Openflow in a day

Indiana Center for Network Translational Research and Education

the research arm of

Instructors

Steven Wallace

Uwe Dahlmann

Ron Milford

Chris Small

Tools that we'll be using today...

Amazon Web Services

Mininet - virtual network environment, includes OpenFlow capable switch

Open VSwitch - the OpenVSwitch distribution includes an OF controller (i.e., ovs-controller) and a useful command-line utility ovs-ofclt.

FlowVisor - FV supports the virtual "slicing" of OF networks.

WireShark - an open source network "sniffer"

Teaching HTML to explain the WWW

<h1>OpenFlow's promise is its application,
not its internal workings</h1>

Yet much of today is about OpenFlow's internal workings, and very little will be polished examples of its application.

OpenFlow's Value

Enterprise
Data Center
WAN

What can OpenFlow bring to the enterprise

- Automated configuration of new equipment in your enterprise network (think controller-based wireless)
- Choose from a marketplace of solutions for common network requirements (e.g., PCI-DSS compliance, NAC network access control, etc.)
- Delegate control of network slices to their proper steward (e.g., CCTV, door locks, etc.)
- Address new requirements (e.g., bonjour printing, guest access, BYOD) through new software, not new equipment

What can OpenFlow bring to the data center

- Standard API for network provisioning (i.e. orchestration)
- Integration with VM-based switches (e.g. Open vSwitch)
- New network behaviors that permit scaling to million-VM data centers
- Potential for ODMs to provide more cost effective solutions

What can OpenFlow bring to the wide area network

- Standard API for network provisioning of bandwidth-on-demand services (e.g. Internet2 OS3E)
- Standard API upon which to address new requirements (e.g. lawful intercept)
- Delegate control of network slices upon which arbitrary virtual networks can coexist on a common network platform

OpenFlow Origin

Clean Slate Program at Stanford

- Early work on SANE circa 2006 (security architecture)
- inspired Ethane circa 2007, which lead to OpenFlow

2009 Stanford publishes OF 1.0.0 spec

2009 Nicira Series A funding

2010 Big Switch seed funding

2011 Open Network Foundation is created

2012 Google announces migration to OF (migration started in 2009)

OpenFlow's Owner:

Open Networking Foundation

ONF members:

A10 Networks, Alcatel-Lucent, Argela, Big Switch Networks, Broadcom, Brocade, Ciena, Cisco, Citrix, Colt, Comcast, CompTIA, Cyan, Dell, Deutsche Telekom, Elbrys, Ericsson, ETRI, Extreme Networks, EZchip, F5, Facebook, Force10 Networks, France Telecom Orange, Fujitsu, Gigamon, Goldman Sachs, Google, Hitachi, HP, Huawei, IBM, Infinera, Infoblox, Intel, IP Infusion, Ixia, Juniper Networks, Korea Telecom, LineRate Systems, LSI, Luxoft, Marvell, Mellanox, Metaswitch Networks, Microsoft, Midokura, NCL Communications K.K., NEC, Netgear, Netronome, Nicira Networks, Nokia Siemens Networks, NTT Communications, Oracle, PICA8, Plexxi Inc., Radware, Riverbed Technology, Samsung, SK Telecom, Spirent, Telecom Italia, Tencent, Texas Instruments, Vello Systems, Verizon, VMware, Yahoo, ZTE Corporation --- Board Members

Open Networking Foundation

Membership-based 30K a year.

Members agree to share IP on reasonable terms.

Working group evolve the standard.

Not like IETF, ITU, IEEE, etc.

OpenFlow's Oxygen

(hype is adrenaline, not oxygen)

Large data center operators can roll their own. They make their own servers, their own data center designs, and their own software. Offer them a standard protocol that provides fine-grain control of COTS network hardware, they will supply lots of oxygen. Examples include:

if “Floor Plan Entropy” has got your bisection bandwidth down, build fat tree networks based on low-cost switches by programming the network for the data center via Openflow (e.g., PortLand)

if network provisioning is slow and manual, leverage an open network API to create better orchestration

OVS in the data center

Problem - need to provided an automated approach to multi-tenant isolation, VM migration, automated provisioning, hi-bisection bandwidth via equal cost multipath routing (without affecting physical data center network).

Solution - deploy OVS as the first hop switch (see Martin Casado quote on first slide) to create flexible network overlays (e.g., VXLAN and STT).

Reducing the oxygen requirement

Merchant Silicon: “off the shelf” chips that perform packet processing at high speed vs. vertically integrated custom designed chips designed & built by switch vendors.

Q: What do the following have in common:

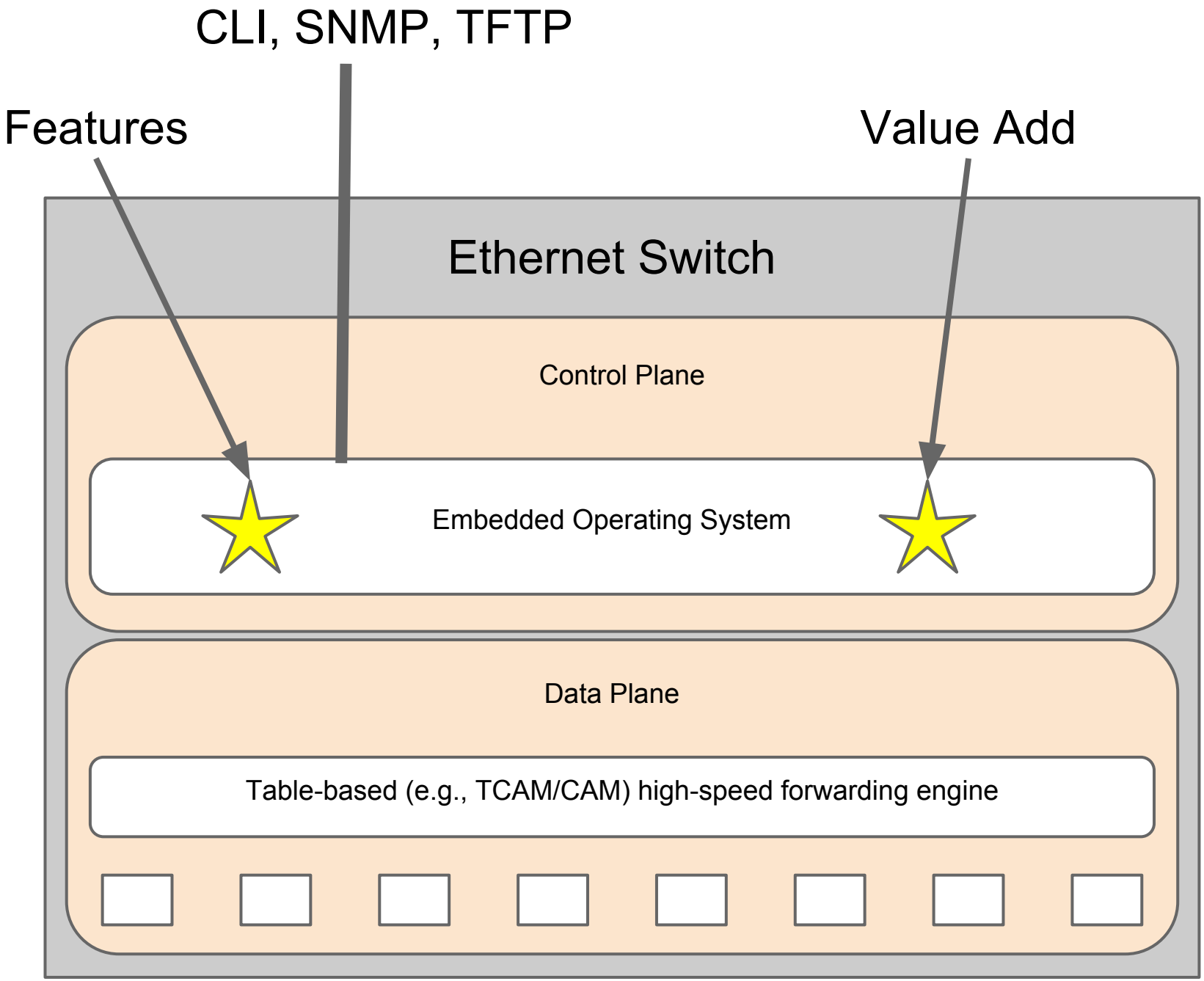
Juniper QFX3500, IBM BNT RackSwitch G8264, Alcatel-Lucent OmniSwitch 6900, Cisco Nexus 3064, HP 5900AF 48XG, Dell Force 10 S4810, and Arista 7050S-64?

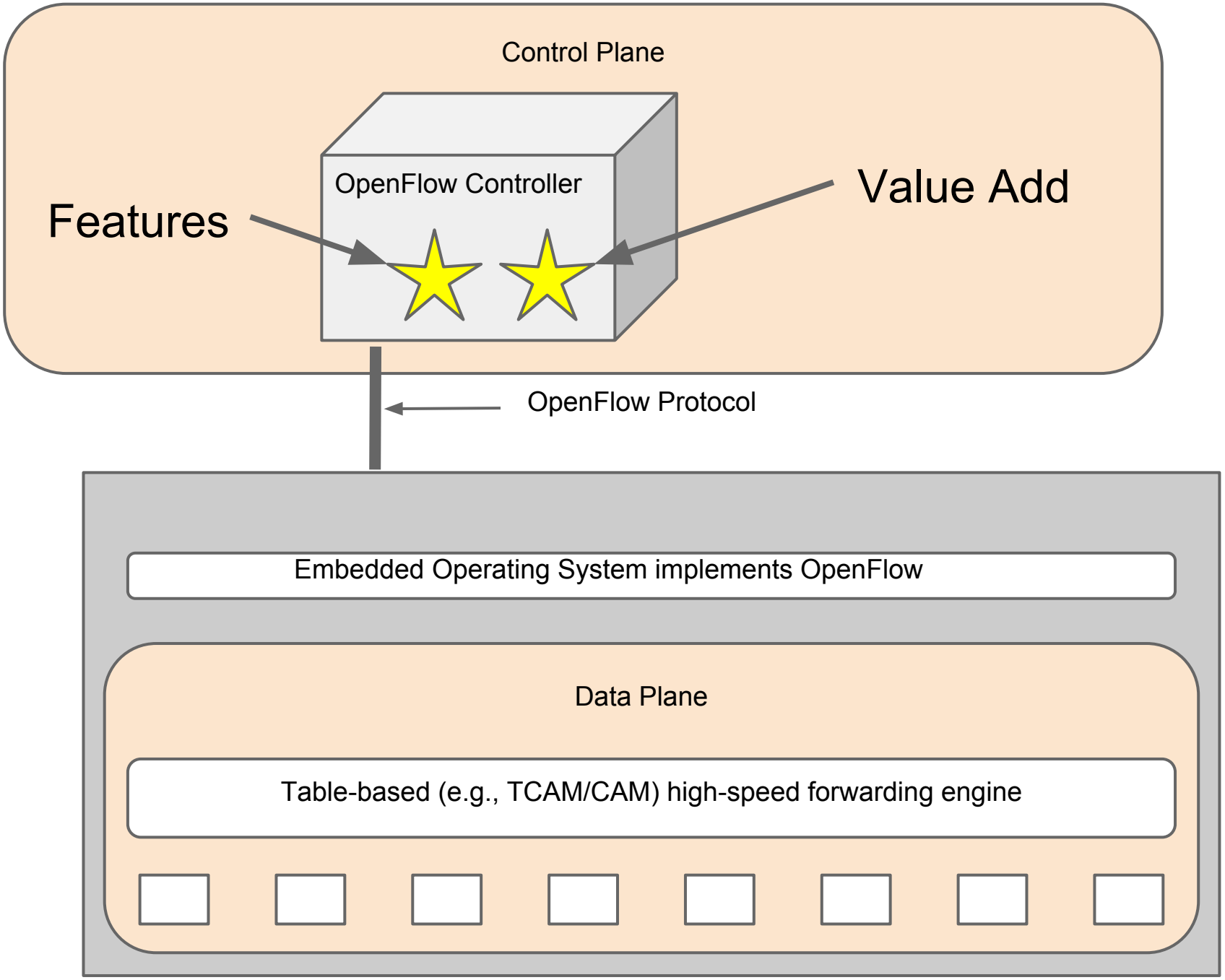
A: Broadcom silicon.

