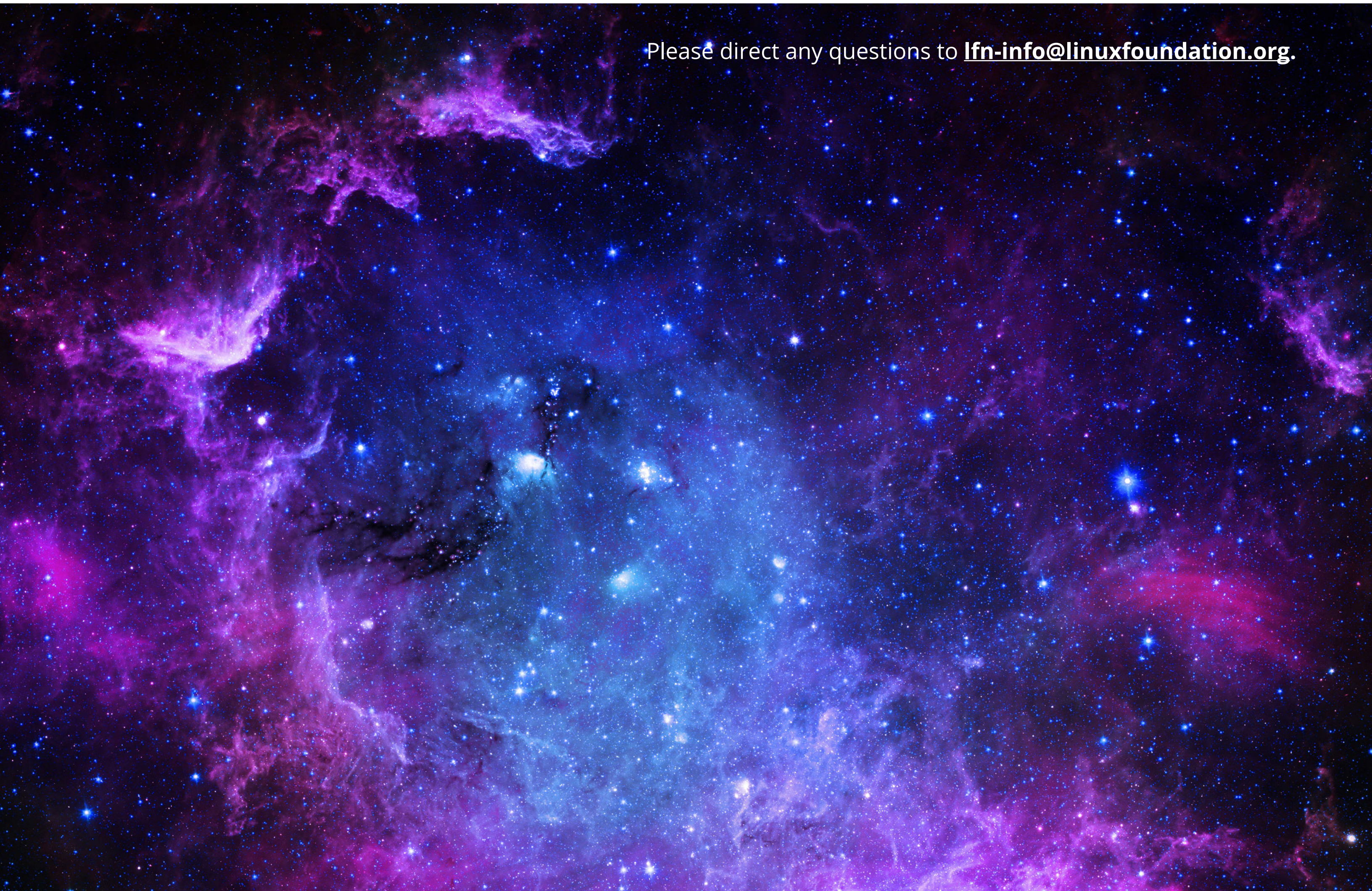
LF NETWORKING

LF Networking Whitepaper

Presented by the LFN Technical Advisory Council (TAC)



Please direct any questions to **lfn-info@linuxfoundation.org**.

Table of Contents

Introduction

Summary

This whitepaper is intended for anyone who is interested in the intersection of modern network design and the open source networking project landscape. System architects, developers, product and project managers, network operators, system integrators, and more should all find useful information here that will help better understand the state-of-the-art in networking technology and determine how the LF Networking (LFN) projects from the Linux Foundation may be used as building blocks for modern networks. It was prepared by a working group of the LFN Technical Advisory Council (TAC).

This document does not intend to prescribe the "right" solution for building a network. There is more than one way of doing that and it all depends on the network designer's preferences and available resources. Instead, we try to introduce the capabilities of each LFN project and suggest potential ways they can be used in harmony. One of the goals of this whitepaper is to solicit engagement from potential users and contributors to the LFN projects. You are strongly encouraged to share your insights and thoughts with the LFN community on this document as well as on any of the projects themselves. The LFN Technical Advisory Committee (TAC) mailing list is one place to start such engagement. Please see the details in chapter 5.

Background

Just over two decades ago, the network was mainly a fixed voice network in widespread use in mature markets but with limited reach in emerging economies. Cellular infrastructure and the internet were only just starting to appear. Each regional network was built and run by a Communications Services Provider (CSP) who would acquire the underlying proprietary technology from Network Equipment Providers (NEPs) and charge subscribers to use the network. The resulting networks were largely homogenous with most of the equipment typically coming from a single vendor.

In this traditional model, the technology and product roadmap of the CSP was the technology and product roadmap of their NEP which was driven by jointly developed standards. Standards-led product development led to decentralized yet globally compatible service offerings, enabling worldwide roaming and an unprecedented level of compatibility over defined reference points and across many vendors. Development costs for the NEPs were high, ultimately resulting in an industry consolidation. Capital and operational costs were high for the CSP, but these costs were predictable and could be recovered over time from the subscriber population in a relatively uncompetitive market.

Situation

Move forward a short twenty years or so and the industry has transformed. Mobile and internet are booming worldwide. Traffic has moved from circuit-switched voice to packet-switched data. The network has far greater reach: hundreds of millions of people in mature and emerging economies worldwide now stay connected to the network to regularly access valued consumer services such as streaming, and business services such as video conferencing. Capacity has significantly increased and demand continues to grow as more devices connect to the network and services consume more bandwidth. Markets today are far more competitive and communications services are increasingly commoditized. As consumers, we pay less and get more. The network itself has become the foundation for the new, global digital economy of the 21st century.

Despite these advances, if we scratch the surface of the industry a little, we see that business models and ways of cooperating around technology remain largely unchanged from twenty or even one hundred years ago.

With network functions deployed as physical appliances, being pre-integrated bundles of hardware and software, new services require changing the physical structure of the network. This takes as long as months or even years and incurs the cost of a field workforce. With Network Functions Virtualization (NFV), there has been movement from appliances to separation of hardware and software which has reduced time to deploy new services.

The industry challenge is that the traditional networks that are the foundation of the CSP business can, in fact, be slowing the business. With consumers paying less to get more each year, the CSP must continuously create new services and provide more bandwidth at a lower cost each year just to remain viable as a business. The underlying network technologies and closed supplier ecosystems prevent the CSP from leveraging the open market to introduce new capabilities to reduce costs or innovate to create a new service. The tipping point has already been reached in highly competitive markets such as India where CSPs are disappearing from the market or are merging but still losing customers to competition. From the once flourishing NEP ecosystem, less than a handful of vendors remain today. Despite the network itself becoming the foundation for the new global digital economy, the industry that provides the network is facing significant challenges.

How then does the communications industry and its suppliers move to the open model of innovation, development and collaboration enjoyed by other technology-based industries? Enter open source. Benefits of open source in the enterprise domain include higher quality software, improved security, lower cost of ownership and greater innovation. Linux has long been a leader in open source for operating systems, successful as a result of strong governance and collaboration and without one vendor controlling development or direction. Traditionally, the pace of innovation in the networking industry has been determined by a process that included standards creation, separate implementation based on each NEP's interoperation, and multi-vendor interoperability testing. Oftentimes, several iterations of the process were required until the technology was ready for wide deployment.

"Standards and open source, better together" means that open source software can accelerate and simplify the process as the open source implementation of the standards provides immediate feedback loop to the standard creation, and a reference implementation for equipment providers and operators.

LF Networking Arrives

In recognizing both the importance of communications to the emerging global digital economy and to improving lives of people everywhere, and the challenges facing the communications industry, the Linux Foundation established LF Networking (LFN) as the umbrella organization to provide platforms and building blocks for network infrastructure and services across service providers, cloud providers, enterprises, vendors, and system integrators that enable rapid interoperability, deployment, and adoption.

LFN increases the availability and adoption of quality open source software to reduce the cost of building and managing networks, thus giving CSPs, cloud providers, enterprises and others the means to:

- significantly reduce cost of the networks on which their business depends
- gain control of their network and product roadmap
- introduce new capabilities and services more quickly
- reduce capital and operational costs, for example by increasing the number of functions that can be remotely deployed and maintained, through automating operations and through increased use of commodity hardware
- increase security through having multiple entities review the software

While there are benefits from using open source, the benefits are greater to those who also contribute to open source. Per an article in the Harvard Business Review, this is because “Companies that contribute and give back learn how to better use the open source software in their own environment,” and “...paying employees to contribute to such software boosts the company’s productivity from using the software by as much as 100 percent, when compared with free-riding competitors.”

