Issue 0.1 | 15 March 2019

Alianza NOS Cookbook

Document Number AKM-003

ALIANZA

alianza.com

Notices

http://www.alianza.com

Table of contents

Introduction

Building a network operating system (NOS) is a significant undertaking for any organization, requiring significant amounts of time and money to complete. Until recently, creating a NOS was the preserve of OEMs and an essential part of their networking solutions. In the last few years a few new disaggregated NOS suppliers (such as Cumulus and SnapRoute) have emerged, providing independent NOS solutions for whitebox hardware. A variety of open source NOSs have also been created (such as SONiC, OpenSwitch and DaNOS), providing operators with a low-cost option for network solutions.

The emergence of cheap but powerful whitebox and britebox hardware coupled with a third-party network operating system represents a challenge for traditional OEMs who could struggle to compete on price and openness. However, these new solutions are mostly limited to data center use cases today. This provides an opportunity for OEMs, and others, to create network operating systems targeting more complex use cases and working on both proprietary and whitebox hardware.

This cookbook will show how this can be achieved quickly using a combination of OCP technologies, open source software and, where appropriate, commercial solutions. We welcome feedback from parties with experience building a NOS using OCP and other open source technologies.

1 Planning your NOS

In this section we will examine the motivations for building a network operating system and the steps that need to be taken before starting.

It is only by completing thorough and comprehensive plans and designs for your NOS that the end result will meet your requirements, especially if they include portability to different types of hardware.

1.1 Why build a NOS?

For many years, operators have used disaggregation and open source software to drive down costs. They have reduced CapEx costs through the use of cheaper hardware and software and OpEx costs through the use of automation, orchestration and telemetry applications. This trend started with compute servers and more recently has moved to network devices in the data center – in particular, Top of Rack and leaf/spine switches.

Until now, these savings have been unavailable to traditional service providers. Their more complex requirements mean their networks are dominated by monolithic devices using proprietary hardware and software from the likes of Cisco, Juniper, Huawei, and a host of smaller OEMs.

The rollout of new 5G networks promises to change that. The cost of building and running new networks to support 5G technology will be massive, so service providers will take this opportunity to look for innovative ways to make savings. Replacing monolithic devices with whitebox hardware and disaggregated software will be one of the main ways to achieve that.

Suppliers of network solutions can take advantage of this opportunity by building a network operating system that supports both their powerful proprietary hardware and whitebox alternatives. By also exposing modern open interfaces such as NETCONF, the NOS can be easily integrated with custom and third-party applications in customer networks. Moreover, such a NOS can be easily upgraded with new capabilities and ported to new platforms, helping to address new use cases and remain at the cutting edge of this fast-moving industry.

1.2 Use Cases

Data center operators have already adopted white box hardware and disaggregated software solutions for their simpler network devices such as top-of-rack switches and network operating systems, of varying capabilities, are widely available for these use cases.

This cookbook focuses instead on networking use cases outside of the data center, in service provider networks. In particular, access and aggregation routers.

These types routers have similar requirements for their operating systems:

- Open management interfaces to enable automation, orchestration, telemetry etc. as well as custom applications.
- Comprehensive control plane support including Layer 2, Layer 3, IP/MPLS, OAM, Traffic Engineering, etc.
- Carrier Grade Reliability with both control plane and data plane redundancy and support for features such as in-service software upgrade.

It is likely that new deployments in such networks are also going to be looking for support for features such as network slicing and containerization. Supporting these sorts of features will require your NOS to be architected to support them from the outset. For example, you could choose to install each of the base OS, control plane and services layer, and management layer in separate containers, facilitating support for independent upgrade of each of those layers.

Network devices targeted at specific use cases have different feature set requirements. The following table summarizes the high-level requirements for some common use cases.

Requirement \ Use Case	DC Switch	DC Interconnect	Cell-Site Gateway	Access / Aggregation	Core Router
L2 Protocols	✓	✓	✓	✓	✓
BGP / IGP	✓	✓	✓	✓	✓
EVPN / VxLAN	✓	✓	✓	✓	✓
MPLS Protocols	✗	✓	✓	✓	✓

QoS Features	✗	✗	✓	✓	✓
High Availability	✗	✗	✗	✓	✓

As can be seen from this table, the complexity of network devices increases as they move closer to the center of a service provider network.

1.3 Open Source vs Commercial vs BYO

The base of any NOS is likely to be Linux, which is open source and freely available. The investment to create an entirely new base OS to replace Linux is unlikely to be justified!

However, for the management and protocol software, there are three high-level options. Choosing the right one is important, because once you've shipped a first version of a product, making significant changes to the user experience can be disastrous.

1.3.1 Build it yourself (BYO)

20 years ago this was the standard OEM approach: create the function you need from scratch using an in-house software development team.

For complex protocols such as OSPF, BGP and MPLS, this approach is now almost never taken.

- Protocol software is incredibly complex, and even a bare-bones implementation of (for example) BGP would be many developer-years of effort.
- Protocol software typically has to interoperate with many other implementations in many different network scenarios. Testing them all up-front is never feasible - it takes elapsed years of trials and lab deployments (and real deployments) before you have something that a carrier will accept into their network.
- Over the years, a vast number of bolt-on features have been added to the protocols. Many of them are considered table-stakes by carriers, and the development effort required to implement even a subset of them is always vastly underestimated.

For these reasons, developing new protocols in place of existing open source or commercial options doesn't make sense in today's world.

The same logic applies to management agents: writing a new one from scratch is unlikely to ever be a viable business model in this day and age, although some OEMs will have sufficiently portable existing implementations that they can re-purpose for a new NOS.

1.3.2 Open Source

There are open source protocol implementations (such as Quagga, BIRD, and FRR) and open source management agents such as OpenYuma. Starting from one of these has advantages over BYO, particularly in terms of time-to-market and short-term cost:

- The implementations already have some level of maturity - fewer bugs, better interoperability.
- You can hire engineers who already understand them.
- You may be able to take advantage of new features that the community contributes over time.

However, there are downsides, particularly for protocol software:

- People rarely contribute back to the community, so the fact that other vendors have deployed products based on open source doesn’t mean you’ll have an easy time doing the same.
- Feature support tends to be minimal for the same reason.
- It tends to be low in scalability, availability, supportability and general quality.
- Without sufficient adoption and commercial backing, the community moves on and you can find yourself locked in to a dead-end product that ends up being as expensive as writing it yourself.

Open source can work for a tightly scoped deployment scenario, but it is unlikely to pay off (versus a commercial alternative) for a product you expect to be selling for many years and want to apply to future platforms.

1.3.3 Commercial

The commercial protocol software model is that a vendor implements a protocol such as BGP and then licenses the source code to many NOS implementers. The result is that the software vendor has enough commercial return to continue to significantly invest in the protocol product (in terms of features and quality), while each NOS implementer only pays a fraction of that overall development cost.

