192.168.56.255
ssh: connect to host 192.168.56.255 port 22: Cannot assign requested address

[2019-01-22 21:22.01]  ~
[p1974.DESKTOP-HVG9A9V] ▸ ssh mininet@192.168.56.101
Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Jan 22 19:18:58 2019
mininet@mininet-vm:~$ ls
install-mininet-vm.sh  loxigen  mininet  oflops  oftest  openflow  pox
mininet@mininet-vm:~$ sudo mn --topo single,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1
mininet>
```

```
"Node: h1"@mininet-vm

root@mininet-vm:~# ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 0e:5c:95:4c:c8:4b
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:693 errors:0 dropped:0 overruns:0 frame:0
          TX packets:693 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1164676 (1.1 MB)  TX bytes:1164676 (1.1 MB)

root@mininet-vm:~#
```

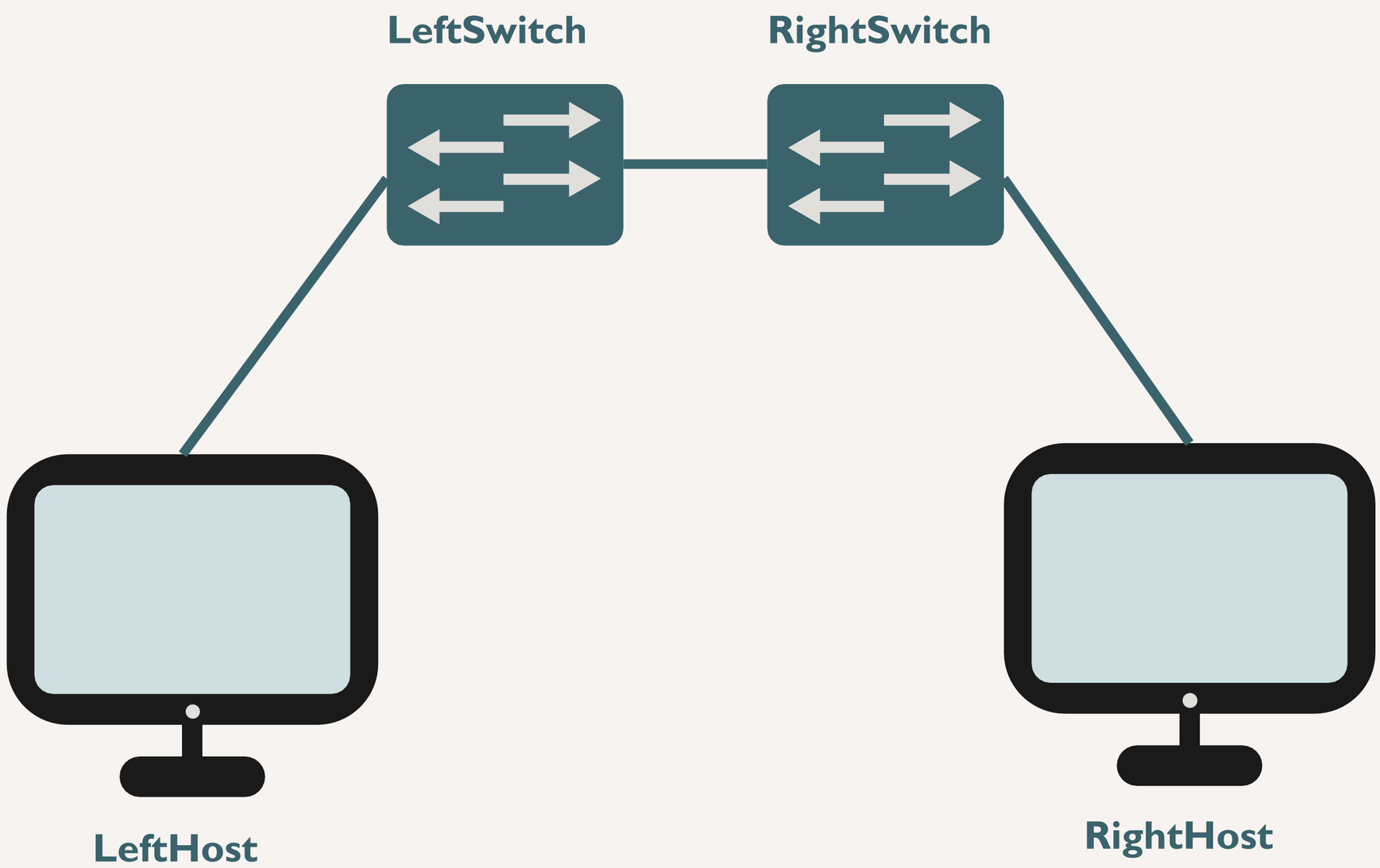
views on the mininet from host1 in Xterm

Mininet Create Custom Topology

- Use python script to create custom mininet topology.
- Some useful function
  - addHost(<host name>) // add host
  - addSwitch(<host name>) // add switch
  - addLink(<node1>,<node2>) // add link

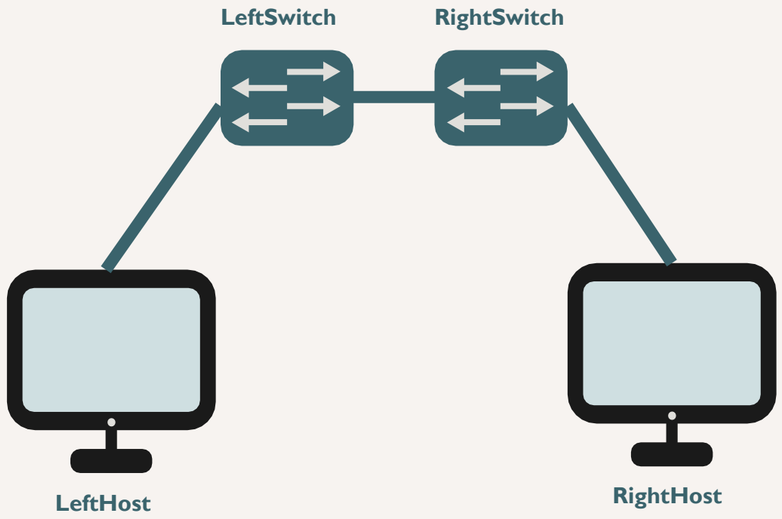
Mininet Create Custom Topology

- Example Topology



Mininet Create Custom Topology

```
from mininet.topo import Topo

class MyTopo (Topo):
    #"Simple topology example.“

    def __init__(self ):
        #“Create custom topo.“
        # Initialize topology
        Topo.__init__(self)

        # Add hosts an switches
        leftHost = self.addHost(‘h1’)
        rightHost = self.addHost(‘h2’)
        leftSwitch = self.addSwitch(‘s1’)
        rightSwitch = self.addSwitch(‘s2’)

        # Add links
        self.addLink(leftHost,leftSwitch)
        self.addLink(leftSwitch,rightSwitch)
        self.addLink(rightSwitch,rightHost)

topos = {'mytopo':(lambda: MyTopo())}
```



→ Setup Part

→ Create Topology Part

→ Crtical Part, which names your topology

Mininet Create Custom Topology

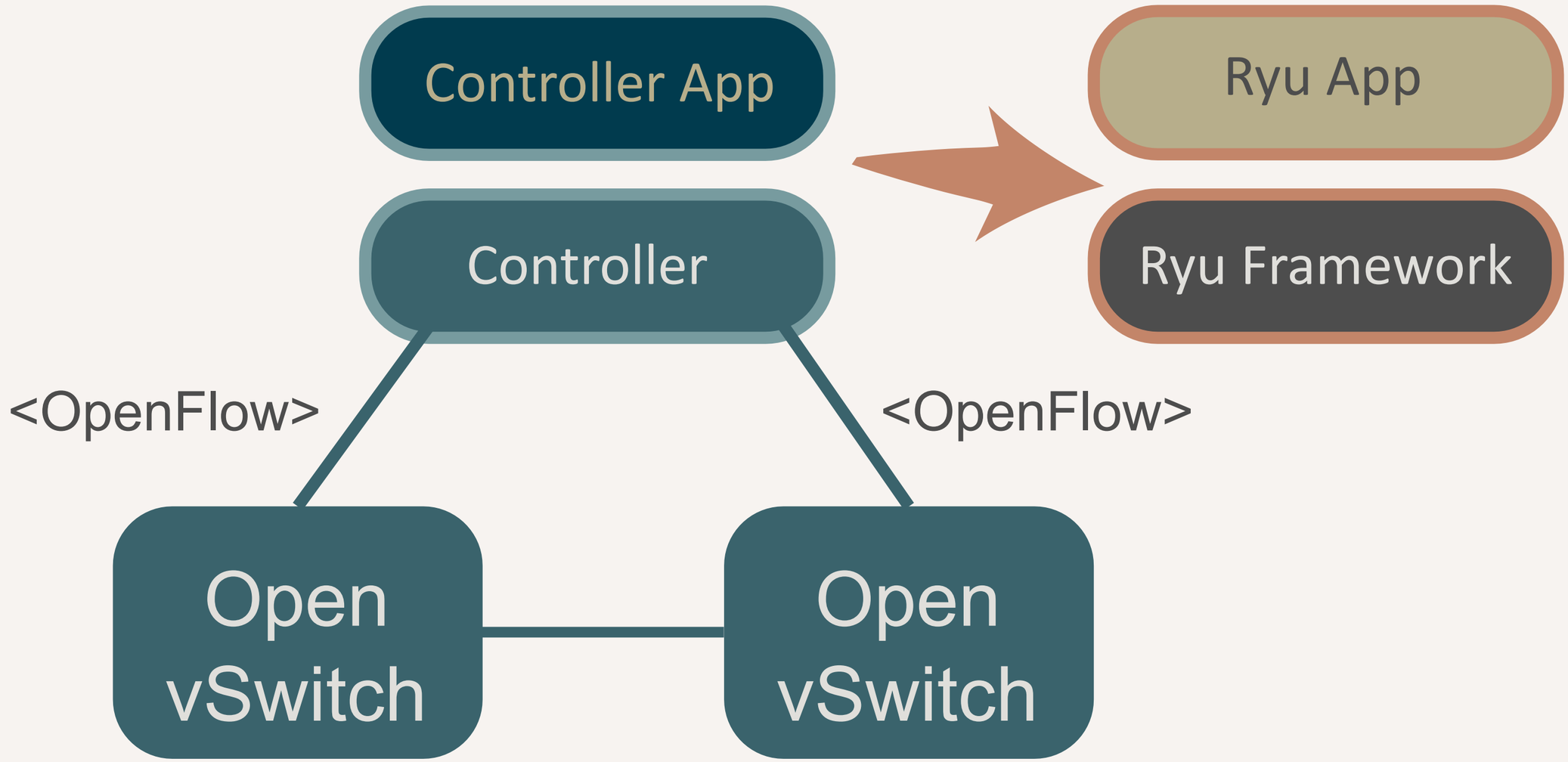
- Save the topology file in the file system
- Use the custom flag and topo flag to tell mininet what topology you want to use.