The value of open source is not missed on the NEPs, many of who use Linux as the operating system for their network equipment. Increased adoption of open source in other areas of their products will help NEPs improve quality and output while reducing development and maintenance costs.

China Mobile, AT&T, and Rakuten are examples of organizations using open source. US military research agency DARPA has stated its intention of establishing an open source program for 5G and the US Congress is legislating to provide funding. It is expected that open source will significantly displace proprietary systems from networks in the coming years, and LFN has a significant contribution to make.

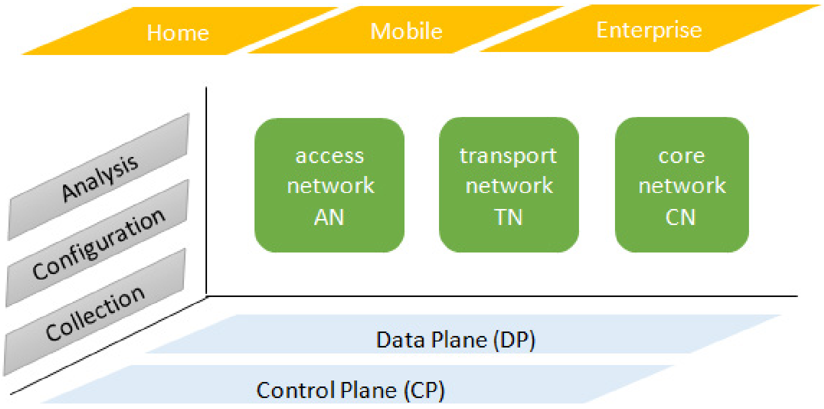
As this transition proceeds, in coming years when you scratch the surface of the network, you will see a markedly transformed network underpinning the modern, digital economy.

LFN Landscape

External Landscape - Other open source projects, Standards Definition Organizations (SDOs)

As an open source community, where operators and equipment vendors collaborate on building the reference implementation of next-generation network construction, LFN has been committed to the collaboration of standards and open source since its establishment, and has promoted cross-organizational industry cooperation including a series of whitepapers.

The traditional CSP has a relatively simple business model, a long network construction and service introduction cycle, and is accustomed to business operations based on user access and basic network planning, construction, and maintenance according to the physical network technology field. Hence, we are now facing a partially standardized Communication Technology (CT) industry chain with highly standardized network functions (NF) as well as highly customized operation and maintenance management.

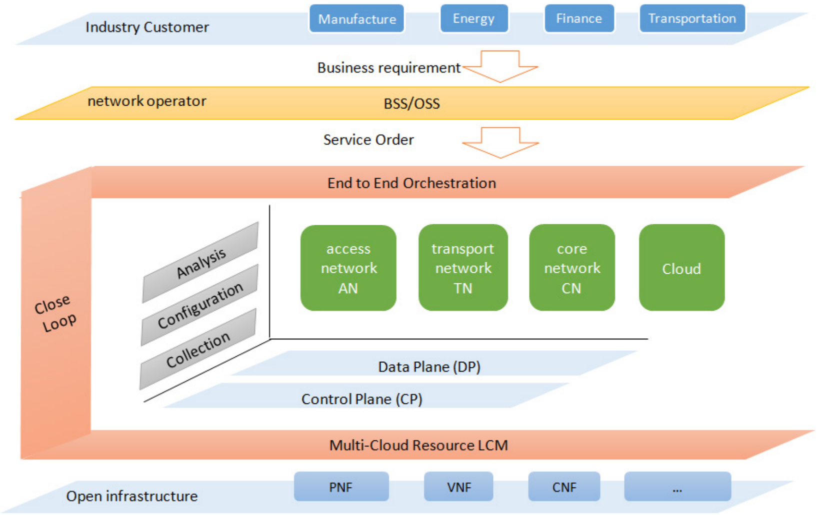


Traditional communication network technology domain Landscape

In order to break the closed business R&D and equipment R&D ecology of the communications industry, the industry's leading CSPs joined hands with vendors in creating LF Networking (or LFN in short), as a vehicle to unite industry forces such as standards and open source, hoping to build a truly open next-generation network innovation technology ecosystem.

This "openness" is embodied in the following three aspects:

- Open infrastructure: Through disaggregation between software and hardware for telecommunications equipment, it is now possible for the CT industry to share a common hardware infrastructure with the IT industry, and even further, to promote jointly a common open hardware platform. This brings benefits for innovation and scale as a result of the industrial integration of CT and IT at the hardware level.
- Open NF O&M (Operation and Maintenance): Disaggregation between the control plane, data planes and management plane of NF software, combined with a generalized and standardized end-to-end orchestration platform to stitch centralized control and management domain controllers, is driving a transition of the NF software from complex monolithic systems to sophisticated micro services. On the one hand, it can better learn and draw on the advanced technology of common IT architecture and public software components, and on the other hand, it can help lower the entry barrier standard of small and medium equipment providers and promote industrial integration at the level of CT and IT software.
- Open business customization: Through the decoupling of the general business design orchestration management platform from specific business scenarios and specific professional fields, the "building block" type of business design and deployment, operation and maintenance customization capabilities are realized. In this way, basic network service providers, communication business service providers, and vertical industry service providers are converging at the business level.



Vision of Next Generation Network Technology Landscape (with LFN)

As shown in the figure, in order to achieve the three-level open target vision described above, it is necessary to provide standardized touch points for interoperability:

- with a common infrastructure and management platform;
- with network elements in various network technology areas for Lifecycle Management (or LCM in short), collection, analysis and configuration control;
- to stitch across various network technologies to achieve an end-to-end service orchestration and capability exposure platform.

Internal Landscape

LFN projects address the touch points mentioned above and offer functionality related to the different layers required for building a modern network.

It's worth looking at the projects within the LFN umbrella in the context of the network itself.

This starts with the Transport Layer (also referred to as the 'Datapath'), where user data is moved from one point to another and speed and reliability are key. The FD.io project focuses on fast packet processing, with the promise to move data up to one hundred times faster. FD.io's work applies to multiple layers of the network including Layer 2 Data, Layer 3 Network, Layer 4 Transport, Layer 5 Session, and Layer 7 Application.

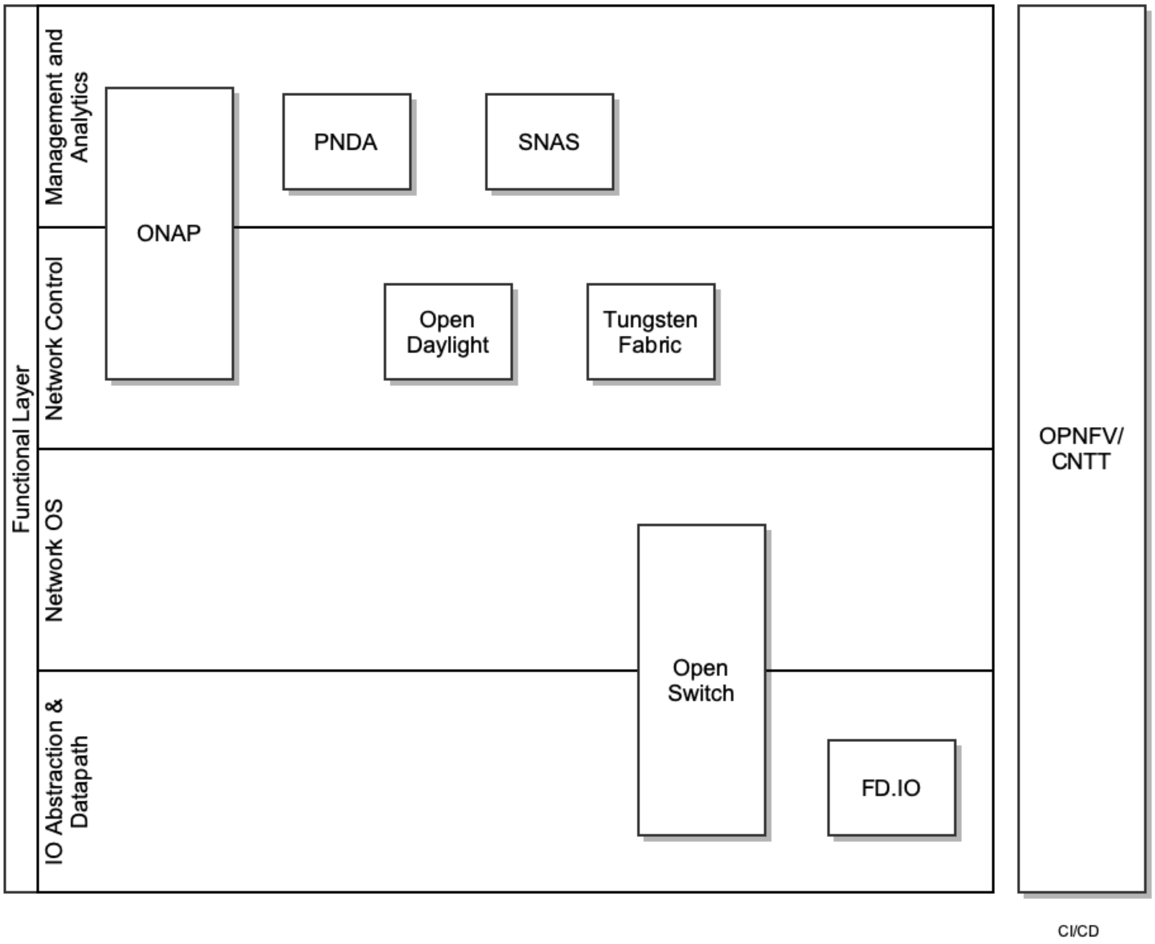
The next layer is the Network Operating System (NOS), where the essential software components required for building a network device are integrated and packaged together. OpenSwitch (OPX) is a NOS that abstracts the complexity and hardware implementation details of network devices, and exposes a unified interface towards the higher network layers.

