Contrail Architecture

Table of Contents

List of Figures

Executive Summary

Juniper Networks® Contrail is an open-source software-defined networking (SDN) solution that automates and orchestrates the creation of highly scalable virtual networks. These virtual networks let you harness the power of the cloud—for applications and network services, increased business agility, and revenue growth. Contrail is a scale-out, standard based virtual networking solution that seamlessly integrates with physical routers and switches to eliminate the challenges of private and public cloud networking.

This white paper introduces Contrail to data center/network architects and provides a comprehensive overview of the architecture, protocols, and technologies that are implemented in Contrail. Additionally, the document discusses enterprise and service provider use cases, and it details the advantages of Contrail when compared to some existing technologies/solutions.

Introduction

SDN is the need of the hour for both enterprise and service providers who want to provide XaaS to their internal as well as external customers. There are lots of solutions that can help automate the provisioning of network devices but most of them are either based on proprietary protocols and standards or developed using outdated technologies like OpenFlow.

Contrail is an open-standard based, proactive overlay SDN solution that works with your existing physical network devices and help address the networking challenges for self-service, automated, and vertically integrated cloud architecture and improves scalability and CapEx inefficiencies through a proactive overlay virtual network. All of the networking features such as switching, routing, security, and load balancing are moved from the physical hardware infrastructure to software running in the hypervisor kernel that is managed from a central orchestration system. This allows the system to scale while keeping the costs of the physical switching infrastructure under control, as the switching hardware has no state of the virtual machines or tenant/application and is only involved in routing traffic from one server to another. The Contrail system also solves the agility problem, as it provides all of the automation for provisioning of the virtualized network, networking services, and integration with cloud orchestration systems such as OpenStack and CloudStack using REST APIs.

Overview of Contrail

This section provides an overview of the Contrail system—an extensible platform for SDN.

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

Use Cases

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

- Cloud Networking—It enables private clouds for enterprises and 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 IaaS use cases all involve 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 can also be interconnected to a physical IP VPN or L2 VPN.

The NFV use case involves orchestration and management of networking functions such as firewalls, intrusion detection service (IDS) or intrusion detection and prevention (IPS) systems, 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.

Contrail SDN Controller and the vRouter

The Contrail system consists of two main components: Contrail SDN Controller and Contrail vRouter.

Contrail SDN Controller is a logically centralized but physically distributed SDN controller that is responsible for providing the management, control, and analytics functions of the virtualized network.

The Contrail 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 the Overlay Networking section. Contrail vRouter is conceptually similar to existing commercial and open-source vSwitches such as the Open vSwitch (OVS), but it also provides routing and higher-layer services (for example, vRouter instead of vSwitch).

The Contrail SDN 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 Contrail system. VNs are logical constructs implemented on top of the physical networks. They 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.

VNs can be connected to and extended across physical MPLS Layer 3 virtual private networks (L3VPNs) and Ethernet VPNs (EVPNs) using a datacenter edge router.

Virtual networks are also used to implement NFV and service chaining. How this is achieved using virtual networks is explained in detail in the *Service Chaining* section.

Overlay Networking

Virtual networks can be implemented using a variety of mechanisms. For example, each VN could be implemented as a virtual LAN (VLAN), virtual private networks (VPNs), etc.

VNs 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 (WLAN) industry for more than a decade, but its application to datacenter 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, nonblocking, 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" among themselves. In the case of Contrail 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 L3 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 (that is, 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 a control plane that are 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 Contrail system is inspired by and conceptually very similar to standard MPLS L3VPNs (for L3 overlays) and MPLS EVPNs (for L2 overlays).

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

The control plane protocol between the control plane nodes of the Contrail 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 L3VPNs and MPLS EVPNs.

The protocol between the Contrail SDN Controller and the Contrail 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 Contrail system uses control plane and data plane protocols that are very similar to the protocols used for MPLS L3VPNs and EVPNs has multiple advantages. These technologies are mature and known to scale, and they are widely deployed in production networks and supported in multivendor physical gear that allows for seamless interoperability without the need for software gateways.

Contrail and Open Source

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

- The Contrail system is integrated with open-source hypervisors such as kernel-based virtual machines (KVMs) and Xen.
- The Contrail system is integrated with open-source virtualization orchestration systems such as OpenStack and CloudStack.
- The Contrail system is integrated with open-source physical server management systems such as Chef, Puppet, Cobbler, and Ganglia.

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

Juniper Networks also provides a commercial version of the Contrail system. Commercial support for the entire open-source stack (not just the Contrail 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 Contrail 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 this document mentioned that the Contrail SDN Controller is logically centralized but physically distributed.

Physically distributed means that the Contrail SDN Controller consists of multiple types of nodes, each of which can have multiple instances for high availability (HA) and horizontal scaling. These 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 northbound 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 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 southbound protocols to “make it so”—that is, to make the actual state of the network equal to the desired state of the network. In the initial version of the

Contrail system these southbound protocols include Extensible Messaging and Presence Protocol (XMPP) to control the Contrail vRouters as well as a combination of the 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 HA reasons.

- **Analytics nodes are responsible for collecting, collating, and presenting analytics information for troubleshooting problems and for determining network usage.** Each component of the Contrail 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 mechanisms to automatically trigger the collection of more detailed records when certain events 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 northbound analytics query REST API.

The physically distributed nature of the Contrail SDN 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 a very large-scale system—tens of thousands of servers.

Contrail is logically centralized and hence 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 Contrail system. A data model consists of a set of objects, their capabilities, and the relationships between them.

With the data model, applications use a declarative rather than an imperative mode to communicate, which is critical in achieving high programmer productivity. A fundamental aspect of Contrail'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 deal with the complexities of HA, 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, 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 Contrail system.

The control nodes are responsible for detecting 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 Contrail SDN Controller provide a northbound REST API to the provisioning or orchestration system. These northbound REST APIs are 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.

These REST APIs are secure—they can use HTTPS for authentication and encryption and also provides role-based authorization. They are also horizontally scalable because the API load can be spread over multiple configuration node instances.

Graphical User Interface

The Contrail system also provides a Graphical User Interface (GUI). This GUI is built entirely using the REST APIs described previously, and this ensures that there is no lag in APIs. It is expected that large-scale deployments or service provider OSS/BSS systems are integrated using the REST APIs.

Note: Juniper is in the process of making changes to the UI code-base that allows the company to make it available in open source.

An Extensible Platform

The initial version of the Contrail 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 Contrail system ships with a specific set of southbound protocols.

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

The low-level service data model that ships with the initial version of the Contrail system is specifically geared toward 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 southbound protocols implemented in the control nodes consists of XMPP, BGP, and NETCONF.

The Contrail system is an extensible platform in the sense that any of the aforementioned components can be extended to support additional use cases, additional network technologies, or both in future versions:

- The high-level service data model can be extended with additional objects to represent new services such as 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. 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 that 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 (that is, a new way to implement an existing service) or to map new high-level service objects to new or existing low-level technology objects (that is, implementing a new service).
- New southbound protocols can be introduced into the control nodes. This might 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 might be needed because new objects are introduced in the low-level technology data models that require new protocols to be implemented.

Contrail Architecture Details

As shown in Figure 1, the Contrail system consists of two parts—a logically centralized but physically distributed Contrail SDN Controller and a set of Contrail vRouters that serve as software forwarding elements implemented in the hypervisors of general-purpose virtualized servers.

Contrail SDN 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) plugin. The REST APIs can also be used by other applications and operator's OSS/BSS. Finally, the REST APIs are used to implement the web-based GUI included in the Contrail system.

The Contrail system provides three interfaces: a set of northbound REST APIs that are used to talk to the orchestration system and the applications, southbound interfaces that are used to communicate to virtual network elements (Contrail 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 Contrail vRouters, and BGP and NETCONF are the southbound interfaces for gateway routers and switches.

Internally, the Contrail SDN Controller consists of three main components:

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

All of these nodes are described in greater detail later in this section.

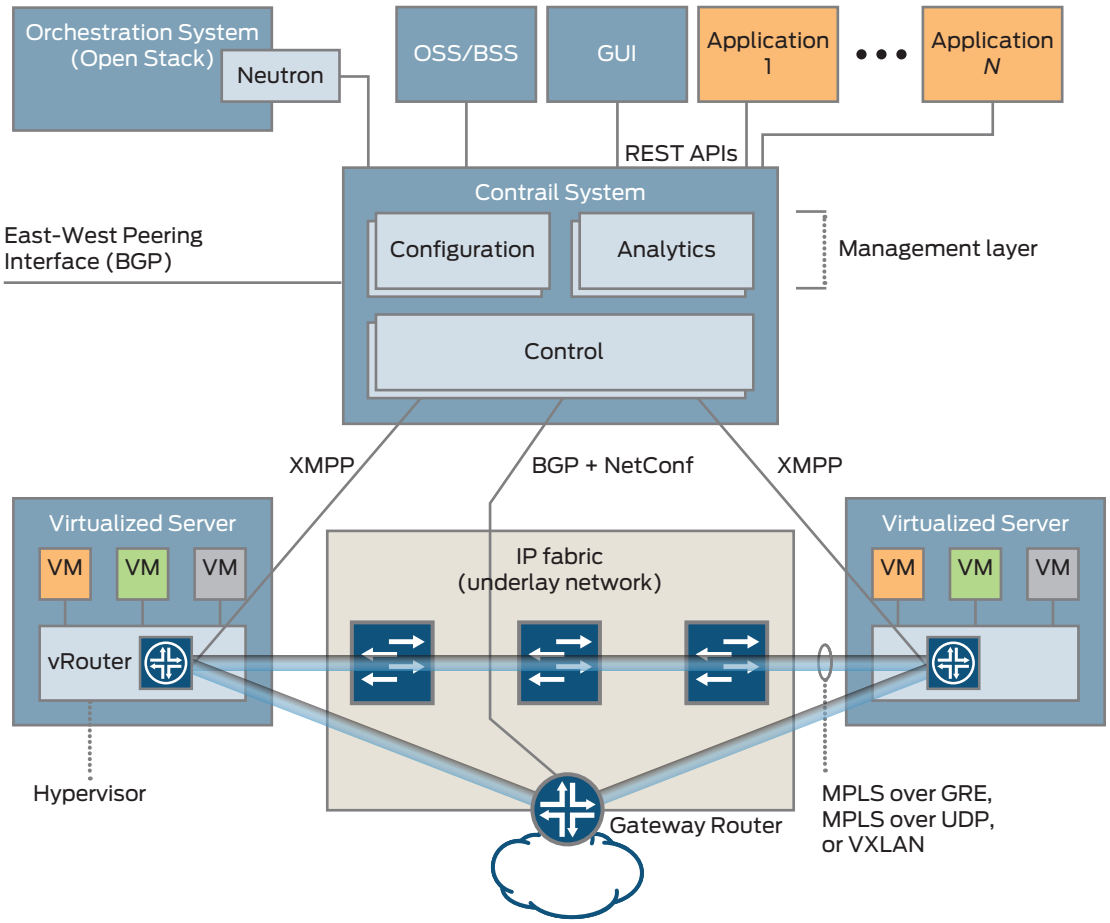


Figure 1: Contrail system overview

Contrail 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 Contrail vRouter consists of two parts: a user space agent that implements the control plane and a kernel module that implements the forwarding engine.

The Contrail 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 to sets of VMs.
2. Gateway functions refer to the ability to connect virtual networks to physical networks via a gateway router (for example, 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 as NFV, is the ability to steer flows of traffic through a sequence of physical or virtual network services such as firewalls, DPI, or load balancers.

Nodes

As shown in Figure 2, the system is implemented as a cooperating set of nodes running on general-purpose x86 servers. Each node can be implemented as a separate physical server, or it can be implemented as a 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. This information is 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 Contrail SDN Controller, this paper also identifies some additional nodes types for physical servers and physical network elements performing particular roles in the overall Contrail system:

- Compute nodes are general-purpose virtualized servers that host VMs. These VMs can be tenants running general applications, or these VMs can 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 non-virtualized servers.
- Service nodes are physical network elements providing network services such as DPI, IDP,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.

