Towards an Open, Disaggregated Network Operating System

Towards an Open, Disaggregated Network Operating System

Contents

1 Introduction

This white paper provides an overview of AT&T's vision for an Open Architecture for a Disaggregated Network Operating System (dNOS). Our goal is to start an industry discussion on technical feasibility, build interest in participating in the formulation of technical detail, and determine suitable vehicles (standards bodies, open source efforts, consortia, etc.) for common specification and architectural realization.

The AT&T Global IP/MPLS network supports all of AT&T's connectivity and application services to consumer and business customers worldwide. This network is comprised of over 100,000 interconnected IP/MPLS routers. These routers have varying levels of network functionality (access aggregation, service-edge, intercity core) and scale. A small number of OEM vendors built generations of IP routers specially targeted at very large-scale, multi-service backbone ISP carrier networks such as AT&T's. These OEM routers were designed, developed and sold as monolithic router platforms with vertically integrated proprietary hardware and software components.

The barrier to entry to creating a Network Operating System has historically been high due to the quantity and complexity of the functional requirements. This complexity extended to both software and hardware. Several previous attempts have been made to create an open NOS, with varying levels of success depending on the targeted use case. However, network vendors, researchers, and developers have made major progress over the last few years.

Advances in software, such as Intel's DPDK and the predominance of YANG models, and in hardware, with silicon chips from vendors such as Broadcom that can meet service provider routing "speeds and feeds" have fostered an ecosystem of networking applications of unprecedented quality and accessibility.

In addition, the growth of merchant silicon forwarding hardware and their corresponding SDKs have launched a hardware ecosystem to address the capability and throughput needs of even the most demanding network appliance role.

Technologies such as the P4 language and interpreting silicon point to even more capable devices soon. The combination of these technologies has created a robust ecosystem of networking applications and building blocks that should be used to create an industry-standard NOS.

The goal: accelerating network innovation.

Data traffic is surging and new customer networking applications (e.g., SD-WAN VPN, IoT networking, and movement of applications into the Cloud) are taking off. A new approach is needed for router platform development and procurement to enable:

- Faster introduction of technologies, designs, and features by means of a collaborative ecosystem of hardware and software component vendors
- Flexibility in network design and service deployment via plug-n-play hardware and software components that can cost-effectively scale up and down
- Unit-cost reduction through use of standard hardware and software technology components with very large economies-of-scale wherever appropriate.

That's how we'll foster an ecosystem of network innovation. That ecosystem is only possible if there is a common open platform on which multiple vendors, companies, organizations, and individuals can build on, contribute to, and certify against. Creating an ecosystem of network software and hardware requires a new level of operating system standardization.

Toward these ends, AT&T has embarked on a plan to create an architecture design and a realization roadmap for a Disaggregated Network Operating System (dNOS) platform with the following high-level design goals:

- Separation of the router's "Network Operating System" (NOS) software from the router's underlying hardware (router chassis, routing controller, forwarding line-cards)
- Well-defined standard interfaces and Application Programming Interfaces (APIs) that provide a framework within the base operating system, control and management plane, and data planes, enabling
    - Customization of each to accommodate size, power, functional and security requirements of specific deployments
    - Modular designs that allow the user to mix and match applications from different private, commercial, and open source suppliers in a model-driven, multi-vendor environment
- Well-defined standard interfaces/APIs that provide a clean separation of control-plane from data plane, enabling
    - A common control plane for multiple forwarding data-plane implementations and technologies including merchant silicon, NPU, x86 CPU and hybrid models
    - Independent scaling of control and data planes by means of dNOS implementations that can run on CPUs contained inside a router hardware platform or alternatively on external servers connected to a router platform, and that can control single or multi-chassis systems, potentially in a geographically distributed environment.