The Network control layer is where end-to-end complex network services are designed and executed. It relies heavily on network modeling that allows network designers to create the desired services. ONAP, OpenDaylight, and Tungsten Fabric take network service definitions as input, break them into their more basic building blocks, and then interface with the lower layers of the network to instantiate and control the service components. The network control layer also provides the interface to Operational and Business Support Systems (OSS/BSS) where ONAP provides the management and orchestration functions that ensure OSS/BSS can manage modern dynamic networks.

The top layer of network functionality includes the components which provide visibility into the state of the network as well as automated network management.

PNDA and SNAS can collect high volumes of network data in real time, and make them available to external management systems. ONAP is another entity that collects network performance and fault data. All the collected data can be used by the ONAP policy-driven control loop automation which can take action to dynamically control the network in response to changing demand. Network faults may also be detected in this layer and in many cases the combined analytics capabilities of the projects can be used to trigger automation that provides self-healing functionality to the network.

The OPNFV project and the Common NFVi Telco Taskforce (CNTT) initiative focus on the integration of the different layers and provide tools and reference architectures for building networks. In addition, OPNFV provides a verification program for network infrastructure and virtual network functions to ensure that the different components of the network are fully compatible with each other and provide the expected functionality and performance.



Brief Project Overview

Table of Contents

3.1 FD.IO

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 home for 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). FD.io provides "universal" dataplane functionality and acceleration at scale, within the LFN ecosystem.

The core component is the highly modular Vector Packet Processing (VPP) library (details below) which allows new graph nodes to be easily "plugged in" without changes to the underlying codebase. 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 that 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 as he 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.

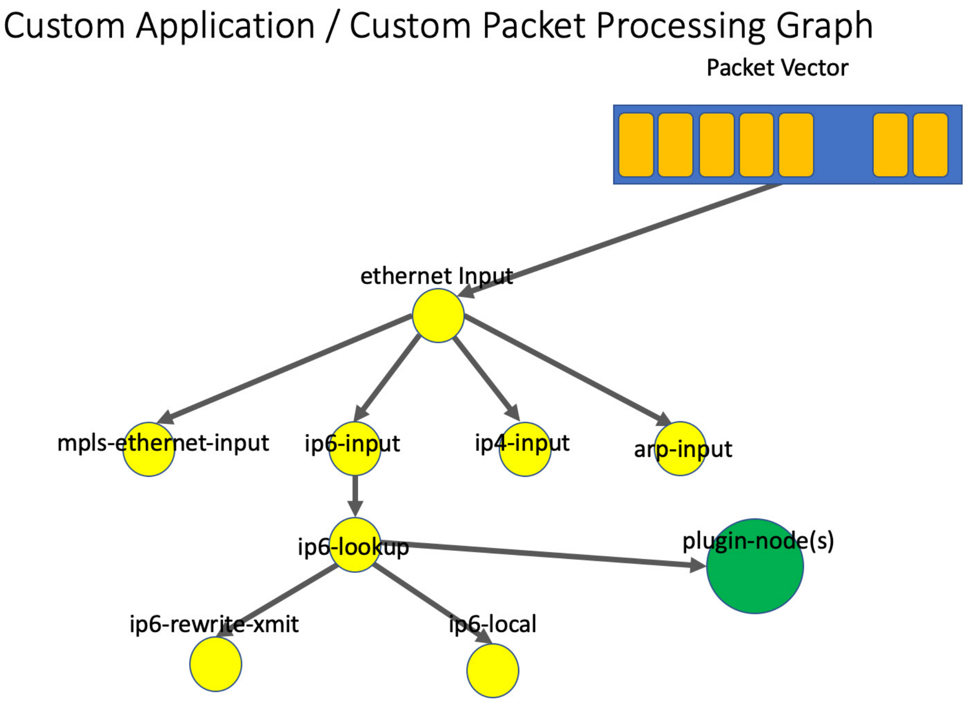
Further optimizations that this approach enables are pipelining and prefetching to minimize read latency on table data and parallelize packet loads needed to process packets.

The Packet Processing Graph

The FD.io VPP packet processing pipeline is decomposed into a 'Packet Processing Graph'. This modular approach at the core of FD.io VPP design 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
- That is fast, where 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.

IPSec

VPP Release 20.01 makes it so IPSec can now be processed in a single solution instance—whether appliance, VM, or cloud instance—across multiple cores. This makes it safe to run multi-core, because now the Security Associations (SAs) are bound to the initial core they were seen on. Learn more here.

Plugins

Plugins are shared libraries and are loaded at runtime by FD.io VPP. It 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 notably 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 a 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 plugins to enhance its capabilities. Cloud load-balancing is just one of a 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 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.

3.2 ONAP

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.

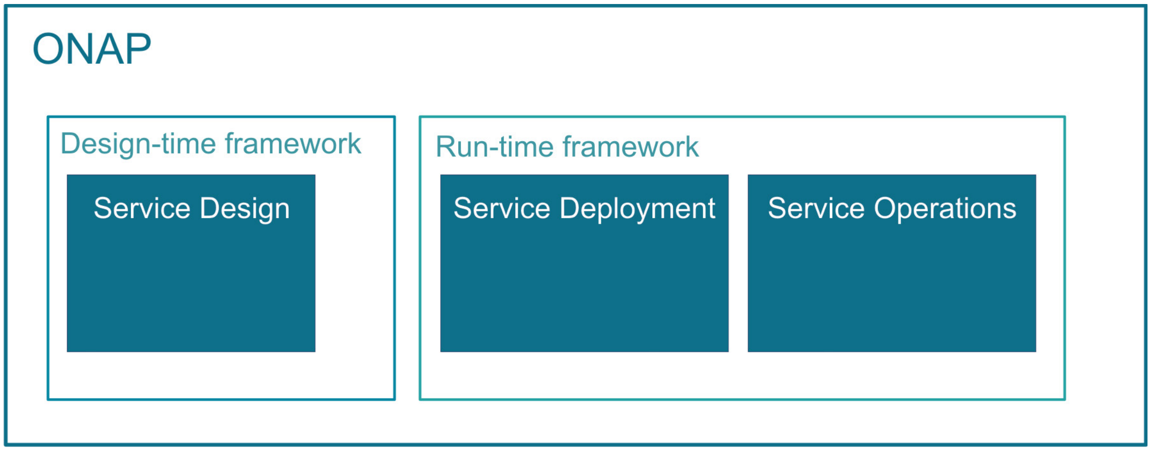
Prior to ONAP, telecommunication network operators had to keep up with the scale and cost of manual changes required to implement new service offerings, from installing new data center equipment to, in some cases, upgrading customer equipment on-premises. Many operators are seeking to exploit Software 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 massively scalable Virtual Infrastructure Manager (VIM) automation capabilities. These capabilities handle not only both physical and virtual network elements, but also multi-site and multi-VIM deployments. 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 primary activities—that is designing, deploying and operating services—are provided based on ONAP's two major frameworks, namely a Design-time framework and a 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; and 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.

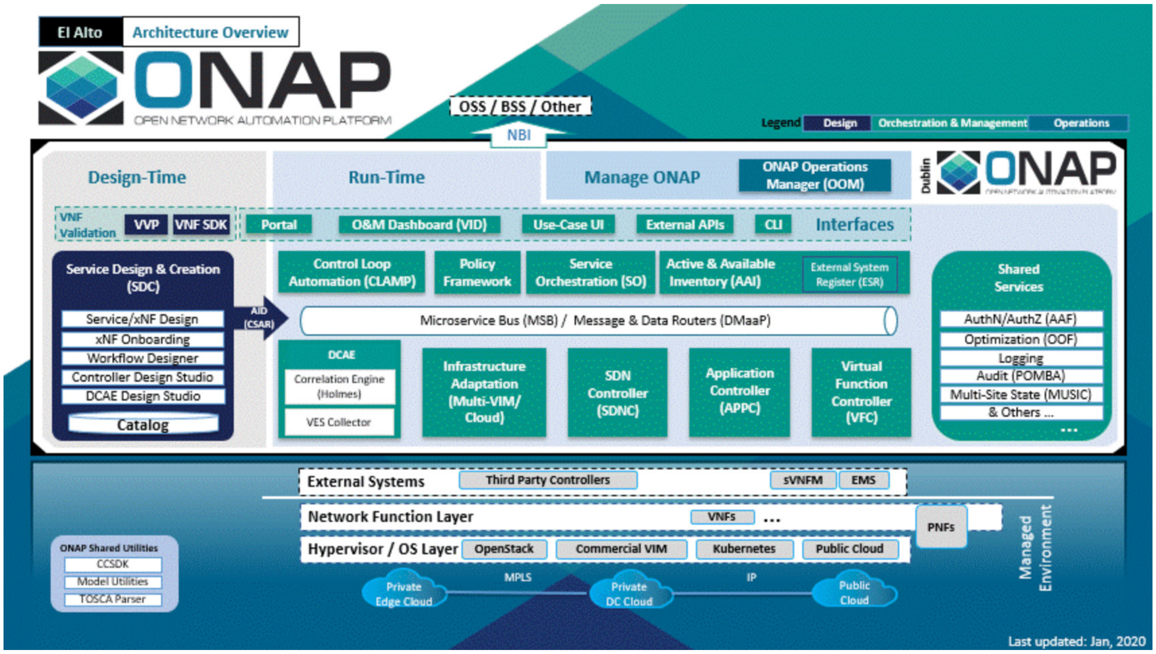
ONAP enables product- or service-independent capabilities for design, deployment and operation, in accordance with the following foundational principles:

