O'REILLY®

EVPN in the Data Center

Dinesh G. Dutt

EVPN in the Data Center

This excerpt contains Chapter 2 of the book *EVPN in the Data Center*. The complete book is available on *oreilly.com* and through other retailers.

Dinesh G. Dutt

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

EVPN in the Data Center
by Dinesh G. Dutt

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Courtney Allen
Development Editor: Andy Oram
Production Editor: Justin Billing
Copyeditor: Octal Publishing, Inc.

Proofreaders: Andrew Clark
Dwight Ramsey
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

June 2018: First Edition

Revision History for the First Edition
2018-06-04: First Release
2018-07-13: Second Release

This work is part of a collaboration between O'Reilly and Cumulus Networks. See our *statement of editorial independence*.

978-1-492-02903-8

[LSI]

Table of Contents

Network Virtualization

Ethernet VPN (EVPN) is a technology for connecting Layer 2 (L2) network segments separated by a Layer 3 (L3) network. It accomplishes this by constructing a virtual L2 network over the underlying L3 network. This setting up of virtual network overlays is a specific kind of network virtualization.

So, we begin our journey to the world of EVPN by studying network virtualization. This chapter covers types of network virtualization, including in more detail the specific type of virtualization called Network Virtualization Overlays (NVOs). Staying true to a practitioner's handbook, this chapter largely focuses on understanding the ramifications of NVOs for a network administrator. We study network tunnels and their effects on administering networks. A little history provides context for the broader technology called *network virtualization* and adds color to the specifics of Virtual Extensible LAN (VXLAN), the primary NVO protocol used with EVPN within the data center. We conclude with a brief survey of alternate control-plane choices and the availability of network virtualization solutions. By the end of this chapter, you will be able to tease apart the meaning of the phrase "virtual L2 network overlay."

What Is Network Virtualization?

This section begins by examining the raison d'ĕtre for virtual networks. We then examine the different kinds of virtual networks, before concluding with the benefits and the challenges of overlay virtual networks.

Network virtualization is the carving up of a single physical network into many virtual networks. Virtualizing a resource allows it to be shared by multiple users. Sharing allows the efficient use of a resource when no single user can utilize the entire resource. Virtualization affords each user the illusion that they own the resource. In the case of virtual networks, each user is under the illusion that there are no other users of the network. To preserve the illusion, virtual networks are isolated from one another. Packets cannot accidentally leak from one virtual network to another.

Types of Virtual Networks

Many different types of virtual networks have sprung up over the decades to meet different needs. A primary distinction between these different types is their model for providing network connectivity. Networks can provide connectivity via bridging (L2) or routing (L3). Thus, virtual networks can be either virtual L2 networks or virtual L3 networks.

The granddaddy of all virtual networks is the Virtual Local Area Network (VLAN). VLAN was invented to reduce the excessive chatter in an L2 network by isolating applications from their noisy neighbors. Virtual Routing and Forwarding (VRF), the original virtual L3 network, was invented along with L3 Virtual Private Network (L3VPN) to solve the problem of interconnecting geographically disparate networks of an enterprise over a public network. When interconnecting multiple enterprises, the public network had to keep each enterprise network isolated from the other. This isolation also helped enterprises reuse the same IP address within their own enterprise. So how do virtual networks help with overlapping address spaces?

Network addresses must be unique only in a contiguously connected network. Consider old-fashioned postal addressing. A common model for a postal address is to use a numbered street address, the city, the state, and maybe the country. Within a city there can be only a single location that is addressed as 463 University Avenue. Similarly, within a state, you cannot have more than one city called Columbus, and within a country you cannot have multiple states called California. The uniqueness of an address is specific to the container it is in.

Similarly, a MAC address, which is an L2 address, needs to be unique only in a contiguously connected L2 network.[1] An IP address needs to be unique only within a contiguous L3 network. Because virtual networks provide the illusion of a single contiguous network, an address needs to be unique only within a virtual network. In other words, the same address can be present in multiple virtual networks. A MAC address is unique within a virtual L2 network. Similarly, an IP address is unique within a virtual L3 network. Packet forwarding uses a forwarding table that stores reachability to known destination addresses. Because a virtual network is carved out of a single physical resource, to allow address reuse, every virtual network gets its own logical copy of the forwarding table.