```
$ sudo mn --custom <path>/mytopology.py
--topo=mytopo
--controller=remote,ip=<host addr.>,port=<port>
```

optional

Ryu

Recall the Conception of Ryu



Installation of Ryu

```
$ sudo apt-get update
$ sudo apt-get install python-pip python-dev -y
$ sudo apt-get install python-eventlet python-routes \
                       python-webob python-paramiko -y
$ sudo pip install ryu
$ sudo pip install --upgrade six
```

Ryu L2 Learning Switch Application

- You can start with the source code of the application (link)
- Features:
  - Learning table: MAC address to port
  - Event handler
- Run the Mininet Topology

```
$ sudo mn --topo single,3 --mac --switch=ovsk,\
  protocols=OpenFlow13 --controller remote
```

- Run the Ryu Application

(Login VM with another xterm windows or Ctrl+Alt+F2 to tty2)

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

Ryu Class Definition

- Must inherit the “app_manager.RyuApp”

```
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 = {}
```

OpenFlow Messages- PacketIn

- A way for the switch to send a captured packet to the controller.
- Occurs on two circumstances:
  - Base on some rule from switch.
  - From a miss in the match tables, so the switch send a PacketIn by default.

OpenFlow Messages- PacketOut

- The response of a PacketIn from SDN controller
