SESSION 3

Using ONOS as the control plane
What is ONOS?

- Open Network Operating System (ONOS)
- Provides the control plane for a software-defined network
  - Logically centralized remote controller
  - Provides APIs to make it easy to create apps to control a network
- Runs as a distributed system across many servers
  - For scalability, high-availability, and performance
- Focus on service provider for access/edge applications
  - In production at scale with a major US telecom provider controlling OpenFlow devices
ONOS releases

4-month release cycles

Avocet (1.0.0) 2014-12

... 

Loon (1.11.0) 2017-08 (*Initial P4Runtime support*)

... 

Raven (2.2.0) 2019-08 (*latest - with P4Runtime, gNMI, gNOI*)
ONOS architecture

Control and configure the network using a **global topology view** and independently of the device-specific details.

Northbound API
Device/protocol-agnostic
Java, REST, CLI, gRPC

Device driver
Allow device-specific variants of standard protocols

Shared protocol libraries

Distributed core
State management, notifications, high-availability & scale-out

Apps

- Topology API
- FlowRule API
- FlowObjective API
- Intent API
- Packet API

Network devices:
- OVS
- BMv2
- Barefoot
- Cavium
- Mellanox
- Ciena
- Cisco
- Corsa
- Fujitsu
- HP
- Huawei
- Juniper
- Lumentum
- Microsemi
- Polatis

Protocols:
- OpenFlow
- P4Runtime
- gNMI

Copyright © 2019 - Open Networking Foundation
Abstract to concrete

Flow Objective
- OF-DPA Pipeline
- Single Table Pipeline (OF 1.0)
- P4 Program Defined Pipeline

Flow Rule

Mapping through drivers
- OpenFlow
- P4Runtime
- Netconf
- ...

98
Flow objective example

Peering Router  Match on Switch port, MAC address, VLAN, IP

FlowObjective Service

HP Pipeliner
HP OpenFlow 1.3 Pipeline
  T0 mac
  T2 port-vlan
  T6 ip

BRCM OF-DPA Pipeliner
BRCM OF-DPA OpenFlow 1.3 Pipeline
  T0 port
  T1 Port-vlan
  T2 mac
  T4 ip
Driver behaviors in ONOS

● **ONOS** defines APIs to interact with device called “behaviors”
  ○ **DeviceDescriptionDiscovery** → Read device information and ports
  ○ **FlowRuleProgrammable** → Write/read flow rules
  ○ **PortStatisticsDiscovery** → Statistics of device ports (e.g. packet/byte counters)
  ○ **Pipeliner** → FlowObjective-to-FlowRules mapping logic
  ○ Etc.

● **Behavior** = Java interface

● **Driver** = collection of one or more behavior implementations
  ○ Implementations use ONOS protocol libraries to interact with device
ONOS key takeways

- **Apps are independent from switch control protocols**
  - High level network programming APIs
  - Same app can work with OpenFlow and P4Runtime devices

- **Different network programming APIs**
  - FlowRule API – pipeline-dependent
  - FlowObjective API – pipeline-independent
    - Drivers translate 1 FlowObjective to many FlowRule

- **FlowObjective API enables application portability**
  - App using FlowObjectives can work with switches with different pipelines
  - For example, switches with different P4 programs
P4 and P4Runtime support in ONOS
P4 and P4Runtime support in ONOS

ONOS originally designed to work with OpenFlow and fixed-function switches.