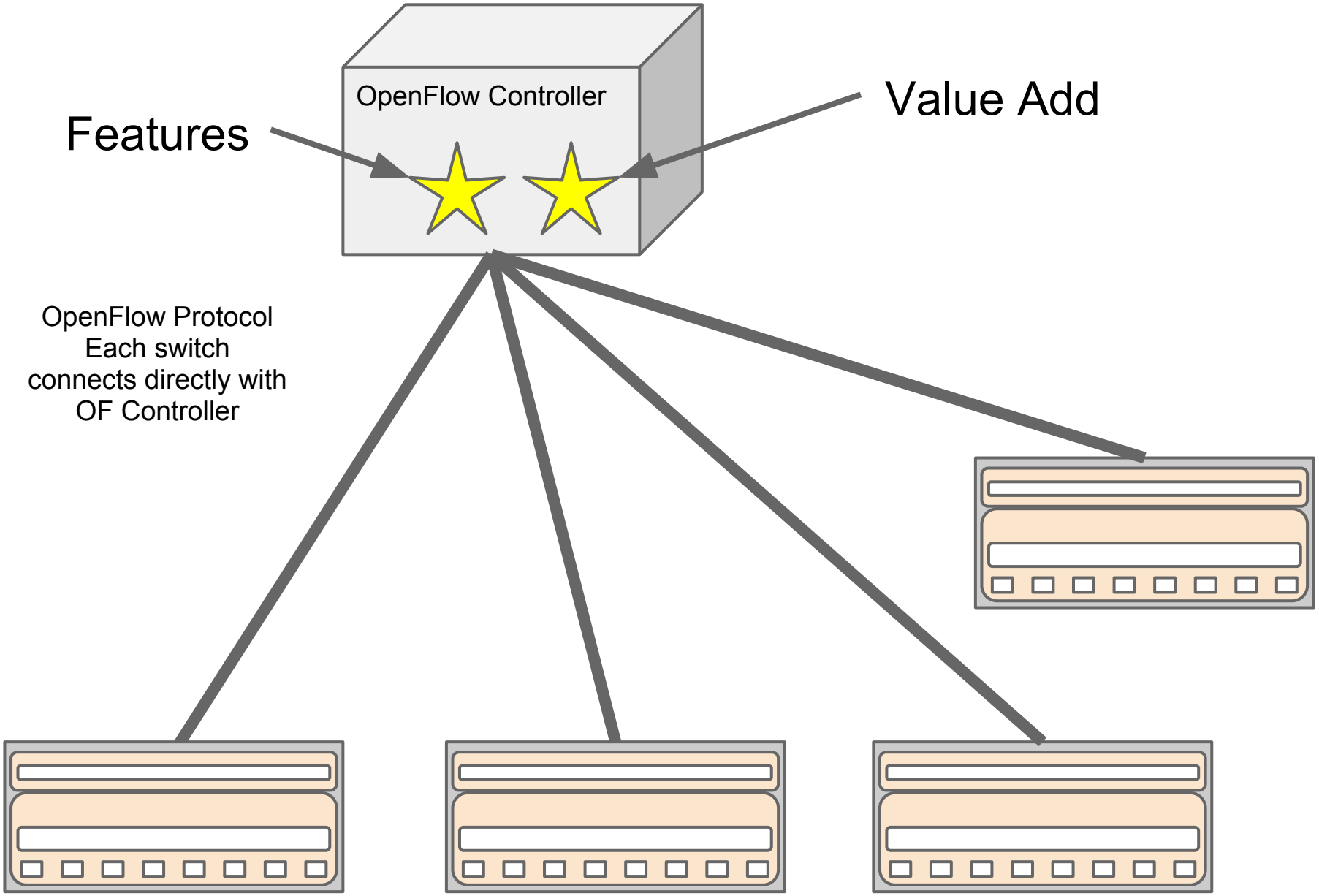
ODMs (Original Design manufacturer**)** have their own design, typically based on merchant silicon.

What is OpenFlow?

- It's a protocol for control the forwarding behavior of Ethernet switches in a Software Defined Network
- Initially released by the Clean Slate Program at Stanford, its specification is now maintained by the Open Networking Forum
- Most of today's material is based on the OpenFlow 1.0 specification
- In April 2012, OpenFlow 1.3 was approved (see also 4/2012 ONF white paper)


CLI, SNMP, TFTP
Features
Value Add
Ethernet Switch
Control Plane
Embedded Operating System
Data Plane
Table-based (e.g., TCAM/CAM) high-speed forwarding engine


Control Plane
OpenFlow Controller
Features
Value Add
OpenFlow Protocol
Embedded Operating System implements OpenFlow
Data Plane
Table-based (e.g., TCAM/CAM) high-speed forwarding engine


OpenFlow Controller
Features
Value Add
OpenFlow Protocol
Each switch
connects directly with
OF Controller

Flow Table

Header Fields	Counters	Actions	Priority
Ingress Port Ethernet Source Addr Ethernet Dest Addr Ethernet Type VLAN id VLAN Priority IP Source Addr IP Dest Addr IP Protocol IP ToS ICMP type ICMP code	**Per Flow Counters** Received Packets Received Bytes Duration seconds Duration nanosecconds	Forward (All, Controller, Local, Table, IN_port, Port# Normal, Flood) Enqueue Drop Modify-Field	

Flow Table

Header Fields	Counters	Actions	Priority
If ingress port == 2		Drop packet	32768
if IP_addr == 129.79.1.1		re-write to 10.0.1.1, forward port 3	32768
if Eth Addr == 00:45:23		add VLAN id 110, forward port 2	32768
if ingress port == 4		forward port 5, 6	32768
if Eth Type == ARP		forward CONTROLLER	32768
If ingress port == 2 && Eth Type == ARP		forward NORMAL	40000

Special Ports

Controller (sends packet to the controller)

Normal (sends packet to non-openflow function of switch)

Local (can be used for in-band controller connection)

Flood (flood the packet using normal pipeline)

Flow Table

Header Fields	Counters	Actions	Priority
If ingress port == 2		Drop packet	32768
if IP_addr == 129.79.1.1		re-write to 10.0.1.1, forward port 3	32768

Each Flow Table entry has two timers:

idle_timeout
seconds of no matching packets after which the flow is removed
zero means never timeout

hard_timeout
seconds after which the flow is removed
zero mean never timeout

If both **idle_timeout** and **hard_timeout** are set, then the flow is removed when the first of the two expires.

Populating the Flow Table

Proactive

Rules are relatively static, controller places rules in switch before they are required.

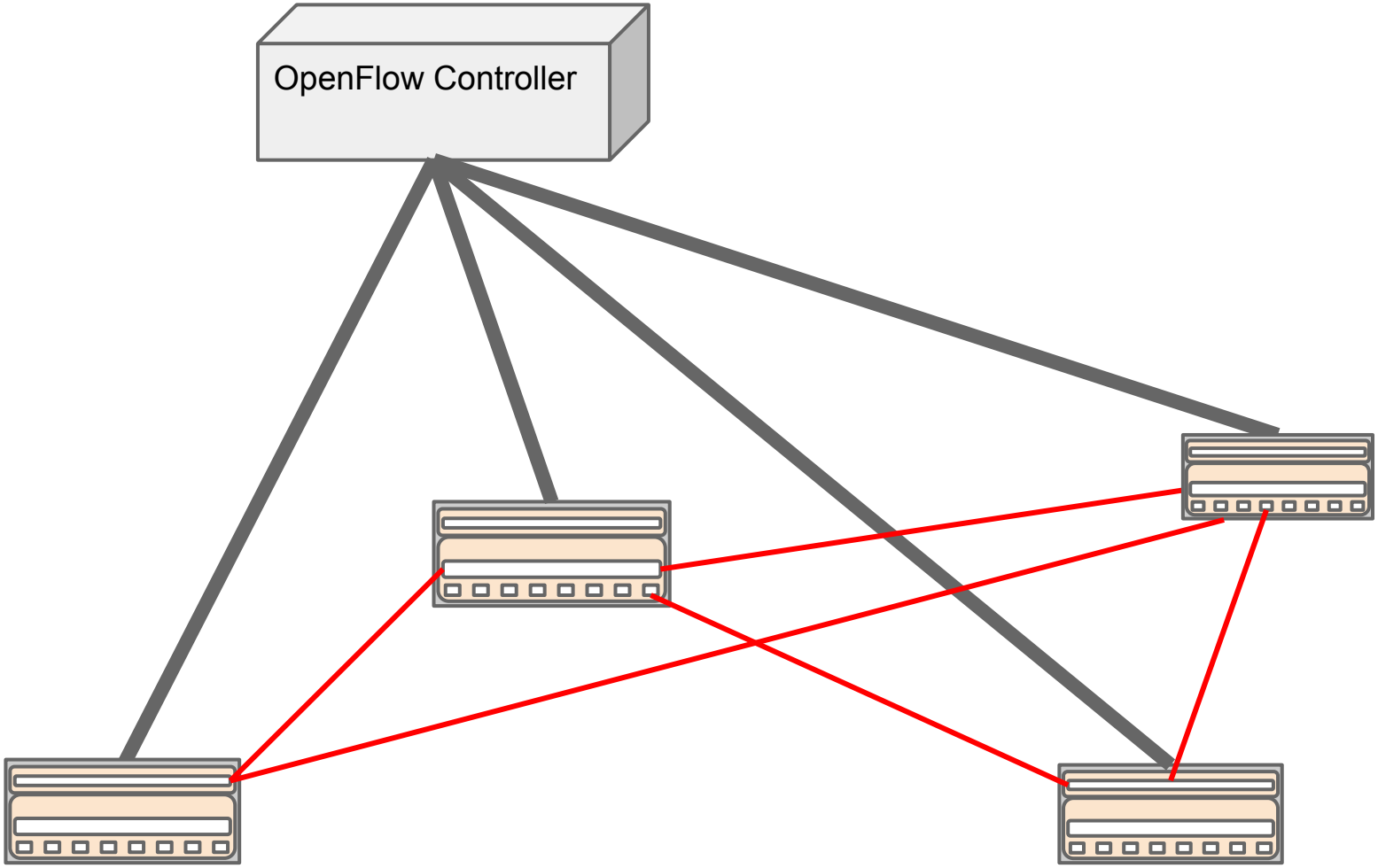
Reactive

Rules are dynamic. Packets which have no match are sent to the controller (packet in). Controller creates appropriate rule and sends packet back to switch (packet out) for processing.

Controller and Switch Communication

- Mode - Controller vs. Listener
  - TCP Communication, who initiates conversation
- Mode and Populating Flow Table independent

Example application: topology discovery



Bootstrapping a new switch

Switch requires minimal initial configuration (e. g., IP address, default GW, and OpenFlow controller)

Switch connects to controller. Controller requests things like a list of ports, etc.

Controller proceeds to determine the switch's location.

Bootstrapping a new switch

Controller *proactively* places a rule in the switch.

If ether_type = LLDP, actions=output:controller

Then the controller creates an LLDP packet, sends it to the switch, and instructs the switch to send it out a port (repeat for all ports).

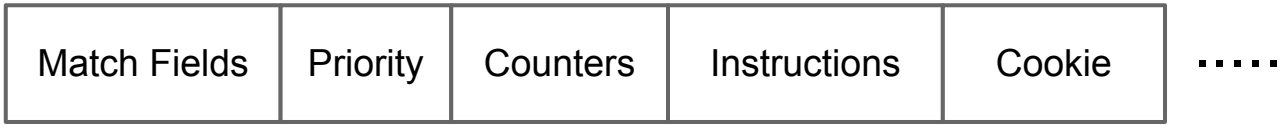
Since all switches in the controller's network have a rule to send LLDP packets to the controller, the controller is able to determine the topology.

OpenFlow 1.0 to 1.1

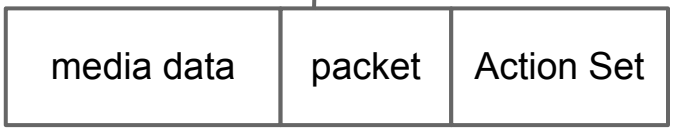
Flow Table

1.0

1.1



New Data Structure in Pipeline



Packet Processing

1.0

Does packet match flow table entry, if so, perform action.

1.1

Does packet match flow table entry, if so, look at instructions...