- The controller has the ability to inject packets into the data plane of a particular switch by PacketOut message, which can either
  - carry a raw packet to inject into the switch, or
  - indicate a local buffer on the switch containing a raw packet to release.

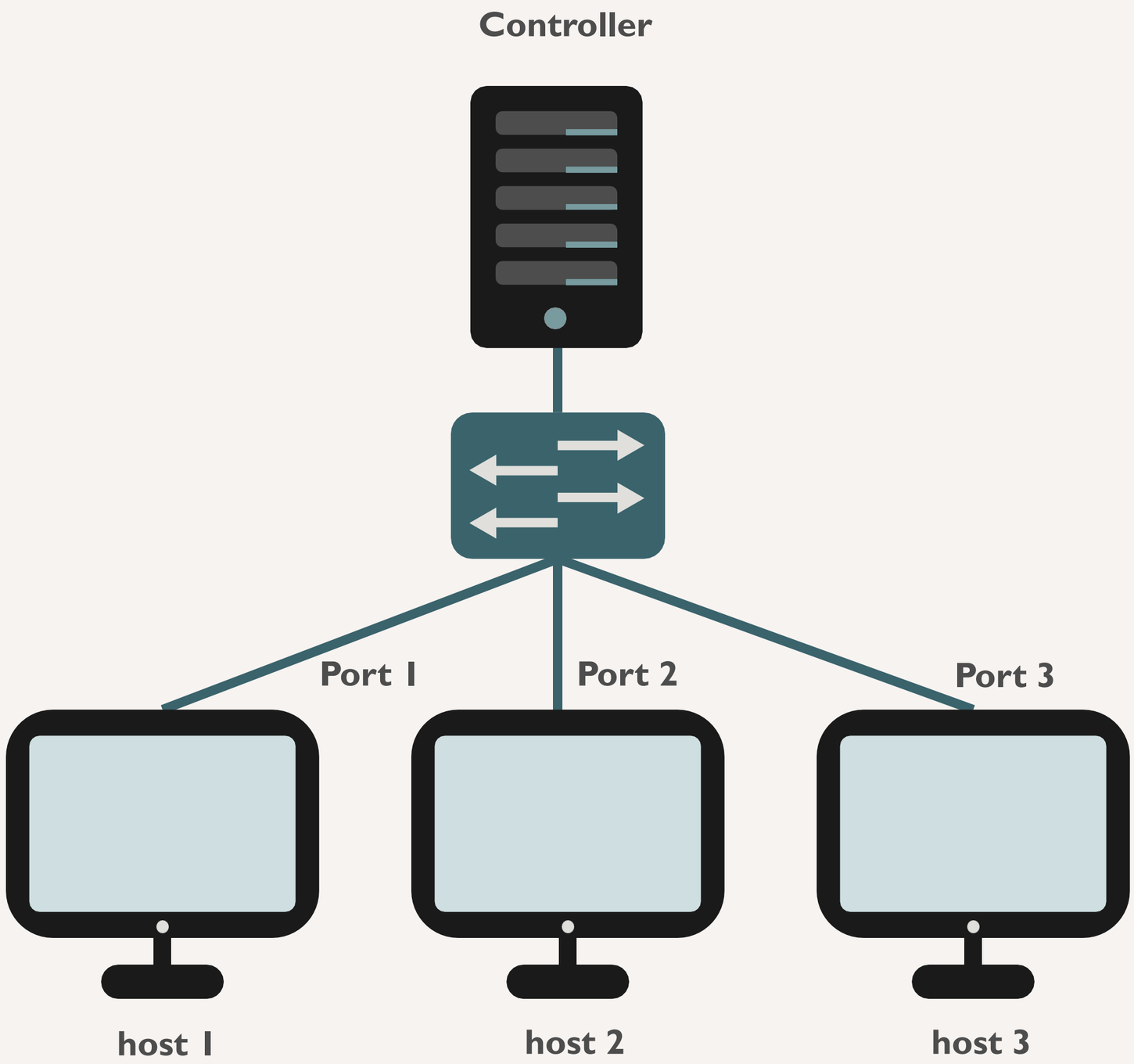
OpenFlow Messages

- FeatureRes

  The FeatureRes is the switch's reply to the controller enumerating its abilities.

- FlowMod
  - One of the main messages, it allows the controller to modify the state of an OpenFlow switch.

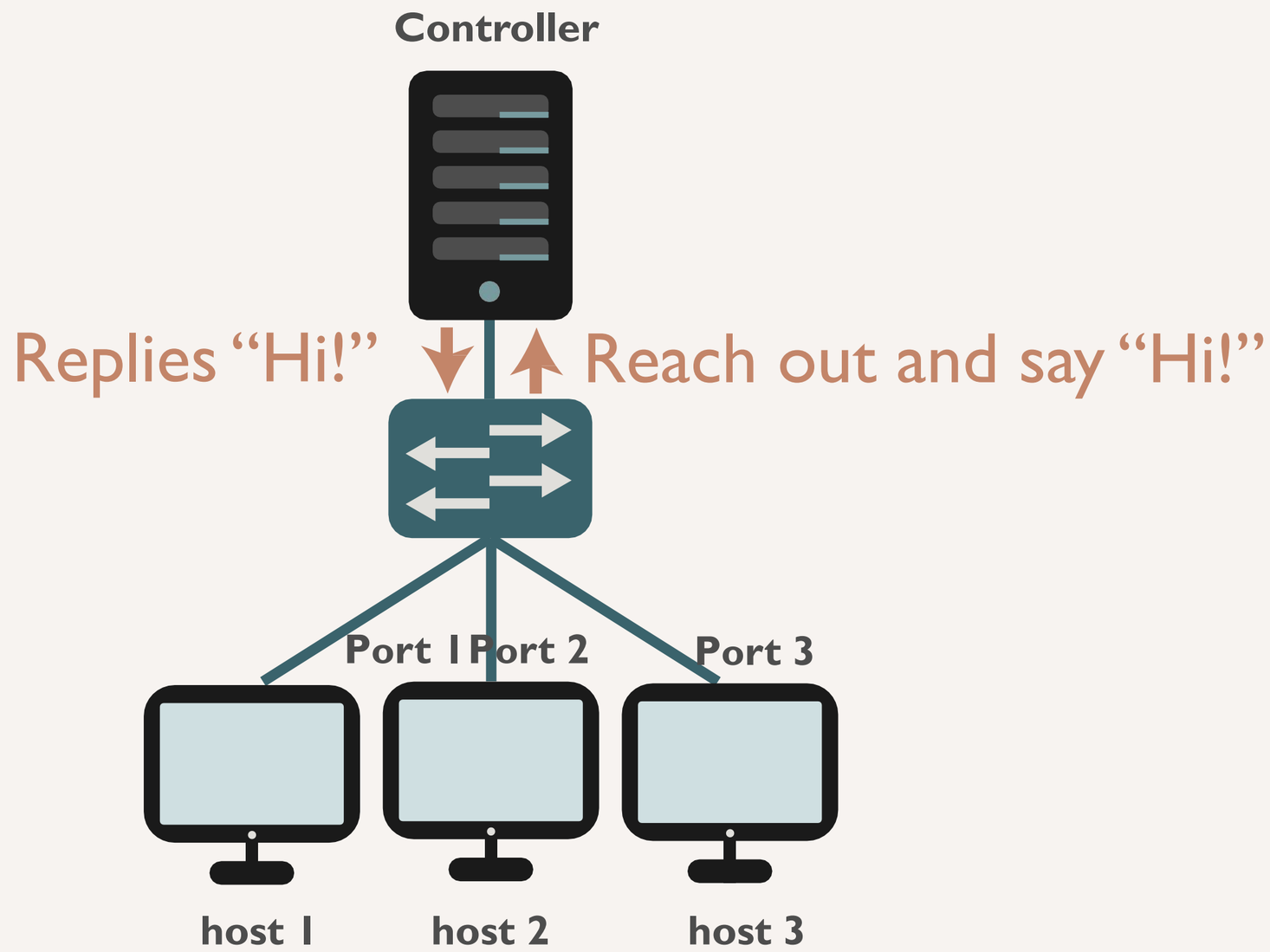
An Example of the Process



An Example of the Process

– Step 1: Handshaking

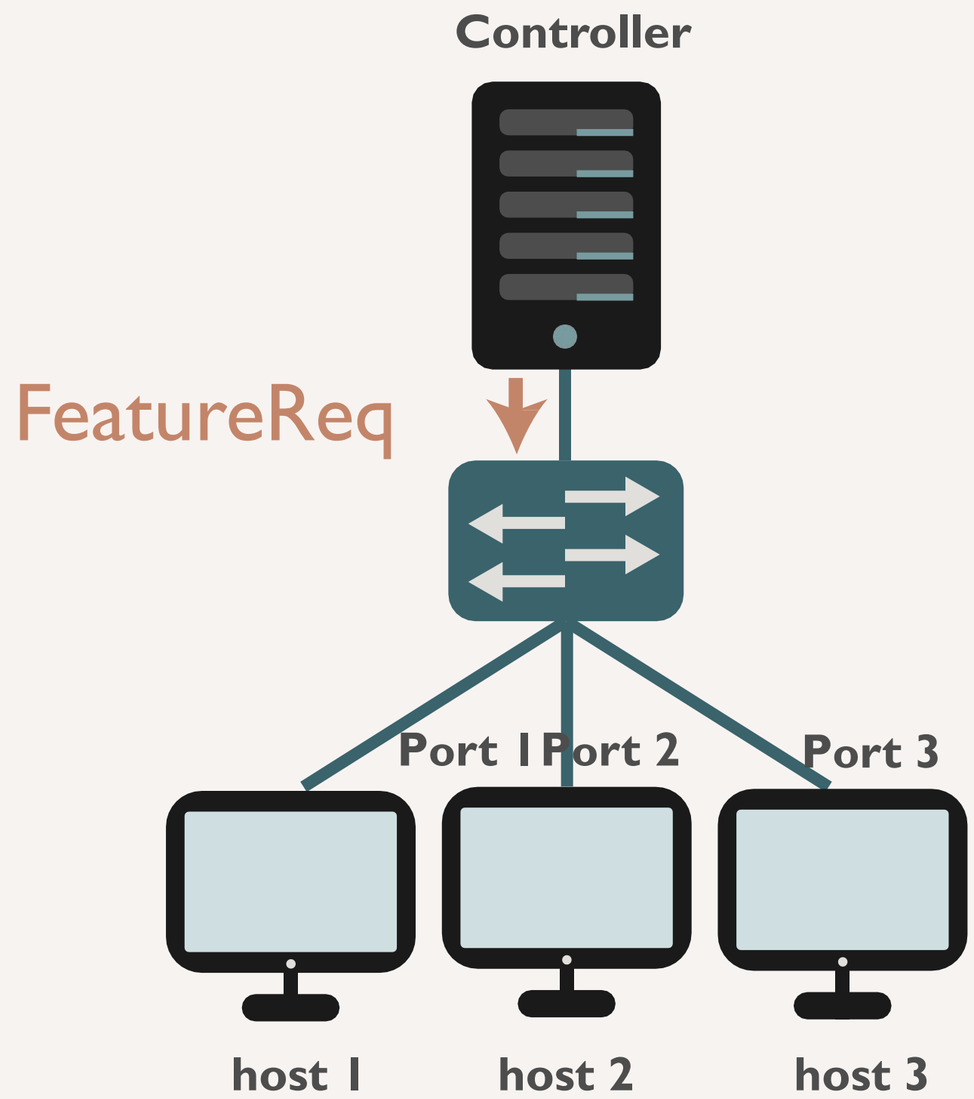
- Controller received hello message from the switch and then made the response. This has been handled by Ryu framework.



An Example of the Process

– Step 2:Featuring

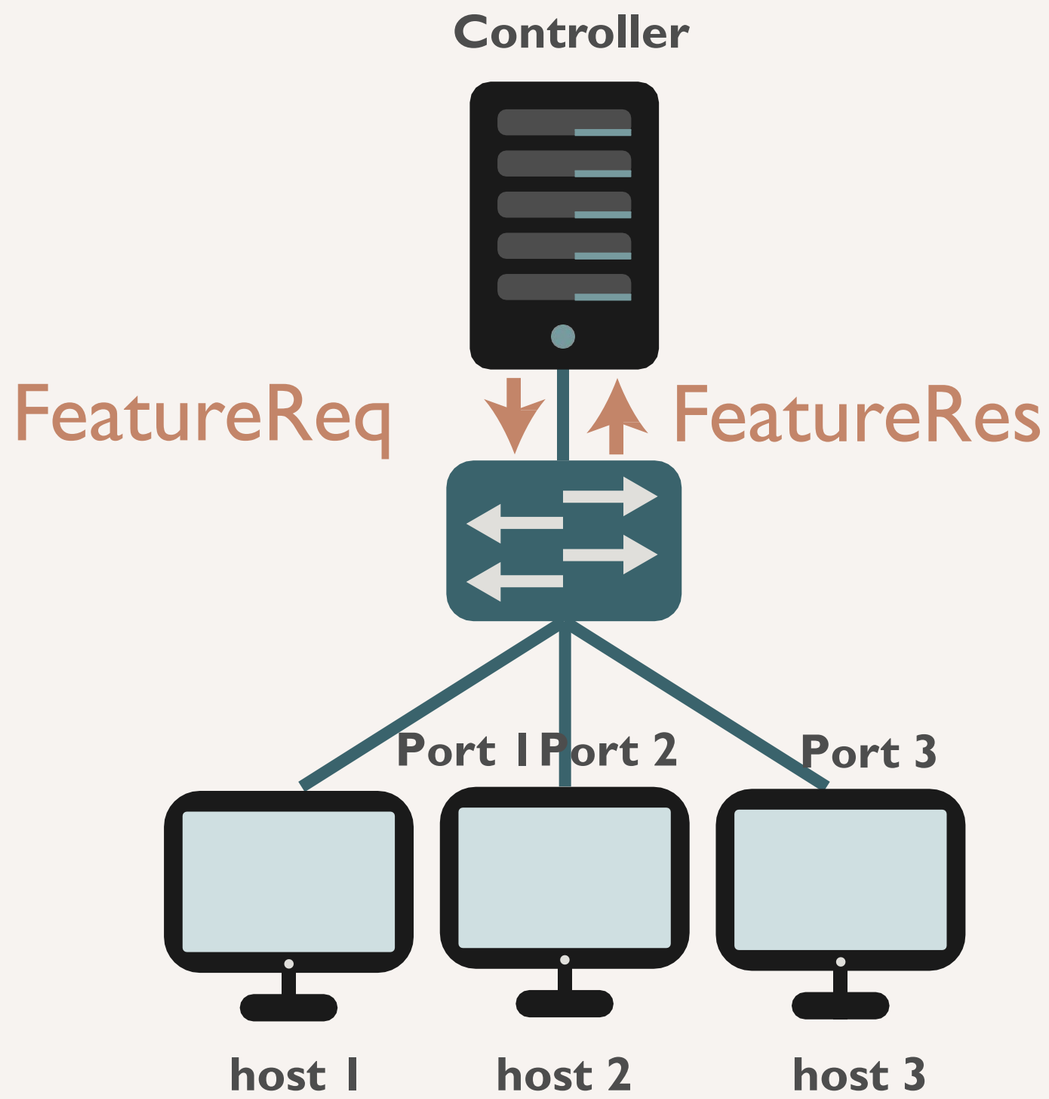
- Controller sent feature request to the switch to get its capabilities.



An Example of the Process

– Step 2:Featuring

- Switch sent feature response to the controller and then controller add the lowest priority rule to the switch. (table miss entry: go to the controller)



An Example of the Process

– Step 2:Featuring

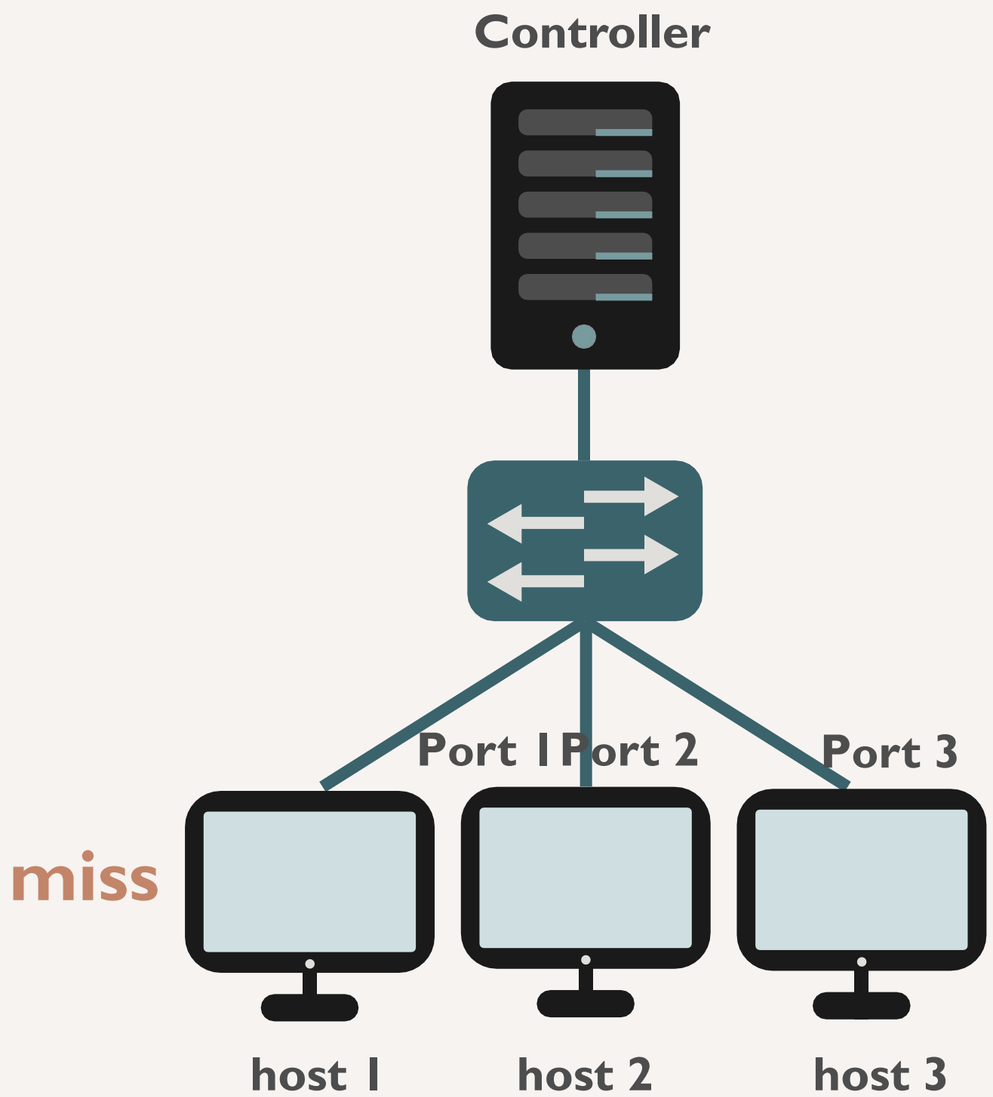
- Switch sent feature response to the controller and then controller add the lowest priority rule to the switch.