The goal of dNOS is to foster an ecosystem of application and hardware options from multiple vendors. To achieve this vision, it's critical that both hardware and software include standardized interfaces that a community of developers can coalesce around. A single, standardized NOS is the most efficient and effective means to this end. A single NOS allows for qualification of a common, shared integration infrastructure and APIs to help developers rapidly launch new applications. It allows for ecosystem developers to focus on value adding

applications rather than the basic building block components required in all network infrastructure. It presents a common management and operational interface to network operators and orchestration systems across all deployment models. Shared development on a common NOS benefits from the network effects of a distributed development model, such as seen in the Linux ecosystem in general. If widely adopted, it also provides a larger commercial footprint and therefore more incentive for vendors to participate. One goal of the dNOS project would be to encourage the community to coalesce around a single open NOS platform.

While bare metal deployment is a prime consideration of the dNOS architecture, the OS should not be limited to that deployment model. dNOS control plane elements should support operating on a range of general purpose CPU platforms in both bare metal and fully virtualized deployment models.

Additionally, dNOS should support pure hardware based forwarding, pure software based forwarding, or a mix of the two.

1.1 Abstract dNOS Components

The disaggregated Network Operating System (dNOS) consists of hardware and software components.

At a high level, the software components include a base operating system, a control and management plane, and data planes.

The hardware components typically include a general-purpose CPU to run the base operating system, the control and management plane and any software data planes required for the use case. This general-purpose CPU may be virtualized and so the software components should support running in a virtual environment. The hardware components may also include a dedicated hardware forwarding device such as a merchant silicon ASIC, NPU, FPGA, or similar. The general-purpose CPU and the specialized forwarding device may be co-resident in the same hardware or may be separated by a bus or network.

1.2 Proposed dNOS Activities

A first proof point of the viability of this vision – separation of the router hardware and software – was recently demonstrated in a proof-of-concept, production field trial[1]. In the trial, the same router NOS ran on different instances of 3rd-party router hardware ("white boxes").

[1] http://about.att.com/story/white_box_collaboration.html

Further effort is required in the management plane, software integration, and data modeling of the control and management plane. These are key parts of the overall solution. More work is required on the integration of applications at both the control plane and the data plane. Previous efforts may have failed because they were too incomplete to be valuable to a broad ecosystem.

As a next-step, we envision an industry initiative that augments existing efforts toward open architectures, such as OCP SAI and ONF P4, but broader in scope. This effort would focus beyond hardware/software disaggregation and beyond cloud data-center networking features to create and foster a routing software component supplier ecosystem to deliver innovative network solutions that meet the large scale and fast-evolving feature requirements for MAN/WAN networking (encompassing cell-site routers, metro Ethernet service and RAN backhaul routers/switches, Internet and VPN service edge routers, carrier intra-POP interconnect fabric, backbone core routers).

2 Key Functional Components of dNOS

The following is an abstract representation of the functional components in the system.

