Juniper Networking Technologies

DAY ONE: UNDERSTANDING OPENCONTRAIL ARCHITECTURE

This reprint from OpenContrail.org provides an overview of OpenContrail, the Juniper technology that sits at the intersection of networking and open source orchestration projects.

By Ankur Singla & Bruno Rijsman

DAY ONE:
UNDERSTANDING OPENCONTRAIL ARCHITECTURE

OpenContrail is an Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network virtualization – SDN controller, virtual router, analytics engine, and published northbound APIs.

This *Day One* book reprints one of the key documents for OpenContrail, the overview of its architecture. Network engineers can now understand how to leverage these emerging technologies, and developers can begin creating flexible network applications.

The next decade begins here.

"The Apache Cloudstack community has been a longtime proponent of the value of open source software, and embraces the contribution of open source infrastructure solutions to the broader industry. We welcome products such as Juniper's OpenContrail giving users of Apache CloudStack open options for the network layer of their cloud environment. We believe this release is a positive step for the industry."

Chip Childers, Vice President, Apache Cloudstack Foundation

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Understand what OpenContrail is and how it operates.
- Implement Network Virtualization.
- Understand the role of OpenContrail in Cloud environments.
- Understand the difference between the OpenContrail Controller and the OpenContrail vRouter.
- Compare the similarities of the OpenContrail system to the architecture of MPLS VPNs.

Juniper Networks Books are singularly focused on network productivity and efficiency. Peruse the complete library at www.juniper.net/books.

Published by Juniper Networks Books

ISBN 978-1936779710

51200

9 781936 779710

JUNIPER NETWORKS

Day One: Understanding OpenContrail Architecture

By Ankur Singla & Bruno Rijsman

Publisher's Note: This book is reprinted from the OpenContrail.org website. It has been adapted to fit this *Day One* format.

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Ankur Singla, Bruno Rijsman
Editor in Chief: Patrick Ames
Copyeditor and Proofer: Nancy Koerbel
J-Net Community Manager: Julie Wider

ISBN: 978-1-936779-71-0 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-72-7 (ebook)

Version History: v1, November 2013
2 3 4 5 6 7 8 9 10

This book is available in a variety of formats at:
http://www.juniper.net/dayone.

Welcome to OpenContrail

This *Day One* book is a reprint of the document that exists on OpenContrail.org. The content of the two documents is the same and has been adapted to fit the *Day One* format.

Welcome to Day One

This book is part of a growing library of *Day One* books, produced and published by Juniper Networks Books.

Day One books were conceived to help you get just the information that you need on day one. The series covers Junos OS and Juniper Networks networking essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

The *Day One* library also includes a slightly larger and longer suite of *This Week* books, whose concepts and test bed examples are more similar to a weeklong seminar.

You can obtain either series, in multiple formats:

- Download a free PDF edition at http://www.juniper.net/dayone.
- Get the ebook edition for iPhones and iPads from the iTunes Store. Search for Juniper Networks Books.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Kindle Store. Search for Juniper Networks Books.
- Purchase the paper edition at either Vervante Corporation (www.vervante.com) or Amazon (amazon.com) for between $12-$28, depending on page length.
- Note that Nook, iPad, and various Android apps can also view PDF files.
- If your device or ebook app uses .epub files, but isn't an Apple product, open iTunes and download the .epub file from the iTunes Store. You can now drag and drop the file out of iTunes onto your desktop and sync with your .epub device.

About OpenContrail

OpenContrail is an Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network virtualization–SDN controller, virtual router, analytics engine, and published northbound APIs. It has an extensive REST API to configure and gather operational and analytics data from the system. Built for scale, OpenContrail can act as a fundamental network platform for cloud infrastructure. The key aspects of the system are:

- Network Virtualization: Virtual networks are the basic building blocks of the OpenContrail approach. Access-control, services, and connectivity are defined via high-level policies. By implmenting inter-network routing in the host, OpenContrail reduces latency for traffic crossing virtual-networks. Eliminating intermediate gateways also improves resiliency and minimizes complexity.
- Network Programmability and Automation: OpenContrail uses a well-defined data model to describe the desired state of the network. It then translates that information into configuration needed by each control node and virtual router. By defining the configuration of the network versus a specific device, OpenContrail simplifies and automates network orchestration.
- Big Data for Infrastructure: The analytics engine is designed for very large scale ingestion and querying of structured and unstructured data. Real-time and historical data is available via a simple REST API, providing visibility over a wide variety of information.

OpenContrail can forward traffic within and between virtual networks without traversing a gateway. It supports features such as IP address management; policy-based access control; NAT and traffic monitoring. It interoperates directly with any network platform that supports the existing BGP/MPLS L3VPN standard for network virtualization.

OpenContrail can use most standard router platforms as gateways to external networks and can easily fit into legacy network environments. OpenContrail is modular and integrates into open cloud orchestration platforms such as OpenStack and Cloudstack, and is currently supported across multiple Linux distributions and hypervisors.

Project Governance

OpenContrail is an open source project committed to fostering innovation in networking and helping drive adoption of the Cloud. OpenContrail gives developers and users access to a production-ready platform

built with proven, stable, open networking standards and network programmability. The project governance model will evolve over time according to the needs of the community. It is Juniper's intent to encourage meaningful participation from a wide range of participants, including individuals as well as organizations.

OpenContrail sits at the intersection of networking and open source orchestration projects. Networking engineering organizations such as the IETF have traditionally placed a strong emphasis on individual participation based on the merits of one's contribution. The same can be said of organizations such as OpenStack with which the Contrail project has strong ties.

As of this moment, the OpenContrail project allows individuals to submit code contributions through GitHub. These contributions will be reviewed by core contributors and accepted based on technical merit only. Over time we hope to expand the group of core contributors with commit privileges.

Getting Started with the Source Code

The OpenContrail source code is hosted across multiple software repositories. The core functionality of the system is present in the contrail-controller repository. The Git multiple repository tool can be used to check out a tree and build the source code. Please follow the instructions.

The controller software is licensed under the Apache License, Version 2.0. Contributors are required to sign a Contributors License Agreement before submitting pull requests.

Developers are required to join the mailing list: dev@lists.opencontrail.org (Join |View), and report bugs using the issue tracker.

Binary

OpenContrail powers the Juniper Networks Contrail product offering that can be downloaded here. Note, this will require registering for an account if you're not already a Juniper.net user. It may take up to 24 hours for Juniper to respond to the new account request.

MORE? It's highly recommended you read the Installation Guide and go through the minimum requirements to get a sense of the installation process before you jump in.

Acronyms Used

AD	Administrative Domain
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
BNG	Broadband Network Gateway
BSN	Broadband Subscriber Network
BSS	Business Support System
BUM	Broadcast, Unknown unicast, Multicast
CE	Customer Edge router
CLI	Command Line Interface
COTS	Common Off The Shelf
CPE	Customer Premises Equipment
CSP	Cloud Service Provider
CO	Central Office
CPU	Central Processing Unit
CUG	Closed User Group
DAG	Directed Acyclic Graph
DC	Data Center
DCI	Data Center Interconnect
DHCP	Dynamic Host Configuration Protocol
DML	Data Modeling Language
DNS	Domain Name System
DPI	Deep Packet Inspection
DWDM	Dense Wavelength Division Multiplexing
EVPN	Ethernet Virtual Private Network
FIB	Forwarding Information Base
GLB	Global Load Balancer
GRE	Generic Route Encapsulation
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IaaS	Infrastructure as a Service
IBGP	Internal Border Gateway Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IF-MAP	Interface for Metadata Access Points
IP	Internet Protocol
IPS	Intrusion Prevention System
IPVPN	Internet Protocol Virtual Private Network
IRB	Integrated Routing and Bridging
JIT	Just In Time
KVM	Kernel-Based Virtual Machines
LAN	Local Area Network
L2VPN	Layer 2 Virtual Private Network

LSP	Label Switched Path
MAC	Media Access Control
MAP	Metadata Access Point
MDNS	Multicast Domain Naming System
MPLS	Multi-Protocol Label Switching
NAT	Network Address Translation
Netconf	Network Configuration
NFV	Network Function Virtualization
NMS	Network Management System
NVO3	Network Virtualization Overlays
OS	Operating System
OSS	Operations Support System
P	Provider core router
PE	Provider Edge router
PIM	Protocol Independent Multicast
POP	Point of Presence
QEMU	Quick Emulator
REST	Representational State Transfer
RI	Routing Instance
RIB	Routing Information Base
RSPAN	Remote Switched Port Analyzer
(S,G)	Source Group
SDH	Synchronous Digital Hierarchy
SDN	Software Defined Networking
SONET	Synchronous Optical Network
SP	Service Provider
SPAN	Switched Port Analyzer
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCG	Trusted Computer Group
TE	Traffic Engineering
TE-LSP	Traffic Engineered Label Switched Path
TLS	Transport Layer Security
TNC	Trusted Network Connect
UDP	Unicast Datagram Protocol
VAS	Value Added Service
vCPE	Virtual Customer Premises Equipment
VLAN	Virtual Local Area Network
VM	Virtual Machine
VN	Virtual Network
VNI	Virtual Network Identifier
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network
XML	Extensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

Chapter 1

Overview of OpenContrail

This chapter provides an overview of the OpenContrail System – an extensible platform for Software Defined Networking (SDN).

All of the main concepts are briefly introduced in this chapter and described in more detail in the remainder of this document.

Use Cases

OpenContrail is an extensible system that can be used for multiple networking use cases but there are two primary drivers of the architecture:

- Cloud Networking – Private clouds for Enterprises or Service Providers, Infrastructure as a Service (IaaS) and Virtual Private Clouds (VPCs) for Cloud Service Providers.
- Network Function Virtualization (NFV) in Service Provider Network – This provides Value Added Services (VAS) for Service Provider edge networks such as business edge networks, broadband subscriber management edge networks, and mobile edge networks.

The Private Cloud, the Virtual Private Cloud (VPC), and the Infrastructure as a Service (IaaS) use cases all involve a multi-tenant virtualized data centers. In each of these use cases multiple tenants in a data center share the same physical resources (physical servers, physical storage, physical network). Each tenant is assigned its own logical resources (virtual machines, virtual

storage, virtual networks). These logical resources are isolated from each other, unless specifically allowed by security policies. The virtual networks in the data center may also be interconnected to a physical IP VPN or L2 VPN.

The Network Function Virtualization (NFV) use case involves orchestration and management of networking functions such as a Firewalls, Intrusion Detection or Preventions Systems (IDS / IPS), Deep Packet Inspection (DPI), caching, Wide Area Network (WAN) optimization, etc. in virtual machines instead of on physical hardware appliances. The main drivers for virtualization of the networking services in this market are time to market and cost optimization.

OpenContrail Controller and the vRouter

The OpenContrail System consists of two main components: the OpenContrail Controller and the OpenContrail vRouter.

The OpenContrail Controller is a logically centralized but physically distributed Software Defined Networking (SDN) controller that is responsible for providing the management, control, and analytics functions of the virtualized network.

The OpenContrail vRouter is a forwarding plane (of a distributed router) that runs in the hypervisor of a virtualized server. It extends the network from the physical routers and switches in a data center into a virtual overlay network hosted in the virtualized servers (the concept of an overlay network is explained in more detail in section 1.4 below). The OpenContrail vRouter is conceptually similar to existing commercial and open source vSwitches such as for example the Open vSwitch (OVS) but it also provides routing and higher layer services (hence vRouter instead of vSwitch).

The OpenContrail Controller provides the logically centralized control plane and management plane of the system and orchestrates the vRouters.

Virtual Networks

Virtual Networks (VNs) are a key concept in the OpenContrail System. Virtual networks are logical constructs implemented on top of the physical networks. Virtual networks are used to replace VLAN-based isolation and provide multi-tenancy in a virtualized data center. Each tenant or an application can have one or more virtual networks. Each virtual network is isolated from all the other virtual networks unless explicitly allowed by security policy.

Virtual networks can be connected to, and extended across physical Multi-Protocol Label Switching (MPLS) Layer 3 Virtual Private Networks (L3VPNs) and Ethernet Virtual Private Networks (EVPNs) networks using a datacenter edge router.

Virtual networks are also used to implement Network Function Virtualization (NFV) and service chaining. How this is achieved using virtual networks is explained in detail in Chapter 2.

Overlay Networking

Virtual networks can be implemented using a variety of mechanisms. For example, each virtual network could be implemented as a Virtual Local Area Network (VLAN), or as Virtual Private Networks (VPNs), etc.

Virtual networks can also be implemented using two networks – a physical underlay network and a virtual overlay network. This overlay networking technique has been widely deployed in the Wireless LAN industry for more than a decade but its application to data-center networks is relatively new. It is being standardized in various forums such as the Internet Engineering Task Force (IETF) through the Network Virtualization Overlays (NVO3) working group and has been implemented in open source and commercial network virtualization products from a variety of vendors.

The role of the physical underlay network is to provide an "IP fabric" – its responsibility is to provide unicast IP connectivity from any physical device (server, storage device, router, or switch) to any other physical device. An ideal underlay network provides uniform low-latency, non-blocking, high-bandwidth connectivity from any point in the network to any other point in the network.

The vRouters running in the hypervisors of the virtualized servers create a virtual overlay network on top of the physical underlay network using a mesh of dynamic "tunnels" amongst themselves. In the case of OpenContrail these overlay tunnels can be MPLS over GRE/UDP tunnels, or VXLAN tunnels.

The underlay physical routers and switches do not contain any per-tenant state: they do not contain any Media Access Control (MAC) addresses, IP address, or policies for virtual machines. The forwarding tables of the underlay physical routers and switches only contain the IP prefixes or MAC addresses of the physical servers. Gateway routers or switches that connect a virtual network to a physical network are an exception – they do need to contain tenant MAC or IP addresses.

The vRouters, on the other hand, do contain per tenant state. They contain a separate forwarding table (a routing-instance) per virtual network. That forwarding table contains the IP prefixes (in the case of a Layer 3 overlays) or the MAC addresses (in the case of Layer 2 overlays) of the virtual machines. No single vRouter needs to contain all IP prefixes or all MAC addresses for all virtual machines in the entire Data Center. A given vRouter only needs to contain those routing instances that are locally present on the server (i.e. which have at least one virtual machine present on the server.)

Overlays Based on MPLS L3VPNs and EVPNs

Various control plane protocols and data plane protocols for overlay networks have been proposed by vendors and standards organizations.

For example, the IETF VXLAN draft [draft-mahalingam-dutt-dcops-vxlan] proposes a new data plane encapsulation and proposes a control plane which is similar to the standard Ethernet "flood and learn source address" behavior for filling the forwarding tables and which requires one or more multicast groups in the underlay network to implement the flooding.

The OpenContrail System is inspired by, and conceptually very similar to, standard MPLS Layer 3VPNs (for Layer 3 overlays) and MPLS EVPNs (for Layer 2 overlays).

In the data plane, OpenContrail supports MPLS over GRE, a data plane encapsulation that is widely supported by existing routers from all major vendors. OpenContrail also supports other data plane encapsulation standards such as MPLS over UDP (better multi-pathing and CPU utilization) and VXLAN. Additional encapsulation standards such as NVGRE can easily be added in future releases.

The control plane protocol between the control plane nodes of the OpenContrail system or a physical gateway router (or switch) is BGP (and Netconf for management). This is the exact same control plane protocol that is used for MPLS Layer 3VPNs and MPLS EVPNs.

The protocol between the OpenContrail controller and the OpenContrail vRouters is based on XMPP [ietf-xmpp-wg]. The schema of the messages exchanged over XMPP is described in an IETF draft [draft-ietf-l3vpn-end-system] and this protocol, while syntactically different, is semantically very similar to BGP.

The fact that the OpenContrail System uses control plane and data plane protocols which are very similar to the protocols used for MPLS

Layer 3VPNs and EVPNs has multiple advantages – these technologies are mature and known to scale, they are widely deployed in production networks, and supported in multi-vendor physical gear that allows for seamless interoperability without the need for software gateways.

