3. Brief project overview

Table of Contents

3.1 FD.IO

Editor: John DeNisco

FD.io (Fast data – Input/Output) is a collection of several projects that support flexible, programmable packet processing services on a generic hardware platform. FD.io offers a landing site with multiple projects fostering innovations in software-based packet processing towards the creation of high-throughput, low-latency and resource-efficient IO services suitable to many architectures (x86, ARM, and PowerPC) and deployment environments (bare metal, VM, container).

The core component is the Vector Packet Processing (VPP) library. VPP using “Vector Packet Processing” technology. The VPP library is highly modular, allowing for new graph nodes to be easily “plugged in” without changes to the underlying code base. This gives developers the potential to easily build any number of packet processing solutions.

FD.io Vector Packet Processor (VPP)

FD.io's Vector Packet Processor (VPP) is a fast, scalable layer 2-4 multi-platform network stack. It runs in Linux user space on multiple architectures including x86, ARM, and Power architectures.

Vector vs Scalar Processing

FD.io VPP is developed using vector packet processing, as opposed to scalar packet processing. Vector packet processing is a common approach among high performance packet

processing applications. The scalar based approach tends to be favored by network stacks that don't necessarily have strict performance requirements.

Scalar Packet Processing

A scalar packet processing network stack typically processes one packet at a time: an interrupt handling function takes a single packet from a Network

Interface, and processes it through a set of functions: fooA calls fooB calls fooC and so on.

```
+---> fooA(packet1) +---> fooB(packet1) +---> fooC(packet1)
+---> fooA(packet2) +---> fooB(packet2) +---> fooC(packet2)
+---> fooA(packet3) +---> fooB(packet3) +---> fooC(packet3)
```

Scalar packet processing is simple, but inefficient in these ways:

- When the code path length exceeds the size of the Microprocessor's instruction cache (I-cache) thrashing occurs, The Microprocessor is continually loading new instructions. In this model, each packet incurs an identical set of I-cache misses.
- The associated deep call stack will also add load-store-unit pressure as stack-locals fall out of the Microprocessor's Layer 1 Data Cache (D-cache).

Vector Packet Processing

In contrast, a vector packet processing network stack processes multiple packets at a time, called 'vectors of packets' or simply a 'vector'. An interrupt handling function takes the vector of packets from a Network Interface, and processes the vector through a set of functions: fooA calls fooB calls fooC and so on.

```
+---> fooA([packet1, +---> fooB([packet1, +---> fooC([packet1, +--->
            packet2,             packet2,             packet2,
            ...                  ...                  ...
            packet256])          packet256])          packet256])
```

This approach fixes:

- The I-cache thrashing problem described above, by amortizing the cost of I-cache loads across multiple packets.

- The inefficiencies associated with the deep call stack by receiving vectors of up to 256 packets at a time from the Network Interface, and processes them using a directed graph of node. The graph scheduler invokes one node dispatch function at a time, restricting stack depth to a few stack frames.

The further optimizations that this approaches enables are pipelining and prefetching to minimize read latency on table data and parallelize packet loads needed to process packets.

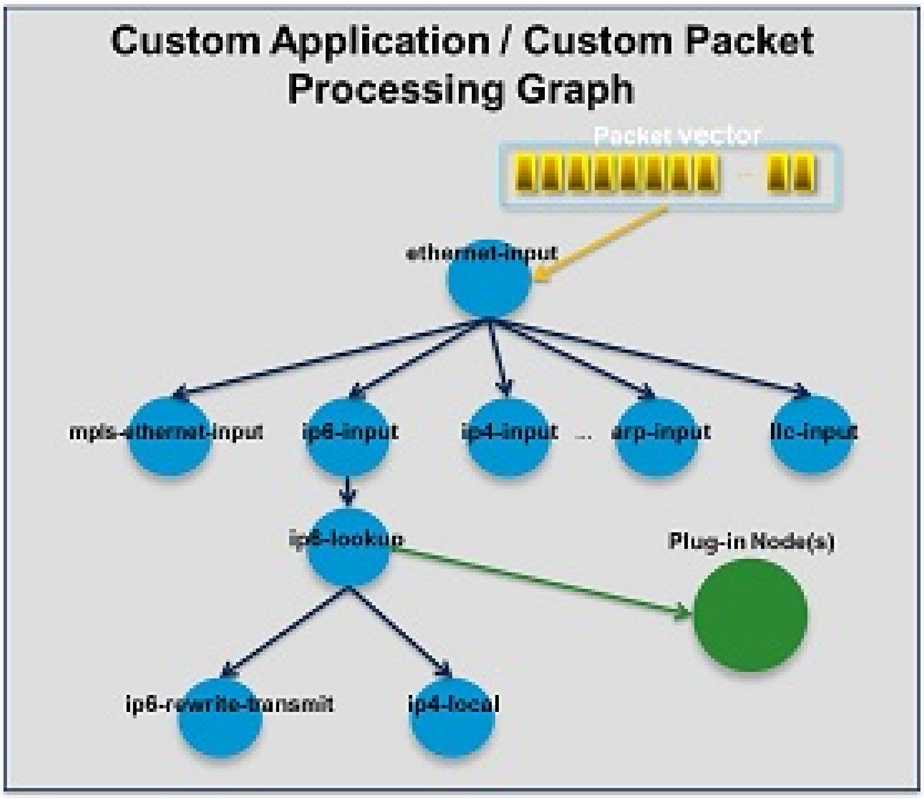
The Packet Processing Graph

At the core of the FD.io VPP design is the **Packet Processing Graph.** The FD.io VPP packet processing pipeline is decomposed into a ‘Packet Processing Graph’. This modular approach means that anyone can ‘plugin’ new graph nodes. This makes VPP easily extensible and means that plugins can be customized for specific purposes.

The Packet Processing Graph creates software:

- That is pluggable, easy to understand and extend
- Consists of a mature graph node architecture
- Allows for control to reorganize the pipeline
- Fast, plugins are equal citizens

A Packet Processing Graph:



At runtime, the FD.io VPP platform assembles a vector of packets from RX rings, typically up to 256 packets in a single vector. The packet processing graph is then applied, node by node (including plugins) to the entire packet vector. The received packets typically traverse the packet processing graph nodes in the vector, when the network processing represented by each graph node is applied to each packet in turn. Graph nodes are small and modular, and loosely coupled. This makes it easy to introduce new graph nodes and rewire existing graph nodes.

Plugins

Plugins are shared libraries and are loaded at runtime by FD.io VPP. FD.io VPP finds plugins by searching the plugin path for libraries, and then dynamically loads each one in turn on startup. A plugin can introduce new graph nodes or rearrange the packet processing graph. You can build a plugin completely independently of the FD.io VPP source tree, which means you can treat it as an independent component.

Features

Most FD.io VPP features are written as plugins. The features include everything from layer 2 switching to a TCP/IP host stack. For a complete list of features please visit FD.io VPP features.

Drivers

FD.io VPP supports and has tested most DPDK drivers (some have not been completely tested). FD.io VPP also has some native drivers most notablly VMXNET3 (ESXI), AVF (Intel), vhostuser (QEMU), virtue, tapv2, host-interface and Mellanox.

Use Cases

Routers, Universal CPE etc.

FD.io VPP supports entry hardware options from number of hardware vendors for building Customer Premise Equipment devices. FD.io VPP based commercial options are available from vendors such as Netgate with TNSR, Cisco with the ASR 9000, Carrier Grade Services Engine and many more.

These implementations are accelerated with DPDK Cryptodev for whole platform crypto.

Broadband Network Gateway

FD.io VPP has a growing list of network traffic management and security features to support gateway uses cases such as Broadband Network Gateway.

Load Balancer

FD.io VPP has a rich set of plugin's to enhance its capabilities. Cloud load-balancing is just one of number of feature enhancing plugins available to the end user. For example: Google Maglev Implementation, Consistent Hashing, Stateful and stateless load balancing, Kube-proxy integration.

Intrusion Prevention

Fd.io VPP has four different Access Control List technologies; ranging from the simple IP-address whitelisting (called COP) to the sophisticated FD.io VPP Classifiers.

More Information

For more information on FD.io VPP please visit FD.io VPP.

Other FD.io projects

There are several other notable FD.io projects. Some of them are listed here.

Continuous System Integration and Testing (CSIT)

The Continuous System Integration and Testing (**CSIT**) project provides functional and performance testing for FD.io VPP. This testing is focused on functional and performance regressions. For more information on the CSIT project please visit the CSIT project pages. For the latest CSIT results please visit the CSIT report.

Hybrid Information-Centric Networking (hiCN)

Hybrid Information-Centric Networking (hICN) is a network architecture that makes use of IPv6 or IPv4 to realize location-independent communications. A scalable stack is available based on VPP and a client stack is provided to support any mobile and desktop operating system. For more information on the hiCN project please visit the hiCN documents.

Universal Deep Packet Inspection (UDPI)

The Universal Deep Packet Inspection (UDPI) project is a reference framework to build a high performance solution for Deep Packet Inspection, integrated with the general purpose FD.io VPP stack. It leverages industry regex matching library to provide a rich set of features, which can be used in IPS/IDS, Web Firewall and similar applications. For more information on UDPI please visit UDPI wiki.