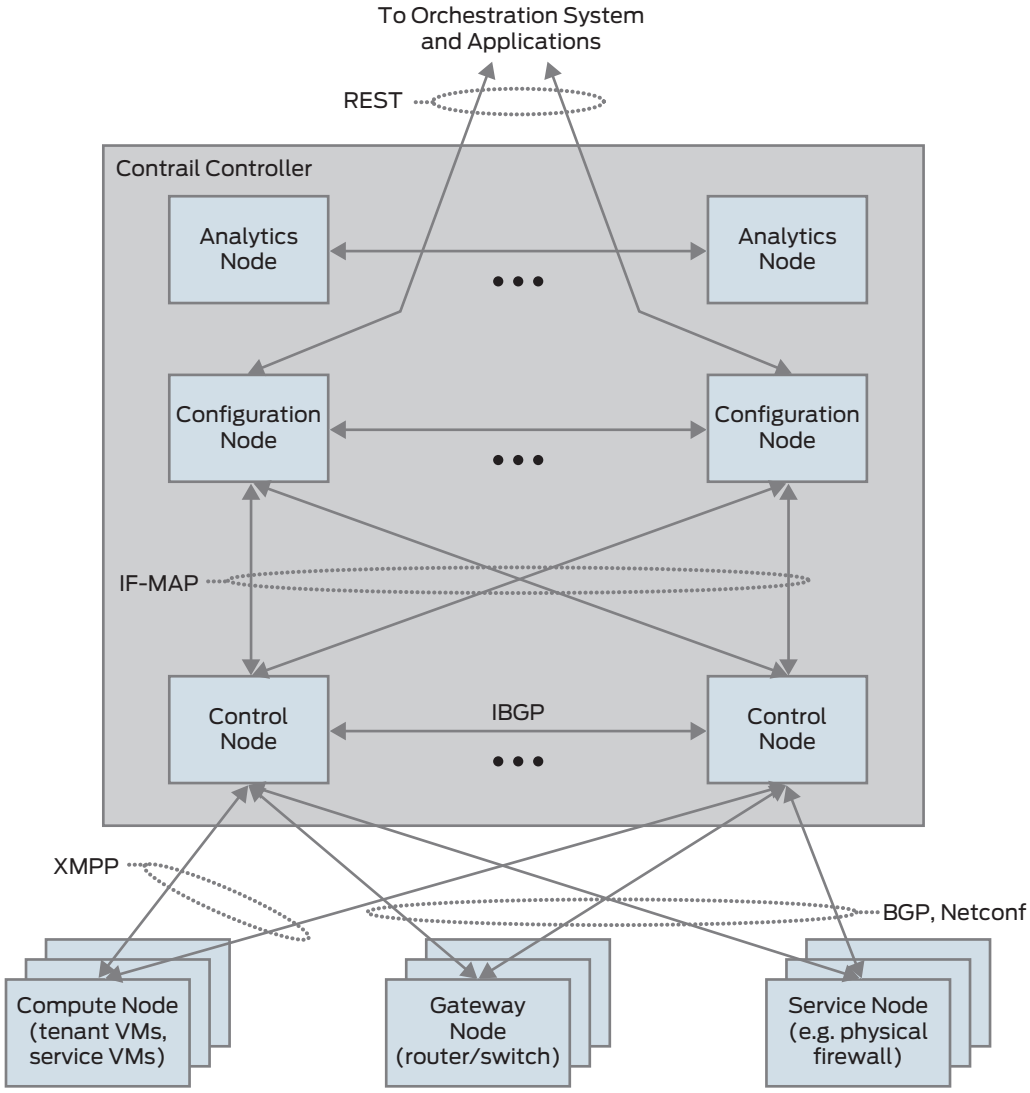


Figure 2: Contrail system implementation

Compute Node

The compute node is a general-purpose x86 server that hosts VMs. These are tenant VMs running customer applications such as Web servers, database servers, enterprise applications or hosting virtualized services used to create service chains. In the standard configuration, Linux is the host OS and KVM or Xen is the hypervisor. The Contrail 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 operating systems and hypervisors such as VMware ESXi or Windows Hyper-V are expected to be supported in the future.

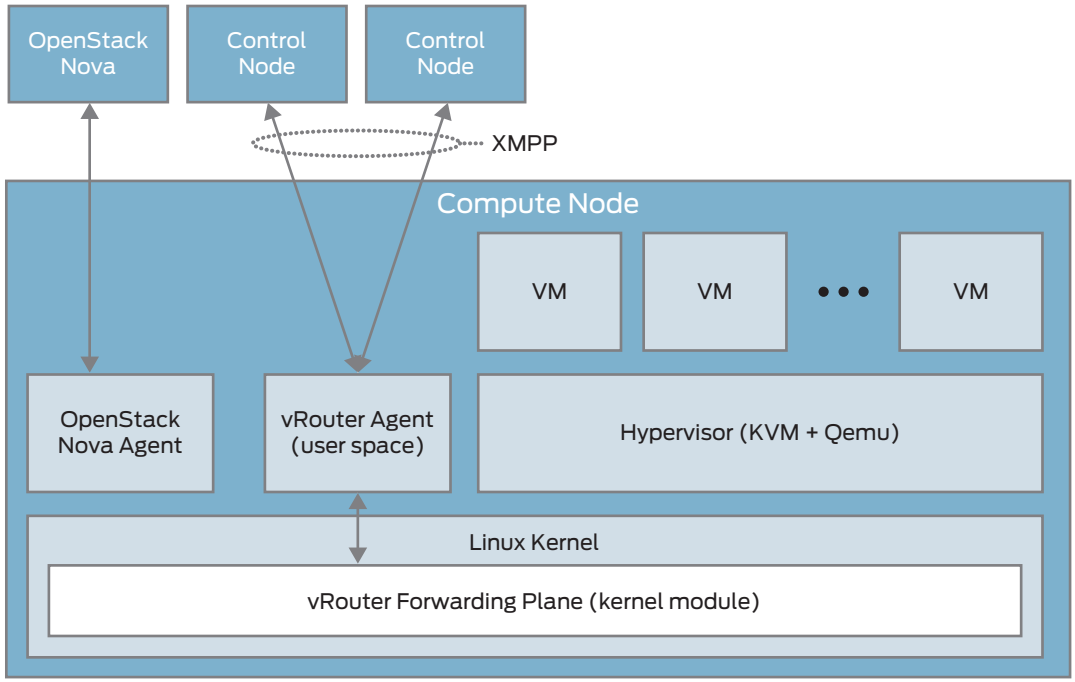


Figure 3: Internal structure of a compute node

Two of the building blocks in a compute node implement a Contrail 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:

- It exchanges control state such as routes with the control nodes using XMPP.
- It receives low-level configuration state such as routing instances and forwarding policy from the control nodes using XMPP.
- It reports analytics state such as logs, statistics, and events to the analytics nodes.
- It installs forwarding state into the forwarding plane.
- It discovers the existence and attributes of VMs in cooperation with the Nova agent.
- It applies forwarding policy for the first packet of each new flow and installs a flow entry in the flow table of the forwarding plane.
- It proxies DHCP, ARP, DNS, and MDNS. Additional proxies can be included 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 loadable kernel module in Linux and is responsible for the following functions:

- It enables encapsulating packets to be sent to the overlay network and decapsulating packets to be received from the overlay network.
- It assigns 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.
- It does a lookup of the destination address in the forwarding information base (FIB)—also known as forwarding table—and forwards the packet to the correct destination. The routes can be Layer 3 IP prefixes or Layer 2 MAC addresses.

- A forwarding policy can be applied using a flow table:
  - It matches packets against the flow table and applies the flow actions.
  - It punts the packets for which no flow rule is found (that is, the first packet of every flow) to the vRouter agent, which then installs a rule in the flow table.
- It punts certain packets such as DHCP, ARP, MDNS to the vRouter agent for proxying.

Figure 4 shows the internal structure of the vRouter forwarding plane.

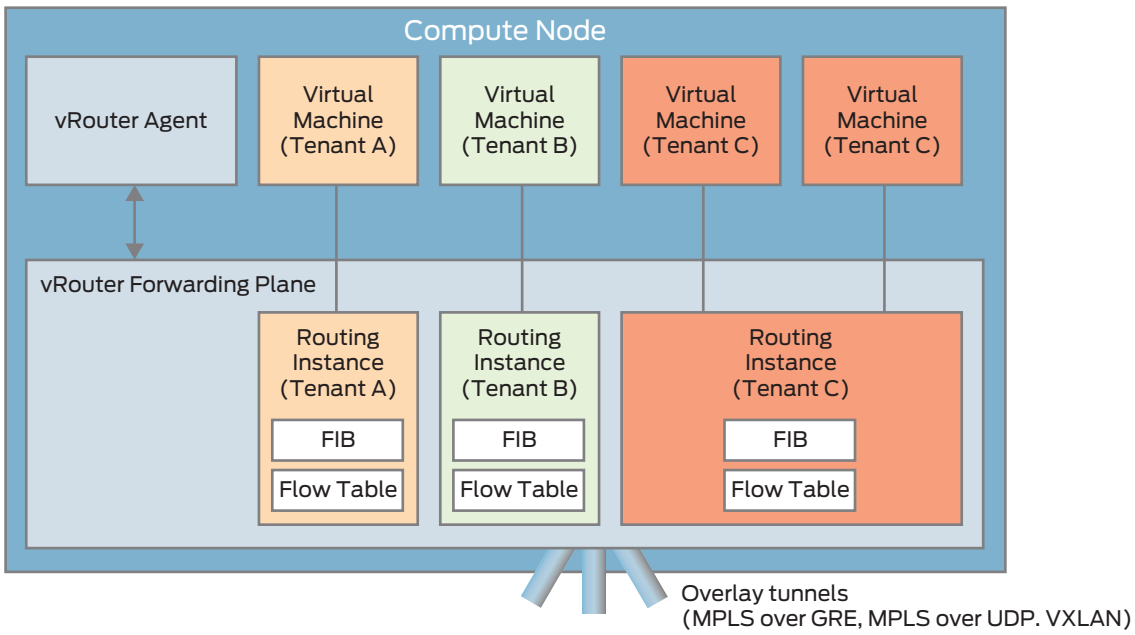


Figure 4: vRouter forwarding plane

The forwarding plane supports MPLS over GRE/UDP and VXLAN encapsulations in the overlay network. The forwarding plane supports both 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 supports only IPv4. Support for IPv6 is expected to be added in the future.

See *The Contrail Forwarding Plane* section 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:

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

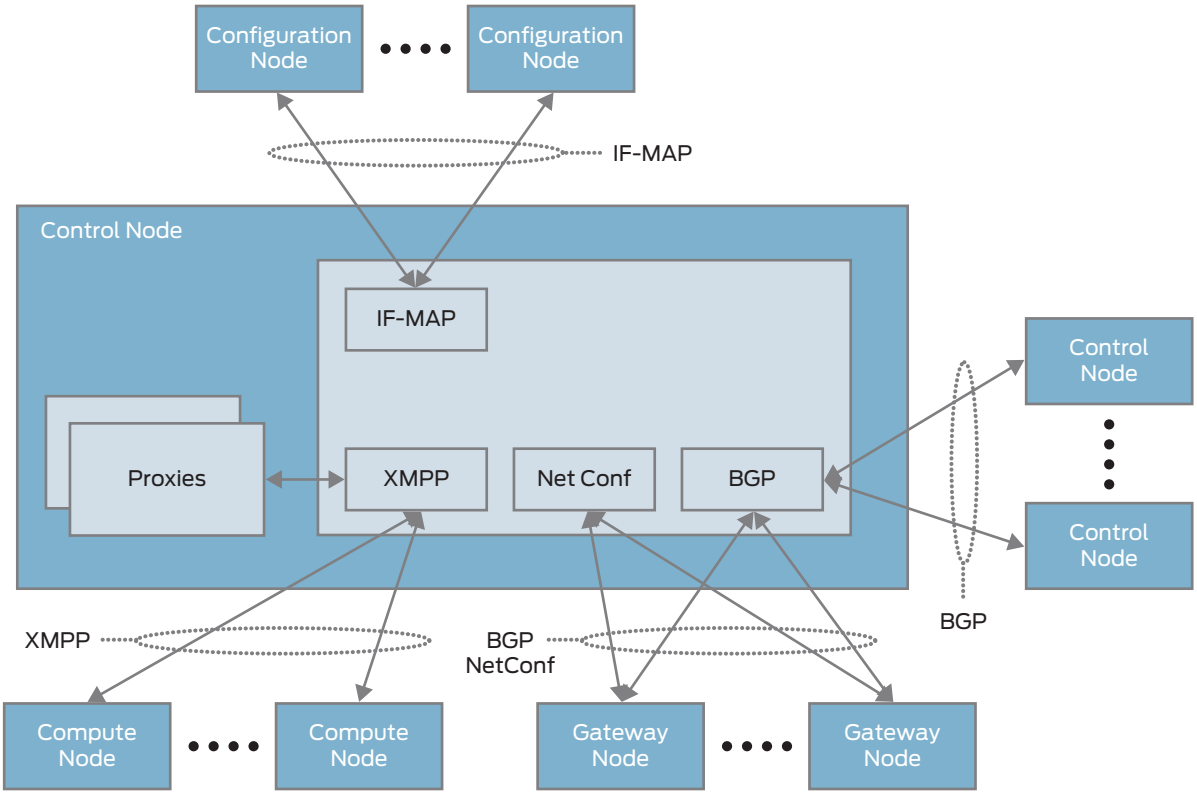


Figure 5: Internal structure of a control node

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 (that is, other nodes providing a particular service). For example, when the vRouter agent in a compute node connects to a control node[1], 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 provides the northbound 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 facilitates communications among internal components.
- A Cassandra [**cassandra**] database provides persistent storage of configuration. Cassandra is a fault-tolerant and horizontally scalable database.
- A Schema transformer detects 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 provides a southbound interface to push the computed low-level configuration down to the control nodes.
- ZooKeeper **[zookeeper]** (not shown in diagram) is used for allocation of unique object identifiers and to implement transactions.

