Realizing Next Generation
SDN/NFV

Brian O’Connor (ONF)
brian@opennetworking.org
ONCon
July 22, 2019
ONF’s History

The ONF has a lot of experience building SDN and NFV solutions

- Trellis (in production with a major operator)
- SEBA (Service Provider Field Trials)
- OpenFlow
- ONOS Open Network Operating System
- CORD

2008 2014 2016 2018 2019
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 alternative targets
  - 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 troubleshooting and end-to-end optimization
  - Separate resource pools

Overall, the benefits of SDN/NFV using 1st generation architectures are not without their costs.
Can we get the benefits of SDN and NFV without paying these costs?
Enabling the Next Generation of SDN

● Development of New Technologies
  ○ Hardware: Programmable ASICs, FPGAs, Smart NICs
  ○ Software: P4

● Adopt “cloud mindset” for deployment and management
  ○ Zero touch operations
  ○ Containerization

● Leverage Open Source Components
  ○ Data planes, control planes, networks functions, and apps
Development of New Technologies
Packet Forwarding Pipelines

Packets

Pipeline of match-action tables

ASIC, FPGA, NPU, or CPU

How is this pipeline specified?
P4 Language

- **Domain-specific language to formally specify a forwarding pipeline**
  - Describe protocol headers, lookup tables, actions, counters, etc.
  - Can describe fast pipelines (e.g. ASIC, FPGA) as well as a slower ones (e.g. SW switch)

- **Good for programmable switches, as well as fixed-function ones**
  - *Programmable*: optimize chip resources to application needs, support new protocols
  - *Fixed-function*: defines “contract” with the control plane for runtime control
PISA: Protocol-Independent Switch Architecture

Abstract machine model of a high-speed **programmable** switch architecture

- Programmer declares which headers should be extracted and in which order
- Programmer defines the tables and the exact processing algorithm
- Programmer declares how the output packet will look on the wire

Programmable Parser → Programmable Match-Action Pipeline → Programmable Deparser

- Programmable Parser
- Memory, ALU
- Match+Action Stage (Unit)
- Programmable Match-Action Pipeline
- Programmable Deparser

*Slide courtesy P4.org*
Compiling a simple logical pipeline on PISA

P4 compiler
Allocate resources to realize the pipeline

Programmable Parser

Programmable Match-Action Pipeline

Programmable Deparser

Matcher+Action Stage (Unit)

IPv6 address table

ACL table
P4 Programs as Fixed-Function Chip Abstraction

- P4 program tailored to apps / role - does not describe the hardware
- Switch maps program to fixed-function ASIC
- Enables portability
Why should we use P4?

1. Explicit packet and pipeline definition enable deployment to heterogeneous targets
   - Same program can be used for fixed-function and programmable targets from different manufacturers

2. Clear language semantics enable automated verification
   - Generate test inputs and results by analyzing the P4 program

For insight into automated verification:

Leveraging P4 for Fixed Function Switches
Speakers: Konstantin Weitz & Stefan Heule (Google)
Links: Slides, Video, or scan the QR codes
Control Interface: P4Runtime

Entries for
Tables, Action Profiles, Meters, Counters, Packet Replication, Parser Values, Registers, Digests, Externs

Slide adapted from P4.org
API Consistency for Network Functions

- Provide a consistent interface for network function programming that is independent of hardware or location
- Implement network functions on hardware that meets performance needs
P4Runtime overview

- **API for runtime control of P4-defined switches**
  - Generic RPCs to manage P4-defined table entries and other forwarding state

- **Community-developed (p4.org API WG)**
  - Initial contribution by Google and Barefoot
  - RC of version 1.0 available: https://p4.org/p4-spec/

- **gRPC/protobuf-based API definition**

- **Enables field-reconfigurability**
  - Ability to push new P4 program, i.e. re-configure the switch pipeline, without recompiling the switch software stack
  - E.g. to add new match-action tables, support parsing of new header formats
P4Runtime TableEntry Example

### Logical view of table entry

```basic
action ipv4_forward(bit<48> dstAddr, bit<9> port) {
    /* Action implementation */
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

- `hdr.ipv4.dstAddr=10.0.1.1/32` -> `ipv4_forward(00:00:00:00:00:10, 7)`

### Protobuf TableEntry message