As a result, commercial software tends to be much higher in quality and functionality, and the vendor typically has a strong roadmap for the newest features.

It also typically comes with a support and maintenance contract for fixes and small features, which costs much less than you would have to pay your own engineers to maintain an open source-based solution.

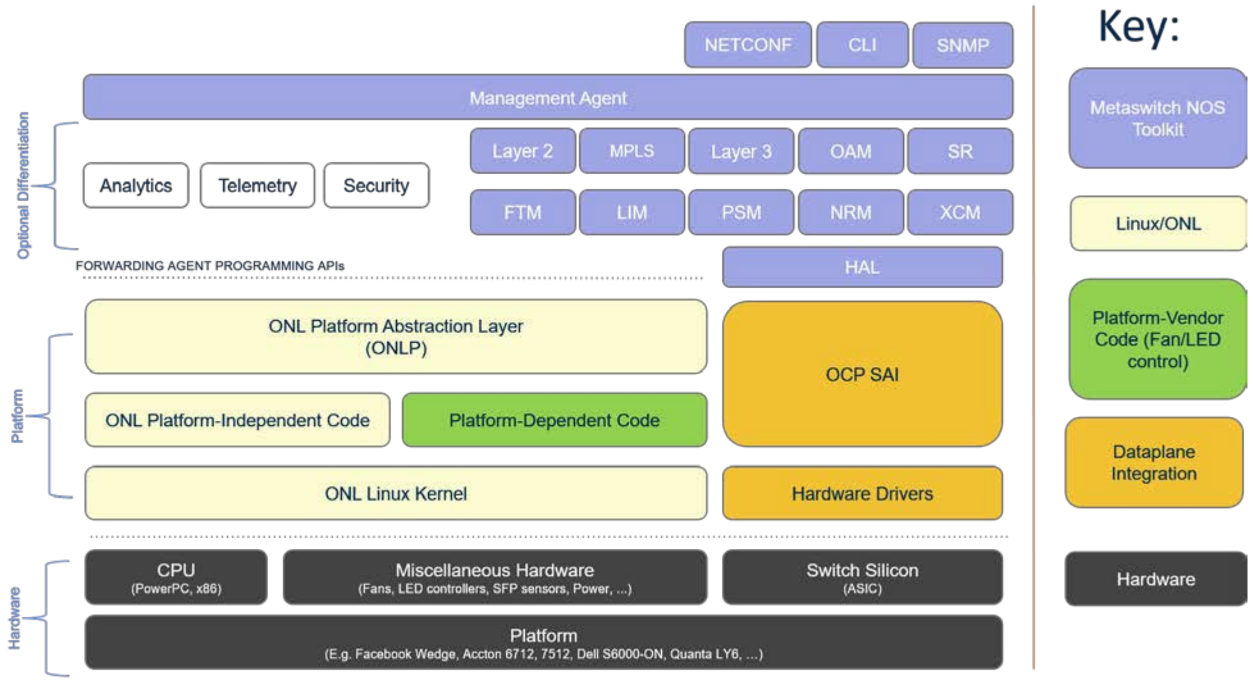
In addition, when building either highly resilient platforms (such as a chassis-based system with multiple process cards) or high-function platforms (such as VPN services over MPLS with sub-50ms packet loss on link failure), commercial software is often the only viable choice - re-engineering open source solutions to support these use cases is incredibly expensive.

The upfront cost of commercial software may of course be a reason to prefer other approaches! However, some vendors now offer subscription-based or revenue-share agreements, making it more attractive.

1.4 Architecture

The key to a successful NOS architecture is taking a rigorous approach so that functionality is implemented in components with well-defined responsibilities, separated by clearly defined interfaces.

The following diagram shows the high-level NOS architecture that Alianza has developed to meet the unique requirements of service provider networks.



The following sections describe each of the parts of this NOS architecture in more detail.

1.5 Operating System Platform

Before the Open Network Install Environment (ONIE) and Open Network Linux (ONL), creators of network operating systems had little choice but to build their own Linux distribution from scratch, including interfaces to all of the hardware components such as fans and LEDs. ONL has been validated on at least 68 open networking platforms across 11 whitebox or britebox suppliers. Today, the combination of ONIE and ONL means that it is now quite a simple task to build the basis of a NOS that will be portable across many hardware systems.

OEMs who have previously built their NOS independently of ONL may choose to continue with their own version of Linux; however, they will have to absorb the cost of porting and validating that NOS on each new piece of target hardware.

1.6 Data Plane Interface

Historically, OEMs creating a NOS have had to interface directly with the SDK provided by the vendor of the switch chipset they selected for their device. This made porting that NOS to different chipsets, sometimes even ones from the same vendor, quite difficult. These SDKs have also only been made available to customers of the chipset vendors, presenting a barrier to third-party innovation.

A number of recent initiatives are attempting to change that. For example: SAI, OF-DPA, OpenNSL and OF-DPA.

1.7 Services Layer

The services layer of the NOS:

- holds state for the control plane in databases such as the RIB and LDB
- receives instructions from the control plane and translates those instructions into the appropriate programming of both Linux and the dataplane
- presents a common interface to the control plane protocols.

There is very limited support for these services in open source software, although some of the open source NOSs such as SONiC and OpenSwitch do include implementations of some of them. Consequently, when planning to build a NOS, you need to decide whether to build these services yourself or purchase a license for a commercial implementation.

The exact set of services required will depend upon the protocols that need to be supported, but include the following functional areas.

1.7.1 Local Interface Manager

The Local Interface Manager (LIM) is responsible for managing interfaces and ports on the local device, ensuring that both Linux and the switch silicon are configured correctly and are in sync with each other. It then advertises the configured ports and interfaces to the control plane layer. LIM also programs LACP bundles, IRB interfaces and MPLS tunnel interfaces.

1.7.2 Forwarding Table Manager

The Forwarding Table Manager (FTM) is told about active routes by the control plane layer and maintains the state of the data plane. It manages the layer 3 hardware routing table and programs the Linux forwarding information base (FIB). FTM also receives packets from unknown IP hosts on attached subnets and creates corresponding entries in the hosts table.

1.7.3 Cross-Connect Manager

The Cross-Connect Manager (XCM) takes information on active MPLS cross-connects from the control plane and maintains the data-plane state to represent this programming. It also programmes MPLS forwarding rules into the data plane, manages resource reservations for LSPs, and initiates protection switches for cross-connects and pseudowires.

1.7.4 Path Set Manager

The Path Set Manager (PSM) is responsible for programming next hops and ECMPs corresponding to path sets used by routes and cross-connects.

1.7.5 Neighbor Resolution Manager

The Neighbor Resolution Manager (NRM) is responsible for resolving IP addresses into Neighbor objects and for programming the resulting Neighbor objects in the data plane.

