Sign-in here:

http://tinyurl.com/nanog57-roster

Workshop Slides:

http://tinyurl.com/nanog57-slides

Openflow 90 minutes

InCNTRE

Indiana Center for Network Translational Research and Education

the research arm of

GlobalNOC

Global Research Network Operations Center

Instructors

Steven Wallace
ssw@iu.edu

Chris Small
chsmall@indiana.edu

31 October 2012

Tools that we'll be using today...

- Amazon Web Services (EC2)
- Open VSwitch - the OpenVSwitch distribution includes an OF controller (i.e., ovs-controller) and a useful command-line utility ovs-ofclt.
- WireShark - an open source network "sniffer"
- Mininet - open source virtual network on desktop

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.

Logistics

Open the roster spreadsheet (**http://tinyurl.com/nanog57-roster**)

Find your row number, call it ***X***

Open two terminal windows via:

ssh openflow@vmX.training.incntre.org

Username: openflow

Password: openflow

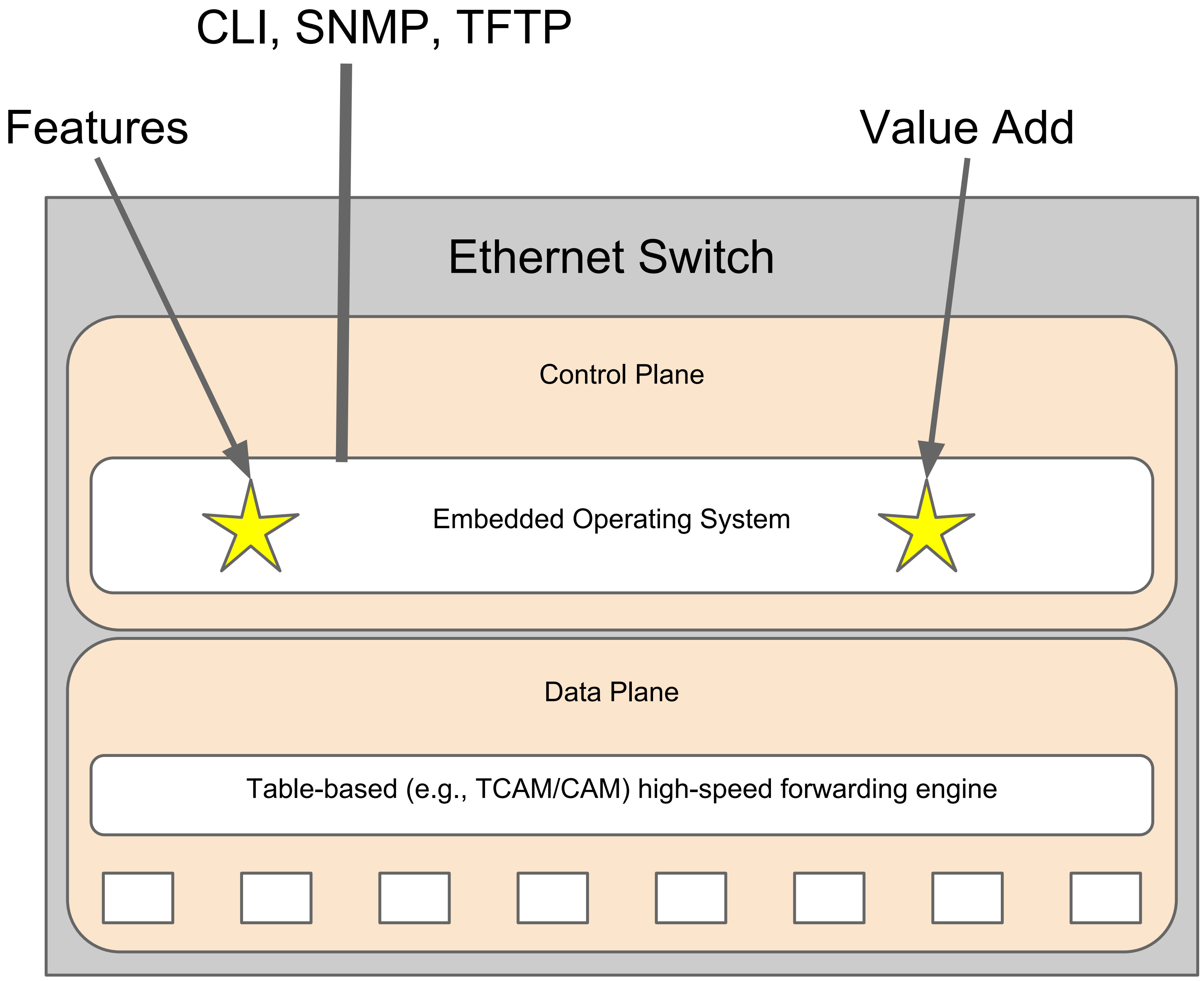
Point your browser to:

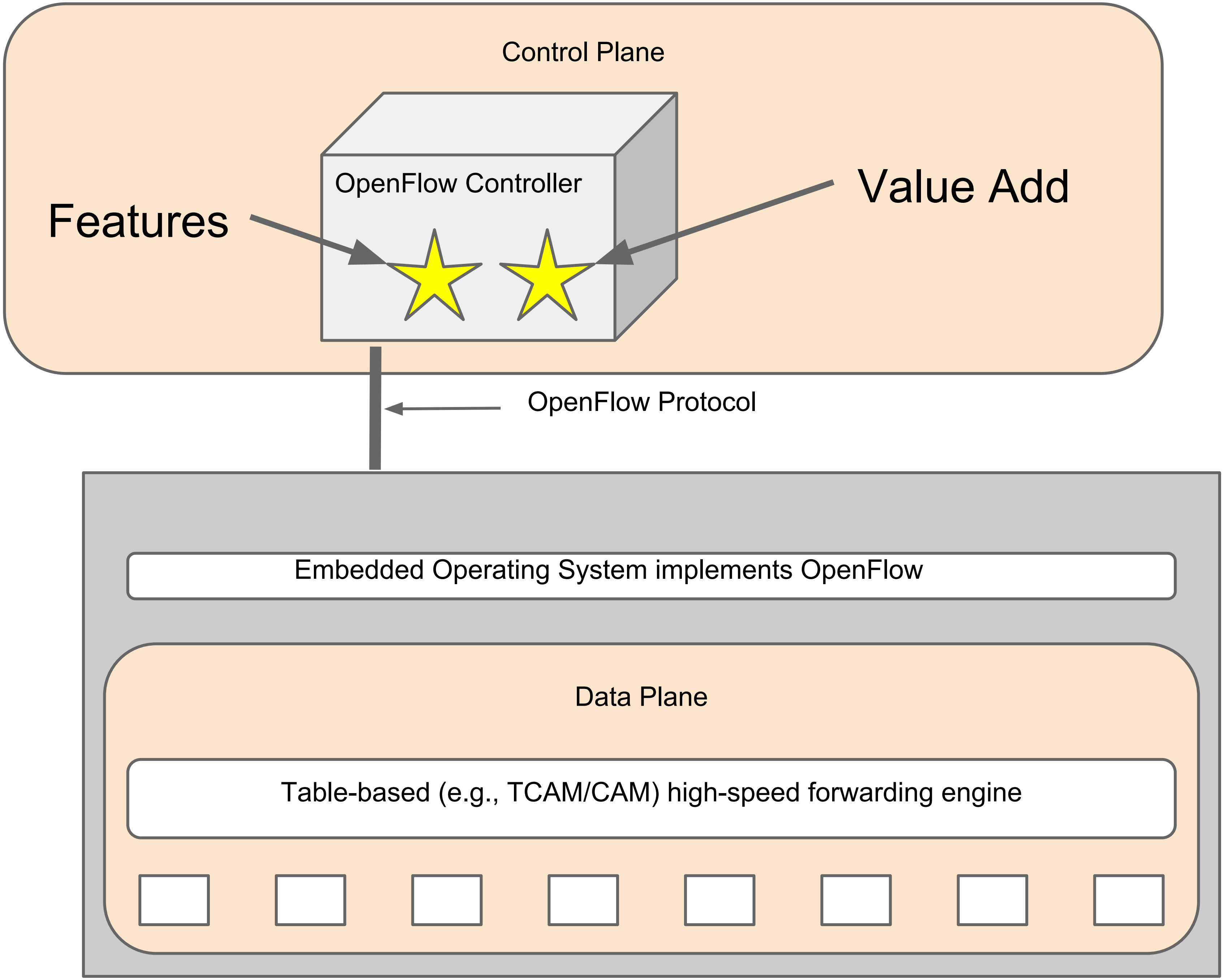
http://vmX.training.incntre.org:8090/guacamole