```protobuf
table_entry {
    table_id: 33581985
    match {
        field_id: 1
        lpm {
            value: '\n\000\001\001'
            prefix_len: 32
        }
    }
    action {
        action_id: 16786453
        params {
            param_id: 1
            value: '\000\000\000\000\000\000\000\n'
        }
        params {
            param_id: 2
            value: '\000\007'
        }
    }
}
```

Control plane generates

- Protobuf TableEntry message
#2
Next Step: Adopting a Cloud Mindset
Zero Touch Operations

- Humans don’t log into individual boxes to configure them
- Configuration is generated automatically and sent to devices
  - Changes are defined by high-level, network-centric intent
- Management plane listens to telemetry events and applications drive network state towards policy objections
  - Rollback happens automatically in network invariants are violated
“70% of network failures happen during management operations, due to the high level of complexity of such operations across a wide variety of network types, devices, and services” - Google

Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure
Authors: Ramesh Govindan, Ina Minei, et al. (Google)

Links: Paper, Video, or scan the QR codes
Zero Touch Operations

Availability
Reliability
3, 4 or 5 “9’s”

Velocity
Daily updates
Zero Touch Operations: Control and Config/Management

To achieve a zero touch network a seamless interplay of control and config/management needs to happen. High level network centric policies always need a combination of both elements to be achieved with no impact.

Zero Touch Operations
Simplify and Centralize Configuration

- Leverage vendor-neutral models as much as practical
- Centralize configuration and management to reduce deployment complexity
- This applies to both data plane and control plane components
OAM Interfaces: gNMI and gNOI

- gNMI for:
  - Configuration
  - Monitoring
  - Telemetry
- gNOI for Operations

Switch Chip Configuration
- QoS Queues and Scheduling
- Serialization / Deserialization
- Port Channelization

Management Network

Fan Speed
- Power supplies

Monitor Sensors
- e.g. temperature

Software Deployment and Upgrade

Port State and Mapping
- LED Control

... and the list goes on.
Enhanced Configuration

- Configuration and Management
- Declarative configuration
- Streaming telemetry
- Model-driven management and operations
  - gNMI - network management interface
  - gNOI - network operations interface
- Vendor-neutral data models

Slide adapted from Google
OpenConfig Model - An Example

module: openconfig-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw config
        |  +--rw name?            string
        |  +--rw type             identityref
        |  +--rw mtu?             uint16
        |  +--rw loopback-mode?   boolean
        |  +--rw description?     string
        |  +--rw enabled?         boolean
      +--ro state
        |  +--ro name?            string
        |  +--ro type             identityref
        |  +--ro mtu?             uint16
        |  +--ro loopback-mode?   boolean
        |  +--ro description?     string
        |  +--ro enabled?         boolean
        |  +--ro ifindex?         uint32
        |  +--ro admin-status     enumeration
        |  +--ro oper-status      enumeration
        |  +--ro last-change?     oc-types:timeticks64
        |  +--ro logical?         boolean
        |  +--ro counters
          |    +--ro in-octets?     oc-yang:counter64
          |    +--ro in-pkts?       oc-yang:counter64
          ...
Closed Loop Control Relies on Telemetry

Device Telemetry Collector

Control Application

Control and Management Plane

INT Collector

Switch

P4Runtime

OpenConfig & gNMI

Report

INT

pkt

pkt

pkt
#3
Leverage Open Source Components
Providing an Implementation

Open Interfaces and Models are necessary, **but not sufficient**, for multi-vender interoperability.

Interfaces are **defined by running code**, so providing an open source implementation helps solidify the interfaces and models. This is not a standards exercise.

If the open source is a fully production ready distribution (ready to run and deploy these interfaces), we can **avoid bugs in different vendor implementations** and improve time to market.
Stratum: Next Generation Data Plane

