Software Defined Networking : Primer

Muhammahmad Moinur Rahman

History of Networking

- Blackbox networking equipments
- Big name companies building switching/routing devices
- Includes Proprietary/OEM Silicon Chip
- Wrapped up with a closed source Operating System (e.g. A desktop PC with MS Windows and MS Office)

Disadvantages of Current Scenario

Technology was not designed keeping today in mind

- Massive Scalability
- Multi Tenant Networks
- Virtualization
- Cloud Computing
- Mobility (Users/Devices/VM)

Disadvantages of Current Scenario(Contd)

Protocols are Box Centric; Not Fabric Centric

- Difficult to configure correctly(consistency)
- Difficult to add new features(upgrades)
- Difficult to debug(look at all devices)

Disadvantages of Current Scenario(Contd)

Closed Systems (Vendor Hardware)

- Stuck with given interfaces (CLI, SNMP, etc.)
- Hard to meaningfully collaborate
- Vendors hesitant to open up
- No way to add new features by yourself

ANSWER: Software Defined Networking

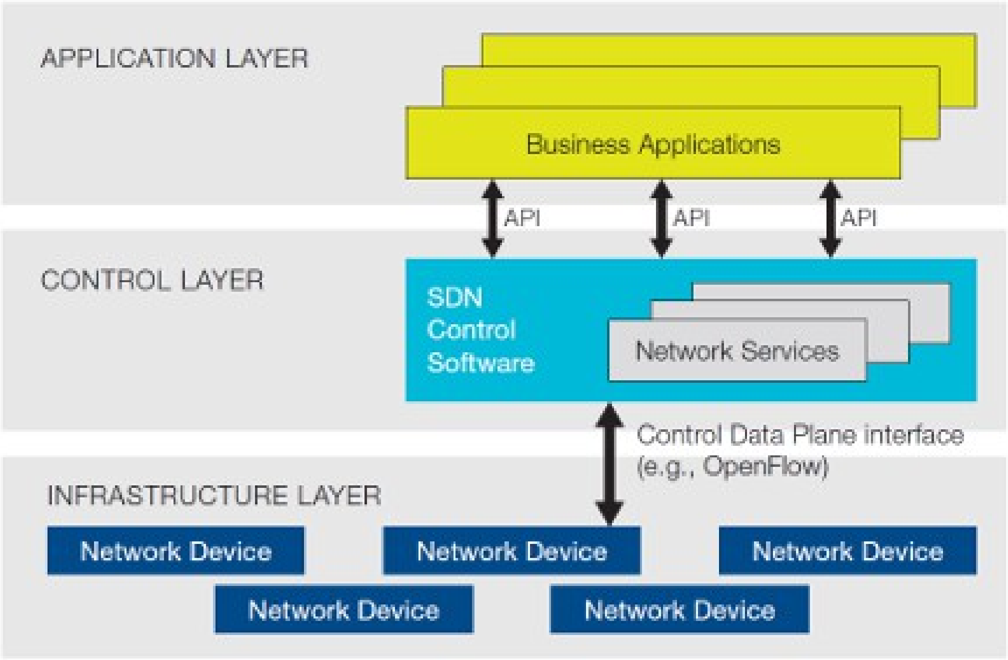
What is SDN?

SDN is a framework to allow network administrators to automatically and dynamically manage and control a large number of network devices, services, topology, traffic paths, and packet handling (quality of service) policies using high-level languages and APIs. Management includes provisioning, operating, monitoring, optimizing, and managing FCAPS (fault, configuration, accounting, performance, and security) in a multi-tenant environment.

Networking Planes

- Data Plane
  - Carries Network User Traffic
- Control Plane
  - Carries Signaling Traffic
- Management Plane
  - Carries Administrative Traffic

SDN Architecture



Need for SDN - Virtualization

Use network resource

- without worrying about where it is physically located
- how much it is
- how it is organized

Need for SDN - Orchestration

Should be able to control and manage thousands of devices with one command

Need for SDN - Programmable

Should be able to change behavior on the fly

Need for SDN - Dynamic Scaling

Should be able to change size, quantity, capacity

Need for SDN - Automation

- To lower OpEx
- Minimize manual involvement
- Troubleshooting
- Reduce downtime
- Policy enforcement
- Provisioning/Re-provisioning/Segmentation of resources
- Add new workloads, sites, devices, and resources

Need for SDN - Visibility

Monitor resources, connectivity

Need for SDN - Performance

Optimize network device utilization

- Traffic engineering/Bandwidth management
- Capacity optimization
- Load balancing
- High utilization
- Fast failure handling

Need for SDN - Multi Tenancy

Tenants need complete control over their

- Addresses
- Topology
- Routing
- Security

Need for SDN - Service Integration

Provisioned on demand and placed appropriately on the traffic path

- Load balancers
- Firewalls
- Intrusion Detection Systems (IDS)

Alternative APIs

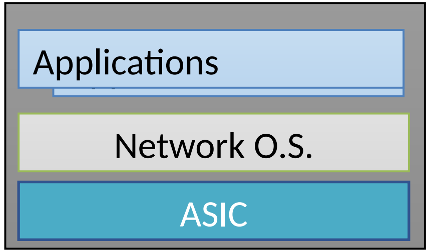
- Southbound APIs: XMPP (Juniper), OnePK (Cisco)
- Northbound APIs: I2RS, I2AEX, ALTO
- Overlay: VxLAN, TRILL, LISP, STT, NVO3, PWE3, L2VPN, L3VPN
- Configuration API: NETCONF
- Controller: PCE, ForCES

History

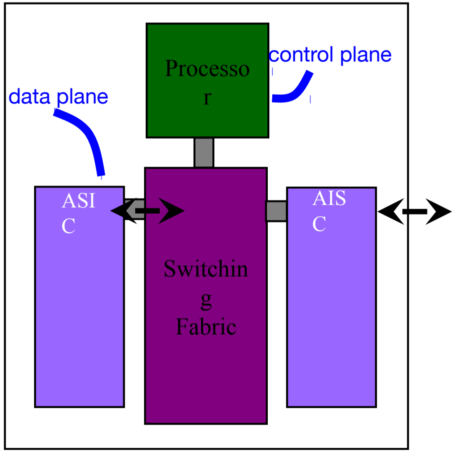
Feb, 2011 - OpenFlow 1.1 Released
Dec, 2011 - OpenFlow 1.2 Released
Feb, 2012 - “Floodlight” Project Announced
Apr, 2012 - Google announces at ONF
Jul, 2012 - Vmware acquires Nicira
Apr, 2013 - “OpenDaylight” Released