OpenContrail and Open Source

OpenContrail is designed to operate in an open source Cloud environment. In order to provide a fully integrated end-to-end solution:

- The OpenContrail System is integrated with open source hypervisors such as Kernel-based Virtual Machines (KVM) and Xen.
- The OpenContrail System is integrated with open source virtualization orchestration systems such as OpenStack and CloudStack.
- The OpenContrail System is integrated with open source physical server management systems such as Chef, Puppet, Cobbler, and Ganglia.

OpenContrail is available under the permissive Apache 2.0 license – this essentially means that anyone can deploy and modify the OpenContrail System code without any obligation to publish or release the code modifications.

Juniper Networks also provides a commercial version of the OpenContrail System. Commercial support for the entire open source stack (not just the OpenContrail System, but also the other open source components such as OpenStack) is available from Juniper Networks and its partners.

The open source version of the OpenContrail System is not a *teaser* – it provides the same full functionality as the commercial version both in terms of features and in terms of scaling.

Scale-Out Architecture and High Availability

Earlier we mentioned that the OpenContrail Controller is logically centralized but physically distributed.

Physically distributed means that the OpenContrail Controller consists of multiple types of nodes, each of which can have multiple instances for high availability and horizontal scaling. Those node instances can be physical servers or virtual machines. For minimal deployments, multiple node types can be combined into a single server. There are three types of nodes:

- Configuration nodes are responsible for the management layer. The configuration nodes provide a north-bound Representational State Transfer (REST) Application Programming Interface (API) that can be used to configure the system or extract operational status of the system. The instantiated services are represented by objects in a horizontally scalable database that is described by a formal service data model (more about data models later on). The configuration nodes also contain a transformation engine (sometimes referred to as a compiler) that transforms the objects in the high-level service data model into corresponding more lower-level objects in the technology data model. Whereas the high-level service data model describes *what* services need to be implemented, the low-level technology data model describes *how* those services need to be implemented. The configuration nodes publish the contents of the low-level technology data model to the control nodes using the Interface for Metadata Access Points (IF-MAP) protocol.
- Control nodes implement the logically centralized portion of the control plane. Not all control plane functions are logically centralized – some control plane functions are still implemented in a distributed fashion on the physical and virtual routers and switches in the network. The control nodes use the IF-MAP protocol to monitor the contents of the low-level technology data model as computed by the configuration nodes that describes the desired state of the network. The control nodes use a combination of south-bound protocols to "make it so," i.e., to make the actual state of the network equal to the desired state of the network. In the initial version of the OpenContrail System these south-bound protocols include Extensible Messaging and Presence Protocol (XMPP) to control the OpenContrail vRouters as well as a combination of the Border Gateway Protocol (BGP) and the Network Configuration (Netconf) protocols to control physical routers. The control nodes also use BGP for state synchronization among each other when there are multiple instances of the control node for scale-out and high-availability reasons.
- Analytics nodes are responsible for collecting, collating and presenting analytics information for trouble shooting problems and for understanding network usage. Each component of the OpenContrail System generates detailed event records for every significant event in the system. These event records are sent to one of multiple instances (for scale-out) of the analytics node that collate and store the information in a horizontally scalable

database using a format that is optimized for time-series analysis and queries. The analytics nodes have mechanism to automatically trigger the collection of more detailed records when certain event occur; the goal is to be able to get to the root cause of any issue without having to reproduce it. The analytics nodes provide a north-bound analytics query REST API.

The physically-distributed nature of the OpenContrail Controller is a distinguishing feature. Because there can be multiple redundant instances of any node, operating in an active-active mode (as opposed to an active-standby mode), the system can continue to operate without any interruption when any node fails. When a node becomes overloaded, additional instances of that node type can be instantiated, after which the load is automatically redistributed. This prevents any single node from becoming a bottleneck and allows the system to manage very large-scale systems – tens of thousands of servers.

Logically centralized means that OpenContrail Controller behaves as a single logical unit, despite the fact that it is implemented as a cluster of multiple nodes.

The Central Role of Data Models: SDN as a Compiler

Data models play a central role in the OpenContrail System. A data model consists of a set of objects, their capabilities, and the relationships between them.

The data model permits applications to express their intent in a declarative rather than an imperative manner, which is critical in achieving high programmer productivity. A fundamental aspect of OpenContrail's architecture is that data manipulated by the platform, as well as by the applications, is maintained by the platform. Thus applications can be treated as being virtually stateless. The most important consequence of this design is that individual applications are freed from having to worry about the complexities of high availability, scale, and peering.

There are two types of data models: the high-level service data model and the low-level technology data model. Both data models are described using a formal data modeling language that is currently based on an IF-MAP XML schema although YANG is also being considered as a future possible modeling language.

The high-level service data model describes the desired state of the network at a very high level of abstraction, using objects that map directly to services provided to end-users – for example, a virtual network, or a connectivity policy, or a security policy.

The low-level technology data model describes the desired state of the network at a very low level of abstraction, using objects that map to specific network protocol constructs such as a BGP route-target, or a VXLAN network identifier.

The configuration nodes are responsible for transforming any change in the high-level service data model to a corresponding set of changes in the low-level technology data model. This is conceptually similar to a Just In Time (JIT) compiler – hence the term "SDN as a compiler" is sometimes used to describe the architecture of the OpenContrail System.

The control nodes are responsible for realizing the desired state of the network as described by the low-level technology data model using a combination of southbound protocols including XMPP, BGP, and Netconf.

Northbound Application Programming Interfaces

The configuration nodes in the OpenContrail Controller provide a northbound Representational State Transfer (REST) Application Programming Interface (API) to the provisioning or orchestration system. This northbound REST API is automatically generated from the formal high-level data model. This guarantees that the northbound REST API is a "first class citizen" in the sense that any and every service can be provisioned through the REST API.

This REST API is secure: it can use HTTPS for authentication and encryption and it also provides role-based authorization. It is also horizontally scalable because the API load can be spread over multiple configuration node instances.

Graphical User Interface

The OpenContrail System also provides a Graphical User Interface (GUI). This GUI is built entirely using the REST API described earlier and this ensures that there is no lag in APIs. It is expected that large-scale deployments or service provider OSS/BSS systems will be integrated using the REST APIs.

NOTE Juniper is in the process of making changes to the UI code-base that will make it available in the open-source.

An Extensible Platform

The initial version of the OpenContrail System ships with a specific high-level service data model, a specific low-level technology data model, and a transformation engine to map the former to the latter. Furthermore, the initial version of the OpenContrail System ships with a specific set of southbound protocols.

The high-level service data model that ships with the initial version of the OpenContrail System models service constructs such as tenants, virtual networks, connectivity policies, and security policies. These modeled objects were chosen to support initial target use cases, namely cloud networking and NFV.

The low-level service data model that ships with the initial version of the OpenContrail System is specifically geared towards implementing the services using overlay networking.

The transformation engine in the configuration nodes contains the "compiler" to transform this initial high-level service data model to the initial low-level data model.

The initial set of south-bound protocols implemented in the control nodes consists of XMPP, BGP, and Netconf.

The OpenContrail System is an extensible platform in the sense that any of the above components can be extended to support additional use cases and/or additional network technologies in future versions:

- The high-level service data model can be extended with additional objects to represent new services such as for example traffic engineering and bandwidth calendaring in Service Provider core networks.
- The low-level service data model can also be extended for one of two reasons: either the same high-level services are implemented using a different technology, for example multi-tenancy could be implemented using VLANs instead of overlays, or new high-level services could be introduced which require new low-level technologies, for example introducing traffic engineering or bandwidth calendaring as a new high-level service could require the introduction of a new low-level object such as a Traffic-Engineered Label Switched Path (TE-LSP).
- The transformation engine could be extended either to map existing high-level service objects to new low-level technology objects (i.e., a new way to implement an existing service) or to map new high-level service objects to new or existing low-level technology objects (i.e., implementing a new service).

New southbound protocols can be introduced into the control nodes. This may be needed to support new types of physical or virtual devices in the network that speak a different protocol, for example the Command Line Interface (CLI) for a particular network equipment vendor could be introduced, or this may be needed because new objects are introduced in the low-level technology data models that require new protocols to be implemented.

Chapter 2

OpenContrail Architecture Details

The OpenContrail System consists of two parts: a logically centralized but physically distributed controller, and a set of vRouters that serve as software forwarding elements implemented in the hypervisors of general purpose virtualized servers. These are illustrated in Figure 1.

The controller provides northbound REST APIs used by applications. These APIs are used for integration with the cloud orchestration system, for example for integration with OpenStack via a neutron (formerly known as quantum) plug-in. The REST APIs can also be used by other applications and/or by the operator's OSS/BSS. Finally, the REST APIs are used to implement the web-based GUI included in the OpenContrail System.

The OpenContrail System provides three interfaces: a set of north-bound REST APIs that are used to talk to the Orchestration System and the Applications, southbound interfaces that are used to talk to virtual network elements (vRouters) or physical network elements (gateway routers and switches), and an east-west interface used to peer with other controllers. OpenStack and CloudStack are the supported orchestrators, standard BGP is the east-west interface, XMPP is the southbound interface for vRouters, BGP and Netconf and the southbound interfaces for gateway routers and switches.

Internally, the controller consists of three main components:

1. Configuration nodes, which are responsible for translating the high-level data model into a lower level form suitable for interacting with network elements;
2. Control nodes, which are responsible for propagating this low level state to and from network elements and peer systems in an eventually consistent way;
3. Analytics nodes, which are responsible for capturing real-time data from network elements, abstracting it and presenting it in a form suitable for applications to consume.

NOTE All of these nodes will be described in detail later in this chapter.

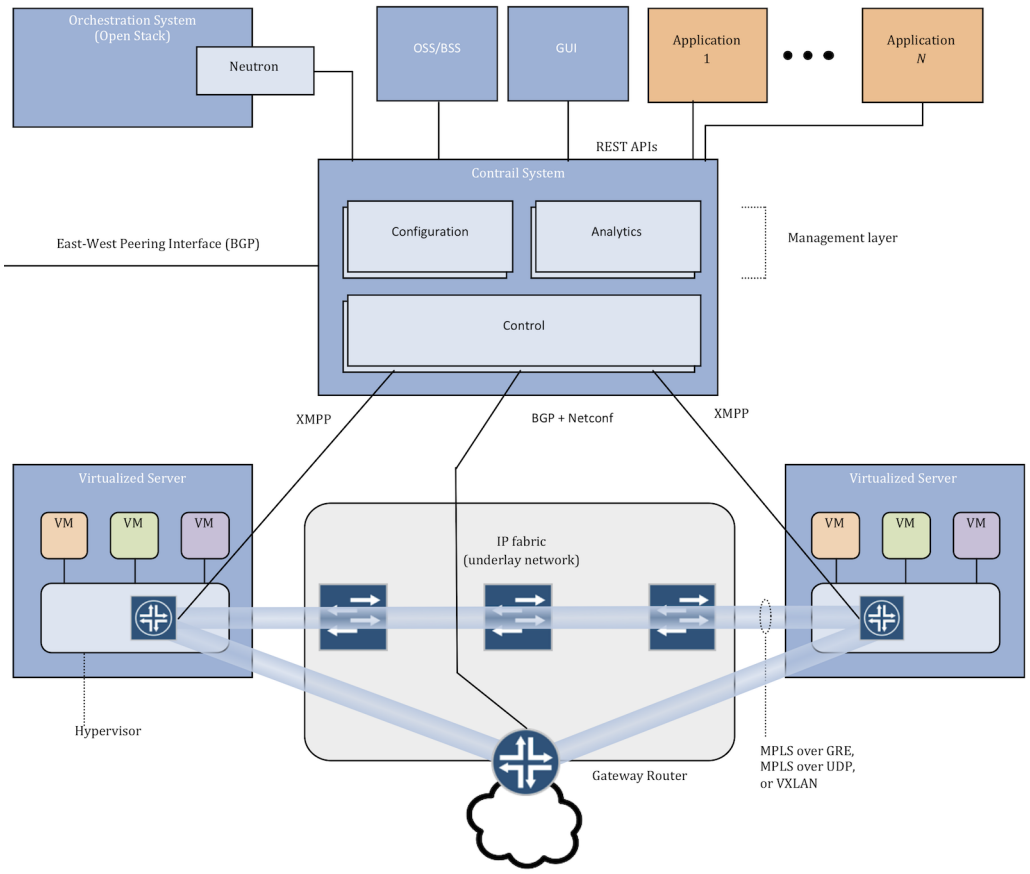

Figure 1 OpenContrail System Overview

The vRouters should be thought of as network elements implemented entirely in software. They are responsible for forwarding packets from one virtual machine to other virtual machines via a set of server-to-server tunnels. The tunnels form an overlay network sitting on top of a physical IP-over-Ethernet network. Each vRouter consists of two parts: a user space agent that implements the control plane and a kernel module that implements the forwarding engine.

The OpenContrail System implements three basic building blocks:

1. Multi-tenancy, also known as network virtualization or network slicing, is the ability to create Virtual Networks that provide Closed User Groups (CUGs) to sets of VMs.
2. Gateway functions: this is the ability to connect virtual networks to physical networks via a gateway router (e.g., the Internet), and the ability to attach a non-virtualized server or networking service to a virtual network via a gateway.
3. Service chaining, also known Network Function Virtualization (NFV): this is the ability to steer flows of traffic through a sequence of physical or virtual network services such as firewalls, Deep Packet Inspection (DPI), or load balancers.

Nodes

We now turn to the internal structure of the system. As shown in Figure 2, the system is implemented as a cooperating set of nodes running on general-purpose x86 servers. Each node may be implemented as a separate physical server or it may be implemented as a Virtual Machine (VM).

All nodes of a given type run in an active-active configuration so no single node is a bottleneck. This scale out design provides both redundancy and horizontal scalability.

- *Configuration nodes* keep a persistent copy of the intended configuration state and translate the high-level data model into the lower level model suitable for interacting with network elements. Both of these are kept in a NoSQL database.
- *Control nodes* implement a logically centralized control plane that is responsible for maintaining ephemeral network state. Control nodes interact with each other and with network elements to ensure that network state is eventually consistent.
- *Analytics nodes* collect, store, correlate, and analyze information from network elements, virtual or physical. This information includes statistics, logs, events, and errors.

In addition to the node types, which are part of the OpenContrail Controller, we also identify some additional nodes types for physical servers and physical network elements performing particular roles in the overall OpenContrail System:

- *Compute nodes* are general-purpose virtualized servers, which host VMs. These VMs may be tenant-running general applications, or these VMs may be service VMs running network services such as a virtual load balancer or virtual firewall. Each compute node contains a vRouter that implements the forwarding plane and the distributed part of the control plane.
- *Gateway nodes* are physical gateway routers or switches that connect the tenant virtual networks to physical networks such as the Internet, a customer VPN, another Data Center, or to non-virtualized servers.
- *Service nodes* are physical network elements providing network services such as Deep Packet Inspection (DPI), Intrusion Detection (IDP), Intrusion Prevention (IPS), WAN optimizers, and load balancers. Service chains can contain a mixture of virtual services (implemented as VMs on compute nodes) and physical services (hosted on service nodes).

For clarity, Figure 2 does not show physical routers and switches that form the underlay IP over Ethernet network. There is also an interface from every node in the system to the analytics nodes. This interface is not shown in Figure 2 to avoid clutter.

Compute Node

The compute node is a general-purpose x86 server that hosts VMs. Those VMs can be tenant VMs running customer applications, such as web servers, database servers, or enterprise applications, or those VMs can be host virtualized services use to create service chains. The standard configuration assumes Linux is the host OS and KVM or Xen is the hypervisor. The vRouter forwarding plane sits in the Linux Kernel; and the vRouter Agent is the local control plane. This structure is shown in Figure 3.

Other host OSs and hypervisors such as VMware ESXi or Windows Hyper-V may also be supported in future.