```
@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)
        #....
```

An Example of the Process

– Step 3: ARP table miss

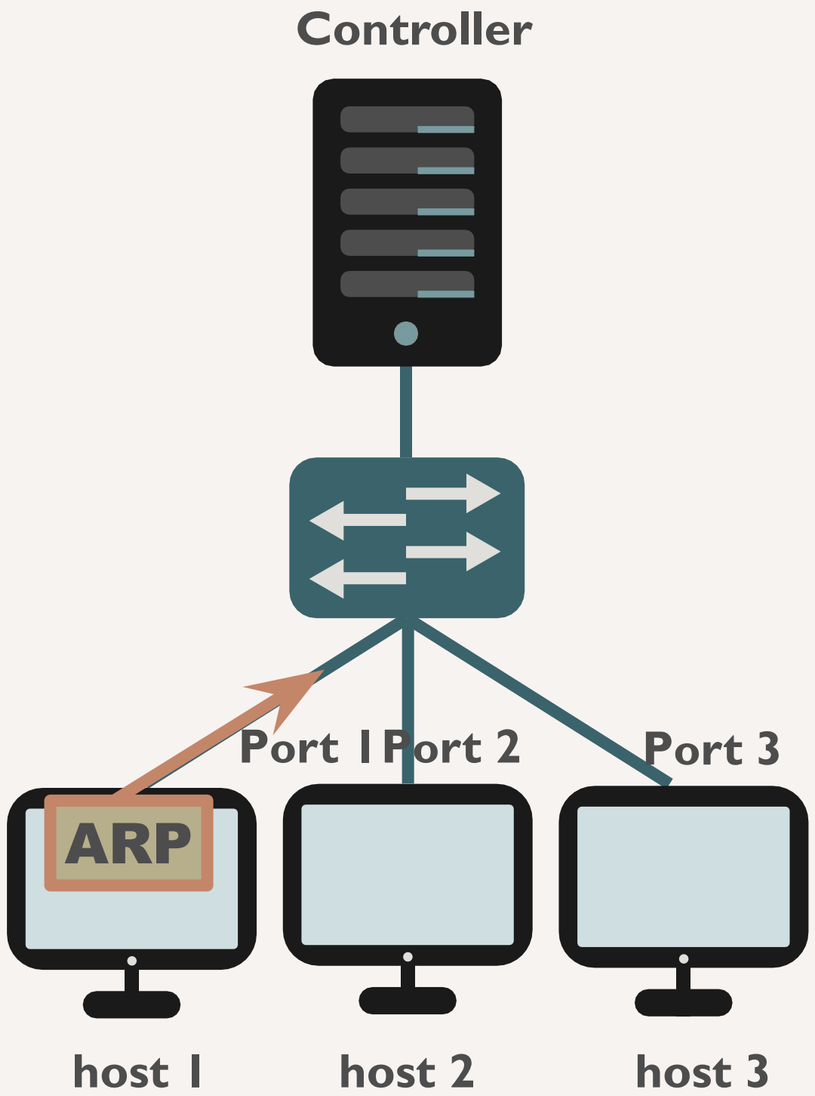
- When a host (host 1) ping another host (host 3), it will trigger a table miss on Network card if the packet hasn't been seen before.



An Example of the Process

– Step 3: ARP table miss

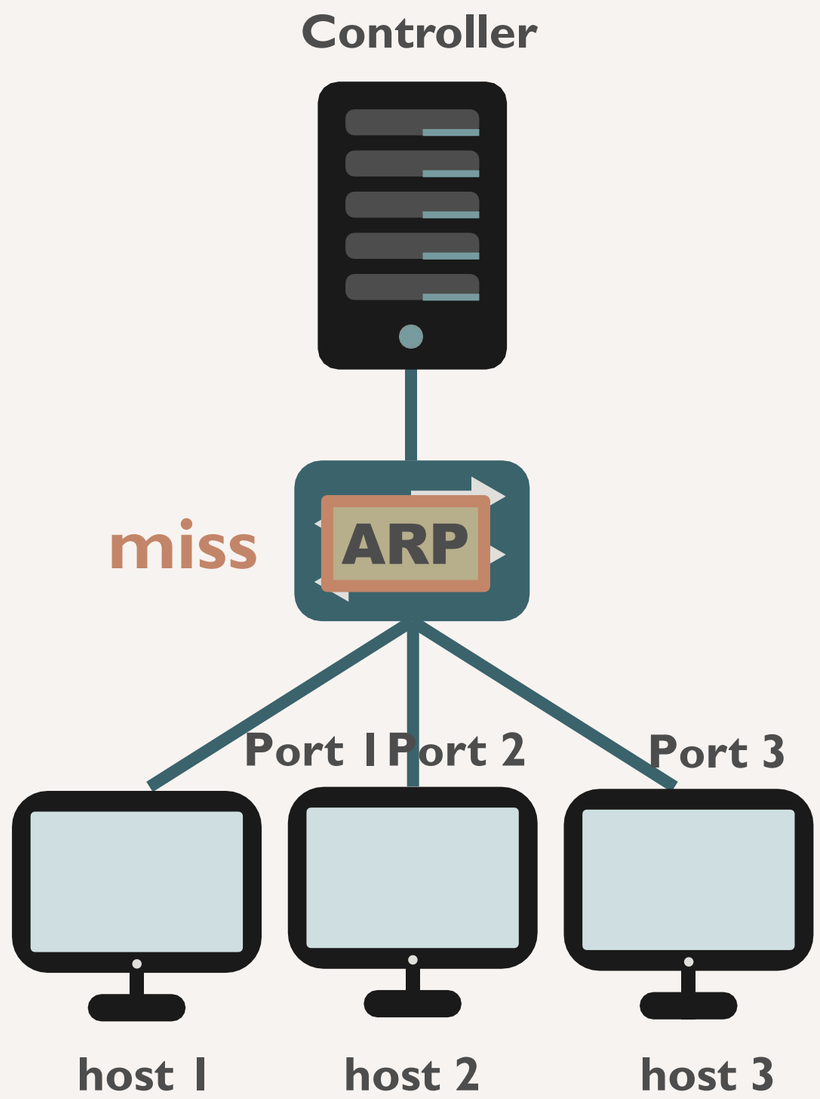
- Then host 1 will broadcast an ARP request to get the MAC address of host 3.



An Example of the Process

– Step 3: ARP table miss

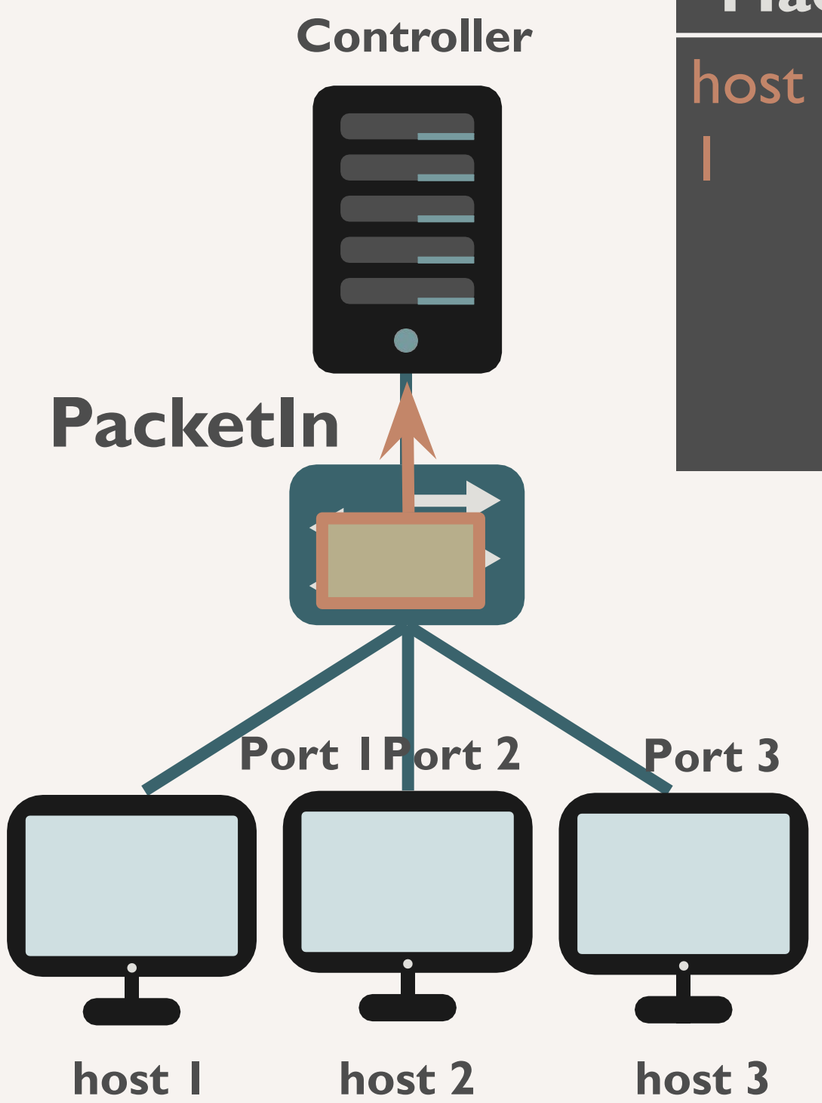
- It will trigger a table miss because the ARP request hasn't been seen by the switch before.



An Example of the Process

– Step 4: PacketIn

- Thus, the switch will sent a PacketIn to the controller.



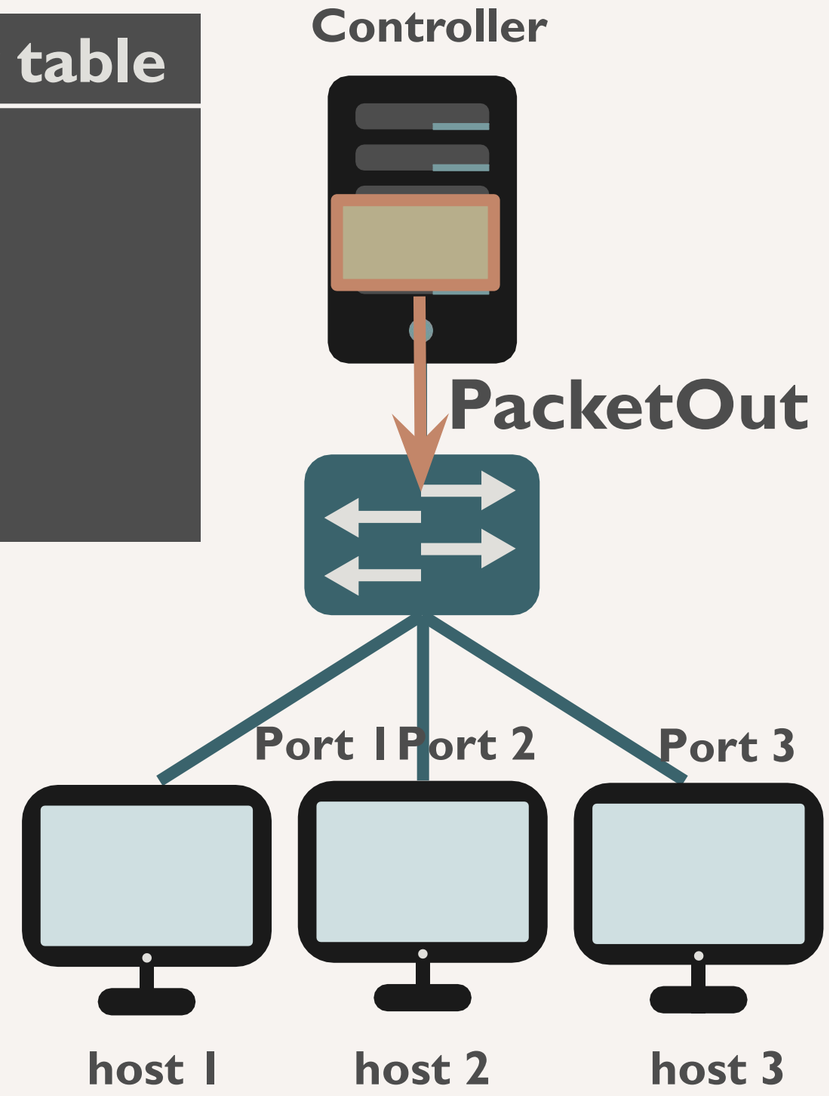
Mac table	Flow table
host 1: Port 1	

An Example of the Process

– Step 5: PacketOut

- Controller learned the source host and then it flooded the packet by PacketOut to broadcast the ARP request.

Mac table	Flow table
host 1: Port 1	



An Example of the Process

– Step 5: PacketOut