Hardware Internals

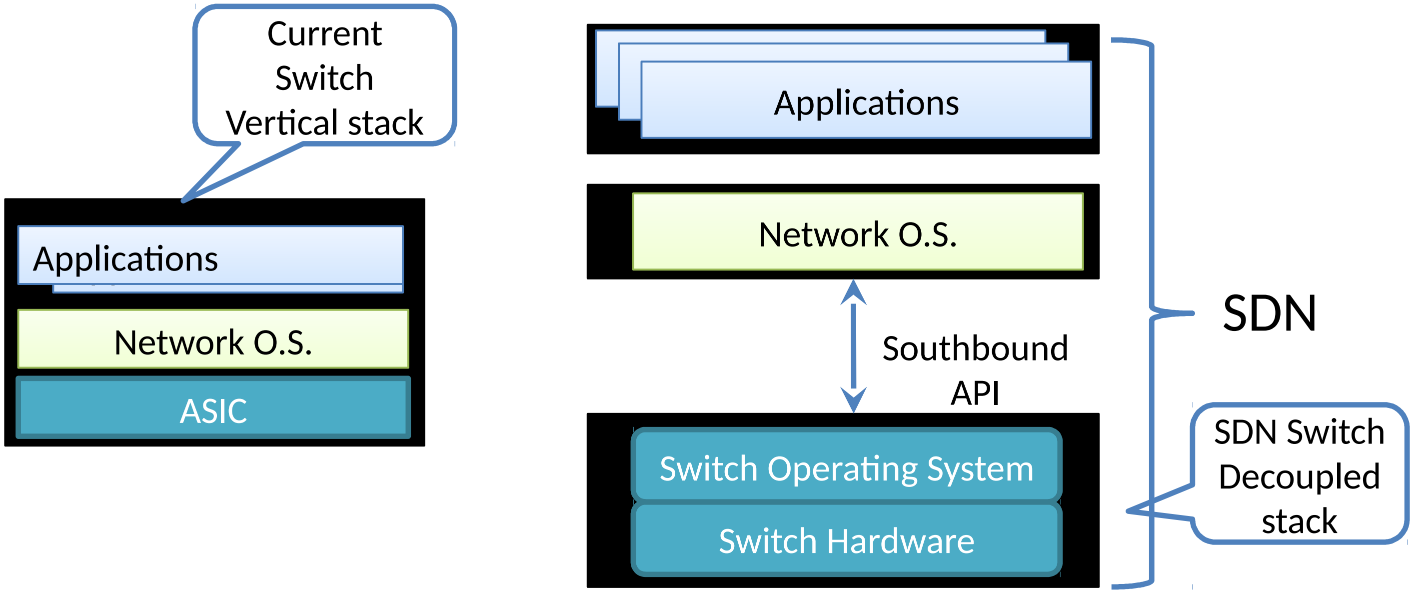
- Logical View of a Switch



- Physical Architecture of a Switch

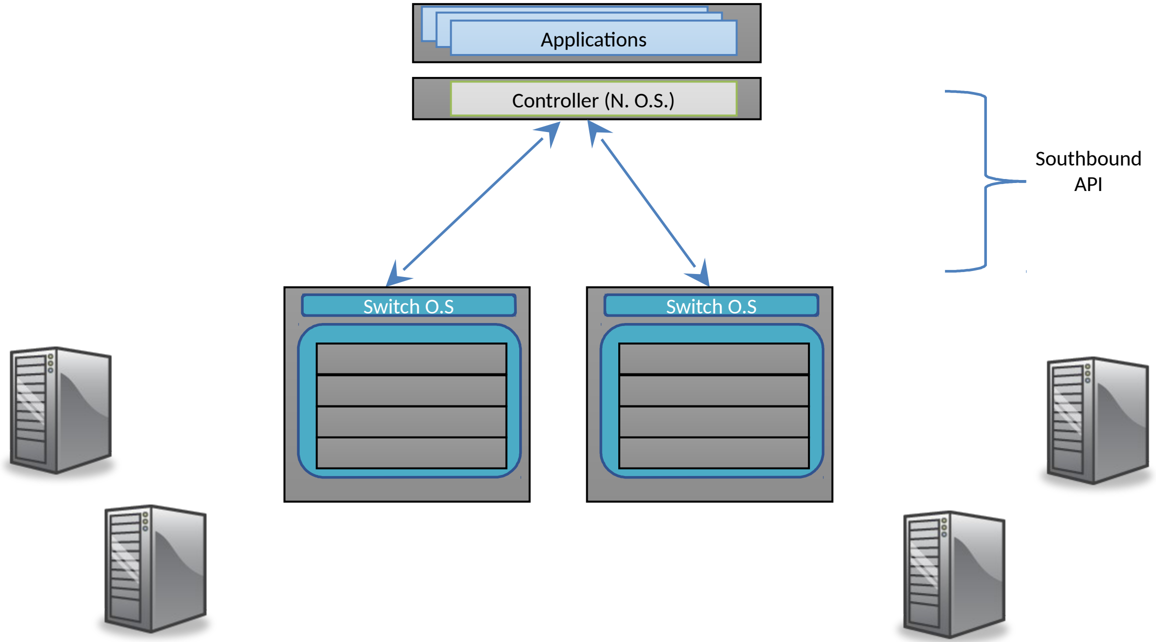


Internals of SDN

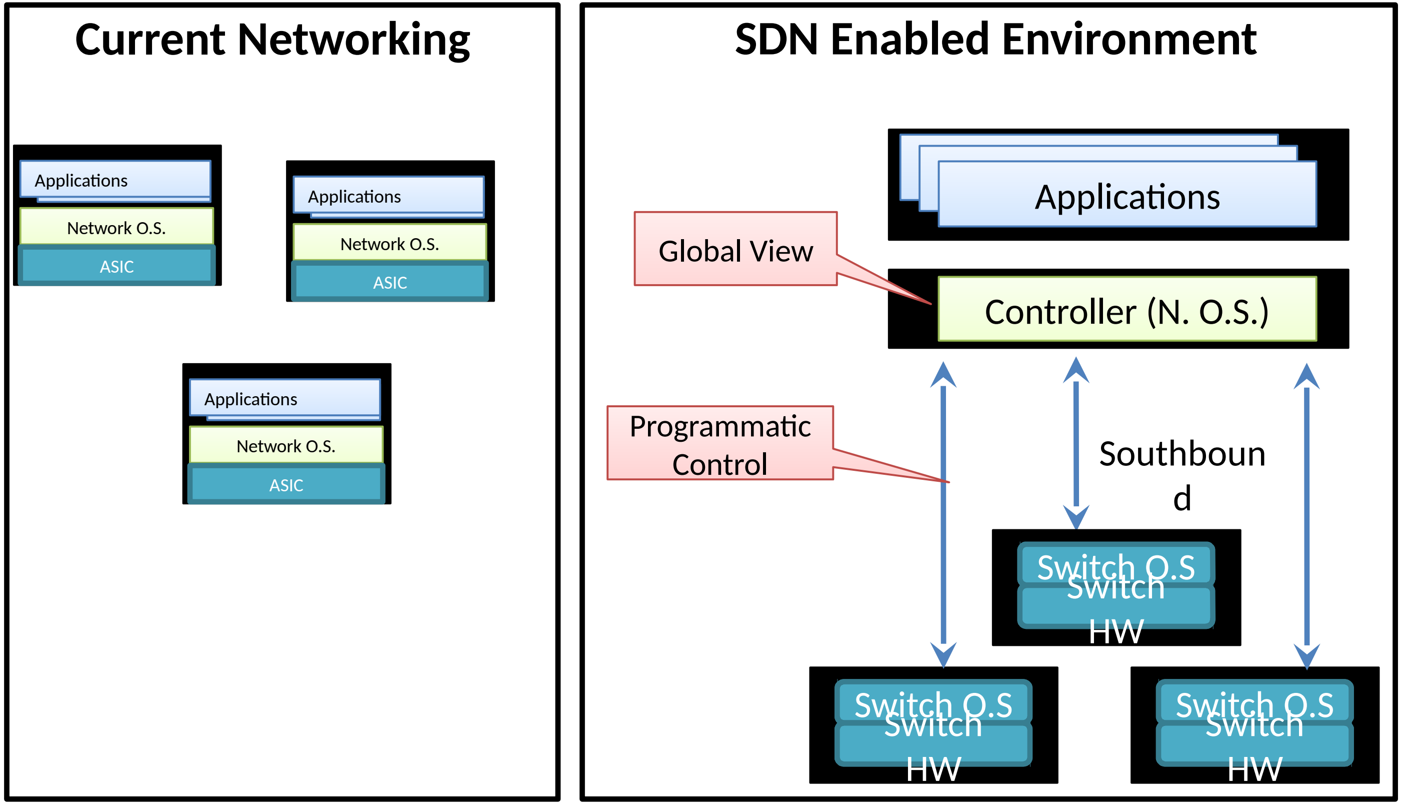


- Southbound API: decouples the switch hardware from control function
  - Data plane from control plane
- Switch Operating System: exposes switch hardware primitives