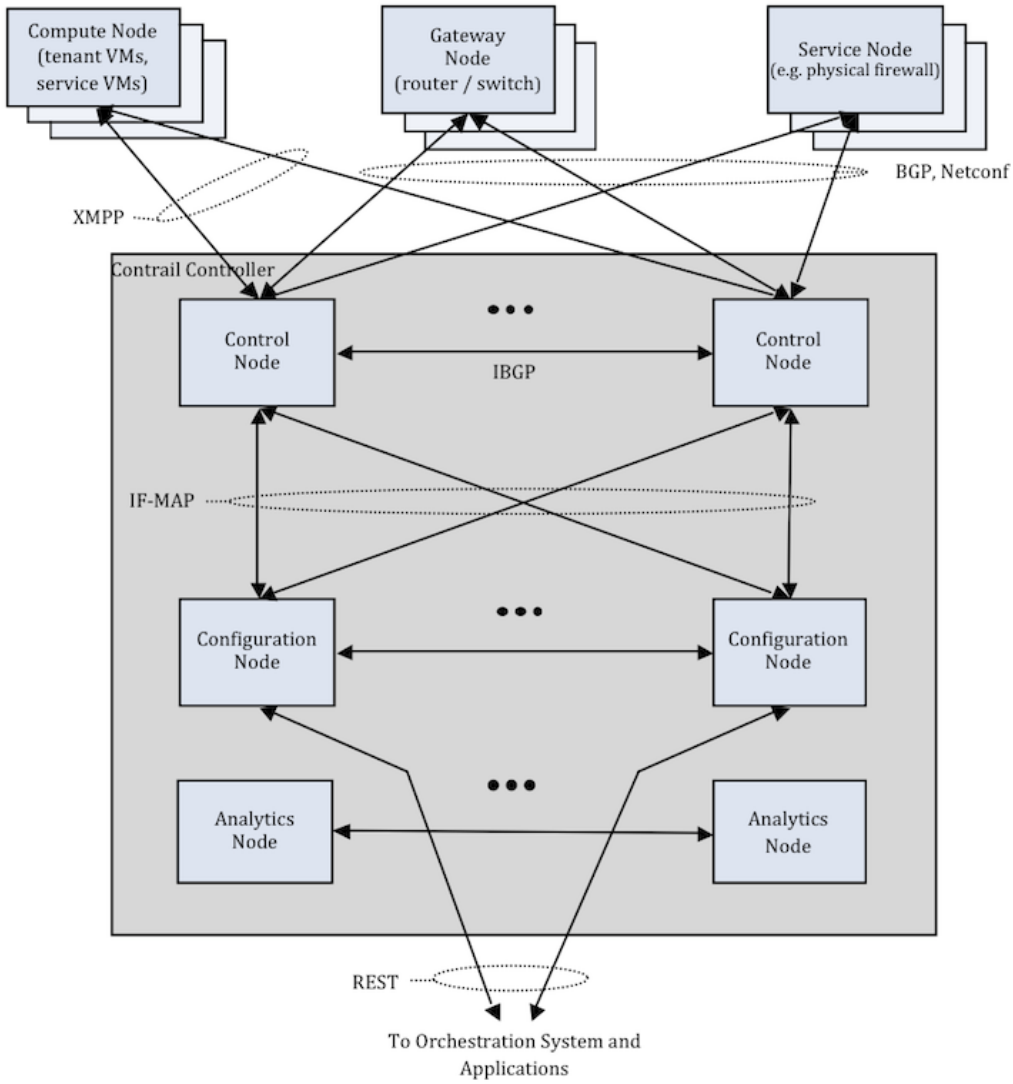

Figure 2 OpenContrail System Implementation

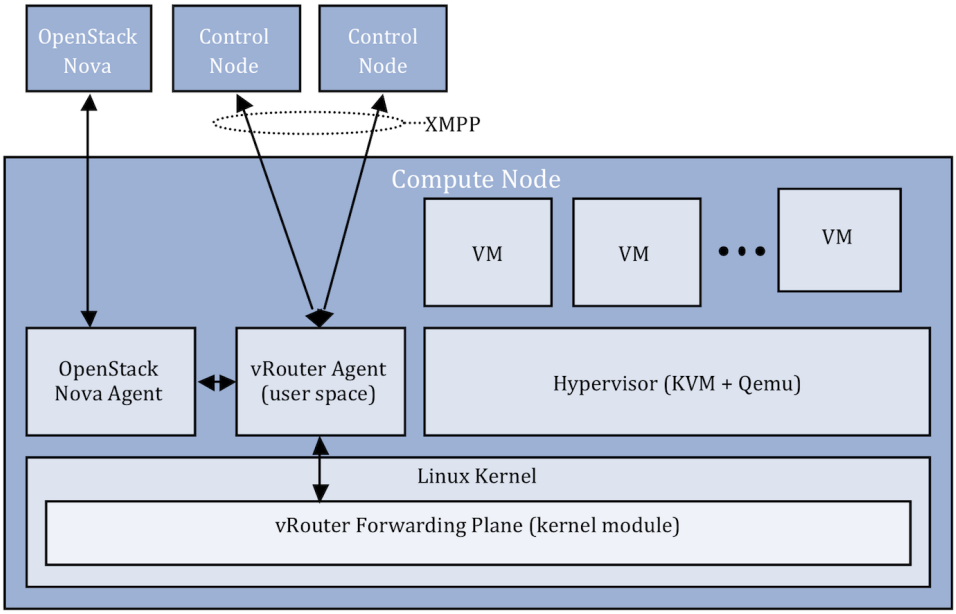

Figure 3 Internal Structure of a Compute Node

Two of the building blocks in a compute node implement a vRouter: the vRouter Agent, and the vRouter Forwarding Plane. These are described in the following sections.

vRouter Agent

The vRouter agent is a user space process running inside Linux. It acts as the local, lightweight control plane and is responsible for the following functions:

- Exchanging control state such as routes with the Control nodes using XMPP.
- Receiving low-level configuration state such as routing instances and forwarding policy from the Control nodes using XMPP.
- Reporting analytics state such as logs, statistics, and events to the analytics nodes.
- Installing forwarding state into the forwarding plane.
- Discovering the existence and attributes of VMs in cooperation with the Nova agent.

- Applying forwarding policy for the first packet of each new flow and installing a flow entry in the flow table of the forwarding plane.
- Proxying DHCP, ARP, DNS, and MDNS. Additional proxies may be added in the future.

Each vRouter agent is connected to at least two control nodes for redundancy in an active-active redundancy model.

vRouter Forwarding Plane

The vRouter forwarding plane runs as a kernel loadable module in Linux and is responsible for the following functions:

- Encapsulating packets sent to the overlay network and decapsulating packets received from the overlay network.
- Assigning packets to a routing instance:
  - Packets received from the overlay network are assigned to a routing instance based on the MPLS label or Virtual Network Identifier (VNI).
  - Virtual interfaces to local virtual machines are bound to routing instances.
  - Doing a lookup of the destination address in the Forwarding Information Base (FIB) and forwarding the packet to the correct destination. The routes may be layer-3 IP prefixes or layer-2 MAC addresses.
- Optionally, applying forwarding policy using a flow table:
  - Match packets against the flow table and apply the flow actions.
  - Optionally, punt the packets for which no flow rule is found (i.e., the first packet of every flow) to the vRouter agent, which then installs a rule in the flow table.
  - Punting certain packets, such as DHCP, ARP, and MDNS, to the vRouter agent for proxying.

Figure 4 shows the internal structure of the vRouter Forwarding Plane.

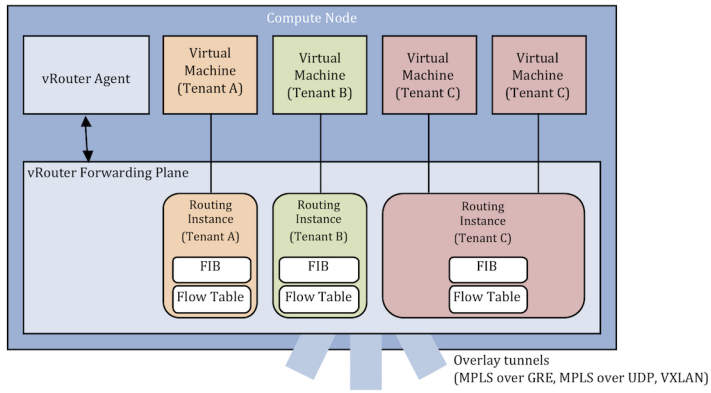

Figure 4 vRouter Forwarding Plane

The forwarding plane supports MPLS over GRE/UDP and VXLAN encapsulations in the overlay network. The forwarding plane supports layer-3 forwarding by doing a Longest Prefix Match (LPM) of the destination IP address, as well as layer-2 forwarding using the destination MAC address. The vRouter Forwarding Plane currently only supports IPv4. Support for IPv6 will be added in the future.

See the section, *Service Chaining*, later in this chapter for more details.

Control Node

Figure 5 shows the internal structure of a control node. The control nodes communicate with multiple other types of nodes:

- The control nodes receive configuration state from the configuration nodes using IF-MAP.
- The control nodes exchange routes with other control nodes using IBGP to ensure that all control nodes have the same network state.
- The control nodes exchange routes with the vRouter agents on the compute nodes using XMPP. They also use XMPP to send configuration state such as routing instances and forwarding policy.
- The control nodes also proxy certain kinds of traffic on behalf of compute nodes. These proxy requests are also received over XMPP.
- The control nodes exchange routes with the gateway nodes (routers and switches) using BGP. They also send configuration state using Netconf.

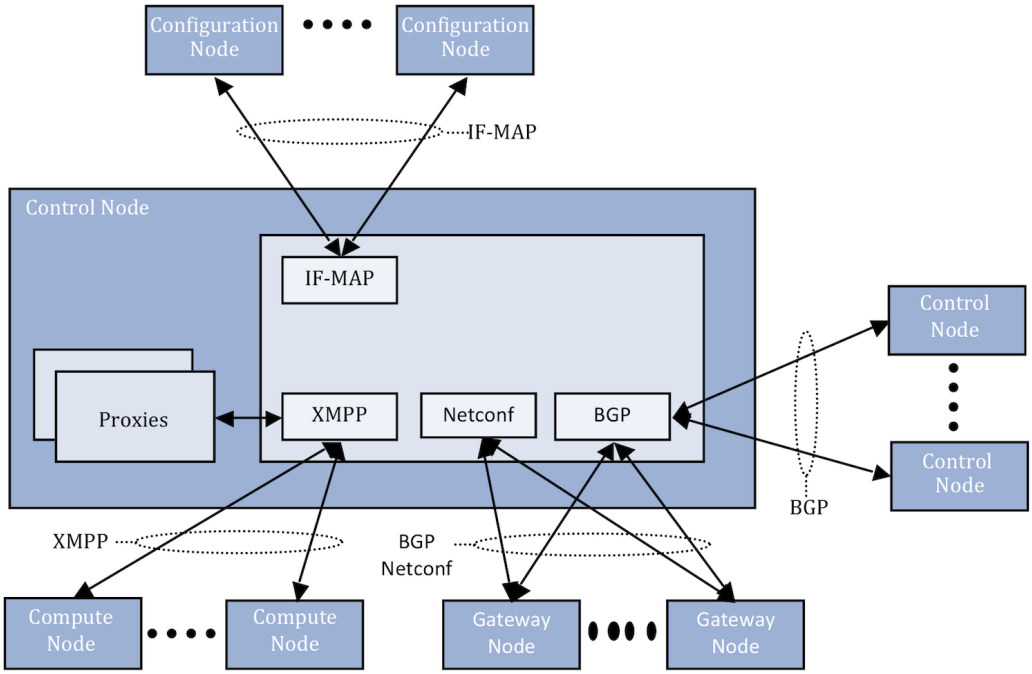

Figure 5 Internal Structure of a Control Node

Configuration Node

Figure 6 shows the internal structure of a configuration node. The configuration node communicates with the Orchestration system via a REST interface, with other configuration nodes via a distributed synchronization mechanism, and with control nodes via IF-MAP.

Configuration nodes also provide a discovery service that the clients can use to locate the service providers (i.e. other nodes providing a particular service). For example, when the vRouter agent in a compute node wants to connect to a control node (to be more precise: to an active-active pair of Control VMs) it uses service discovery to discover the IP address of the control node. The clients use local configuration, DHCP, or DNS to locate the service discovery server.

Configuration nodes contain the following components:

- A REST API Server that provides the north-bound interface to an Orchestration System or other application. This interface is used to install configuration state using the high-level data model.
- A Redis [redis] message bus to facilitate communications among internal components.

- A Cassandra [cassandra] database for persistent storage of configuration. Cassandra is a fault-tolerant and horizontally scalable database.
- A Schema transformer that learns about changes in the high level data model over the Redis message bus and transforms (or compiles) these changes in the high level data model into corresponding changes in the low level data model.
- An IF-MAP Server that provides a southbound interface to push the computed low-level configuration down to the Control nodes.
- Zookeeper [zookeeper] (not shown in Figure 6) is used for allocation of unique object identifiers and to implement transactions.

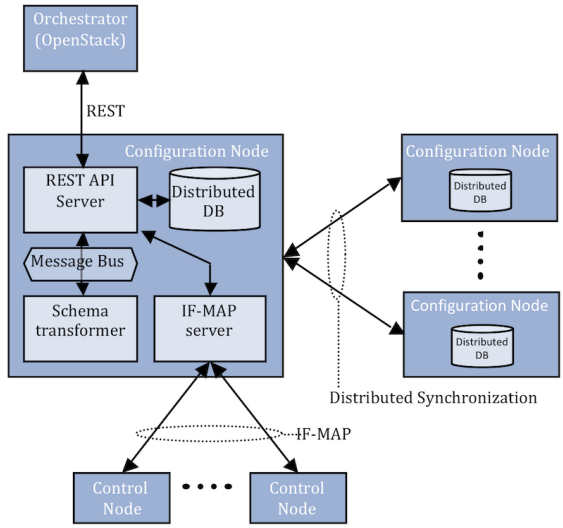

Figure 6 Internal Structure of a Configuration Node

Analytics Node

Figure 7 shows the internal structure of an analytics node. An analytics node communicates with applications using a north-bound REST API, communicates with other analytics nodes using a distributed synchronization mechanism, and communicates with components in control and configuration nodes using an XML-based protocol called Sandesh, designed specifically for handling high volumes of data.

The analytics nodes contain the following components:

- A Collector that exchanges Sandesh messages (see this chapter's section, *Control and Management Plane Protocols/Sangesh*) with components in control nodes and configuration nodes to collect analytics information
- A NoSQL database for storing this information
- A rules engine to automatically collect operational state when specific events occur
- A REST API server that provides a northbound interface for querying the analytics database and for retrieving operational state.
- A Query Engine for executing the queries received over the northbound REST API. This engine provides the capability for flexible access to potentially large amounts of analytics data.

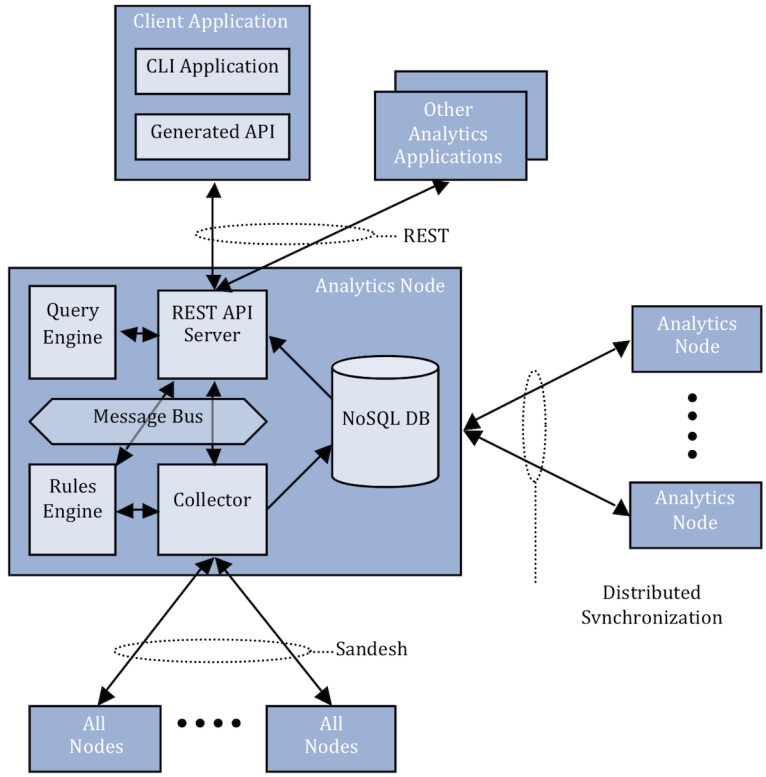

Figure 7 Internal Structure of an Analytics Node

Sandesh carries two kinds of messages: asynchronous messages, received by analytics nodes for the purpose of reporting logs, events, and traces; and synchronous messages, whereby an analytics node can send requests and receive responses to collect specific operational state.