[1]To be more precise—to an active-active pair of control VMs.

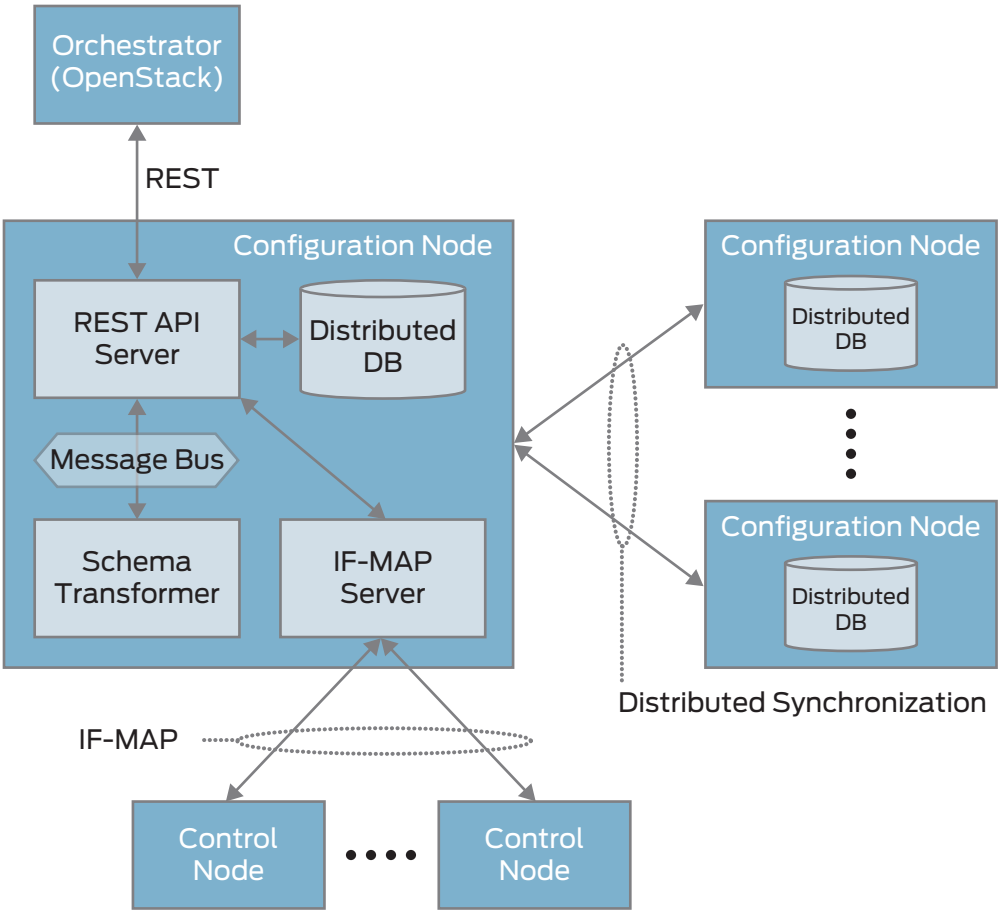


Figure 6: Internal structure of a configuration node

Analytics Node

Figure 7 shows the internal structure of an analytics node. An analytics node communicates with applications using a northbound REST API, with other analytics nodes using a distributed synchronization mechanism, and 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 exchanges Sandesh messages (see the *Sandesh* section) with components in control nodes and configuration nodes to collect analytics information.
- A NoSQL database stores this information.
- A rules engine automatically collects operational state when specific events occur.
- A REST API server provides a northbound interface for querying the analytics database and for retrieving operational state.
- A query engine executes the queries received over the northbound REST API. This engine provides the capability for flexible access to potentially large amounts of analytics data.

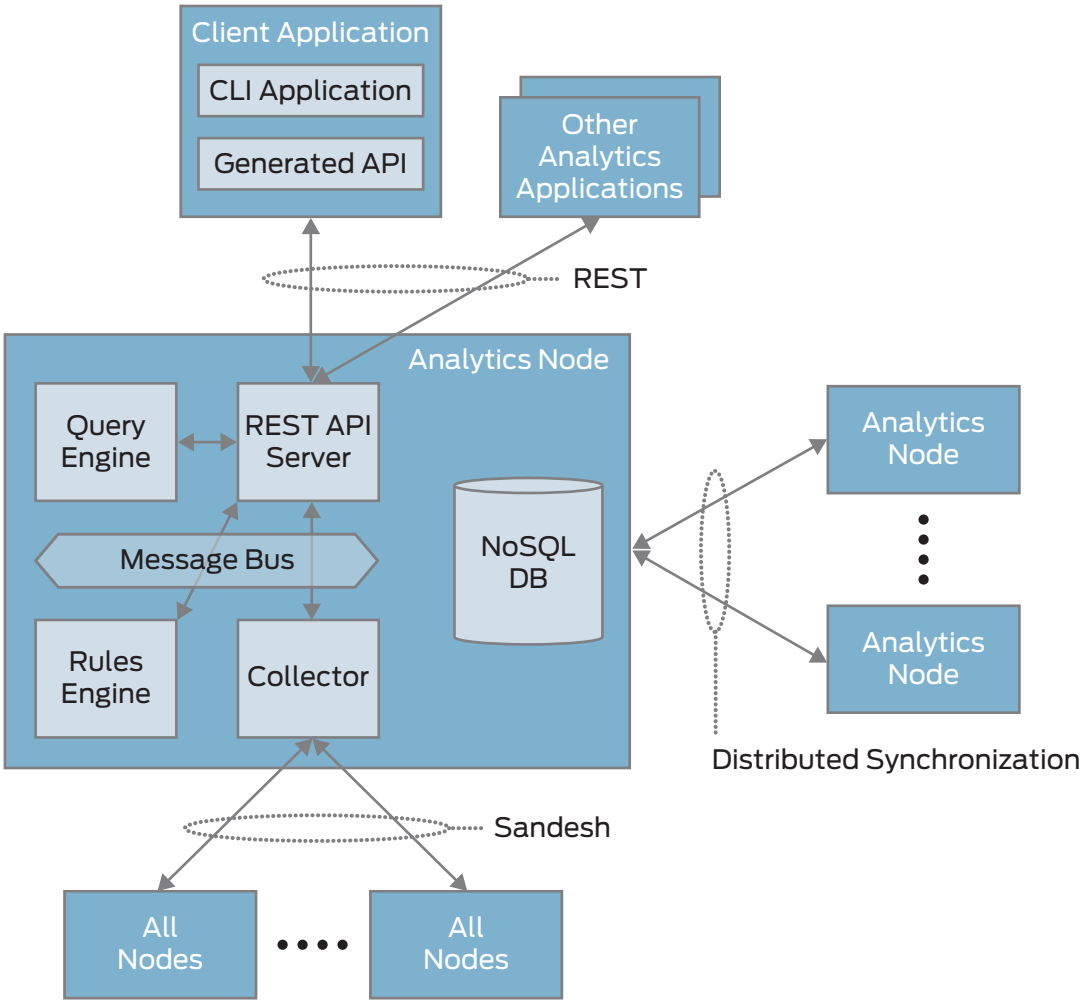


Figure 7: Internal structure of an analytics node

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 Contrail queries are time series.

The Contrail 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 is expected to 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.

MPLS over GRE

Figure 8 shows the MPLS over GRE packet encapsulation format for L3 and L2 overlays.

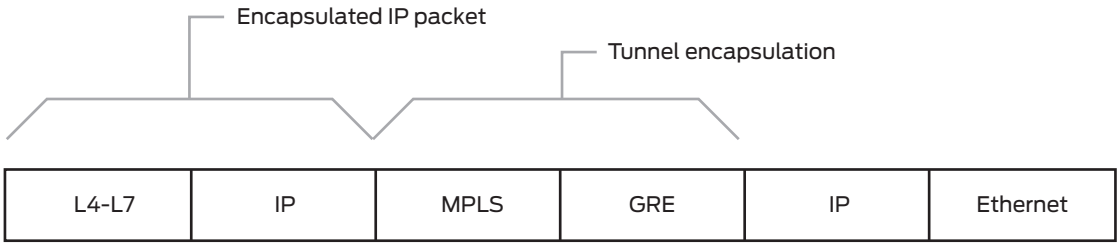


Figure 8: IP over MPLS over GRE packet format

Figure 9 shows the MPLS over GRE packet format for L2 overlays.

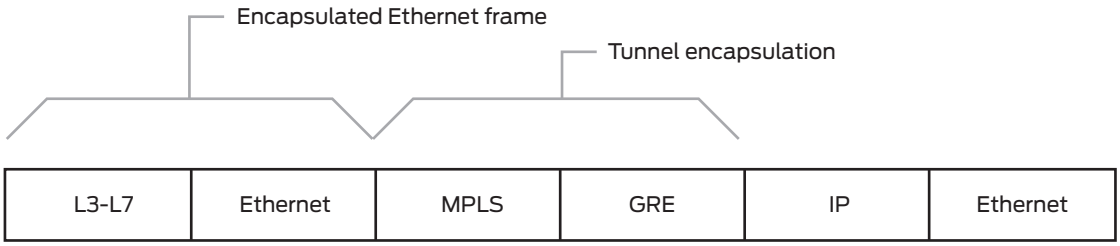


Figure 9: Ethernet over MPLS over GRE packet format

MPLS L3VPNs [**RFC4364**] and EVPNs [**draft-raggarwa-sajassi-l2vpn-evpn**] typically use MPLS over MPLS encapsulation, but they can use MPLS over GRE [**RFC4023**] as well if the core is not MPLS enabled. Contrail uses the MPLS over GRE 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 might not want the complexity of running MPLS in the data center. Third, there is no need for traffic engineering inside the data center because the bandwidth is overprovisioned.

VXLAN

For L2 overlays, Contrail also supports VXLAN encapsulation [**draft-mahalingam-dutt-dcops-vxlan**]. This is shown in Figure 10.

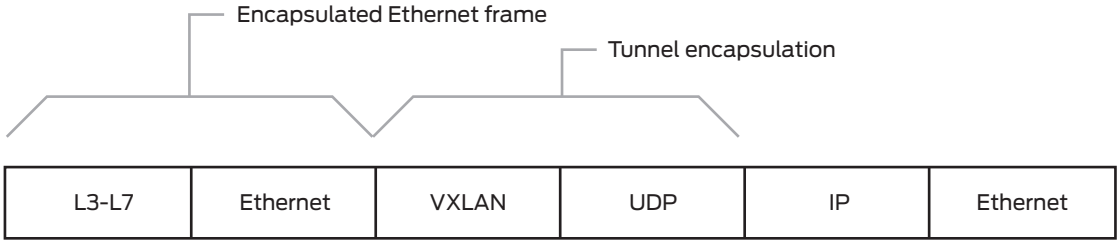


Figure 10: Ethernet over VXLAN packet format

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

Contrail'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 section. 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

Contrail supports a third encapsulation, namely MPLS over UDP. It is a cross between the MPLS over GRE and the VXLAN encapsulation. It supports both L2 and L3 overlays, 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 multipathing in the underlay (like VLXAN).

Figure 11 shows the MPLS over UDP packet encapsulation format for L3 overlays.

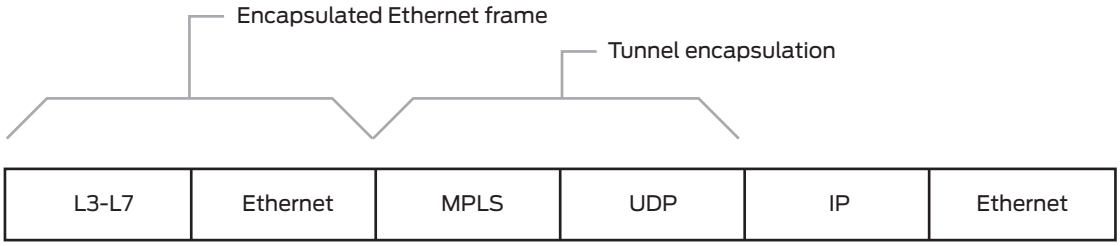


Figure 11: IP over MPLS over UDP packet format

Figure 12 shows the MPLS over UDP packet encapsulation format for L2 overlays.