Dual Mode, Multi-protocol, Multi-instance (DMM)

Dual Mode, Multi-protocol, Multi-instance (DMM) is to implement a transport agnostic framework for network applications that can

- Work with both user space and kernel space network stacks
- Use different network protocol stacks based on their functional and performance requirements (QOS)
- Work with multiple instances of a transport protocol stack

Use and engage or adopt a new protocol stack dynamically as applicable. For more information on DMM please visit the DMM wiki page.

Sweetcomb

Sweetcomb is a management agent written in C that runs on the same host as a VPP instance, and exposes yang models via NETCONF, RESTCONF and gNMI to allow the management of VPP instances. Sweetcomb works as a plugin (ELF shared library) for sysrepo datastore. For more information on Sweetcomb please the Sweetcomb wiki page.

More Information

For more information in the FD.io Project please visit FD.io or FD.io Documentation.

3.2 ONAP

Editor: Chaker Al-Hakim

Introduction to ONAP

The Open Network Automation Platform (ONAP) project addresses the rising need for a **common automation platform for telecommunication, cable, and cloud service providers**—and their solution providers— that enables the **automation of different lifecycle processes**, to deliver differentiated network services on demand, profitably and competitively, while leveraging existing investments.

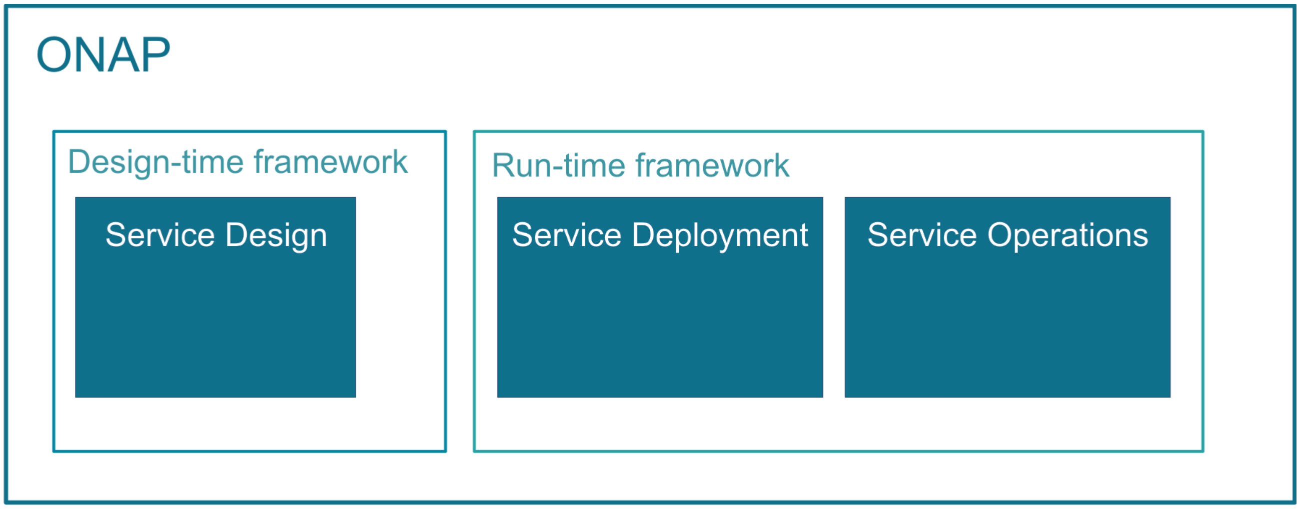
Prior to ONAP, telecommunication network operators had to keep up with the scale and cost of manual changes required to implement new service offerings, from installing new data center equipment to, in some cases, upgrading customer equipment on-premises. Many operators are seeking to exploit Software Defined Network (SDN) and Network Function Virtualization (NFV) to improve service velocity, simplify equipment interoperability and integration, and reduce overall CapEx and OpEx costs. In addition, the current, highly fragmented management landscape makes it difficult to monitor and guarantee service-level agreements (SLAs).

ONAP is addressing these challenges by developing global and massive scale (multi-site and multi-Virtual Infrastructure Manager (VIM)) automation capabilities for both physical and virtual network elements. It facilitates service agility by supporting data models for rapid service and resource deployment, by providing a common set of Northbound REST APIs that are open and interoperable, and by supporting model driven interfaces to the networks. ONAP's modular and layered nature improves interoperability and simplifies integration, allowing it to support multiple VNF environments by integrating with multiple VIMs, virtualized network function managers (VNFMs), SDN Controllers, and even legacy equipment. ONAP's consolidated VNF requirements enable commercial development of ONAP-compliant VNFs. This approach allows network and cloud operators to optimize their physical and virtual infrastructure for cost and performance; at the same time, ONAP's use of standard models reduces integration and deployment costs of heterogeneous equipment, while minimizing management fragmentation.

Scope of ONAP

ONAP enables end user organizations and their network or cloud providers to collaboratively instantiate network elements and services in a dynamic, closed control loop process, with real-time response to actionable events.

ONAP's major activities, that is designing, deploying and operating services, are provided based on ONAP's two major frameworks, namely on Design-time framework and Run-time framework:



In order to design, deploy and operate services and assure these dynamic services, ONAP activities are built up as follows:

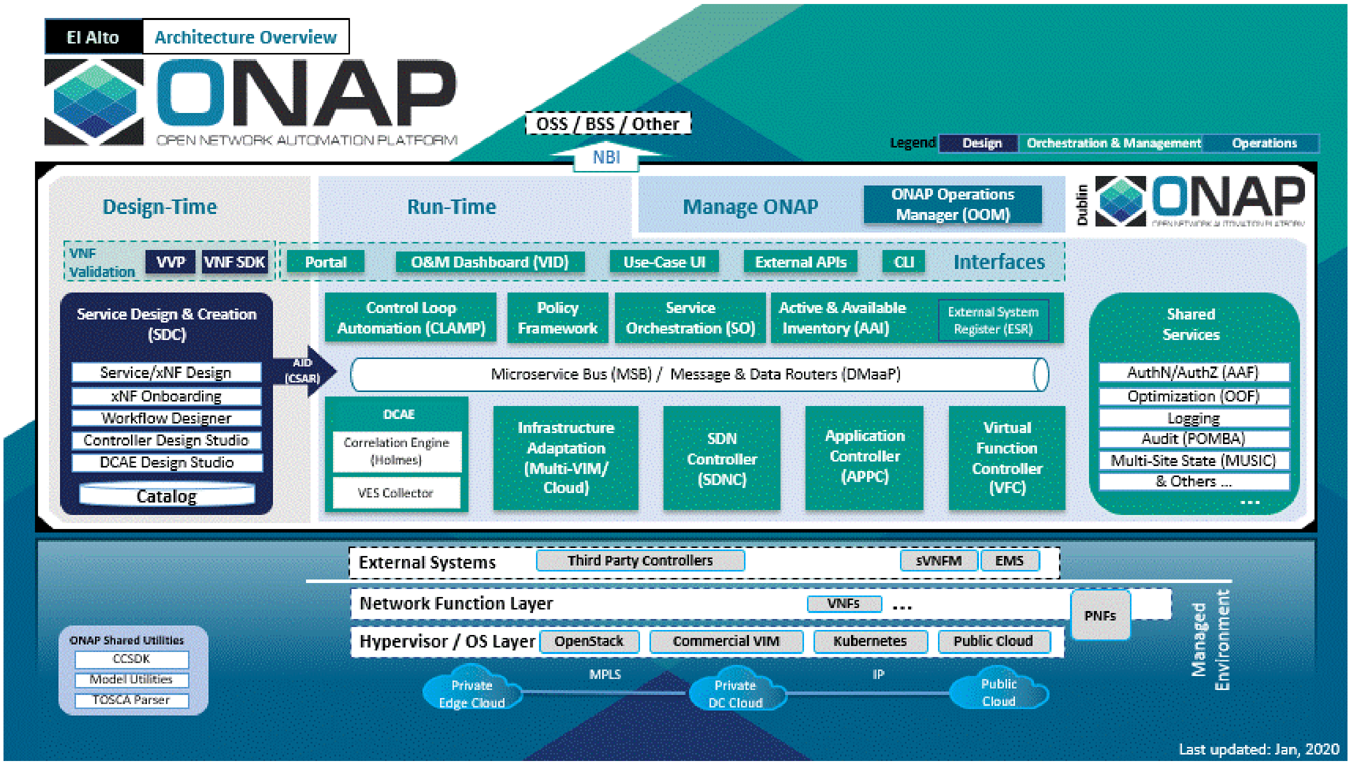
- **Service design** – Service design is built on a robust design framework that allows specification of the service in all aspects – modeling the resources and relationships that make up the service, specifying the policy rules that guide the service behavior, specifying the applications, analytic and closed control loop events needed for the elastic management of the service.
- **Service deployment** – Service deployment is built on an orchestration and control framework that is policy-driven (Service Orchestrator and Controllers) to provide automated instantiation of the service when needed and managing service demands in an elastic manner.
- **Service operations** – Service operations are built on an analytic framework that closely monitors the service behavior during the service lifecycle based on the specified design, analytics and policies to enable response as required from the control framework, to deal with situations ranging from those that require healing to those that require scaling of the resources to elastically adjust to demand variations.