All information gathered by the collector is persistently stored in the NoSQL database. No filtering of messages is done by the information source.

The analytics nodes provide a northbound REST API to allow client applications to submit queries.

Analytics nodes provide scatter-gather logic called "aggregation." A single GET request (and a single corresponding CLI command in the client application) can be mapped to multiple request messages whose results are combined.

The query engine is implemented as a simple map-reduce engine. The vast majority of OpenContrail queries are time series.

The OpenContrail Forwarding Plane

The forwarding plane is implemented using an overlay network. The overlay network can be a layer-3 (IP) overlay network or a layer-2 (Ethernet) overlay network. For layer-3 overlays, initially only IPv4 is supported; IPv6 support will be added in later releases. Layer-3 overlay networks support both unicast and multicast. Proxies are used to avoid flooding for DHCP, ARP, and certain other protocols.

Packet Encapsulations

The system supports multiple overlay encapsulations, each described in detail below.

MPLS over GRE

Figure 8 shows the MPLS over GRE packet encapsulation format for Layer 3 and Layer 2 overlays.

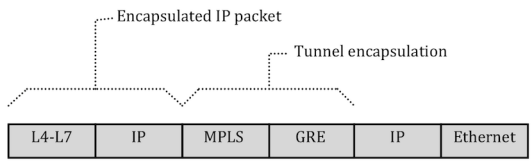

Figure 8 IP Over MPLS Over GRE Packet Format

Figure 9 shows the MPLS over GRE packet encapsulation format for Layer 2 overlays.

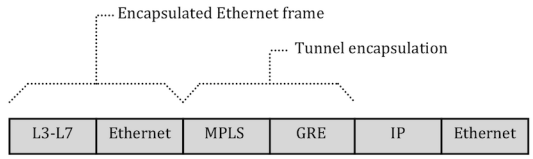

Figure 9 Ethernet Over MPLS Over GRE Packet Format

MPLS Layer 3VPNs [RFC4364] and EVPNs [draft-raggarwa-sajassi-l2vpn-evpn] typically use MPLS over MPLS encapsulation, but they can use MPLS over GRE encapsulation [RFC4023] as well if the core is not MPLS enabled. OpenContrail uses the MPLS over GRE encapsulation and not the MPLS over MPLS for several reasons: *first*, underlay switches and routers in a Data Center often don't support MPLS, *second*, even if they did, the operator may not want the complexity of running MPLS in the Data Center, and *third*, there is no need for traffic engineering inside the Data Center because the bandwidth is overprovisioned.

VXLAN

For Layer 2 overlays, OpenContrail also supports VXLAN encapsulation [draft-mahalingam-dutt-dcops-vxlan]. This is shown in Figure 10.

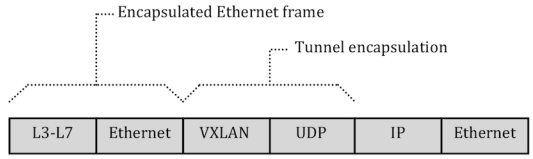

Figure 10 Ethernet Over VXLAN Packet Format

One of the main advantages of the VXLAN encapsulation is that it has better support for multi-path in the underlay by virtue of putting entropy (a hash of the inner header) in the source UDP port of the outer header.

OpenContrail's implementation of VXLAN differs from the VLAN IETF draft in two significant ways. *First*, it only implements the packet encapsulation part of the IETF draft; it does not implement the flood-and-learn control plane, instead it uses the XMPP-based control

plane described in this chapter, and as a result, it does not require multicast groups in the underlay. *Second*, the Virtual Network Identifier (VNI) in the VXLAN header is locally unique to the egress vRouter instead of being globally unique.

MPLS Over UDP

OpenContrail supports a third encapsulation, namely MPLS over UDP. It is a cross between the MPLS over GRE and the VXLAN encapsulation; it supports both Layer 2 and Layer 3 overlays, and it uses an "inner" MPLS header with a locally significant MPLS label to identify the destination routing-instance (similar to MPLS over GRE), but it uses an outer UDP header with entropy for efficient multi-pathing in the underlay (like VLXAN).

Figure 11 shows the MPLS over UDP packet encapsulation format for Layer 3 overlays.

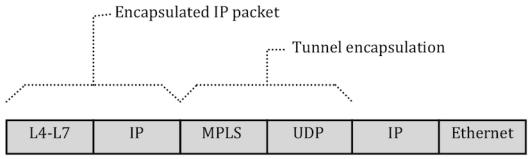

Figure 11 IP Over MPLS Over UDP Packet Format

Figure 12 shows the MPLS over UDP packet encapsulation format for Layer 2 overlays.

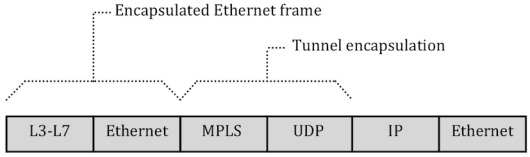

Figure 12 Ethernet Over MPLS Over UDP Packet Format

Layer 3 Unicast

A summary of the sequence of events for sending an IP packet from VM 1a to VM 2a is given below. For a more detailed description see [draft-ietf-l3vpn-end-system].

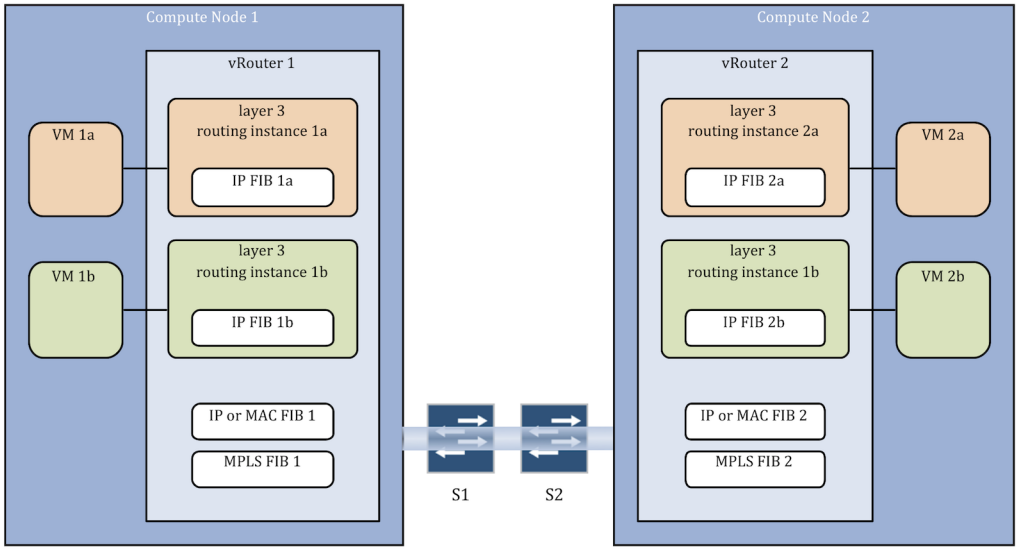

Figure 13 Data Plane: Layer 3 Unicast Forwarding Plane

The following description assumes IPv4, but the steps for IPv6 are similar.

1. An application in VM 1a sends an IP packet with destination IP address VM 2a.

2. VM 1a has a default route pointing to a 169.254.x.x link-local address in routing instance 1a.

3. VM 1a sends an ARP request for that link local address. The ARP proxy in routing instance 1a responds to it.

4. VM 1a sends the IP packet to routing instance 1a.

5. IP FIB 1a on routing instance 1a contains a /32 route to each of the other VMs in the same virtual network including VM 2a. This route was installed by using the control node using XMPP. The next-hop of the route does the following:

- Imposes an MPLS label, which was allocated by vRouter 2 for routing instance 2a.
- Imposes a GRE header with the destination IP address of Compute Node 2.

6. vRouter 1 does a lookup of the new destination IP address of the encapsulated packet (Compute Node 2) in global IP FIB 1.

7. vRouter 1 sends the encapsulated packet to Compute Node 2. How exactly this happens depends on whether the underlay network is a Layer 2 switched network or a Layer 3 routed network. This is described in detail below. For now we will skip this part and assume the encapsulated packet makes it to Compute Node 2.

8. Compute Node 2 receives the encapsulated packet and does an IP lookup in global IP FIB 2. Since the outer destination IP address is local, it decapsulates the packet, i.e., it removes the GRE header which exposes the MPLS header.

9. Compute Node 2 does a lookup of the MPLS label in the global MPLS FIB 2 and finds an entry which points to routing instance 2a. It decapsulates the packet, i.e., it removes the MPLS header and injects the exposed IP packet into routing instance 2a.

10. Compute Node 2 does a lookup of the exposed inner destination IP address in IP FIB 2a. It finds a route that points to the virtual interface connected to VM 2a.

11. Compute Node 2 sends the packet to VM 2a.

Now let's return to the part that was glossed over in step 7: *How is the encapsulated packet forwarded across the underlay network?*

If the underlay network is a Layer 2 network then:

- The outer source IP address (Compute Node 1) and the destination IP address (Compute Node 2) of the encapsulated packet are on the same subnet.
- Compute Node 1 sends an ARP request for IP address Compute Node 2. Compute Node 2 sends an ARP reply with MAC address Compute Node 2. Note that there is typically no ARP proxying in the underlay.
- The encapsulated packet is Layer 2 switched from Compute Node 1 to Compute Node 2 based on the destination MAC address.

If the underlay network is a Layer 3 network, then:

- The outer source IP address (Compute Node 1) and the destination IP address (Compute Node 2) of the encapsulated packet are on different subnets.
- All routers in the underlay network, both the physical router (S1 and S2) and the virtual routers (vRouter 1 and vRouter 2), participate in some routing protocol such as OSPF.

- The encapsulated packet is Layer 3 routed from Compute Node 1 to Compute Node 2 based on the destination IP address. Equal Cost Multi Path (ECMP) allows multiple parallel paths to be used. For this reason the VXLAN encapsulation includes entropy in the source port of the UDP packet.

Layer 2 Unicast

Forwarding for Layer 2 overlays works exactly the same as forwarding for Layer 3 overlays as described in the previous section, except that:

- The forwarding tables in the routing instances contain MAC addresses instead of IP prefixes.
- ARP is not used in the overlay (but it is used in the underlay).

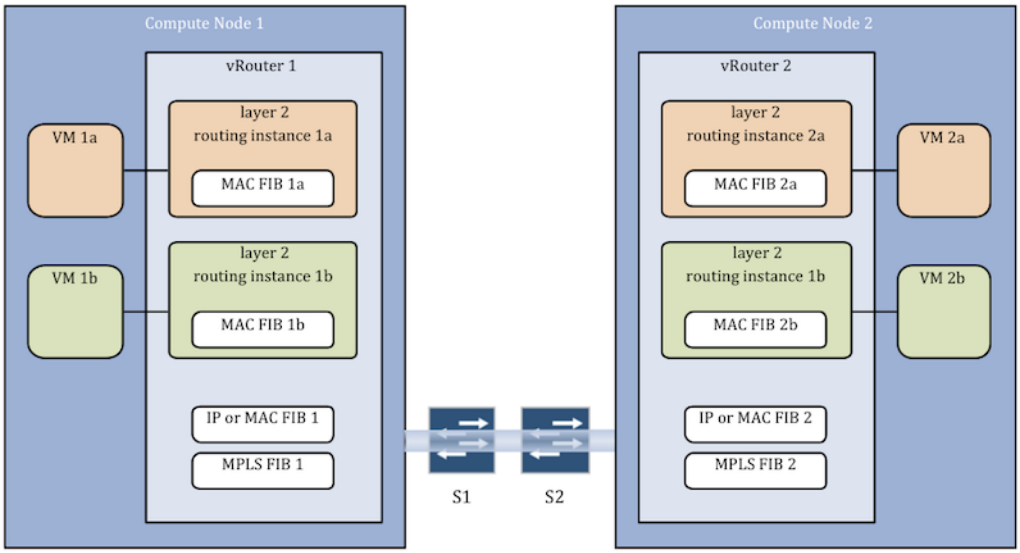

Figure 14 Data Plane: Layer 2 Unicast

Fallback Switching

OpenContrail supports a hybrid mode where a virtual network is both a Layer 2 and a Layer 3 overlay simultaneously. In this case the routing instances on the vRouters have both an IP FIB and a MAC FIB. For every packet, the vRouter first does a lookup in the IP FIB. If the IP FIB contains a matching route, it is used for forwarding the packet. If the IP FIB does not contain a matching route, the vRouter does a lookup in the MAC FIB – hence the name fallback switching.

NOTE The "route first and then bridge" behavior of fallback switching is the opposite of the "bridge first and then route" behavior of Integrated Routing and Bridging (IRB).

Layer 3 Multicast

OpenContrail supports IP multicast in Layer 3 overlays. The multicast elaboration is performed using multicast trees in the overlay or using multicast trees in the underlay. Either way, the trees can be shared (*,G) trees or source specific (S,G) trees.

Overlay Multicast Trees

OpenContrail does multicast elaboration using multicast trees in the overlay instead of the underlays. The details are described in [draft-marques-l3vpn-mcast-edge]; here we only summarize the basic concepts.

Figure 15 illustrates the general concept of creating multicast trees in the overlay. The vRouter at the root of the tree sends N copies of the traffic to N downstream vRouters. Those downstream vRouters send the traffic to more N downstream vRouters, and so on, until all listener vRouters are covered. In this example, N equals 2. The number N does not have to be the same at each vRouter.

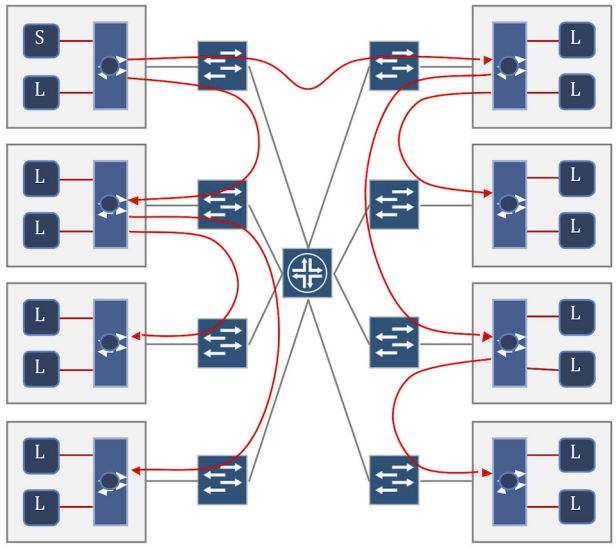

Figure 15 Multicast Tree in the Overlay (General Case)

The system uses XMPP signaling to program the FIBs on each of the vRouters to create the overlay multicast tree(s) for each virtual network. The details of the protocol are too complicated to describe here; see [draft-marques-l3vpn-mcast-edge] for details.

Ingress replication, shown in Figure 16, can be viewed as a special degenerate case of general overlay multicast trees. In practice, however, the signaling of ingress replication trees is much simpler than the signaling of general overlay multicast trees.

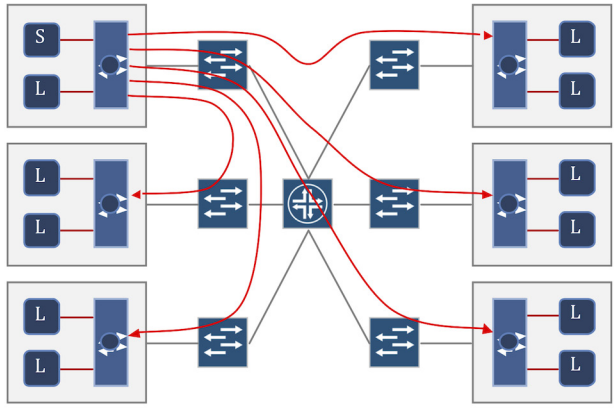

Figure 16 Multicast Tree in the Overlay (Ingress Replication Special Case)

Underlay Multicast Trees

An alternative approach is to do multicast elaboration using multicast trees in the underlay as shown in Figure 17.