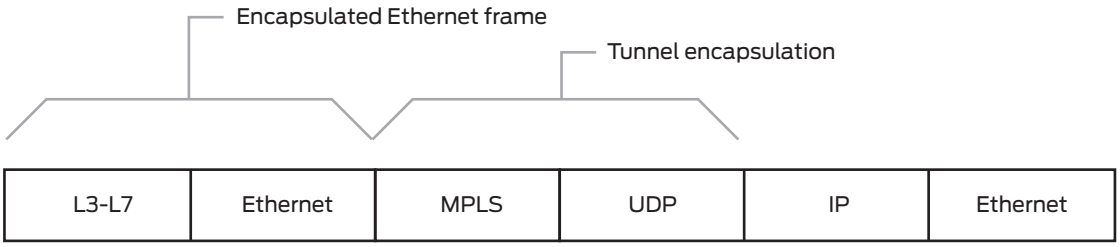


Figure 12: Ethernet over MPLS over UDP packet format

Layer 3 Unicast

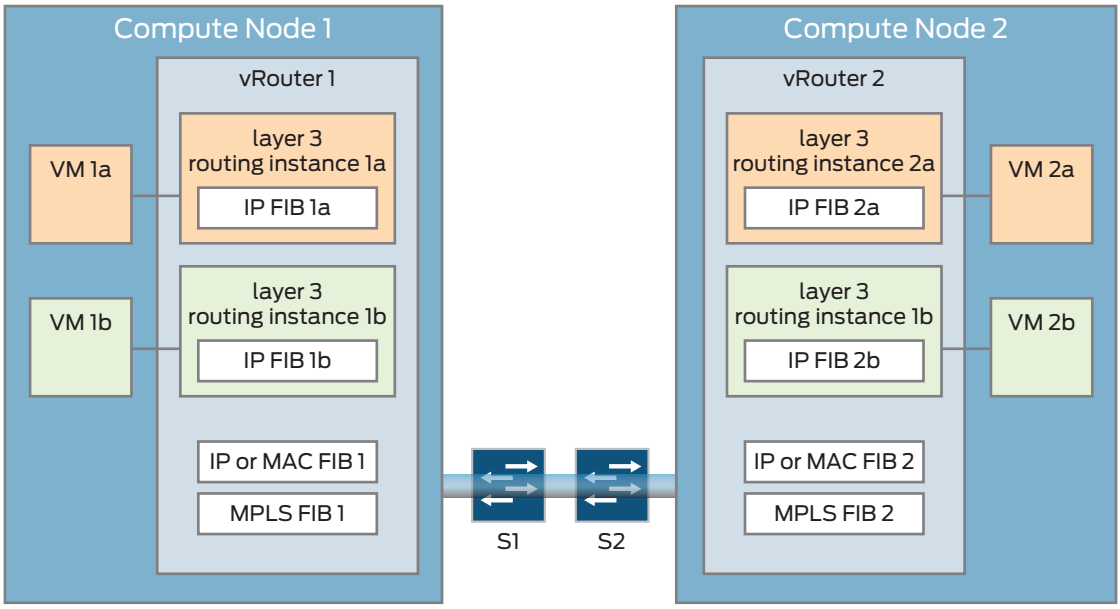


Figure 13: Data plane—layer 3 unicast forwarding plane

A summary of the sequence of events for sending an IP packet from VM 1a to VM 2a is given in the following steps. For a more detailed description, see [**draft-ietf-l3vpn-end-system**]. The following description applies to 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 the control node using XMPP. The next hop of the route does the following:
   a. It imposes an MPLS label that was allocated by vRouter 2 for routing instance 2a.
   b. It 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 (which is compute node 2) in global IP FIB 1.
7. vRouter 1 sends the encapsulated packet to compute node 2. How this happens exactly depends on whether the underlay network is a layer 2 switched network or a layer 3 routed network. This is described in detail in the following section. For now, skip this part and assume that 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—that is, it removes the GRE header that 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 that points to routing instance 2a. It decapsulates the packet—that is, 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.

The next section addresses what was glossed over in step 7—how the encapsulated packet is forwarded across the underlay network.

If the underlay network is a layer 2 network, then the following occurs:

1. 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.
2. 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.
3. 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 following occurs:

1. The outer source IP address (compute node 1) and the destination IP address (compute node 2) of the encapsulated packet are on different subnets.
2. 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.
3. The encapsulated packet is layer 3 routed from compute node 1 to compute node 2 based on the destination IP address. Equal-cost multipath (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

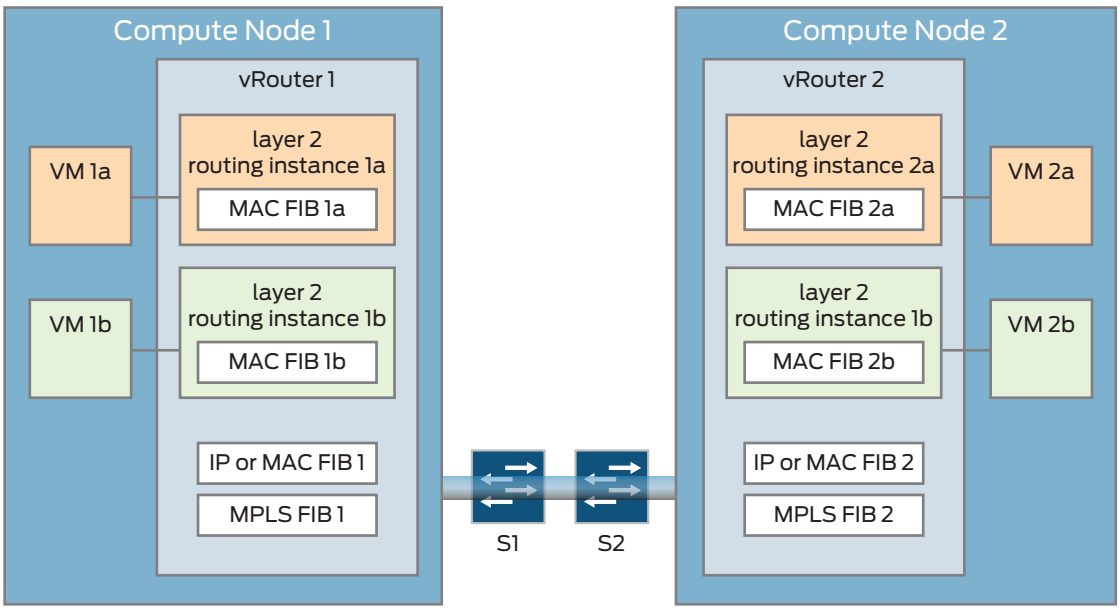


Figure 14: Data plane—layer 2 unicast

Forwarding for L2 overlays works exactly the same as forwarding for L3 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.

Fallback Switching

Contrail supports a hybrid mode where a virtual network is both a L2 and a L3 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 that 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

Contrail supports IP multicast in L3 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

Contrail does multicast elaboration using multicast trees in the overlay instead of the underlays. In addition to basic concepts that are summarized in this paper, details are described in [**draft-marques-l3vpn-mcast-edge**].

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 N more 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.

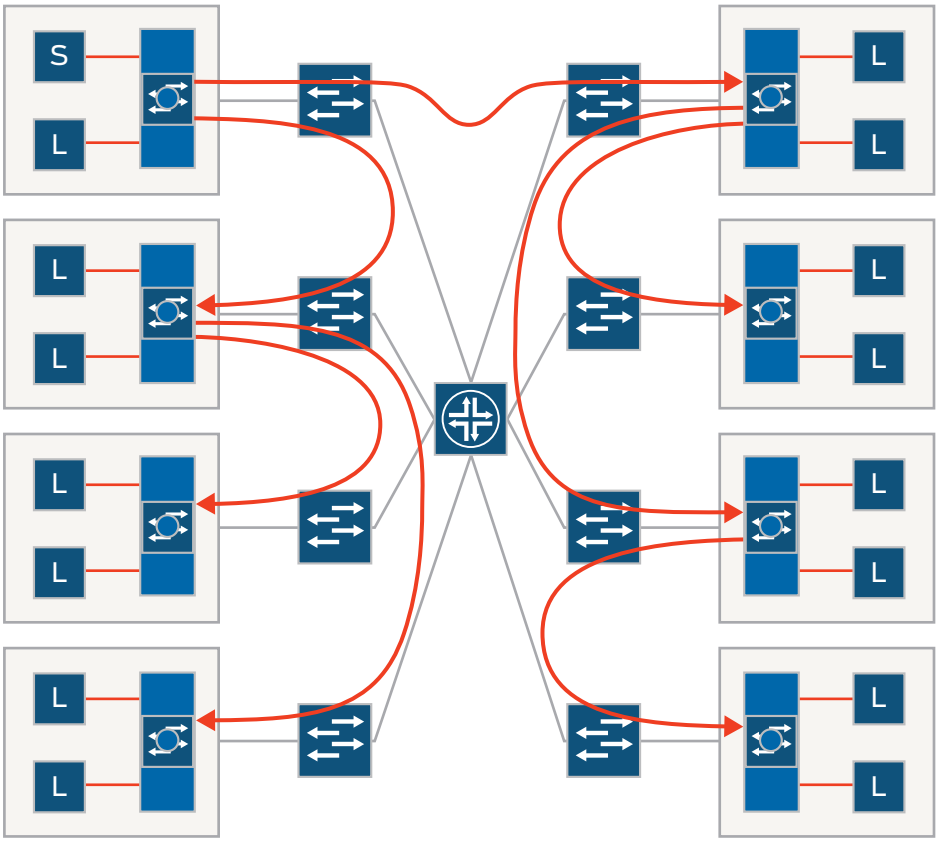


Figure 15: Multicast tree in the overlay (general case)

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. You can find more details here [**draft-marques-l3vpn-mcast-edge**].

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.

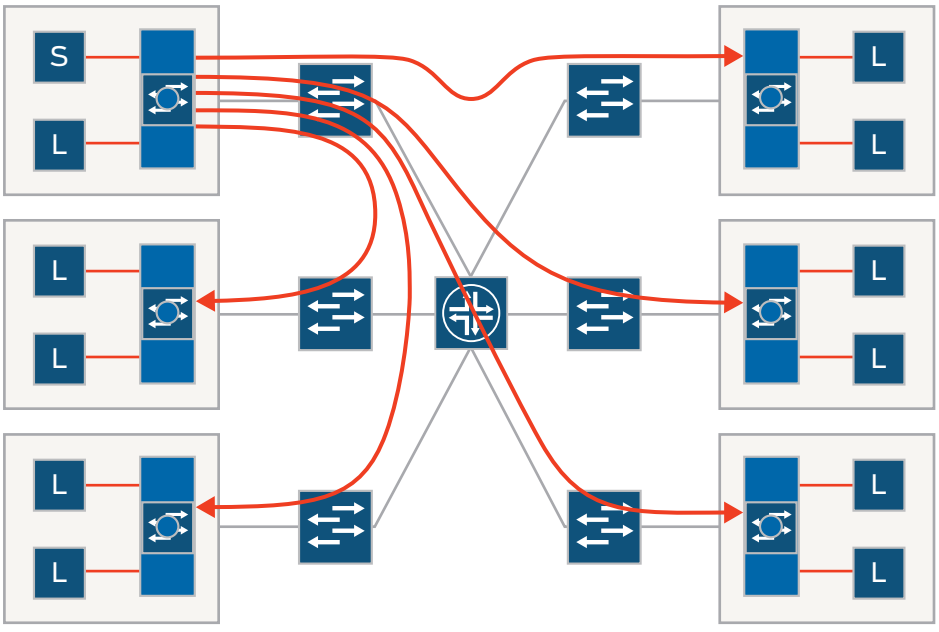


Figure 16: Multicast tree in the overlay (ingress replication special case)

Underlay Multicast Trees

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

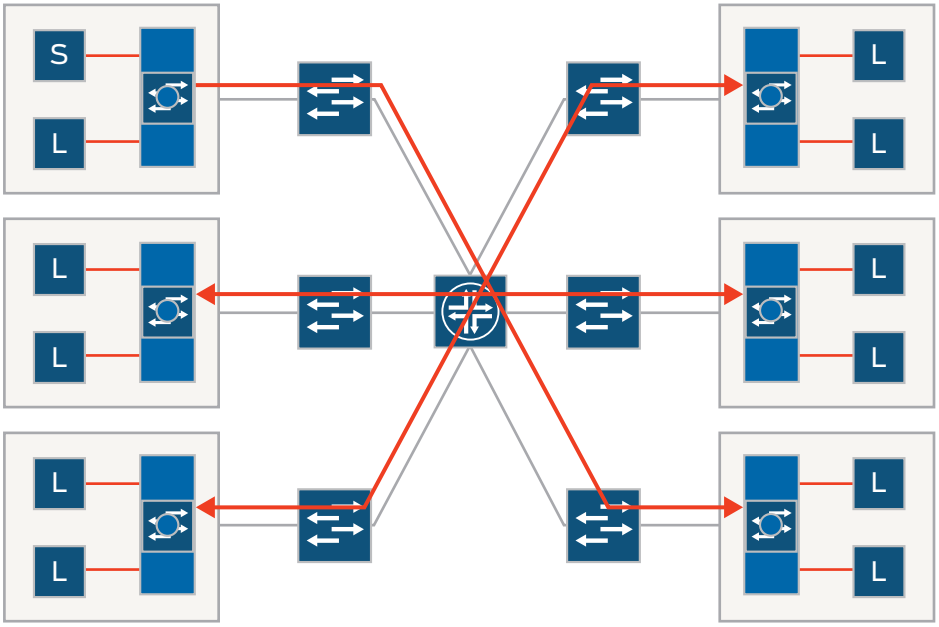


Figure 17: Multicast tree in the underlay

This is the approach that is generally used to implement multicast in MPLS L3VPNs (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 might 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 multicast is run 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:

- 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 that provides full nonblocking 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) need to be flooded in a L2 network. In Contrail, 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. L2 broadcasts are also avoided because most L2 broadcasts are caused by a small set of protocols.