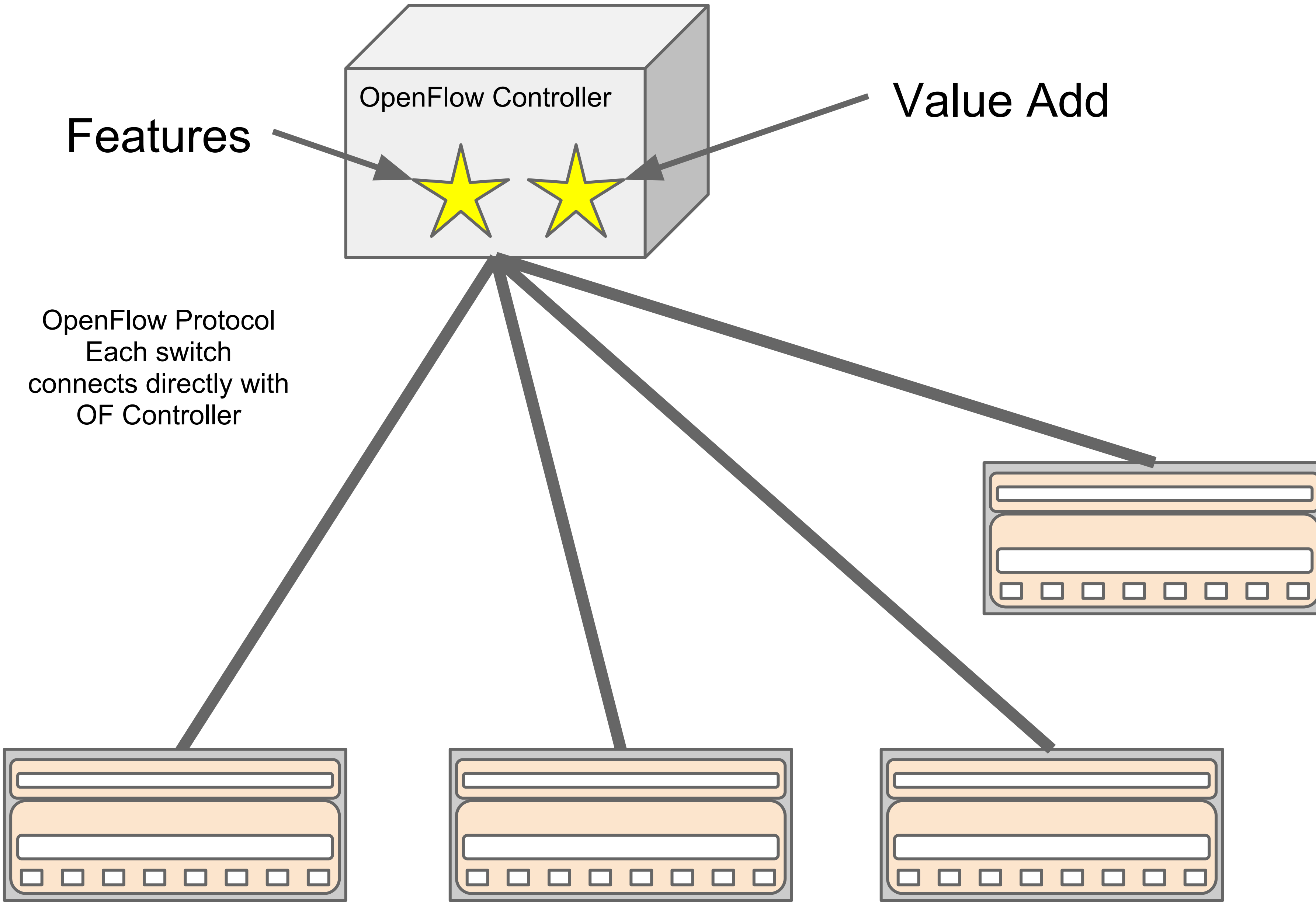
%p

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.

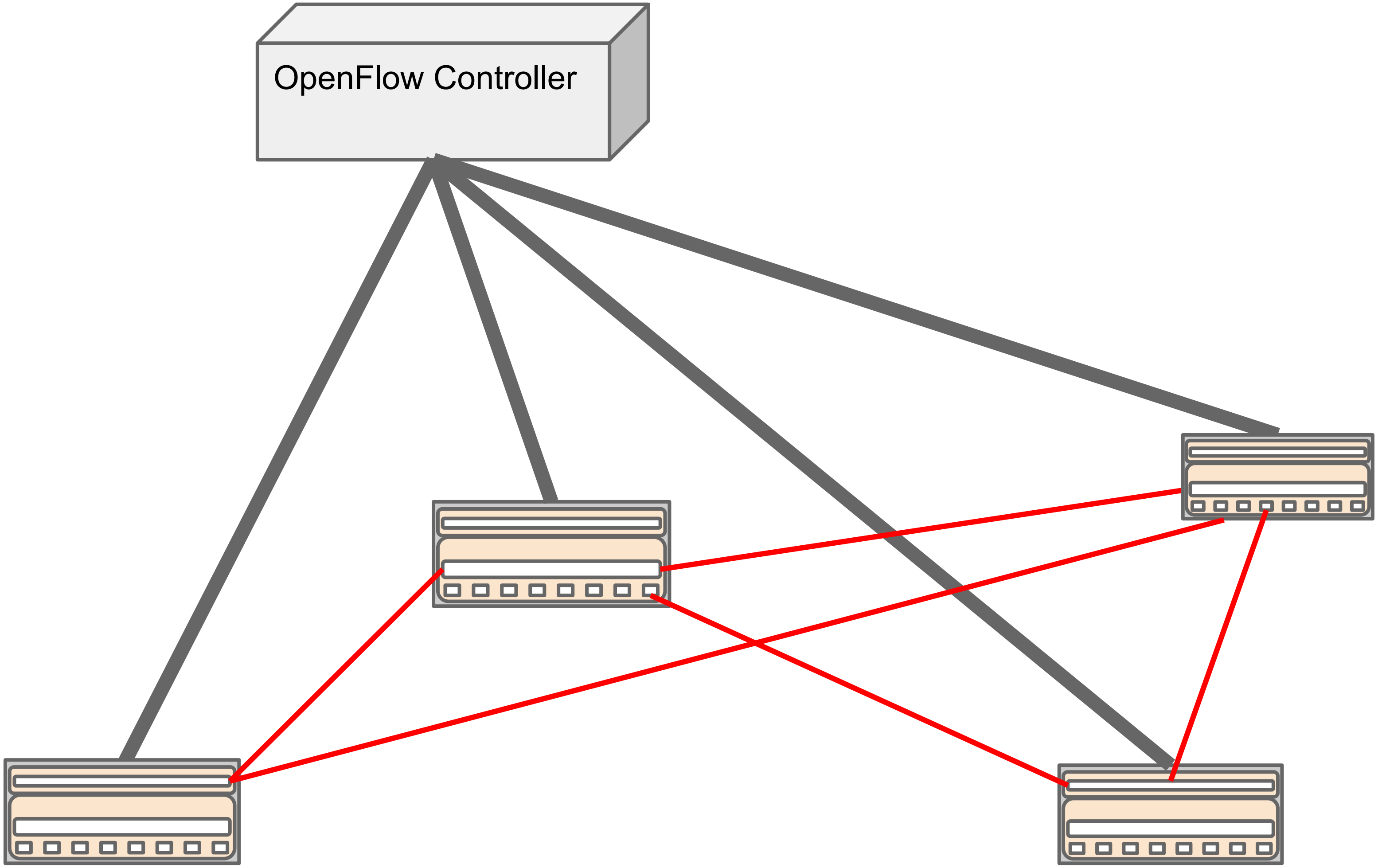
Reactive

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

Header Fields	Counters	Actions	Priority

1.1

Match Fields	Priority	Counters	Instructions	Cookie	.....

New Data Structure in Pipeline

media data	packet	Action Set

Group ID	Type	Counters	Action Buckets	.....

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 <u>may</u> be immediate action(s), or
- instructions <u>may</u> 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

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.

