O'REILLY®

Compliments of CUMULUS

EVPN in the Data Center

Dinesh G. Dutt

EVPN in the Data Center

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

Acknowledgments

I want to acknowledge the people who were instrumental to me in the creation of this work.

First on this list are my editors at O'Reilly. Courtney Allen supported and nurtured my desire to write. Without her steadfast support, I doubt this book would have seen the light of day. Andy Oram who has done most of the editing has been nothing but tireless and prompt in his reviews, encouraging and thorough in his editing, and always pushing me to clarify my explanations. Courtney and Andy, thank you for round two.

Next up are the engineers at Cumulus Networks who have been among the most brilliant and supportive engineers I've worked with. Specifically, Vivek Venkataraman and Roopa Prabhu fielded my calls at all hours and never complained—at least to me :). Vivek and his FRR team worked with me to make the FRR model for EVPN simple and intuitive.

Pete Lumbis, also at Cumulus, reviewed the book on short notice, taking on this work in addition to the million other things he does. Neela Jacques, a close friend, read the initial drafts of the first chapters and helped me clarify the explanations to be understandable to non-engineers, as well. Thank you both for helping me make this book better.

My daughter and wife, Maya and Shanthala, rolled their eyes and put up with the side effects of my writing a second time. And I was afraid that my parents, who encouraged me throughout my life, would burst with pride and joy. Thank you all for nurturing and sustaining me through life.

And you, my reader, who makes all this toil fruitful, thank you for your encouragement and support of my first book. I hope you find this book useful, too.

CHAPTER 1
Introduction to EVPN

A wet California winter and spring had started to make way to sunny summer skies when I was invited to meet with a large financial company. The organization wanted me to critique its data center network design. Its use case revolved around a Layer 3 (L3) network. Clos-based topology was the basic network architecture it had chosen. Everything was done as nicely as I could suggest. No longer did I have to explain why the company had to move away from bridging as the centerpiece of its data center or why Clos networks were a better fit. One more conversion accomplished. I moved on.

As the summer turned to fall, the company approached me again to discuss a new constraint it had to deal with. The enterprise was going to deploy a new storage cluster solution in the network. This solution expected a Layer 2 (L2) connectivity to work. Needless to say, the L2 connectivity had to be across multiple racks. "Dinesh, how do I fit a solution that expects L2 connectivity in a network that has L3 as its foundation?" engineers at the company asked.

Increasingly that fall, I heard the same refrain over and over again. "How do I deploy an application that requires L2 in an L3 network?"

Another group of companies I spoke to were building new data centers and wanted to embrace the new world of white boxes and Clos networks. They had newer applications either like Hadoop or that relied on constucts like containers, so the new world was a great fit. Yet another group of companies wanted to upgrade from the buggy, difficult-to-maintain, and less reliable L2 heavy networks with the modern, resilient, robust world of Clos topologies. But they all had

to sooner or later deal with their legacy applications. Some decided to build a different, smaller, sunset network for these applications. Others wanted to figure out how to make the new network support these older applications. "After all, haven't you been saying that Clos networks are a Lego building block that can support myriad use cases?" they asked.

Some of these newer applications continue to rely on L2 multicast and broadcast for cluster membership discovery and heartbeat. The other common reliance on bridging comes from the assumption that the IP address of an endpoint stays the same, even when the endpoint is destroyed and re-created elsewhere. There are solutions to pass around /32 routes using either routing from the host or ideas such as redistribute Address Resolution Protocol (ARP). Nevertheless, support concerns and age-old habits limited virtual machine or container mobility to L2. And, of course, the older applications built for the old world could not be rewritten or decommissioned.

In the simplest of terms, Ethernet VPN (EVPN) is a technology that connects L2 network segments separated by an L3 network. EVPN accomplishes this by building the L2 network as a virtual Layer 2 network overlay over the Layer 3 network. It uses Border Gateway Protocol (BGP) as its control protocol.

EVPN is a mature technology that has been available in Multiprotocol Label Switching (MPLS) networks for some time. A draft standard that adopted this to Virtual Extensible LAN (VXLAN) has been available and relatively stable with multiple vendor implementations. There has been a lot of additional work in progress at the IETF (Internet Engineering Task Force), the standards body that governs IP-based technologies. In short, EVPN has slowly been gathering force as the alternative to controller-based VXLAN solutions. And by the summer of 2017, its moment in the data center had come.