How SDN Works

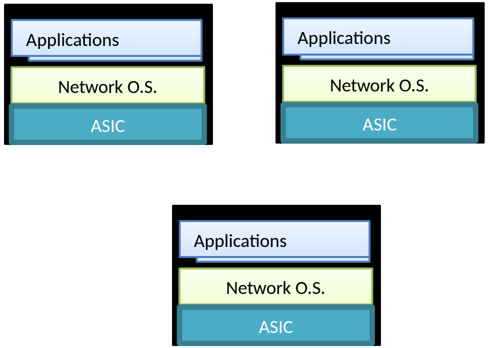


Implications of SDN



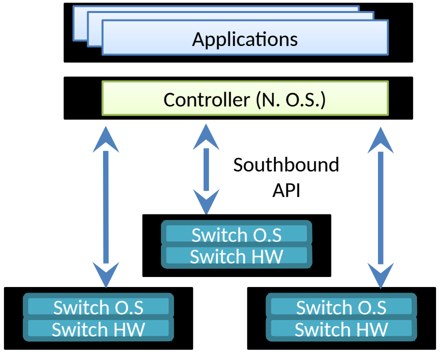
Implications of SDN(Cont)

Current Networking



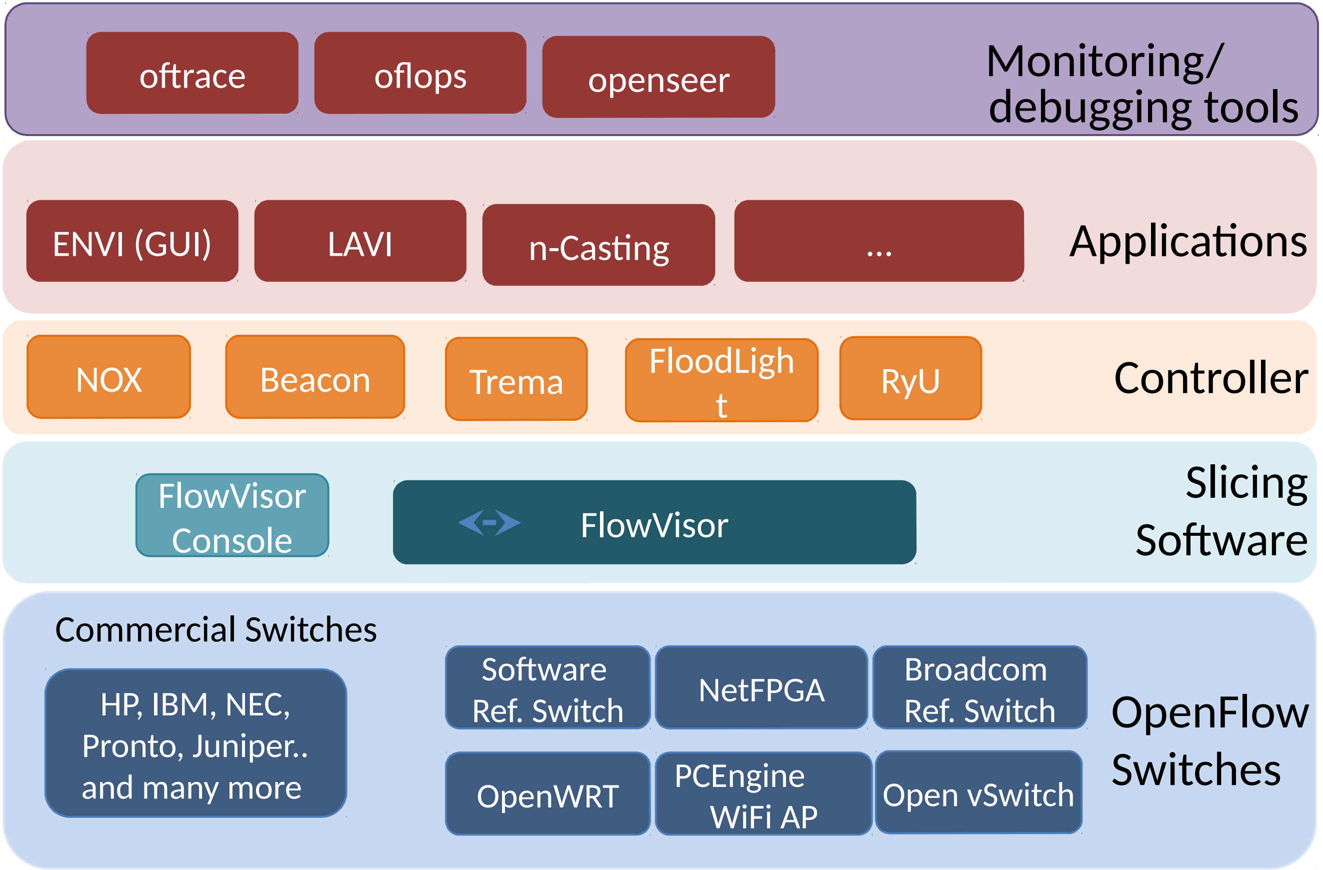
- Distributed protocols
  - Each switch has a brain
  - Hard to achieve optimal solution
- Network configured indirectly
  - Configure protocols
  - Hope protocols converge