Virtual L2 and L3 networks behave just like their nonvirtual counterparts. The uniqueness of the MAC or IP address within a contiguous network is one example. Another example is that a device in one virtual L2 network can communicate with a device in a different virtual network via routing.

VLAN, VRF, and L3VPN highlight two other characteristics that distinguish different types of virtual networks. The first is the way in which a packet switching node decides to associate a packet with a virtual network. The second is whether transit nodes in a network path are aware of virtual networks.

The most common way to associate a packet with its virtual network is to carry a *Virtual Network Identifier* (VNI) in the packet header. VLAN, L3VPN, and VXLAN are examples of solutions carrying the VNI in the packet.[2] A less common way is to derive the virtual network at every hop based on the incoming interface and the packet header. Only the plain VRF model (without the L3VPN) uses this latter method.

In VLAN and VRF, every transit node needs to be aware of and process the virtual network to which the packet belongs. However, in L3VPN, the public network over which each enterprise's private network is transported is unaware that it is transporting multiple pri-

1 Anycast addresses, which can be used to represent a logical entity, can be shared by multiple physical entities. This is akin to the way bulk mail is addressed with "Resident of Sunnyvale."

2 In VLAN, the VNI is called VLAN ID, and in VPN it is called VPN ID. In VXLAN, it is called VNI.

vate networks. A virtual network implemented with protocols that leave the transit nodes unaware of it is called a *virtual network overlay*. This is because the virtual network looks like it is overlaid on top of the physical network. The physical network itself is called the *underlay network*. VLAN and VRF are called *inline virtual networks*, or non-overlay virtual networks. In the models of virtual networks, overlay virtual networks are widely considered to be more scalable and easier to administer. For the remainder of the book, we focus on this architecture.

Benefits of overlay virtual networks

The primary benefit of virtual network overlays over non-overlays is that they scale much better. Because the network core does not have to store forwarding table state for the virtual networks, it operates with much less state. In any network, the core sees the aggregate of all the traffic from the edges. So this scalability is critical. As a consequence, a single physical network can support a larger number of virtual networks.

The second benefit of overlay networks is they allow for rapid provisioning of virtual networks. Rapid provisioning is possible because you configure only the affected edges, not the entire network. To understand this better, contrast the case of a VLAN with that of an L3VPN. In the case of VLAN, every network hop along the path from a source to a destination must know about a VLAN. In other words, configuring a VLAN involves configuring it on every hop along the path. In the case of L3VPN, only the edges that connect to the virtual network need to be configured with information about that virtual network. The core of the L3VPN network is unaware of these virtual networks and so does not need to be configured.

The final major benefit of overlay networks is that they allow the reuse of existing equipment. Only the edges participating in the virtual networks need to support the semantics of virtual networks. This also makes overlays extremely cost effective. If you want to try out an update to the virtual network software, only the edges need to be touched, whereas the rest of the network can hum along just fine. In reality, this last benefit is a property of the solution chosen for the overlay, as we shall see in "The Consequences of Tunneling" on page 7.

Network Tunneling

The most common way to identify the virtual network of a packet is to carry a VNI in the packet. Where is the VNI carried? What is it called? How big is it? These questions have been answered more than once, alas with different answers each time. But the concept of a network tunnel is common to them all.

In real life, a tunnel connects two endpoints separated by something that prevents such a connection (such as a mountain). So it is with network tunnels, too. A network tunnel allows communication between two endpoints through a network that does not allow such communication.

Let's use Figure 1-1 to understand the behavior of network tunnels. R1, R2, and R3 are routers, and their forwarding table state is shown in the box above them. The arrow illustrates the port the router needs to send the packet out to reach the destination associated with that entry. In the upper part of the picture, R2 knows only how to forward packets destined to R1 or R3. So, when a packet from A to B reaches it from R1, R2 drops the packet. In the lower part of the picture, R1 adds a new header to the packet, with a destination of R3 and a source of R1. R2 knows how to forward this packet. On reaching R3, R3 removes the outer header and sends the packet to B because it knows how to reach B. Between R1 and R3, the packet is considered to be in a *network tunnel*. A common example of network tunnels that behave this way is the VPN from an employee's laptop at home to a lab machine in the office lab.