ONAP enables product- or service-independent capabilities for design, deployment and operation, in accordance with the following foundational principles:

1. Ability to dynamically introduce full service lifecycle orchestration (design, provisioning and operation) and service API for new services and technologies without the need for new platform software releases or without affecting operations for the existing services
2. Carrier-grade scalability including horizontal scaling (linear scale-out) and distribution to support large number of services and large networks

3. Metadata-driven and policy-driven architecture to ensure flexible and automated ways in which capabilities are used and delivered
4. The architecture shall enable sourcing best-in-class components
5. Common capabilities are ‘developed’ once and ‘used’ many times
6. Core capabilities shall support many diverse services and infrastructures
7. The architecture shall support elastic scaling as needs grow or shrink

ONAP Functional Architecture



The above diagram shows the main ONAP activities in a chronological order.

Service design

ONAP supports Service Design operations, using the TOSCA approach. These service design activities are built up of the following subtasks:

1. Planning VNF onboarding – checking which VNFs will be necessary for the required environment and features
2. Creating resources, composing services
3. Distributing services - Distributing services constitutes of 2 subtasks:
    a. TOSCA C-SAR package is stored in the Catalog * new service notification is published

Service orchestration and deployment

1. Defining which VNFs are necessary for the service
2. Defining orchestration steps
3. Selecting valid cloud region
4. Service orchestration calling cloud APIs to deploy VNFs
    a. The onboarding and instantiation of VNFs in ONAP is represented via the example of onboarding and instantiating a virtual network function (VNF), the virtual Firewall (vFirewall). Following the guidelines and steps of this example, any other VNF can be similarly onboarded and instantiated to ONAP. See virtual Firewall Onboarding and Instantiating examples.
5. Controllers applying configuration on VNFs

Service operations

1. Closed Loop design and deployment
2. Collecting and evaluating event data

Benefits of ONAP

Open Network Automation Platform provides the following benefits:

- common automation platform, which enables common management of services and connectivity, while the applications run separately
- a unified operating framework for vendor-agnostic, policy-driven service design, implementation, analytics and lifecycle management for large-scale workloads and services
- orchestration for both virtual and physical network functions
- ONAP offers Service or VNF Configuration capability, in contrast to other open-source orchestration platforms
- the model-driven approach enables ONAP to support services, that are using different VNFs, as a common service block
- service modelling enables operators to use the same deployment and management mechanisms, beside also using the same platform

ONAP Releases

ONAP is enhanced with numerous features from release to release. Each release is named after a city. A list of past and current releases may be found here:

https://wiki.onap.org/display/DW/Long+Term+Roadmap

3.3 OpenDaylight

Editor: Abhijit Kumbhare

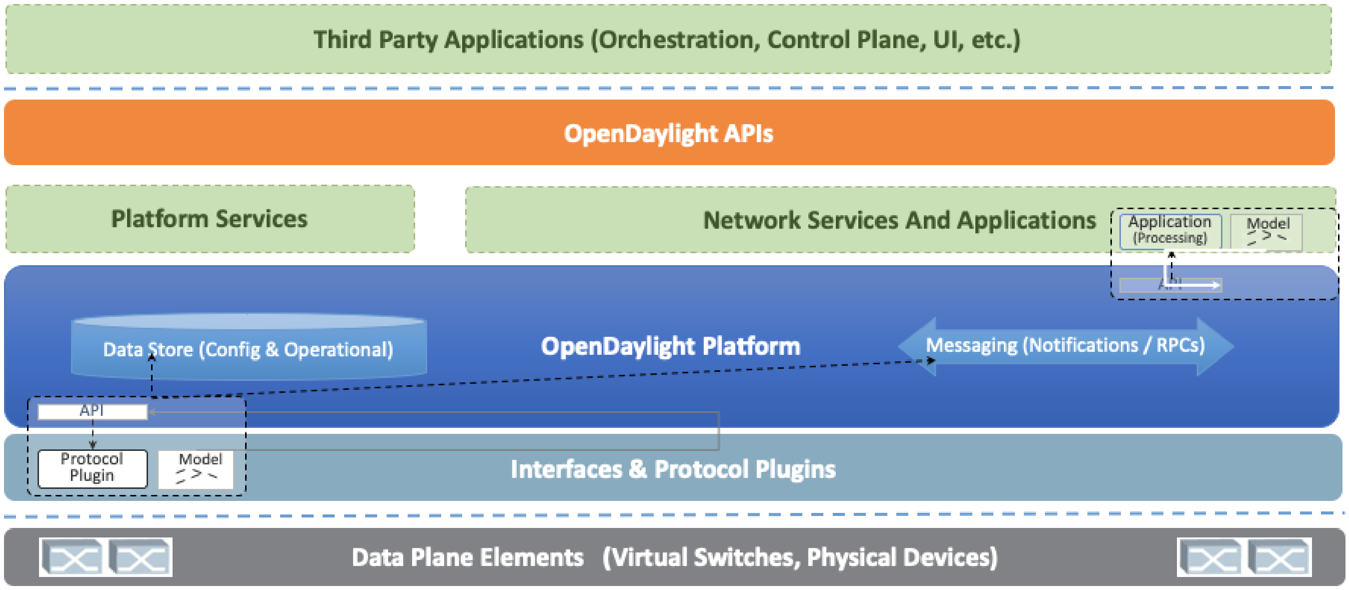
Introduction

OpenDaylight (ODL) is a modular open platform for customizing and automating networks of any size and scale. The OpenDaylight Project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments.

OpenDaylight Architecture

Model-Driven

The core of the OpenDaylight platform is the Model-Driven Service Abstraction Layer (MD-SAL). In OpenDaylight, underlying network devices and network applications are all represented as objects, or models, whose interactions are processed within the SAL.


Third Party Applications (Orchestration, Control Plane, UI, etc.)
OpenDaylight APIs
Platform Services
Network Services And Applications
Application (Processing)
Model
API
Data Store (Config & Operational)
OpenDaylight Platform
Messaging (Notifications / RPCs)
API
Protocol Plugin
Model
Interfaces & Protocol Plugins
Data Plane Elements (Virtual Switches, Physical Devices)

The SAL is a data exchange and adaptation mechanism between YANG models representing network devices and applications. The YANG models provide generalized descriptions of a device or application's capabilities without requiring either to know the specific implementation details of the other. Within the SAL, models are simply defined by their respective roles in a given interaction. A "producer" model implements an API and provides the API's data; a "consumer" model uses the API and consumes the API's data. While 'northbound' and 'southbound' provide a network engineer's view of the SAL, 'consumer' and 'producer' are more accurate descriptions of interactions within the SAL. For example, protocol plugin and its associated model can either be a producer of information about the underlying network, or a consumer of application instructions it receives via the SAL.

The SAL matches producers and consumers from its data stores and exchanges information. A consumer can find a provider that it's interested in. A producer can generate notifications; a consumer can receive notifications and issue RPCs to get data from providers. A producer can insert data into SAL' s storage; a consumer can read data from SAL's storage. A producer implements an API and provides the API's data; a consumer uses the API and consumes the API's data.

Modular and Multiprotocol

The ODL platform is designed to allow downstream users and solution providers maximum flexibility in building a controller to fit their needs. The modular design of the ODL platform allows anyone in the ODL ecosystem to leverage services created by others; to write and incorporate their own; and to share their work with others. ODL includes support for the broadest set of protocols in any SDN platform – OpenFlow, OVSDB, NETCONF, BGP and many more – that improve programmability of modern networks and solve a range of user needs.

Southbound protocols and control plane services, anchored by the MD-SAL, can be individually selected or written, and packaged together according to the requirements of a given use case. A controller package is built around four key components (odlparent, controller, MD-SAL and yangtools). To this, the solution developer adds a relevant group of southbound protocols plugins, most or all of the standard control plane functions, and some select number of embedded and external controller applications, managed by policy.

Each of these components is isolated as a Karaf feature, to ensure that new work doesn't interfere with mature, tested code. OpenDaylight uses OSGi and Maven to build a package that manages these Karaf features and their interactions.

This modular framework allows developers and users to:

- Only install the protocols and services they need
- Combine multiple services and protocols to solve more complex problems as needs arise

- Incrementally and collaboratively evolve the capabilities of the open source platform
- Quickly develop custom, value-added features for highly specialized use cases, leveraging a common platform shared across the industry.

Bottomline about the Architecture

The modularity and flexibility of OpenDaylight allows end users to select whichever features matter to them and to create controllers that meets their individual needs.

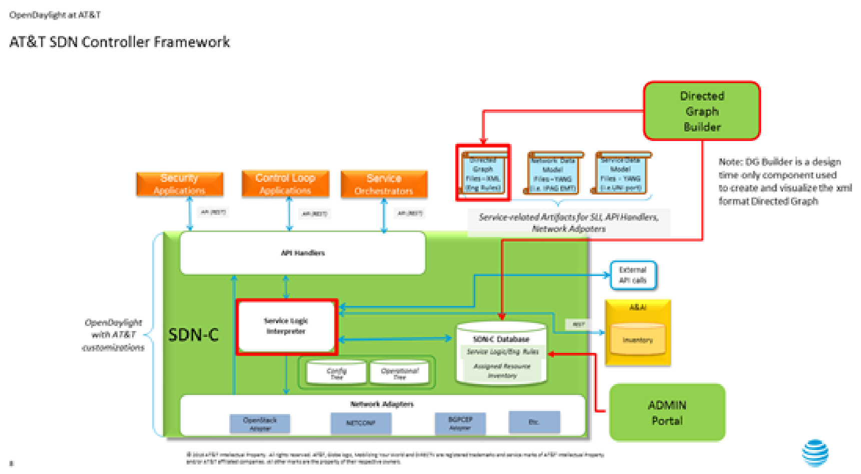
Use Cases

As we saw, the OpenDaylight platform (ODL) provides a flexible common platform underpinning a wide variety of applications and Use Cases. Some of the most common use cases are mentioned here.