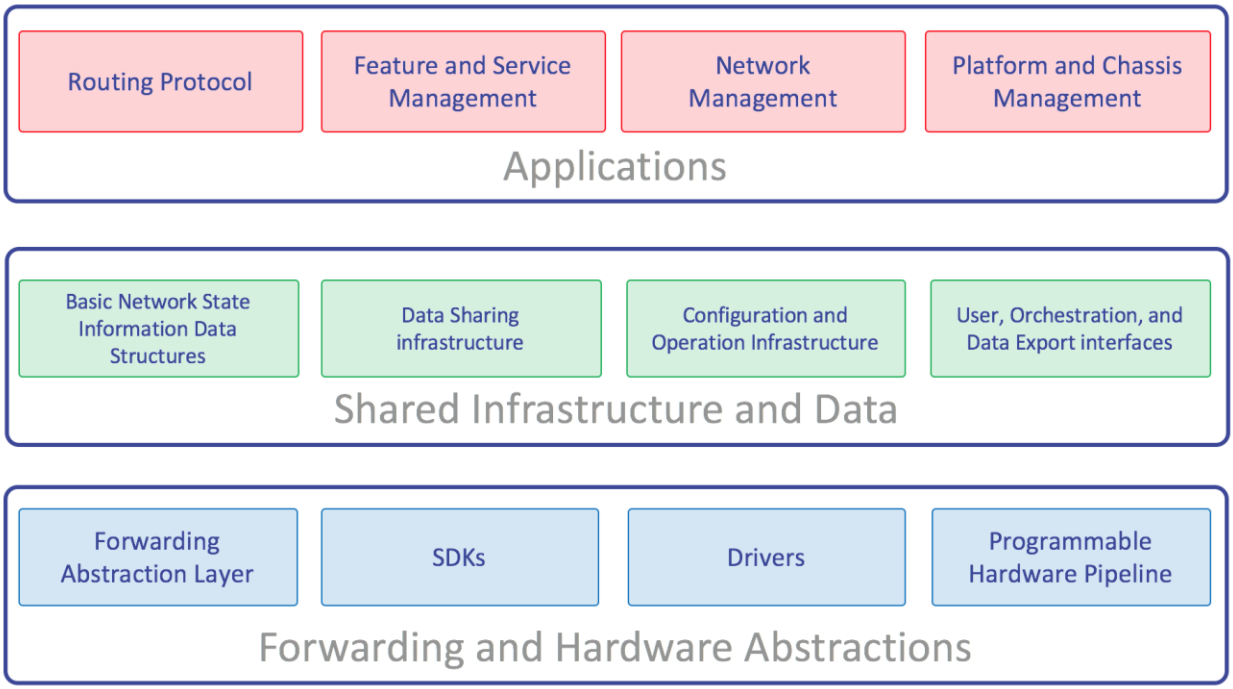

FIGURE 1 - DNOS FUNCTIONAL LAYERS AND COMPONENTS

2.1 Applications

Applications are any network feature on the router that has a control plane or management plane component. dNOS contains a basic set of application classes that interact with their counterparts in other network routers, as well as with ONAP-based management and control systems, to establish and manage the operational states of the given router to affect its service and routing/forwarding features. Not all applications may be categorized by the basic classifications represented in Figure 1.

Routing protocol applications implement the network routing protocol stacks including:

- Transport layer routing protocols for intra- and inter-nodal path selection and traffic engineering, i.e., IGP (OSPF, IS-IS), MPLS label-binding (LDP, SR), BGP-LU, RSVP-TE, BGP-LS, PCEP.
- Service layer routing protocols for end-to-end service routing, i.e., Multi-protocol BGP, BGP-Flow-Spec.

Routing protocol applications interface east/west with other network appliances, but can also interface northbound with ONAP. They interface with their peers using their native protocol definitions, such as with OSPF or BGP peering relationships. Routing protocols are typically responsible for the network–wide routing topology. In this case, they collectively update a common Routing Information Base (RIB), to be discussed in section 2.2, to determine the authoritative topology view for the network appliance. However, they could also be used to define inter-chassis routing topologies, in which case they interface with a chassis manager as opposed to a shared RIB. The chassis manager is further discussed in section 3.2.

Features & Service Management applications maintain configuration control of the network-wide and customer-specific features, for example:

- Access control and security: ACL-based filtering and rate control, encryption.
- Class-of-service: customer and backbone traffic management profiles.

Network Management applications support telemetry and diagnostic capabilities for the platform as well as network-wide capacity, performance and fault management.

Platform Management applications collectively enable management of the physical hardware of the router – hardware component detection, monitoring, diagnosis, as well as environmental control (power, cooling, etc.). One required Platform Management application is the Baseboard Management Controller (BMC) that interfaces to the dNOS device hardware.

The chassis management application is responsible for managing the lifecycle and state of local and remote data planes. It is further responsible for selectively syncing state between control and data planes and managing inter data plane connectivity. The chassis manager does not assume a chassis or an environment with multiple distributed switches. In the case of a single

pizza box type switch, the chassis manager still functions as the interface between the control plane and the single on-box data plane.