1.8 Control Plane Layer

The control plane layer is where implementations of the different protocols supported by the NOS live.

Different network device use cases have varying needs for control plane protocols. The following table lists the main control plane protocols and the use cases for which they are essential. Note that many devices targeting one of these use cases support more, sometimes many more, protocols than are listed here.

Protocol	DC Switch	DCI	CSG	Access	Core	FOSS Available?
Layer 2 Protocols						
LACP	✓	✓	✓	✓	✓	✓
ECMP	✓	✓	✓	✓	✓	
STP				✓	✓	
SyncE		✓	✓	✓	✓	✗
EVPN		✓	✓	✓	✓	✓
L2VPN	✗		✓	✓	✓	✗
Layer 3 Protocols						
BGP	✓	✓	✓	✓	✓	✓
ISIS	✓	✓	✓	✓	✓	✗
OSPF	✓	✓	✓	✓	✓	✓
PIM				✓	✓	
IGMP				✓	✓	
VRRP	✓	✓	✓	✓	✓	✓
L3VPN			✓	✓	✓	✗
MPLS Protocols						
IP/MPLS		?	✓	✓	✓	✗
LDP			✓	✓	✓	✗
RSVP			✓	✓	✓	✗
LMP			✓	✓	✓	✗

OAM						
BFD	✓		✓	✓	✓	
LSP-Ping			✓	✓	✓	
Y.1731			✓	✓	✓	✗

As can be seen from this table, it is difficult to fully implement a network device with only open source control plane protocols. As with the services layer, you will likely need to consider in-house or commercial implementations of at least some of the control plane protocols for the use cases you need to support.

1.9 Management Layer

In the past most network devices were configured using a command line interface and, possibly, SNMP. In today's world, operators expect their network elements to be as programmable as any other device in their networks and many technologies have emerged to help automate the configuration and management of those devices.

Providing NETCONF and RESTCONF interfaces is therefore increasingly seen as essential for all network devices. However, operators still expect fully featured CLI and SNMP support as well (although demand appears to be starting to decrease).

The best approach for supporting all of these interface types is to use a NETCONF server, for which there are open source options, with the interface definitions provided by YANG models. This approach will allow your NOS to support all of the interface types above and be easily extensible. In addition, standard YANG models for all of the main control plane protocols are expected to emerge in the next few years - adopting these will ensure your device confirms with industry expectations.

Importantly, carriers expect a single consistent transactional interface for configuration of the entire device. It would not be acceptable to have to (for example) log into different CLI sessions for different functions.

1.10 Conclusion

In this section of the NOS cookbook we have described the key elements and architecture of a modern, portable network operating system. We have also shown how many of these elements can utilize open source software. However, there remain a number of areas where it is necessary to either use commercial software or to build your own, or a combination of the two, in order to build a device that meets the demands of service providers.

In the next section we will explain the steps required to build a basic NOS using open source software and the further work required for a viable product.

2 Building a NOS with OCP Technology

In the second part of the NOS Cookbook we look at how to use the advice in Part 1 to build the basic components of a simple NOS and highlight the steps remaining to reach a functioning network operating system.

2.1 OCP & Open Source Options

Today there are many open source projects that provide various building blocks of a network operating system. Many of those projects are part of OCP.

2.1.1 Operating System Platform

Open Network Linux (ONL) should be the default choice of operating system platform for any modern network operating system. ONL uses the Open Network Install Environment (ONIE) for installation and is part of the Open Compute Project (OCP).

A key advantage of ONL is the existence of the Open Network Linux Platform (ONLP) APIs which provide a standardized abstraction interface for the management of components such as LEDs, fans, power supplies, SFP/SFP+/QSFP and temperature sensors. Every manufacturer producing ONL-compatible hardware supplies the drivers to integrate their devices with ONLP, making it reasonably simple to create a portable management interface for this part of the network operating system.

2.1.2 Data Plane Integration

As described in Part 1, there are multiple choices for the layer that integrates with the vendor switch silicon. However, only one of those is supported by multiple vendors - the OCP Switch Abstraction Interface (SAI). Primarily for that reason, SAI is the favoured choice for a portable NOS. However, SAI currently has limited support for more advanced features, like tunneling, required by protocols such as MPLS.

Alternatives like OpenNSL do support those features, but currently only for the Broadcom XGS switch silicon family.

The only other alternative is to write direct to vendor SDKs, which has a substantial cost in both difficulty and portability.

2.1.3 Services Layer

The services layer has few open source options, although some of the forwarding agents referred to in the next section include some limited services layer functionality.

2.1.4 Control Plane Layer

There are a number of different options for open source control planes, including:

- Quagga - the most well known of the open source control planes, it supports OSPF, ISIS and BGP.
- FRRouting - an evolution of Quagga, supporting BGP, OSPF, ISIS, PIM, BFD and LDP.
- BIRD - a separate project that supports BGP, OSPF and BFD.

We are not aware of mature open source options for Layer 2 function (such as Multiple Spanning Tree Protocol (MSTP), Link Aggregation Control Protocol (LACP), and Ethernet Ring Protection System (ERPS)) or OAM (such as the Y.1731 protocols).

2.1.5 Management Layer

The open source control plane options described above all provide a basic CLI for management and configuration.