SDN Enabled Environment



- Global view of the network
  - Applications can achieve optimal
- Southbound API gives fine grained control over switch
  - Network configured directly
  - Allows automation
  - Allows definition of new interfaces

The SDN Stack



Dimensions of SDN Environments: Vendor Devices

Vertical Stacks	Whitebox Networking

• Vendor bundles switch and switch OS
  • Restricted to vendor OS and vendor interface
• Low operational overhead
  • One stop shop
	• Vendor provides hardware with no switch OS
	• Switch OS provided by third party
	  • Flexibility in picking OS
	• High operational overhead
	  • Must deal with multiple vendors

Dimensions of SDN Environments: Switch Hardware

Virtual: Overlay

- Pure software implementation
  - Assumes programmable virtual switches
  - Run in Hypervisor or in the OS
  - Larger Flow Table entries (more memory and CPU)
- Backward compatible
  - Physical switches run traditional protocols
- Traffic sent in tunnels
  - Lack of visibility into physical network

Physical: Underlay

- Fine grained control and visibility into network
- Assumes specialized hardware
  - Limited Flow Table entries

Dimensions of SDN Environments: Southbound Interface

OpenFlow

- Flexible matching
  - L2, L3, VLAN, MPLS
- Flexible actions
  - Encapsulation: IP-in-IP
  - Address rewriting:
    - IP address
    - Mac address

BGP/XMPP/IS-IS/NetConf

- Limited matching
  - IS-IS: L3
  - BGP+MPLS: L3+MPLS
- Limited actions
  - L3/l2 forwarding
  - Encapsulation

Dimensions of SDN Environments: Controller Types

Modular Controllers

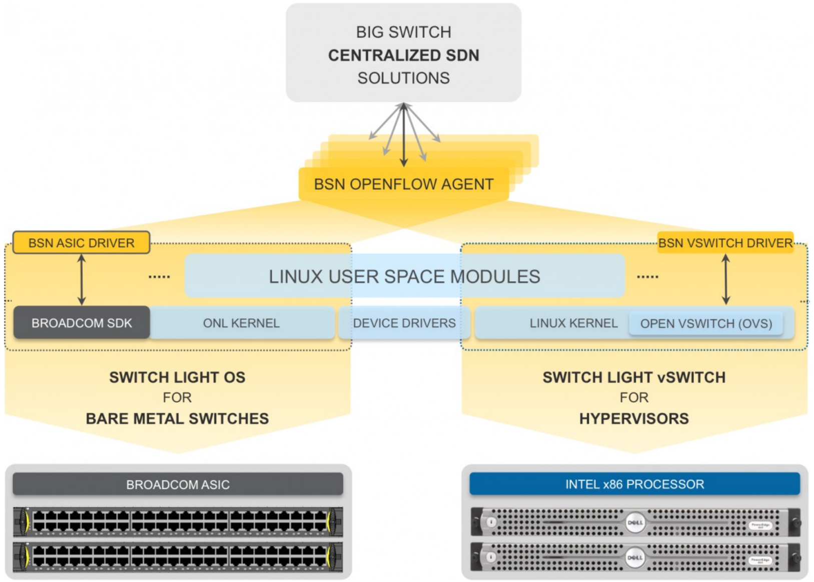
- Application code manipulates forwarding rules
  - E.g. OpenDaylight, Floodlight
- Written in imperative languages
  - Java, C++, Python
- Dominant controller style

High Level Controllers

- Application code specifies declarative policies
  - E.g. Frenetic, McNettle
- Application code is verifiable
  - Amendable to formal verification
- Written in functional languages
  - Nettle, OCamal

Ecosystem : BigSwitch

- Controller Type
  - Modular: Floodlight
- Southbound API: OpenFlow
  - OpenFlow 1.3
- SDN Device: Whitebox
  - (indigo)
- SDN Flavor
  - Underlay+Overlay



Ecosystem : Juniper

- Controller Type
  - Modular: OpenContrail
- Southbound API: XMPP/NetConf
  - BGP+MPLS
- SDN Device: Vertical Stack
  - Propriety Junos
- SDN Flavor
  - Overlay