The behavior of R1, R2, and R3 resemble the behavior of a virtual network overlay. A and B are in a private network that is unknown to the core R2. This is why virtual network overlays are implemented using network tunnels.

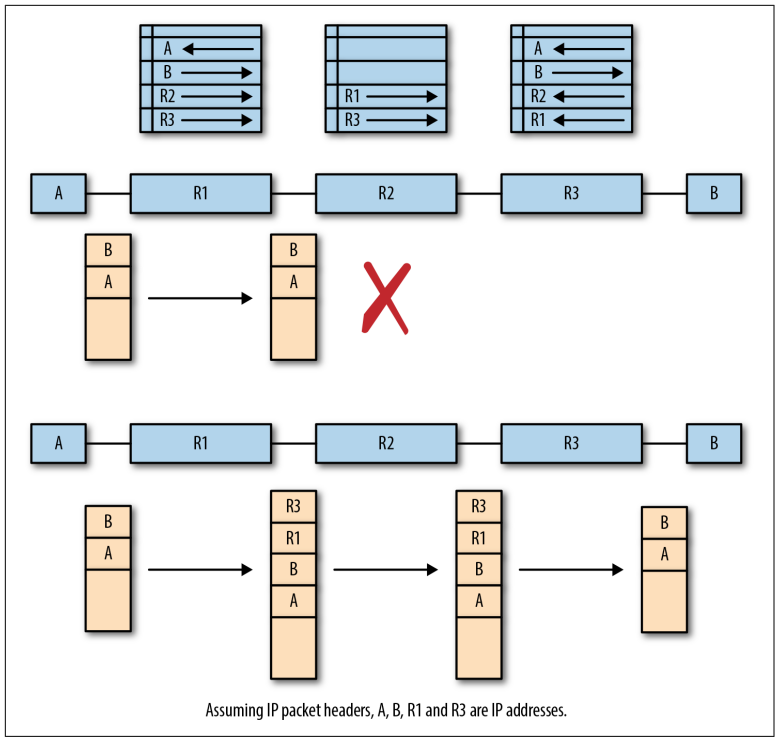

Figure 1-1. Illustrating network tunnels, when A sends a packet to B

In an overlay virtual network, a tunnel endpoint (R1 and R3 in Figure 1-1) is termed a *network virtualization edge* (NVE). The *ingress NVE*, which marks the start of the virtual network overlay (R1 in our example), adds the tunnel header. The *egress NVE*, which marks the end of the virtual network overlay (R3 in our example), strips off the tunnel header.

Network tunnels come in various shapes and forms. The tunnel header can be constructed using an L2 header or an L3 header. Examples of L2 tunnels include double VLAN tag (Q-in-Q or double-Q), TRILL, and Mac-in-Mac (IEEE 802.1ah). Popular L3 tunnel headers include VXLAN, IP Generic Routing Encapsulation (GRE) and Multiprotocol Label Switching (MPLS). L2 tunnel headers are of course constrained by their inability to cross an L3 boundary.

Network tunnels also specify whether their payload is an L2 packet or an L3 packet. Tunnels based on L2 headers always carry an L2

payload, whereas L3 tunnels can carry either an L2 payload or an L3 payload. The tunnel definition and setup define the kind of payload the tunnel will carry.

Another difference in network tunnels is whether they connect only two specific endpoints (called *point-to-point*) or one endpoint with multiple other endpoints (called *point-to-multipoint*). L3VPN with MPLS is an example of the former, and Virtual Private LAN Switching (VPLS) is an example of the latter.

The size of the VNI in each of these tunnels is different. MPLS defines a 20-bit VNI (called the VPN ID), whereas the other encapsulations use a 24-bit VNI. This means MPLS can carry 1 million (2^{20}) unique virtual networks, whereas the other tunnels can carry 16 million (2^{24}) unique virtual networks.

The Consequences of Tunneling

The primary benefit of these tunneling protocols was supposed to keep the core underlay from having to know anything about these virtual networks. However, there are no free lunches. The following subsections discuss traditional aspects of networking where virtualization has unintended consequences. Some of these we can address, whereas some others we cannot.