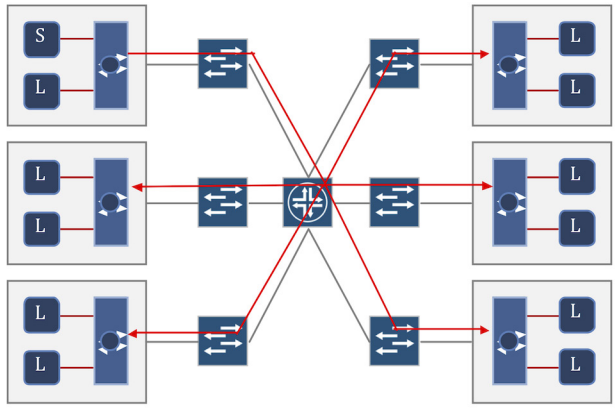

Figure 17 Multicast Tree in the Underlay

This is the approach that is generally used to implement multicast in MPLS Layer 3VPNs (see [RFC6513]). Also, the flood-and-learn control plane for VXLAN described in [draft-mahalingam-dutt-dcops-vxlan] relies on underlay multicast trees.

The underlay multicast tree is implemented as a GRE tunnel with a multicast destination address. This implies that the underlay network must support IP multicast, it must run some multicast routing protocol typically Protocol Independent Multicast (PIM), and it must have one multicast group per underlay multicast tree.

Comparison

Multicast trees in the underlay require IP multicast support on the Data Center switches. In practice this can be a problem for a number of reasons:

- Even if the underlay network supports multicast, the operator may not be willing to enable it due to the complexity of management and troubleshooting.
- Switches based on merchant silicon typically support only a relatively small number of multicast groups in the forwarding plane. For optimal multicast there needs to be one group in the underlay for each group of each tenant in the overlay, which can be a very large number of groups. The number of multicast groups in the underlay can be reduced by using a single shared tree per virtual network, or by sharing a single tree among multiple virtual networks. This comes at the expense of reduced optimality and increased complexity.
- When running multicast in the underlay, the Data Center switches must maintain control plane state about the listeners for each group. The amount of control plane state can be extremely large.
- Multicast control plane protocols can be very CPU intensive because the multicast tree needs to be updated every time a listener joins or leaves.

Overlay multicast trees, on the other hand, also suffer from their own set of problems but

- The first problem is that multiple copies of the same packet are sent over the same physical link, in particular links close to the source. This wastes bandwidth, however, in Data Center networks this is not as big a problem as in WANs because the Data Center fabric is generally a Clos fabric which provides full non-blocking any-to-any connectivity.

- The second problem is that overlay multicast trees put the burden of doing multicast elaboration on the software vRouters which can be CPU intensive. The underlay hardware switches typically have hardware support for multicast elaboration. This problem can be alleviated by spreading the burden over multiple vRouters using cascaded elaboration as shown in Figure 15.

Layer 2 BUM Traffic

Layer 2 Broadcast, Unknown unicast, and Multicast traffic (BUM traffic) needs to be flooded in a Layer 2 network. In OpenContrail, unknown unicast traffic is dropped instead of being flooded because the system does not rely on flood-and-learn to fill the MAC tables. Instead, it uses a control plane protocol to fill the MAC tables and if the destination is not known, there is some other malfunction in the system. Layer 2 broadcasts are also avoided because most Layer 2 broadcasts are caused by a small set of protocols.

For any remaining Layer 2 broadcast and multicast, the system creates one distribution tree per virtual network connecting all routing instances for that virtual network. That tree can be constructed in either the overlay or in the underlay, with the same pros and cons for each approach.

Proxy Services

The vRouter proxies several types of traffic (from the VM) and avoids flooding. The vRouter intercepts specific request packets and proxies them to the control node using XMPP. The control node sends the response back over XMPP.

Currently the system proxies the following types of traffic (additional proxies will be added):

- DHCP request. The Control Node provides the DHCP response based on the configuration of the VM
- ARP requests. The Control Node provides the IP to MAC address mapping
- DNS and MDNS requests. The Control Node provides the name to IP address mapping

Forwarding Policies

The vRouter forwarding plane contains a flow table for multiple different functionality – firewall policies, load balancing, statistics, etc. The flow table contains flow entries that have a match criteria and

associated actions. The match criteria can be a *N*-tuple match on received packets (wildcard fields are possible). The actions include dropping the packet, allowing the packet, or redirecting it to another routing instance. The flow entries are programmed in the forwarding plane by the vRouter Agent.

The flow table is programmed to punt packets to the vRouter Agent for which there is no entry in the flow table. This allows the vRouter agent to see the first packet of every new flow. The vRouter agent will install a flow entry for each new flow and then re-inject the packet into the forwarding plane.

Service Chaining

OpenContrail supports a high-level policy language that allows virtual networks to be connected, subject to policy constraints. This policy language is similar to the Snort [snort] rules language [snort-rules-intro] but that may change as the system is extended. The policy rule looks similar to this:

```
allow   any   src-vn -> dst-vn   svc-1, svc-2
```

This rule allows all traffic to flow from virtual network *src-vn* to virtual network *dst-vn* and forces the traffic through a service chain that consists of service *svc-1* followed by service *svc-2*. In the above example, the rule applies when any virtual machine in virtual network src-vn sends traffic to any virtual machine in virtual network dst-vn.

The system is mostly concerned with traffic steering, i.e., injecting the traffic flows into the right virtual machines using a virtual interface. The virtual machines provide network services such as firewalls, DPI, IDS, IPS, caching, etc.

The system creates additional routing instances for service virtual machines in addition to the routing instances for tenant virtual machines. Traffic is steered:

- By manipulating the route targets for routes to influence importing and exporting routing from one routing instance to another routing instance.
- By manipulating the next-hops and/or the labels of the routes as they are leaked from routing instance to routing instance to force the traffic through the right sequence of routing instances and the right sequence of corresponding virtual machines.

Figure 18 illustrates the general idea on routing instance manipulation for service chaining.

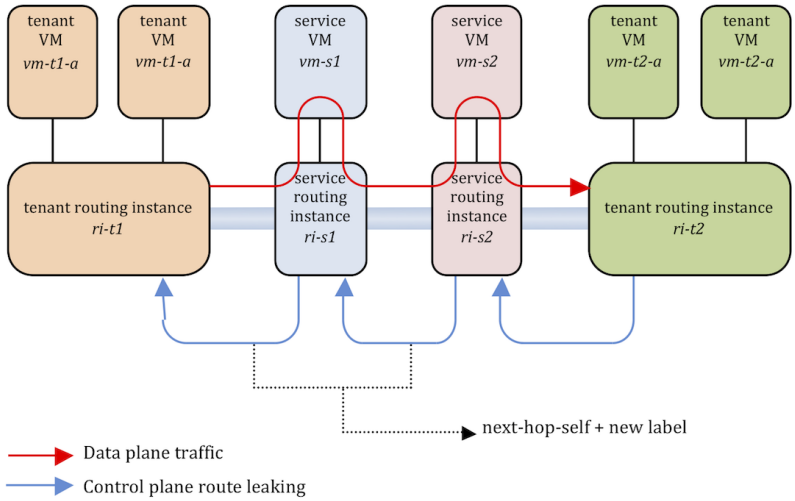

Figure 18 Service Chaining

In the above example:

- The import and export route targets of the routing instances are chosen in such a way that the routes are leaked from routing instance *ri-t2* to *ri-s2*, and then to *ri-s1*, and then to *ri-t1*.
- When the service routing instances export the routes they do a next-hop-self and they allocate a new label:
  - The next-hop-self steers the traffic to the server on which the service is hosted.
  - The label steers the traffic to the service virtual machine on that server.

The IETF draft [draft-rfernando-virt-topo-bgp-vpn] describes a similar mechanism for service chaining.

Control and Management Plane Protocols

IF-MAP

The Interface for Metadata Access Points (IF-MAP) [if-map] is an open standard client / server protocol developed by the Trusted Computer Group (TCG) as one of the core protocols of the Trusted Network Connect (TNC) open architecture.

The original application of IF-MAP was to provide a common interface between the Metadata Access Point (MAP), a database server acting as a clearinghouse for information about security events and objects, and other elements of the TNC architecture.

IF-MAP provides an extensible mechanism for defining data models. It also defines a protocol to publish, subscribe, and search the contents of a data store.

OpenContrail uses the IF-MAP protocol to distribute configuration information from the Configuration Nodes to the Control nodes. Control nodes can use the subscribe mechanism to receive only the subset of configuration in which they are interested. The system also uses IF-MAP to define the high level and low level configuration data models.

XMPP

The Extensible Messaging and Presence Protocol (XMPP) [xmpp] is a communications protocol for message-oriented middleware based on XML. XMPP was originally named Jabber and was used for instant messaging, presence information, and contact list maintenance. Designed to be extensible, the protocol has since evolved into a general publish-subscribe message bus and is now used in many applications.

OpenContrail uses XMPP as a general-purpose message bus between the compute nodes and the control node to exchange multiple types of information including routes, configuration, operational state, statistics, logs and events.

IETF drafts [draft-ietf-l3vpn-end-system] and [draft-marques-l3vpn-mcast-edge] describe the XMPP message formats.

BGP

OpenContrail uses BGP [RFC4271] to exchange routing information amongst the Control nodes. BGP can also be used to exchange routing information between the Control nodes and the Gateway nodes (routers and switches from major networking vendors)

Sandesh

Sandesh is a XML based protocol for reporting analytics information. The structure of the XML messages is described in published schemas. Each component on every node has a Sandesh connection to one of the analytics nodes. Sandesh carries two kinds of messages:

- All components in the system send asynchronous messages to the analytics node to report logs, traces, events, etc.
- The analytics node can send request messages and receive response messages to collect a specific operational state.

OpenStack Integration

Figure 19 shows the integration between OpenStack, Nova, Neutron, and OpenContrail.

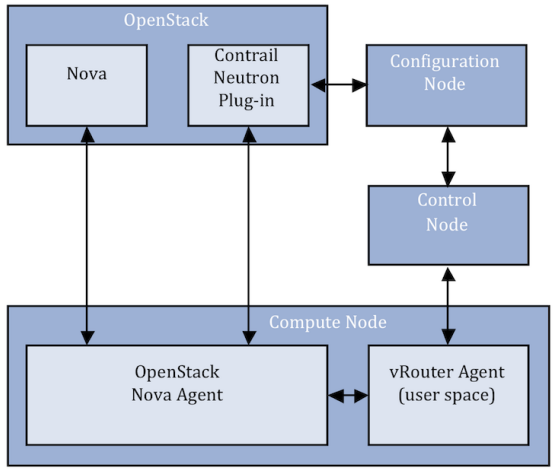

Figure 19 OpenStack Integration

The Nova module in OpenStack instructs the Nova Agent in the compute node to create the virtual machine. The Nova Agent communicates with the OpenContrail Neutron plug-in to retrieve the network attributes of the new virtual machine (e.g. the IP address). Once the virtual machine is created, the Nova Agent informs the vRouter agent, who configures the virtual network for the newly created virtual machine (e.g. new routes in the routing-instance).

Security

All control plane protocols run over Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) to provide authentication and integrity. They can also be used to provide confidentiality although there is typically no need for that in the confines of Data Center.

For the initial service discovery, certificates are used for authentication. For all subsequent communications token-based authentication is used for improved performance. The service discovery server issues the tokens to both the servers and the clients over certificate-authenticated TLS connections.

The distribution of the certificates is out of the scope of this document. In practice this is typically handled by the server management system such as Puppet or Chef.

All REST APIs in the system use role-based authorization. Servers establish the identity of clients using TLS authentication and assigns them one or more roles. The roles determine what operations the client is allowed to perform over the interface (e.g., read-only versus read-write) and which objects in the data model the client is allowed to access.

Horizontal Scalability and High Availability

For high availability as well as for horizontal scaling there are multiple instances of the control nodes, the configuration nodes, and the analytics nodes. All nodes are active-active – OpenContrail does not use the active-standby concept.

Control Nodes

Currently, each control node contains all operational states for the entire the system. For example, all routes for all virtual machines in all virtual networks. The total amount of control state is relatively small to fit comfortably in the memory of each control node. As more features are added, aggregation and sharding of control state across the Control nodes may be introduced in the future using similar principles as route target specific route reflectors in BGP.

Each vRouter agent connects to two or more control nodes where all nodes are active-active. The vRouter agent receives all its state (routes, routing instance configuration, etc.) from each of the control nodes. The state received from the two or more nodes is guaranteed to be eventually consistent but may be transiently inconsistent. It makes a local decision about which copy of the control state to use. This is similar to how a BGP PE router receives multiple copies of the same route (one from each BGP neighbor) and makes a local best route selection.

If a control node fails, the vRouter agent will notice that the connection to that control node is lost. The vRouter agent will flush all state from the failed control node. It already has a redundant copy of all the

state from the other control node. The vRouter can locally and immediately switch over without any need for resynchronization. The vRouter agent will contact the service discovery server again to re-establish a connection with a new control node to replace the failed control node.

Configuration Nodes

The configuration nodes store all configuration state in a fault tolerant and highly available NoSQL database. This includes the contents of the high-level data model, i.e., the configuration state that was explicitly installed by the provisioning system. And it also includes the contents of the low-level data model, i.e., the configuration state that the transformation module derived from the high-level data model.

The Control nodes use IF-MAP to subscribe to just the part of the low-level data model that is needed for the control plane. The service discovery server assigns each control node to a particular configuration node. If that configuration server fails, the control Node re-contacts the service discovery server and is assigned to a different configuration server.

After the switch-over, the control node resynchronizes its state with the new configuration node using the same "mark everything stale, full replay, and flush remaining stale state" approach which is used for graceful restart in routing protocols.

Analytics Nodes

The system provides complete high-availability functionality for all the analytics components, similar to the configuration nodes. The analytics nodes are stateless, hence failure of the analytics components do not cause the system to lose messages. When an analytics node goes down, the system's discovery services will transition the effected generators to a functioning analytics node. And the upstream REST API clients can also use discovery service to detect the failure of a node and transition to a functioning node. The failed analytics node is taken out of the pool of available nodes and one of the remaining analytics nodes takes over the work of collecting data and handling queries.

OpenContrail provides complete high availability functionality across the database components. The database cluster is set up in a multiple replication manner, hence the data itself will be resilient to database node failures. The cluster will be resilient to multiple database node failures. Upon failure of a database node, the analytics nodes will smoothly transition from the failed node to a functioning node. During

this process, we will queue up the data and hence during this transition the data loss will be very minimal.

vRouter Agent

vRouter high availability is based on the graceful restart model used by many routing protocols including BGP. If the vRouter agent restarts for any reason (crash, upgrade) the vRouter forwarding plane continues to forward traffic using the forwarding state which was installed by the vRouter agent prior to the restart.

When the vRouter agent goes down the vRouter forwarding plane is running in a headless mode. All forwarding state is marked as stale. When the vRouter agent restarts, it re-establishes connections to a pair of redundant Control nodes. It re-learns all state from the Control nodes and re-installs the fresh state in the forwarding plane replacing the stale state.

When the vRouter agent finishes re-learning the state from the control nodes and completes re-installing fresh state in the forwarding plane, any remaining stale state in the forwarding plane is flushed.

vRouter Forwarding Plane

If the vRouter forwarding plane restarts for any reason (crashes, upgrades) there will be an interruption in traffic processing for that particular server.

This is unavoidable because there is only a single instance of the vRouter forwarding plane on each router. It is important to keep the vRouter forwarding plane as simple as possible to minimize the probability of crashes or upgrades.

Chapter 3

The Data Model

Underlying all states in the system, whether configuration, operational or analytics, is a set of data models. Each data model defines a set of objects, their semantics, and the relationships between them. The system operates on these data models to perform its tasks: creating and updating objects and relationships, translating "high-level" objects into "low-level" objects, and instantiating low-level objects in networking elements to create the required connectivity and services. These data models offer certain capabilities to the modules that manipulate them, and in turn impose certain requirements on them. The main result of this data model-based design is that the system is distributed, scalable, highly available, easily upgradable, and elastic.