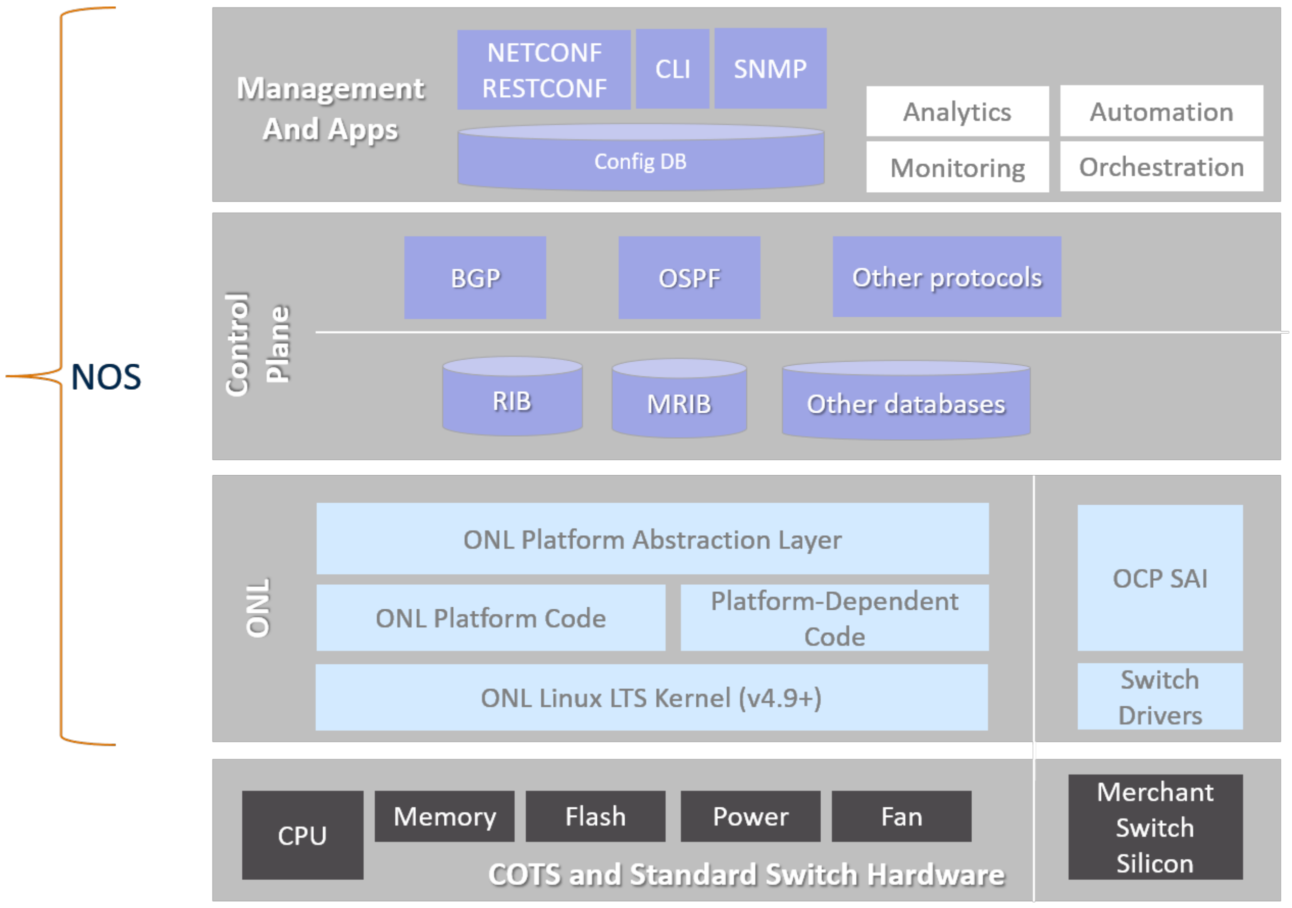
However, as discussed in part 1, a modern NOS should include integration with a NETCONF server, and must provide a single coherent management interface covering all function.

The only open source NETCONF server is OpenYuma. Tail-F (a Cisco company) supply a free version of their ConfD product known as ConfD Basic; this will allow your NOS to support a NETCONF.

Neither option provides a CLI, and in both cases, all areas of function require work to integrate the NETCONF server.

2.2 Architecture

Taking into account the features available today as open source components and greying out those that are not available, we have the following architecture that can be implemented using open source technology.



The following sections provide high level instructions on how to install and configure a NOS using these open source components.

2.3 Installing the OS

Assuming that the target hardware supports ONL, which can be checked on their hardware compatibility list, the installation of ONL is quite simple and is described in the ONL Deployment guide.

2.4 Installing the Data Plane Integration and Control Plane

The other two areas with some open source support are the data plane integration and control plane layer.

2.4.1 Installing the OCP Switch Abstraction Interface

Each switch silicon vendor has their own implementation of the SAI, with its own installation instructions. For example: Mellanox and Broadcom.

2.4.2 Installing Quagga

We've picked Quagga as the simplest example. Installation instructions can be found here.

2.5 Completing the NOS

The remaining work is split across five areas.

2.5.1 Integrating with the ONL platform code

Most whitebox hardware vendors will supply their own drivers. For example, see https://github.com/opennetworklinux/onlp-quanta.

Alternatively, documentation for how to create your own platform drivers can be found at https://opencomputeproject.github.io/OpenNetworkLinux/onlp/.

2.5.2 Integrating Quagga with SAI instead of Linux.

Out of the box, most layer 3 protocol implementations (including Quagga and FRR) are integrated with Linux using the Netlink API. This means routes get programmed into Linux, but not into the forwarding hardware. This may be acceptable for proof-of-concept testing, because data is forwarded by the Linux kernel, but it is entirely unsuitable for production: real networks require fast forwarding in hardware.

There are two options here, both of which require work.

a) Re-engineer the open source protocol implementation to integrate directly with the SAI.
b) Customize Netlink to program the SAI as well as Linux. See https://www.infradead.org/~tgr/libnl/doc/core.html for an example of how.

Option (b) is likely to be less work. However, it has disadvantages:

- The netlink API is limited, and doesn't allow you to program (for example) backup next hops or backup MPLS LSPs for fast switchover.
- By doing this you are effectively creating your own custom version of Linux (because Netlink is such a standardized library), which is likely to make upgrading to new kernels harder.

2.5.3 Supporting additional Network Protocols

You may need to add other simple networking protocols that aren't supported by Quagga and aren't part of ONL. For example, VRRP and DHCP relay are common requirements. Some of these have open source implementations, for example:

- http://www.keepalived.org/
- https://github.com/42wim/isc-dhcp

2.5.4 Management

Quagga comes with its own CLI, but the rest of the system has to be configured by different mechanisms. Service providers expect a consistent management interface for the whole system.

Ideally, you provide a single management agent that owns all configurable state for your device, and provides at least two northbound APIs - a CLI and NETCONF, defined by a set of YANG modules.

An example open source management agent for providing a NETCONF interface is OpenYuma: https://github.com/OpenClovis/OpenYuma.

There are currently no viable open source CLIs - this is a major part of the work of creating a NOS from open source components.

Remember you'll need to cover at least the following functional areas in the interface you present to users:

- Interface/address configuration
- User authentication (e.g. RADIUS, TACAS+)
- Logging control (e.g. syslog)
- Control Plane (BGP, OSPF, ISIS, Static routes, Layer 2, OAM, etc)
- VRFs
- Common Linux utilities like ping, traceroute, viewing routes in the FIB, and so on.

2.5.5 Licensing

You should license your NOS based on standard practice in your organization. For example, you might generate and distribute a license key tied to a specific item of hardware (such as a module/chassis number) without which the software will not run (or will not run with all features enabled). Alternatively, you might trust your customers to comply with the law and audit their usage based on a commercial agreement.

2.6 Deploying the NOS

Having created your NOS, you will need to create an image that can be deployed on OCP hardware.

For any components not included with ONL, you should create a Debian package to install it. See https://wiki.debian.org/Packaging/Intro?action=show&redirect=IntroDebianPackaging for instructions. You can then use https://opennetlinux.org/docs/build to build a new ONIE installer image.

2.7 Conclusion

Building a basic NOS from scratch using purely open source technologies is conceptually straightforward, although there's a fair amount of engineering work to do.