Packet Load Balancing

Tunneled (or encapsulated) packets pose a critical problem when used with existing networking gear. That problem lies in how packet forwarding works in the presence of multiple paths. In the presence of multiple paths to a destination, a node has the choice of either randomly selecting a node to which to forward the packet or ensuring that all packets belonging to a flow take the same path. A flow is roughly defined as a group of packets that belong together. Most commonly, a Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) flow is defined as the 5-tuple of source IP address, destination IP addresses, the Layer 4 (L4) protocol (TCP/UDP), the L4 source port, and the L4 destination port. Packets of other protocols have other definitions of flow. A primary reason to identify a flow is to ensure the proper functioning of the protocol associated with that flow. If a node forwards packets of the same flow along different paths, these packets can arrive at the destination in a different order from the order in which they were transmitted

by the source. This out-of-order delivery can severely affect the performance of the protocol. However, it is also critical to ensure maximum utilization of all the available network bandwidth; that is, utilize all the network paths to a destination. Every network node makes decisions that optimize both constraints.

When a packet is tunneled, the transit or underlay nodes see only the tunnel header. They use this tunnel header to determine what packets belong to a flow. An L3 tunnel header typically uses a different L4 protocol type to identify the tunnel type (IP GRE does this, as an example). For traffic between the same ingress and egress NVE, the source and destination addresses are always the same. However, a tunnel usually carries packets belonging to multiple flows. This flow information is after the tunnel header. Because existing networking gear cannot look past a tunnel header, all packets between the same tunnel ingress and egress endpoints take the same path. Thus, tunneled packets cannot take full advantage of multipathing between the endpoints. This leads to a dramatic reduction in the utilized network bandwidth. Early networks had little multipathing, and so this limitation had no practical impact. But multipathing is quite common in modern networks, especially data center networks, thus this problem needed a solution.

A clever fix for this problem is to use UDP as the tunnel. Network nodes have load balanced UDP packets for a long time. Like TCP, they send all packets associated with a UDP flow along the same path. When used as a tunnel header, only the destination UDP port identifies the tunnel type. The source port is not used. So, when using UDP for constructing tunnels, the tunnel ingress sets the source port to be the hash of the 5-tuple of the underlying payload header. Ensuring that the source port for all packets belonging to a TCP or UDP flow is set to the same value enables older networking gear to make maximal use of the available bandwidth for tunneled packets without reordering packets of the underlying payload. Locator Identity Separation Protocol (LISP) was the first protocol to adopt this trick. VXLAN copied this idea.

Network Interface Card Behavior

On compute nodes, a network interface card (NIC) provides several important performance-enhancing functions. The primary ones include offloading TCP segmentation and checksum computation for the IP, TCP, and UDP packets. Performing these functions in the

NIC hardware frees the CPU from having to perform these compute-intensive tasks. Thus, end stations can transmit and receive at substantially higher network speeds without burning costly and useful CPU cycles.

The addition of packet encapsulations or tunnels foils this. Because the NIC does not know how to parse past these new packet headers to locate the underlying TCP/UDP/IP payload or to provide additional offloads for the tunnel's UDP/IP header, the network performance takes a significant hit when these technologies are employed at the endpoint itself. Although some of the newer NICs understand the VXLAN header, this problem has been a primary reason VXLAN from the host has not taken off. So, people have turned to the network to do the VXLAN encapsulation and decapsulation. This in turn contributed to the rise of EVPN.

Maximum Transmission Unit

In an L3 network, every link is associated with a maximum packet size called the *Maximum Transmission Unit* (MTU). Every time a packet header is added, the maximum allowed payload in a packet is reduced by the size of this additional header. The main reason this is important is that modern networks typically do not fragment IP packets, and if end stations are not configured with the proper reduced MTU, the introduction of virtual networks into a network path can lead to difficult-to-diagnose connectivity problems.

Lack of Visibility

Network tunnels obscure the ecosystem they plow through. Classic debugging tools such as traceroute will fail to reveal the actual path through the network, presenting instead the entire network path represented by the tunnel as a single hop. This means troubleshooting networks using tunnels is painful.

VXLAN

VXLAN is a relatively new (only eight years old) tunneling technology designed to run over IP networks while providing L2 connectivity to endpoints. It uses UDP/IP as the primary encapsulation technology to allow existing network equipment to load balance packets over multiple paths, a common condition in data center net-

works. VXLAN is primarily deployed in data centers. In VXLAN, the tunnel edges are called *VXLAN tunnel end points* (VTEPs).