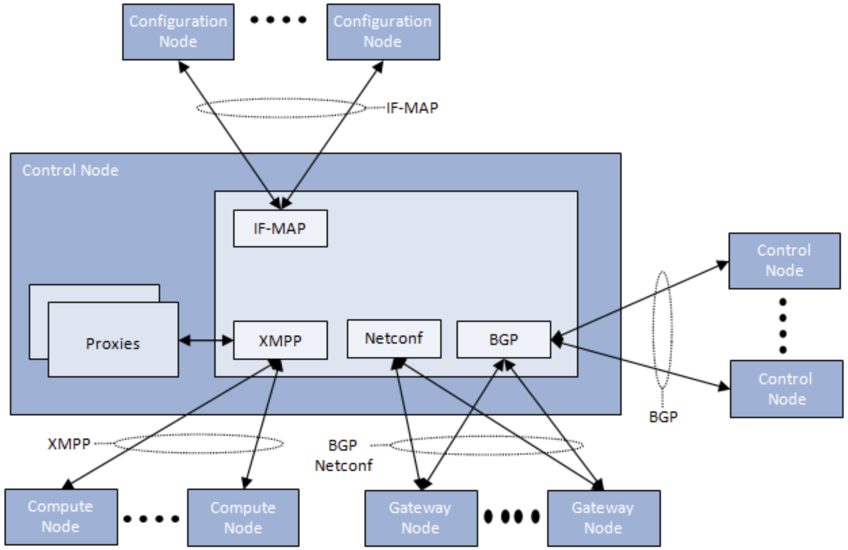


Figure 5: Internal Structure of a Control Node

SDN EcoSystem

Arista	Broadcom	Cisco
OF + proprietary	OF + proprietary	OF + proprietary
Underlay	Underlay	Underlay+Overlay
Vertical Stack	Vertical Stack	Vertical Stack

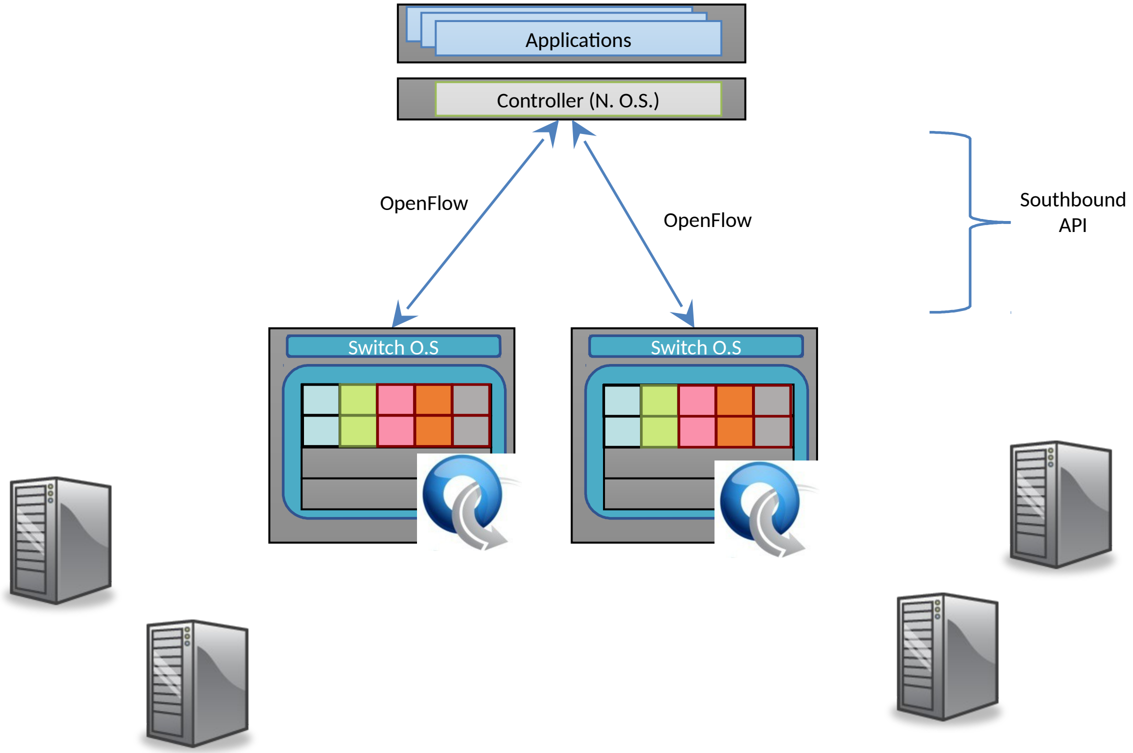
HP	Dell	FloodLight	HP
OF	OF	OF	OF
Underlay	Underlay	Underlay+Overlay	Underlay
Vertical Stack	Vertical Stack	Whitebox	Vertical Stack