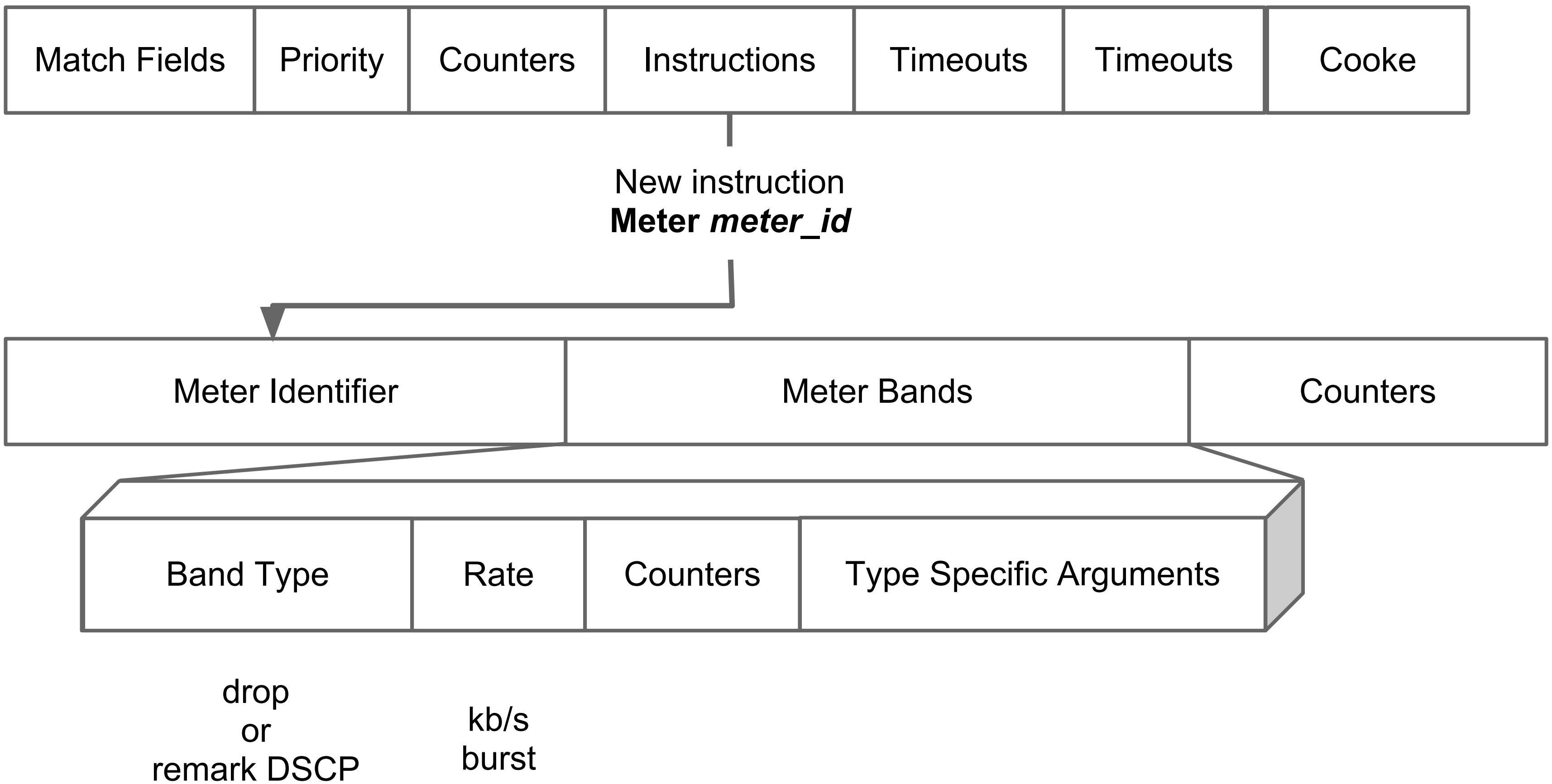
OpenFlow QoS

OF 1.3

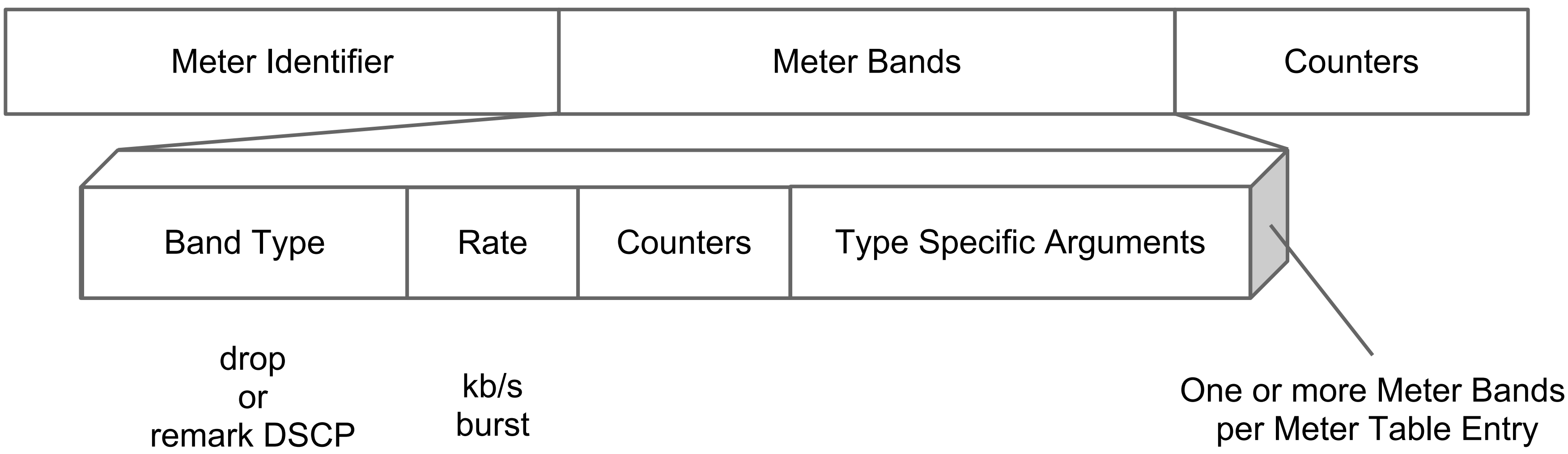
- Stuff from 1.0
- New table "Meter Table"

