ONF Connect 2019

Next-Gen SDN Tutorial

September 10, 2019

These slides:

Exercises and VM:
- Get USB keys with VM from instructors
  - Option to use Docker instead of VM

- Copy and import VM into VirtualBox
  - User: `sdn` - Password: `rocks`

- If VirtualBox complains about a missing network adapter, remove that in the VM configuration (adapter 2)

- Update deps inside VM (requires Internet access)
  - `cd ~/ngsdn-tutorial`
  - `git pull origin master`
  - `make pull-deps`

These slides:
Mininet topology for hands-on exercises

Each host is configured with IPv6 gateway address on the same subnet as the host, but ending with ...::ff.
E.g., for h1a, the gateway address is 2001:1:1::ff

Copyright © 2019 - Open Networking Foundation
Instructors

Brian O’Connor
ONF

Carmelo Cascone
ONF
Schedule

8.00am-9.00am registration / breakfast / technical set up for hands-on lab

9.00am-9:20am - NG-SDN overview

9.20am-10.45 - P4 and P4Runtime basics (with hands-on lab)

10.45am-11.15am - break

11.15am-12.30pm - YANG, gNMI and OpenConfig basics (with hands-on lab)

12.30pm-1.30pm- lunch

1.30pm-3.00pm - Using ONOS as the control plane (with hands-on lab)

3.00pm-3.30pm - break

3.30pm-5.00pm - Use cases (with hands-on lab)
NG-SDN Overview
Software Defined Networking (SDN) v1

- **Introduction of Programmatic Network Interfaces**
  - Data Plane programming: OpenFlow
  - Configuration and Management: NETCONF and YANG

- **Promise: Enable separation of data plane and control plane**
  - Unlock control and data plane for independent innovation
SDN v1 Problems

- **Programmatic Network Interfaces are Inconsistent**
  - OpenFlow provided no data plane pipeline specification; every vendor’s pipeline is different
  - Every vendor provides their own models for configuration or management
  - Differences in protocol implementations require custom handling in the control plane

- **Reality: Control planes are written and tested against specific hardware**
  - Some control planes have worked around this by building their own abstractions to handle these differences, but new abstractions are either least common denominator (e.g. SAI) or underspecified (e.g. FlowObjectives)
  - Other control planes have exploited specific APIs are essentially locked in to specific vendors
Network Function Virtualization (NFV) v1

- Migrate specialized networking hardware (e.g. firewall, load balancer) to commodity servers
- Virtualized network functions (VNFs) are packaged and distributed as VMs or containers, which are easier to deploy
NFV v1 Problems

● CPUs are not the right hardware for many network functions
  ○ Latency and jitter are higher than alternatives
  ○ Higher cost per packet and increased power consumption

● NFV data plane topologies are inefficient
  ○ Additional switching hops required to implement sequences of VNFs (service chains), especially when placement algorithms are not optimized
Combining SDN and NFV

- SDN (fabric) and NFV (overlay) are managed separately
  - Increased operational complexity / opex
  - Difficult to optimize across different stacks
  - Lack of visibility for troubleshoot and end-to-end optimization
  - Separate resource pools

Overall, the benefits of SDN/NFV using 1\textsuperscript{st} generation architectures are not without their costs.
Questions

● Can we get the benefits of SDN and NFV without paying these costs?
● Can we incorporate lessons learned from production deployments of SDN v1 and NFV v1?
● Can we take advantage of new networking hardware efficiently and rapidly?
Next-Gen Software Stack Components

- **ONOS**
  - Supports Next-Gen SDN interfaces (P4Runtime, gNMI, gNOI)
  - Cloud-native: microservices, Kubernetes, gRPC, etc.
  - Enable apps to take advantage of the new capabilities

- **Stratum**
  - Thin switch OS
  - Supports Next-Gen SDN interfaces (P4Runtime, gNMI, gNOI)
  - Supports OpenConfig YANG models

- **Forwarding devices**
  - Supports programmable forwarding (P4)
  - Also supported fixed function and partially programmable devices
    - Enables smooth migration and diversity of silicon options
NG-SDN Big Picture

**Revolutionary New Capabilities**

- Top-Down Programmability
- Hardware Independence
- Zero-Touch & Automated Full-Lifecycle Operation
- Verifiability

**Evolutionary Roadmap**

- Next-Gen SDN Interfaces are defined
  - P4, P4Runtime, OpenConfig, gNMI, gNOI
- Stratum now released to Open Source
- ONOS 2.2 supports NG APIs
  - µONOS will provide new configuration subsystem that will be compatible
- Cloud native tool chains established
  - Kubernetes
- Ready to embark on Verification
What is Stratum?

Open source, production targeted, thin switch OS that implements NG-SDN interfaces and models.

TestVectors

TestVectors

Runner

P4 Program

P4 Compiler

P4Runtime

gNMI

gaNOI

Switch (Broker) Interface

Table Mgr

Node

Chassis Mgr

Chip Abstraction Managers

Platform Mgr

Switch SDK

Platform API

Switch Chip(s)

Peripheral(s)

Copyright © 2019 - Open Networking Foundation
Stratum = implementation of 3 APIs

- **Control** – P4Runtime with P4-defined pipelines
  - Manage match-action table entries and other forwarding pipeline state

- **Configuration** – gNMI with OpenConfig models
  - Configure everything else that is not the forwarding pipeline.
    e.g. set port speed, read port counters, manage fans, etc.