Data models are essentially annotated graphs with vertices that represent objects and links that represent relationships between objects, and the system uses a Data Modeling Language (DML) to specify them. Some of the semantics of objects and the relationships between them are captured directly in the data model, for example, a vertex in the graph may represent an abstract or concrete object, and a link may represent a parent-child relationship or a dependency between a pair of objects. The remaining semantics are captured in the annotations on vertices and links, for example, a link that represents connectivity between a pair of vertices may be annotated with the required bandwidth, or a link between routing instances may be annotated with the desired routing policy.

Programming Model

The data model for configuration and operational state is defined using IF-MAP, with the data itself being kept in a Cassandra database. This database provides persistence, availability, and scale-out characteristics. A "pub-sub" bus is overlaid on top using Redis as the in-memory key-value store. Modules that interact with the database may choose to subscribe to certain types of updates. When a module publishes an update to an object, the update is sent to all other modules that subscribe to that type of object.

All modifications to a data model must be backward compatible: in other words, any program that references a data model must continue to work as before without requiring modification. This means that all changes to a data model may only extend existing objects, links and annotations, but must never change the semantics of, or the relations between, existing objects. Further, modifications must never deprecate an object or link type. An example of how a relation between objects can be made backward compatible is shown below. A further requirement is that modifications to a data model must be incremental to allow changes to be pushed out to running modules without requiring recompilation.

Access to a data model is via a RESTful API that is auto-generated from the model's specification. In addition, bindings for various languages (Python, Java, etc.) are also generated, allowing programmers in these languages to manipulate data in the model.

Modules that operate on these data models must be event-driven. This means two things: first, a module must listen for updates on the pub-sub bus; and second, when it gets all the information it needs for a particular action, it performs the action. The updates may come in any order—the pub-sub bus does not guarantee the order updates or manage dependencies—that is the responsibility of each module. Furthermore, a module must be restart-able. If it crashes, or is forcibly restarted (e.g., upgraded), it simply reconnects to the database, reacquires its state and continues processing. To do this, a module must keep all non-temporary state either in the database via data models or "in the network", in which case the module reacquires state from its peers. Finally, modules must be elastic. That is, they must be able to work in a distributed fashion; instances may be spawned or terminated as determined by the current load. This is accomplished by having a distributed database hold the data model, and by having a service for coordination of distributed entities (e.g., for unique ID allocation).

Configuration and Operational Data Model

This data model consists of an object hierarchy that is represented as a rooted, annotated Directed Acyclic Graph (DAG). The vertices in the DAG represent objects that may be administrative entities, or physical or logical resources. Links in the DAG represent relationships between objects. An object may have zero or more children, but only one parent (except the root, which has no parents). Deleting an object implicitly deletes the subtree below it. An object may refer to another object, in which case a "reference count" is maintained. Deleting the referring object decrements the reference count. Deleting an object that has outstanding references is not allowed. "Weak references" to objects that do not exist as yet are permitted. A weakly referred to object may be deleted.

As shown in Figure 20, the root of the DAG represents the universe of objects that the system is responsible for, i.e., an "administrative domain" (AD). This might be a Data Center cluster, a Point Of Presence (POP), a Central Office (CO), or a Wide Area Network (WAN). An AD has an overall administrator that is responsible for managing it. An AD also has an associated namespace for all identifiers that may be used within it. Examples of identifiers are IP addresses, domain names, and routing instance IDs.

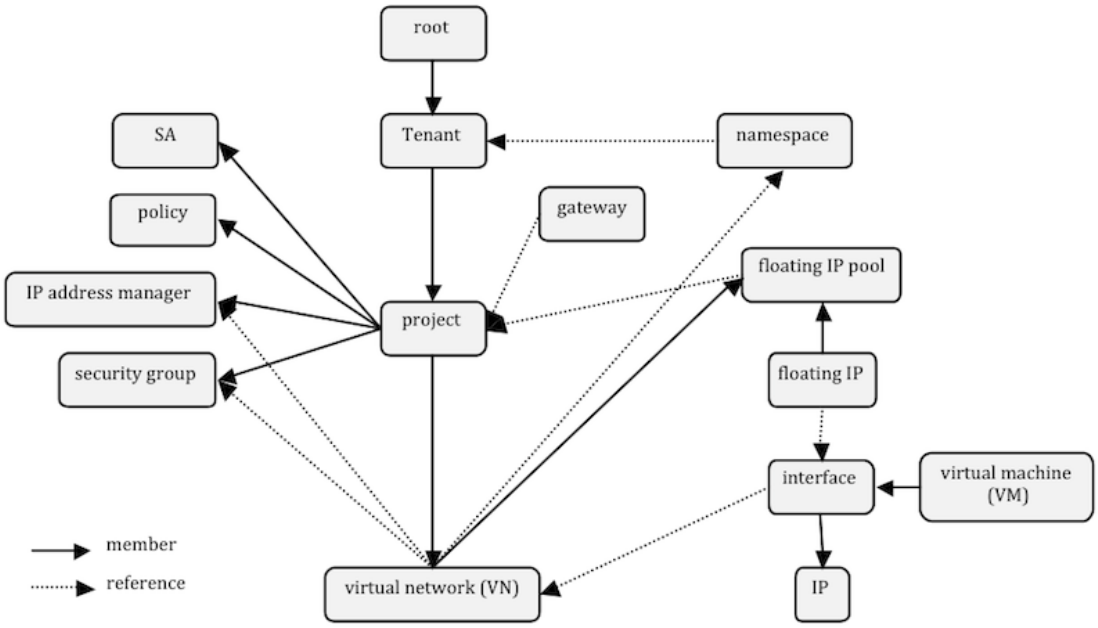

Figure 20 OpenContrail System High Level Data Model Definition

An AD contains one or more "tenants" or "domains." For example, tenants in a DC may be Coke and Pepsi; tenants in a POP or CO may be the service departments for business customers and broadband.

A tenant may contain a number of projects, or a department within a tenant, say Marketing is an example of a project. Projects contain Virtual Networks (VNs). A VN in a DC may consist of virtual machines in an application tier. A VN in a POP may represent a VPN for a business customer, or a grouping of mobile subscribers that have a similar profile, or a grouping of users in a Campus environment.

A project contains service instances, gateways, and policies. A project also has Security Groups that administrators can use to assign "roles" to endpoints and these roles may define further policies for endpoints.

VNs contain endpoints. In the DC domain, these are virtual machines; for the business edge, these are customer sites (CEs), and for wired/wireless edge they are subscribers. Endpoints that are part of a Security Group have a reference to that Security Group.

Other objects in the data model are "routing instances" (RIs). A VN is "compiled" into one or more RIs, and is implemented as such in vRouters. Yet other objects are Route Targets, which are used to control routing leaking between RIs.

This hierarchy was developed in the context of a Data Center, but it appears to be sufficiently general to encompass other domains. In some cases, certain levels in the hierarchy may be redundant. In this case, a single instance can be used at that level to maintain the hierarchy. For example, if the notion of tenant is not needed, then a single tenant can be instantiated to retain the tenant level in the hierarchy. If a new level is needed in the hierarchy, this needs to be introduced in a backward compatible manner.

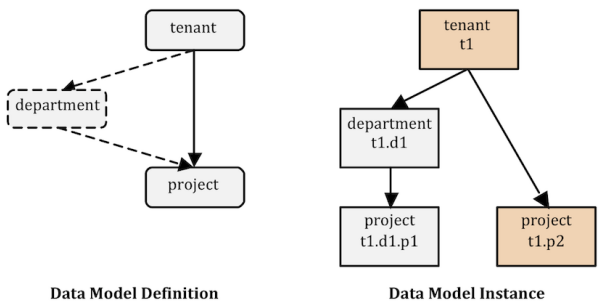

Figure 21 Data Model Extensibility

Suppose a new level in the hierarchy, *department,* is desired between *tenant* and *project*. One can easily extend the data model to add this. However, the requirement for backward compatibility means three things:

1. Department must be an optional level in the hierarchy. That is, it must be possible to create a project immediately under a tenant as well as under a department.

2. A project that was created as a child of a tenant must remain as that tenant's child until it is deleted.

3. A new project can either be the child of a tenant or of a department, but not both.

Also, an application that asks for children of a tenant must be prepared to receive children of types other than *project*. It can simply ignore them, but it must cause an error or otherwise fail.

Figure 21 above on the left shows the data model for *tenant* and *project* with a parent-child relationship. The dotted object and links represent a newly added level for *department.* The diagram on the right shows an instance of the data model with a tenant *t1* and a project *t1.p2* in orange; these follow the original data model. Also shown is a department *t1.d1* and another project *t1.d1.p1*; these follow the modified data model. Note that each project has only one parent: the orange project has *t1* as its parent, and the other project has *t1.d1* as its parent.

High-level and Low-level Data Models

All the above objects and relations are in a common data model. There is a loose distinction of objects into "high-level" and "low-level." Objects created by entities outside the system (the orchestration system or an application) are considered configuration state, and are thus part of the high-level data model. Examples include tenants and projects. Objects generated by the modules are operational state, and are considered part of the low-level data model because they are closer to the abstractions used inside network elements; examples include routing instances. Some objects may be either configured or generated, blurring the line between high and low levels of the data model; an example is a Route Target, which is typically generated, but may also be configured so the system can talk to external BGP speakers.

In addition, some physical entities are represented in the data model. For example, every instance of a vRouter on a server is represented. Every BGP speaker of interest to the system (those in Control nodes, as well as those to which OpenContrail peers, such as a DC gateway) is

also represented. Finally, physical and virtual service appliances are also represented. These objects are used to track BGP or XMPP peering sessions and the service connectivity between them.

Service Connectivity Data Model

A service "chain" is the specification of traffic flows between a pair of VNs. Traffic from a VN may go through an arbitrary graph of service nodes before reaching another VN. The traffic may take different paths based on a service (for example, an IDS may redirect traffic to a DPI engine), or be replicated for monitoring purposes, for example. Although the term "service chain" does not cover these more general cases, which should more correctly be called "service graphs," *service chain* will be used.

The specification of a service chain consists of two parts: the traffic path from an ingress VN to a set of service elements to an egress VN, and the service profile to apply within each service element. At present, both of these specifications are annotations to objects. The former as an annotation to a policy, and the latter as an annotation to a service object. It might be beneficial to capture the former in an explicit data model; this data model would be a graph whose vertices are services and whose links are connections between services. This graph would then be inserted between the pair in the system.

Chapter 4

OpenContrail Use Cases

There are three immediate use-cases for OpenContrail: (a) Private Cloud for the Enterprise, (b) Infrastructure as a Service and Virtual Private Cloud for Service Providers, and (c) Network Function Virtualization for Service Provider Networks. The goal is not to provide an exhaustive list of use cases, but to provide an illustrative sampling of use cases.

Data Center Domain Use Cases

Before diving into the specific use cases for the Data Center, it is useful to first discuss the role of orchestration in the Data Center.

The Role of Orchestration in the Data Center

In the Data Center, the orchestrator (Openstack, CloudStack, VMware, Microsoft System Center, etc.) manages many critical aspects of the Data Center:

- Compute (virtual machines)
- Storage
- Network
- Applications

The Software Defined Networking (SDN) controller's role is to orchestrate the network and networking services like Load Balancing and Security based on the needs of the application it's assigned compute and storage resources.

The orchestrator uses the northbound interface of the SDN controller to orchestrate the network at a very high level of abstraction, for example:

- Create a virtual network for a tenant, within a Data Center or across Data Centers.
- Attach a VM to a tenant's virtual network.
- Connect a tenant's virtual network to some external network, e.g., the Internet or a VPN.
- Apply a security policy to a group of VMs or to the boundary of a tenant's network.
- Deploy a network service (e.g. a load balancer) in a tenant's virtual network.

The SDN controller is responsible for translating these requests at a high level of abstraction into concrete actions on the physical and virtual network devices such as:

- Physical switches, e.g., Top of Rack (ToR) switches, aggregation switches, or single-tier switch fabrics.
- Physical routers.
- Physical service nodes such as firewalls and load balancers.
- Virtual services such as virtual firewalls in a VM.

Virtualized Multi-tenant Data Center

The virtualized multi-tenant Data Center use case allows multiple tenants to be hosted in a Data Center. Multi-tenancy means that tenants share the same physical infrastructure (servers, network, storage) but are logically separated from each other.

The concept of a tenant can mean different things in different circumstances:

- In a service provider Data Center providing public cloud services it means a customer or applications belonging to a customer.
- In an enterprise Data Center implementing a private cloud, it could mean a department or applications belonging to a customer.

The number of tenants is important because some architectural approaches (specifically native end-to-end VLANs) have a limit of 4096 tenants per Data Center.

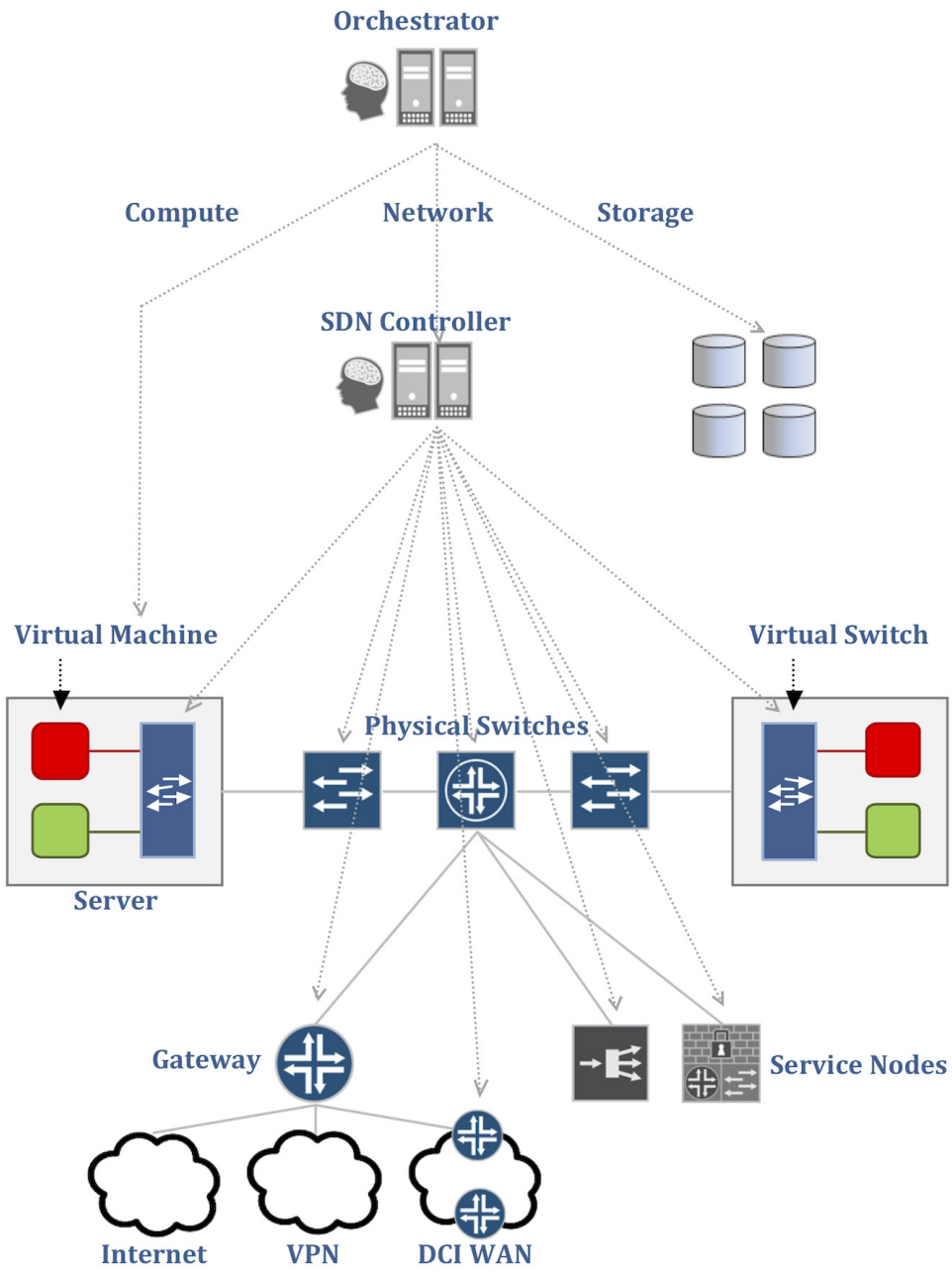

Figure 22 The Role of Orchestration in the Data Center

Not all Data Centers are multi-tenant. Some large content providers (e.g. Facebook) have private Data Centers that are only used for internal applications and not yet for providing Cloud services. Even those Data Centers that do support multi-tenancy do not all define multi-tenancy in the same way. For example, the original Amazon Web Services (AWS) [AWS] did support multi-tenancy but from a networking point of view the tenants were not logically separated from each other (all tenants were connected to the same Layer 3 network). Since then Amazon has introduced a more advanced service called Virtual Private Cloud (VPC) [AWS-VPC] which does allow each tenant to get one or more private isolated networks.