Meter Identifier	Meter Bands	Counters
32 bit integer used to identify the meter	list of meter bands each band specifies rate and behavior	

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"

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

Header Fields	Counters	Actions	Priority

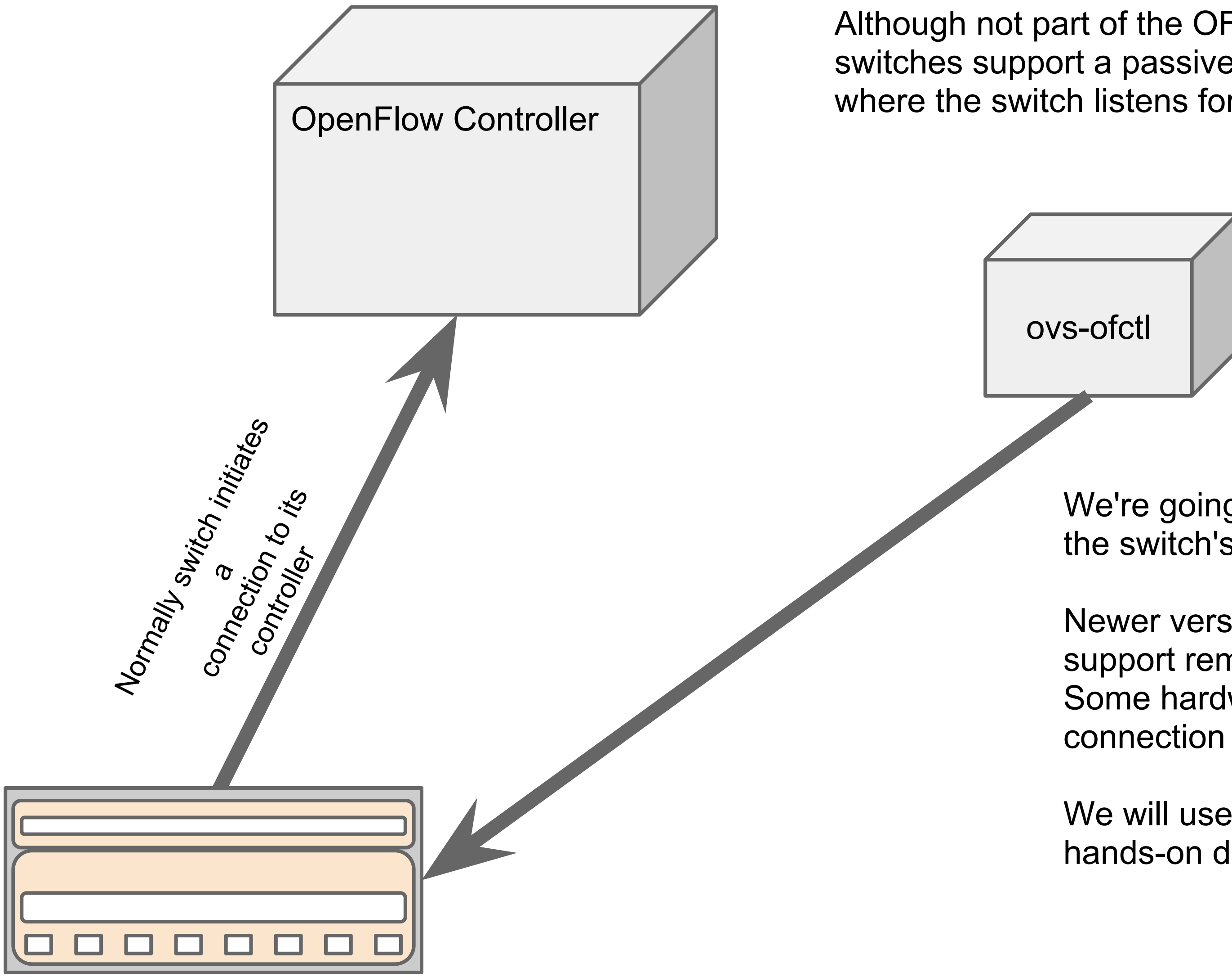
Ingress Port
Ethernet Source Addr
Ethernet Dest Addr
Ethernet Type
VLAN id
VLAN Priority
IP Source Addr
IP Dest Addr
IP Protocol
IP ToS
ICMP type
ICMP code

