Software Defined Networking: Primer

Muhammad 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

Controller (N. O.S.)

Applications

Southbound API

Switch H.W

Switch O.S
Implications of SDN

Current Networking

SDN Enabled Environment

Applications

Network O.S.

ASIC

Controller (N. O.S.)

Applications

Global View

Programmatic Control

Southbound

Switch O.S

Switch

HW

Switch O.S

Switch

HW

Switch O.S

Switch

HW
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

- **Controller**
  - NOX
  - Beacon
  - Trema
  - FloodLight
  - RyU

- **Monitoring/debugging tools**
  - oftrace
  - oflops
  - openseer

- **Applications**
  - ENVI (GUI)
  - LAVI
  - n-Casting
  - ...

- **Slicing Software**
  - FlowVisor Console

- **OpenFlow Switches**
  - HP, IBM, NEC, Pronto, Juniper, and many more
  - Software Ref. Switch
  - NetFPGA
  - Broadcom Ref. Switch
  - OpenWRT
  - PCEngine WiFi AP
  - Open vSwitch

- **Commercial Switches**
## Dimensions of SDN Environments: Vendor Devices

<table>
<thead>
<tr>
<th>Vertical Stacks</th>
<th>Whitebox Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Vendor bundles switch and switch OS</td>
<td>• Vendor provides hardware with no switch OS</td>
</tr>
<tr>
<td>• Restricted to vendor OS and vendor interface</td>
<td>• Switch OS provided by third party</td>
</tr>
<tr>
<td>• Low operational overhead</td>
<td>• Flexibility in picking OS</td>
</tr>
<tr>
<td>• One stop shop</td>
<td>• High operational overhead</td>
</tr>
<tr>
<td></td>
<td>• Must deal with multiple vendors</td>
</tr>
</tbody>
</table>
## Dimensions of SDN Environments: Switch Hardware

<table>
<thead>
<tr>
<th>Virtual: Overlay</th>
<th>Physical: Underlay</th>
</tr>
</thead>
</table>
| • 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 |
| • Fine grained control and visibility into network  
  • Assumes specialized hardware  
  • Limited Flow Table entries |
Dimensions of SDN Environments: Southbound Interface

<table>
<thead>
<tr>
<th>OpenFlow</th>
<th>BGP/XMPP/IS-IS/NetConf</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Flexible matching</td>
<td>• Limited matching</td>
</tr>
<tr>
<td>• L2, L3, VLAN, MPLS</td>
<td>• IS-IS: L3</td>
</tr>
<tr>
<td>• Flexible actions</td>
<td>• BGP+MPLS: L3+MPLS</td>
</tr>
<tr>
<td>• Encapsulation: IP-in-IP</td>
<td>• Limited actions</td>
</tr>
<tr>
<td>• Address rewriting:</td>
<td>• L3/l2 forwarding</td>
</tr>
<tr>
<td>• IP address</td>
<td>• Encapsulation</td>
</tr>
<tr>
<td>• Mac address</td>
<td></td>
</tr>
</tbody>
</table>

- Flexible matching
  - L2, L3, VLAN, MPLS
- Flexible actions
  - Encapsulation: IP-in-IP
  - Address rewriting:
    - IP address
    - Mac address
- Limited matching
  - IS-IS: L3
  - BGP+MPLS: L3+MPLS
- Limited actions
  - L3/l2 forwarding
  - Encapsulation
## Dimensions of SDN Environments: Controller Types