2.2 Shared Infrastructure and Data

A multi-vendor NOS must be able to share data between different vendors' applications using a common methodology and protocol. One key goal of dNOS is to create an open environment that facilitates this sharing between applications. To do this, dNOS includes common infrastructure components and shared data structures.

Basic network state information is stored in a set of common shared data structures.

These data structures include information about interface state, neighbor resolution tables, forwarding information base (FIB) state, and other such items. Additionally, a routing information base (RIB) aggregates route information from higher layer routing protocols and determines optimal route for injection into the FIB. The RIB holds the transport- and/or service-layer network reachability information created and maintained by the router based on the routing information exchanges with other network elements (routers, route-reflectors, external elements, etc.) and routing control systems (SDN route controllers). There are distinct RIB types for different routing services (e.g., Internet vs. L3-VPN vs. EVPN, and unicast vs. multicast) and addressing types (IPv4 vs. IPv6). Each router may have more than one RIB type depending on the particular set of network functionality (e.g., P-router vs. PE routers) and services it supports.

Applications use a common methodology to update relevant basic network state shared data. Further, application-specific data is stored within the applications themselves. For example, a BGP application would select a best path from all learned paths and then communicate that selection to the shared common RIB. However, the BGP application would be solely responsible for storing the number of prefixes learned from a given peer as it is specific to the design, form, and function of that specific application.

Applications can share this application-specific data amongst themselves using common data sharing infrastructure provided by dNOS. This infrastructure should support a query methodology to learn what data is available and where it is located. It should also support both subscription and polling methodologies to get regular updates of the data. For example, the BGP application could share the neighbor prefix count periodically with a gRPC application so that an off-system management platform can calculate a route churn histogram for that neighbor.

A common configuration and operation infrastructure is responsible for providing unified configuration and operations interfaces for all of the applications. Some applications may exist in the control plane only as interfaces to the common configuration and operations

infrastructure. For example, access control lists (ACLs) are filters applied to packets forwarded through the data plane. ACLs have no little or no functional code in the system's control and management plane. ACL control plane implementation consists largely of code that interfaces with the configuration and operation infrastructure and translates that configuration into data structures which are applied to the data plane. But many control and management plane applications will include control plane code, scripts, processes, etc. These applications will use the common configuration and operations infrastructure through a standardized API shim layer with the goal of minimal changes to existing applications.

All shared infrastructure data is defined by standardized data models expressed using a suitable standard data structure and/or data-modeling language (e.g., YANG).

The data sharing infrastructure and the basic network state shared data structures enables sharing of common data amongst different protocols (e.g., link state data accessible by classic IGPs such as OSPF and ISIS and by BGP-LS which is used in SDN control applications), as well as graceful evolution of protocol choices (e.g., using gRPC in place of SNMP to collect OAM data).

User, Orchestration, and Data Export interfaces provide a common infrastructure for connecting applications and system infrastructure to external systems for management and analysis. Northbound interfaces to ONAP management & control systems include NetConf/YANG and gRPC, and to support both streaming and (legacy) polling mechanisms for telemetry data collection.

2.3 Forwarding and Hardware Abstractions

dNOS includes a set of components to support multiple different forwarding layers. A forwarding abstraction layer (FAL) is responsible for taking high-level network state input from the shared infrastructure and data components and translating into vendor specific APIs for various software and hardware forwarding options.

The FAL uses SDKs and drivers supplied by the ASIC vendors in the case of NPU ASIC forwarders; and software supplied by combinations of NOS vendors, open-source communities (e.g., DPDK, FD.io), and hardware (e.g., NIC) vendors in the case of software forwarders running on x86 CPU.

The goal of the FAL is to have ASIC supplier diversity and that multiple ASIC's share a common abstraction layer that is target independent.

The long-term goal is to have the FAL include an abstraction that supports fully programmable hardware pipelines. This abstraction would be based on a formal network programming language such as P4 or another similarly industry recognized standardized language.