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 the sourcing of 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



ONAP Architectural Overview Diagram

The Architecture Overview 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:
    a. TOSCA C-SAR package is stored in the Catalog
    b. A 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.
5. Controllers applying configuration on VNFs

Service Operations

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

Use Cases

As part of each release, the ONAP community also defines blueprints for key use cases, which the user community expects to pursue immediately. Testing these blueprints with a variety of open source and commercial network elements during the development process provides the ONAP platform developers with real-time feedback on in-progress code, and ensures a trusted framework that can be rapidly adopted by other users of the final release. Current blueprints include Broadband Service (BBS), 5G (updated), CCVPN (updated), VoLTE, and vCPE.

Benefits of ONAP

Open Network Automation Platform provides the following benefits:

- a 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
- both Service and VNF Configuration capability
- 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 global city. A list of past and current releases may be found here.

3.3 OPNFV and CNTT

Open Platform for NFV (OPNFV)

OPNFV is a project and community for Communication Service Providers (CSPs) and their supply chains, focused on network transformation and collaboration, to continuously improve the efficiency and predictability of consuming and deploying NFV infrastructure, VNFs, and CNFs. Iterating through implementations of toolsets, automation, verification, conformance, and performance, aligned with normalized architectures, and enabling the community to drive down cost and time to revenue for network services. OPNFV also collaborates with CNTT to realize the reference implementation and reference conformance tests.

TEST TOOLS

Functest: 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. This approach does the following:

- breaks down the use-case into simple operations and functions required
- specifies necessary network topologies
- develops traffic profiles
- develops necessary test traffic

Note: Ideally VNFs will be open source; however, proprietary VNFs may also be used as needed.

The 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 continuous 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 community test labs worldwide). Many test projects are integrated into a single, lightweight framework for automation (x-testing) that leverages the OPNFV test-api and testdb frameworks 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 allows deployment in multiple scenarios (such as containers and OpenStack). 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.

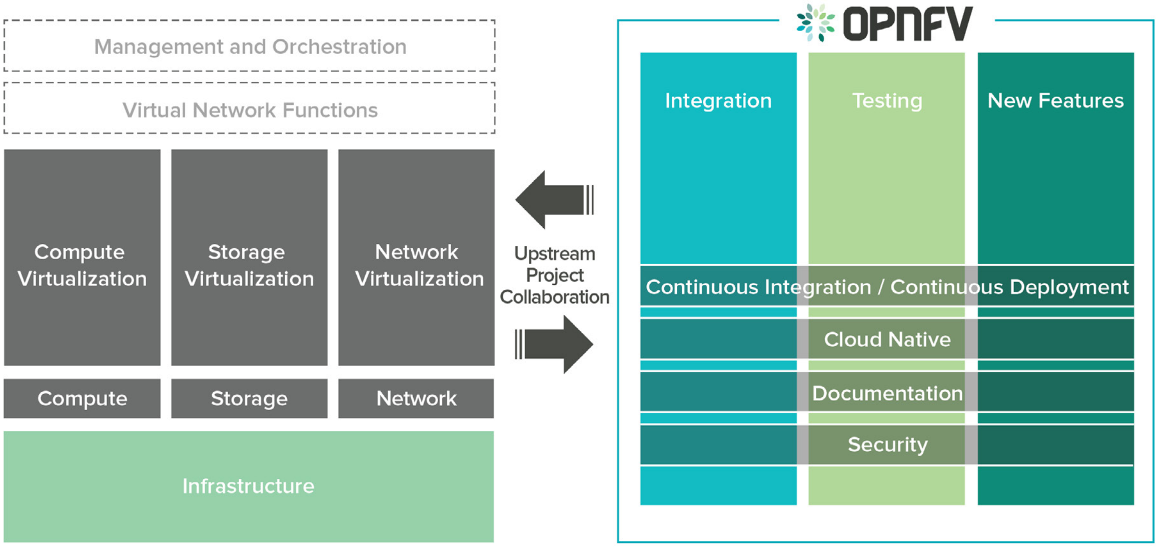
NFVBench: NFVbench is a lightweight end-to-end dataplane benchmarking framework project. It includes traffic generator(s) and measures a number of packet performance related metrics.

Lab as a Service (LaaS): LaaS is a “bare-metal cloud” hosting resource for the LFN community. This comprises 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 and scheduled in blocks of time, ensuring individual projects and users do not monopolize resources. By providing a lab environment to developers, LaaS enables more testing, faster development, and better collaboration between LFN projects.

CI/CD for Continuous Deployment and Testing of NFVI Stacks

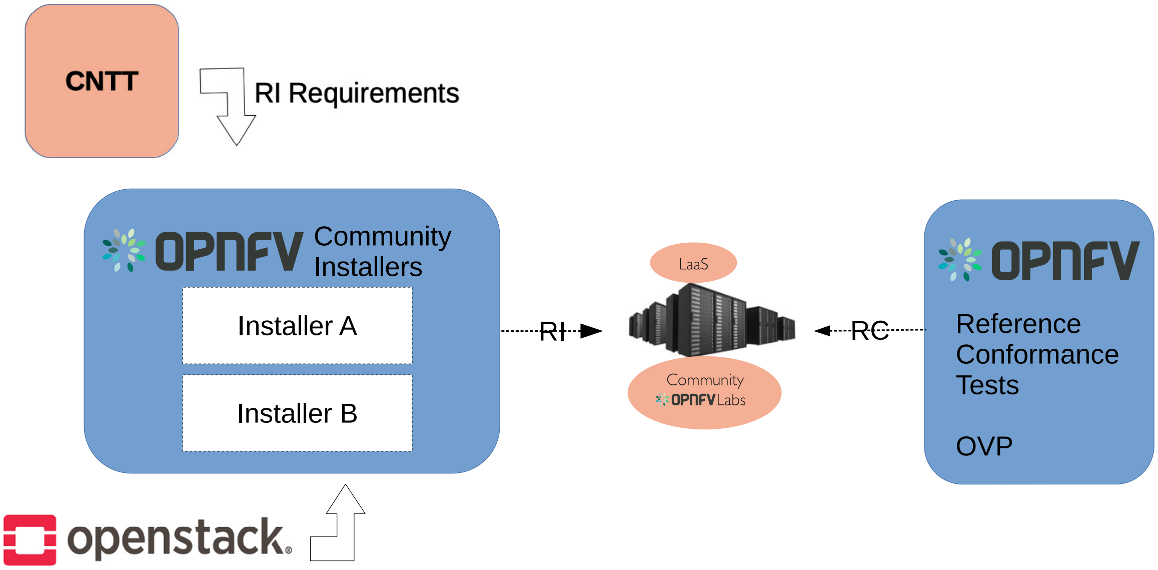
OPNFV Lab Infrastructure: OPNFV leverages globally distributed community labs provided by LFN member organizations. These labs are used by both developers of OPNFV projects as well as the extensive CI/CD tooling infrastructure to continuously deploy and test OPNFV reference stacks. In order to ensure a consistent environment across different labs, OPNFV community labs follow a lab specification (the Pharos spec) defining a high-level hardware configuration and network topology. In the context of CNTT reference implementations, all updates will be added to the Pharos spec in future releases.

Note: OPNFV **Feature projects** are working towards closing feature gaps in upstream open source communities providing the components for building full NFVI stacks, and OPNFV Deployment tools include Airship and Fuel / MCP.