However, adding support for a full range of network protocols to target use cases in service provider networks, and providing a consistent management interface, is unlikely to be practical in-house, and will require engaging with a commercial supplier of the missing components.

In the final part of this cookbook, we will discuss the Alianza NOS Toolkit, which includes the majority of the components required to build a fully functional, carrier-grade NOS for disaggregated network hardware.

3 Building a NOS with the Alianza NOS Toolkit

If you are building a new NOS, the Alianza NOS Toolkit is the best option. The Alianza NOS Toolkit is a full source-code reference solution all the way from management (CLI, Netconf, SNMP), to protocols, to integration with the Broadcom SDK.

Benefits

- Great TTM and low risk – there's very little work for you to ship product, so you can focus on whatever your unique value-add is.
- Total flexibility – you have the source code, so can change any aspect of the NOS if you need to.
- Reference uses Open Networking Linux (ONL) for easy integration of platform-specific controls (fans, LEDs, etc.) and your own software applications, but works on all Linux variants.
- Suitable for all Broadcom XGS-based platforms today, with DNX support coming soon.
- Contains Alianza protocols – the industry choice for feature-rich control plane software with carrier class quality and scalability, which we're constantly adding the latest protocol support to.
- Comes with world-leading Alianza support – your support contact is an engineer that wrote the code.

3.1 Overview of the Alianza NOS Toolkit

The Alianza NOS Toolkit provides a complete software solution for OAM, Switching, Routing and MPLS, which can run on both software and hardware data planes.

This solution comprises the following layers.

- The core of the solution is a range of portable protocol implementations that are designed to meet the requirements of Carrier, Datacenter and Enterprise applications at Internet scale.

- On the Northbound side, the NOS toolkit offers a range of management interfaces for integration with CLIs, NETCONF or SNMP-based management systems and proprietary management agents, and is pre-integrated with ConfD from Tail-F.
- On the Southbound side, Alianza offers pre-integration with Linux, P4 and OpenvSwitch, or hardware data planes such as those provided by Broadcom. The NOS Toolkit also supports integration with abstraction interfaces such as SAI and OpenNSL.

3.2 Use Cases

As described in Part 1, different networking use cases requires support for different sets of control plane protocols and other features. The table below expands on the one in Part 1 to show those protocols and other features supported by the Alianza NOS Toolkit.

Protocol	DC Switch	DCI	CSG	Access	Core	FOSS Available?	MSw NOS Toolkit?
Layer 2 Protocols							
LACP	✓	✓	✓	✓	✓	✓	✓
ECMP	✓	✓	✓	✓	✓		✓
STP				✓	✓		✓
SyncE		✓	✓	✓	✓	✗	✓
EVPN	✓	✓	✓	✓	✓	✓	✓
L2VPN			✓	✓	✓	✗	✓
Layer 3 Protocols							
BGP	✓	✓	✓	✓	✓	✓	✓
ISIS	?	✓	✓	✓	✓	✗	✓
OSPF	✓	✓	✓	✓	✓	✓	✓
PIM	✓			✓	✓		✓
IGMP	✓			✓	✓		✓

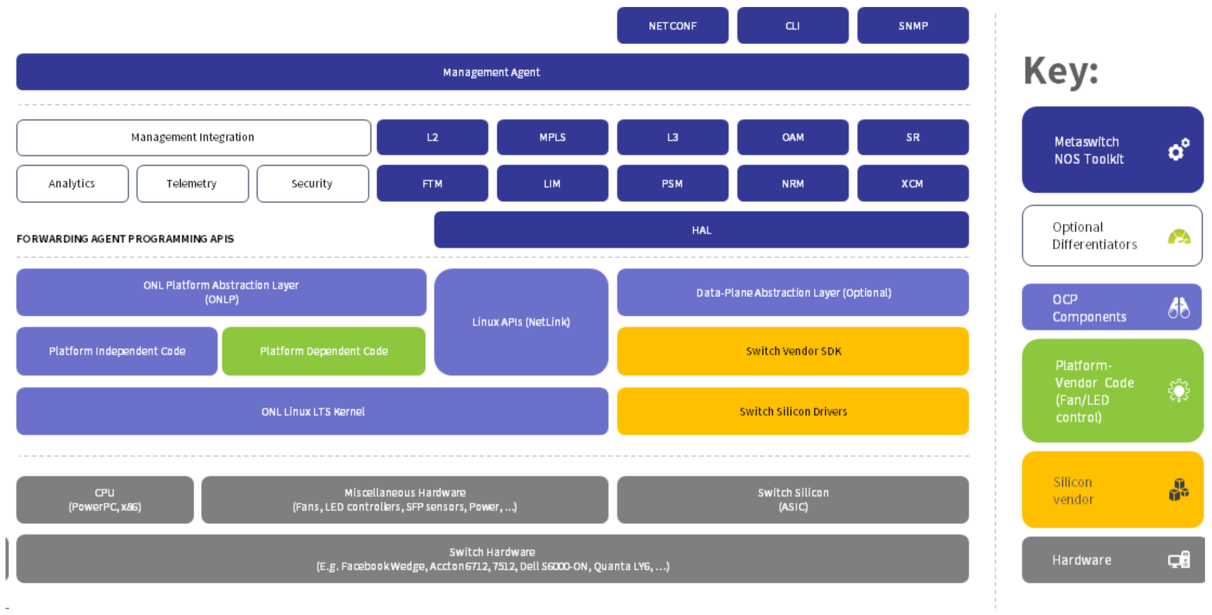
VRRP	✓	✓	✓	✓	✓	✓	✓�
L3VPN			✓	✓	✓	✗	✓
MPLS Protocols							
IP/MPLS		?	✓	✓	✓	✗	✓
LDP			✓	✓	✓	✗	✓
RSVP			✓	✓	✓	✗	✓
LMP			✓	✓	✓	✗	✓
OAM							
BFD	?		✓	✓	✓		✓
LSP-Ping			✓	✓	✓		✓
Y.1731			✓	✓	✓	✗	✓
Management Interface Support							
CLI	?	✓	✓	✓	✓	?	✓
NETCONF	✓	✓	✓	✓	✓	✗	✓
SNMP			✓	✓	✓	✗	✓
Use Case Support							
FOSS	?	?	✗	✗	✗		
MSw NOS Tookit	✓	✓	✓	✓	✓		

(* VRRP supported in NOS Toolkit via open source implementation)

As can be seen from this table, the Alianza NOS Toolkit provides support for all the primary networking use cases and is fully integrated with the management interfaces that operators require.

3.3 Architecture

The architecture supported by the NOS Toolkit is a slight variant of that described in Part 1 above. The following diagram shows the NOS Toolkit architecture.



The following section describe the components of this architecture and explain how the Alianza NOS Toolkit provides the capabilities needed to build a networking operating system meeting the use cases highlighted in the last section.