3 High Level Software Architecture Overview

The software architecture of dNOS envisions three high level layers; the base operating system, the control and management plane, and the data plane. The key functional components and their interfaces are depicted in Figure 2.

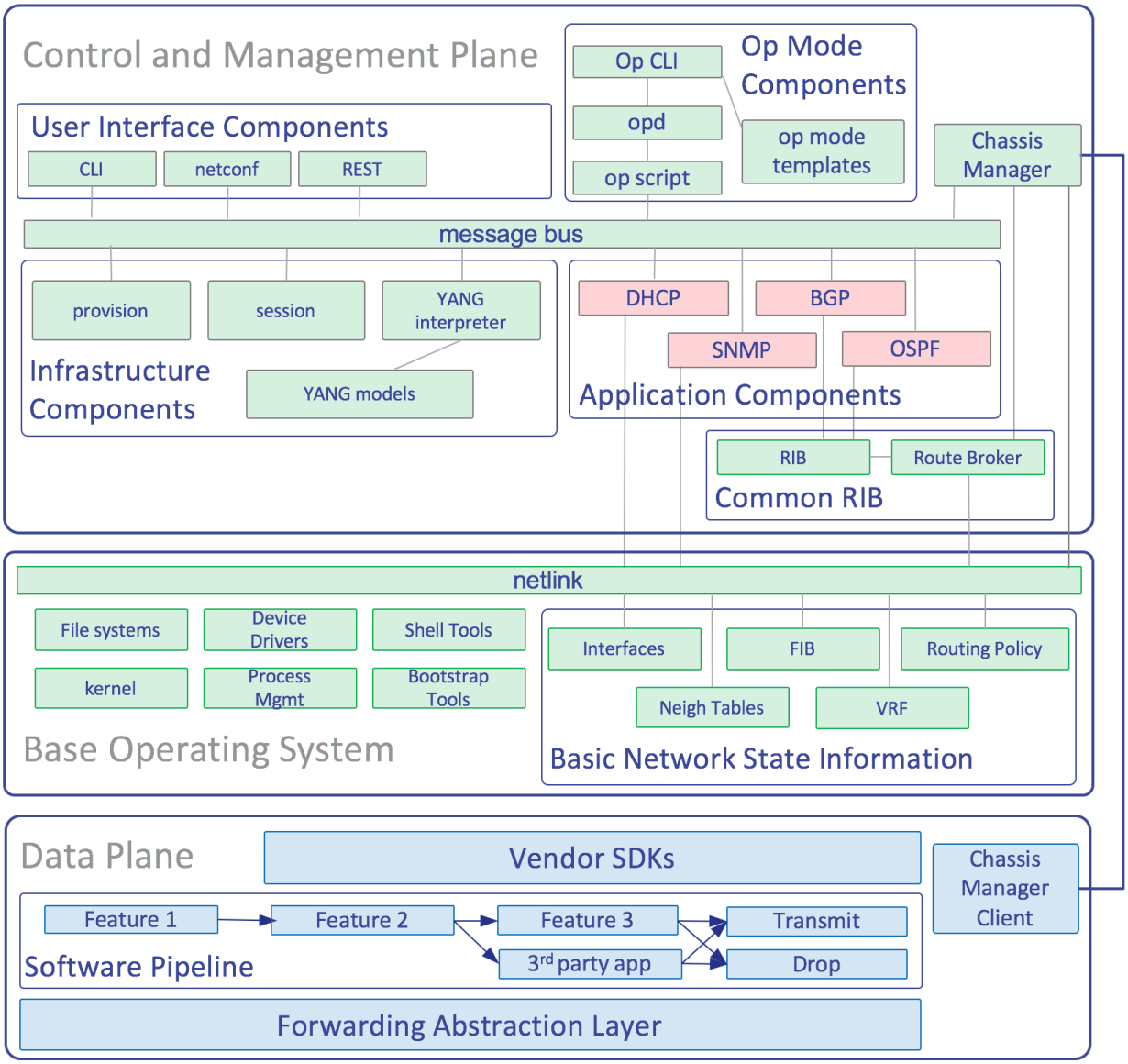

FIGURE 2 - DNOS SOFTWARE ARCHITECTURAL OVERVIEW

Colors correspond to layers represented in Figure 1

3.1 Base Operating System Layer

The base operating system has two primary responsibilities;