For any remaining L2 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 are to be added):

- DHCP request—The control node provides the DHCP response based on the configuration of the VM.
- ARP request—The control node provides the IP to MAC address mapping.
- DNS and MDNS request—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 vRouter agent programs the flow entries in the forwarding plane.

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 detect the first packet of every new flow. The vRouter agent installs a flow entry for each new flow and then re-injects the packet into the forwarding plane.

Service Chaining

Contrail 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 might change as the system is extended. The policy rule looks similar to the following:

```
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 previous 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 focused on traffic steering—that is, 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 in the following ways:

- The route targets for routes are manipulated to influence importing and exporting routing from one routing instance to another routing instance.
- The next hops, labels, or both of the routes are manipulated 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 of routing instance manipulation for service chaining.

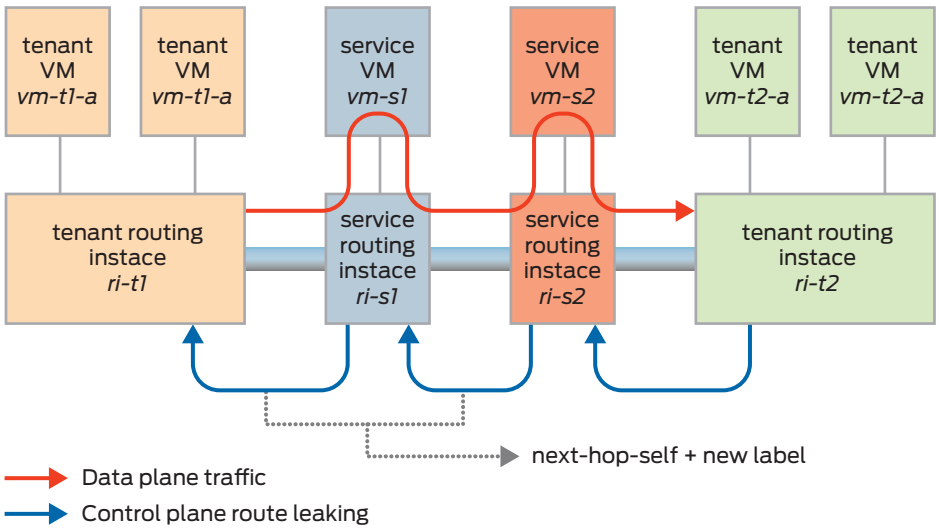


Figure 18: Service Chaining

In the previous 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 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.

Control and Management Plane Protocols

IF-MAP

The Interface to Metadata Access Point (**IF-MAP**) [if-map] is an open standard client/server protocol developed by the Trusted Computing 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 (MAPs), 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.

Contrail 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 only receive the subset of configuration that is needed. 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.

Contrail 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

Contrail uses BGP [RFC4271] to exchange routing information among 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

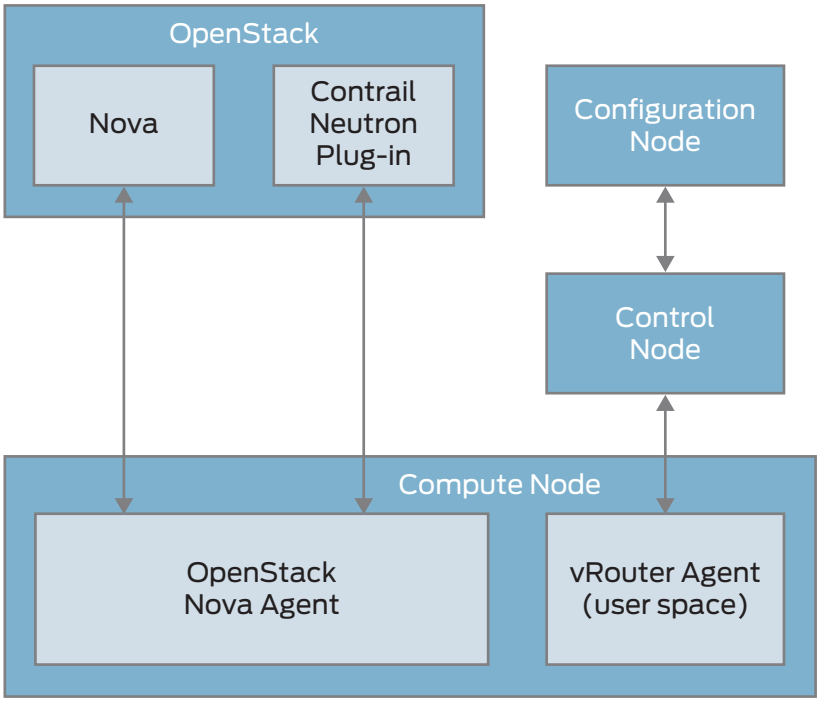
Sandesh is an XML-based protocol for reporting analytics information. The structure of the XML messages is described in published schemas. Each component on every node has a Sandesh connection to one of the analytics nodes. Sandesh carries two kinds of messages:

- 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 specific operational state.

OpenStack Integration

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

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 Contrail Neutron plug-in to retrieve the network attributes of the new virtual machine (for example, the IP address). Once the virtual machine is created, the Nova agent notifies the vRouter agent, which configures the virtual network for the newly created virtual machine (for example, 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. The protocols can also be used to provide confidentiality, although there is typically no need for that in the confines of a 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 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 assign one or more roles. The roles determine what operations the client is allowed to perform over the interface (for example, 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 (HA), 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—Contrail does not use the active-standby concept.

Control Nodes

Currently, each control node contains all operational state for the entire 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 might 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 might be transiently inconsistent. It makes a local determination 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 determines that the connection to that control node is lost. The vRouter agent flushes 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 contacts the service discovery server again to reestablish 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—that is, the configuration state that was explicitly installed by the provisioning system. And it also includes the contents of the low-level data model—that is, 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 recontacts the service discovery server and is assigned to a different configuration server.

After the switchover, 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 that is used for graceful restart in routing protocols.

Analytics Nodes

The system provides complete HA 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 transition the affected 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.

Contrail provides complete HA functionality across the database components. The database cluster is set up in a multiple replication manner, hence the data itself is resilient to database node failures. The cluster is resilient to multiple database node failures. Upon failure of a database node, the analytics nodes smoothly transition from the failed node to a functioning node. During this process, you queue up the data and hence during this transition the data loss is very minimal.

vRouter Agent

vRouter HA 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 that 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 reestablishes connections to a pair of redundant control nodes. It re-learns all state from the control nodes and reinstalls the fresh state in the forwarding plane, replacing the stale state.

When the vRouter agent finishes relearning the state from the control nodes and completes reinstalling 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 is 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.

The Data Model