Figure 23 shows the virtualization and multi-tenancy requirements for various market segments.

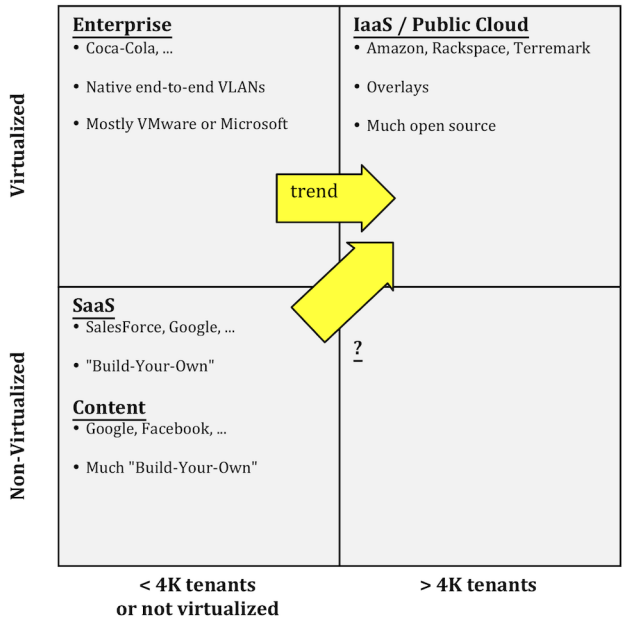

Figure 23 Multi-Tenancy Requirements

Where virtualization is used, different market segments tend to use different orchestrators and hypervisors:

- For the enterprise market segment, commercial orchestration systems are widely used. With the growing adoption of the Cloud and movement towards software defined Data Center, there is a

desire to adopt an integrated open source stack such as OpenStack or CloudStack

- In the Infrastructure as a Service (IaaS) and public cloud market, open source orchestrators (e.g. OpenStack, CloudStack) and hypervisors (e.g. KVM, Xen) are often used for customizability, cost, and scalability reasons.
- Very large content providers (e.g. Google and Facebook) often build their own orchestration software and don't use hypervisors for performance and scale reasons.

Generally, each tenant corresponds to a set of virtual machines hosted on servers running hypervisors as shown in Figure 24. The hypervisors contain virtual switches ("vSwitches") to connect the virtual machines to the physical network and to each other. Applications may also run "bare-metal" on the server (i.e. not in a virtual machine) as shown in the green server (B) in the lower right corner of Figure 24.

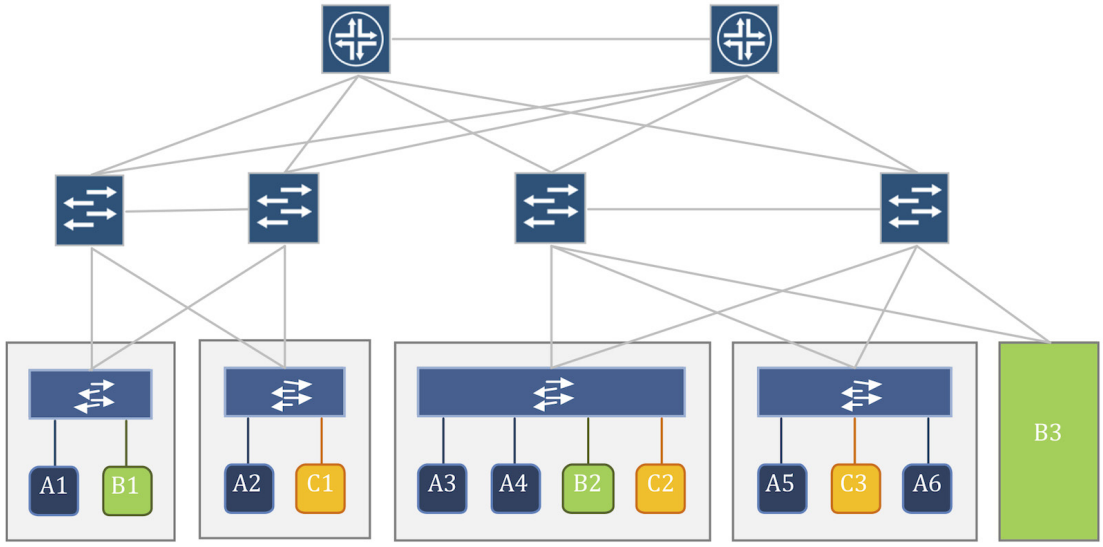

Figure 24 Use Case for Multi-Tenant Virtualized Data Center (Multi-Tier Data Center Network)

The Data Center network may be a multi-tier network as shown in Figure 24, or the Data Center may be single-tier network (e.g. Fabric) as shown in Figure 25.

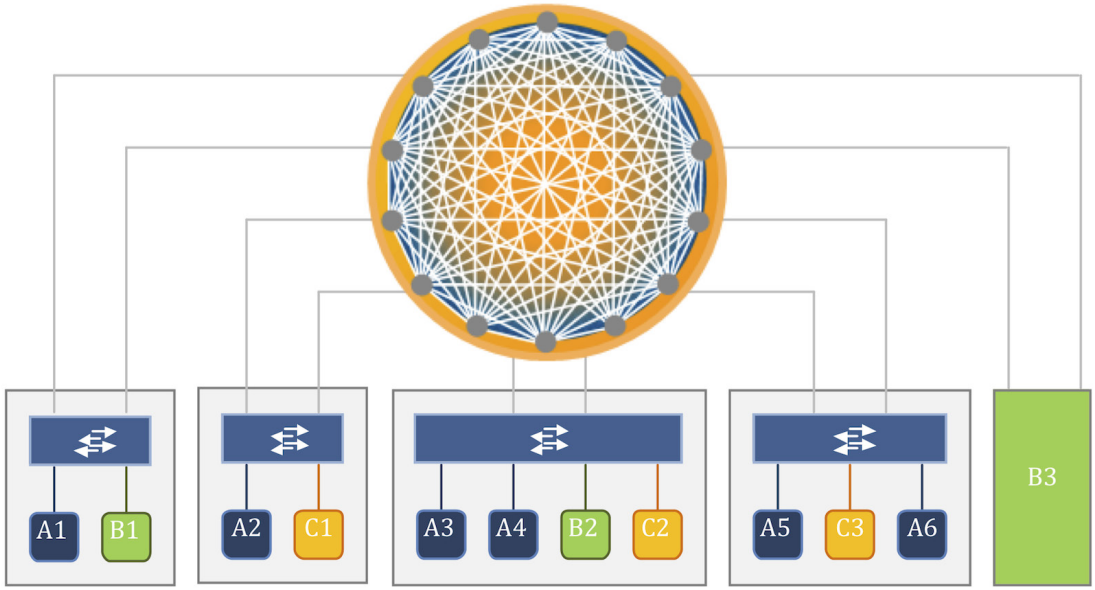

Figure 25 Use Case for Multi-Tenant Virtualized Data Center (SINGLE-Tier Data Center Network

The servers are interconnected using a physical Data Center network. In Figure 25 the network is depicted as a two-tier (access, core) network. It could also be a three-tier (access, aggregation, core) network or a one-tier (e.g., Q-Fabric) network. For overlay solutions the Data Center network is recommended to be a Layer 3 network (IP or MPLS).

In the simplest scenario, shown in Figure 26, the cloud provider assigns an IP address to each virtual machine. The virtual machines of a given tenant are not on the same Level 2 network. All virtual machines (whether from the same tenant or from different tenants) can communicate with each other over a routed IP network.

For example, in Amazon Web Services [AWS] the Elastic Compute Cloud (EC2) [AWS-EC2] by default assigns each virtual machine one private IP address (reachable from within the Amazon EC2 network) and one public IP address (reachable from the Internet via NAT) [AWS-EC2-INSTANCE-ADDRESSING]. Amazon dynamically allocates both the private and the public IP address when the VM is instantiated. The Amazon EC2 Elastic IP Address (EIP) feature [AWS-EC2-EIP] assigns a limited (default five) number of static IP addresses (to a tenant that can be assigned to VMs) that are reachable from the Internet.

Figure 26 One Big Layer 3 Network (Not Part of the Multi-Tenant Use Case)

In order to isolate the tenants from each other in a network, each tenant can be assigned a private Layer 2 network as shown in Figure 27. The tenant's network allows each virtual machine to communicate with all of the other virtual machines of the same tenant, subject to policy restrictions. The tenant networks are isolated from each other: a virtual machine of one tenant is not able to communicate with a virtual machine of another tenant unless specifically allowed by policy. Also, the virtual machines are not reachable from the Internet unless specifically allowed by policy.

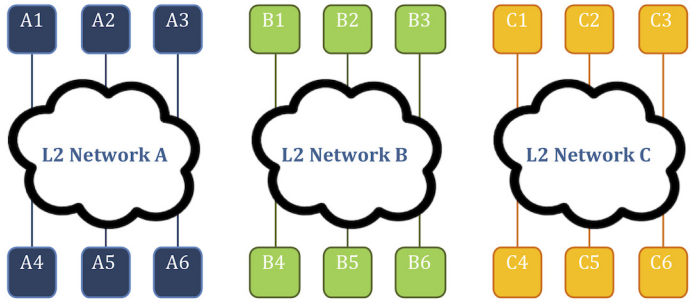

Figure 27 Network Abstraction Presented to Tenants

The tenant private networks are generically called virtual networks; all virtual machines on a given tenant network are on the same Layer 3 subnet. The tenant may be allowed to pick his own IP addresses for the VMs or the cloud provider may assign the IP addresses. Either way, the IP addresses may not be unique across tenants (i.e. the same IP address may be used by two VMs of two different tenants).

A single tenant may have multiple virtual networks. Those virtual networks may or may not be connected to each other using a Layer 3 router, a firewall, a NAT, a load balancer, or some other service.

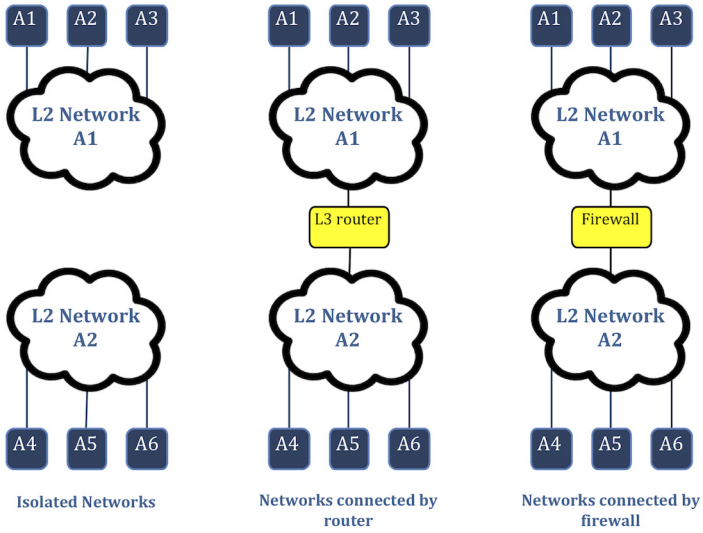

Figure 28 Multiple Networks for a Tenant

As an example of isolated virtual tenant networks, the Amazon Virtual Private Cloud (VPC) service [AWS-VPC] allows tenants to create one or more subnets and to connect them to each other, or to the Internet, or to a customer network using routers or services (e.g. NAT). [AWS-VPC-SUBNETS]

The use case includes a logically centralized orchestration layer (not shown in any of the diagrams above) for the management of tenant networks:

- adding and removing tenants,
- adding and removing virtual machines to and from tenants,
- specifying the bandwidth, quality of service, and security attributes of a tenant network, etc.

This orchestration layer must cover all aspects of the Data Center (compute, storage, network, and storage) and support a high rate of change.

Connect Tenant to Internet / VPN

In this use case, tenants connect to the Internet or the Enterprise network via a VPN as shown in Figure 29. The VPN can be a Layer 3VPN, Layer 2VPN, an SSL VPN, an IPsec VPN, etc.

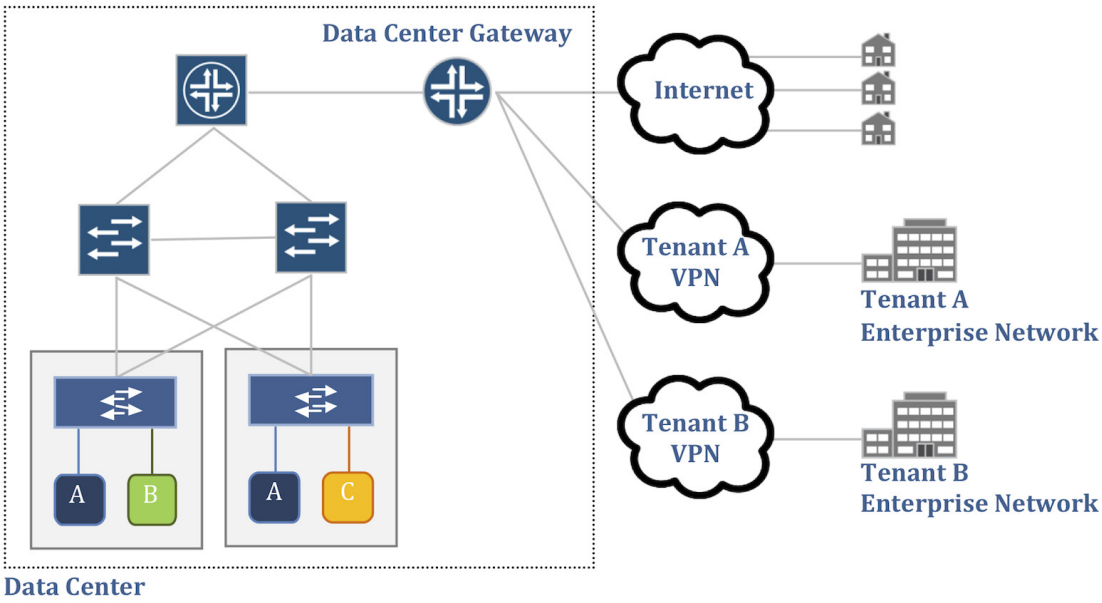

Figure 29 Use Case for Connect Tenant to Internet/VPN

The Data Center gateway function is responsible for connecting the tenant networks to the Internet or the VPNs. The gateway function can be implemented in software or in hardware (e.g. using a gateway router).

Data Center Interconnect (DCI)

In this use case multiple Data Centers are interconnected over a Wide Area Network (WAN) as illustrated in Figure 30.

Data centers may be active/standby for disaster recovery, temporarily active/active for disaster avoidance, or permanently active/active. In the active/active case a tenant may have virtual machines in multiple Data Centers. The Data Center Interconnect (DCI) puts all VMs of a given tenant across all Data Centers on the same virtual tenant network.