- Basic function of the system including bootstrapping, device drivers, process management, shell access, etc.
- Authoritative ownership of the basic network state information

Basic system functions are a prerequisite to any operating system. However, they shouldn't be the focus of the dNOS development community. An existing general purpose operating system should be used to bootstrap dNOS development. Further, as dNOS matures, it should make every effort to continue stay as closely aligned as possible with that general-purpose OS to easily use ongoing development, fixes, and new features. The general-purpose OS should have a well-defined and intuitive existing development process and developer tool chains supported on multiple different hardware architectures. It should be based on open source.

For the purpose of this paper, the base operating system is assumed to be derived from Linux. The base OS should be as closely aligned with one of the major Linux distributions to take advantage of their existing efforts and ecosystems. This distribution is referred to as a parent distribution. For Base Operating System Layer components, development from from the parent distribution is primarily related to non-networking components such as file systems, hardware drivers, shells, etc. However, the control and management plane should also be able to use many network applications from the parent distribution as well. Customization of the base operating system, outside the scope of the networking applications, should be discouraged if it increases development burden. Some Base Operating System Layer customization is unavoidable due to the need to mimic a network appliance operating model. For example, the Base Operating system should be modified to support multiple simultaneous installations and the ability to roll back to previous installs. However, the more customized the Base Operating System is, the more difficult it is to integrate parent distribution features and fixes.

The Base Operating System should support multiple CPU architectures for system portability. At a minimum, it must include support for Intel x86 and ARM. It must be modular, using one of the major package formats (deb or rpm) to allow for easy customization of dNOS to meet the unique requirements of different network devices. The base operating system should support multiple simultaneously installed images using a live boot mechanism or similar. It must support deployments in bare metal and virtualized environments. When installed in a bare metal environment, it should support the ability to run virtual machines and/or containers within the dNOS.

It is important the base operating system is the authoritative repository for basic network state information. In a Linux environment, this means network state is stored in the Linux kernel data

structures. The netlink protocol is the primary method for populating those structures and acts as the conduit between the base operating system and the control and management plane applications. The intent behind this design decision is to be able to easily integrate existing Linux applications without modification. Use of the native Linux network stack also enables a seamless dev/ops environment allowing traditional sysadmin interfaces and scripts to operate within the dNOS.

In this model, all network interfaces attached to the system are represented in the Linux kernel whether or not they are physically attached to the device. One additional benefit to using the native Linux data structures for basic network state data is to support mixed network interface environments. One example of this might be combining Ethernet interfaces on a merchant silicon switch chip with an LTE modem interface supported by the Linux kernel. This is especially important if the goal is to integrate existing unmodified Linux applications that can use all network interface types. These applications expect a set of standard APIs, such as sockets and netlink. If these APIs don't function across all interface types, then the application must be re-written.

The base operating system must include support for Open Network Install Environment (ONIE) to facilitate boot strapping on the most common merchant silicon platforms. It must support booting in both legacy and Unified Extensible Firmware Interface (UEFI) environments.

3.2 Control and Management Plane Layer

The control and management plane is responsible for:

- Managing control plane network feature applications
- Providing the infrastructure that integrates applications into dNOS
- Exposing dNOS configuration, operation and management interfaces to external orchestration systems and end users
- Communicating control plane and system state information to a chassis manager that is responsible for managing the interface to a single local or multiple remote distributed data planes

The control and management plane is primarily responsible for the operation of network feature applications. Examples include a BGP daemon, an SNMP server, a TWAMP server, an IPsec daemon, and a firewall configuration service.