Figure 1-2 shows the packet format of VXLAN.

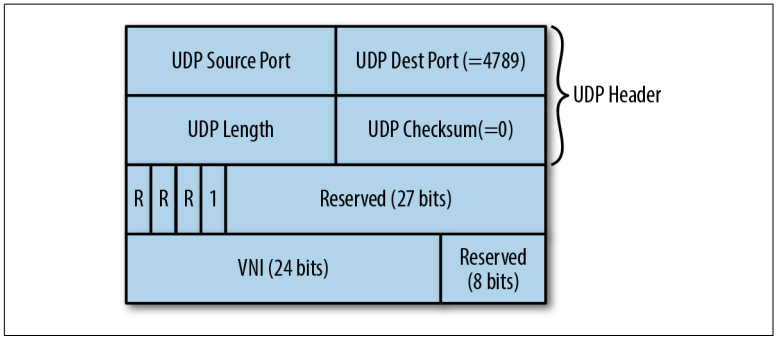

Figure 1-2. VXLAN header

As mentioned in "Packet Load Balancing" on page 7, the UDP source port is computed at the ingress VTEP using the inner payload's packet header. This allows a VXLAN packet to be correctly load balanced by all the transit nodes. The rest of the network forwards packets based on the outer IP header.

VXLAN is a point-to-multipoint tunnel. Multicast or broadcast packets can be sent from a single VTEP to multiple VTEPs in the network.

You might have noticed several oddities in the header. Why did we need yet another tunneling protocol? Why is the VNI 24 bits? Why are there so many reserved bits? The entire VXLAN header could have been just 4 bytes, so why is it 8? Why have a bit that is always 1? The main reason for all this is historical, and I am mostly responsible for this.

History Behind the VXLAN Header

Circa 2010, Amazon's AWS had taken off in a big way, especially its elastic compute service (ECS). VMWare, the reigning king of virtualizing compute, approached Cisco, the reigning king of networking, for help with network virtualization. VMWare wanted to enable its enterprise customers to build their own internal AWS-like infrastructures (called private clouds). VMWare wanted an L2 virtual network, like VLANs, but based on an overlay model with

the ability to support millions of virtual networks. It also wanted the network to be IP-based due to IP's ubiquity and better scalability than L2-based technologies. The use of MPLS was nonstarter because MPLS was considered too complex and not supported inside an enterprise.

As one of the key architects in the data center business unit at Cisco, I was tasked with coming up with such a network tunnel. I first looked at IP-GRE, but then quickly rejected it because we wanted a protocol that was easy for firewalls to pass. Configuring a UDP port for passage through a firewall was easy, but an L4 protocol like GRE was not. Moreover, GRE was a generic encapsulation, with no specific way to identify the use of GRE for purposes other than network virtualization. This meant the header fields could be used differently in other use cases, preventing underlying hardware from doing something specific for network virtualization. I was tired of supporting more and more tunneling protocols in the switching silicon, each just a little different.

I already had over-the-top virtualization (OTV—a proprietary precursor to EVPN) and LISP protocols to support. I wanted VXLAN to look like OTV and for both to resemble LISP, given that LISP was already being discussed in the standards bodies. But there were already existing OTV and LISP deployments, so whatever header I constructed had to be backward-compatible. Thus I made the VNI 24 bits because many L2 virtual networks already supported 24-bit VNIs, and I didn't want to build stateful gateways just to keep VNI mappings between different tunneling protocols. The reserved bits and the always 1 bit are there because those bits mean something else in the case of LISP and OTV. In other words, the rest of the header format is a consequence of trying to preserve backward compatibility. The result is the VXLAN header you see.

Protocols to Implement the Control Plane

The control plane in a network overlay solution has to provide the following:

- A mechanism to map the inner payload's destination address to the appropriate egress NVE's address.

- A mechanism to allow each NVE to list the virtual networks it is interested in, to allow point-to-multipoint communication such as broadcast.

Because VXLAN is an example of a virtual L2 network, the mapping of the inner MAC address to the outer tunnel egress IP address is a mapping of a {VNI, MAC} tuple to the NVE's IP address. Therefore, the forwarding table typically involved in providing this mapping is the MAC forwarding table. We'll see why this is important in Chapter 5.