Extended it to:

1. Allow ONOS users to bring their own P4 program
   ○ For example, today’s tutorial

2. Allow built-in apps to control *any* P4 pipeline without changing the app
   ○ Today: topology and host discovery via packet-in / packet-out

3. Allow new apps to control custom/new protocols as defined in the P4 program
Pipeconf - Bring your own pipeline!

- Package together everything necessary to let ONOS understand, control, and deploy an arbitrary pipeline

- Provided to ONOS as an app
  - Can use .oar binary format for distribution

1. Pipeline model
   - Description of the pipeline understood by ONOS
   - Automatically derived from P4Info

2. Target-specific extensions to deploy pipeline to device
   - E.g. BMv2 JSON, Tofino binary, etc.

3. Pipeline-specific driver behaviors
   - E.g. “Pipeliner” implementation: logic to map FlowObjectives to P4 pipeline
Pipeconf support in ONOS

- **Pipeline-agnostic app (e.g. built-in apps)**
- **Pipeline-aware app**

**ONOS Core**
- Translation services: Uses pipeconf’s pipeline drivers
- Pipeconf Store

**Device drivers**
- stratum-tofino
- stratum-bmv2

**Protocol**
- P4Runtime
- gNMI
- gNOI

**gRPC**

Diagram:
- Allow control of new or uncommon data plane protocols, e.g. GTP, PPPoE, etc.
- Define flow rules using *same headers/action names as in the P4 program*. E.g match on “hdr.my_protocol.my_field”
Device discovery and pipeconf deploy

1. **my-pipeconf.oar**
   - Extensions: BMV2.JSON P4INFO

2. **Device Provider**
   - **Deploy pipeconf**
   - **Connect device**

3. **Device bmv2:1**
   - **P4Runtime Pipeline**
   - **Stratum Device Handshaker**
     - Open connection to gRPC server
     - **SetPipelineConfig**

4. **ONOS**
   - **Pipeconf Service**
     - Get pipeconf
     - Bind pipeconf+device driver
   - **PUSH**
   - **REGISTER**

**DeviceID**: bmv2:1
**Management address**: grpc://192.168.56.1:5001
**Pipeconf**: my-pipeconf
**Driver**: stratum-bmv2

**my-pipeconf.oar**
- BMV2.JSON
- P4INFO

**Extensions**: BMV2.JSON P4INFO

**DeviceID**: bmv2:1
**Management address**: grpc://192.168.56.1:5001
**Pipeconf**: my-pipeconf
**Driver**: stratum-bmv2

**Open connection to gRPC server**

**Deploy pipeconf**

**Connect device**

**PUSH**

**REGISTER**
Flow operations

Pipeconf-based 3 phase translation:

1. Flow Objective → Flow Rule
   - Maps 1 flow objective to many flow rules

2. Flow Rule → Table entry
   - Maps standard headers/actions to P4-defined ones
     E.g. ETH_DST→“hdr.ethernet.dst_addr”

3. Table Entry → P4Runtime message
   - Maps P4 names to P4Info numeric IDs

Define flow rules using same headers/action names as in the P4 program. E.g. match on “hdr.my_protocol.my_field”
Pipeline Interpreter

- Driver behavior
- Provides mapping between ONOS well-known types and P4 program-specific ones

<table>
<thead>
<tr>
<th>Mapping</th>
<th>ONOS (Java)</th>
<th>P4 (P4Info)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match field</td>
<td>ETH_DST (enum)</td>
<td>“hdr.ethernet.dst_addr”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Match field name in P4Info table definition</td>
</tr>
<tr>
<td>Packet-in</td>
<td>InboundPacket.java.receivedFrom().port()</td>
<td>“ingress_port”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of metadata field in P4Runtime PacketIn message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defined in P4Info as controller_packet_metadata with name “packet_in”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Copyright © 2019 - Open Networking Foundation
## P4Runtime support in ONOS 2.2 (Sparrow)

<table>
<thead>
<tr>
<th>P4Runtime control entity</th>
<th>ONOS northbound API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table entry</td>
<td>Flow Rule Service, Flow Objective Service</td>
</tr>
<tr>
<td></td>
<td>Intent Service</td>
</tr>
<tr>
<td>Packet-in/out</td>
<td>Packet Service</td>
</tr>
<tr>
<td>Action profile group/members, PRE multicast groups, clone sessions</td>
<td>Group Service</td>
</tr>
<tr>
<td>Meter</td>
<td>Meter Service (indirect meters only)</td>
</tr>
<tr>
<td>Counters</td>
<td>Flow Rule Service (direct counters)</td>
</tr>
<tr>
<td>Pipeline Config</td>
<td>Pipeconf</td>
</tr>
</tbody>
</table>