These applications can be sourced from open source, commercial vendors or could be privately built custom applications. As discussed in section 3.1 Base Operating System Layer, feature applications can be unmodified native Linux applications due to the use of standard Linux APIs in the base operating system.

The control and management plane includes infrastructure components that are responsible for integrating applications into dNOS. These infrastructure components include a set of APIs for integrating applications into the system, including their configuration and operational interfaces, in addition to mechanisms for data sharing between applications. Communication between applications and the control and management plane infrastructure is done through a shared Inter-Process Communication (IPC) bus mechanism. Applications may also share their data using this bus with other applications. The flat bus model for communicating shared data is preferable to a modeled hierarchy as the intent is to allow any application to communicate with any other application on the system.

Applications are the ultimate authority for their feature specific data. All configuration, operation, and shared data for each application is modeled in YANG by the feature developer and exposed to the infrastructure components.

Where appropriate, applications continue to use native Linux IPC mechanisms, infrastructure, and shared data. This includes the traditional Linux netlink interface to query and update basic network state information which resides in the Linux kernel. The intent is to be able to easily integrate the existing ecosystem of Linux network applications with minimal or no modification.

Some network features may have no control and management plane function other than to operate as a data store and information proxy with the data plane. One example would be a firewall, where the control plane acts as a configuration and operation proxy to functional code that resides in the data plane. In these cases, the application in the control and management plane still needs to store configuration and translate operational mode data into a form that can be used by the user interface components and other applications. The infrastructure components in the control and management plane provide a framework on which to build these types of applications. Information between applications and data planes are routed through the chassis manager.

All dNOS configuration and operation information is exposed to network operators and external orchestration and management systems through the user interface components. It is critical that dNOS supports common external interfaces for all applications on the system.

A common Command Line Interface (CLI) that combines Linux ecosystem commands with a more familiar network appliance style CLI means easier operation and less training for network staff. Like a traditional network appliance, the CLI should support loading system configuration from a single file to allow for existing operational workflows. It should support commit confirm, configuration rollback, configuration diffs, etc. Additionally, the CLI interface must be scriptable in multiple languages to support dev/ops environments. Where possible and allowed, through the infrastructure components' common Role Based Access Control (RBAC) system, the CLI should allow for single line actions that combine network specific commands modeled in YANG with Linux shell commands.

Common netconf and REST interfaces mean network operators have a single management channel into the system. That single channel can be certified against upstream management and orchestration systems without having to requalify for every new application.

The control and management plane should include a common RIB component. The RIB maintains the traditional role and functions of a router RIB. It is responsible for selecting the best path from a set of options learned from the higher layer routing protocols and distributing that path to various components in the system. It does this through a route broker, who's role is to insure lossless, synchronized routing updates to the various route consumers. The RIB is also responsible for routing policy implementation. The intent with dNOS is to standardize on a single common RIB. This RIB interfaces with the base operating system through the traditional Netlink interface. The interface with upper layer routing protocols is handled through a user space messaging protocol. The adoption of the common RIB and the standardization of the communications path between the RIB and the upper layer routing protocols is to encourage multi-vendor protocol environments for OSPF, ISIS, BGP, etc.

The control and management plane includes a chassis manager responsible for synchronizing state with the data planes. It accepts data from applications and the control and management plane infrastructure across the shared IPC bus. It listens to netlink messages from the base operating system in order to synchronize basic network state information. Finally, it receives routing information from the route broker. This data is selectively transmitted to the appropriate data planes using an IP based transport mechanism. The chassis manager must be able to support basic virtual line card functions including removal and hot plug, multiple line cards in a system, high availability, etc. The chassis manager must be able to support line cards that are physically separated from the control and management plane even if they are geographically dispersed.

The control and management plane must support highly available, redundant deployments.