```
def _packet_in_handler(self, ev):
    #...
    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 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)
        self.add_flow(datapath, 1, match, actions)
    #...
```

Learn the mac addr. of in_port.

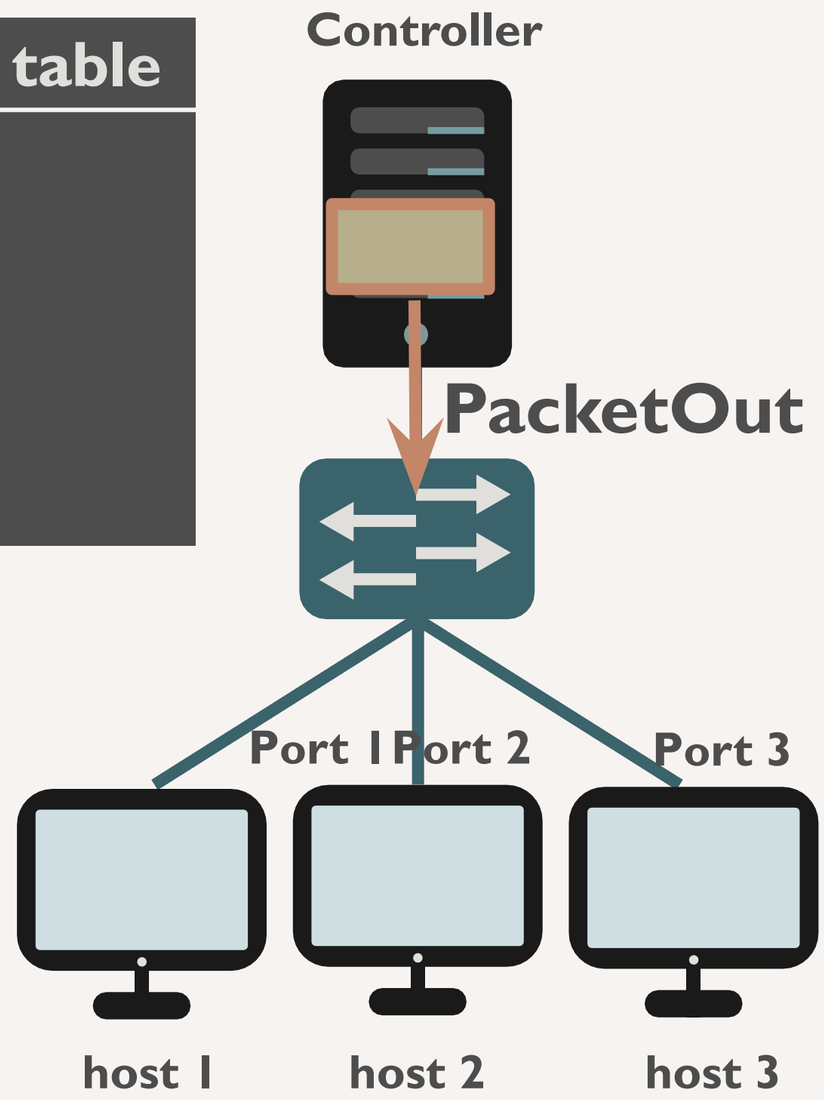
Flood the packet.

An Example of the Process

– Step 5: PacketOut

- Controller learned the source host and then it flooded the packet by PacketOut to broadcast the ARP request.

Mac table	Flow table
host 1: Port 1	

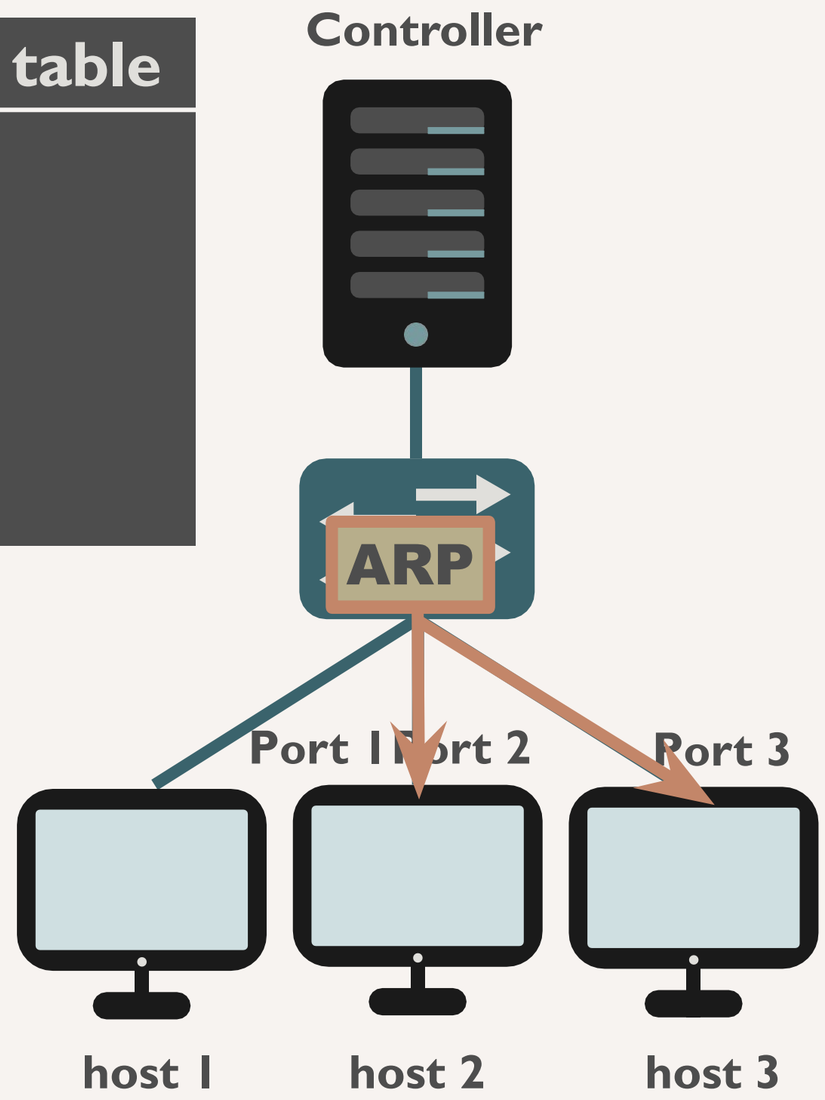


An Example of the Process

– Step 5: PacketOut

- Controller learned the source host and then it flooded the packet by PacketOut to broadcast the ARP request.

Mac table	Flow table
host 1: Port 1	

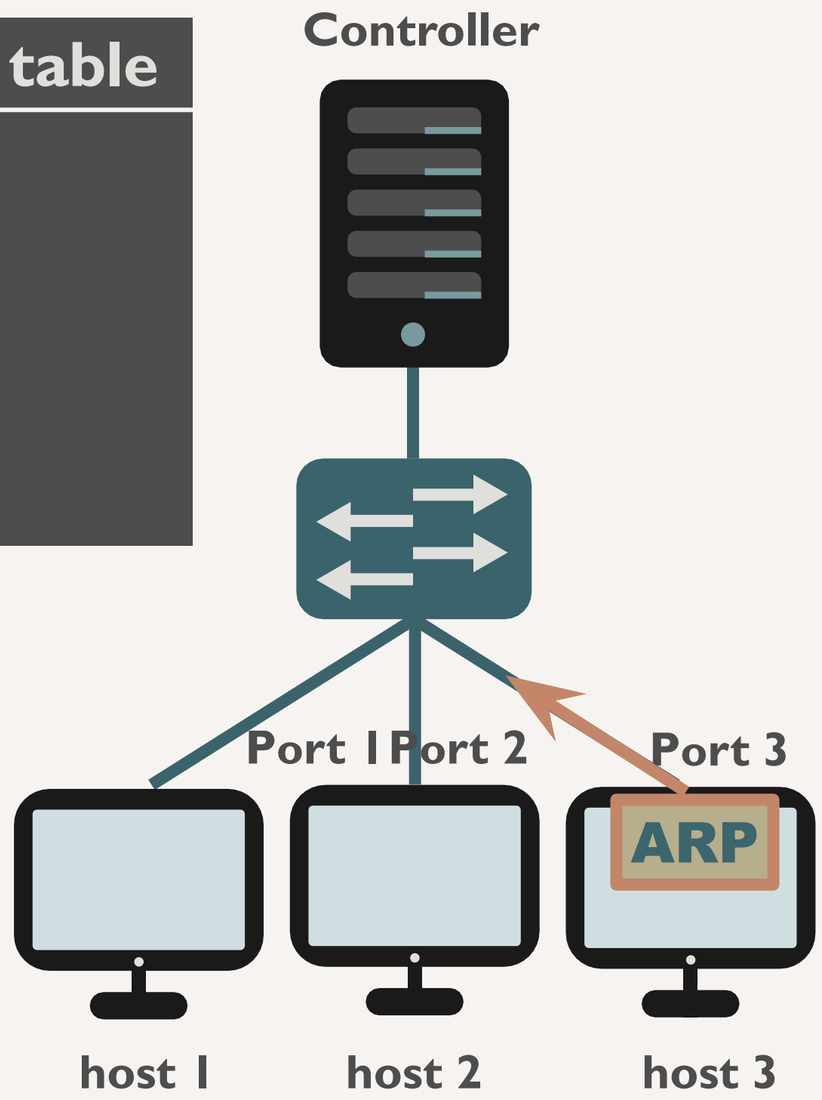


An Example of the Process

– Step 6: PacketIn again

- The destination host (host 3) will reply the ARP request using an ARP replay and trigger another PacketIn.

Mac table	Flow table
host 1: Port 1	

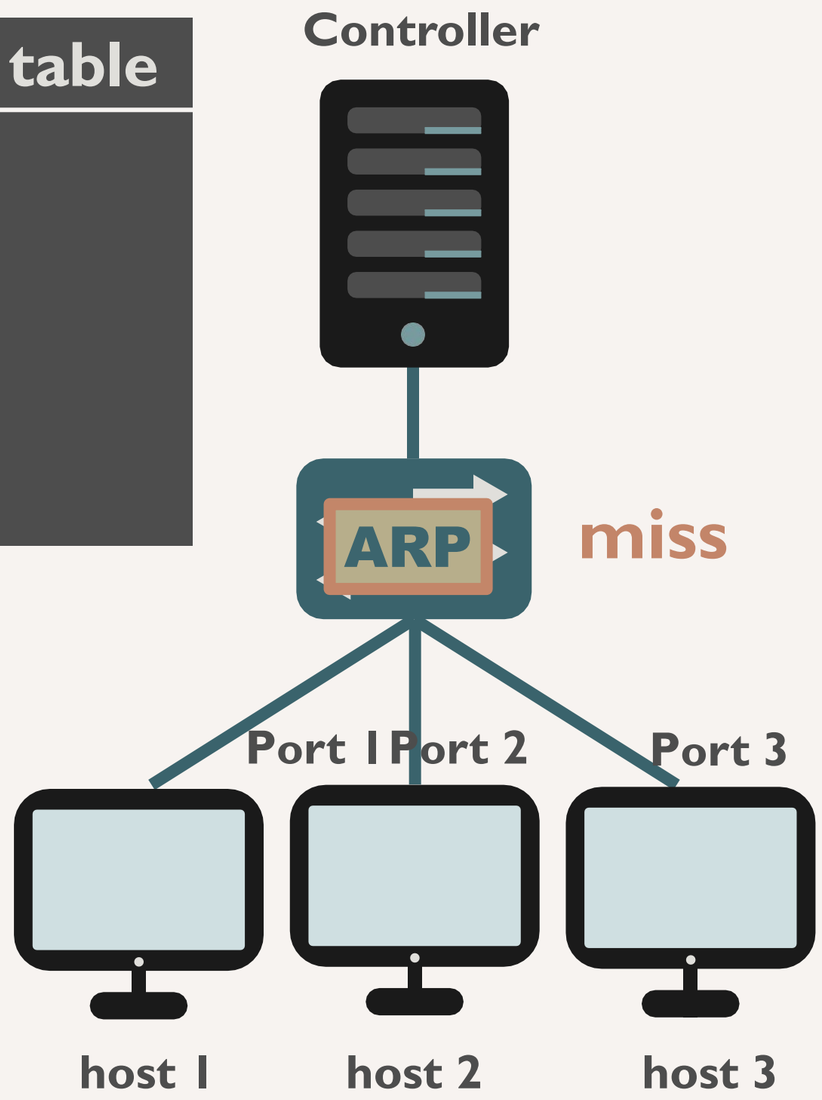


An Example of the Process

– Step 6: PacketIn again

- The destination host (host 3) will reply the ARP request using an ARP replay and trigger another PacketIn.

Mac table	Flow table
host 1: Port 1	

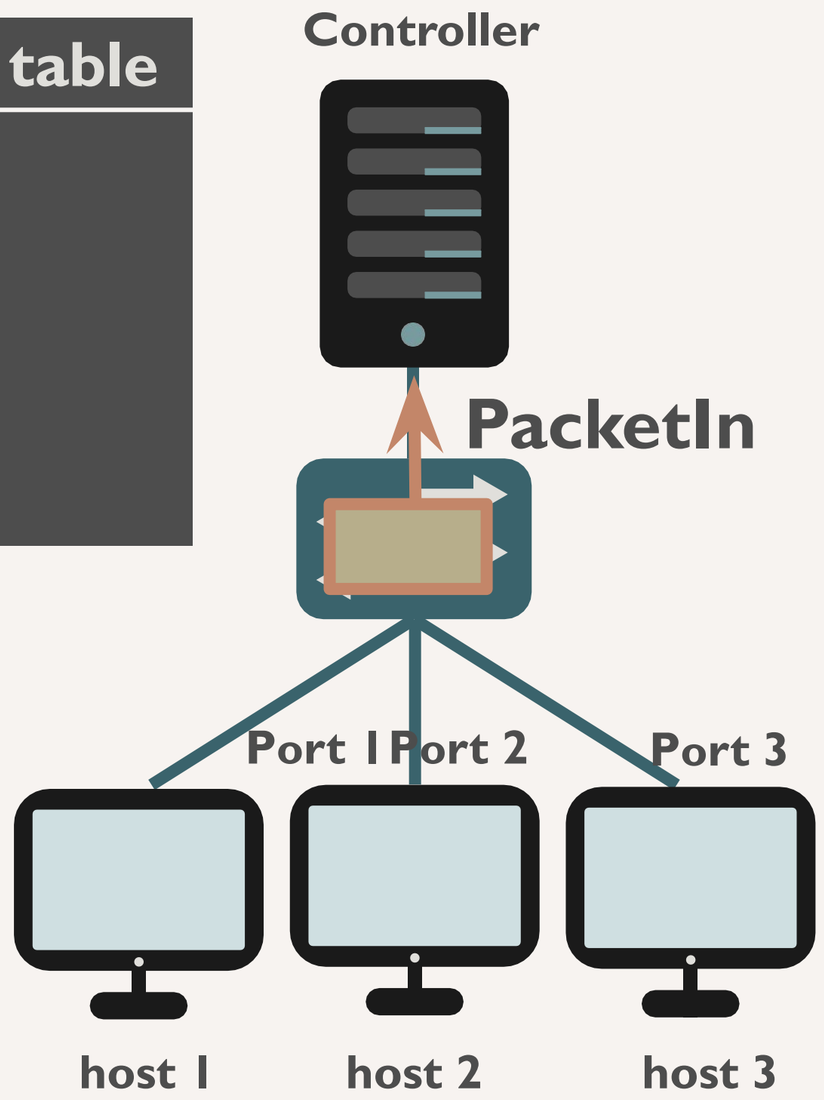


An Example of the Process

– Step 6: PacketIn again

- The destination host (host 3) will reply the ARP request using an ARP replay and trigger another PacketIn.

Mac table	Flow table
host 1: Port 1	

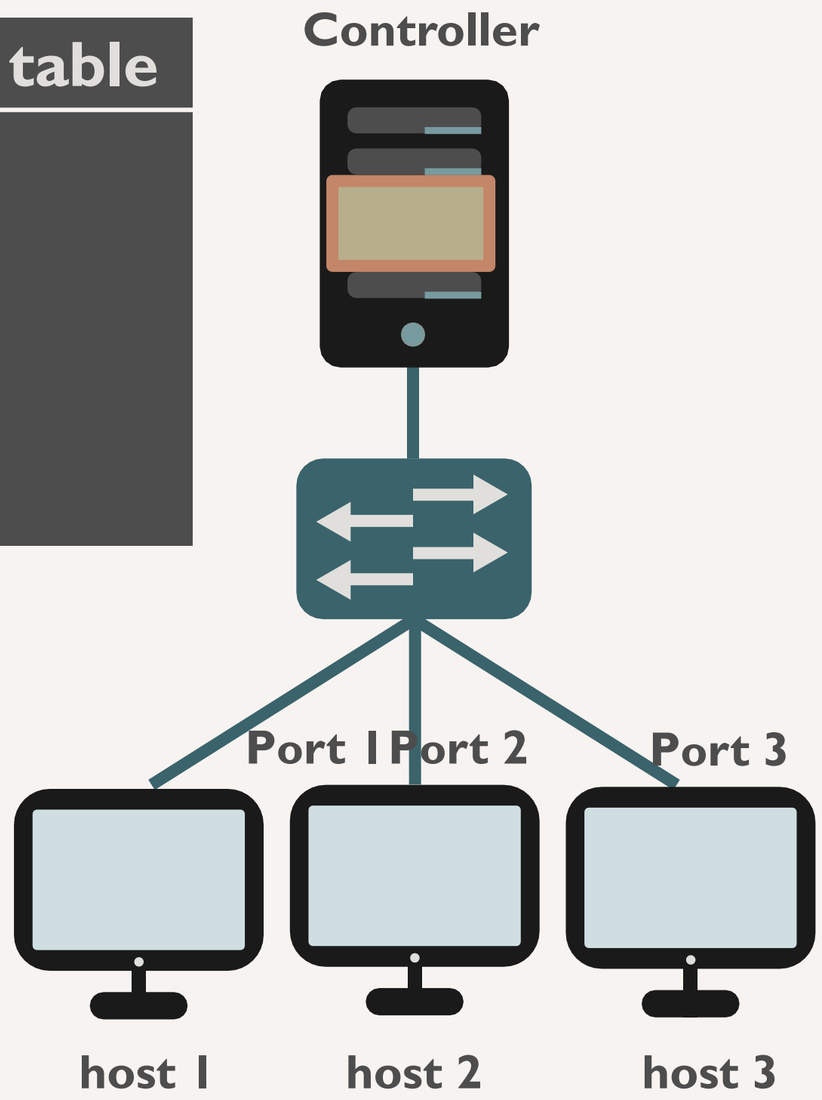


An Example of the Process

– Step 6: PacketIn again

- The destination host (host 3) will reply the ARP request using an ARP replay and trigger another PacketIn.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	



An Example of the Process

– Step 6: PacketIn again

- In _packet_in_handler