- Vendor Neutral
- Extensible
Stratum High-level Architectural Components

- **Common (HW agnostic)**
- **Chip specific**
- **Platform specific**
- **Chip and Platform specific**

**Kernel**
- User
- Hardware

**Remote or Local Controller(s)**
- P4 Runtime
- gNMI
- gNOI

**Switch Broker Interface**
- Table Manager
- Node/Chip Manager
- Chassis Manager
- Platform Manager

**Chip Abstraction Managers**
- E.g. ACL, L2, L3, Packet I/O, Tunnel

**Platform Manager**
- Platform API
- Platform Drivers

**Switch Chip(s)**
- Switch Chip Drivers

**Peripheral(s)**
- Switch SDK

PI and fpm-based implementations

ONLP
Stratum Implementation Details

- Implements **P4Runtime**, **gNMI**, and **gNOI** services
- Controlled locally or remotely using **gRPC**
- Written in **C++11**
- Runs as a **Linux** process in user space
- Can be distributed with **ONL**
- Built using **Bazel**

Available to the public end of August 2019!
Comprehensive Test Framework

Is an open-source implementation enough for interop? How to we prevent implementation discrepancies?

There will be other implementations, and they need to be qualified. We also need to make sure that vendor-specific pieces are implemented as expected.

Solution: Provide a vendor-agnostic, “black box” test framework for any target that complies with Stratum open APIs (P4Runtime, gNMI, gNOI) along with a repository of tests.
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.
Black Box Qualification

Vendor Test (Vendor)

Functional Test on standalone testbeds

Integration Test in fabric testbeds with SDN controller

Release (or further SDN controller qual)

TV creation or mod

Misc. input (trace, etc)

Vendor space

Operator space

Vendor

Product Requirements

Body of TVs

P4 specs, YANG models, open APIs

Open Source

Slide courtesy of Google
Project Genesis
μONOS

rationale & tenets for next-gen SDN controller
NG SDN Controller Architecture

Apps
- Topology
  - NB API
  - SB API
- Configuration
  - gNMI/gNOI
- Control
  - P4Runtime
- RAN
  - NB API
  - SB API

Protocols & Drivers
- gNMI/gNOI
- P4Runtime
NG SDN Controller Architecture

topology
NetModel
NetDisco
certs
NetCerts
k/v
k/v
k/v
config
gNMI/gNOI
P4Runtime
gNMI/gNOI
P4Runtime
control
NetModel
NetDisco
certs
NetCerts
gNMI/gNOI
P4Runtime
ran
gRNI
SD-RAN
gRNI
gRNI
app
gNMI/gNOI/P4Runtime
adapter
various protocols

discovery
network
helm / k8s

topology
NetModel
NetDisco
certs
NetCerts
k/v
k/v
k/v
config
gNMI/gNOI
P4Runtime
gNMI/gNOI
P4Runtime
control
NetModel
NetDisco
certs
NetCerts
gNMI/gNOI
P4Runtime
ran
gRNI
SD-RAN
gRNI
gRNI
app
gNMI/gNOI/P4Runtime
adapter
various protocols

discovery
network
helm / k8s
Configuration Subsystem

- Work hosted under GitHub in the open
  - [http://github.com/onosproject/onos-config](http://github.com/onosproject/onos-config)

- Primarily staffed by ONF at this point
  - external contributions are wanted and welcome

- Bi-weekly updates and demos given at ONOS TST
  - deployment via Helm/Kubernetes
  - multi-device transactions and rollback
  - integrated validation of data via ygot
  - Atomix 4.x with support for gRPC and Go primitives client libraries
  - currently prepping start work on distributed stores

- Planning ONF Connect demos
Topology Subsystem

- Exploring use of Google’s Unified Network Model
  - initiating discussions with Google about using UNM or a derivative
  - UNM was part of Jeff Mogul’s presentation at Stanford last year

- Goal is to use UNM-like model as a canonical representation
  - allows to capture design intent and supports schema evolution
  - ability to project to alternate representations, eg.
    - RFC 8345 IETF Network Topology model to exchange topology data and changes to topology state
    - custom graph structures and gRPC streaming for run-time performance
Control & RAN Subsystems

● **SB API for the subsystem will be P4Runtime**
  ○ well-defined, low-profile interfaces with support for transactions
  ○ allows direct use with Stratum-compliant switches
  ○ adapters can be created for devices that do not support P4Runtime

● **NB API will be P4Runtime and admin APIs**
  ○ requires network-wide table mapping, e.g. network-sized chassis
  ○ design work for amin and diagnostic APIs will start shortly

● **Provide abstractions and controls relevant to the RAN domain**