Actions vs. Instructions

1.1

- Flow entries contain instructions.
- Instructions may be immediate action(s), or
- instructions may set actions in the action set
- Instructions can also change pipeline processing:
  - Goto table X
  - Goto group table entry x

More Tables

1.1

- Allows for multiple flowtables
- Includes a group table with multiple group table types
- Instructions can jump to other tables, but only in a positive direction

Group Table Types

all - execute each bucket (each bucket gets copy of packet, used for flooding, multicast, etc.)

select - execute one bucket in group (used for span ports)

indirect - used for next hops

fast failover - execute first live bucket

OpenFlow QoS

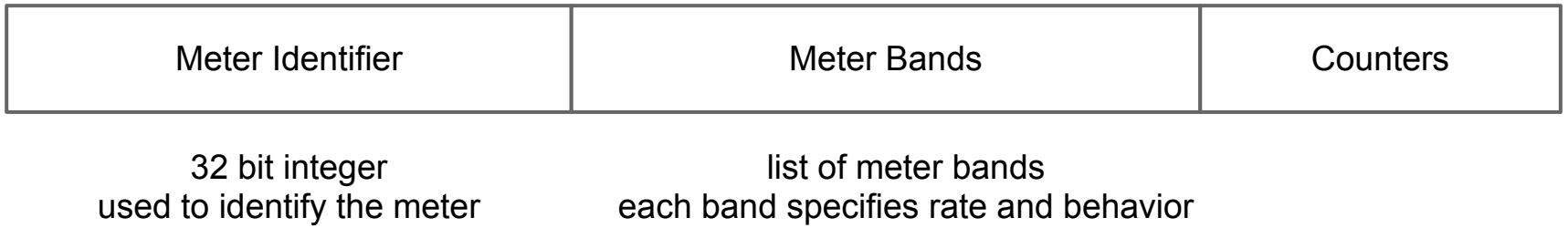
OF 1.0

- Optional action "Enqueue"
  Forwards packet through a queue attached to a port. The behavior of the queue is determined outside the scope of OF.
- Header fields can include VLAN priority and IP ToS, so they can be matched against and re-written.

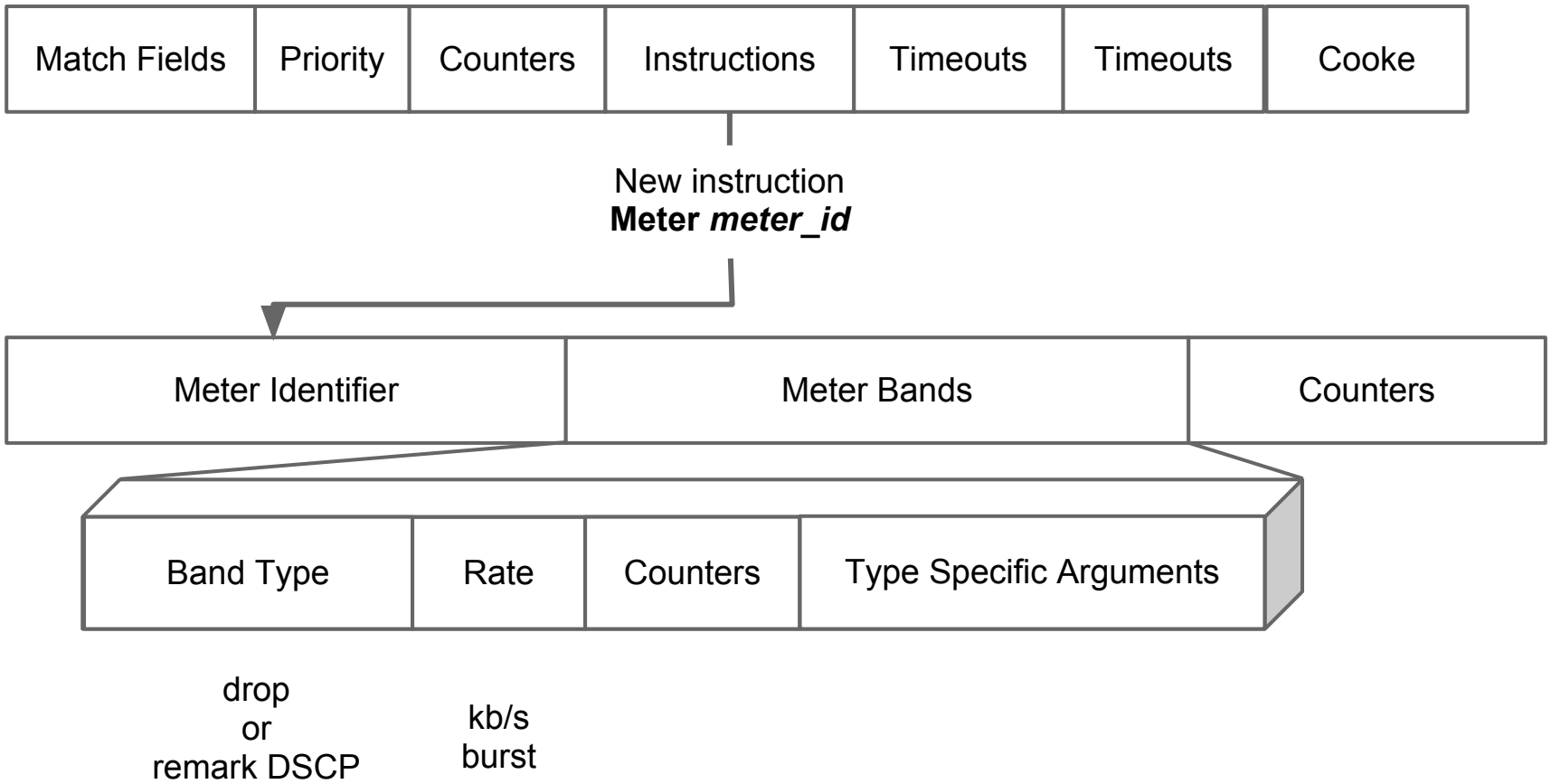
OpenFlow QoS

OF 1.3

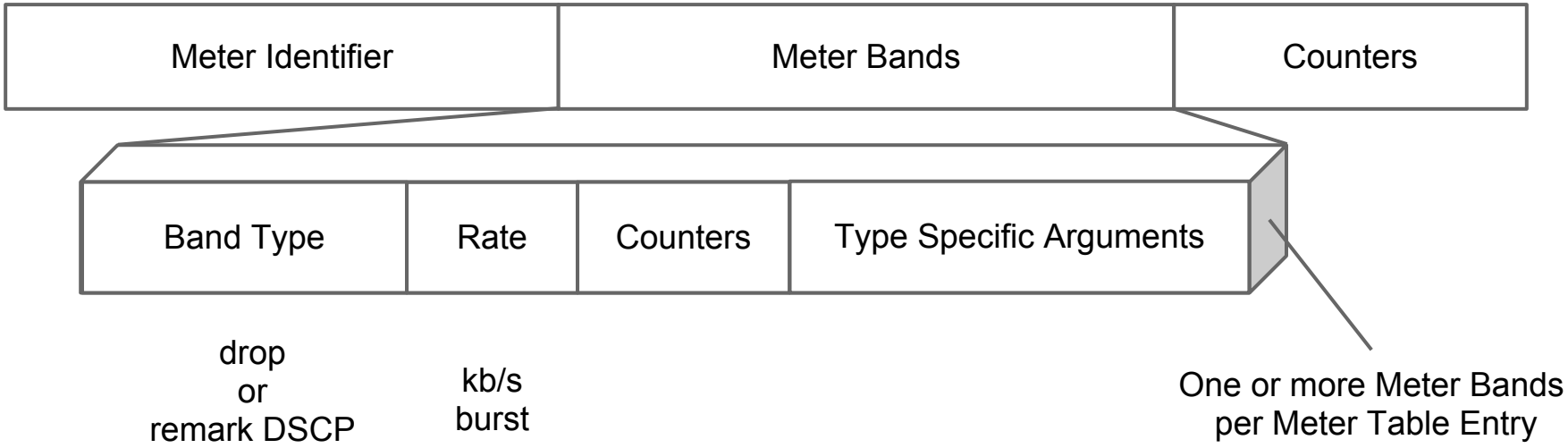
- Stuff from 1.0
- New table "Meter Table"



OpenFlow QoS (1.3 cont.)

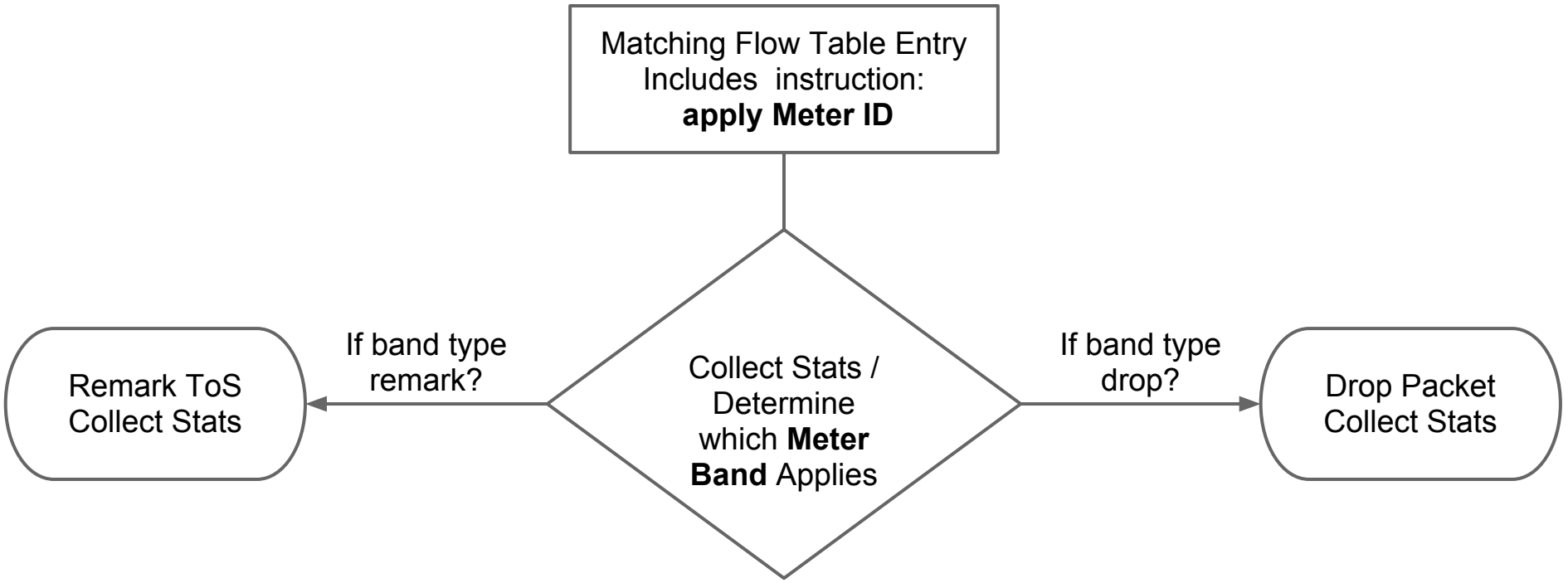


OpenFlow QoS (1.3 cont.)



"the meter applies the meter band with the highest configured rate that is lower than the current measured rate"

OpenFlow QoS (1.3 cont.)