Juniper	Alcatel
BGP+NetConf	BGP
Overlay	Overlay
Vertical Stack	Vertical Stack

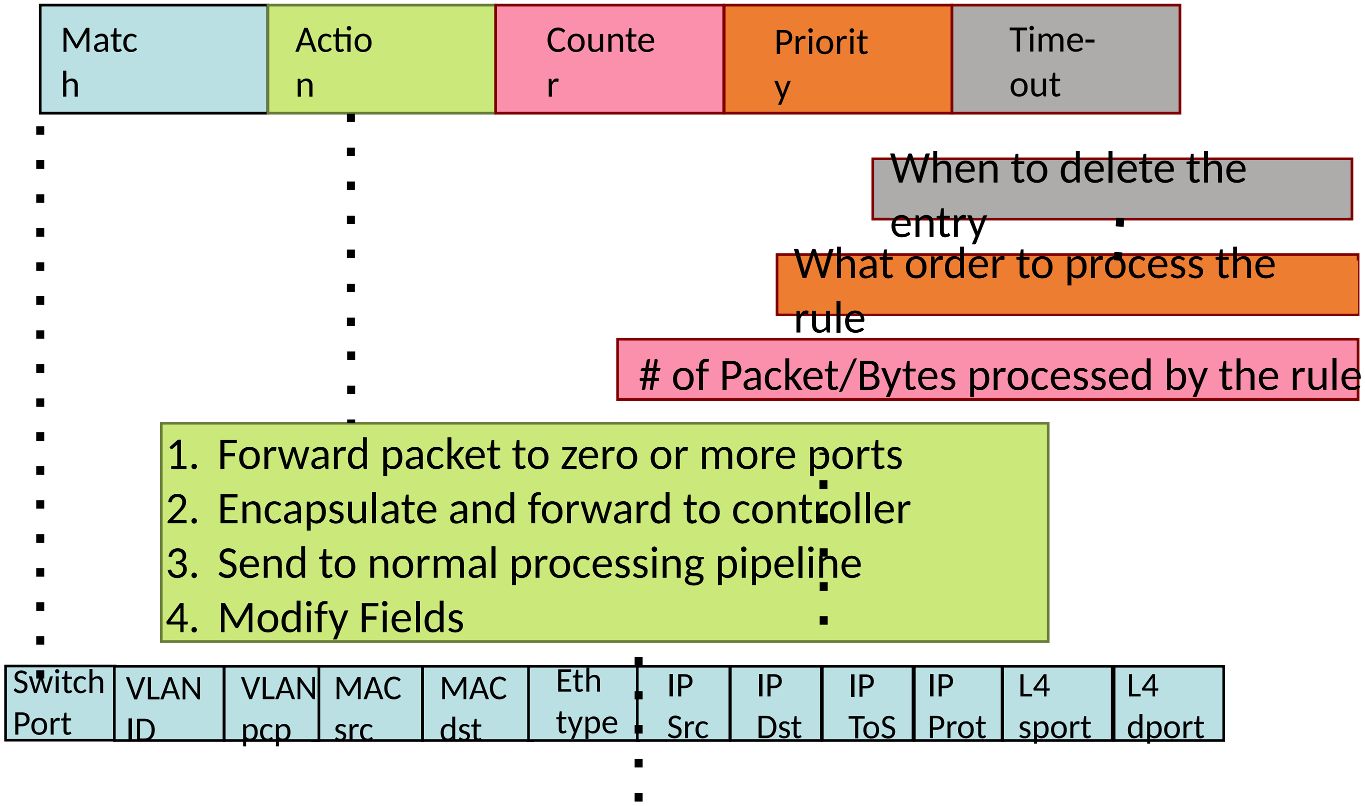
OpenFlow

- Developed in Stanford
  - Standardized by Open Networking Foundation (ONF)
  - Current Version 1.4
    - Version implemented by switch vendors: 1.3
- Allows control of underlay + overlay
  - Overlay switches: OpenVSwitch/Indigo-light

How SDN Works: OpenFlow



OpenFlow: Anatomy of a Flow Table Entry



Examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

Examples

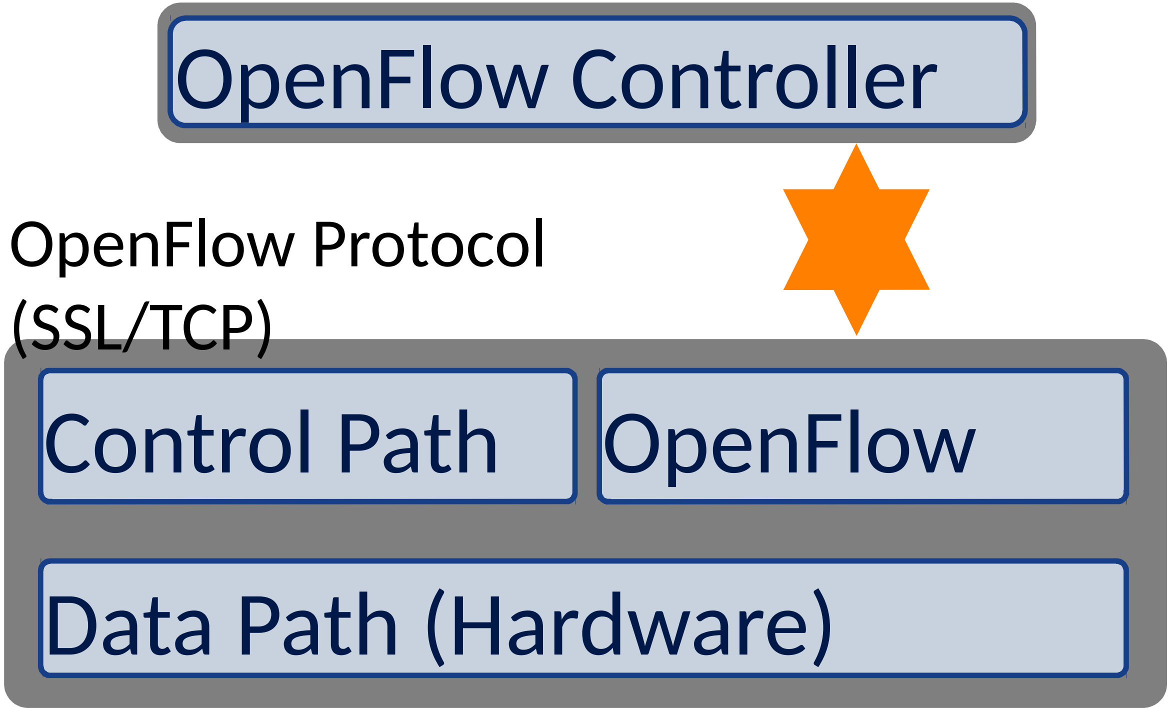
Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