VXLAN was designed to allow compute nodes to be the NVEs. Because the spin up and spin down of virtual machines (VMs) or containers was known only to the compute nodes, it seemed sensible to allow them to be NVEs. Furthermore, making the compute nodes the NVEs meant that the physical network itself could be quite simple.

Multiple software vendors signed up to provide such a solution. Examples of such solutions include VMWare's NSX, Nuage Networks, and Midokura. Networks running VXLAN were the original Software-Defined Network (SDN), where the software (orchestration software that spun up new VMs and their associated virtual networks) controlled the provisioning of the virtual network over an immutable underlay. But for various reasons, this solution did not take off the way it was expected.

An alternate approach to this SDN solution was to rely on traditional networking protocols such as Border Gateway Protocol (BGP) to provide this mapping information. EVPN belongs to this category of solutions, which are called controller-less VXLAN.

Support for Network Virtualization Technologies

We conclude our journey on network virtualization with a survey of what is supported, both in the open networking ecosystem as well as with traditional networks. We also briefly examine the work in various standards bodies associated with these technologies.

Merchant Silicon

The era of networking companies building their own custom switching Application-Specific Integrated Circuits (ASICs) is seemingly near the end. Everyone is increasingly relying on merchant silicon vendors for their switching chips. As if to highlight this very switch (pun intended), just about every traditional networking vendor first supported VXLAN on merchant switching silicon. Broadcom introduced support for it with its Trident2 platform, adding VXLAN routing support in the Trident2+ and Trident3 chipsets. Mellanox first added support for VXLAN bridging and routing in its Spectrum chipset. Other merchant silicon vendors such as Cavium via its Xpliant chipset and Barefoot Networks also support VXLAN, including bridging and routing. All these chips also support VRF. Most switching silicon at the time of this writing did not support using IPv6 as the VXLAN tunnel header. VXLAN of course happily encapsulates and transmits inner IPv6 payloads.

Software

The Linux kernel itself has natively supported VXLAN for a long time now. VRF support in the Linux kernel was added by Cumulus Networks in 2015. This is now broadly available across multiple modern server Linux distributions (for example, as early as Ubuntu's 16.04 had basic IPv4 VRF support). The earliest kernel version with good, stable support for VRF is 4.14.

Cumulus Linux in the open networking world as well as all traditional networking vendors have supported VXLAN for several years. Routing across VXLAN networks is newer. Although the Linux kernel has supported routing across VXLAN networks from the start, some additional support that was required for EVPN has been added. We'll examine these additions in the subsequent chapters.

Standards

The Internet Engineering Task Force (IETF) is the primary body involved with network virtualization technologies, especially those based on IP and MPLS. VXLAN is an informational RFC, RFC 7348. Most of the network virtualization work is occurring under the auspices of the NVO3 (Network Virtualization Over L3) working group at the IETF. Progress is slow, however. Except for some

agreement on basic terminology, I'm not aware that any work from the NVO3 working group is supported by any major networking vendor or by Linux. However, EVPN-related work is occurring in the L2VPN working group. Aspects of EVPN in conjunction with VXLAN is still in the draft stages of the standards workflow. But the specification itself has been stable for quite some time and the base specification was made a standard document in early 2018. Multiple vendors, along with FRR (Free Range Routing, the open source routing suite), support EVPN with most of its major features.

Summary

In this chapter, we studied the fundamental technology and ideas behind network virtualization. In Chapter 3, we switch gears to look at the fundamentals of EVPN support. This chapter also introduces how EVPN configuration in the data center differs from its counterpart in the service provider world.

About the Author

Dinesh G. Dutt has been in the networking industry for the past 20 years, most of it at Cisco Systems. Most recently, he was the chief scientist at Cumulus Networks. Before that, he was a fellow at Cisco Systems. He has been involved in enterprise and data center networking technologies, including the design of many of the ASICs that powered Cisco's mega-switches such as Cat6K and the Nexus family of switches. He also has experience in storage networking from his days at Andiamo Systems and in the design of FCoE. He is a coauthor of TRILL and VxLAN and has filed for over 40 patents.