ONAP

Leveraging the common code base provided by Common Controller Software Development Kit (CCSDK), ONAP ~~has~~provides two application level configuration and lifecycle management controller modules called ONAP SDN-C and ONAP App-C . These controllers manage the state of a single Resource (Network or Application). Both provide similar services (application level configuration using NetConf, Chef, Ansible, RestConf, etc.) and life cycle management functions (e.g. stop, resume, health check, etc.). The ONAP SDN-C has been used mainly for Layer1-3 network elements and the ONAP App-C is being used for Layer4-7 network functions. The ONAP SDN-C and the ONAP App-C components are extended from OpenDaylight controller framework.

The ONAP SDN-C leverages the model driven architecture in OpenDaylight. As illustrated , the ONAP SDN-C leverages the OpenDaylight framework composed of API handlers, operational and configuration trees, and network adapters for network device configurations. Within this framework, the ONAP Service Logic Interpreter (SLI) newly introduced provides an extensible scripting language to express service logic through the Directed Graph builder based on Node-Red . The service logic is written how network service parameters (e.g. L3VPN) given from the northbound API are mapped onto network device configuration parameters consumed by external SDN controllers attaching to the ONAP SDN-C.



OpenDaylight at AT&T

AT&T SDN Controller Framework

Note: DG Builder is a design time-only component used to create and visualize the xml format Directed Graph

Directed Graph Builder
Directed Graph Files – XML (Eng Rules)
Network Data Model Files – YANG (i.e. IPAG EMT)
Service Data Model Files – YANG (i.e UNI port)
Service-related Artifacts for SLI, API Handlers, Network Adapters
Security Applications
Control Loop Applications
Service Orchestrators
API (REST)
API (REST)
API (REST)
API Handlers
External API calls
A&AI
Inventory
REST
OpenDaylight with AT&T customizations
SDN-C
Service Logic Interpreter
SDN-C Database
Service Logic/Eng. Rules
Assigned Resource Inventory
Config Tree
Operational Tree
Network Adapters
OpenStack Adapter
NETCONF
BGPCEP Adapter
Etc.
ADMIN Portal
8

External SDN controller

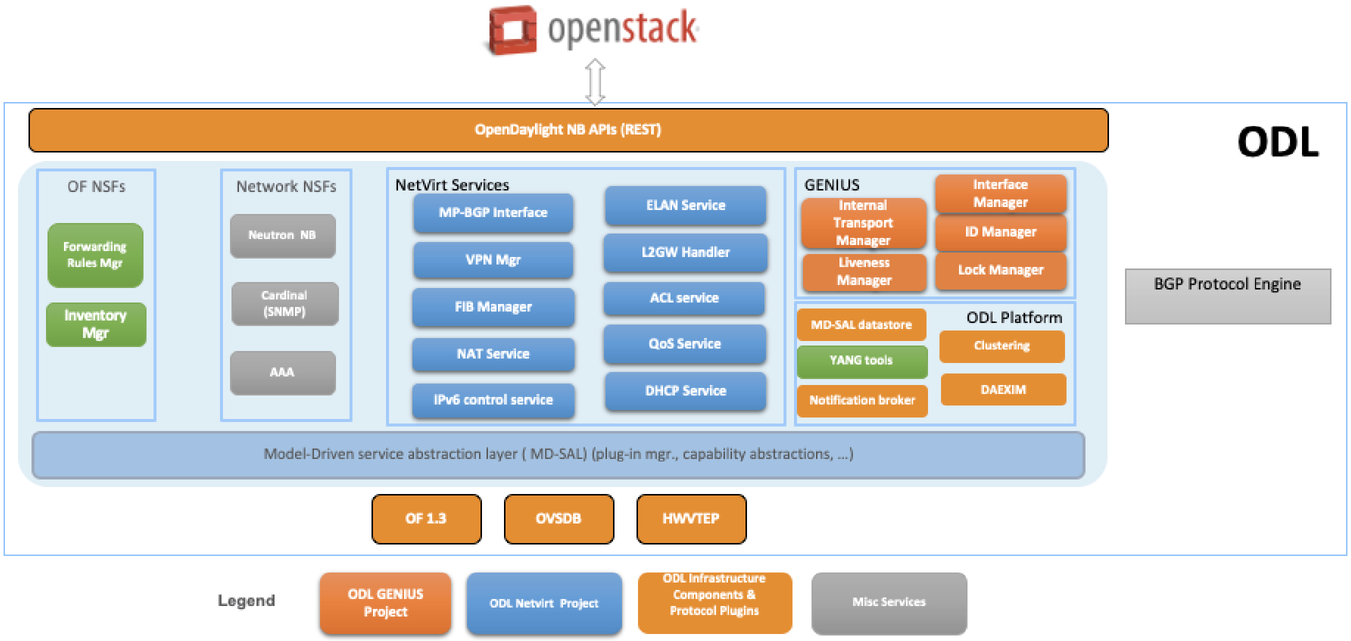
External SDN controller interfaces with the southbound interface of the ONAP SDN-C and is used to manage the Layer1-3 network devices in each network domain. Over the interface, the network configuration parameters extracted from the service logic are passed to the external SDN controllers. An OpenDaylight as an external SDN controller supports parameters for L3VPN, L2VPN, PCEP, NETCONF and more. The external OpenDyalight controller deploy the given configurations into the network devices.

Network Virtualization for Cloud and NFV

OpenDaylight NetVirt App can be used to provide network virtualization (overlay connectivity) inside and between data centers for Cloud SDN use case:

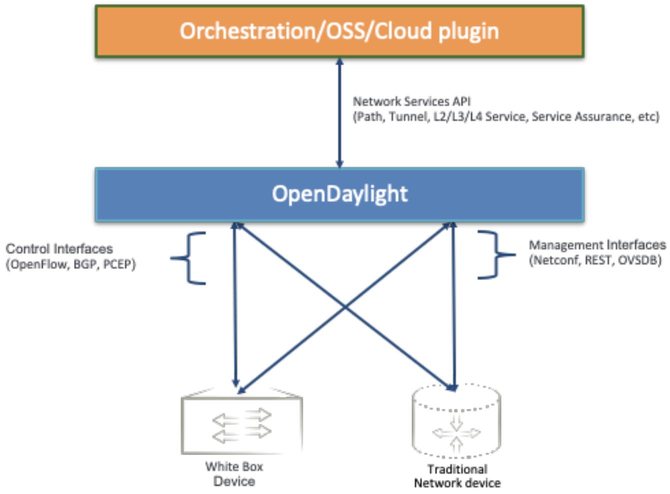
- VxLAN within the data center
- L3 VPN across data centers

The components used to provide Network Virtualization is shown in the diagram below:



Network Abstraction

OpenDaylight can expose Network Services API for northbound applications for Network Automation in a multi vendor network.



These are just a few of the common use cases but the platform can and continues to be tailored to several other use cases.

3.4 Open Switch

Editor: Mike Lazar

Overview

OpenSwitch (OPX) – an open source network operating system (NOS) and ecosystem - is an early adopter of emerging concepts and technologies (hardware and software disaggregation, use of open source, SDN, NFV and DevOps) which disrupt how networks and networking equipment are built and operated. Designed using a standard Debian Linux distribution with an unmodified Linux kernel, OpenSwitch provides a programmable high-level abstraction of network components, such as switching ASICs (Network Processors) and optical transceivers. Architected as a scalable, cloud-ready, agile solution, the open source OpenSwitch software implements a flexible infrastructure to enable both network operators and vendors to rapidly on-board open source Networking OS applications. OpenSwitch provides a YANG based programmatic interface, that can be accessed using Python, thus providing an environment well-suited for DevOps.

OpenSwitch Features

OpenSwitch (or OPX) provides an abstraction of hardware network devices in a Linux OS environment. It has been designed from its inception in order to support the newest technologies and concepts in the networking industry:

- In OPX, software is disaggregated from hardware, and software components are disaggregated as well.
    - OpenSwitch can be deployed on diverse networking hardware – only the low-level software layers SAI (Switch Abstraction Interface) and System Device Interface (SDI) are hardware specific and may need to be adapted. A minimum requirement is for hardware to be build around a standard ASIC, with Layer 2 switching, Layer 3 routing, ACL and QoS functionality.
- Makes extensive use of standard open source software, for instance:
    - ONIE installer
    - Linux Debian distribution with an unmodified Linux kernel
    - Switch Abstraction Interface (SAI) defined in Open Compute Project for interfacing with the networking ASIC.
- Integrates Linux native APIs with networking ASIC functionality. In OpenSwitch, networking features are also accessible using the Linux standard API's ("netlink"). Thus standard open source network packages (such as FRR) can be installed and supported in binary format.
- OPX supports containers and NFV. The Docker container environment (Docker CE), or any other Linux container environment, can be installed on any OpenSwitch device - in this environment, users can deploy their own containerized virtualized network functions (VNF).
- Supports programmability, automation and DevOps:
    - A robust and flexible programmatic interface – namely the Control Plane Services (CPS). The API is defined using YANG models and is accessible through Python (and C/C++).
    - The availability of a programmatic interface (CPS API/YANG models) allows integration with external orchestrators and SDN controllers
- Provides a rich set of networking features including full access to the networking ASIC ACL and QoS functionality using CPS API/YANG models.