OpenFlow Example Implementations

- HP/NEC switches run in hybrid Openflow mode
  - can act as a regular switch or as an openflow switch
  - implemented on a per VLAN basis or aggregation mode
  - capable of running multiple openflow instances
  - openflow capabilities:
- Use *mac-address-table*
  - Waste of resources on native openflow switch

HP Switch Configuration

- Enter configuration mode
  - # configure
- Create a VLAN for your Openflow instance
  - # vlan 10
- Add ports to the VLAN
  - In our case we have untagged traffic coming in on ports 1-20
    - untagged 1-20
  - Port 21 is used for management, 23-24 interconnects
    - # tagged 21
    - # tagged 23-24

HP Switch Configuration

- Now to enable Openflow on the VLAN
  - # openflow vlan 10 enable
- Tell the Openflow instance to actively connect to an Openflow controller
  - # openflow controller tcp:10.101.1.39:6633
  - 6633 is the port that is listening on the controller
- If the switch can't connect to the controller, we want the switch to forward using current rules
- # openflow fail-secure on

HP Switch Configuration

- Lastly, we want the ability to manually connect to the switch to check and set state
  - the openflow instance on the vlan will be listening on port 6633 for dpctl ovs-ofctl commands
    - # openflow listener ptcp:6633
  - Limit the listener to a specific IP address
    - # openflow listener ptcp:10.101.1.210:6633

(to see status of listener port and state for vlan 10: "show openflow 10")

Actual Switch Configuration

Running configuration:

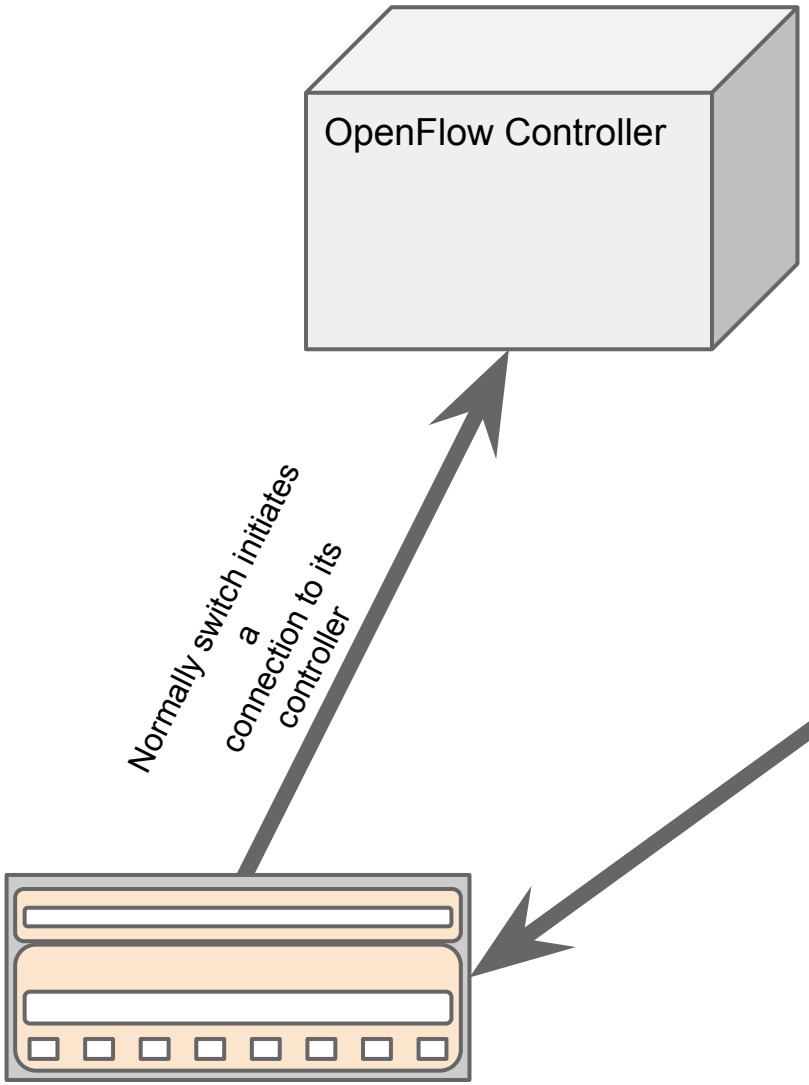
```
; J9470A Configuration Editor; Created on release #K.15.06.5008
; Ver #02:10.0d:1f

hostname "sw-1"
time timezone -300
time daylight-time-rule Continental-US-and-Canada
module 1 type J94ddA
vlan 1
   name "DEFAULT_VLAN"
   untagged 22
   no untagged 1-21,23-24
   no ip address
   exit
vlan 2
   name "VLAN2"
   untagged 21
   ip address 10.101.1.101 255.255.255.0
   exit
vlan 10
   name "VLAN10"
   untagged 1-20
   tagged 21,23-24
   no ip address
   exit
openflow
   vlan 10
      enable
      controller "tcp:10.101.1.50:6633" listener "ptcp:6633" fail-secure on
      exit
   exit
snmp-server community "public" unrestricted
```

Hands-on with OpenFlow
(quick review of the table)

Header Fields	Counters	Actions	Priority
Ingress Port Ethernet Source Addr Ethernet Dest Addr Ethernet Type VLAN id VLAN Priority IP Source Addr IP Dest Addr IP Protocol IP ToS ICMP type ICMP code	**Per Flow Counters** Received Packets Received Bytes Duration seconds Duration nanosecconds	Forward (All, Controller, Local, Table, IN_port, Port# Normal, Flood) Enqueue Drop Modify-Field	

Hands-on with OpenFlow



Although not part of the OF spec, many switches support a passive OF connection, where the switch listens for a connection.

We're going to use ovs-ofctl to query the switch's status.

Newer versions of OpenVSwitch do not support remote passive connections. Some hardware supports passive connection and some doesn't.

We will use local connections in this hands-on demonstration

Mininet

We will be using Mininet to simulate switches and hosts in a network.

Mininet uses OpenVSwitch as the switch and creates LXC Container VMs as hosts

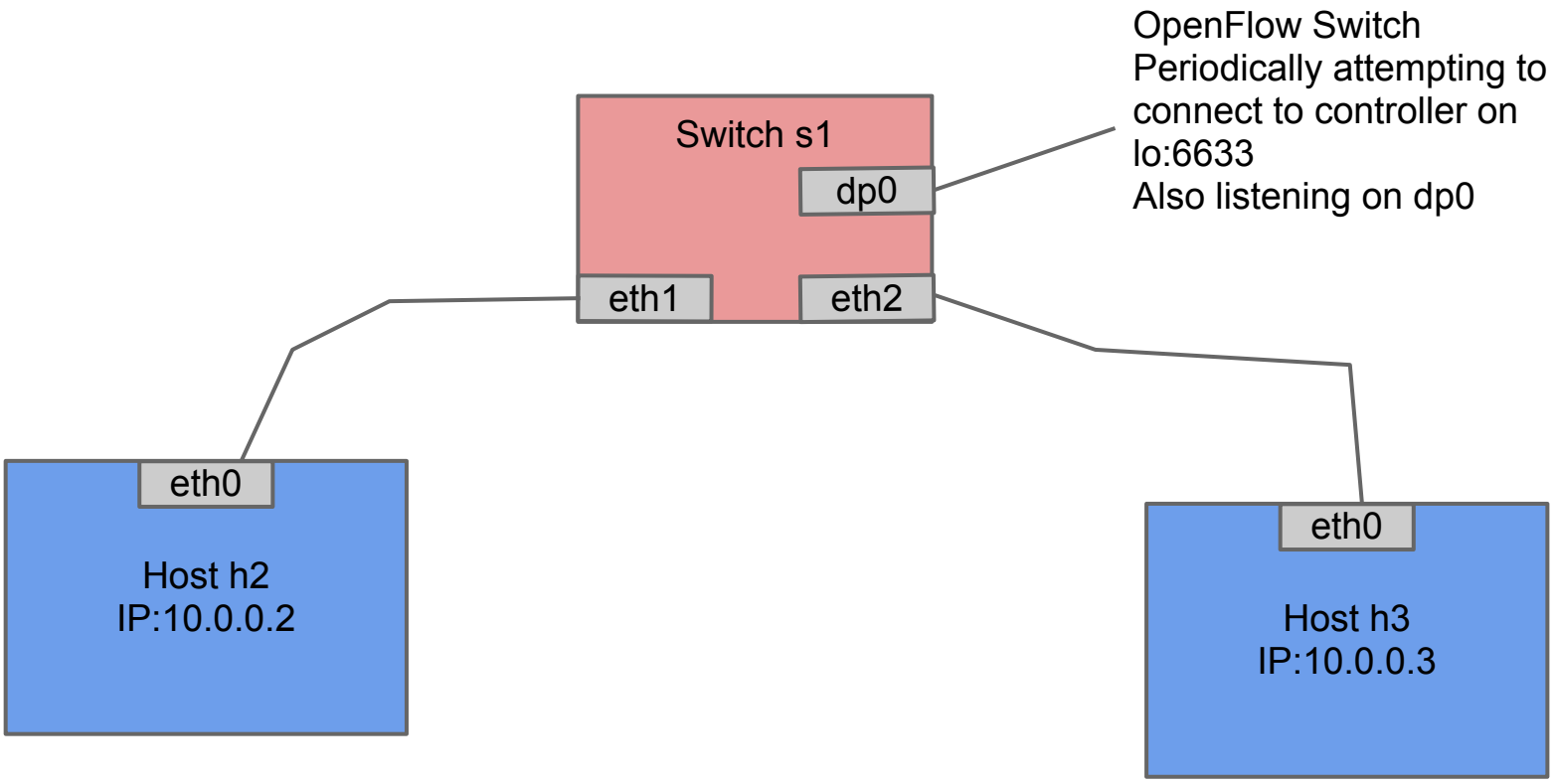
Once started, the mininet prompt "mininet>" allows commands to be run on its virtual hosts. For example

mininet>***h2 ping h3***

causes host h2 to ping host h3

To start mininet and construct a simple network, run the following in one of the terminal windows:

$***sudo mn --mac --switch ovsk --controller remote***



Getting WireShark Ready (something interesting coming up)

configure WireShark to capture on the "lo" interface

Type "of" (without the quotes) in the WireShark Filter

A bit about *ovs-ofctl*

- packaged with openvswitch-common
- alternative to *dpctl* (openflow reference controller)
- command-line utility that sends basic Openflow messages
  - useful for viewing switch port and flow stats, plus manually inserting flow entries
  - tool for early debugging
- Talks directly to the switch
  - This does not require a controller
- Switch must support a listener port (normally via TCP, but in our case via dp0

First Step!

- Run:

  $ ***sudo ovs-ofctl show dp0***

  - The 'show' command connects to the switch and prints out port state and OF capabilities
- What were the results?

- Type:

  $ ***sudo ovs-ofctl dump-flows dp0***

- Need to sudo when using a local datapath socket (dp0) because Mininet/OpenVSwitch creates it as root
- No flow? Start the ping again using mininet and recheck