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

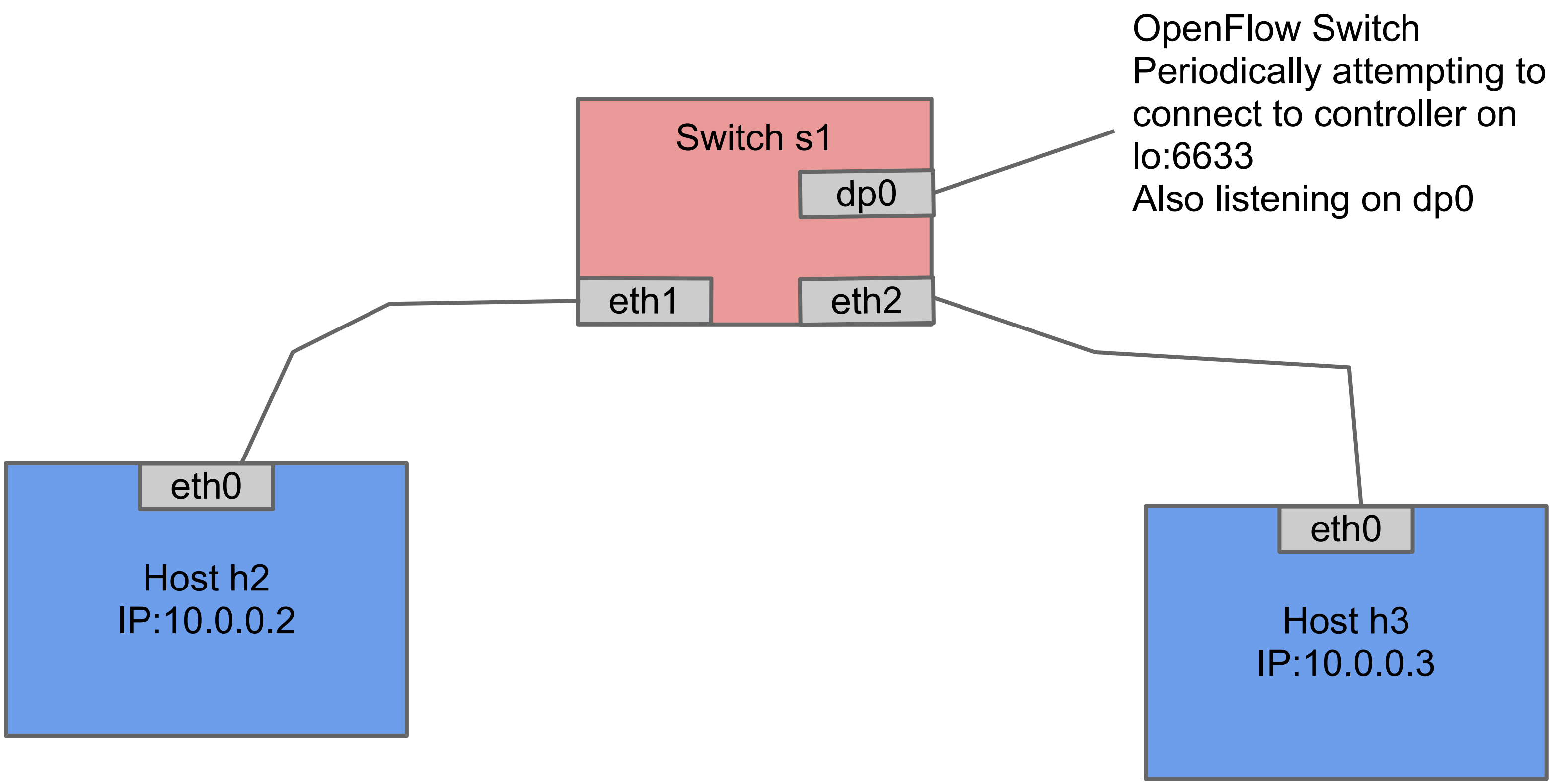
Once started, the mininet prompt "mininet>" allows commands to be run on its virtual hosts. For example

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

Exercise #1

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

mininet>***h2 ping h3***

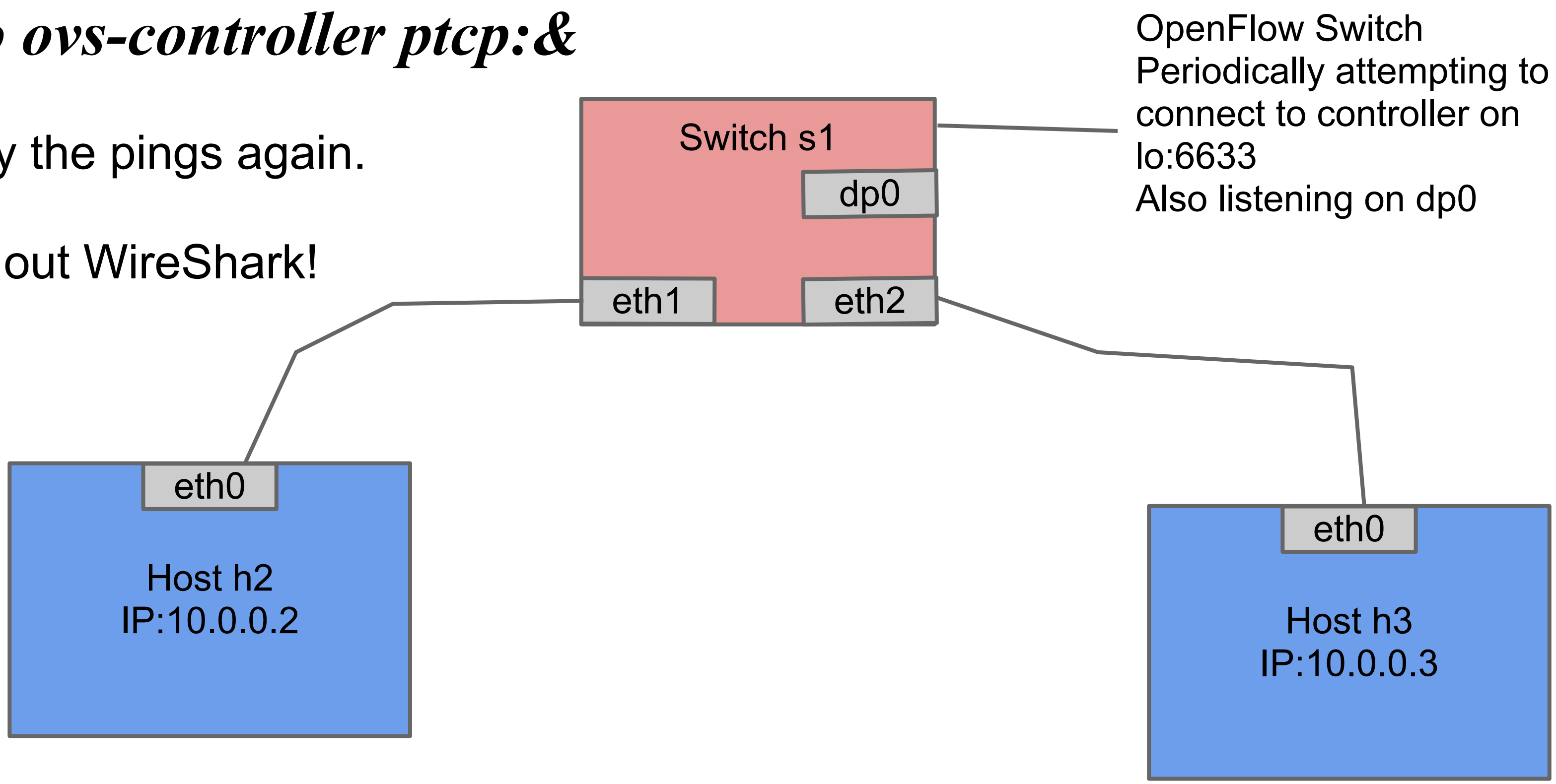
After a bit you can type control-C to stop the ping. What happened?