3.4 Operating System platform

As discussed in Part 1, Open Network Linux (ONL) is recommended for use as the base platform when creating a new network operating system. ONL can be easily installed on any networking hardware that supports the Open Network Install Environment (ONIE), and most manufacturers of such hardware have implemented the drivers necessary to fully support the Open Network Linux Platform APIs to allow NOS implementors to configure and manage hardware components such as LEDs, fans, power supplies, SFPs etc.

The ONL website also contains a hardware compatibility list that is kept up to date with all ONL-certified equipment.

While other distributions could be used as the basis for a NOS, considerably more work would be required to implement these parts of the network operating system and that work is not covered in this cookbook.

3.5 Hardware Abstraction Layer

The interface between the control plane layer and the switch silicon and the Linux networking APIs is a key part of any network operating system. When designing a portable NOS it is vital to include a hardware abstraction layer (HAL).

The HAL is a software component that takes instructions from higher-level software and converts them into a series of commands that are understood by the underlying hardware. It presents a common interface to the software (northbound) while hiding the specific implementation details of the hardware (southbound). As such, a HAL is specific to its hardware platform. Historically, the range of different hardware architectures was vast, and so each platform needed its own HAL.

Now, there is an industry drive towards standardization on a much smaller range of chipsets with significant areas of similarity across vendors, along with the development of common interfaces such as the Open Compute Project's Switch Abstraction Interface (SAI). This has made it possible to develop a HAL that can present a common interface southbound as well as northbound, allowing manufacturers to port their products to different hardware platforms with minimal integration effort.

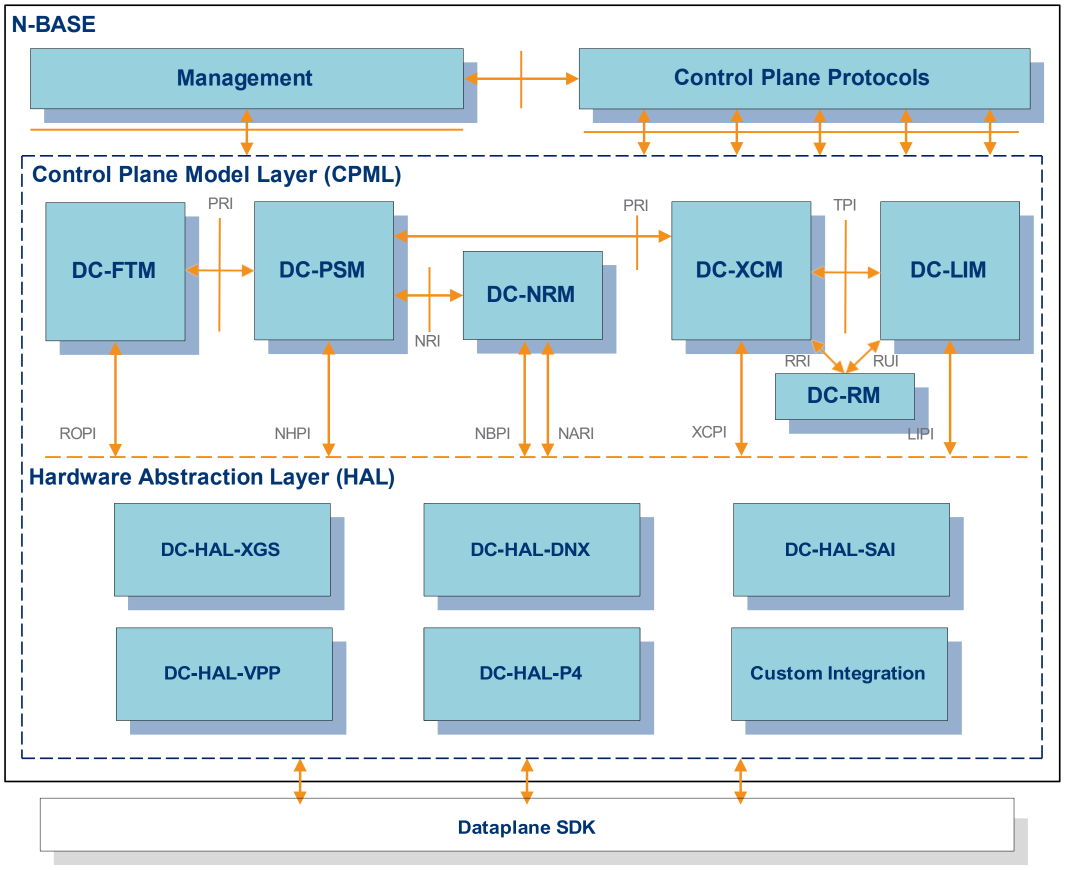
Alianza has more than 30 years of experience in helping customers to integrate the Alianza control plane with a wide range of different platforms. We have used that experience to develop the Alianza HAL. The Alianza HAL is fully integrated with the Alianza control plane and services components and provides them with a common interface for programming both switching silicon and the Linux networking stack. Alianza can provide versions of the HAL that are pre-integrated with various industry-standard platforms, along with all of the support required to write your own.

The Alianza HAL provides disaggregation of the control plane software from the dataplane hardware. This gives you the freedom to upgrade, migrate, or provide a choice of hardware platforms without the time and cost associated with significant integration work. Similarly, you can take advantage of the latest control plane and services functions from Alianza without needing that integration work, which can significantly reduce time to market.

3.6 Services Layer (CPML)

The Control Plane Model Layer (CPML) components provide all the processing intelligence required to integrate the Alianza control plane with a dataplane.

The following diagram shows the basic architecture of the Alianza CPML. Alianza provides the components in blue.



3.6.1 LIM (Local Interface Manager)

- Manages all interface properties, such as queue settings, MTU etc.
- Creates and manages virtual interfaces:
    - VLAN sub-interfaces
    - LACP bundles
    - Integrated Routing and Bridging (IRB) interfaces
    - Tunnels
    - PWs.
- Programs the hardware with properties for physical and virtual interfaces.
- Monitors interface oper status via Fault Reporting Interface (FRI).
- Advertises interface properties and status to control plane components.

3.6.2 FTM (Forwarding Table Manager)

- Manages hardware route table:

    - Creates routes in the layer 3 hardware routing table
    - Tracks unique next hops and requests egress objects for those next hops
    - Tracks unique equal-cost path lists and requests ECMP objects for those path lists
    - Maintains multiple VRF routing tables in hardware, to support layer 3 VPNs.
- Manages OS route table:
    - Programs IP routes into the OS software routing table
    - Reprograms the OS routing table to use backup path in case BFD reports failure of the next hop IP address.
- Manages hardware hosts table:
    - Processes packets sent to unknown IP hosts on attached subnets
    - Creates entries in the hardware hosts table corresponding to attached hosts.