The Scope of OPNFV

CNTT and OPNFV



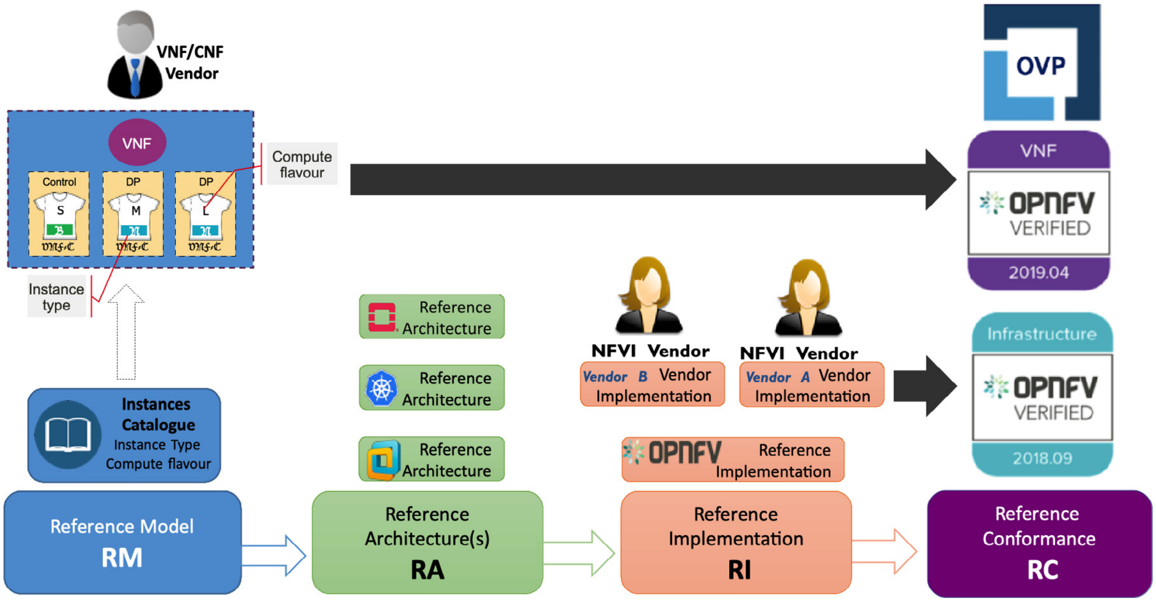
Relation of CNTT OpenStack RI and OPNFV

CNTT Introduction

The Common NFVI Telco Taskforce (CNTT) is a LFN and GSMA sponsored taskforce focused on minimizing the number of Network Function Virtualisation Infrastructure (NFVI) configurations used in Telco deployments. By reducing the variability in the NFVIs, network operators expect to reduce testing times and accelerate deployment of new capabilities. Network function providers should also appreciate economies of scale as fewer variants of the NFVI will be requested by the network operators. The set of standardized infrastructure profiles are optimized for common NFV and IT workloads as jointly defined by network operators and network function providers active in CNTT. (More information about CNTT can be found in CNTT GitHub).

CNTT components include:

- **Reference Model (RM):** describes how the infrastructure is exposed to a workload in a standard way.
- **Reference Architecture (RA):** OpenStack based and Kubernetes (K8s) based architectures to deliver a conformant infrastructure based on those technologies.
- **Reference Implementation(RI):** OpenStack and K8s based NFVI implementations used as the basis for testing and validation activities.
- **Reference Conformance (RC):** testharnesses used to verify the conformance of vendor implemented infrastructure to the CNTT specifications.



The Scope of CNTT

3.4 OpenDaylight

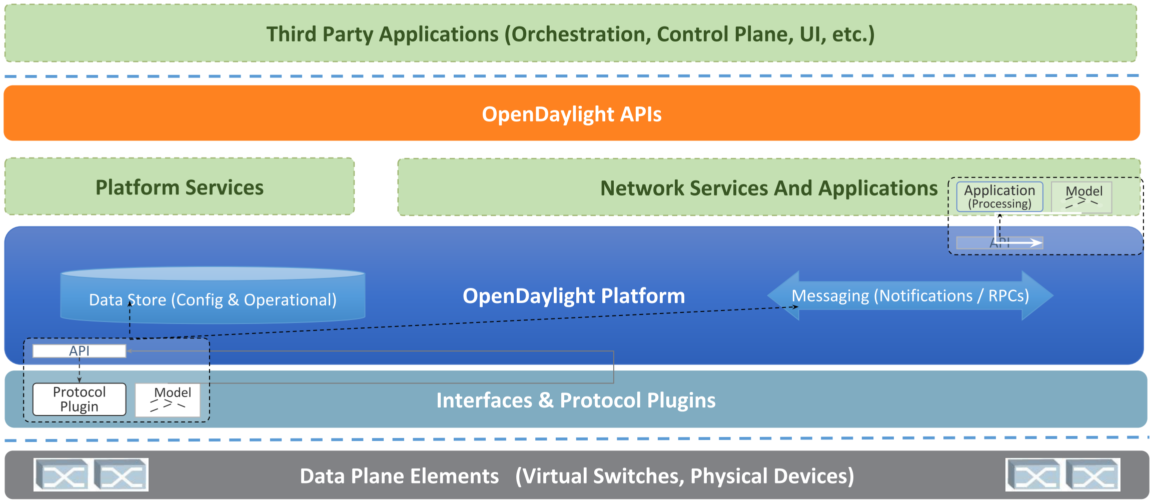
Introduction

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)

OpenDaylight Architectural View

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.

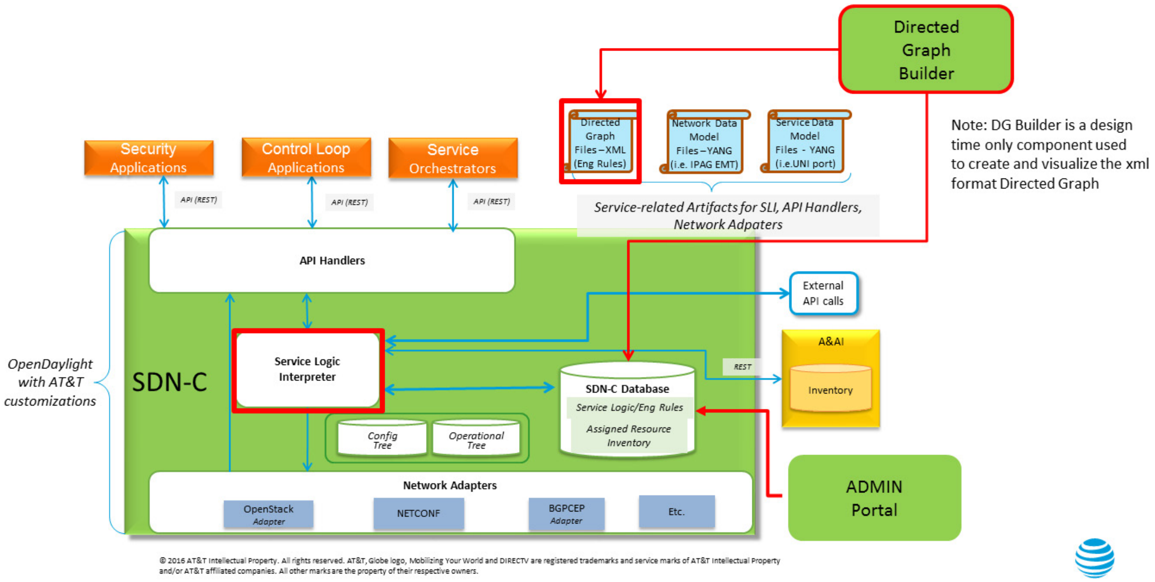
BOTTOMLINE ABOUT THE ARCHITECTURE
The modularity and flexibility of OpenDaylight allows end users to select whichever features matter to them and to create controllers that meet their individual needs.

Use Cases

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 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 Layer 4-7 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.


Directed
Graph
Builder
Note: DG Builder is a design time only component used to create and visualize the xml format Directed Graph
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 Adpaters
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
ADMIN Portal
Network Adapters
OpenStack Adapter
NETCONF
BGPCEP Adapter
Etc.
© 2016 AT&T Intellectual Property. All rights reserved. AT&T, Globe logo, Mobilizing Your World and DIRECTV are registered trademarks and service marks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks are the property of their respective owners.

How AT&T Uses OpenDaylight

EXTERNAL SDN CONTROLLER

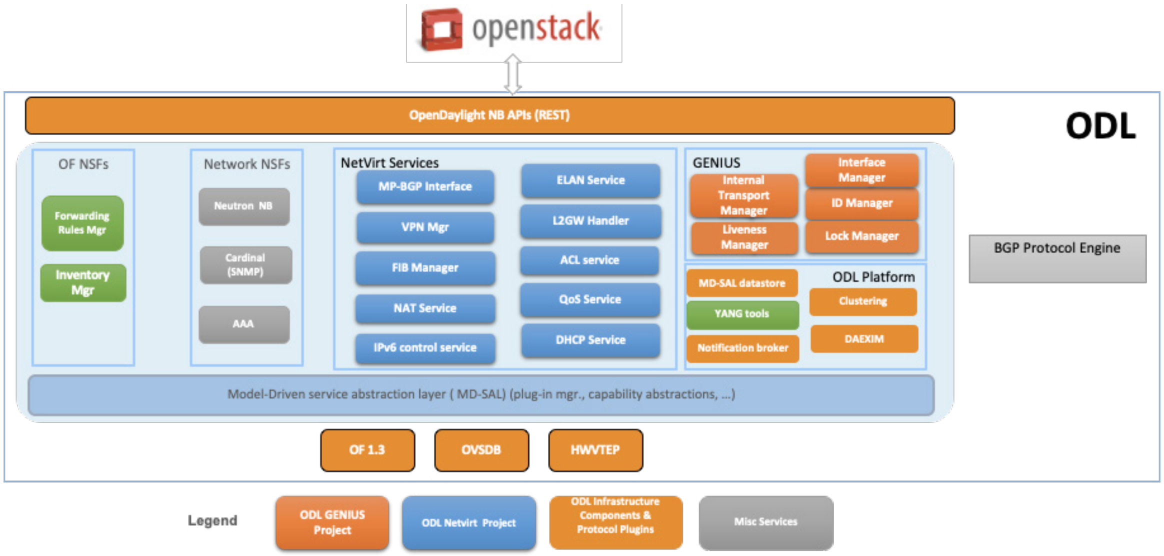
External SDN controller interfaces with the southbound interface of the ONAP SDN-C and is used to manage the Layer1-3 network devices in each network domain. Over the interface, the network configuration parameters extracted from the service logic are passed to the external SDN controllers. An OpenDaylight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

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 for OpenDaylight. The platform can and continues to be tailored to several other industry use cases.