In the other terminal windows start the ovs-controller:

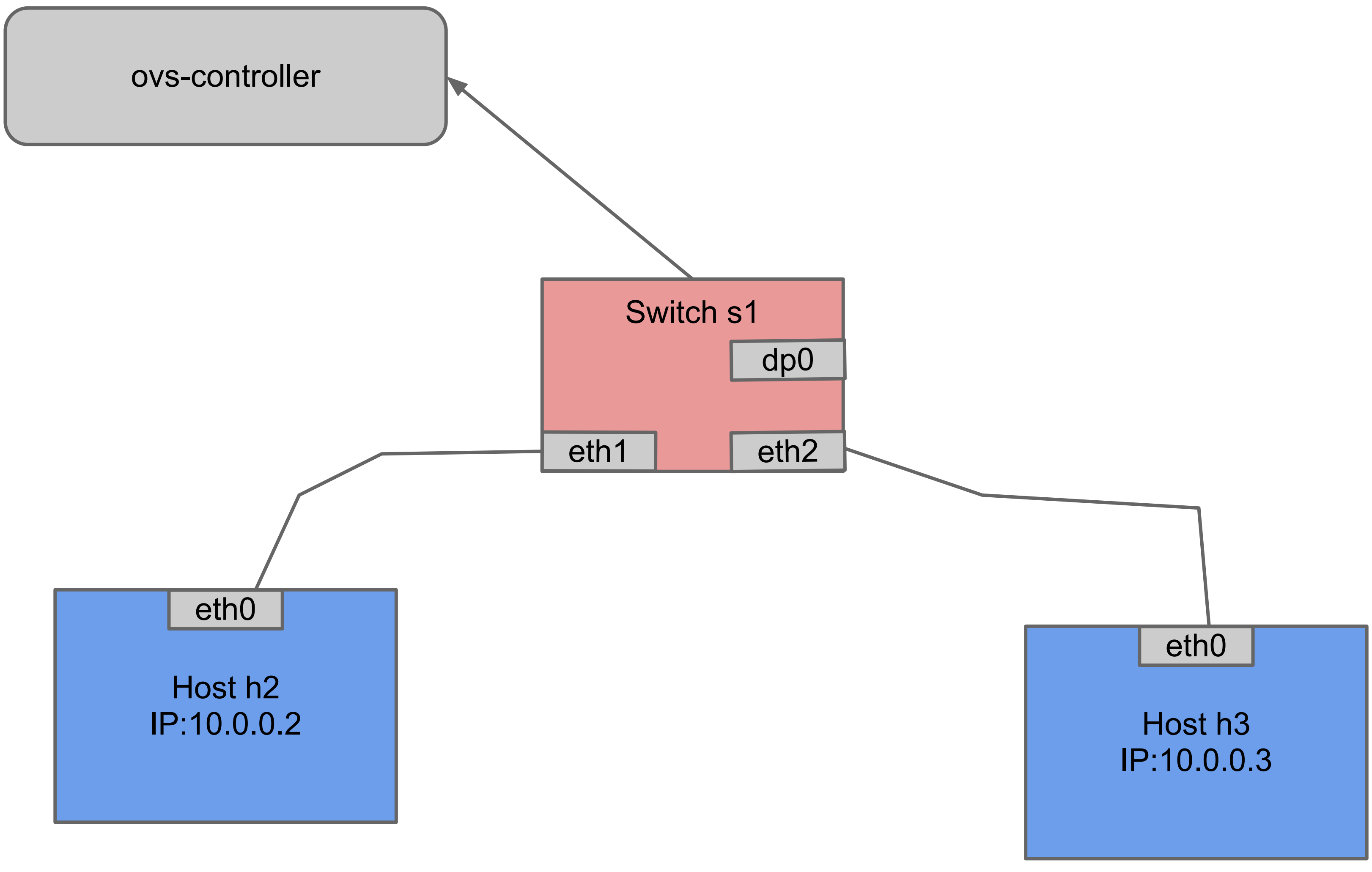
$***sudo ovs-controller ptcp:&***

Now try the pings again.

Check out WireShark!



Learning Switch



Openflow Learning Switch

Check flow table

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

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?

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.