```
# install a flow to avoid packet_in nexttime
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
self.add_flow(datapath, 1, match, actions)
#...
```

- In add_flow

```
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst)
    datapath.send_msg(mod)
```

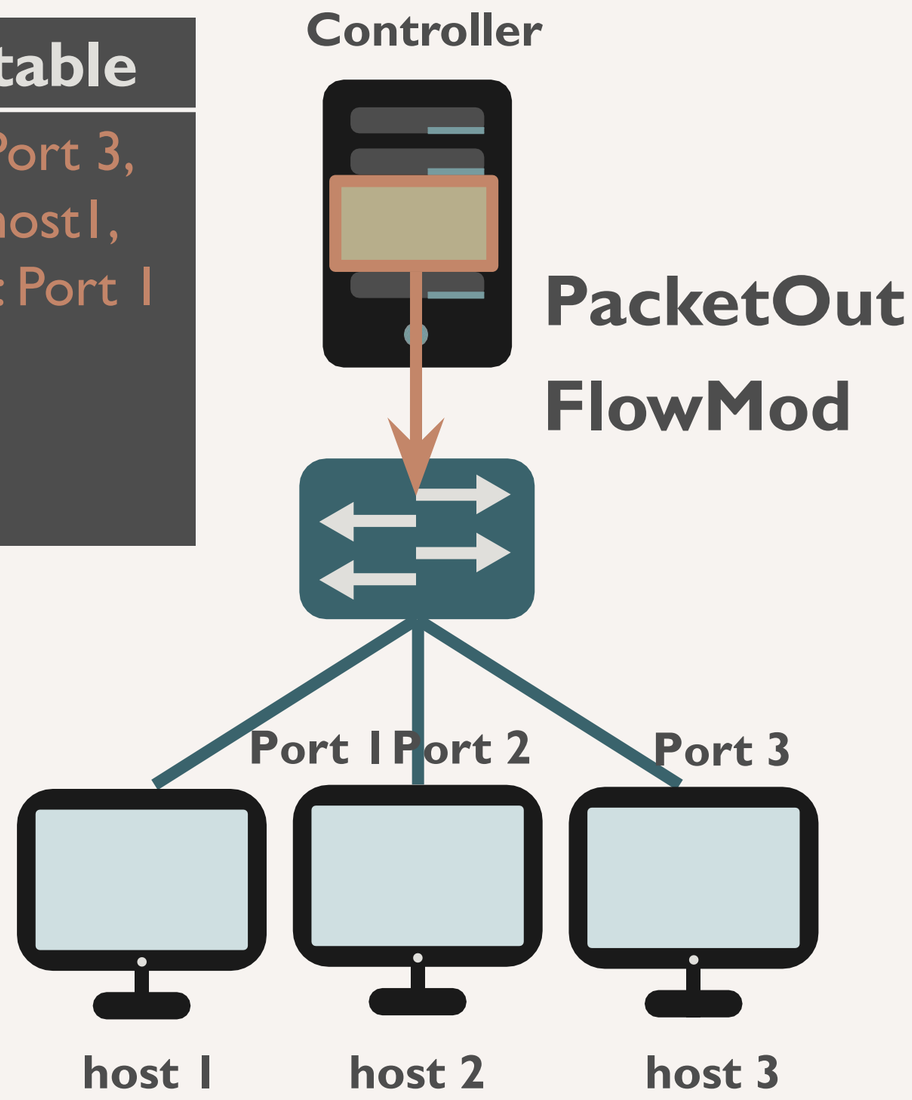
datapath: which switch
priority: the higher value, the higher priority
match: the rule set that want to match
actions: the action set

An Example of the Process

– Step 7: FlowMod & PacketOut

- This time, the controller knows the destination and source. So it adds a rule using flow mod along with the Packet out.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1

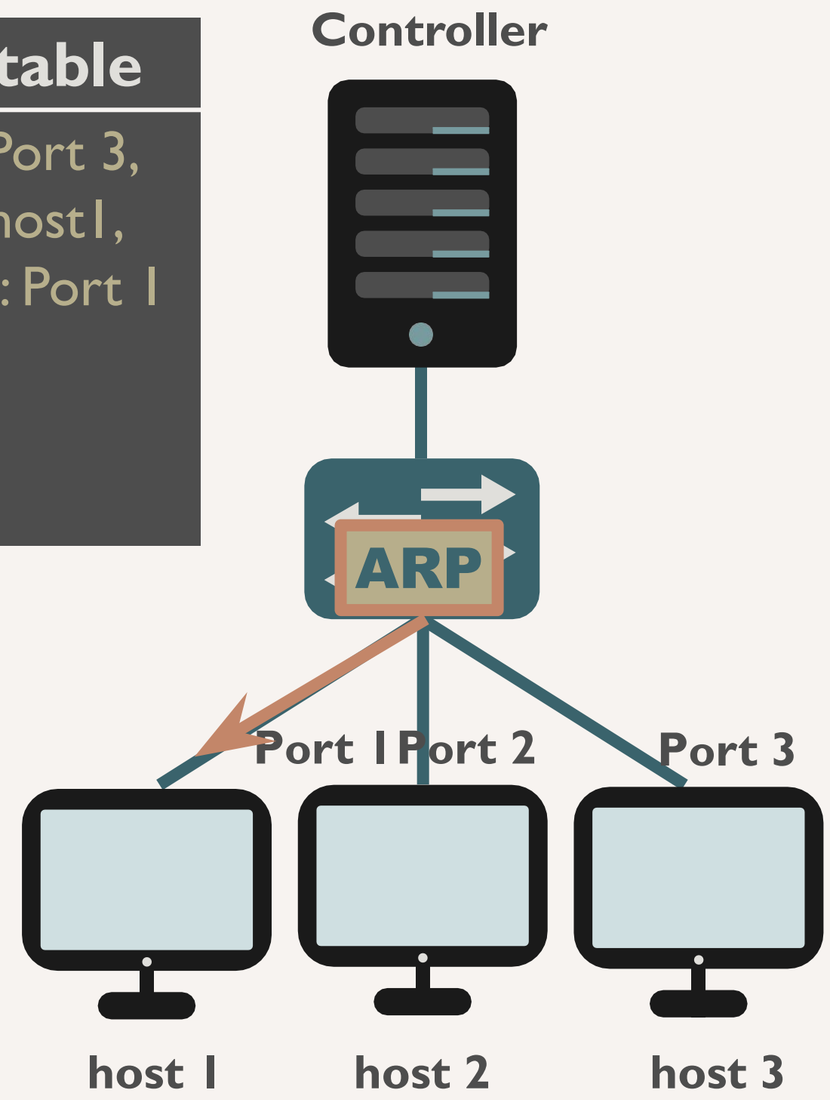


An Example of the Process

– Step 7: FlowMod & PacketOut

- This time, the controller knows the destination and source. So it adds a rule using flow mod along with the Packet out.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1



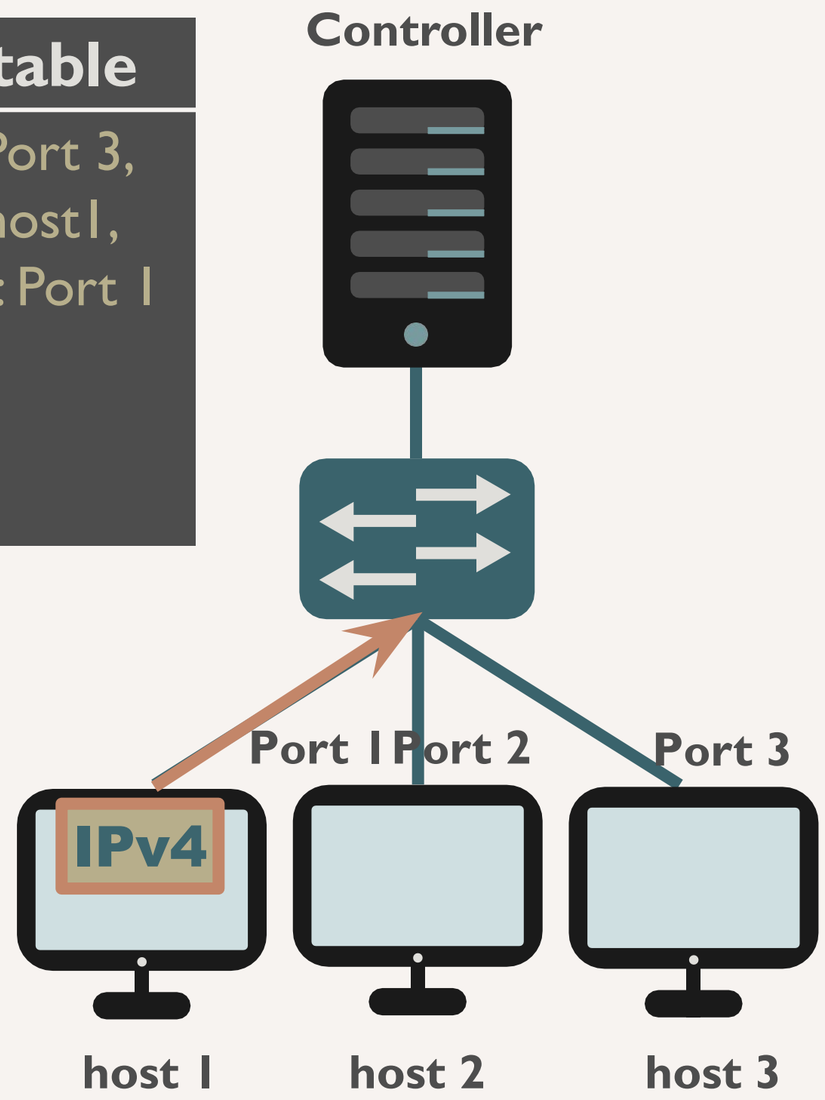
Learned the MAC

An Example of the Process

– Step 8: PacketIn of the 2nd iteration

- The next ping for host 1 to host 3 inside the network.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1

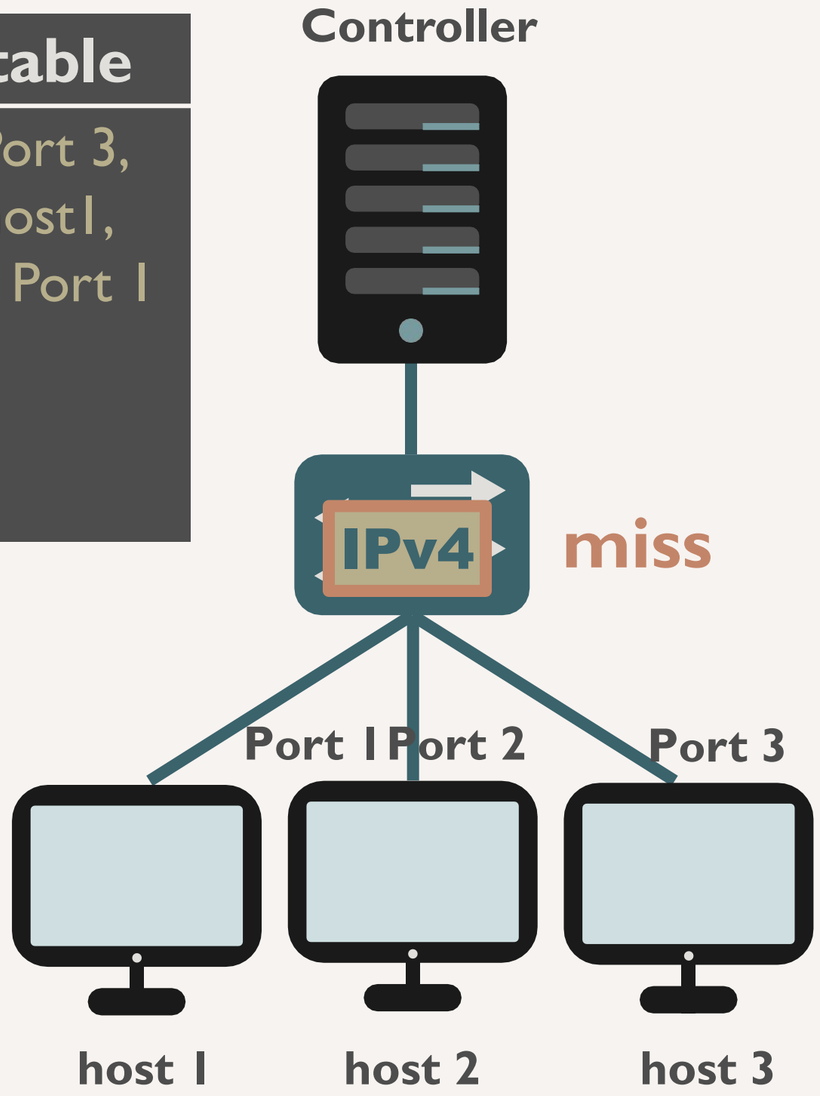


An Example of the Process

– Step 8: PacketIn of the 2nd iteration

- It will trigger a table miss because it's the first time for the switch to get a packet sent from host 1 to host 3.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1