**Unsupported features - community help needed!**
Parser value sets, registers, digests
ONOS+P4 workflow recap

● Write P4 program and compile it
  ○ Obtain P4Info and target-specific binaries to deploy on device

● Create pipeconf
  ○ Implement pipeline-specific driver behaviours (Java):
    ■ Pipeliner (optional - if you need FlowObjective mapping)
    ■ Pipeline Interpreter (to map ONOS known headers/actions to P4 program ones)
    ■ Other driver behaviors that depend on pipeline

● Use existing pipeline-agnostic apps
  ○ Apps that program the network using FlowObjectives

● Write new pipeline-aware apps
  ○ Apps can use same string names of tables, headers, and actions as in the P4 program
Exercise 3 overview
Pipeline-agnostic apps use FlowObjective API
Pipeline-aware app use FlowRule API

ONOS single instance
BMv2/Stratum Driver

LLDP Provider (link discovery)
Host Provider (host discovery)

mn-stratum Docker container
stratum_bmiv2
IPv6 hosts (Linux net namespace)
Mininet script topo.py

netcfg.json

onos:2.2.0 Docker container

P4Runtime, gNMI

Tutorial app
Tutorial Pipeconf

Environment
Exercise 3 steps

● Modify pipeconf Java implementation
  ○ Map P4Runtime packet-in/out to ONOS-specific representation

● Start ONOS and Mininet

● Load app with pipeconf and netcfg.json

● Verify that link discovery works
  ○ Requires both packet-in and packet-out support

● Verify ping for hosts in the same subnet (via bridging)
  ○ Requires packet-in support for host discovery
Topology discovery via packet-in

[Diagram showing ONOS connected to Switch 1 and Switch 2 through P4Runtime channels]
Topology discovery via packet-out/in

For each switch, send LLDP as packet-out on all ports (discovered via gNMI)

ONOS

Switch 1

Switch 2

 LLDP: origin switch1-port 2
Topology discovery via packet-in

ONOS

Switch 1

SWITCH 2

LLDP: origin
switch1-port 2

Intercept LLDP via packet-in at
the other end.
Topology discovery via packet-in

Learn about link by looking at:
1. LLDP payload (source device/port)
2. Packet-in device and ingress port

LLDP: origin
switch1-port 2

Received on
switch2-port1 !

ONOS

Switch 1

Switch 2

LINK DISCOVERED!
LLDP Provider App

● **Automatically discover network links by injecting LLDP packets in the network**

● **Reacts to device events (e.g., new switch connection)**
  ○ Periodically sends LLDP packets via packet-out for each switch port

● **Install packet-in requests (flow objective) on each device**
  ■ Match: ETH_TYPE = LLDP, BDDP
  ■ Instructions: OUTPUT(CONTROLLER)
Host Provider App

- Learns location of hosts and IP-to-MAC mapping by intercepting ARP, NDP and DHCP packets
- Reacts to device events (e.g., new switch connection)
- Install packet-in requests (flow objective) on each device
  - Match: ARP, NDP, etc
  - Instructions: OUTPUT(CONTROLLER)
- Parses packet-in to discover hosts
Pipeconf implementation

- **ID:** org.onosproject.ngsdn-tutorial
- **Driver behaviors:**
  - Pipeliner
    - Maps FlowObjective from LLDP and HostProvider apps
    - Use P4Runtime/v1model clone sessions to send packets to the CPU (packet-in)
  - Interpreter
    - Maps packet-in/out to/from ONOS internal representation
    - Maps ONOS known headers to P4Info-specific ones:
      - e.g. ETH_TYPE → “hdr.ethernet.type”
- **Target-specific extensions**
  - bmv2.json, p4info.txt
netcfg.json (devices)

{
    "devices": {
        "device:leaf1": {
            "basic": {
                "managementAddress": "grpc://mininet:50001?device_id=1",
                "driver": "stratum-bmv2",
                "pipeconf": "org.onosproject.ngsdn-tutorial"
            },
            "fabricDeviceConfig": {
                "myStationMac": "00:aa:00:00:00:01",
                "isSpine": false
            }
        }
    }
...