3.5 OpenSwitch (OPX)

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, OPX 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 onboard open source networking OS applications. OPX provides a YANG based programmatic interface, that can be accessed using Python, thus providing an environment well-suited for DevOps.

OPX Features

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.
    - OPX 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 built 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 the 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 OPX 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.

OPX 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

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

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

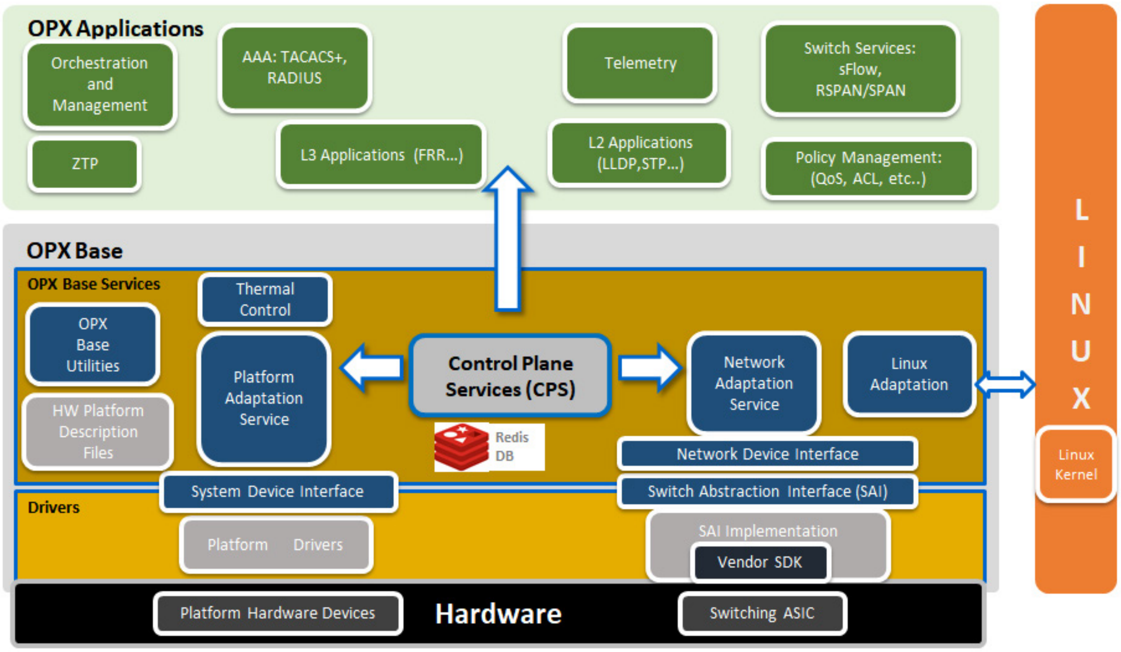
NORTH-BOUND PROGRAMMATIC INTERFACES

The OPX 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 OPX specific commands are available and can be invoked from a Linux shell (e.g. display the current software version, hardware inventory etc.).

OPX Architecture

The figure below illustrates the main areas of the OPX 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
L I N U X
Linux Kernel

OPX Architecture

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 OPX:

- 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 OPX. In a "disaggregated" model, users select applications to install them based on the requirements of a given network deployment.

In general, since OPX 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.

3.6 PNDA

Introduction

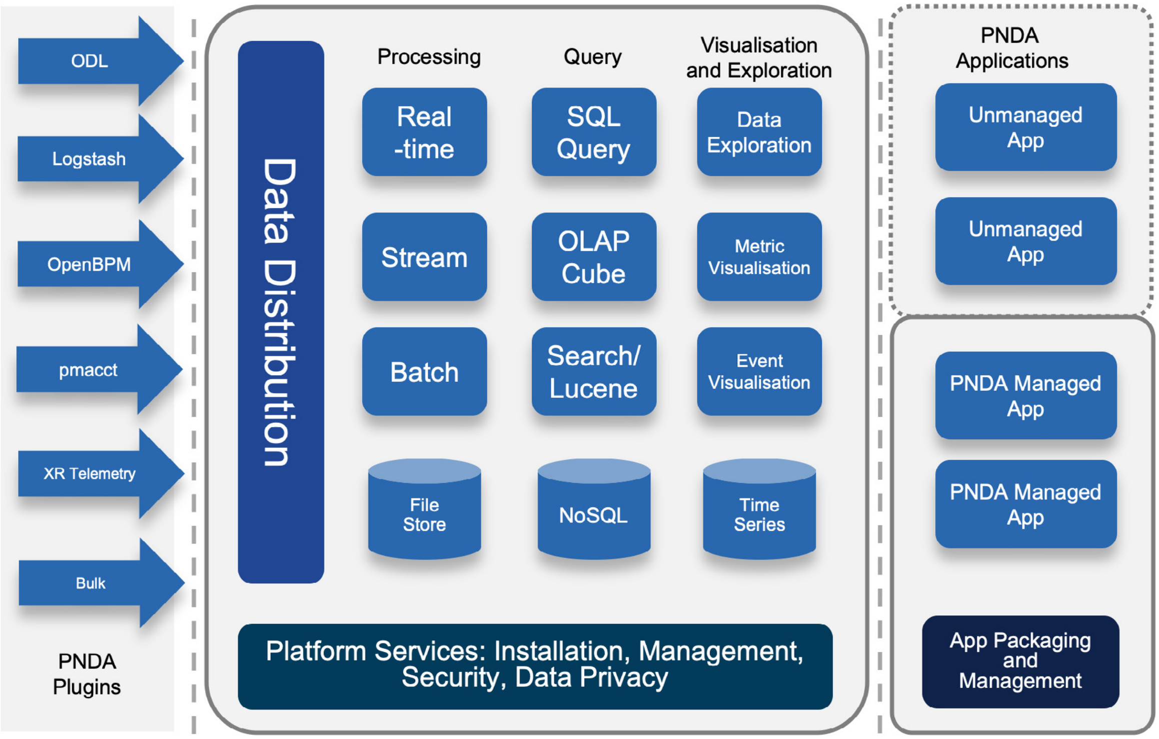
Innovation in the big data space is extremely rapid, but composing multitudes of technologies together into an end-to-end solution can be extremely complex and time-consuming. The vision of PNDA is to remove this complexity, and allow you to focus on your solution instead. PNDA is an integrated big data 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 PNDA project aims to deliver a fully cloud native PNDA data platform on Kubernetes. The current focus has been migrating to a containerized and helm orchestrated set of components, which has simplified PNDA development and deployment as well as lowered project maintenance costs. The goal of 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:

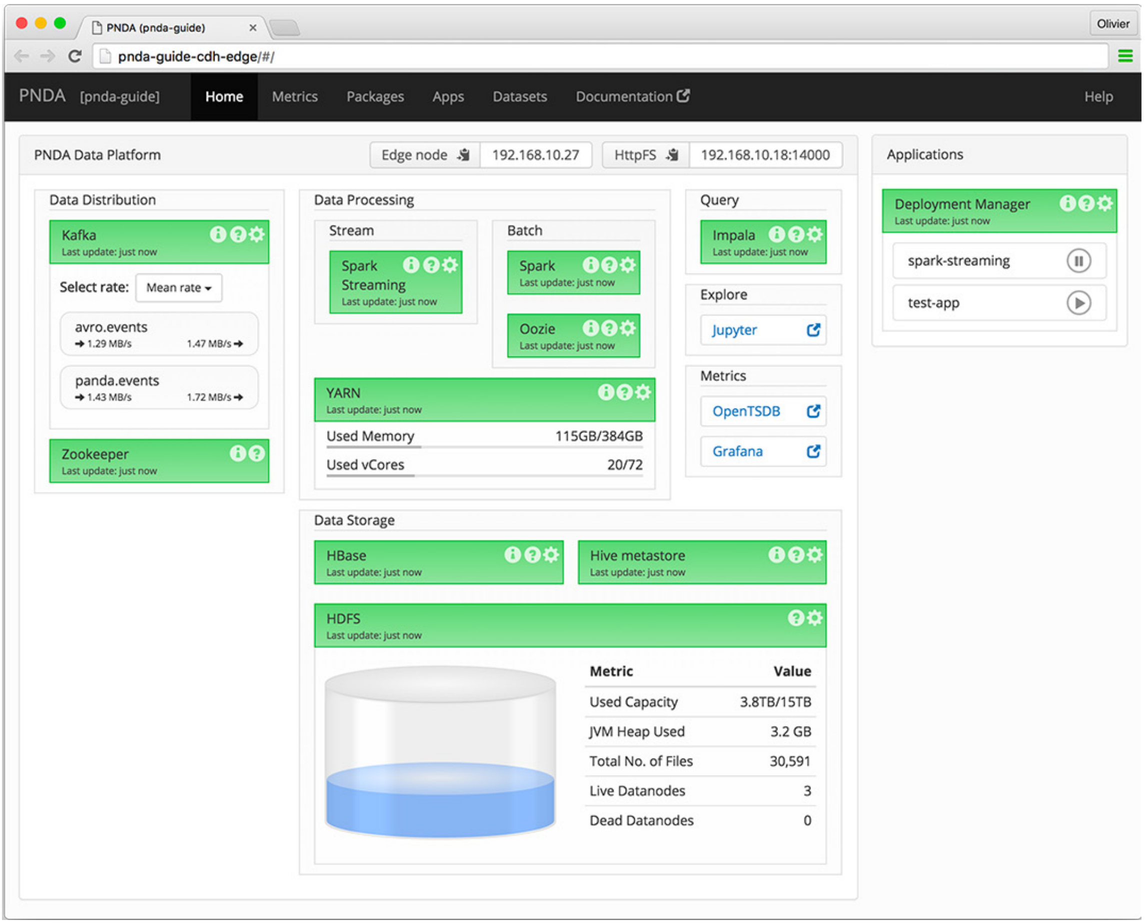
- Aggregate data like logs, metrics, and network telemetry
- Scale up to consume millions of messages per second
- Efficiently distribute data with publish and subscribe models
- Process bulk data in batches, or streaming data in real time
- Manage lifecycle of applications that process and analyze data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



Edge node 192.168.10.27

HttpFS 192.168.10.18:14000

Select rate: Mean rate

avro.events → 1.29 MB/s 1.47 MB/s →

panda.events → 1.43 MB/s 1.72 MB/s →

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 SNAS

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.