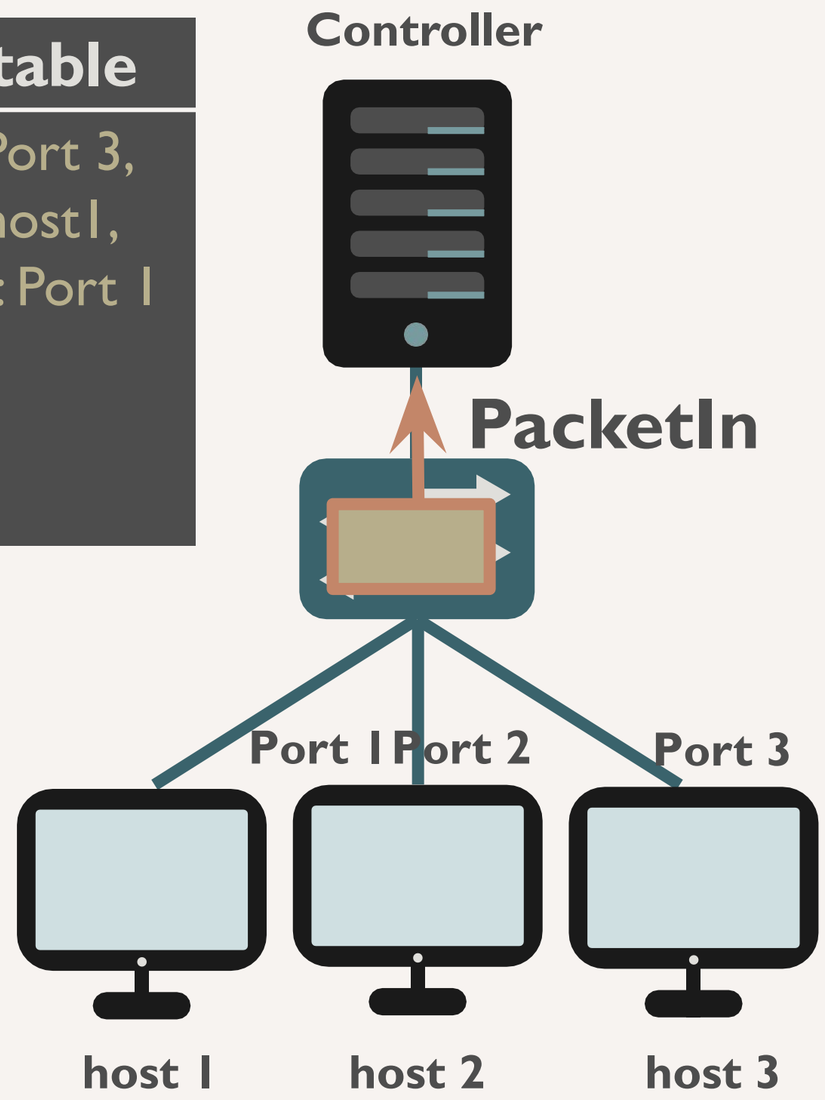


An Example of the Process

– Step 8: PacketIn of the 2nd iteration

- It will trigger a table miss because it's the first time for the switch to get a packet sent from host 1 to host 3.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1

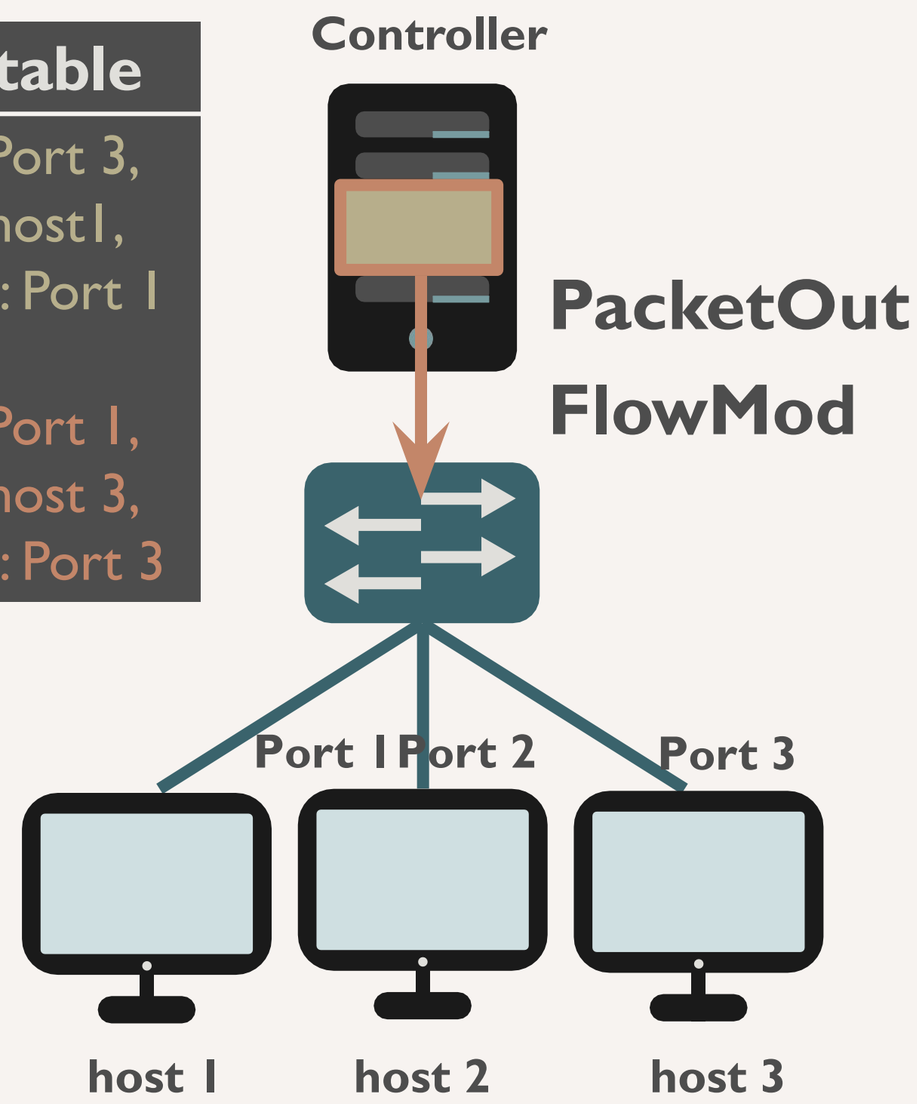


An Example of the Process

– Step 8: PacketIn of the 2nd iteration

- This time, the controller knows the destination and the source. So it adds a rule using FlowMod and send a PacketOut.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1 in_port: Port 1, eth_dst: host 3, out_port: Port 3

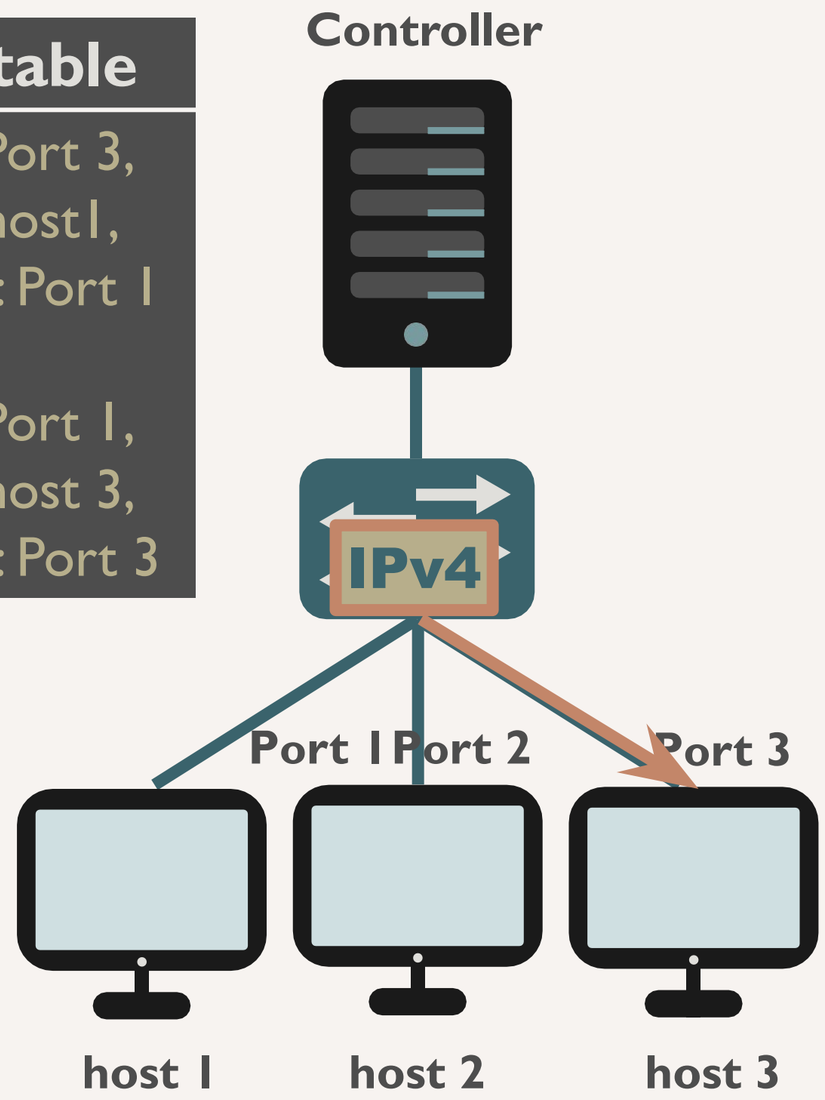


An Example of the Process

– Step 8: PacketIn of the 2nd iteration

- This time, the controller knows the destination and the source. So it adds a rule using FlowMod and send a PacketOut.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1 in_port: Port 1, eth_dst: host 3, out_port: Port 3

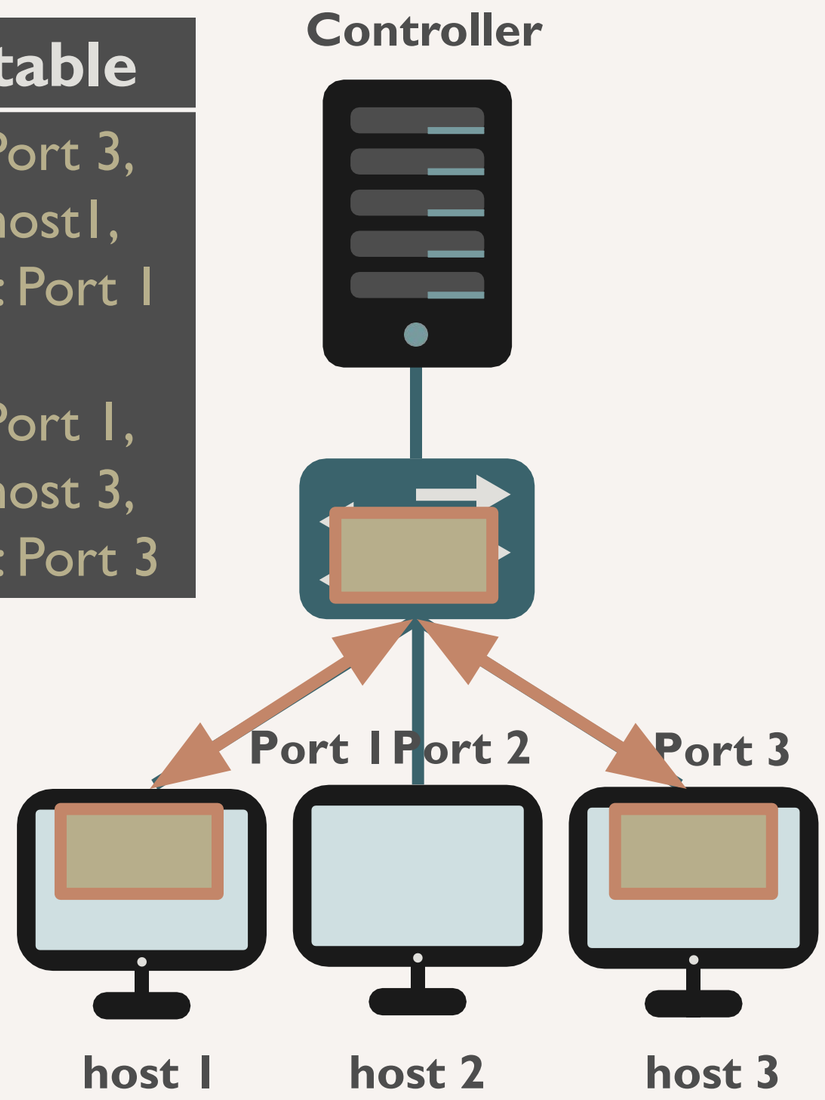


An Example of the Process

– Result: the Path Has Made

- The destination host generates reply packet and straightly forward to the source by the switch.

Mac table	Flow table
host 1: Port 1 host 3: Port 3	in_port: Port 3, eth_dst: host1, out_port: Port 1 in_port: Port 1, eth_dst: host 3, out_port: Port 3



Result - Mininet

$ mininet> pingall