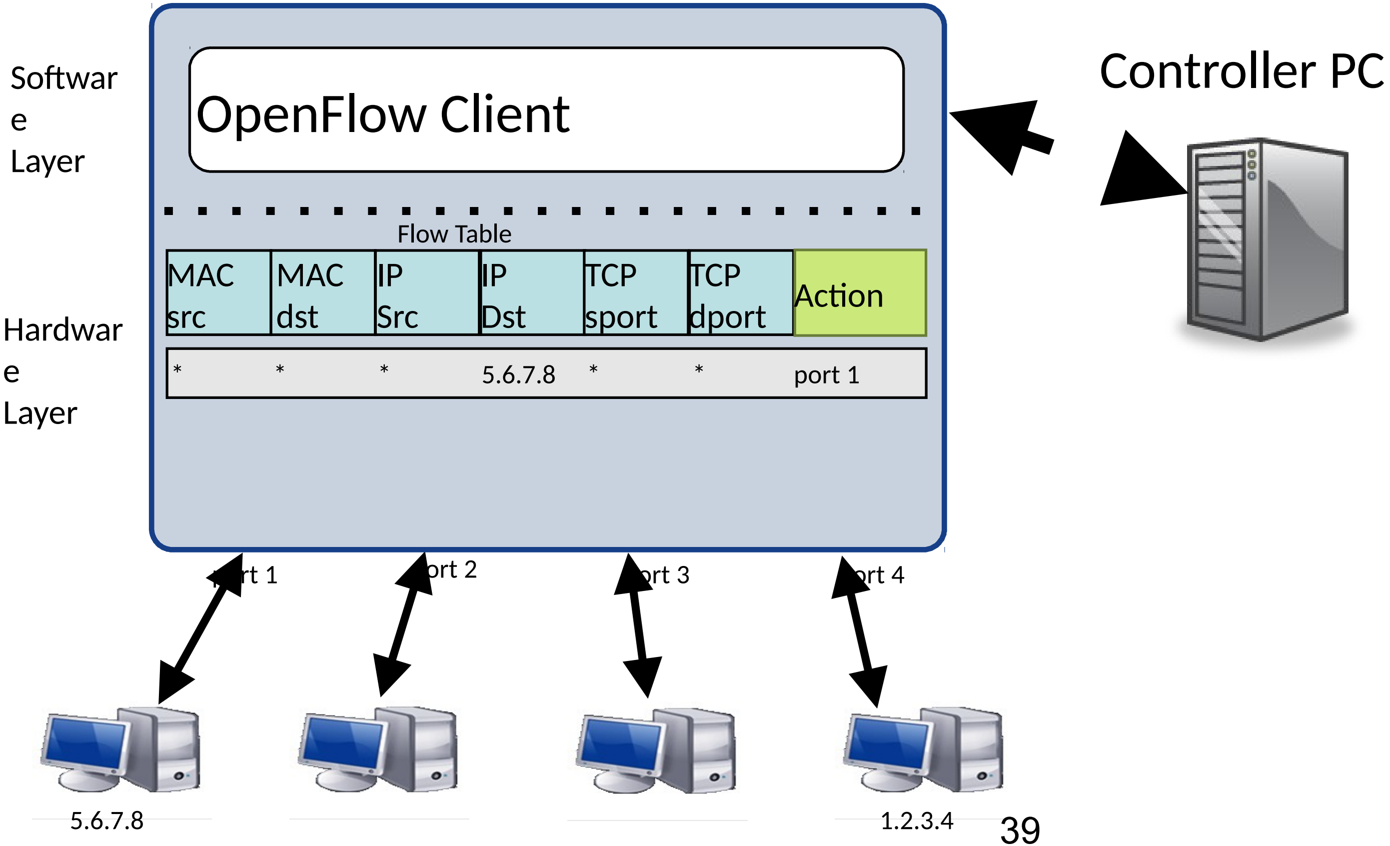
VLAN Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f..	*	vlan1	*	*	*	*	*	port6, port7, port9

OpenFlow: How it works



OpenFlow: Anatomy of a Flow Table Entry



SDN Components : Hardwares

OpenFlow Compliant (1.0-1.4) Switch

- HP 8200 ZL, 6600, 6200ZL
- Brocade 5400ZL, 3500
- IBM NetIron
- Juniper OCX1100
- Baremetal Switch
- OpenVSwitch

SDN Components : Controllers

- OpenFlow Compliant (1.0-1.4) Controller
- POX: (Python) Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for virtualization.
- IRIS: (Java) a Resursive SDN Openflow Controller created by IRIS Research Team of ETRI.
- MUL: (C) MūL, is an openflow (SDN) controller.
- NOX: (C++/Python) NOX was the first OpenFlow controller.

SDN Components : Controllers (Contd)

- Jaxon: (Java) Jaxon is a NOX-dependent Java-based OpenFlow Controller.
- Trema: (C/Ruby) Trema is a full-stack framework for developing OpenFlow controllers in Ruby and C.
- Beacon: (Java) Beacon is a Java-based controller that supports both event-based and threaded operation.
- ovs-controller (C) Trivial reference controller packaged with Open vSwitch.

SDN Components : Controllers (Contd)

- Floodlight: (Java) The Floodlight controller is Java-based OpenFlow Controller. It was forked from the Beacon controller, originally developed by David Erickson at Stanford.
- Maestro: (Java) Maestro is an OpenFlow "operating system" for orchestrating network control applications.
- NodeFlow (JavaScript) NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS.
- NDDI - OESS: OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.
- Ryu: (Python) Ryu is an open-sourced Network Operating System (NOS) that supports OpenFlow.

SDN Components : Controllers (Contd)

- NDDI - OESS: OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.
- Ryu: (Python) Ryu is an open-sourced Network Operating System (NOS) that supports OpenFlow.

Demonstration Lab

Objectives

- Basics of running Mininet in a virtual machine.
  - Mininet facilitates creating and manipulating Software Defined Networking components.
- Explore OpenFlow
  - An open interface for controlling the network elements through their forwarding tables.
- Experience with the platforms and debugging tools most useful for developing network control applications on OpenFlow.
- Run the Ryu controller with a sample application
- Use various commands to gain experience with OpenFlow control of OpenvSwitch

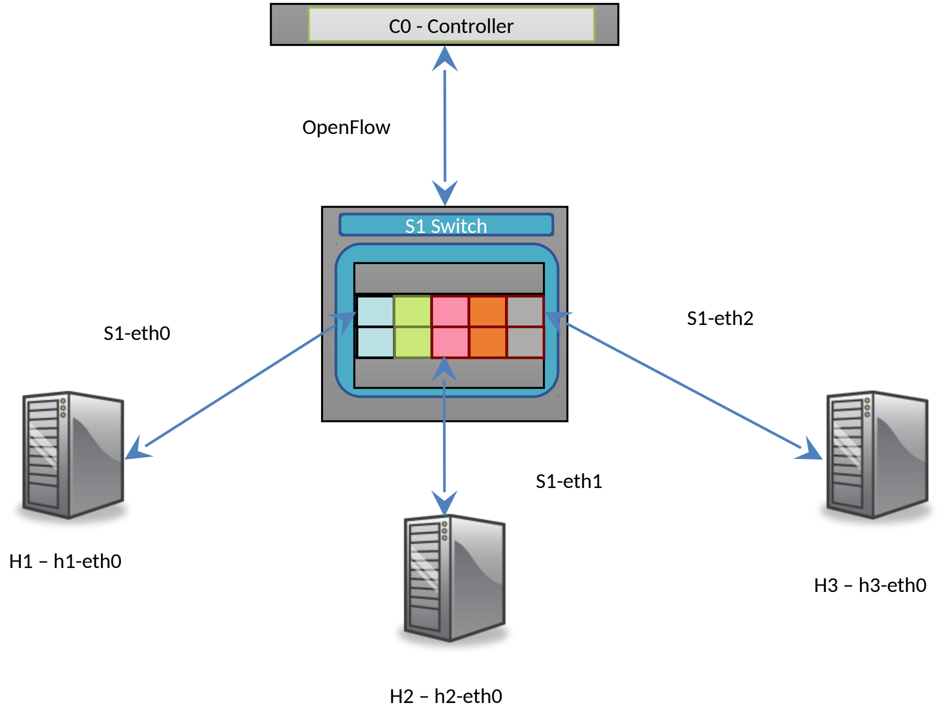
Objectives (Contd)

- Run the Ryu controller with a sample application
- Use various commands to gain experience with OpenFlow control of OpenvSwitch

Topology

- Three hosts named h1, h2 and h3 respectively. Each host has an Ethernet interface called h1-eth0, h2-eth0 and h3-eth0 respectively.
- Three hosts are connected through a switch names s1. The switch s1 has three ports named s1-eth1, s1-eth2 and s1-eth3.
- The controller is connected on the loopback interface (in real life this may or may not be the case, it means the switch and controller are built in a single box). The controller is identified as c0 and connected through port 6633.