The project is currently be re-architectured by a team at UCSD with more information expected in 2H2020. Learn more.

3.8 Tungsten Fabric

Introduction

Tungsten Fabric is a software-defined network and security fabric built for rapid deployment at scale. It 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.

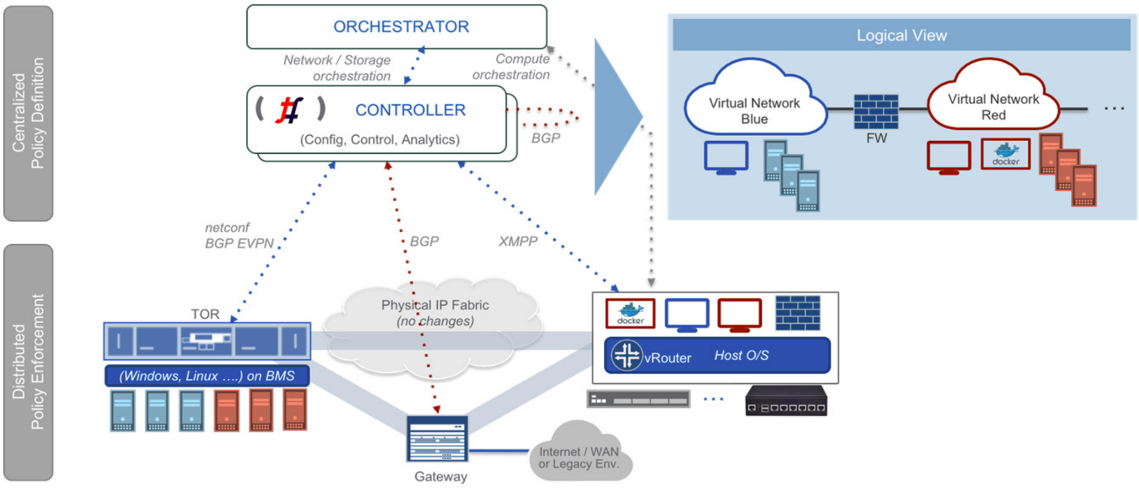
Tungsten Fabric enables usage of the same controller and forwarding components for every deployment, providing a consistent interface for managing connectivity in all the environments it supports, and is able to provide seamless connectivity between workloads managed by different orchestrators, whether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

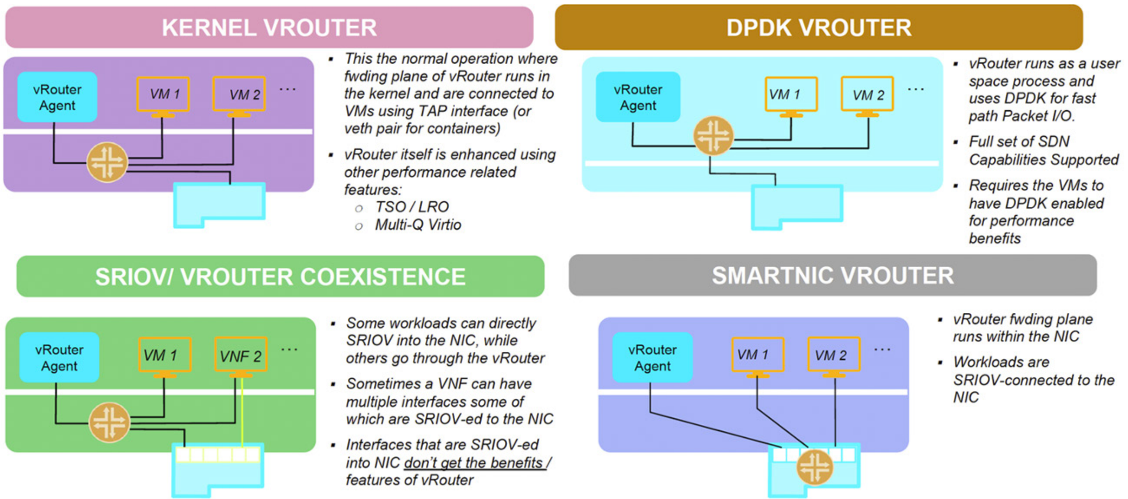
- Tungsten Fabric Controller – a set of software services that maintains a model of networks and network policies, typically running on several servers for high availability
- Tungsten Fabric vRouter – installed in each host that runs workloads (virtual machines or containers), the vRouter performs packet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


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

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)

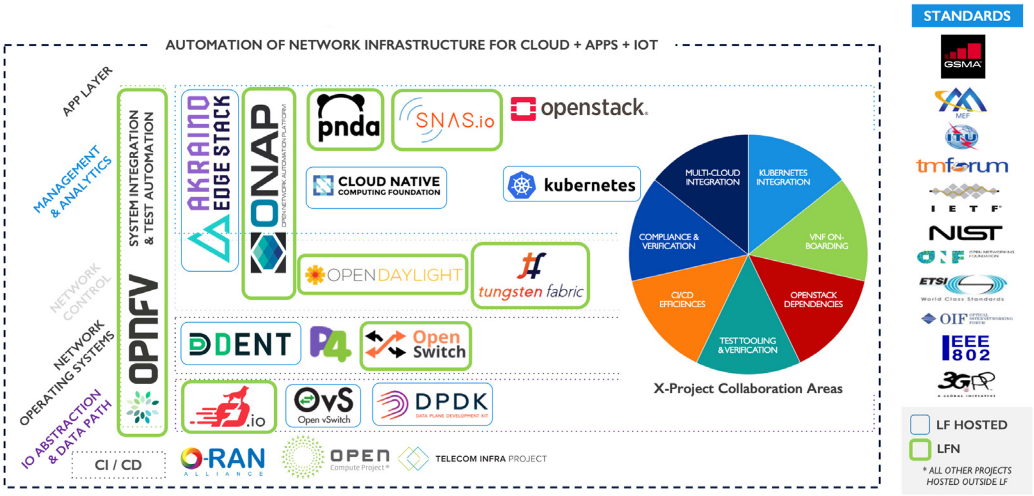
Key Features

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

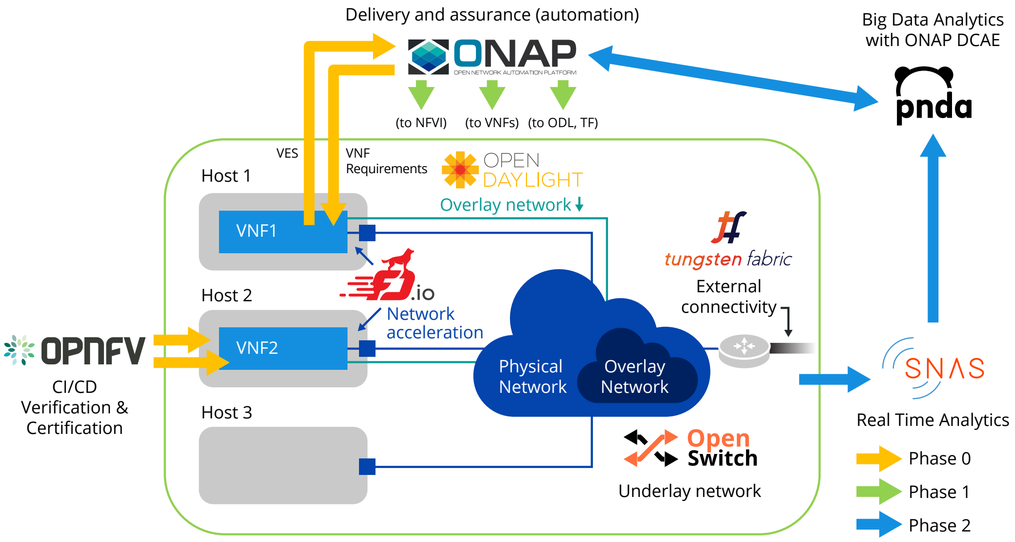
4. LFN Integration Points

As highlighted in the previous chapters, LFN projects are designed to be part of end-to-end modern networks. As such, many LFN projects have integration points with other LFN projects, as well as external open source projects. Here is a look at the LFN projects (highlighted in green) in the open source networking stack.