Underlying all state 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 might represent an abstract or concrete object, and a link might 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 might be annotated with the required bandwidth, or a link between routing instances might 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 might 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 might only extend existing objects, links, and annotations, but they 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 in the following section. 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. Second, when a module gets all the information it needs for a particular action, it performs the action. The updates might 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 restartable. If it crashes, or is forcibly restarted (for example, 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, modules must be able to work in a distributed fashion—instances might 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 (for example, 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 might be administrative entities, or physical or logical resources. Links in the DAG represent relationships between objects. An object might have zero or more children but has exactly one parent (except the root, which has no parents). Deleting an object implicitly deletes the sub-tree below it. An object might 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 might be deleted.

As shown in Figure 20, the root of the DAG represents the universe of objects that the system is responsible for—that is, an "administrative domain" (AD). This might be a data center cluster, a point of presence (POP), a central office (CO), or a WAN. An AD has an overall administrator that is responsible for managing it. An AD also has an associated namespace for all identifiers that might be used within it. Examples of identifiers are IP addresses, domain names, and routing instance IDs.

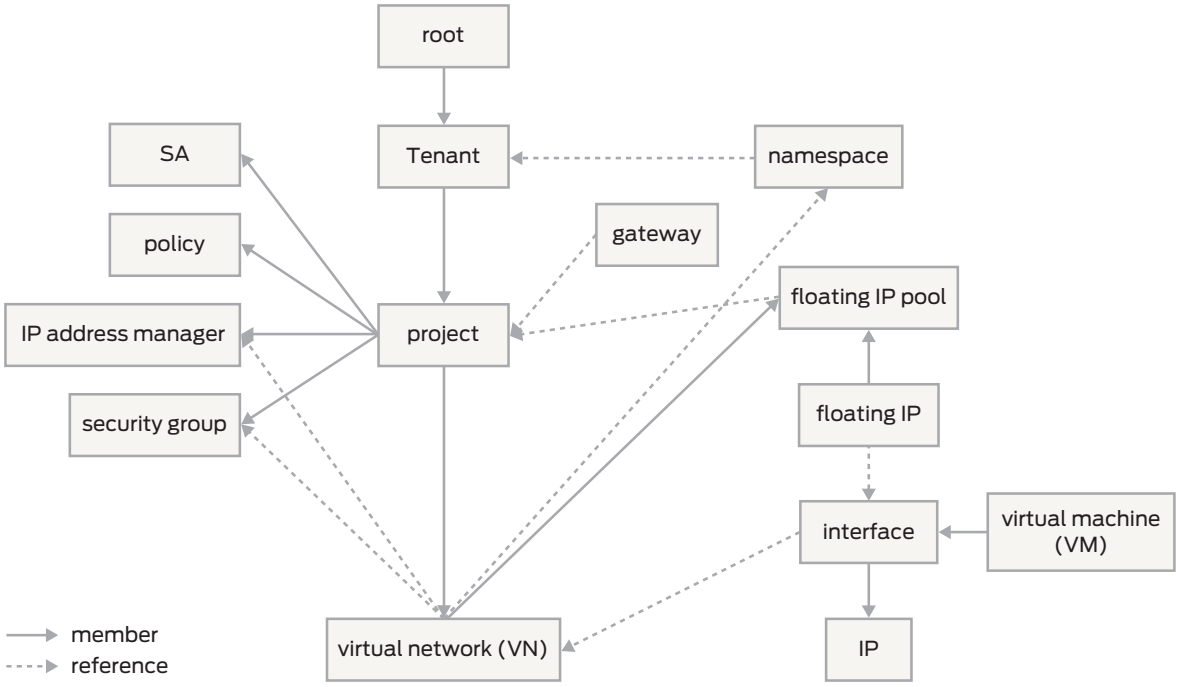


Figure 20: The Contrail system high-level data model definition

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

A tenant might contain a number of projects such as a department within a tenant—marketing is an example of a project. Projects contain virtual networks (VNs). A VN in a DC might consist of virtual machines in an application tier. A VN in a POP might represent a VPN for a business customer, 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 might 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). For the wired/wireless edge, there 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 it 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 might be redundant. In this case, a single instance can be used at that level to maintain the hierarchy. For example, if the notion of a 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.

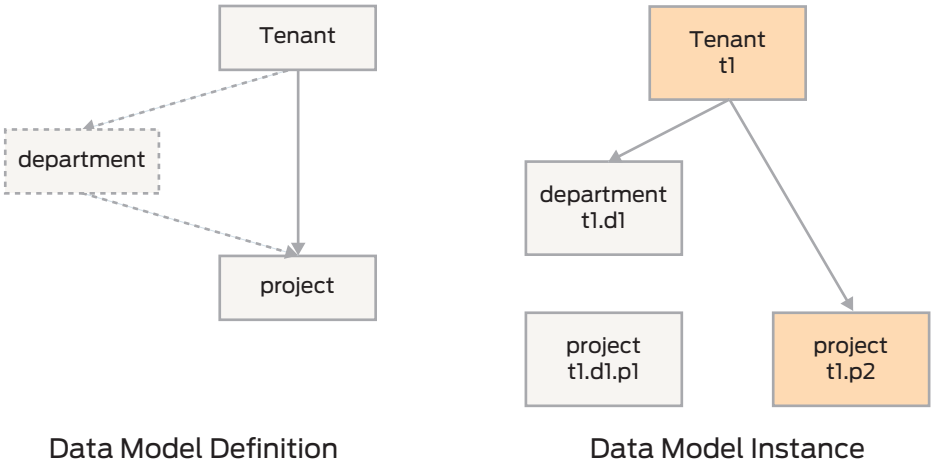


Figure 21: Data model extensibility

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.

The previous diagram 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* in purple—these follow the modified data model. Note that each project has only one parent. The orange project has *t1* as its parent, and the purple project has *t1.d1* as its parent.

High-Level and Low-Level Data Models

All the aforementioned 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 they 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 can 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 it can 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 Contrail peers, such as a data center 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 can go through an arbitrary graph of service nodes before reaching another VN. The traffic can take different paths based on a service (for example, an IDS can redirect traffic to a DPI engine), or it can 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," this document uses it.

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 is an annotation to a policy, and the latter is 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 a pair of the system.

Contrail Use Cases

There are three immediate use cases for Contrail

a) Private Cloud for the Enterprise

b) Infrastructure as a Service and Virtual Private Cloud for Service Providers

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

The Role of Orchestration in the Data Center

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.

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 has 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, such as in the following examples:

- 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—for example, 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 (for example, 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—for example, 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

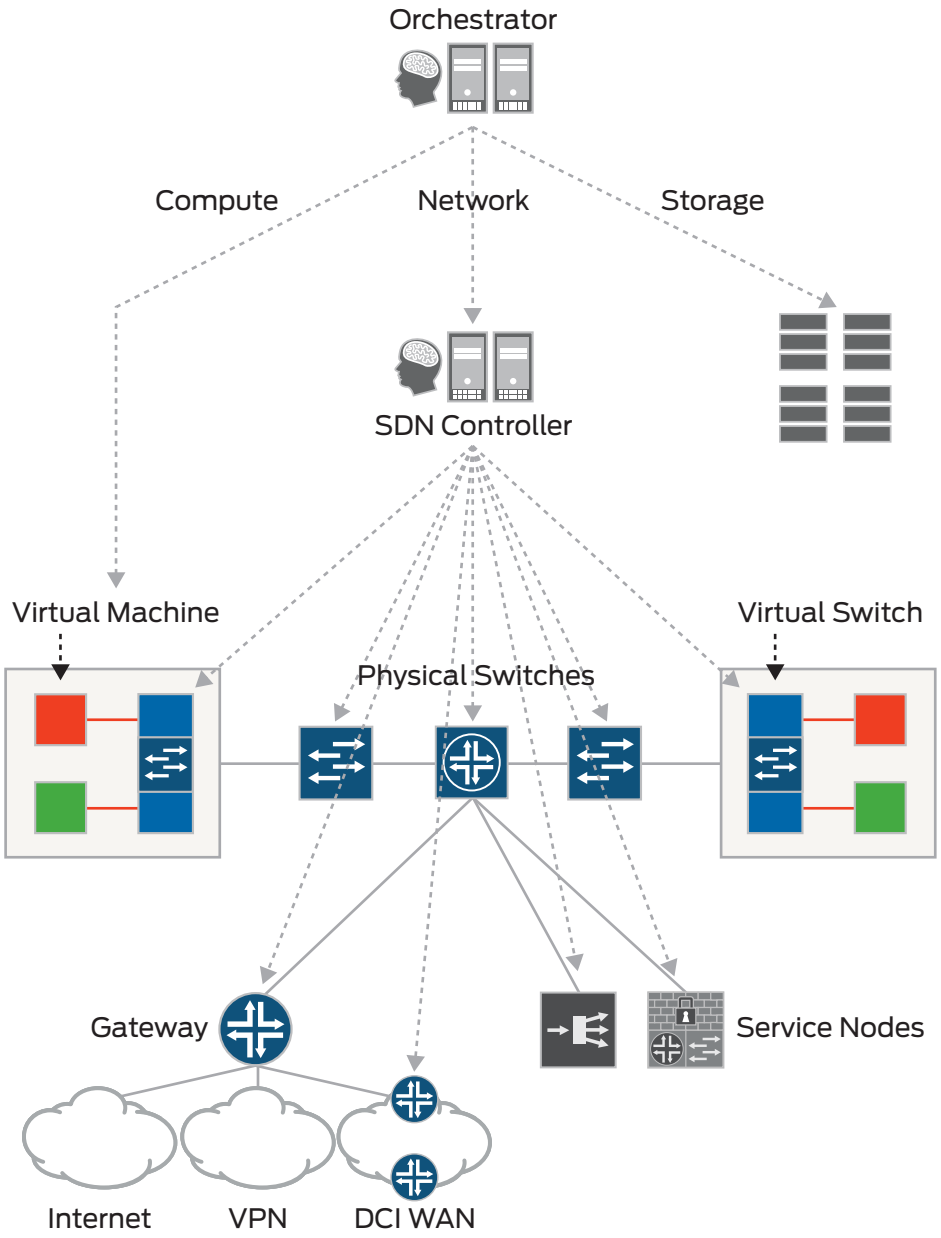


Figure 22: The role of orchestration in the data center

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.

Not all data centers are multi-tenant. Some large content providers (for example, 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. 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 cloud and the movement toward software-defined data centers, 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 (for example, OpenStack, CloudStack) and hypervisors (for example, KVM, Xen) are often used for customizability, cost, and scalability reasons.
- Very large content providers (for example, Google and Facebook) often build their own orchestration software and don't use hypervisors for performance and scale reasons.

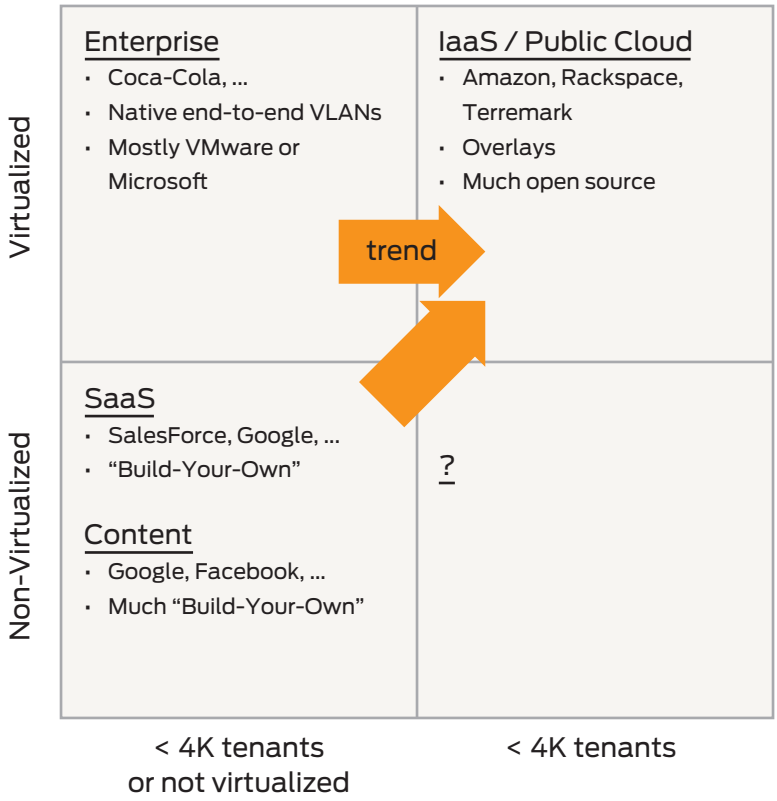


Figure 23: Multi-tenancy requirements

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 can also run "bare-metal" on the server (that is, not in a virtual machine) as shown in the green server (B) in the lower right corner.

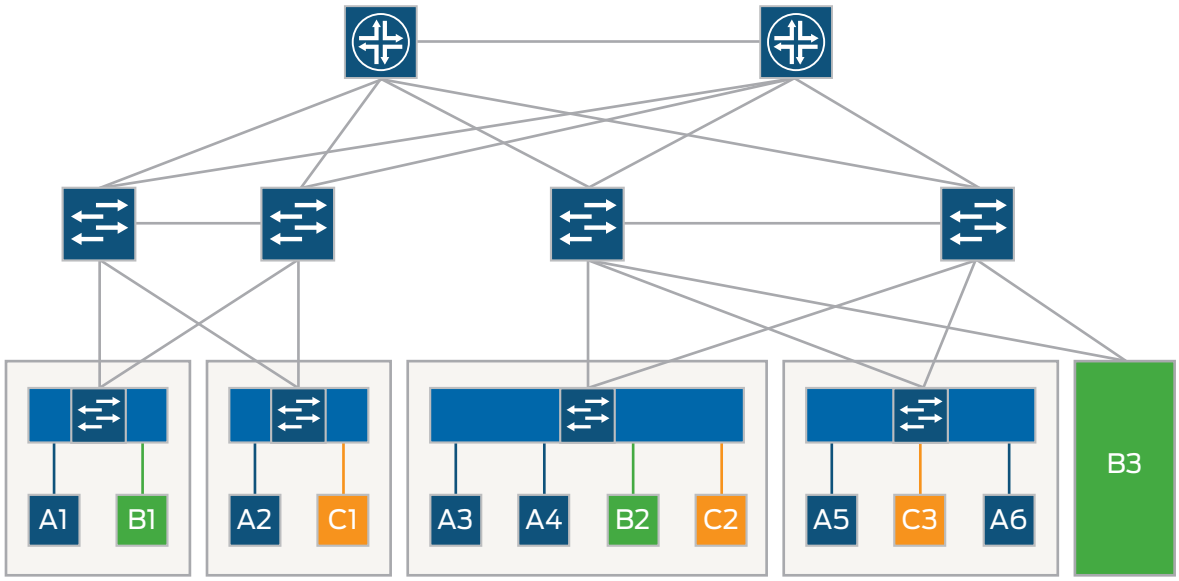


Figure 24: Use case "Multi-Tenant Virtualized Data Center" (multi-tier data center network)