- **Operations** – gNOI
  - Execute operational commands on device.
    e.g, reboot, push SSL certificates, etc.

**All of Stratum’s APIs are defined gRPC / Protobuf**
Aside: gRPC (gRPC Remote Procedure Call)

- Use Protocol Buffers to define service API and messages
- Automatically generate client/server stubs in:
  - C / C++
  - C#
  - Dart
  - Go
  - Java
  - Node.js
  - PHP
  - Python
  - Ruby
- Transport over HTTP/2.0 and TLS
  - Efficient single TCP connection implementation that supports bidirectional streaming
An Aside: Protocol Buffers

- Google’s Lingua Franca for serializing data: RPCs and storage
- Binary data representation
- Structures can be extended and maintain backward compatibility
- Code generators for many languages
- Strongly typed
- Not required for gRPC, but very handy

```protobuf
syntax = "proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
}

enum PhoneType {
  MOBILE = 0;
  HOME = 1;
  WORK = 2;
}

message PhoneNumber {
  string number = 1;
  PhoneType type = 2;
}

repeated PhoneNumber phone = 4;
```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {} 
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}

More details here: https://grpc.io/docs/guides/
Achieving ASIC Independence

My Station (Routing Classifier)  
L3 Routing (IPv6 w/ ECMP)  
ACL (Redirect, drop & Pkt in)  
L2 Forwarding

demo.p4

P4 compiler
Allocate resources to realize the pipeline, and generate runtime mapping

fpm backend (Broadcom)  
bmv2 backend  
Tofino backend (Barefoot)

Generate control plane contract

demo.p4info

Control plane

p4runtime.proto

P4Runtime server
Target driver
Switch ASIC

Copyright © 2019 - Open Networking Foundation
Achieving Platform Independence

ONLP Platform impl.
libonlp-<platform>.so

ONL Platform API (ONLPv2)

OpenConfig Mapper
Stratum Config DB

gNMI Server

gnmi.proto

Stratum Platform Manager

openconfig.yang

OpenConfig, OpenNetworkLinux

Control plane

Interfaces, Stratum

INetConfig, ONLPv2

Dell, Delta, QCT
### Stratum Switch Support Today

<table>
<thead>
<tr>
<th>Switch Vendor</th>
<th>Switching ASIC</th>
<th>Tofino</th>
<th>Up to 6.5 Tbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAREFOOT NETWORKS</td>
<td>AG9064v1 64 x 100 Gbps</td>
<td>Wedge100BF-32X 32 x 100 Gbps</td>
<td>Wedge100BF-65X 65 x 100 Gbps</td>
</tr>
<tr>
<td>Tomahawk</td>
<td>Z9100 32 x 100 Gbps</td>
<td>AS7712 32 x 100 Gbps</td>
<td>D7032 32 x 100 Gbps</td>
</tr>
</tbody>
</table>

+ 2 software switches: **bmv2** (functional software switch) & **dummy switch** (used for API testing)

**Near-term future platforms:**
- Additional platforms for existing targets
  - Existing vendors + Asterfusion, ...
- Mellanox SN2700 (Spectrum)
- Datacom platforms (PowerPC-based)
Building and Installing Stratum

**Step 1:**
Build ONL image offline (or download a pre-built one)

**ONL Builder**
- ONL Image
- ONLPv2_AMD64_INSTALLEDinstaller
- Switch SDKs
- ONLP Drivers
- Stratum

**ONL Image**
ONL-ONLPv2_AMD64_INSTALLED.installer

**Container Repository**
stratum_bcm
stratum_bf

**Step 2:**
Install ONL on boot via ONIE

**Stratum can be packaged with the switch OS image or deployed as a container on a bare switch OS**

**ONL Image with Stratum**
ONL-ONLPv2-STRATUM_AMD64_INSTALLED.installer

**Switch Image Repository**

Copyright © 2019 - Open Networking Foundation
Testing Stratum Devices

Test Vectors serve as compliance tests for Stratum-based devices

They can be written manually or generated automatically
- Stratum comes with a Contract Definition language (cdlang) for generating test vectors

Test Vector
Test Case
Stimulus 1  Expectation 1
Stimulus 2  Expectation 2
...  ...

TestVectors Runner is a data-driven framework that enables users to execute TestVectors
- Reference impl. in golang; supports P4RT/gNMI

Test Scenarios in CDLang
CDLang Compiler
Test Vectors
Humans
Automatic Test Vector Generators

TestVectors Runner (low-level, can be plugged in to complex test frameworks)

gRPC
Switch Under Test
Switch Ports
Traffic generators and validators
Tutorial Goals

- Learn how to work with P4 and YANG code
- Understand P4Runtime and gNMI and use CLI utilities to communicate with Stratum devices
- Gain experience running ONOS and Stratum
- Modify a simple Control Plane application that interacts with a P4-defined pipeline
Hands-on overview

Goal: Build IPv6-based leaf-spine fabric with P4, Stratum and ONOS

Getting there step-by-step:

- Exercise 1 - P4 and P4Runtime basics
- Exercise 2 - Yang, OpenConfig, and gNMI basics
- Exercise 3 - Using ONOS as the control plane for Stratum
- Exercise 4 - Modify P4 program and ONOS app to enable IP routing