OpenSwitch provides support for:

- L2 protocols: LLDP, LACP (link aggregation interfaces), 802.1q (VLAN interfaces), STP and bridge interfaces
- L3 protocols (e.g. BGP)
- ACL and QoS network functions (through CPS / YANG API's)
- Instrumentation: sFlow, telemetry
- Orchestration and management

Programmability and Automation

OpenSwitch supports a rich ecosystem for automated deployment, for instance:

- Ansible – various modules are already defined for OpenSwitch
- Zero-touch provisioning (ZTP) allows provisioning of OpenSwitch ONIE-enabled devices automatically, without manual intervention
- Puppet

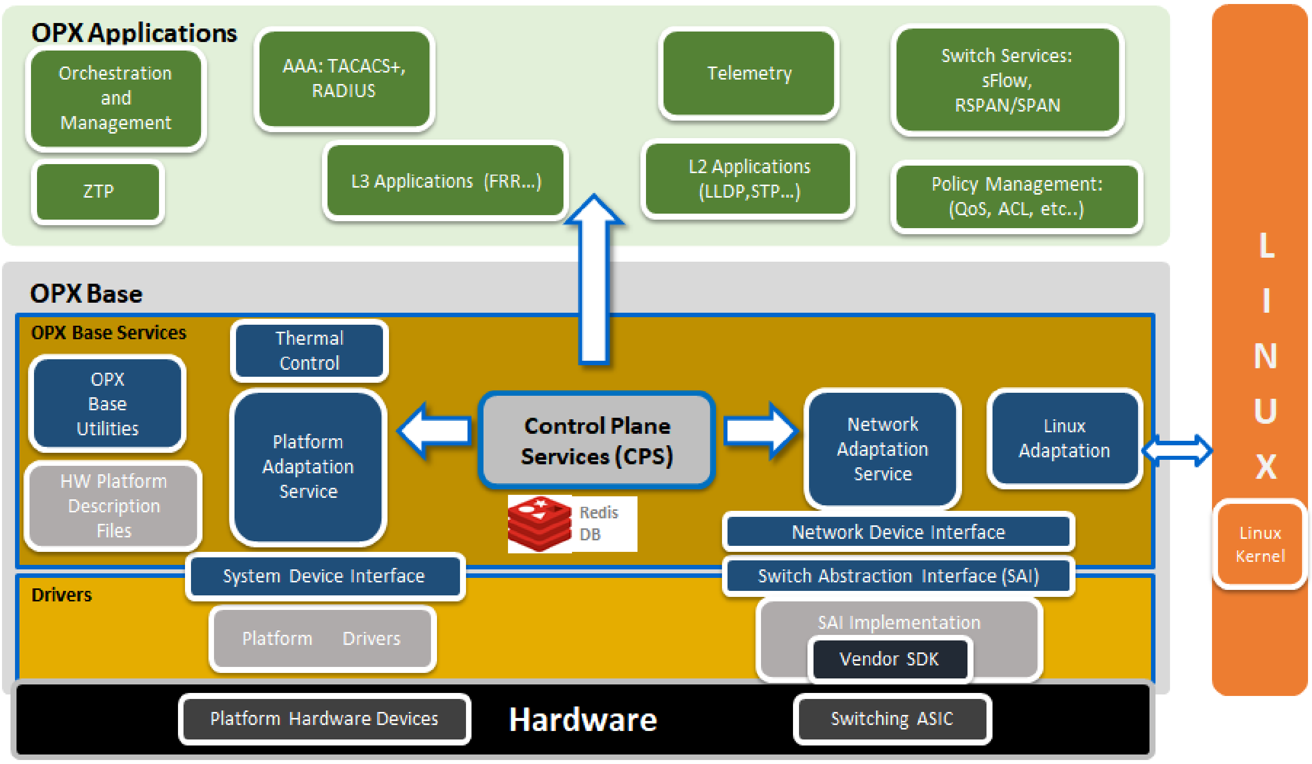
North-Bound Programmatic Interfaces

The OpenSwitch CPS programmatic interface is defined using YANG models, and in combination with Python, provides support for programming the network functionality, automation and DevOps. While the CPS API is the native OpenSwitch API, a REST API can be added as well, by mapping REST requests to CPS.

In addition, a set of OpenSwitch specific commands are available and can be invoked from a Linux shell (e.g. display the current software version, hardware inventory etc.).

OpenSwitch Architecture

The figure below illustrates the main areas of the OpenSwitch architecture:


OPX Applications
Orchestration and Management
AAA: TACACS+, RADIUS
Telemetry
Switch Services: sFlow, RSPAN/SPAN
ZTP
L3 Applications (FRR...)
L2 Applications (LLDP,STP...)
Policy Management: (QoS, ACL, etc..)
OPX Base
OPX Base Services
Thermal Control
OPX Base Utilities
Platform Adaptation Service
Control Plane Services (CPS)
Network Adaptation Service
Linux Adaptation
HW Platform Description Files
Redis DB
Network Device Interface
System Device Interface
Switch Abstraction Interface (SAI)
Drivers
Platform Drivers
SAI Implementation
Vendor SDK
Platform Hardware Devices
Hardware
Switching ASIC
LINUX
Linux Kernel

OPX Base

The key components of OPX Base are:

NAS – Network Adaptation Service

- Manages the high level abstraction of the switching ASIC
- NAS manages the middle-ware that associates physical ports to Linux interfaces, and adapts Linux native API calls (e.g. netlink) to the switching ASIC

PAS- Platform Adaptation Service

- A higher-level abstraction and aggregation of the functionality provided by the SDI component

CPS – Control Plane Service

- Object centric framework
- Mediates between application software components and the platform
- Provides a pub/sub model and set/get/delete/create
- Provides the framework for defining YANG modeled APIs - with Python and C/C++ bindings. In OPX, YANG models are used with an efficient CPS binary encoding.

SAI – Switch Abstraction Interface

- SAI API is an open interface that abstracts vendor-specific switching ASIC behavior

SDI – System Device Interface

- An API that provides a low level abstraction of platform specific hardware devices (e.g. fans, power supplies, sensors…)

OPX Applications

A variety of open source or vendor specific applications have been tested and can be deployed with OpenSwitch:

- FRR - BGP
- AAA: TACACS+, RADIUS
- Telemetry: Broadview, Packet Trakker
- Inocybe OpenDaylight integration
- NetSNMP
- Puppet
- Chef

It should be noted that these applications are not pre-installed with OpenSwitch. In a "disaggregated" model, users select applications to install them based on the requirements of a given network deployment.

In general, since OpenSwitch is based on Linux Debian distribution with an unmodified kernel, any Debian binary application can be installed and executed on OpenSwitch devices.

Hardware Simulation

OPX software supports hardware virtualization (or simulation). Software simulation of basic hardware functionality is also provided (simulation specific SAI and SDI components), and the higher layer software functionality can be developed and tested on generic PC/server hardware. OPX hardware simulation can be executed under Virtual Box, GNS3 / QEmu etc.

3.5 OPNFV and CNTT

Editor: Rabi Abdel

Introduction

The Common NFVI telco Taskforce (CNTT) is a LFN sponsored Taskforce (alongside GSMA sponsorship) and is an initiative that is aiming to minimise the number of Network Function Virtualisation Infrastructure (NFVI) configurations available (as a result of fragmentation of technologies/solutions provided by Open Source Project) and come up with few set of standardised infrastructure profiles for various NFV workloads as well as IT workloads (More information about CNTT can be found in CNTT GitHub).

CNTT compromises of:

- **Reference Model:** that explains how the infrastructure is exposed to workload in a standard way.
- **Reference Architecture(s):** Open Stack based and Kubernetes based (at the time when this WP is written) to deliver a conformant infrastructure based on those technologies.
- **Reference Implementation(s):** Open Stack and K8s based (at the time when this WP is written) to be used as a basis of any testing and validation activities.
- **Reference Certification(s):** Creating extensive set of testing harnesses to be used to test the conformance of any vendor implemented infrastructure to the CNTT specifications.