<table>
<thead>
<tr>
<th>Modular Controllers</th>
<th>High Level Controllers</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Application code manipulates forwarding rules</td>
<td></td>
</tr>
<tr>
<td>• E.g. OpenDaylight, Floodlight</td>
<td></td>
</tr>
<tr>
<td>• Written in imperative languages</td>
<td></td>
</tr>
<tr>
<td>• Java, C++, Python</td>
<td></td>
</tr>
<tr>
<td>• Dominant controller style</td>
<td></td>
</tr>
</tbody>
</table>

| • 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
<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Technology Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arista</td>
<td>OF + proprietary</td>
</tr>
<tr>
<td></td>
<td>Underlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>Broadcom</td>
<td>OF + proprietary</td>
</tr>
<tr>
<td></td>
<td>Underlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>Cisco</td>
<td>OF + proprietary</td>
</tr>
<tr>
<td></td>
<td>Underlay+Overlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>HP</td>
<td>OF</td>
</tr>
<tr>
<td></td>
<td>Underlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>Dell</td>
<td>OF</td>
</tr>
<tr>
<td></td>
<td>Underlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>FloodLight</td>
<td>OF</td>
</tr>
<tr>
<td></td>
<td>Underlay+Overlay</td>
</tr>
<tr>
<td></td>
<td>Whitebox</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>Juniper</td>
<td>BGP+NetConf</td>
</tr>
<tr>
<td></td>
<td>Overlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
<tr>
<td>Alcatel</td>
<td>BGP</td>
</tr>
<tr>
<td></td>
<td>Overlay</td>
</tr>
<tr>
<td></td>
<td>Vertical Stack</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>VLAN ID</th>
<th>VLAN pcp</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP ToS</th>
<th>IP Prot</th>
<th>L4 sport</th>
<th>L4 dport</th>
</tr>
</thead>
</table>

### Matches
- Switch
- Port
- Source MAC
- Destination MAC
- Ethertype
- VLAN ID
- IP Source
- IP Destination
- IP TOS
- IP Protocol
- L4 Source Port
- L4 Destination Port

### Actions
1. Forward packet to zero or more ports
2. Encapsulate and forward to controller
3. Send to normal processing pipeline
4. Modify Fields

### Counts

### Priorities

### Timeouts

### When to delete the entry

### What order to process the rule

### # of Packet/Bytes processed by the rule
### Examples

#### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>00:1f...</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>port6</td>
</tr>
</tbody>
</table>

#### Flow Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>port3</td>
<td>00:20..</td>
<td>00:1f..</td>
<td>0800</td>
<td>vlan1</td>
<td>1.2.3.4</td>
<td>5.6.7.8</td>
<td>4</td>
<td>17264</td>
<td>80</td>
<td>port6</td>
</tr>
</tbody>
</table>

#### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
<td>drop</td>
<td>36</td>
</tr>
</tbody>
</table>
### Examples

#### Routing

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>port6</td>
</tr>
</tbody>
</table>

#### VLAN Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>00:1f..</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>vlan1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>port6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>port7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>port9</td>
</tr>
</tbody>
</table>
OpenFlow: How it works

OpenFlow Controller

OpenFlow Protocol (SSL/TCP)

Control Path  OpenFlow

Data Path (Hardware)
OpenFlow: Anatomy of a Flow Table Entry

Flow Table

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port 1</td>
<td></td>
</tr>
</tbody>
</table>

Controller PC

OpenFlow Client

Hardware Layer

Software Layer

Port 1

Port 2

Port 3

Port 4

5.6.7.8

5.6.7.8

5.6.7.8

1.2.3.4
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 Recursive 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

- **C0 - Controller**
- **Switch H.W**
- **S1 Switch**
- **OpenFlow**
- **S1-eth0**
- **S1-eth1**
- **H1 - h1-eth0**
- **H2 - h2-eth0**
- **H3 - h3-eth0**
RYU Openflow controller

Ensure that no other controller is present

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

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.
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+-\w]+)/'
*** Cleanup complete.
```
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)
moves onto main mode
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>
```
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=38.043s, 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>
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=00:00:00:00:00:01
actions=output:1
...
... cookie=0x0, duration=392.419s, table=0, n_packets=150, n_bytes=11868,
priority=0 actions=FLOOD, CONTROLLER:6
```
References

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