3.6.3 XCM (Cross-Connect Manager)

- Programs MPLS forwarding rules:
    - Manages MPLS cross-connect objects in hardware, which associate an incoming label with:
        - an outgoing labelled egress object (LSP transit)
        - an outgoing unlabelled egress object (LSP penultimate-hop pop), or
        - a rule to loop the packet back to another part of the pipeline (LSP termination).
    - Manages PW objects in hardware, which control encapsulation and decapsulation of pseudowire traffic.
- Tracks unique labelled and unlabelled out-segments and requests layer 2 egress objects for those addresses.
- Creates tunnel programming, for tunnel initiation and termination.

3.6.4 PSM (Path Set Manager)

- Manages paths and path sets.
- Programs next hops and ECMPs corresponding to path sets used by routes and cross-connects.

- Manages Next Hops (labelled or unlabelled)
- Manages ECMP objects, which group paths into sets that are equal-cost for the routes that use them.
- Manages protection group objects, which group primary and backup next hops.
- Requests neighbour information from NRM, and programs next hops/ECMP objects.

3.6.5 NRM (Neighbor Resolution Manager)

- Resolves IP addresses into Neighbor objects (MAC address/interface) as requested by PSM.
- Monitors ARP cache for changes.
- Programs neighbors into dataplane.

3.6.6 RM (Resource Manager)

- Supports QoS and bandwidth reservations in MPLS-TE networks.
- Tracks free and utilized bandwidth on each interface per bandwidth priority level.
- Allocates interface resources to MPLS LSPs, taking sharing into account.
- Controls pre-emption of lower-priority LSPs.
- Updates LIM with bandwidth availability per interface:
  - Supports flooding by OSPF-TE and ISIS-TE
  - RM implements water-marks and hysteresis to prevent excessive flooding and flapping.

3.7 Control Plane Layer

Access, aggregation and DCI networks require extensive control plane functionality.

- Layer 2 and Layer 3 VPN services over MPLS (static, LDP, RSVP-TE).
- Traffic protection using RSVP-TE End-to-end Protection or LDP Fast Reroute (LFA), allowing sub-50ms switchover times in the event of network failure.
- Highly available routing - OSPF, ISIS, BGP with Graceful Restart and VRFs.

- Multicast - IGMP and PIM
- Rich QoS functionality for traffic policing and shaping.
- Full Layer 2 capabilities including MSTP, ERPS, LACP/LAG, LLDP, SyncE, PTP, VLANs and QinQ.
- Full OAM capabilities - including BFD and LSP Ping at the IP/MPLS level, as well as Y.1731.

The following table lists the supported protocols, noting whether the software comes from the Alianza NOS Toolkit, or from downloadable open source.

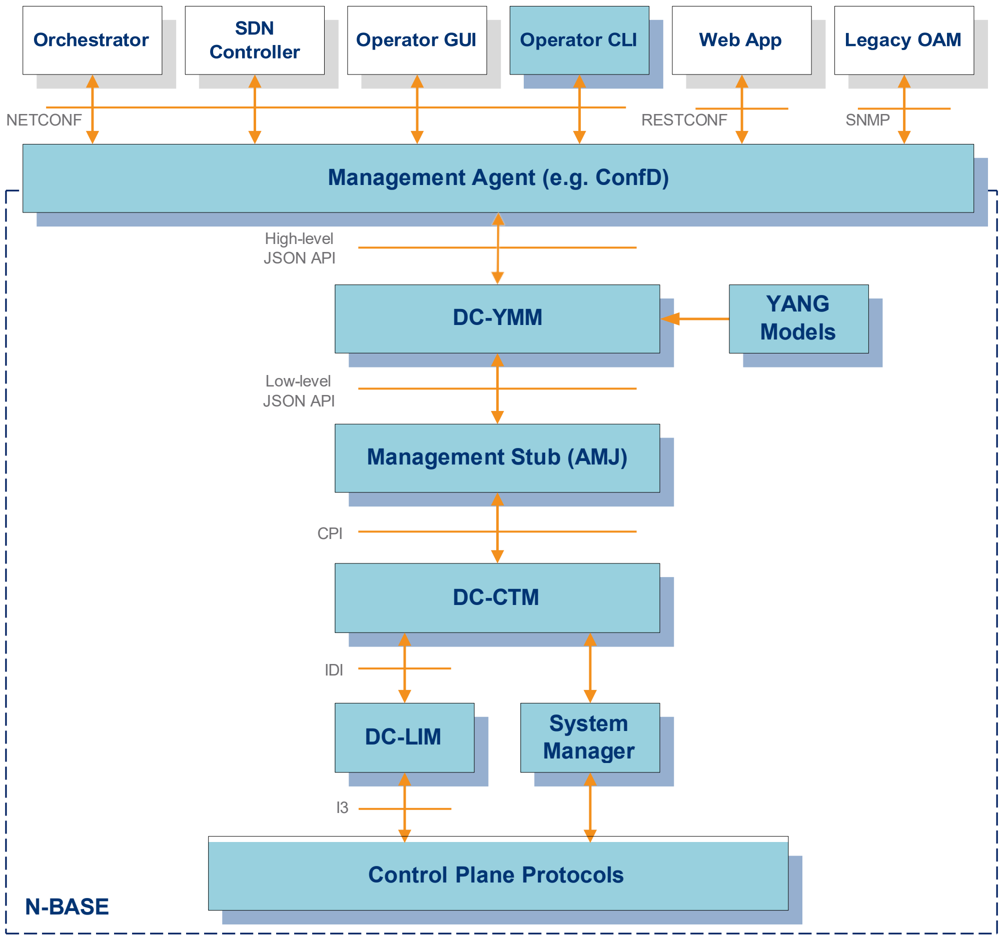
Protocol/Function	**NOS Toolkit**	**Open source**
OSPF	✓	
ISIS	✓	
BGP	✓	
RIP	✓	
PIM	✓	
IGMP	✓	
L2VPNs	✓	
L3VPNs	✓	
Static MPLS	✓	
RSVP-TE with E2E protection	✓	
LDP with Fast Reroute	✓	
High Availability	✓	
QoS	✓	
VLANs/QinQ	✓	
MSTP, RSTP, STP	✓	
ERPS	✓	
LACP, LAG, MC-LAG	✓	
LLDP	✓	

E-LMI	✓	
PTP		✓
SyncE	✓	
VRRP		✓
DHCP relay		✓
BFD	✓	
LSP Ping/Traceroute	✓	
Y.1731	✓	

3.8 Management Layer

Alianza provides a modern management solution that employs NETCONF and YANG for control of the control plane protocols.

The following diagram shows the basic architecture of the Alianza management solution. Alianza provides the components in blue.



3.8.1 YMM (YANG Model Manager)

The YANG Model Manager (DC-YMM) is responsible for mapping between the high-level configuration model used at the command line and the low-level configuration model used by Alianza products.