ovs-ofctl - *show*

$ *sudo ovs-ofctl show dp0*

```
OFPT_FEATURES_REPLY (xid=0x1): ver:0x1, dpid:0000000000000001
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
1(s1-eth1): addr:3a:e2:98:4e:fe:aa
    config:     0
    state:      0
    current:    10GB-FD COPPER
2(s1-eth2): addr:36:29:c4:d7:a4:c1
    config:     0
    state:      0
    current:    10GB-FD COPPER
LOCAL(dp0): addr:ca:5d:78:2d:b6:40
    config:     PORT_DOWN
    state:      LINK_DOWN
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

ovs-ofctl *dump-flows*

- ***sudo ovs-ofctl dump-flows dp0***
  - Gives us information about the flows installed
  - Rule itself
  - Timeouts
  - Actions
  - Packets and bytes processed by flow

ovs-ofctl *dump-flows*

$ ***sudo ovs-ofctl dump-flows dp0***

1. NXST_FLOW reply (xid=0x4):
2. cookie=0x0, duration=30.625s, table=4, n_packets=0, n_bytes=2612, idle_timeout=180,priority=33000,in_port=1 actions=output:2
3. cookie=0x0, duration=22.5s, table=4, n_packets=0, n_bytes=2612, idle_timeout=180,priority=33000,in_port=2 actions=output:1

ovs-ofctl *dump-ports*

$ ***sudo ovs-ofctl dump-ports dp0***

- Gives physical port information
- Rx, tx counters
- Error counters

1. OFPST_PORT reply (xid=0x1): 14 ports
2. port 2: rx pkts=25211, bytes=3856488, drop=0, errs=0, frame=0, over=0, crc=0tx pkts=7144, bytes=767594, drop=0, errs=0,coll=0
3. port 5: rx pkts=18235, bytes=3142702, drop=0, errs=0, frame=0, over=0, crc=0tx pkts=0, bytes=0, drop=0, errs=0, coll=0

ovs-ofctl *del-flows*

- we can remove all or individual flows from the switch

$ ***sudo ovs-ofctl del-flows*** <expression>

  - ex. $ sudo ovs-ofctl del-flows dp0 dl_type=0x800
  - ex. $ sudo ovs-ofctl del-flows dp0 in_port=1

Exercise #1

So let's see if the network is working. Ping h2 from h3 using the following command:

mininet>***h2 ping h3***

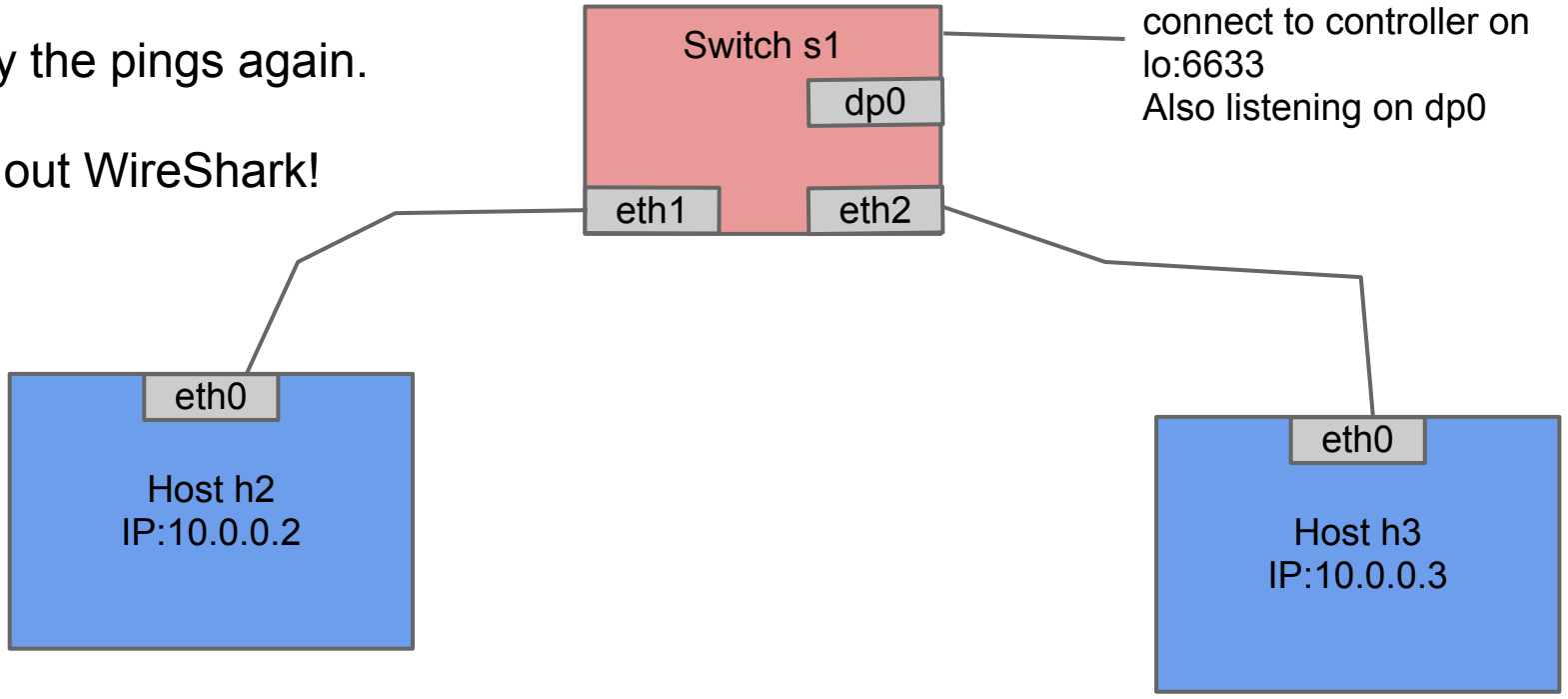
After a bit you can type control-C to stop the ping. What happened?

In the other terminal windows start the ovs-controller:

$***sudo ovs-controller ptcp:&***

Now try the pings again.

Check out WireShark!



Openflow Learning Switch

Check flow table

$ ***sudo ovs-ofctl dump-flows dp0***

Control-C ovs-controller

In that window where you started ovs-controller, enter "fg" then a control-C to kill the controller. We'll get back to it later.

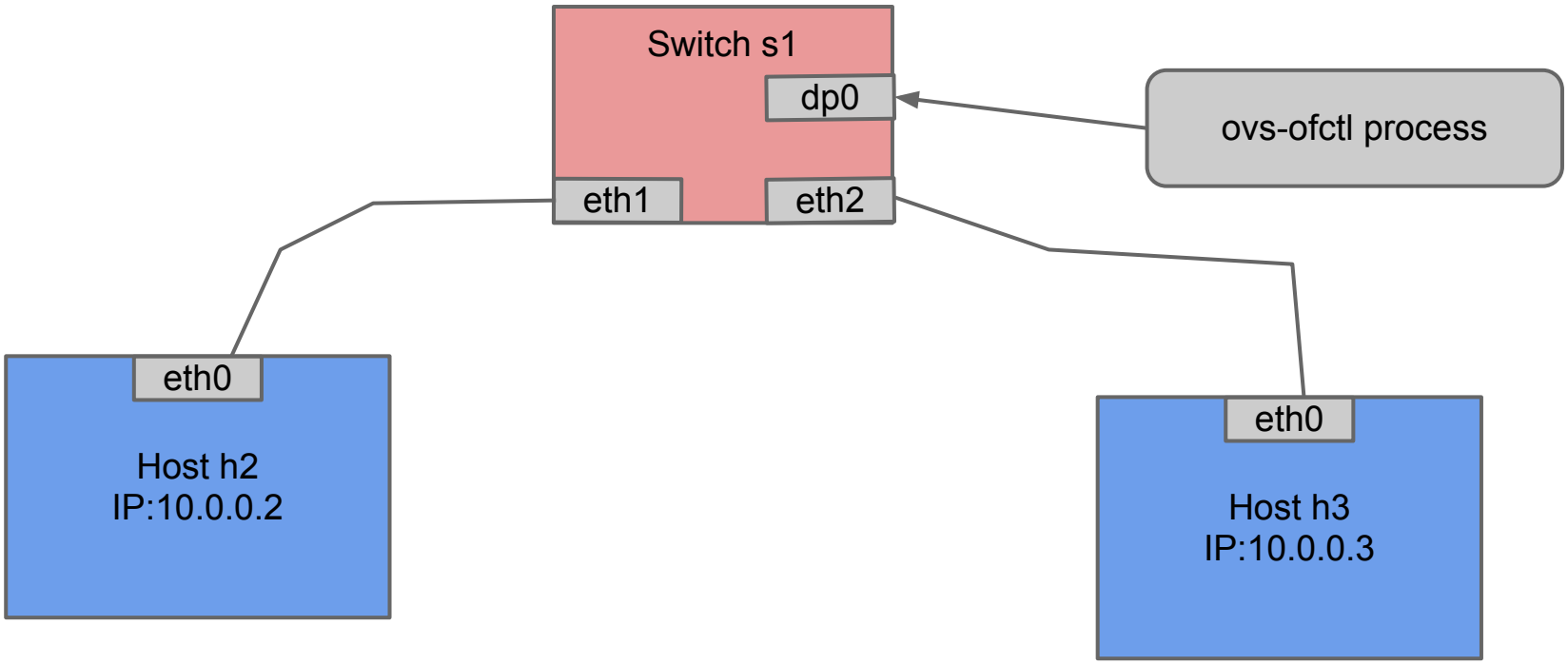
Exercise #2

Using ovs-ofctl to insert simple, port-based rules

Let's make sure switch has no existing flows:

$***sudo ovs-ofctl del-flows dp0***

Port-based Rules




```
$sudo ovs-ofctl add-flow dp0 idle_timeout=180,priority=33000,in_port=1,actions=output:2
$sudo ovs-ofctl add-flow dp0 idle_timeout=180,priority=33000,in_port=2,actions=output:1