3.3 Data Plane Layer

The data plane layer is responsible for:

- Synchronizing state between the control and management plane and the data planes
- Providing a forwarding abstraction layer between the control and management plane and the hardware/software data planes
- Providing general packet forwarding functions in hardware and software

A control plane may manage a single or multiple data planes. Those data planes may be co-resident on the same hardware as the control plane, or they may be distributed across a network.

The data plane includes a chassis manager client which is responsible for synchronizing state with the control and management plane. It then presents that data to a forwarding abstraction layer (FAL). That FAL is responsible for translating from abstract network data representation into vendor specific APIs for software and hardware forwarding data planes. This includes state local to the individual data plane(s) such as interface state, FIB state, QoS and firewall configuration, etc.

The software forwarding pipeline is a vector based pipeline that chains together packet forwarding functions using a standardized API. This API forms the basis of the ecosystem for data plane network application developers. These applications could be as simple as a high speed longest prefix match lookup algorithm or as complex as a layer 7 firewall. They can be sourced from commercial vendors, open source organizations, or can be privately developed. The software forwarding pipeline should be a feature complete fully stateful pipeline capable of firewall, NAT, DPI and analytics export functions. It should be capable of terminating various tunnel encapsulation types including encrypted tunnels such as IPsec.

While a pure software pipeline is good at certain tasks (namely high touch, low speed, high memory footprint applications), hardware forwarding is needed for many use cases, especially those requiring high throughput, high port density, and low touch forwarding. The FAL in the data plane acts as a translation element between multiple vendors' merchant silicon SDKs and the control plane's data representation to enable hardware abstraction. The goal of the FAL is to be able to provide an abstraction layer by which dNOS can use multiple vendors' silicon forwarding once a "driver" has been written.

The ultimate dNOS deployment can use both software and hardware forwarding simultaneously. This allows, for example, high speed switching on a merchant silicon device with punt path offload to a software data plane for high speed IPsec termination. Further, the merchant silicon switch and the software forwarding planes need not be co-resident on the same physical hardware or have a 1 to 1 relationship.

4 Realization

AT&T's plan to realize the disaggregated Network Operating System will be based on two mutually supporting approaches: initiating and fostering the industry-wide development of a standardized dNOS platform and adopting processes that favor that platform.

A fundamental goal of this effort is to motivate developers to create value-added applications and innovative solutions for this platform. Such an approach was heretofore nearly impossible with closed and vertically integrated router solutions.

The architecture standardization initiative will seek to define and develop standards for the following key aspects of dNOS architecture:

- Functional definition of key components in each NOS layer

- Common and essential set of northbound and southbound interfaces/APIs for each NOS layer
- Data-models for the common RIB, applications and common infrastructure layers

We call on leading hardware & software vendors to participate in the architecture standardization. The process will also include, where appropriate, existing industry bodies and standard forums (e.g., Linux Foundation, OCP, OpenConfig, P4, IETF).

In parallel, as the effort matures, AT&T will evolve its router-platform sourcing process to give preference to dNOS vendors whose products (or committed product-roadmap) are based on using this platform.

This whitepaper presents information about AT&T's vision for an Open Architecture for a Disaggregated Network Operating System. This information is subject to change without notice to you. No information contained in this whitepaper is or should be interpreted by you as a legal representation, express or implied warranty, agreement, or commitment by AT&T or any of the authors concerning (1) any information or subjects contained in or referenced by this whitepaper, or (2) the furnishing of any products or services by AT&T to you, or (3) the purchase of any products or services by AT&T from you, or (4) any other topic or subject. AT&T may own intellectual property that relates to the information contained in this whitepaper. Notwithstanding anything in this whitepaper to the contrary, no rights or licenses in or to this intellectual property are granted, either expressly or impliedly to you. Rights to AT&T intellectual property may be obtained only by express written agreement with AT&T, signed by AT&T's Chief Technology Officer (CTO) or the CTO's authorized designate.