Copyright © 2019 - Open Networking Foundation
App architecture

App ID: org.onosproject.ngsdn-tutorial

- **Pipeconf**
- **L2 Bridging Component**
- **IPv6 Routing Component**
- **Other Components**

- Topology events (device, links, hosts)
- Insert flow rules, groups, etc.

- ONOS Core (northbound APIs)

Register at app activation
L2 bridging

Leaf switches should provide bridging for hosts in the same subnet:

- Hosts send NDP Neighbor Solicitation (NS) requests to resolve the MAC address of other hosts
- NDP NS packets are replicated (multicast) to all host-facing ports
- Other host replies with NDP Neighbor Advertisement (NA)
- ONOS learns about hosts by requesting a clone of all NDP NA/NS packets as packet-ins (hostprovider built-in app)
- For each learned host, app installs a flow rule to forward packets for the host MAC destination (bridging table)
How is bridging implemented?

ONOS

leaf1

h1a  h1b  h1c
Host discovery (NDP NS)

1. NDP NS message (e.g. who has h1b MAC addr)
2. Multicast replication + CPU clone session
3. Insert L2 unicast entry for h1a

ONOS

h1a
h1b
h1c

leaf1

Learn h1a
Host discovery (NDP NA)

1. NDP NA message (e.g. h1b is at 00:00:...)
2. Unicast forwarding + clone session
3. Insert L2 unicast entry for h1b

h1a
h1b
h1c
Unicast forwarding

ONOS

leaf1

Unicast forwarding

Unicast ethernet frame

h1a  h1b  h1c
L2BridgingComponent.java

- **Listens to device and topology events**
- **For each device, install:**
  - Flow rule and group to replicate NDP NS to all host-facing ports (l2_ternary_table)
  - Flow rule to intercept NDP NS/NA (ACL table)
  - Flow rule with L2 forwarding rule for each learned host (l2_exact_table)
- **Support bridging only for hosts attached to the same leaf**
- **Looks at topology to derive multicast group with host-facing ports, no need to replicate NDP NS towards spines**
ONOS terminology

● **Criteria**
  ○ Match fields used in a FlowRule

● **Treatment**
  ○ Actions/instructions of a FlowRule

● **Pi* classes**
  ○ Classes used to describe entities similar to P4Runtime ones
  ○ PI stands for protocol-independent
  ○ Examples
    ■ **PiTableId**: name of a table as in the P4 program
    ■ **PiMatchFieldId**: name of a match field in a table
    ■ **PiCriterion**: match fields each one defined by its name and value
    ■ **PiAction**: action defined by its name and list of parameters
Exercise 3: Get Started

Open:
~/ngsdn-tutorial/README.md
~/ngsdn-tutorial/EXERCISE-3.md

Or use GitHub markdown preview:

Solution:
~/ngsdn-tutorial/solution

These slides:

Before starting!
Update tutorial repo
(requires Internet access)
cd ~/ngsdn-tutorial
git pull origin master
make pull-deps

You can work on your own using the instructions.
Ask for instructors help when needed.

P4 language cheat sheet:
http://bit.ly/p4-cs
Packet-in/out metadata

controller_packet_metadata {
    preamble {
        id: 67135753
        name: "packet_out"
        alias: "packet_out"
        annotations: "@controller_header("packet_out")"
    }
    metadata {
        id: 1
        name: "egress_port"
        bitwidth: 9
    }
    metadata {
        id: 2
        name: "_pad"
        bitwidth: 7
    }
}