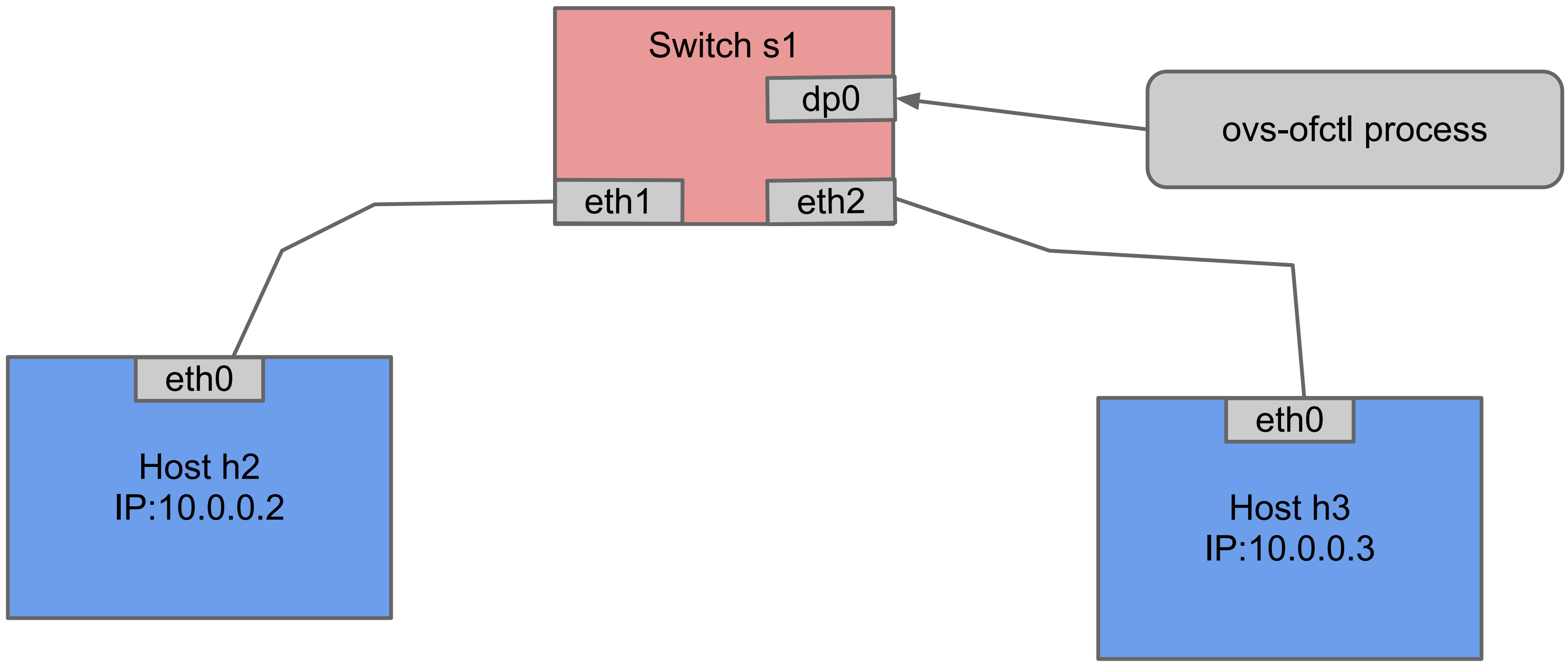
Exercise #2

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?

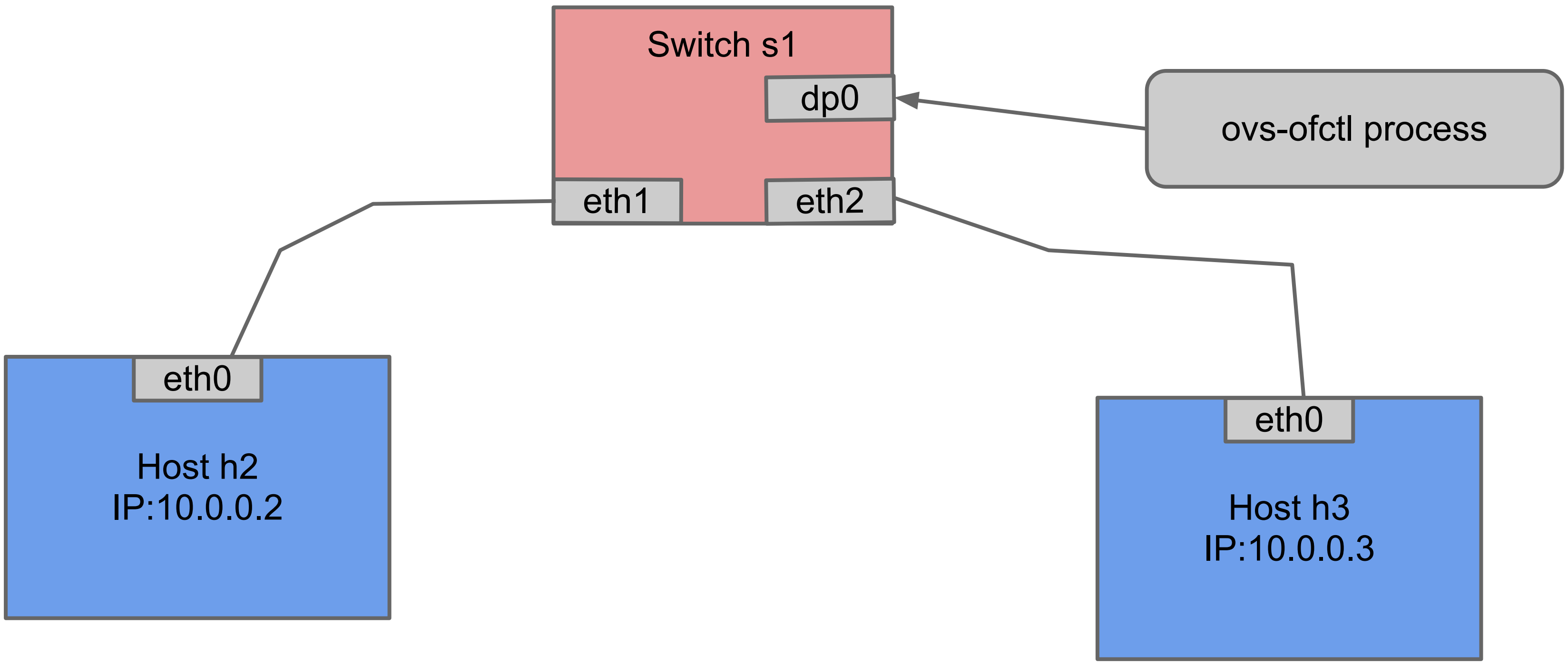
What's going on with the timeouts?

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?

Example of OpenFlow's Game Changing Potential

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)