DCI must address the following network requirements:

- Storage replication.
- Allow tenant networks to use overlapping IP address spaces across Data Centers.
- Global Load Balancing (GLB).
- VM migration across Data Centers for disaster avoidance

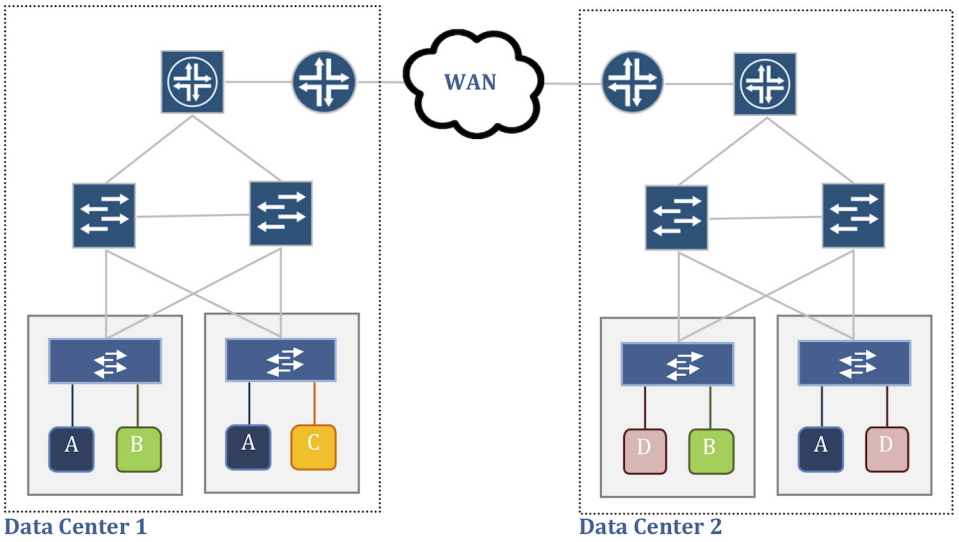

Figure 30 Use Case for Center Interconnect (DCI)

Multiple transport options are available for DCI interconnect, including dark fiber, SONET/SDH, DWDM, pseudo-wires, Layer 3 VPNs, E-VPNs, etc. Unlike the Data Center network, bandwidth is a scarce resource in the DCI WAN, so Traffic Engineering (TE) is often used to use available resources efficiently.

Network Monitoring

In Data Center networks it is often necessary to make a copy of specific flows of traffic at specific points in the network and send that copy of the traffic to one or more monitoring devices for further analysis. This is referred to as the network monitoring or tap use case.

The monitoring may be temporary, for example, for when debugging network issues. Or the monitoring may be permanent, for example, for regulatory compliance reasons.

Traditionally monitoring is implemented by manually configuring the Switched Port Analyzer (SPAN) feature on the switches in the network to send a copy of traffic flows to a specific port. Remote SPAN (RSPAN) is a more sophisticated version of the feature; it allows the copied traffic flow to be sent into a GRE tunnel for remote analysis.

A centralized SDN system can be used to:

- Create tunnels from the monitoring collection points (the "taps") in the network to the monitoring devices, which collect and analyze the traffic.
- Instruct the switches and routers in the network to steer particular flows of traffic into those tunnels for analysis.

Dynamic Virtualized Services

In this use-case, networking services such as firewalls, Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), load balancers, SSL off-loaders, caches, and WAN optimizers are deployed in tenant networks.

These services are provided by Service Nodes that can be located in various places as shown in Figure 31.

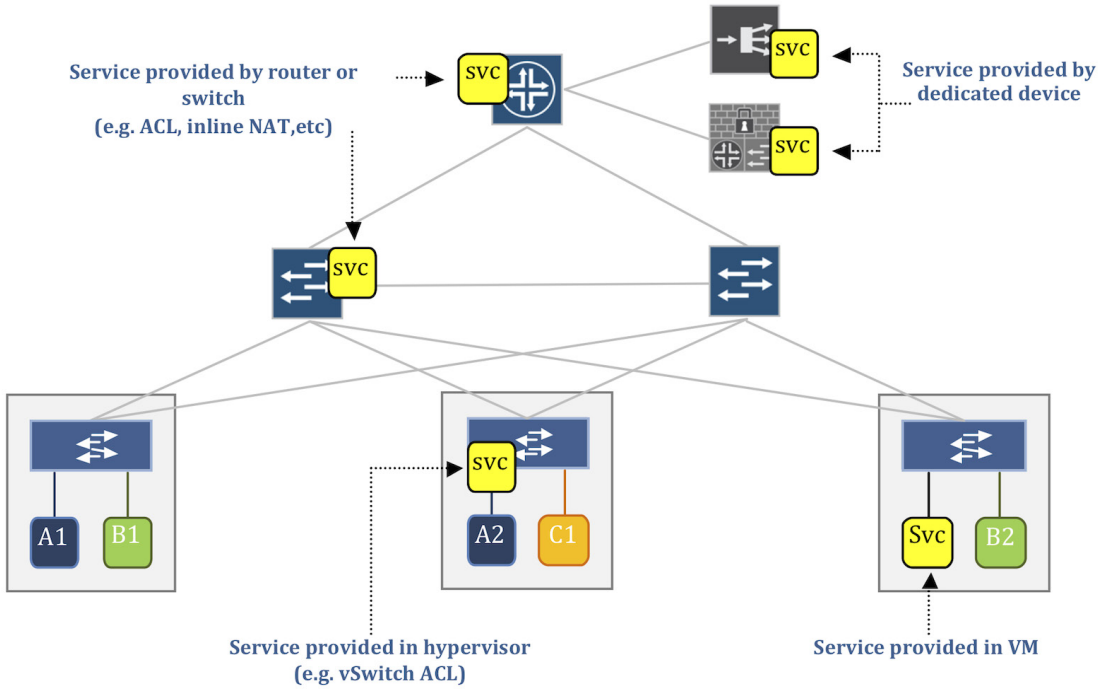

Figure 31 Service Node Locations

Services may be deployed in multiple locations:

- In the hypervisor
- In a virtual machine
- In a physical device
- Using Access Control Lists on the physical access switch or the vSwitch
- In-line services on a router or switch on a service card or natively on the forwarding ASICs

Services may be associated with one or more VMs, for example as a result of attaching a security policy to one or more VMs.

Alternatively, services may be associated with network boundaries, for example by attaching a security policy to a network boundary or by inserting a load balanced at a network boundary. As shown in Figure 32, this network boundary may be:

- The boundary between a tenant network and an external network (the Internet or the VPN to the enterprise network).
- The boundary between the network of one tenant and the network of another tenant.
- The boundary between multiple networks of the same tenant.

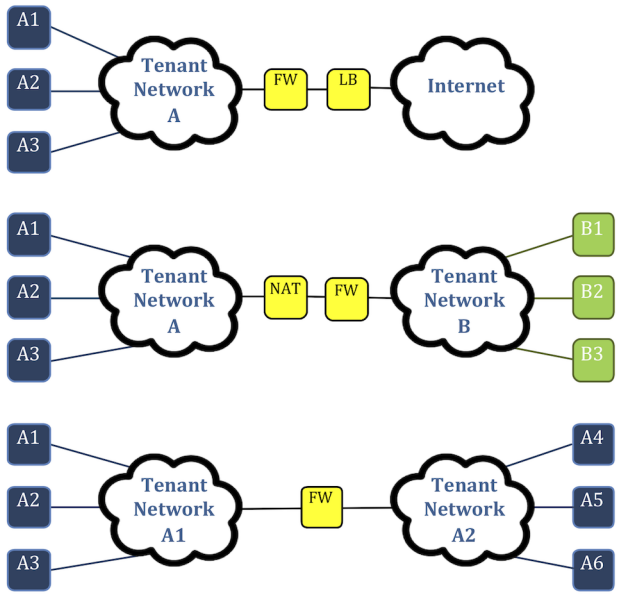

Figure 32 Services at Network Boundaries

Network Functions Virtualization for SP Networks

Service Insertion

An edge router wants to apply some services (firewall, DPI, caching, HTTP header enrichment, etc.) to traffic from subscribers. Those services may be provided by a service card in the router, or by physical service appliances, or by virtual service appliances in the Cloud.

The SDN system is used to create and manage virtualized or physical services and create service chains to steer subscriber traffic through these services. This can be done based on local configuration but it is more typically done using a centralized policy servicer.

Service Example – Virtualized CPE (vCPE)

In Broadband Subscriber Networks (BSN) each subscriber is provided with a Customer Premises Equipment (CPE) such as a multi-services router. Operators need more functionality in these CPEs to compete with Over The Top (OTT) services but are challenged to do so because:

- CPE vendors are slow to add new features and truck rolls for hardware feature additions or replacements are expensive.
- Many different CPE devices are present in a network that leads to inconsistent feature support.

In the Virtual CPE use case (also known as the Cloud CPE use case) the operator addresses these problems by:

- Using a simplified CPE device, which only implements basic layer-2/layer-3 functionality.
- Virtualizing the remaining functionality in a virtual machine or container running on common x86 hardware that is centrally orchestrated and provisioned.

The servers hosting the virtualized CPE functionality may be located in different places:

- Tethered to the Broadband Network Gateway (BNG)
- On a service card in the BNG
- In-line between the BNG and the CPE
- In a Data Center
- A combination of the above

Chapter 5

Comparison of the OpenContrail System to MPLS VPNs

The architecture of the OpenContrail System is in many respects similar to the architecture of MPLS VPNs (Another analogy [with a different set of imperfections] is to compare the Control VM to a routing engine and to compare a vRouter to a line card) as shown in Figure 33.

The parallels between the two architectures include the following:

- Underlay switches in the OpenContrail System correspond to P routers in an MPLS VPN. Since the OpenContrail System uses MPLS over GRE or VXLAN as the encapsulation protocol there is no requirement that the underlay network support MPLS. The only requirement is that it knows how to forward unicast IP packets from one physical server to another.
- vRouters in the OpenContrail System correspond to PE routers in an MPLS VPN. They have multiple routing instances just like physical PE routers.
- VMs in the OpenContrail System correspond to CE routers in an MPLS VPN. In the OpenContrail System there is no need for a PE-CE routing protocol because CE routes are discovered through other mechanisms described later.
- MPLS over GRE tunnels and VXLAN tunnels in the OpenContrail System correspond to MPLS over MPLS in MPLS VPNs.
- The XMPP protocol in the OpenContrail System combines the functions of two different protocols in an MPLS VPN:
  - XMPP distributes routing information similar to what IBGP does in MPLS VPNs.
  - XMPP pushes certain kinds of configuration (e.g. routing instances) similar to what DMI does in MPLS VPNs.

- The OpenContrail System provides three separate pieces of functionality:

    1. Centralized control, similar to a BGP Route Reflector (RR) in an MPLS VPN.

    2. Management, which pushes down configuration state to vRouters similar to a Network Management System (NMS) in an MPLS VPN.

    3. Analytics.

- OpenContrail supports both Layer 3 overlays, which are the equivalent of MPLS L3-VPNs and Layer 2 overlays, which are the equivalent of MPLS EVPNs.

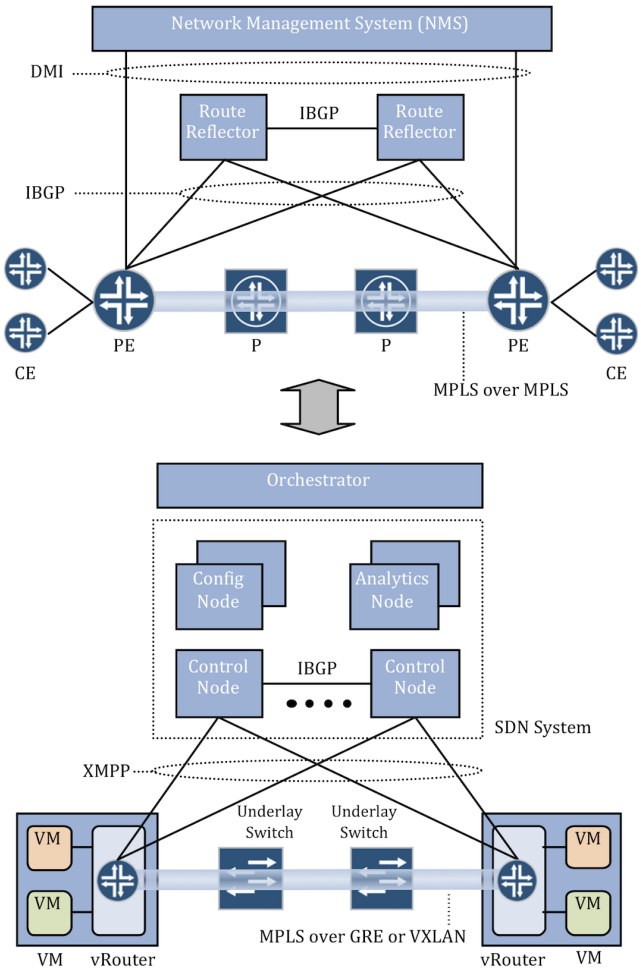

Figure 33 Comparison of the OpenContrail System to MPLS VPNs

References

NOTE This *Day One* book is reprinted from the online document, *OpenContrail Architecture Documentation*, located at: http://opencontrail.org/opencontrail-architecture-documentation/.

[AWS] Amazon Web Services. http://aws.amazon.com/

[AWS-EC2] Amazon Eleastic Compute Cloud (Amazon EC2) http://aws.amazon.com/ec2/

[AWS-EC2-EIP] Amazon EC2 Elastic IP Address. http://aws.amazon.com/articles/1346

[AWS-EC2-INSTANCE-ADDRESSING] Amazon EC2 Instance IP Addressing. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html

[AWS-VPC] Amazon Virtual Private Cloud (Amazon VPC). http://aws.amazon.com/vpc/

[AWS-VPC-SUBNETS] Amazon Virtual Private Cloud Subnets. http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Subnets.html

[cassandra] Apache Cassandra website. http://cassandra.apache.org/

[draft-rfernando-virt-topo-bgp-vpn] "Virtual Service Topologies in BGP VPNs." *IETF Internet Draft draft-rfernando-virt-topo-bgp-vpn*. https://datatracker.ietf.org/doc/draft-rfernando-virt-

topo-bgp-vpn/

[draft-mahalingam-dutt-dcops-vxlan] "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." *IETF Internet Draft draft-mahalingam-dutt-dcops-vxlan*. https://datatracker.ietf.org/doc/draft-mahalingam-dutt-dcops-vxlan/

[draft-marques-l3vpn-mcast-edge] "Edge Multicast Replication for BGP IP VPNs." *IETF Internet Draft draft-marques-l3vpn-mcast-edge*. https://datatracker.ietf.org/doc/draft-marques-l3vpn-mcast-edge/

[draft-ietf-l3vpn-end-system] "BGP-signaled end-system IP/VPNs." *IETF Internet Draft draft-ietf-l3vpn-end-system*. https://datatracker.ietf.org/doc/draft-ietf-l3vpn-end-system/

[draft-raggarwa-sajassi-l2vpn-evpn] "BGP MPLS Based Ethernet VPN." *IETF Internet Draft draft-raggarwa-sajassi-l2vpn-evpn*. https://datatracker.ietf.org/doc/draft-raggarwa-sajassi-l2vpn-evpn/

[ietf-xmpp-wg] IETF XMPP working group. http://datatracker.ietf.org/wg/xmpp/

[if-map] if-map.org website. http://www.if-map.org/

[juniper-why-overlay] "Proactive Overlay versus Reactive End-to-End." *Juniper Networks*. http://www.juniper.net/us/en/local/pdf/whitepapers/2000515-en.pdf

[redis] Redis website. http://redis.io/

[RFC4023] "Encapsulating MPLS in IP or Generic Routing Encapsulation." *IETF RFC4023*. http://tools.ietf.org/html/rfc4023

[RFC4271] "A Border Gateway Protocol 4 (BGP-4)." *IETF RFC4271*. http://www.ietf.org/rfc/rfc4271.txt

[RFC4364] "BGP/MPLS IP Virtual Private Networks (VPNs)." *IETF RFC4364*. http://tools.ietf.org/html/rfc4364

[RFC6513] "Multicast in BGP/MPLS VPNs." *IETF RFC6513*. http://tools.ietf.org/html/rfc6513

[snort] Snort Website. http://www.snort.org/

[snort-rules-intro] "A Brief Introduction to Snort Rules." *The Security Analysts*. http://www.secanalyst.org/2010/05/27/a-brief-introduction-to-snort-rules/

[xmpp] XMPP.org Website. http://xmpp.org/

[zookeeper] Apache Zookeeper website. http://zookeeper.apache.org/