  ○ near real-time requirements, e.g. latency sensitive, predictable
Looking Ahead

- **ONOS 2.x already supports P4Runtime and gNMI**
- With ONOS 2.x being a stable platform for some time to come, now is the time to consider next generation architecture
- With Stratum starting to materialize as UPAN data plane, now is the time to consider UPAN control plane
- Goal is to establish the next generation SDN controller architecture
  - kicked off collaboration at start of 2019
  - completely in the open and with the help of the community at large
- Project is named µONOS and will become ONOS 3.0 when ready
- Continue to curate ONOS 1.x & 2.x maintenance and releases
  - core team to do LTS bug fixes, code reviews and release engineering
  - community to continue new feature and applications development
Using Docker to Deploy Applications

Control Plane / SDN App
- Shared libraries / runtime

Host OS: Linux

Stratum Agent / Network Fn.
- Shared libraries / runtime

Host OS: Linux

Docker

whitebox server

whitebox switch
Using Kubernetes to Deploy to Common Infra

- **Container Repository**
  - Docker
  - Helm
  - Kubernetes

- **CI Testing**

**Goal is continuous (daily!) deployment**
Deploy components on common infrastructure

- Deploy control plane and data plane functions on a converged network infrastructure
- Place functions in appropriate locations using an intelligent scheduler
- Deploy functions on hardware that meets performance needs
Next Generation SDN picture

Control and Management Plane

- SDN Control Services
- Configuration Services
- Monitoring & Telemetry Services
- Admin & Orchestration Services

P4Runtime

- spine.p4
- leaf.p4

OpenConfig

- gNMI

Stratum

ONOS

Inventory

Global Orcherstrator

OSS / BSS
Enabling the Next Generation of SDN

● Development of New Technologies
  ○ Hardware: Programmable ASICs, FPGAs, Smart NICs
  ○ Software: P4

● Adopt “cloud mindset” for deployment and management
  ○ Zero touch operations
  ○ Containerization

● Leverage Open Source Components
  ○ Data planes, control planes, networks functions, and apps

If this sounds interesting, please get involved!
For questions, email brian@opennetworking.org
Backup Slides
**Architecture**

**Mixed P4/OpenFlow multi-vendor white-box switches**
- Broadcom, Barefoot, Edge-Core, Inventec, Delta

**ONOS**
- Trellis (Segment routing, multicast, vRouter, etc)
- In-band Network Telemetry (INT)
- VNF Offloading Control (S/PGW)

**P4Runtime**
- OpenFlow

**fabric.p4**
- L2/L3/MPLS
- INT
- GTP termination

**Upstream BGP routers**
- Internet

**Telemetry collector**
- Barefoot DeepInsight

**P4 SmartNIC**
- Netcope

**Multicast video source**
- INT Collector (emulated)
Life of a Whitebox Switch: Day 0 to Day N

1. Design
2. Installation & Bootstrap
3. Switch Configuration
4. Start the Data Plane
5. Monitoring & Telemetry
6. Reboot
7. Upgrade

Network Design → Installation and Bootstrap → Switch Configuration → Start the Data Plane → Maintenance or Upgrade

Monitoring and Troubleshooting
Use-case 1
Chaining and Scaling Edge Gateway

- Flexible traffic chaining with BGP FlowSpec
- Auto chaining/scaling
- In-band telemetry between VNFs
Use-case 2
DDoS Detection and Steering Function

- Collect flow-statistics from stratum switches
- Steering traffic to mitigation function when collector detects flow burst
Use-case 3

Edge Router on Fixed Networks

- There are thousands of NTT buildings that has the edge-router(s)
- Can edge-routers be replaced by Stratum?
Use-case 3 Edge Router on Fixed Networks

Today’s Service Edge Router

- Edge-router contains service functions (BRAS/BGF/Video-Multicast/VPN-GW…) and Hierarchal QoS function

Service Functions

BRAS:
- PPPoE termination
- AAA(Radius)

BGF:
- NAPT
- Flow-based shaping
- Diffserv

MC(Video Multicast):
- PIM/MLD
- IP Multicast

VPN-GW
- Tunnel termination
- Dynamic routing
Transforming Tencent’s Network: One Datacenter at a Time

- Data center fabric as disaggregated modular switch

- Centralized control does not mean the entire network must have one controller.

- Rather we opt for a network of controllers, enabled by ONF CORD, Trellis and Stratum.
  - Freedom to use different protocols or RPC at outside controllers.
  - Facilitates integration with legacy networks.