mininet> h2 ping h3
```

Do the pings work?

What do you see with

$ ***sudo ovs-ofctl dump-flows dp0***

Do the counters increase as expected?

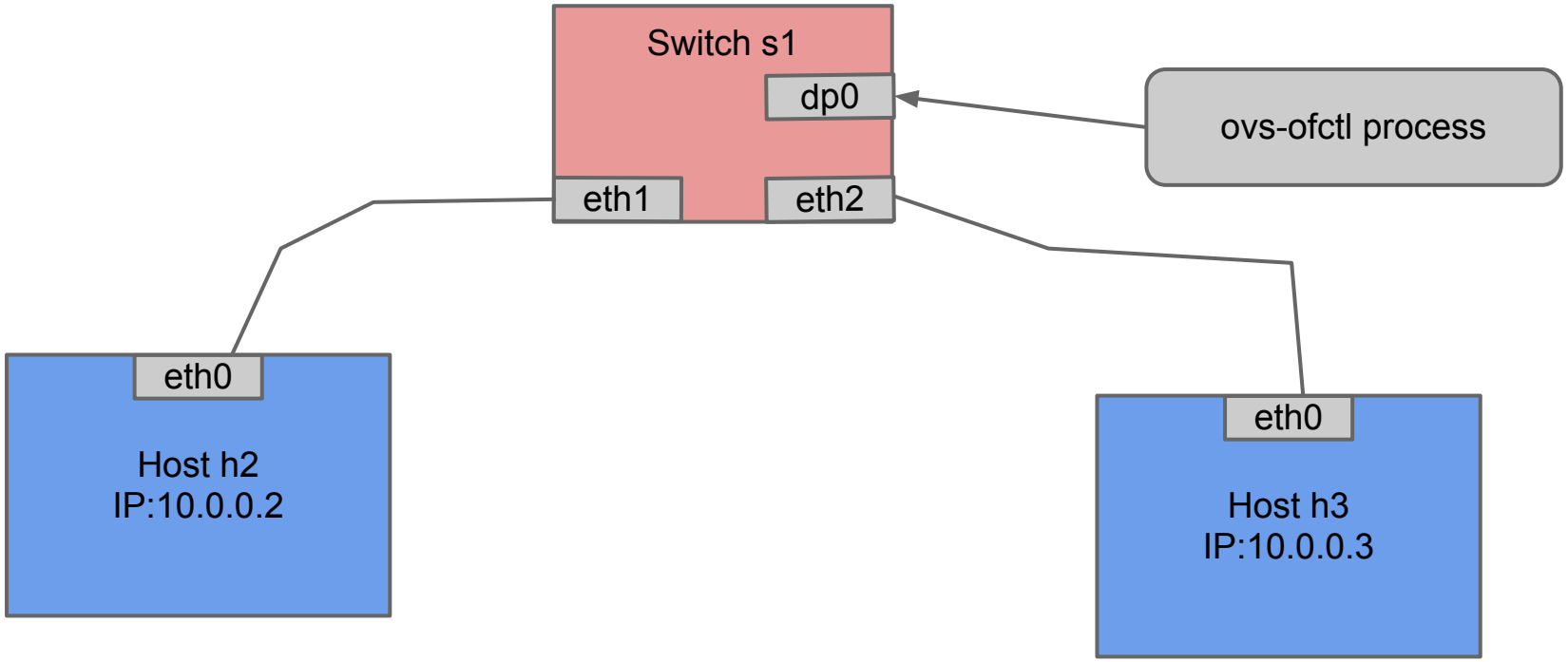
What's going on with the timeouts?

Exercise #3 - Moving up the stack...

First rule was port-based.

Next rule is IP source address-based.

IP Address-based Rules



type:

$ sudo ovs-ofctl add-flow dp0 idle_timeout=180,priority=33001,dl_type=0x800,nw_src=10.0.0.2,actions=output:2

$ sudo ovs-ofctl add-flow dp0 idle_timeout=180,priority=33001,dl_type=0x800,nw_src=10.0.0.3,actions=output:1

Do the pings work?

Did the port-based rules timeout?

If there are no port-based rules, why would the pings fail?

Can you verify this hypothesis by looking at the counters?

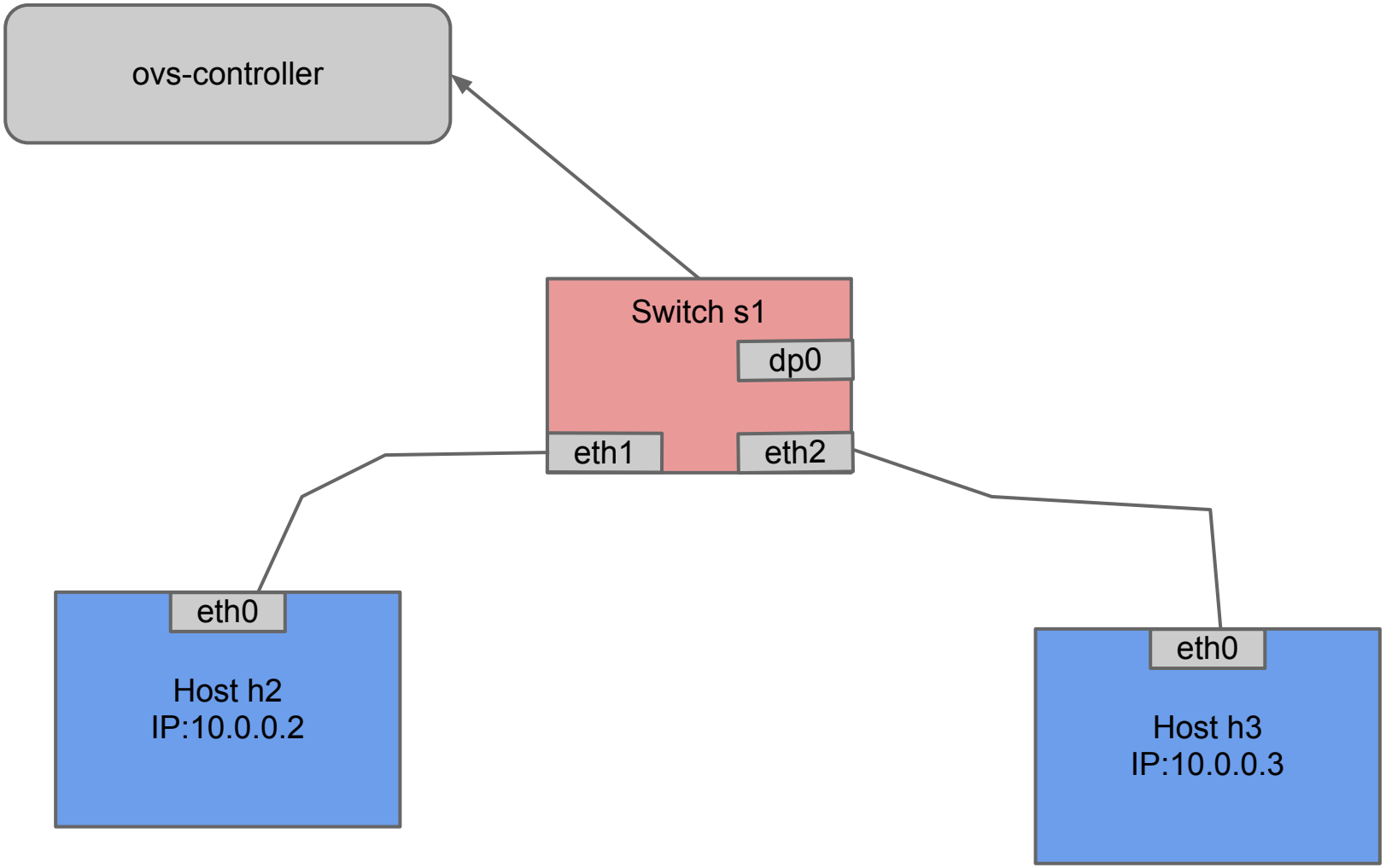
Learning Switch

What is the state of the flow table?

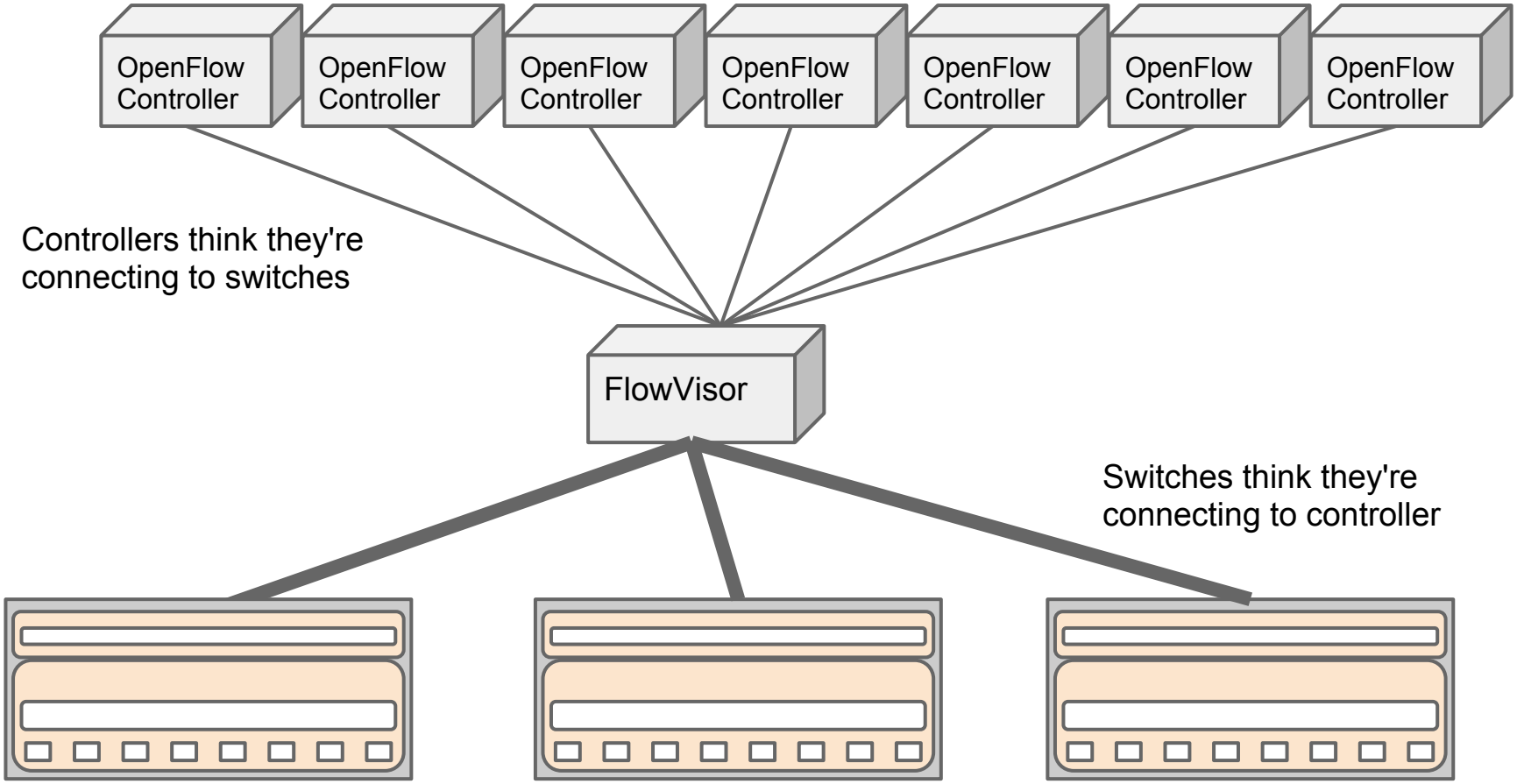
What is the ovs-controller workflow?

What happens when a broadcast packet gets sent? Multicast?

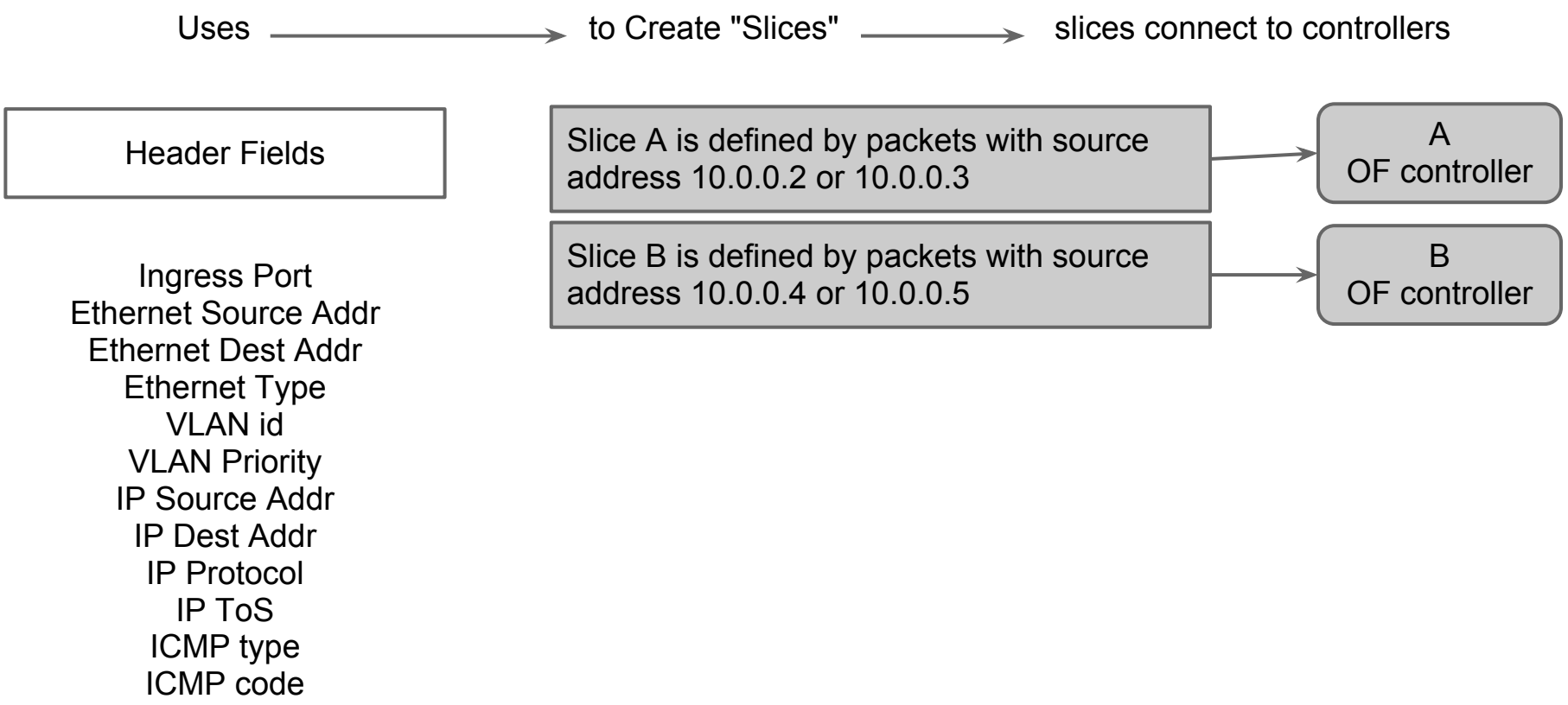
Learning Switch



FlowVisor


OpenFlow Controller
OpenFlow Controller
OpenFlow Controller
OpenFlow Controller
OpenFlow Controller
OpenFlow Controller
OpenFlow Controller
Controllers think they're connecting to switches
FlowVisor
Switches think they're connecting to controller

FlowVisor

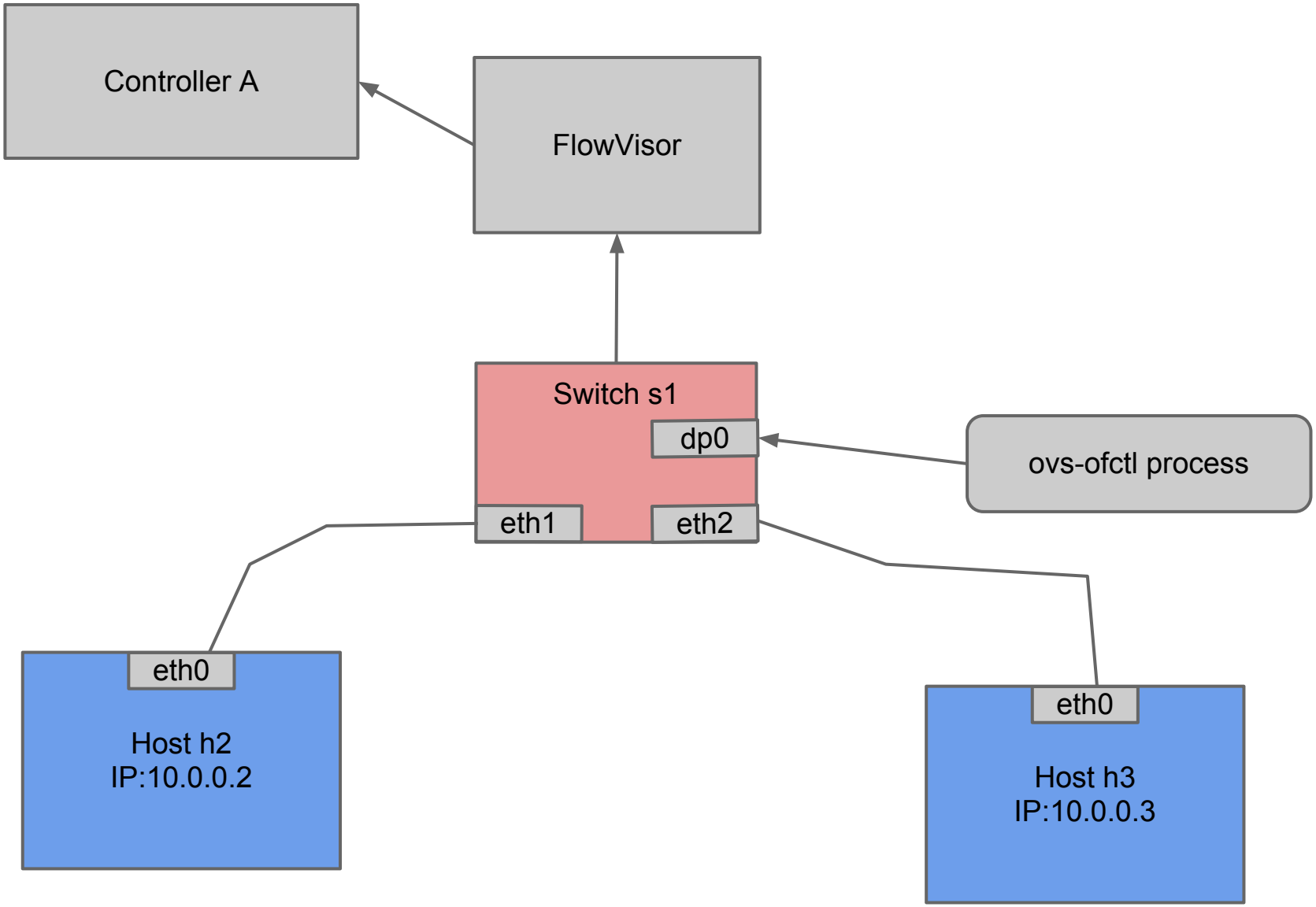

Uses
to Create "Slices"
slices connect to controllers
Header Fields
Ingress Port
Ethernet Source Addr
Ethernet Dest Addr
Ethernet Type
VLAN id
VLAN Priority
IP Source Addr
IP Dest Addr
IP Protocol
IP ToS
ICMP type
ICMP code
Slice A is defined by packets with source address 10.0.0.2 or 10.0.0.3
A
OF controller
Slice B is defined by packets with source address 10.0.0.4 or 10.0.0.5
B
OF controller

Let's make a fresh topology in mininet

quit mininet, then enter $ ***sudo mn -c*** (this cleans things up)

then enter

$ sudo mn --mac --switch ovsk --controller remote --topo single,2



Need to start FlowVisor

$ sudo /etc/init.d/flowvisor start