YMM:

- maps configuration requests from the NETCONF Server into the set of low-level configuration requests required by Alianza products to apply that configuration

- injects the set of configuration requests into the ICP as a single transaction
- maps responses to configuration requests and reports these responses back to the NETCONF Server
- maps requests for read-only state from the NETCONF Server and issues the set of GET requests required to obtain the full set of state from the product code
- maps GET responses from the product code and returns high level read-only state to the NETCONF Server
- maps notifications from the product code to notifications defined in the high-level YANG models
- maps remote procedure calls (RPCs) from the NETCONF Server to the management requests required by product code to carry out the required action(s)
- maintains a database of interface identifier to interface index mappings (using information supplied by the LIM) and uses these mappings to convert between the interface identifiers used at the command line and the indices used by the product code.

In the Alianza Management solution, multiple YANG models are used to define the management schema.

YANG models

The YANG models describe the high-level configuration interface for the Alianza Integrated Control Plane. They are used to generate the end user interfaces such as the CLI. They are based on IETF, OpenConfig and other standard YANG models.

You can optionally define further YANG models to manage your own components.

3.8.2 CTM (Configuration Transaction Manager)

CTM is used to configure Alianza products. It implements a transactional IPS interface.

CTM:

- processes configuration transactions from YMM and ensures that the set of configuration is applied successfully or, in the case of a failure of any part of that configuration, the system is returned to the same state as prior to the transaction being applied.
- sequences the individual configuration requests which make up a transaction in such a way that any ordering restrictions required by the Alianza components are met. For example, it may toggle the administrative status of a row, or deactivate and reactive parent configuration rows to allow a change to take place

- maintains a copy of the configuration held by product code, which it can replicate to a backup.
- injects the configuration cache into product code rapidly on graceful restart.

3.8.3 Management Agent

The Management Agent provides the master database for configuration on the device. It provides the northbound interfaces, assumed to include at least NETCONF. The Alianza solution is fully integrated with ConfD Premium from Tail-f – the industry leading Management Agent.

The Management Agent:

- provides the NETCONF Server API for configuration
- validates requests against YANG models
- processes requests for GETs (SHOW commands) on configuration
- inject requests to configure product code and to retrieve state as described in the high-level YANG models
- processes requests to receive notifications as described in the high-level YANG models
- replicates the master copy of the configuration to provide high availability
- interfaces to authentication and access control (such as RADIUS and TACACS+).

3.8.4 Management user interfaces

The Management Agent exposes northbound interfaces for NETCONF, RESTONF, Web, and SNMP clients. It also provides an industry standard Command Line Interface (CLI) based on the Alianza YANG models.

3.8.5 LIM (Local Interface Manager)

LIM (Local interface Manager) manages all interfaces in the system, including physical interfaces and virtual interfaces (such as LDP tunnels).

It provides the Interface Discovery Interface (IDI) which is used by YMM to obtain the following information.

- It provides a mapping between interface names and the interface identifiers used by the control plane so that interface names can be used at the user interface.

- For virtual interfaces, it provides a pre-allocated interface index when this is required by YMM (for example, where a configuration transaction contains both an LDP tunnel and a pseudowire that uses that tunnel).

3.8.6 System Manager

System Manager provides a unified means of controlling the system. System Manager receives instructions from the Management Stub and configures the Alianza products and stubs as required.

3.8.7 JSON Management Stub (AMJ)

The JSON Management Stub is responsible for receiving JSON transaction requests that are mapped to internal interfaces.

3.9 Management Interfaces

The Alianza solution presents the configuration and management interface in a YANG format suitable for end users and abstracts away complexities of the low-level configuration model. It provides a transactional interface, handling any sequencing requirements of the control plane protocols and rolling back configuration in the event of failure.

The Alianza solution is easy to integrate with any YANG-based Management Agent. It is pre-integrated with the industry leading Management Agent, Tail-f's ConfD Premium, to provide a powerful CLI and NETCONF/RESTCONF interfaces. It also supports a simple JSON-over-TCP API that makes it easy to map onto your existing management agent and configuration database, and we can provide reference integrations with open source Management Agents on request.

In addition, Alianza continues to support integration with a range of legacy management systems using industry-standard and custom MIBs.

3.10 Building the NOS

Once the set of control plane components required for your solution has been identified, a flexible configuration script allows you to determine the set of protocols to be built. The control plane executable, including the dataplane integration code for your target chipset (e.g. Broadcom XGS) is built using standard Linux make tools.

The management plane is built using Scala's build tools, either pre-integrated with Tail-F's ConfD, or to allow integration with another NETCONF server. Some simple configuration is required to set up the communication between the NETCONF server and management plane, and between the management plane and control plane.

Alianza provides documentation and support to guide you through this process (see the *Getting Started Guide for New Users* and the *DC-YMM Integration Guide*).

3.11 Supporting Containerization

The control plane and management plane communicate via a standard socket interface, so placing these into separate containers is straightforward. Modern container solutions such as Docker contain an inbuilt networking framework to provide the forwarding of traffic between the control plane and management plane.

Within the control plane, function is broken out into independent components and Alianza provides a flexible distribution model that allows these components to be distributed across multiple different executables, with an Inter-Location Transport (ILT) mechanism used to communicate between components in different executables. For example, the dataplane integration code could be located in one container, some subset of IP routing function in one container, the remainder in a third container and MPLS function in a fourth container. Containers can then be restarted independently to allow, for example, MPLS to be upgraded without disrupting IP routing function on the device.

Modifying the set of components associated with each container requires a change to a single source file. Each container requires a single configuration file to specify an IP address for the other containers.

Your Alianza support representative can provide you with detailed instructions for supporting containerization on your platform.

3.12 Completing the NOS

At this point you already have something that you can use in demos and trials.

To be able to ship this software, you need to also complete the following steps from section 2.5, **Completing the NOS**:

- **Integrating with the ONL platform code**
- **Supporting additional Network Protocols**
- **Licensing**.

3.13 Deploying the NOS

See section 2.6, **Deploying the NOS**.

3.14 Conclusion

The Alianza NOS Toolkit is available today and is an integrated solution for devices targeting a wide range of networking use cases including cell site gateways, access and aggregation routers and data center interconnect. The comprehensive set of control plane protocols supported by the toolkit come fully integrated with a management layer to provide modern interfaces including NETCONF, RESTCONF, CLI and SNMP. The toolkit also includes all the services and databases required to manage and program the data plane through a hardware abstraction layer to the switch silicon and Linux networking stack.

By leveraging the Alianza NOS Toolkit, any networking operating system provider will benefit from significant time to market advantages as well as being able to rely on Alianza's reputation for supplying high-quality software with a 100% record of on-time delivery.

For further information please contact Alianza via https://www.alianza.com/products/protocol-stacks.