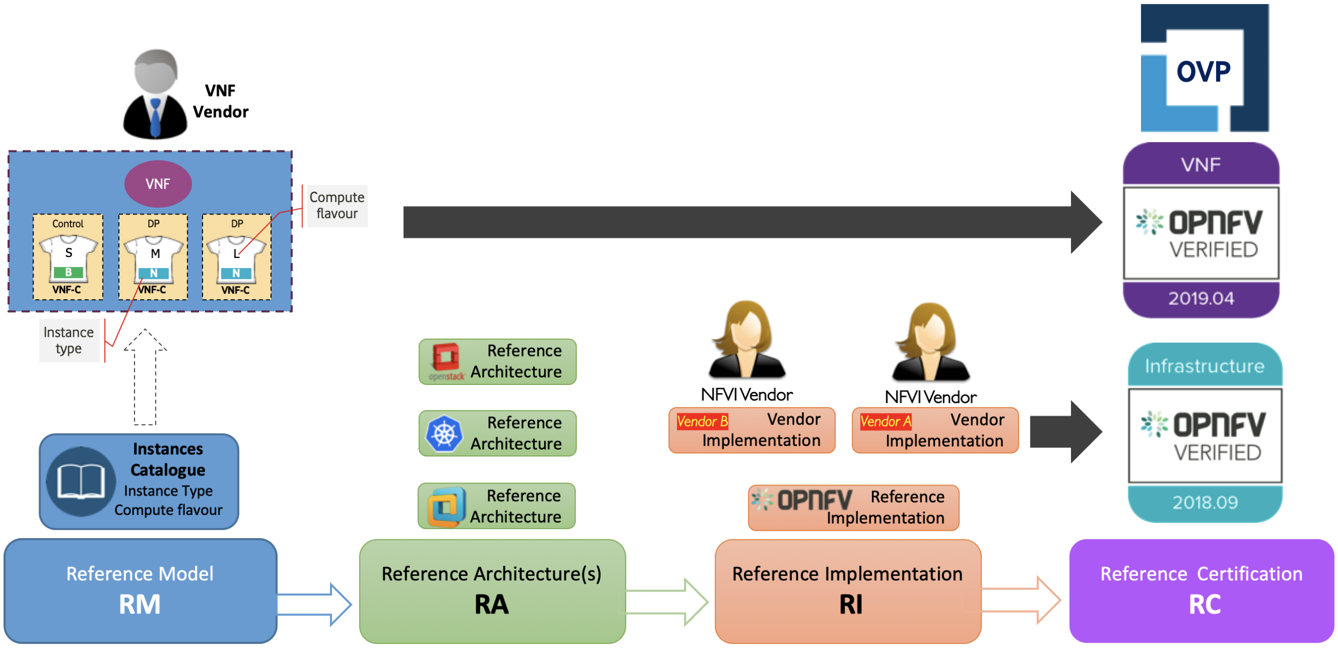


Figure X: The scope of CNTT.

Open Platform for NFV (OPNFV) is a project and community that facilitates a common NFVI, continuous integration (CI) with upstream projects, stand-alone testing toolsets, and a compliance and verification program for industry-wide testing and integration to accelerate the transformation of enterprise and service provider networks. Participation is open to anyone, whether you are an employee of a member company or just passionate about network transformation.

- Test tools
  - Functest and X-testing: The Functest project provides comprehensive testing methodology, test suites and test cases to test and verify OPNFV Platform functionality that covers the VIM and NFVI components.This project uses a "top-down" approach that will start with chosen ETSI NFV use-case/s and open source VNFs for the functional testing. The approach taken will be to
    - break down the use-case into simple operations and functions required.
    - specify necessary network topologies

        - develop traffic profiles
        - develop necessary test traffic
        - Ideally VNFs will be Open Source however proprietary VNFs may also be used as needed.

      This project will develop test suites that cover detailed functional test cases, test methodologies and platform configurations which will be documented and maintained in a repository for use by other OPNFV testing projects and the community in general. Developing test suites will also help lay the foundation for a test automation framework that in future can be used by the continuation integration (CI) project (Octopus). Certain VNF deployment use-cases could be automatically tested as an optional step of the CI process. The project targets testing of the OPNFV platform in a hosted test-bed environment (i.e. using the OPNFV test labs world wide).
      The X-testing aspect intends to integrate many test projects into a single, lightweight framework for automation that leverages the existing test-api and testdb for publishing results.
    - VSPerf: Although originally named to emphasize data plane benchmarking and performance testing of vSwitch and NFV Infrastructure, VSPerf has expanded its scope to multiple types of networking technologies (Kernel Bypass and Cloud-Native) and allow deployment in multiple scenarios (such as containers). VSPerf can utilize several different Traffic Generators and Receivers for testing, including several popular Hardware and Software-based systems. The VSPerf tool has many modes of operation, including the "traffic-generator-only" mode, where any virtual network manager sets up the path to be tested, and VSPerf automates the traffic generation and results reporting. VSperf is compliant with ETSI NFV TST009 and IETF RFC 2544.
    - StorPerf: A key challenge to measuring disk performance is to know when it is performing at a consistent and repeatable level of performance. Initial writes to a volume can perform poorly due to block allocation, and reads can appear instantaneous when reading empty blocks. The Storage Network Industry Association (SNIA) has developed methods which enable manufacturers to set, and customers to compare, the performance specifications of Solid State Storage devices. StorPerf applies this methodology to virtual and physical storage services to provide a high level of confidence in the performance metrics in the shortest reasonable time.
    - NFVBench: NFVbench is a lightweight end-to-end dataplane benchmarking framework project. It includes traffic generator(s) and measures a number of packet performance related metrics.
- Lab as a service
    - Lab as a Service (LaaS) is a “bare-metal cloud” hosting resource for the LFN community. We host compute and network resources that are installed and configured on demand for the developers through an online web portal. The highly configurable nature of LaaS means that users can reserve a Pharos compliant or CNTT compliant POD. Resources are booked are scheduled in blocks of time, ensuring individual projects and users do now monopolize resources. By providing a lab environment to developers, we enable more testing, faster development, and better collaboration between LFN projects.
- CI/CD for continuous deployment and testing of NFVI stacks
- OPNFV Lab Infrastructure
    - OPNFV leverages globally distributed community labs provided by member organizations. These labs are used by both developers of OPNFV projects as well as the extensive CI/CD tooling to continuously deploy and test OPNFV reference stacks. In order to ensure a consistent environment across different labs, OPNFV community labs follow a lab specification (Pharos spec) defining a high-level hardware configuration and network topology. In the context of CNTT reference implementations, any updates will be added to the Pharos specification in future releases.
- Feature projects working towards closing feature gaps in upstream open source communities providing the components for building full NFVI stacks
- Deployment tools
    - Airship
    - Fuel / MCP

blocked URL

Figure X: The scope of OPNFV.

CNTT and OPNFV

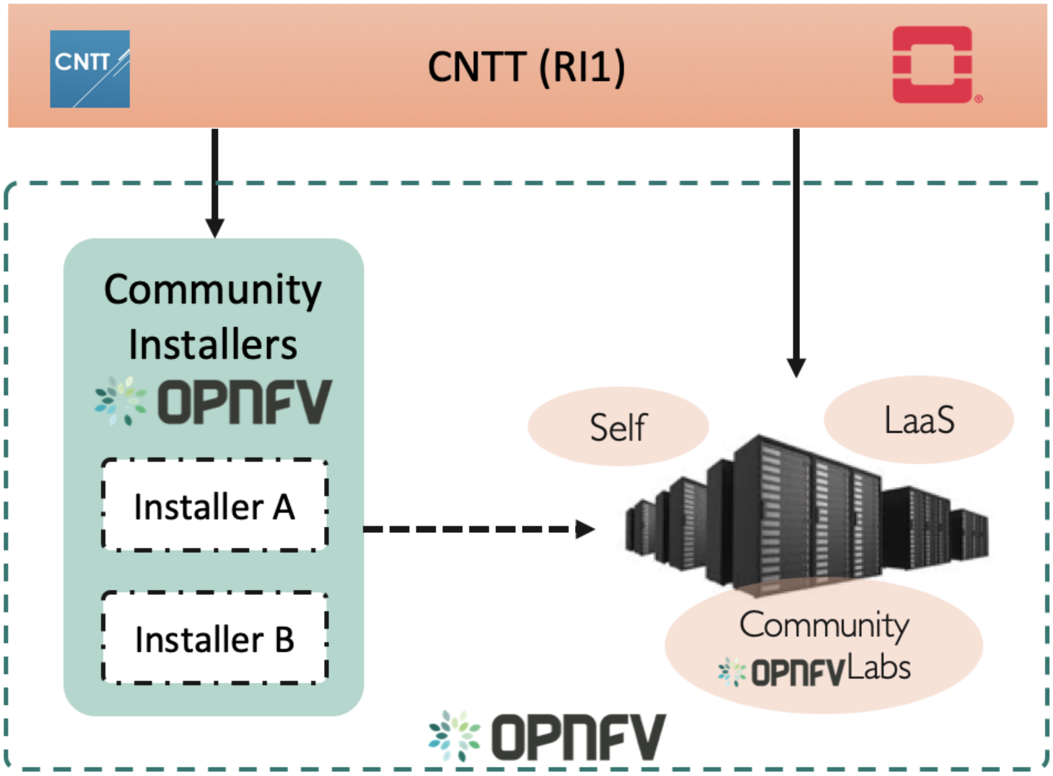


Figure X: Relation of CNTT OpenStack RI and OPNFV.