The default port for OpenFlow is 6633. The current OF decoder for wireshark is hardcoded to 6633. Since we're running FlowVisor and the Controller on the same host, something has to give...result is that wireshark is going to see only the FV OF channel.

Creating Slices

Example:

$ fvctl --passwd-file=/etc/flowvisor/fvpasswd createSlice slicename controller_url email

Actual:

$ ***fvctl --passwd-file=/etc/flowvisor/fvpasswd createSlice controllerA tcp:localhost:6634 fakemail@you.com***

[at "New Password:" prompt hit return]

Verify that your slice exists:

$ fvctl --passwd-file=/etc/flowvisor/fvpasswd listSlices

Adding Flowspace

You need to find the DPID of your switch

$ fvctl --passwd-file=/etc/flowvisor/fvpasswd listDevices

Example of addFlowSpace

$fvctl --passwd-file=/etc/flowvisor/fvpasswd addFlowSpace [switch_dpid] [priority] [flow_match] [slice_name]=[permissions]

Actual:

$fvctl --passwd-file=/etc/flowvisor/fvpasswd addFlowSpace 00:00:00:00:00:00:00:01 100 dl_type=0x806,dl_src=00:00:00:00:00:02 Slice:controllerA=4

$fvctl --passwd-file=/etc/flowvisor/fvpasswd addFlowSpace 00:00:00:00:00:00:00:01 100 dl_type=0x806,dl_src=00:00:00:00:00:03 Slice:controllerA=4

(continued on next page)

Add Flowspace (cont.)

(continued from previous page)

$fvctl --passwd-file=/etc/flowvisor/fvpasswd addFlowSpace 00:00:00:00:00:00:00:01 100 dl_type=0x800,nw_src=10.0.0.2 Slice:controllerA=4

$fvctl --passwd-file=/etc/flowvisor/fvpasswd addFlowSpace 00:00:00:00:00:00:00:01 100 dl_type=0x800,nw_src=10.0.0.3 Slice:controllerA=4

Create some flows to push into a switch

use pico, vi, emacs, or whatever...

to create a text file that contains flow entries in the same format as the ovs-ofctl command; name the file *flowsA.tx*t

For Controller A flowsA.txt would look like this:

priority=33000,in_port=1,actions=output:2

priority=33000,in_port=2,actions=output:1

$ ***sudo ovs-controller --verbose --noflow --mute --with-flows flowsA.txt ptcp:6634***

What happened?

What do you see in debug output?

Did a switch connect to the controller?

What shows up on wireshark?

Does the ping work???

$ sudo ovs-ofctl dump-flows dp0

How did these rules:

priority=33000,in_port=1,actions=output:2
priority=33000,in_port=2,actions=output:1