Companies adopted VXLAN and the world of network virtualization but wanted native VXLAN routing (or RIOT, as it is often called, for Routing In and Out of Tunnels). Network operators had tried to love the one they were with and failed. Merchant switching silicon with RIOT support started to arrive in volumes to support real deployments. The missing piece was a technology that enabled this new functionality without the use of controllers. EVPN was that missing piece.

What had happened to the promised world of Software-Defined Networking (SDN), where endpoints would set up and control their own membership lists and the network had a single job as the great connector? For one reason or the other, some technical and some not, that play had failed to be the blockbuster it had been promised to be.

So, why should you pick up this book? If you perform a web search for EVPN, I venture that what you'll uniformly find is something that is very complex to understand. Owing to EVPN's origins in the service provider (SP) world, the standards document is peppered with terminology that does not make sense in the data center world. Furthermore, the explanations of even the most basic concepts are spread across several documents, leaving the task of piecing it all together to you.

My aim is to explain EVPN in the simplest terms possible—to make the technology accessible so that network operators and architects can understand its use for the cases cited at the beginning of this book. And hopefully, the book does more than that, explaining the concepts and practicalities in a way that helps you to use it in other, novel cases. This is a book that explores the *why*, not just the *how*. I remain vendor agnostic in all this to the extent possible.

And I expect you, my reader, to be a network architect or network operator. I assume that you are somewhat familiar with the basics of BGP and Clos networks. If not, I recommend, if a little abashedly, the prequel to this book, *BGP in the Data Center* (O'Reilly, 2017), for more detailed explanations of these concepts.

The story begins with a study of the two basic building blocks of EVPN: network virtualization, and the adaptation of BGP to the needs of network virtualization. We then explore how bridging and routing work in an EVPN world. After that, we turn to the configuration and management of EVPN networks. We conclude with some thoughts on considerations for deploying EVPN in larger deployments. I do not discuss L3 multicast and the data center interconnect use cases in this book. They're evolving quickly, from both a standards and a deployment standpoint. Although some early implementations are available, I prefer to see a little more experience before talking about them in more than generic terms.

The hounds of complexity are forever at the gates. EVPN is a complex piece of technology, but one that you can tame, if you refrain

from chasing after every single knob and optimization drafted and designed and sold. If you choose perfection as the destination, you savor it but for a moment, as the ever-changing world barges in. If you choose perfection as a journey, you can savor it much longer. One of the key ingredients of success is the KISS principle—Keep it Simple, Stupid—that has made networks, especially in the data center, interesting, scalable, and reliable. Keep your intent simple, and you don't need to pay others to decipher your intent, often to their benefit, not yours.

If there is one takeaway and one alone, it is that EVPN in the data center can be a far simpler and, dare I say, more attractive beast than its SP cousin.

And, oh yes, the large financial company that I referred to earlier has deployed EVPN.

Software Used in This Book

I have used the open source routing suite FRR as the basis of configuration and examples, largely because it is open source and shows how simple EVPN configuration can be. There is a companion GitHub site to this book that allows you to use Vagrant to build out and play with the topology and configuration described in Chapter 6.

CHAPTER 2

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'être 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 11.

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 2-1 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.