AUTOMATION OF NETWORK INFRASTRUCTURE FOR CLOUD + APPS + IOT
APP LAYER
MANAGEMENT & ANALYTICS
NETWORK CONTROL
NETWORK OPERATING SYSTEMS
IO ABSTRACTION & DATA PATH
CI / CD
SYSTEM INTEGRATION & TEST AUTOMATION
OPNFV
AKRAINO EDGE STACK
ONAP
OPEN NETWORK AUTOMATION PLATFORM
pnda
SNAS.io
openstack
CLOUD NATIVE COMPUTING FOUNDATION
kubernetes
OPENDAYLIGHT
tungsten fabric
DENT
P4
Open Switch
FD.io
OvS Open vSwitch
DPDK
DATA PLANE DEVELOPMENT KIT
O-RAN ALLIANCE
OPEN Compute Project
TELECOM INFRA PROJECT
MULTI-CLOUD INTEGRATION
KUBERNETES INTEGRATION
VNF ON-BOARDING
OPENSTACK DEPENDENCIES
TEST TOOLING & VERIFICATION
CI/CD EFFICIENCES
COMPLIANCE & VERIFICATION
X-Project Collaboration Areas
STANDARDS
GSMA
MEF
ITU
tmforum
IETF
NIST
ONF OPEN NETWORKING FOUNDATION
ETSI World Class Standards
OIF
IEEE 802
3GPP
LF HOSTED
LFN
* ALL OTHER PROJECTS HOSTED OUTSIDE LF

2020 LF Networking / SDO Landscape

This section aims to present an end-to-end use case example where the LFN projects work in harmony to deliver a "service" that includes VNFs, connectivity and analytics-powered assurance as shown in the following picture:



In this example, 2 VNFs (for the sake of simplicity, provided by the same vendor) must be deployed on top of an NFVI (e.g., OpenStack), be interconnected and provided with external connectivity to the Internet. Moreover, the VNFs require network acceleration and the whole service must be assured using Analytics driven closed loop operations.

Using the 8 LFN projects, an end user (e.g., a carrier) can realize the above as follows:

Phase 0 - Building the network infrastructure and preparing the network functions

Following the CNTT Reference model, the operator decides which CNTT OpenStack based Reference Architecture may best suit its needs. This is followed by picking a set of infrastructure components that fit a CNTT Reference Implementation of choice. The infrastructure is built using the deployment tolls and CI/CD provided by OPNFV. Next, the infrastructure is certified using the CNTT RC and OPNFV CVC.

Several LFN projects may be used as infrastructure building blocks for addressing the needs of network functions, such as high throughput/low latency networking:

- **OpenDaylight** and **Tungsten Fabric** can be used as 3rd party SDN solutions to provide network connectivity.
- **Open Switch (OPX)** can be used to configure the physical (underlay) network that connects the physical hosts used to deploy OpenStack The network topology may follow the leaf and spine topology as a physical infrastructure is recommended in the requirements of physical infrastructure of the CNTT Reference Architecture.
- **FD.io** provides data plane network acceleration through its Vector Packet Processor (VPP).

VNFs are prepared for deployment and inclusion in network services:

- An NFV vendor pre-validates and certifies a couple of VNFs (i.e., VNF1 and VNF2) through the **OPNFV** Verification Program (OVP).
- The NFV vendor ensures that the VNF complies with the ONAP VNF requirements. This will enable ONAP to properly control the lifecycle of the VNF as part of a network service.

Phase 1 - Network service design and deployment

At design time, ONAP is used to onboard the VNFs that are compliant with the ONAP requirements and pre-certified using the OPNFV CVC . Those compliant resources can later be used to design any E2E service using **ONAP** Service Design and Creation (ONAP SDC).

At runtime, ONAP orchestrates the deployment of the whole service either through ONAP internal functions/components or leveraging the capability to interwork with 3rd party components.

In particular, ONAP Service Orchestrator (ONAP SO) instructs the underlying ONAP functions in order to deploy all of the elements that compose the end-to-end service.

ONAP deploys the VNFs in the available NFVI and the overlay network connecting them using ONAP SDN-C. SDN-C uses its **OpenDaylight** based architecture to model and deploy the L1-L3 network. Next the ONAP APP-C is used to configure the network functions and their L4-L7 functionality. This is also done leveraging the **OpenDaylight** architecture.

OpenDaylight may be used to stitch together the physical switch fabric of the infrastructure with the virtual networking in the NFVI (e.g. OpenStack Neutron). Through the **OpenDaylight** Northbound Interface, ONAP-SDNC is able to instruct the OpenDaylight SDN controller for underlay network management. The southbound interfaces (e.g. NETCONF, etc) support interactions with OpenSwitch running on the leaf and spine fabric switches in the NFVI.

By leveraging its SDN-C southbound interface, ONAP instructs **Tungsten Fabric** to create the external connectivity that will enable customers to "consume" the services offered by the VNFs. The predefined policies that will control the lifecycle of the network service are designed using the ONAP design-time components such as SDC and CLAMP.

Phase 2 - Network service operation

Naturally, being a Network Automation Platform, **ONAP** plays a central role in the delivery and assurance of the service. The VNFs report their performance and fault data to the ONAP DCAE using the VNF Event Streaming (VES) interface. This information is constantly analyzed and may trigger predefined policies that were created at design-time. The policies are used to invoke closed loop automation actions such as scaling and healing of service components in order to assure the required SLA and respond to changing demands and network conditions.

Finally, closed loop operations may be further enriched by combining LFN Real Time Analytics capabilities of **SNAS.io** and the synergies offered by ONAP and **PNDA.io**. Information about changes in the network topology gathered by **SNAS** can be used to trigger **ONAP** policies that will spawn more instances of packet routing network functions. The data analytics capabilities of **PNDA** may be used to trigger **ONAP** policies based on data streams produced by all layers of the infrastructure as well as the network functions. **ONAP** may respond to an infrastructure issue detected by **PNDA** by migrating VNFs from an affected location to one that is healthy and has the available resources.

5. Conclusion, Call for Action, and Further Reading

The use cases and integrations described in this whitepaper are just a few of many possible synergies. LFN projects may be integrated with one another in many other ways. They may also become part of a larger solution, involving other open source projects and commercial products. The set of projects under the LFN should not be considered a monolithic bunch. Any network designer may pick just one, several, or all the projects when designing their network. The use of well defined interfaces makes it easier to integrate the projects with any other system.

The LFN community is eager to learn about new use cases that might stem from reading this document. We encourage readers who come up with ideas for use cases to share them with the community using the LFN Technical Advisory Committee (TAC) mailing list at:

https://lists.lfnetworking.org/g/lfn-tac

The best way to learn about an open source community is to participate and contribute. Learn about getting started on the LFN website. For further reading, please refer to the Wikis and documentation links below:

LF Networking

Wiki: https://wiki.lfnetworking.org/

ONAP

Wiki: https://wiki.onap.org/

Docs: https://docs.onap.org/

FD.io

Wiki: https://wiki.fd.io/view/Main_Page

Docs: https://fd.io/documentation/

OpenDaylight

Wiki: https://wiki.lfnetworking.org/display/ODL/

Docs: https://docs.opendaylight.org

OPNFV/CNTT

OPNFV Wiki: https://wiki.opnfv.org/

OPNFV Docs: https://docs.opnfv.org

CNTT Wiki: https://wiki.lfnetworking.org/display/LN/Common+NFVI+Telco+Task+Force+-+CNTT

CNTT Docs: https://github.com/cntt-n/CNTT

OpenSwitch (OPX)

Wiki / Docs: https://github.com/open-switch/opx-docs/wiki

Docs: https://github.com/open-switch/opx-docs/wiki

PNDA

Wiki: https://wiki.pnda.io/

Docs: http://pnda.io/guide

SNAS

Docs: https://www.snas.io/docs

Tungsten Fabric

Wiki: https://wiki.tungsten.io/

Docs: https://tungstenfabric.github.io/website/

6. Glossary

Term	Description
API	Application Programmable Interface
BSS	Business Support System
CI/CD	Continuous Integration/Continuous Deployment
CNF	Cloud-native Network Function
CP	Control plane
CSP	Communications Service Provider
CT	Communication Technology
DP	Data Plane
IT	Information Technology
K8S	Kubernetes
LCM	Life Cycle Management
LFN	LF Networking umbrella project
NEP	Network Equipment Provider
NETCONF	Network Configuration Protocol
NF	Network Function
NFVi (or NFVI)	Network Function Virtualization Infrastructure
O&M	Operations and Maintenance
OSS	Operations Support System
SDN	Software Defined Networking
SDO	Standards Definition Organization
TOSCA	Topology and Orchestration Specification for Cloud Applications (OASIS standard)
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
YANG	Yet Another Next generation (NETCONF modeling language)

Credits

This paper was made possible by the following community contributors. Thank you.

Ranny Haiby, *Samsung* (lead)

Abhijit Kumbhare, *Ericsson*

Chaker Al-Hakim, *Futurewei*

Jason Hunt, *IBM*

Ligli Deng, *China Mobile*

Christian Olrog, *Ericsson*

David Cherubini, *Vodafone*

Tom Nadeau, *Red Hat*

Mike Lazar, *Dell EMC*

Prabhjot Singh Sethi, *ATS*

Laurence Plant, *IBM* (reviewer)

Donald Hunter, *Cisco*

Rabi Abdel, *Vodafone*

Hiroshi Dempo, *NEC*

Al Morten, *AT&T*