The data center network can be a multi-tier network as shown in Figure 24, or the data center can be a single-tier network (for example Fabric) as shown in Figure 25.

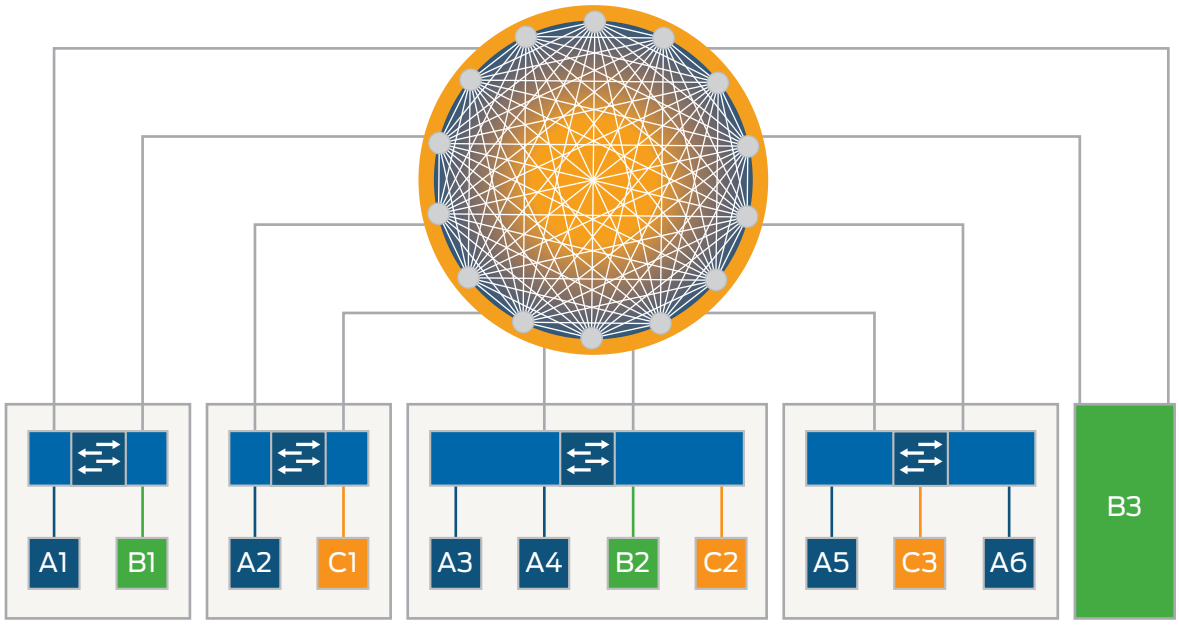


Figure 25: Use case "Multi-Tenant Virtualized Data Center" (single-tier data center network)

The servers are interconnected using a physical data center network. In Figure 24 the network is depicted as a 2-tier (access, core) network. It could also be a 3-tier (access, aggregation, core) network or a 1-tier (for example, Juniper Networks QFabric™ system) 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 L2 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 L2 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.

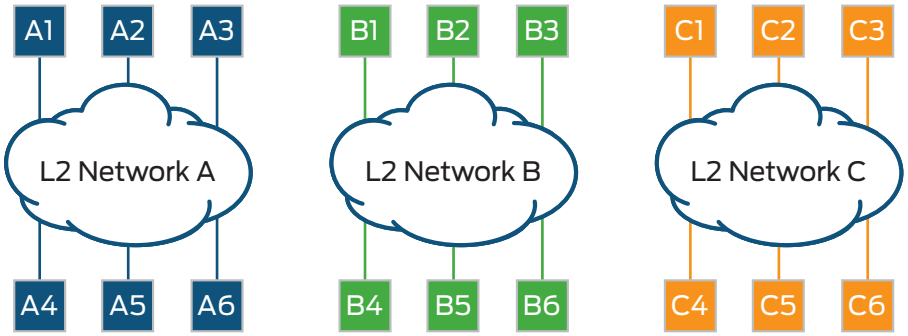


Figure 27: Network abstraction presented to tenants

The tenant private networks are generically called virtual networks—all virtual machines on a given tenant network are on the same L3 subnet. Tenants can be allowed to pick their own IP addresses for the VMs, or the cloud provider can assign the IP addresses. Either way, the IP addresses might not be unique across tenants (that is, the same IP address might be used by two VMs of two different tenants).

A single tenant can have multiple virtual networks. Those virtual networks might or might not be connected to each other using a L3 router, a firewall, a NAT, a load balancer, or some other service.

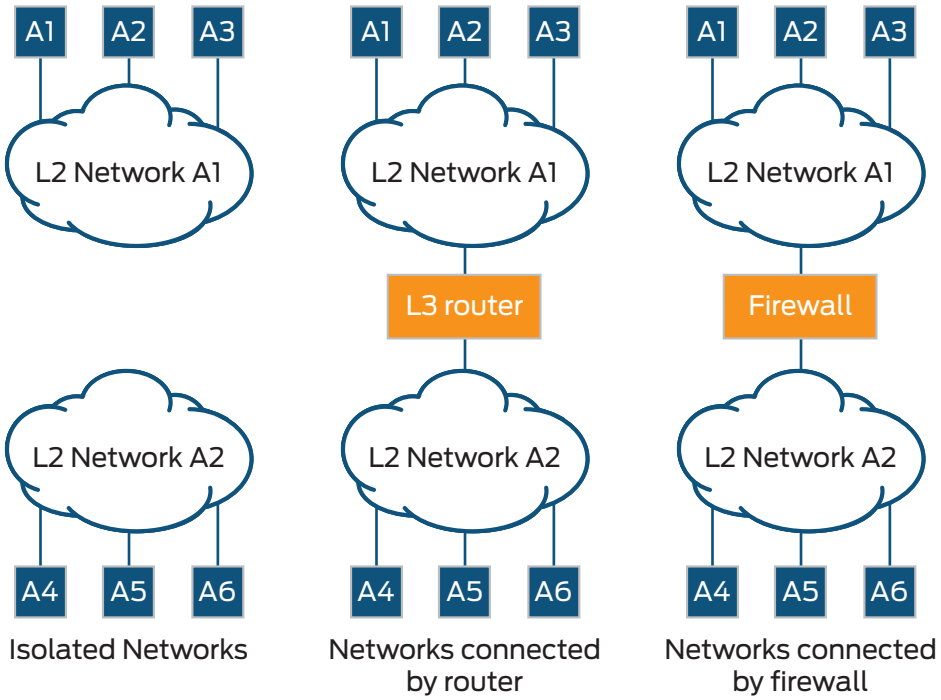


Figure 28: Multiple networks for a tenant

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, to the Internet, or to a customer network using routers or services (for example, NAT). [**AWS-VPC-SUBNETS**]

The use case includes a logically centralized orchestration layer (not shown in any of the previous diagrams) 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 L3VPN, L2VPN, an SSL VPN, an IPsec VPN, etc.

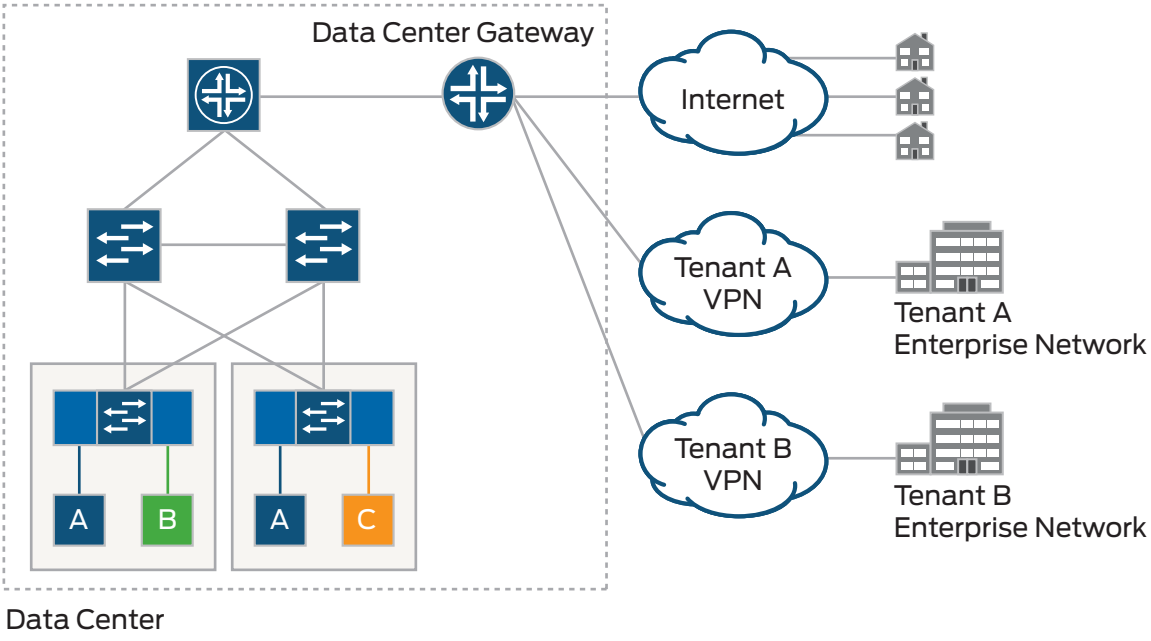


Figure 29: Use case "Connect Tenant to Internet / VPN"

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 (for example, use a gateway router).

Data Center Interconnect (DCI)

In this use case, multiple data centers are interconnected over a wide area network (WAN).

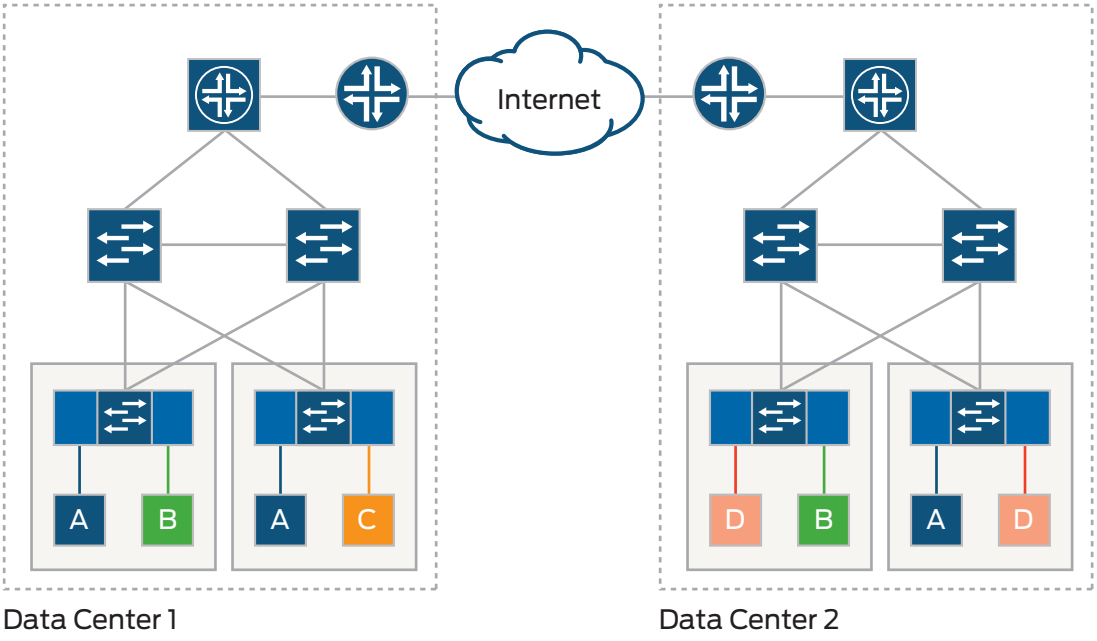


Figure 30: Use case "Data Center Interconnect (DCI)"

Data centers can be active/standby for disaster recovery, temporarily active/active for disaster avoidance, or permanently active/active. In the active/active case, a tenant might 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:

- Enable storage replication
- Allow tenant networks to use overlapping IP address spaces across data centers
- Provide global load balancing (GLB)
- Allow VM migration across data centers for disaster avoidance

Multiple transport options are available for DCI interconnect, including dark fiber, SONET/SDH, DWDM, pseudowires, Layer 3 VPNs, EVPNs, 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 might be temporary—for example, when debugging network issues. Or the monitoring might 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 do the following:

- It creates tunnels from the monitoring collection points (the "taps") in the network to the monitoring devices, which collect and analyze the traffic.
- It instructs 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 service (IDS), intrusion prevention systems (IPS), load balancers, SSL offloaders, 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.

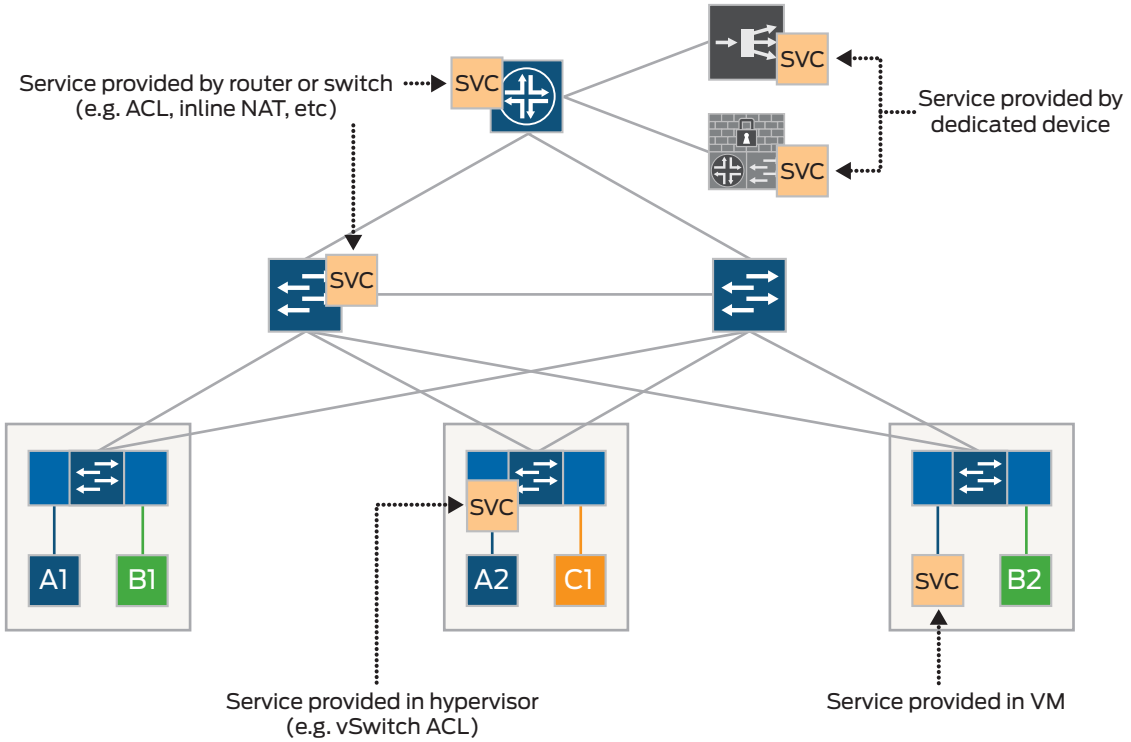


Figure 31: Service node locations

Services can 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
- Inline services on a router or switch on a service card or natively on the forwarding ASICs

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

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

- 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

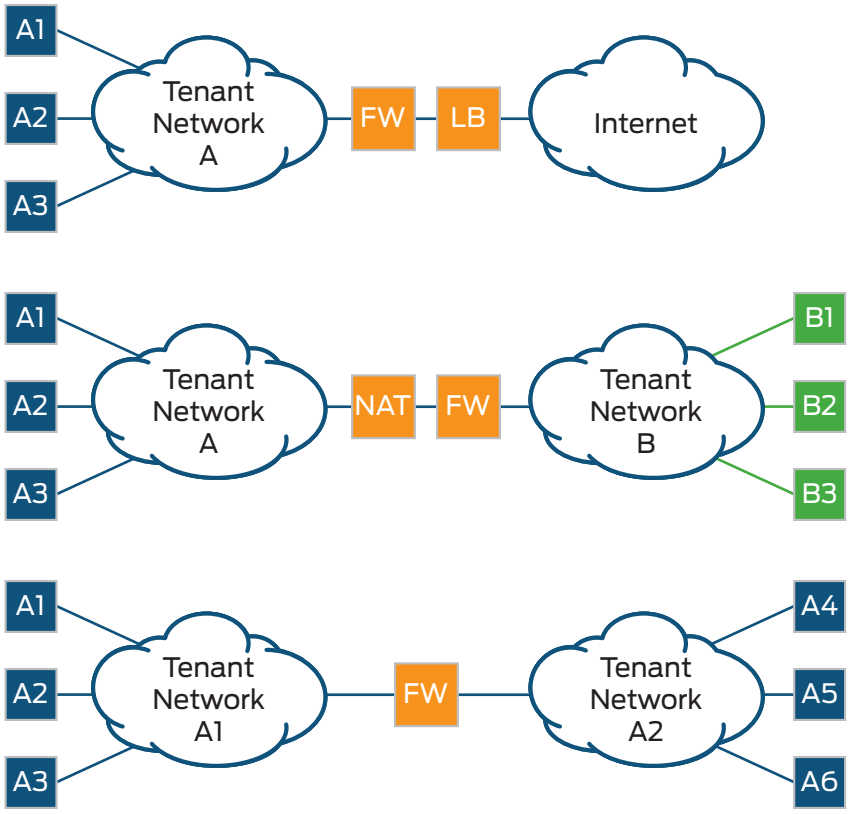


Figure 32: Services at network boundaries

Network Functions Virtualization for Service Provider Networks

Service Insertion

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

The SDN Controller 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) device such as a multiservices router. Operators need more functionality in these CPEs to compete with over-the-top (OTT) services but are challenged to do so because of the following:

- 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 that only implements basic layer 2/layer 3 functionality
- Virtualizing the remaining functionality in a virtual machine or container running on commodity x86 hardware that is centrally orchestrated and provisioned

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

- Tethered to the broadband network gateway (BNG)
- On a service card in the BNG
- Inline between the BNG and the CPE
- In a data center
- A combination of the aforementioned

Comparison of the Contrail System to MPLS VPNs

The architecture of the Contrail system is in many respects similar to the architecture of MPLS VPNs[2] as shown in Figure 33.

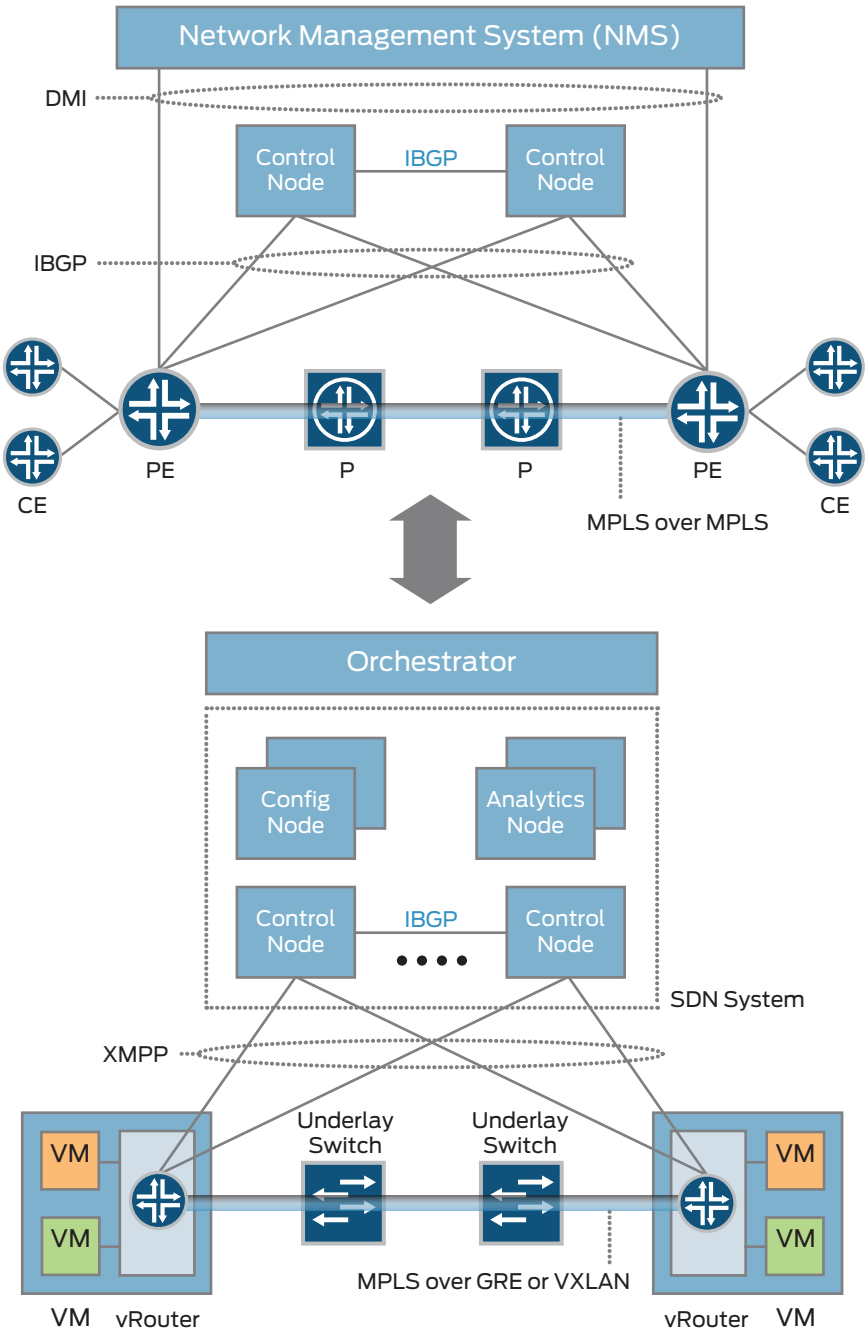


Figure 33: Comparison of the Contrail system to MPLS VPNs

[2]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.

The parallels between the two architectures include the following:

- Underlay switches in the Contrail system correspond to P routers in an MPLS VPN. Since the Contrail 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 detects how to forward unicast IP packets from one physical server to another.
- vRouters in the Contrail system correspond to PE routers in an MPLS VPN. They have multiple routing instances just like physical PE routers.
- VMs in the Contrail system correspond to CE routers in an MPLS VPN. In the Contrail 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 Contrail system correspond to MPLS over MPLS in MPLS VPNs.