```
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 27.992 seconds
mininet@mininet-vm:~$ ryu-manager --verbose ryu.app.simple_switch_13
ryu-manager: command not found
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller
remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> ryu-manager --verbose ryu.app.simple_switch_13
*** Unknown command: ryu-manager --verbose ryu.app.simple_switch_13
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Result - Ryu

- Format: switch_id, src_mac, dst_mac, inport

```
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPHello
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb63c8aec> address:('127.0.0.1', 5500
4)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0xb63c840c>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x92b165c,OFPSwitchFeatures(auxiliary_i
d=0,capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
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
```

Open vSwitch
&
OpenFlow

Open vSwitch

- A production quality, multilayer virtual switch
- It can operate both as a soft switch running within the hypervisor, and as the control stack for switching silicon.
- It has been ported to multiple virtualization platforms and switching chipsets.

Open vSwitch Commands

- Show Interfaces

```
$ sudo ovs-vsctl show
```

- Set Controller on the specific OvS

```
$ sudo ovs-vsctl set-controller <Bridge>:tcp:1.2.3.4.6633
```

- Set OpenFlow version that the specific OvS can support

```
$ sudo ovs-vsctl set bridge <Bridge> protocols=OpenFlow13
```

- Clear the setting of OpenFlow version on the specific OvS

```
$ sudo ovs-vsctl clear bridge <Bridge> protocols
```

- Query the flow rules on the specific OvS

```
$ sudo ovs-ofctl dump-flows <Bridge> -O OpenFlow13
```

Result - Mininet

$ mininet> pingall

```
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 27.992 seconds
mininet@mininet-vm:~$ ryu-manager --verbose ryu.app.simple_switch_13
ryu-manager: command not found
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller
remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> ryu-manager --verbose ryu.app.simple_switch_13
*** Unknown command: ryu-manager --verbose ryu.app.simple_switch_13
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Result - Open vSwitch s1

Flow table (7 rules)

Top 6 rules: host to host
The other: to the controller

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=23.221s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=2,dl_src=00:00:
00:00:00:02,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=23.228s, table=0, n_packets=3, n_bytes=238, priority=1,in_port=3,dl_src=00:00:
00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=23.222s, table=0, n_packets=3, n_bytes=238, priority=1,in_port=3,dl_src=00:00:
00:00:00:03,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=23.227s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=1,dl_src=00:00:
00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=23.232s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=1,dl_src=00:00:
00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=23.233s, table=0, n_packets=3, n_bytes=238, priority=1,in_port=2,dl_src=00:00:
00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=28.92s, table=0, n_packets=9, n_bytes=546, priority=0 actions=CONTROLLER:65535
mininet@mininet-vm:~$
```

A Useful Tool - IPerf

- Create TCP or UDP data stream to measure the throughput of a network
- To become a iperf server

```
$ iperf -s [-u]
```

- To become a iperf client and send data frame

```
$ iperf -c <ip> [-u] [-n <packet_size>] [-b <bandwidth>]
```

- The controller need to detect whether the uploading traffic of a host is beyond limitation.

Reference

- Mininet (link)
- Mininet Walkthrough (link)
- Data center topology (link)
- Open vSwitch Commands (link)
- OpenFlow Messages (link)
- Ryu Book (link)
- Switching Hub (link)
- Spanning tree (link)
- Traffic Monitor (link)
- Topology Viewer (link)