Become these:

priority=33000,ip,in_port=1,nw_src=10.0.0.2 actions=output:2
priority=33000,ip,in_port=2,nw_src=10.0.0.2 actions=output:1
priority=33000,ip,in_port=1,nw_src=10.0.0.3 actions=output:2
priority=33000,ip,in_port=2,nw_src=10.0.0.3 actions=output:1
priority=33000,arp,in_port=1,dl_src=00:00:00:00:00:02 actions=output:2
priority=33000,arp,in_port=1,dl_src=00:00:00:00:00:03 actions=output:2
priority=33000,arp,in_port=2,dl_src=00:00:00:00:00:03 actions=output:1
priority=33000,arp,in_port=2,dl_src=00:00:00:00:00:02 actions=output:1

listFlowSpace

- verify the flowspace that you added

  $ *fvctl --passwd-file=/etc/flowvisor/fvpasswd listFlowSpace*

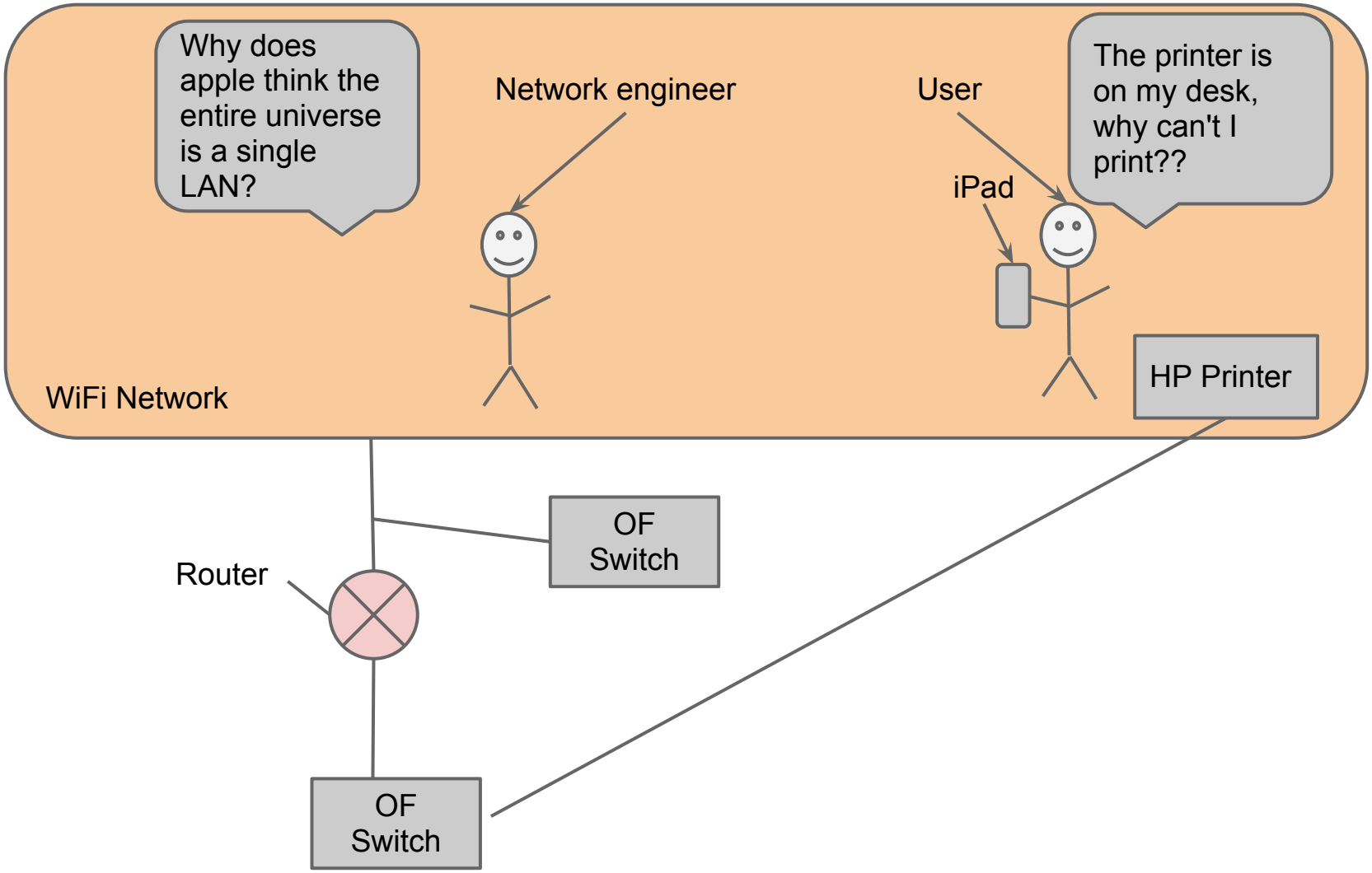
OpenFlow "peering"

GENI model - slices from multiple networks point to a single controller

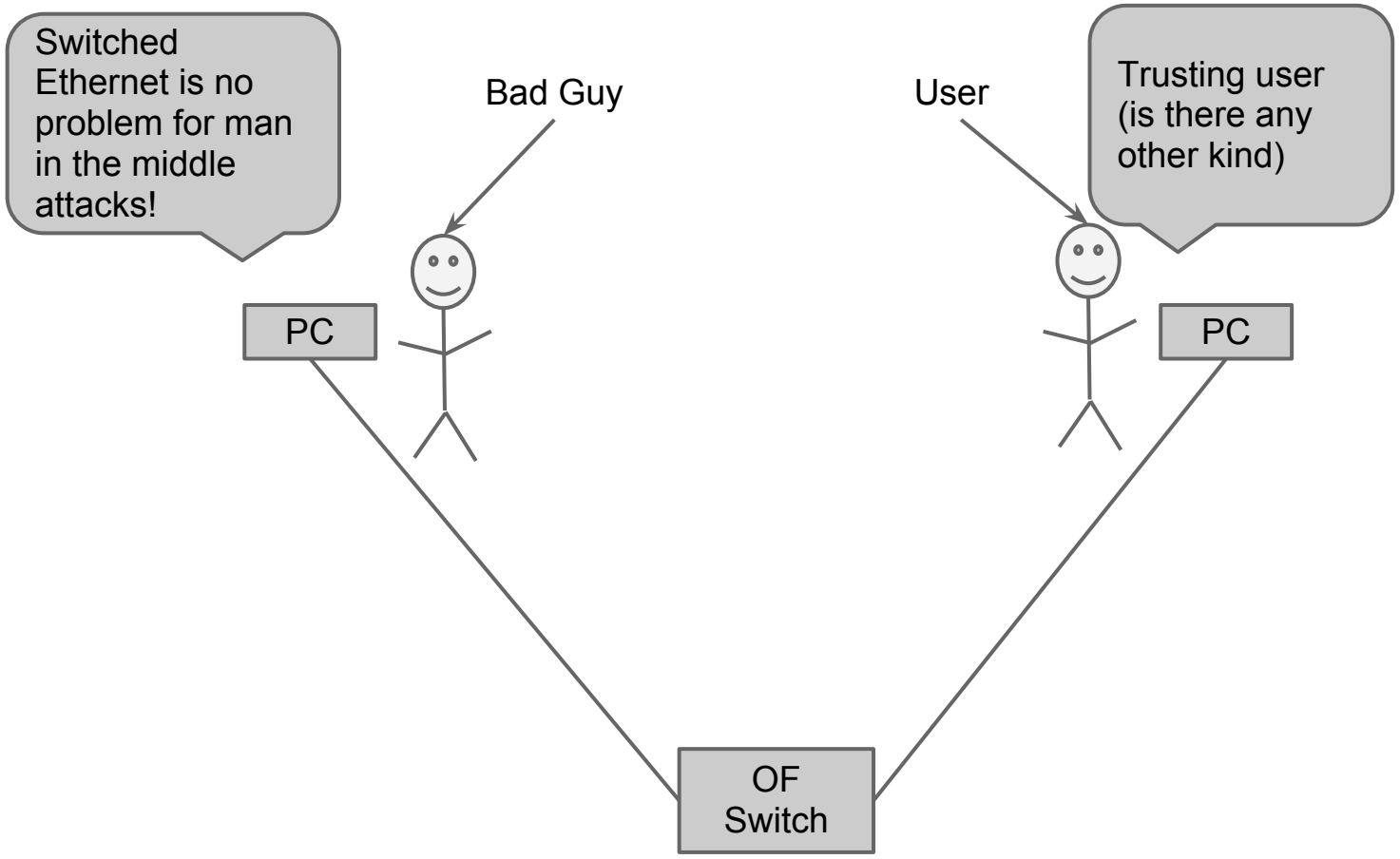
OE-SS model - interdomain is limited to abstractions supported by IDCP

something new?

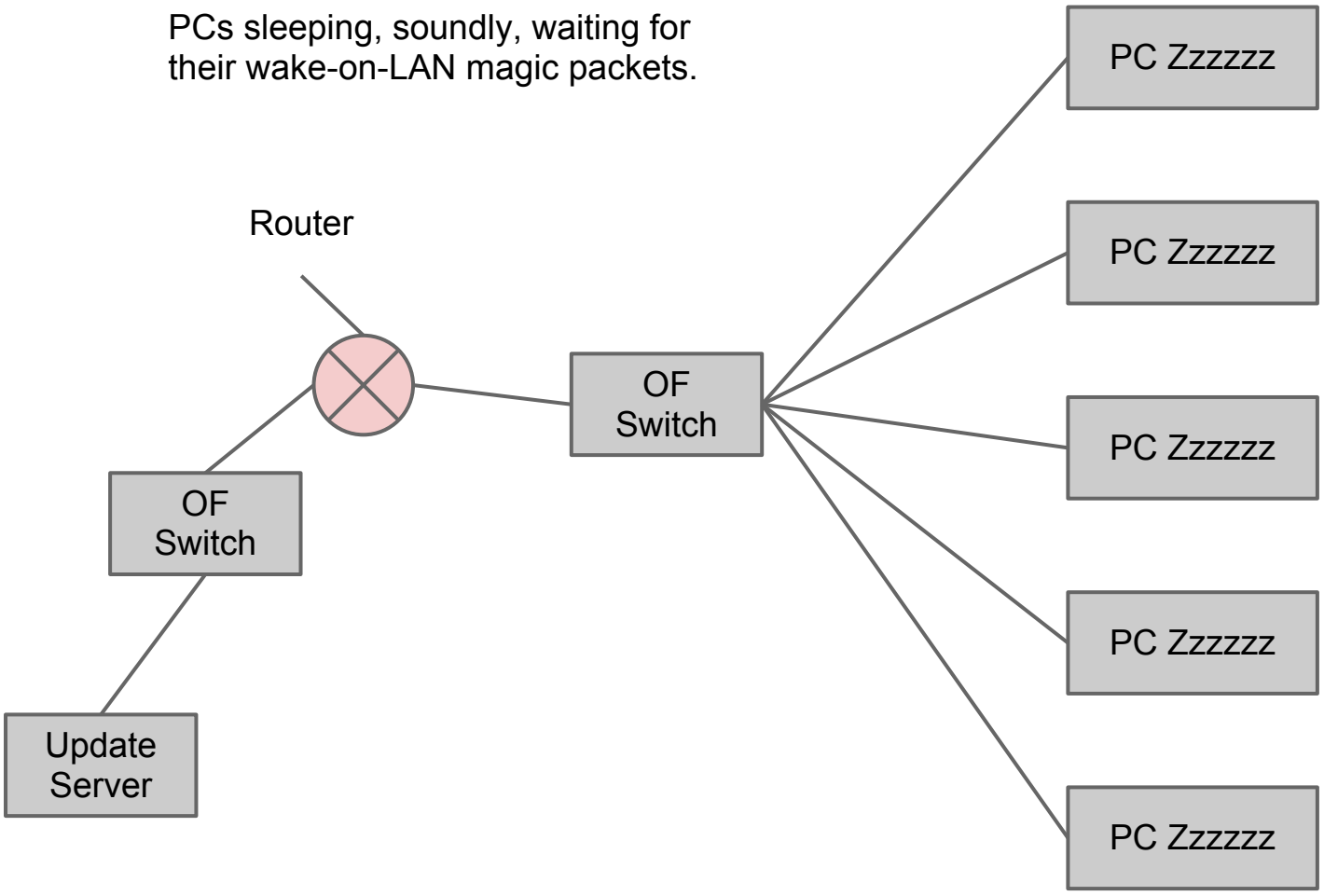
If OpenFlow was your only tool...



If OpenFlow was your only tool...

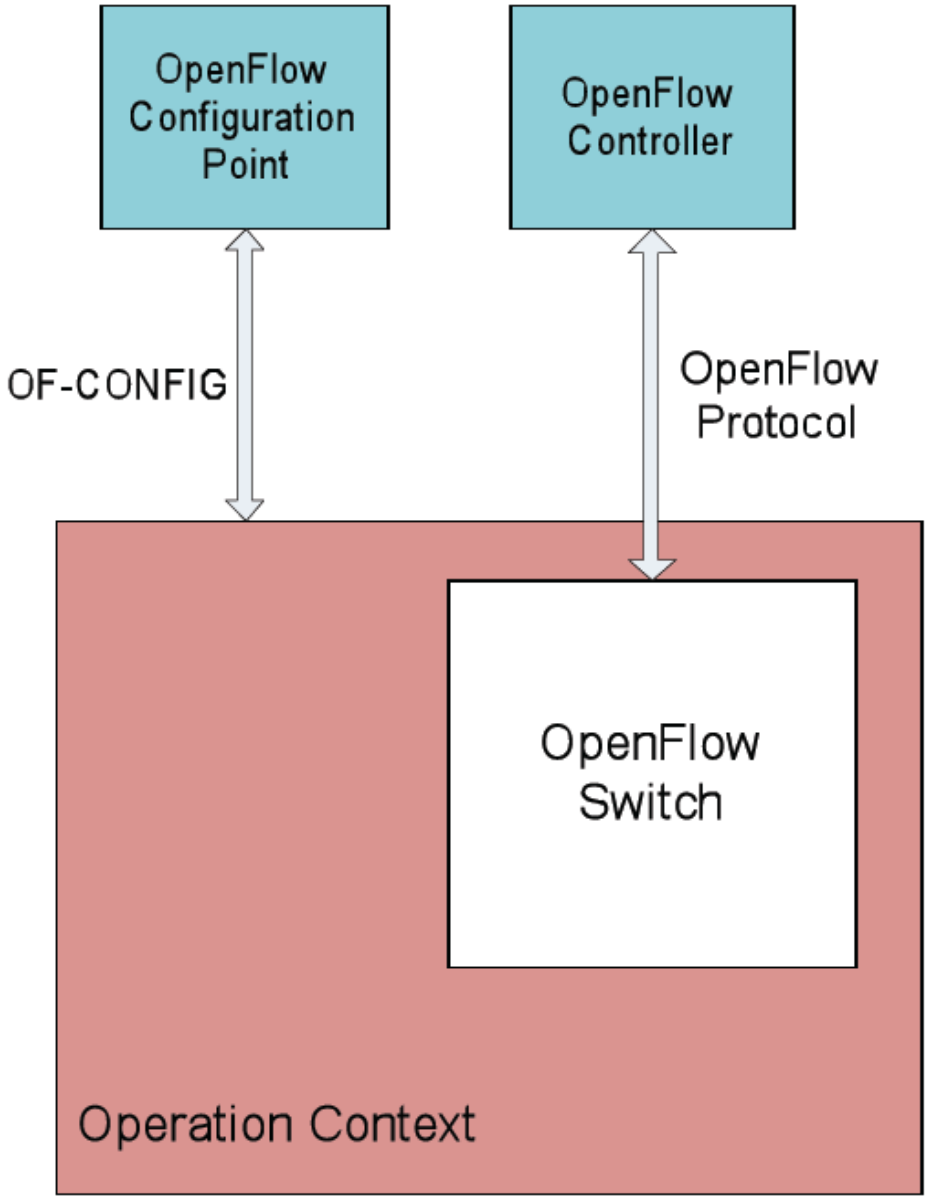


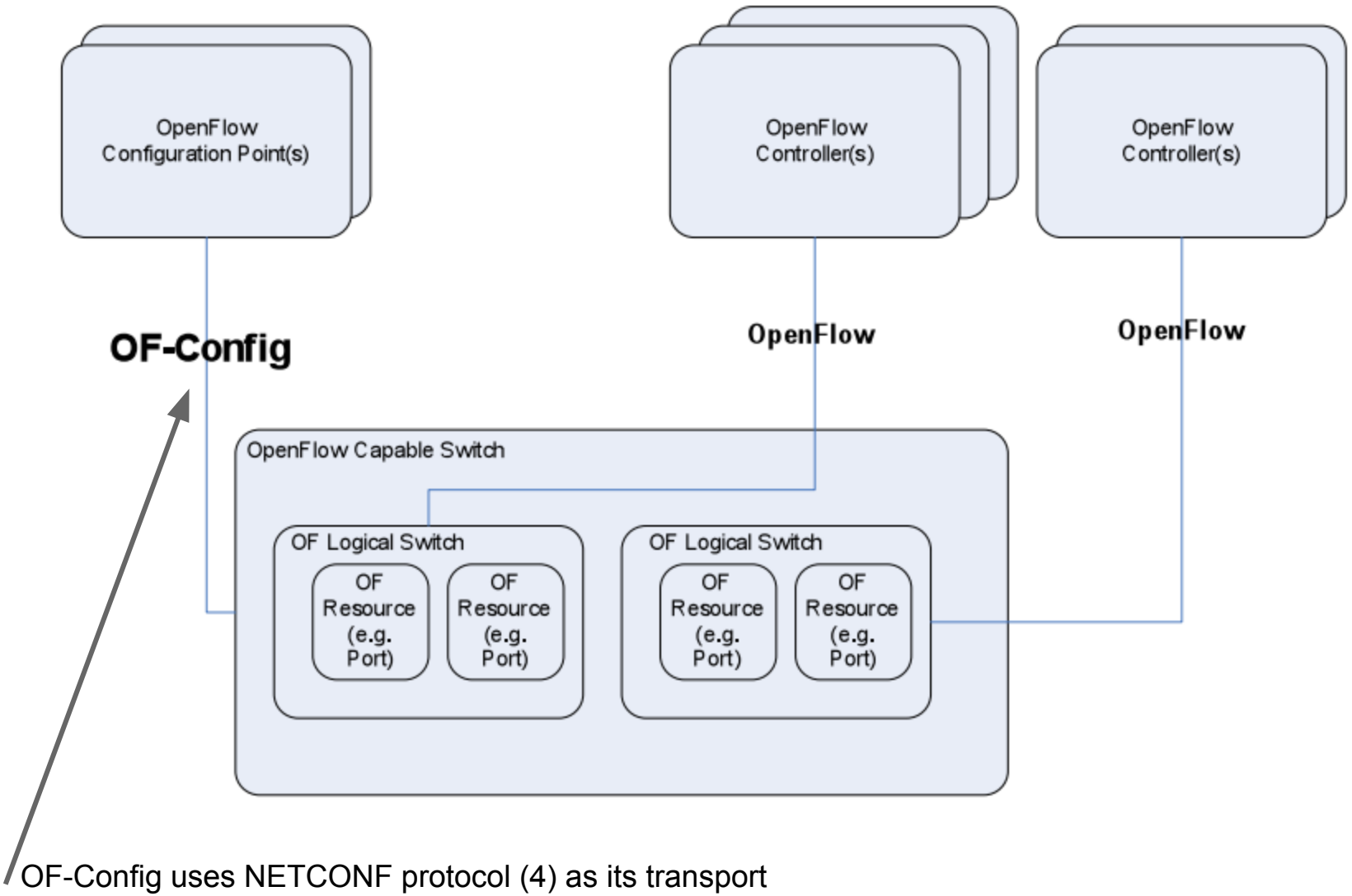
If OpenFlow was your only tool...



OF-Config 1.1

"OF-CONFIG frames an OpenFlow datapath as an abstraction called an OpenFlow Logical Switch. The OF-CONFIG protocol enables configuration of essential artifacts of an OpenFlow Logical Switch so that an OpenFlow controller can communicate and control the OpenFlow Logical switch via the OpenFlow protocol."


OpenFlow Configuration Point
OpenFlow Controller
OF-CONFIG
OpenFlow Protocol
OpenFlow Switch
Operation Context



OF-Config uses NETCONF protocol (4) as its transport

OF-CONFIG 1.1 is focused on the following functions:

- The assignment of one or more OpenFlow controllers
- The configuration of queues and ports
- The ability to remotely change some aspects of ports (e. g. up/down)
- Configuration of ceritificates for secure communication between the OpenFlow Logical Switches and OpenFlow Controllers
- Discovery of capabilities of an OpenFlow Logical Switch
- Configuration of a small set of tunnel types such as IP-in-GRE, NV-GRE, VxLAN