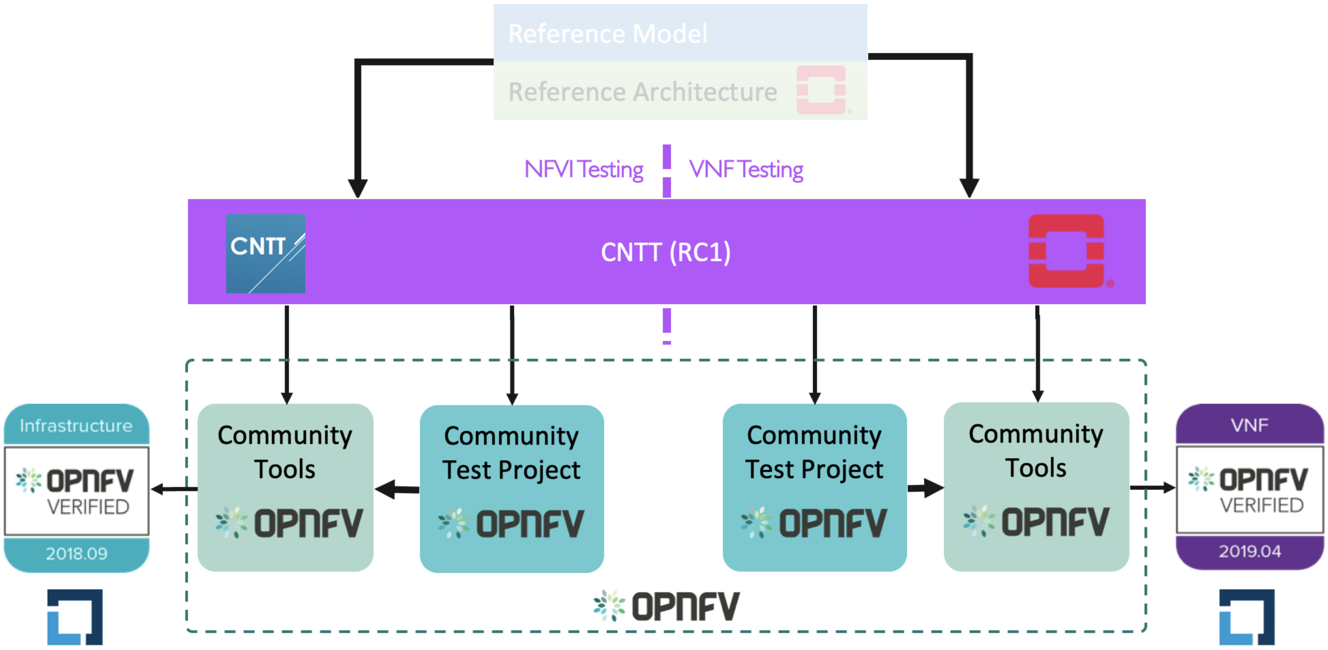


Figure X: Relation of CNTT OpenStack RC and OPNFV/CVC.

3.6 PNDA

Editor: Donald Hunter

Introduction

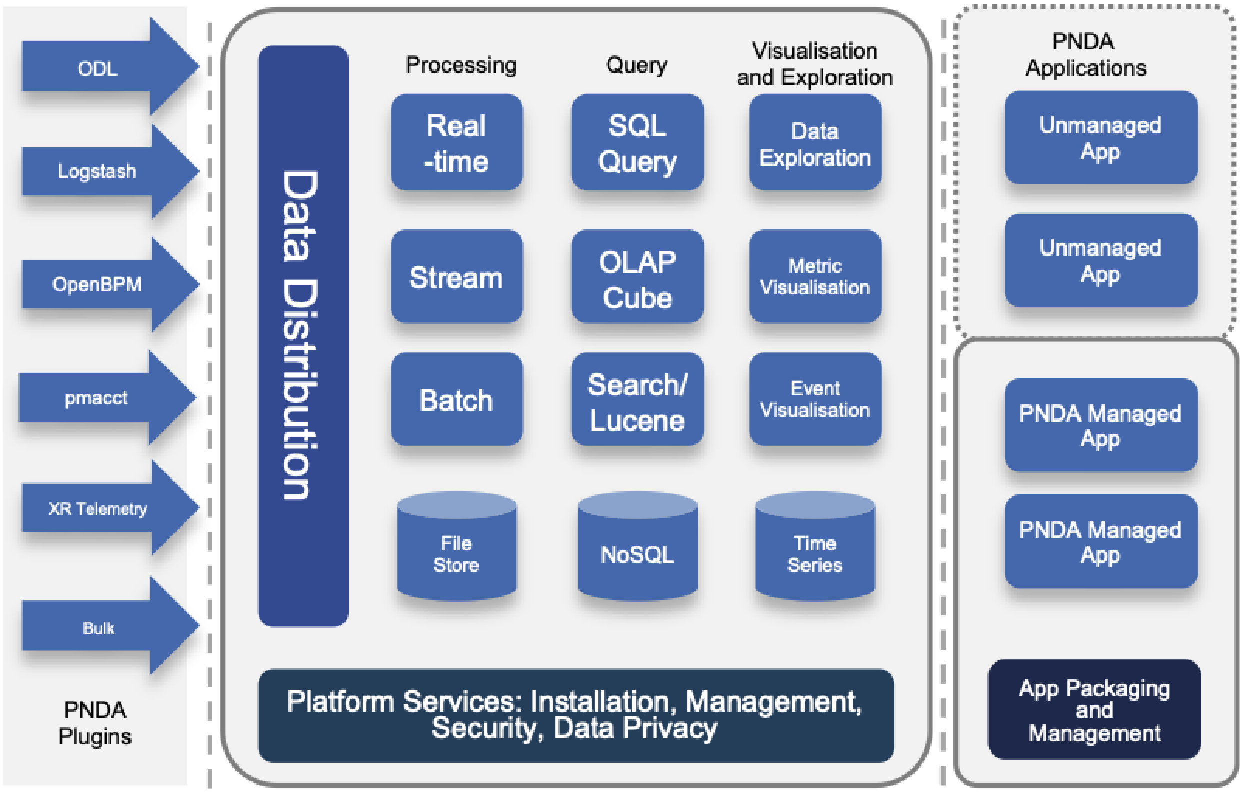
Innovation in the big data space is extremely rapid, but composing the multitude of technologies together into an end-to-end solution can be extremely complex and time-consuming. The vision of PNDA is to remove this complexity, and allow you to focus on your solution instead. PNDA is an integrated big data platform for the networking world, curated from the best of the Hadoop ecosystem. PNDA brings together a number of open source technologies to provide a simple, scalable, open big data analytics platform that is capable of storing and processing data from modern large-scale networks. It supports a range of applications for networks and services covering both the Operational Intelligence (OSS) and Business intelligence (BSS) domains. PNDA also includes components that aid in the operational management and application development for the platform itself.

The current focus of the PNDA project is to deliver a fully cloud native PNDA data platform on Kubernetes. Our focus this year has been migrating to a containerised and helm orchestrated set of components, which has simplified PNDA development and deployment as well as lowering our project maintenance cost. Our current goal with the Cloud-native PNDA project is to deliver the PNDA big data experience on Kubernetes in the first half of 2020.

PNDA provides the tools and capabilities to:

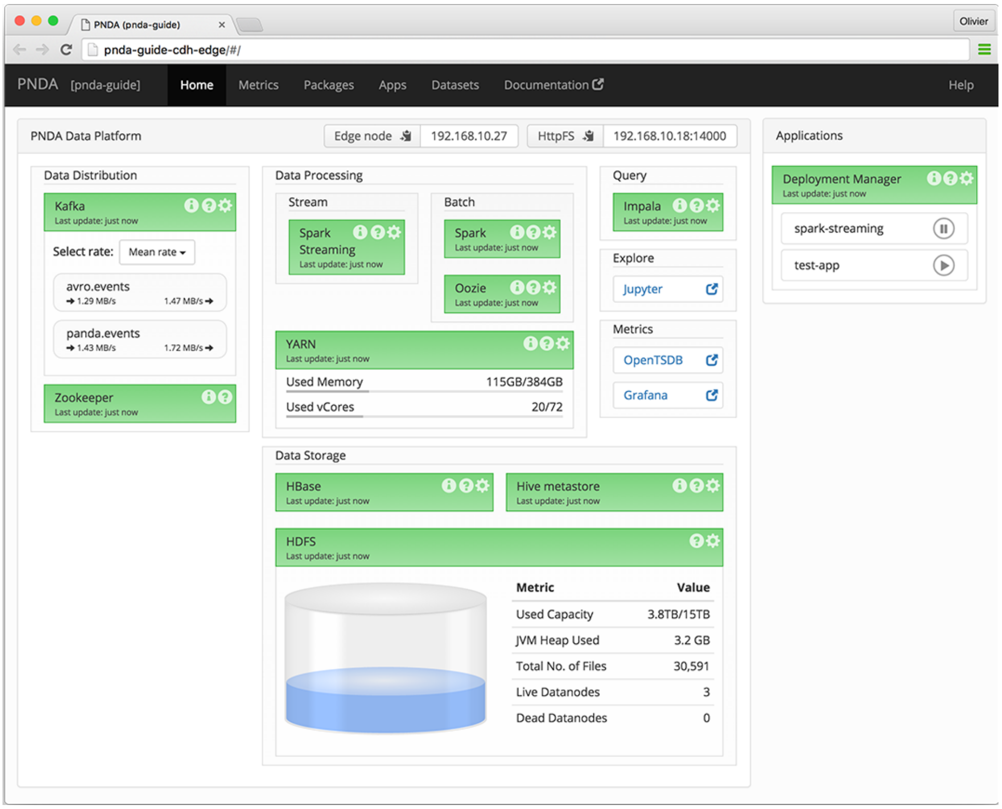
- Aggregate data like logs, metrics and network telemetry
- Scale up to consume millions of messages per second
- Efficiently distribute data with publish and subscribe model
- Process bulk data in batches, or streaming data in real-time
- Manage lifecycle of applications that process and analyze data
- Let you explore data using interactive notebooks

PNDA Architecture



PNDA Operational View

The PNDA dashboard provides an overview of the health of the PNDA components and all applications running on the PNDA platform. The health report includes active data path testing that verifies successful ingress, storage, query and batch consumption of live data.



Used Memory 115GB/384GB

Used vCores 20/72

Metric	Value
Used Capacity	3.8TB/15TB
JVM Heap Used	3.2 GB
Total No. of Files	30,591
Live Datanodes	3
Dead Datanodes	0

3.7 Tungsten Fabric

Editor: Prabhjot Singh Sethi

Introduction:

Tungsten Fabric provides a highly scalable virtual networking and security platform that works with a variety of virtual machine and container orchestrators, integrating them with physical networking and compute infrastructure. It is designed to support multi-tenant networks in the largest environments while supporting multiple orchestrators simultaneously.