Topology Diagram



RYU Openflow controller

Ensure that no other controller is present

```
root@mininet-vm:~# killall controller
controller: no process found
root@mininet-vm:~#
```

Note that 'controller' is a simple OpenFlow reference controller implementation in linux. We want to
ensure that this is not running before we start our own controller.

RYU Openflow controller(Cont)

Clear all mininet components

```
root@mininet-vm:~# mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-
openflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core
ovsopenflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapathsovs-vsctl --timeout=1 list-br
ovs-vsctl del-br s1
ovs-vsctl del-br s2
ovs-vsctl del-br s3
ovs-vsctl del-br s4
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '(\w+-eth\w+)'
*** Cleanup complete.
```

RYU Openflow controller(Cont)

Start the Ryu controller

```
root@mininet-vm:~# ryu-manager --verbose ./simple_switch_13.py
loading app ./simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPPacketIn
BRICK ofp_event
PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
CONSUMES EventOFPHello
CONSUMES EventOFPErrorMsg
CONSUMES EventOFPEchoRequest
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPSwitchFeatures
Understanding simple_switch.py
```

SDN vs OpenFlow

- Leading SDN protocol
- Decouples control and data plane by giving a controller the ability to install flow rules on switches(Bare Metal)
- Hardware or software switches can use OpenFlow
- Spec driven by ONF

MiniNet Environment

```
root@mininet-vm:~# mn --topo=tree,1,3 --mac --controller=remote --switch
ovsk,protocols=OpenFlow13
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

MiniNet Environment(Cont)

Monitor controller to ensure that the switch connects

```
connected socket:<eventlet.greenio.GreenSocket object at 0xa986c0c>
address: ('127.0.0.1', 42733)
connected socket:<eventlet.greenio.GreenSocket object at 0xa986cec>
address: ('127.0.0.1', 42734)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0xa9897ac>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0xb15cb575
OFPSwitchFeatures(auxiliary_id=0,capabilities=71,datapath_id=1,n_buffers
=256,n_tables=254)
move onto main mode
```

MiniNet Environment(Cont)

Dump flows on switch s1

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ----------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=2.481s, table=0,
n_packets=0, n_bytes=0, priority=0
actions=FLOOD,CONTROLLER:64
mininet>
```

MiniNet Environment(Cont)

Passing Packets

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=5.10 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.238 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.051 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.051/1.360/5.100/2.160 ms
mininet>
```

Controller Environment

Monitor new messages in the controller window

```
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in from 00:00:00:00:00:01 port 1 to 00:00:00:00:00:02 on dpid 1
associate 00:00:00:00:00:01 with port 1 on dpid 1
packet in from 00:00:00:00:00:02 port 2 to 00:00:00:00:00:01 on dpid 1
associate 00:00:00:00:00:02 with port 2 on dpid 1
add unicast flow from 00:00:00:00:00:02 port 2 to 00:00:00:00:00:01 port 1 on dpid 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in from 00:00:00:00:00:01 port 1 to 00:00:00:00:00:02 on dpid 1
add unicast flow from 00:00:00:00:00:01 port 1 to 00:00:00:00:00:02 port 2 on dpid 1
```

Mininet Environment

Dump flows again to view differences

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ------------------------------------------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=38.044s, table=0, n_packets=0, n_bytes=0,
priority=10,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=ff:ff:ff:ff:ff:ff actions=ALL
cookie=0x0, duration=37.044s, table=0, n_packets=3, n_bytes=238,
priority=100,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=38.043s, table=0, n_packets=0, n_bytes=0,
priority=10,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=ff:ff:ff:ff:ff:ff actions=ALL
cookie=0x0, duration=38.043s, table=0, n_packets=4, n_bytes=336,
priority=100,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=38.043s, table=0, n_packets=0, n_bytes=0,
priority=5,in_port=2,dl_src=00:00:00:00:00:02,dl_type=0x88cc actions=drop
cookie=0x0, duration=38.043s, table=0, n_packets=0, n_bytes=0,
priority=5,in_port=1,dl_src=00:00:00:00:00:01,dl_type=0x88cc actions=drop
cookie=0x0, duration=38.043s, table=0, n_packets=0, n_bytes=0,
priority=10,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=01:00:00:00:00:00/01:00:00
:00:00:00 actions=ALL
cookie=0x0, duration=38.044s, table=0, n_packets=0, n_bytes=0,
priority=10,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=01:00:00:00:00:00/01:00:00
:00:00:00 actions=ALL
cookie=0x0, duration=73.001s, table=0, n_packets=3, n_bytes=294, priority=0
actions=FLOOD,CONTROLLER:64
```

Mininet Environment

Running a high bandwidth flow

```
mininet> iperf
*** Iperf: testing TCP bandwidth between
h1 and h2
Waiting for iperf to start up...***
Results: ['5.52 Gbits/sec', '5.52
Gbits/sec']
mininet>
```

Mininet Environment

Dump flows to see the flows which match

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1
------------------------------------------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
...
cookie=0x0, duration=209.485s, table=0, n_packets=2384026,
n_bytes=3609389036,
priority=100,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:0a
actions=output:10
...
cookie=0x0, duration=209.485s, table=0, n_packets=27163,
n_bytes=1792770,
priority=100,in_port=10,dl_src=00:00:00:00:00:0a,dl_dst=00:00:00:00:00:0
1 actions=output:1
...
cookie=0x0, duration=392.419s, table=0, n_packets=150, n_bytes=11868,
priority=0 actions=FLOOD,CONTROLLER:6
```

Refereces

1. Mininet/Openflow Tutorials – Dean Pemberton
2. SDN – The Next Wave of Networking – Siva Valiappan

Questions