- The XMPP protocol in the Contrail 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 (for example, routing instances) similar to what DMI does in MPLS VPNs.
- The Contrail system provides three separate pieces of functionality:
  - Centralized control is similar to a BGP route reflector in an MPLS VPN.
  - Management, pushes down configuration state to vRouters similar to a network management system (NMS) in an MPLS VPN.
  - Analytics.
- Contrail supports both layer 3 overlays, which are the equivalent of MPLS L3-VPNs and layer 2 overlays, which are the equivalent of MPLS EVPNs.

Acronyms

Acronym	Meaning
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	Commercial-Off-The-Shelf
CPE	Customer Premises Equipment
CSP	Cloud Service Provider
CO	Central Office
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
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 Routing Encapsulation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Sockets Layer
IaaS	Infrastructure as a Service
IBGP	Internal Border Gateway Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IF-MAP	Interface to Metadata Access Point
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 Name 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 Systems
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 of a multicast packet and destination multicast group address
SDH	Synchronous Digital Hierarchy
SDN	Software-Defined Networking
SONET	Synchronous Optical Network
SPAN	Switched Port Analyzer
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCG	Trusted Computing Group

TE	Traffic Engineering
TE-LSP	Traffic Engineered Label-Switched Path
TLS	Transport Layer Security
TNC	Trusted Network Connect
UDP	User 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

References

Amazon Web Services. **http://aws.amazon.com/**

Amazon Eleastic Compute Cloud (Amazon EC2) **http://aws.amazon.com/ec2/**

Amazon EC2 Elastic IP Addresses. **http://aws.amazon.com/articles/1346**

Amazon EC2 Instance IP Addressing. **http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html**

Amazon Virtual Private Cloud (Amazon VPC). **http://aws.amazon.com/vpc/**

Amazon Virtual Private Cloud Subnets. **http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Subnets.html**

Apache Cassandra website. **http://cassandra.apache.org/**

"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/**

"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/**

"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/**

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

"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 working group. **http://datatracker.ietf.org/wg/xmpp/**

IF-MAP.org website. **http://www.if-map.org/**

"Proactive Overlay versus Reactive End-to-End." Juniper Networks. **www.juniper.net/us/en/local/pdf/whitepapers/2000515-en.pdf**

Redis website. **http://redis.io/**

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

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

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

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

Snort website. **http://www.snort.org/**

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

XMPP Standards Foundation. **http://xmpp.org/**

Apache ZooKeeper website. **http://zookeeper.apache.org/**

Conclusion

Juniper Networks Contrail is a scale-out networking solution that creates virtual networks while seamlessly integrating with existing physical routers and switches. It automates service chaining of virtualized or physical network services, orchestrates networks across public, private, and hybrid clouds, and provides advanced analytics capability for automation, visualization, and diagnostics.

Contrail brings advanced networking capabilities to the cloud and eliminates the barriers to cloud adoption by making the cloud dynamic and flexible. Today, Contrail is offering a unique and significantly differentiated approach that is built on a number of key benefits:

- Provides a simple way to connect physical networks with a virtual environment and provision underlying services, reducing the time, cost, and risk for customers when configuring the network
- Uses NFV for service chaining to make provisioning and management of network and security services such as JunosV Firefly easy, enhancing the efficiency and agility with which customers deploy and use network resources
- Eliminates the risk of vendor lock-in by leveraging a standards-based architecture that integrates with a wide variety of hypervisors, physical networks, and orchestration platforms, including compatibility with both CloudStack and OpenStack
- Seamlessly integrates with most industry switches and routers, including MX Series, EX Series, and QFX Series appliances, to provide customers with a quick and easy migration path to SDN without any disruption to underlying physical network architecture and investment
- Accelerates the connection of virtual resources and enables the federation of private, public, or hybrid cloud environments, increasing the speed of business and service innovation by making the network more dynamic, flexible, and automated
- Speeds up troubleshooting and diagnostics through unique analytics capability, enabling customers to more intelligently and e_ciently manage their networks

About Juniper Networks

Juniper Networks is in the business of network innovation. From devices to data centers, from consumers to cloud providers, Juniper Networks delivers the software, silicon and systems that transform the experience and economics of networking. The company serves customers and partners worldwide. Additional information can be found at www.juniper.net.

Corporate and Sales Headquarters
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or +1.408.745.2000
Fax: +1.408.745.2100
www.juniper.net

APAC and EMEA Headquarters
Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
Phone: +31.0.207.125.700
Fax: +31.0.207.125.701

2000535-001-EN Sept 2015