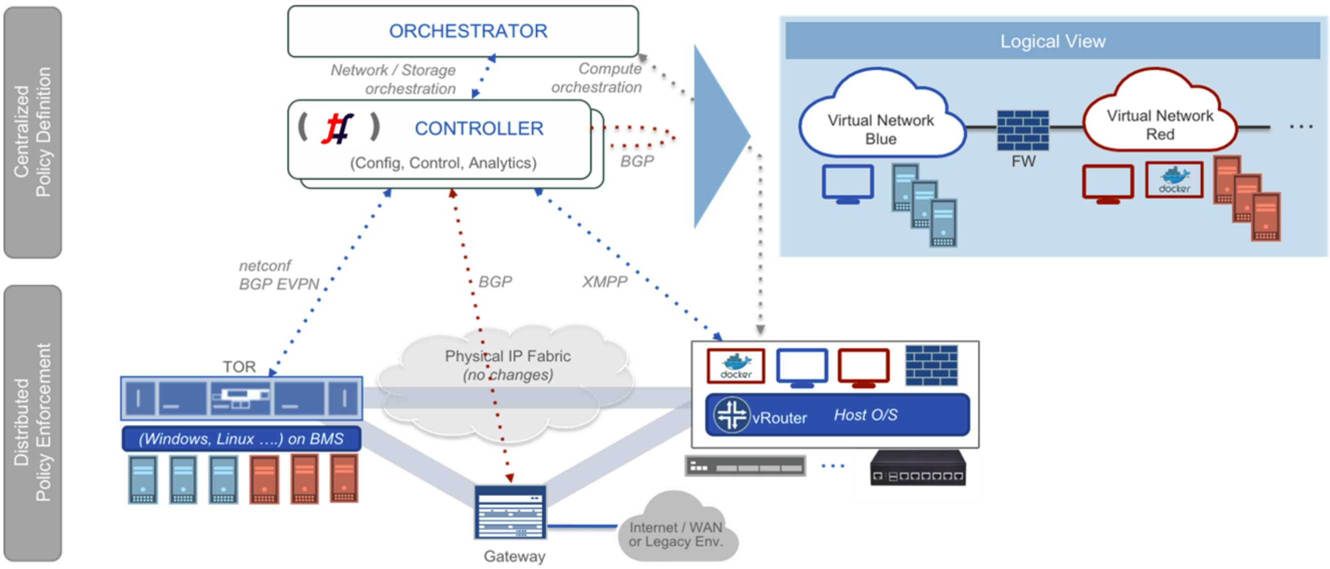
Tungsten Fabric enables usage of same controller and forwarding components for every deployment, providing a consistent interface for managing connectivity in all the environments it supports, and is able to provide seamless connectivity between workloads managed by different orchestrators, whether virtual machines or containers, and to destinations in external networks.

Architecture Overview:

Tungsten Fabric controller integrates with cloud management systems such as OpenStack or Kubernetes. Its function is to ensure that when a virtual machine (VM) or container is created, it is provided with network connectivity according to the network and security policies specified in the controller or orchestrator.

Tungsten Fabric consists of two primary pieces of software

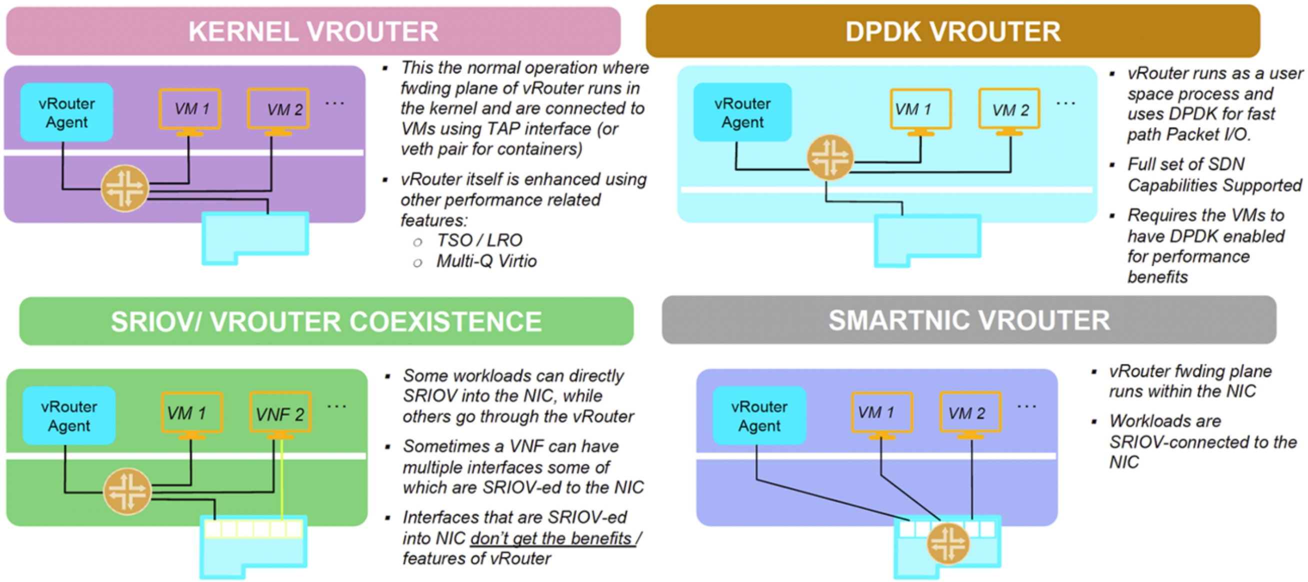
- Tungsten Fabric Controller– a set of software services that maintains a model of networks and network policies, typically running on several servers for high availability
- Tungsten Fabric vRouter– installed in each host that runs workloads (virtual machines or containers), the vRouter performings packet forwarding and enforces network and security policies.


Centralized Policy Definition
Distributed Policy Enforcement
ORCHESTRATOR
Network / Storage orchestration
Compute orchestration
CONTROLLER
(Config, Control, Analytics)
BGP
Logical View
Virtual Network Blue
FW
Virtual Network Red
...
docker
netconf BGP EVPN
BGP
XMPP
TOR
Physical IP Fabric (no changes)
(Windows, Linux ....) on BMS
vRouter
Host O/S
Gateway
Internet / WAN or Legacy Env.

Technologies used:

Tungsten Fabric uses networking industry standards such as BGP EVPN control plane and VXLAN, MPLSoGRE and MPLSoUDP overlays to seamlessly connect workloads in different orchestrator domains. For example, virtual machines managed by VMware vCenter and containers managed by Kubernetes.

Tungsten Fabric supports four modes of datapath operation:



KERNEL VROUTER

- *This the normal operation where fwding plane of vRouter runs in the kernel and are connected to VMs using TAP interface (or veth pair for containers)*
- *vRouter itself is enhanced using other performance related features:*
  - *TSO / LRO*
  - *Multi-Q Virtio*

DPDK VROUTER

- *vRouter runs as a user space process and uses DPDK for fast path Packet I/O.*
- *Full set of SDN Capabilities Supported*
- *Requires the VMs to have DPDK enabled for performance benefits*

SRIOV/ VROUTER COEXISTENCE

- *Some workloads can directly SRIOV into the NIC, while others go through the vRouter*
- *Sometimes a VNF can have multiple interfaces some of which are SRIOV-ed to the NIC*
- *Interfaces that are SRIOV-ed into NIC don't get the benefits / features of vRouter*

SMARTNIC VROUTER

- *vRouter fwding plane runs within the NIC*
- *Workloads are SRIOV-connected to the NIC*

Tungsten Fabric connects virtual networks to physical networks:

- Using gateway routers with BGP peering
- Using ToR with OVSDB
- Using ToR managed with Netconf and BGP-EVPN peering
- Directly through datacenter underlay network (Provider networks)

Key Features:

Tungsten Fabric manages and implements virtual networking in cloud environments using OpenStack and Kubernetes orchestrators. Where it uses overlay networks between vRouters that run on each host. It is built on proven, standards-based networking technologies that today support the wide-area networks of the world's major service providers, but repurposed to work with virtualized workloads and cloud automation in data centers that can range from large scale enterprise data centers to much smaller telco POPs. It provides many enhanced features over the native networking implementations of orchestrators, including:

- Highly scalable, multi-tenant networking
- Multi-tenant IP address management
- DHCP, ARP proxies to avoid flooding into networks

- Efficient edge replication for broadcast and multicast traffic
- Local, per-tenant DNS resolution
- Distributed firewall with access control lists
- Application-based security policies based on tags
- Distributed load balancing across hosts
- Network address translation (1:1 floating IPs and distributed SNAT)
- Service chaining with virtual network functions
- Dual stack IPv4 and IPv6
- BGP peering with gateway routers
- BGP as a Service (BGPaaS) for distribution of routes between privately managed customer networks and service provider networks
- Integration with VMware orchestration stack

3.8 SNAS

Editor: <TBD>

Streaming Network Analytics System (project SNAS) is a framework to collect, track and access tens of millions of routing objects (routers, peers, prefixes) in real time.

SNAS extracts data from BGP routers using a BGP Monitoring Protocol (BMP) interface. The data is parsed and made available to consumers through a Kafka message bus. Consumers applications in turn can perform further analytics and visualization of the topology data.

blocked URL