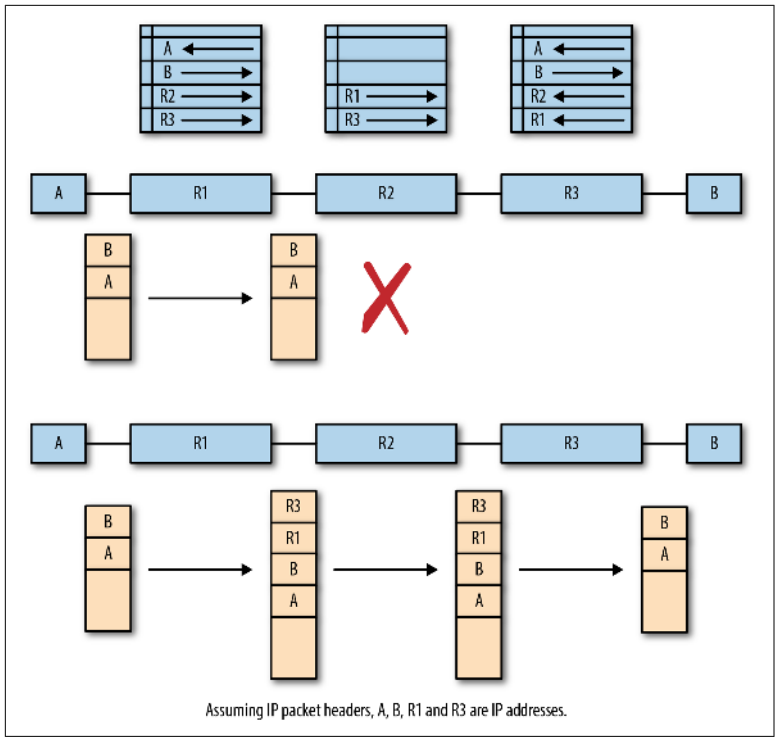

Figure 2-1. Illustrating network tunnels, when A sends a packet to B

In an overlay virtual network, a tunnel endpoint (R1 and R3 in Figure 2-1) 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 2-2 shows the packet format of VXLAN.

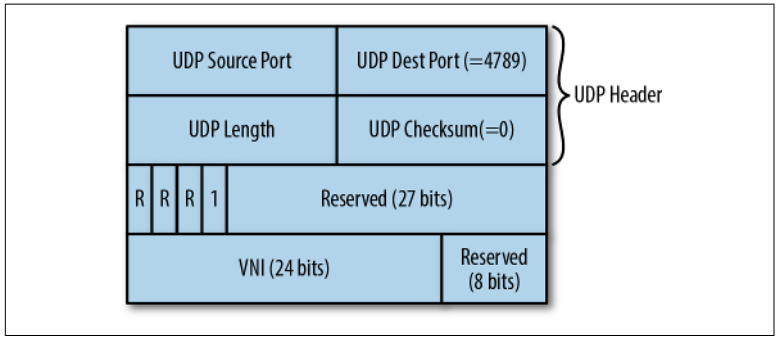

Figure 2-2. VXLAN header

As mentioned in "Packet Load Balancing" on page 11, 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.

CHAPTER 3

The Building Blocks of Ethernet VPN

Border Gateway Protocol (BGP) is the brain that drives Ethernet VPN (EVPN). In this chapter, we study the building blocks of BGP that are responsible for the construction of virtual network overlays. There are two facets to these building blocks: the BGP peering model, and the model for exchanging routing information.

The reason to study these models is to help a network administrator understand how to deploy EVPN in the data center. The primary takeaway of this chapter is that EVPN deployment in the data center can be simpler than its service provider (SP) counterpart.

Let's kickstart the chapter with a brief history of EVPN. This helps the reader understand the motivation behind EVPN. We next look at how BGP peering for EVPN was designed for the SP network. This leads us to see how BGP peering works in the data center in the absence of EVPN and how this affects the way EVPN is deployed in the data center. The next section deals with the fundamental BGP constructs that EVPN uses. The final two sections deal with additional constructs necessary to allow EVPN to work with external Border Gateway Protocol (eBGP) in Free Range Routing (FRR) and non-FRR routing suites. At the end of this chapter, you should be able to understand the choices in deploying EVPN in the data center.

A Brief History of EVPN

Virtual Private Networks (VPNs) interconnect multiple private networks across a public network. As discussed in Chapter 2, the interconnection can happen at Layer 2 (L2) or at Layer 3 (L3). L2 VPNs originally mimicked the model of L2: flood[1] a packet when the destination is unknown and use the spanning-tree protocol as the control plane to prune redundant paths. Virtual Private LAN Service (VPLS) is an example of L2VPN. But the inefficiency of flood-and-learn is well known. EVPN was born as an answer to this problem.

The precursor to EVPN was Over-the-Top Virtualization (OTV), a proprietary technology invented by Dino Farinacci at Cisco. It used Intermediate System–to–Intermediate System (IS-IS) as the control plane and ran over IP networks. IS-IS can build paths for both unicast and multicast routes. Juniper Networks first introduced EVPN as an answer to OTV. The company made a few fundamental changes to OTV in its proposal for EVPN. First, it used Multiprotocol Label Switching (MPLS) instead of IP networks. Next, the company proposed the use of BGP as the control plane protocol. Finally, Cisco offered it as a standard to the IETF (Internet Engineering Task Force), the standards body for Internet Protocols. MPLS was chosen because it is very flexible and more popular with SPs. Thanks to its use of MPLS and as a vendor-neutral standard, administrators began to adopt EVPN. EVPN is now a widely deployed standard. The document that defines the EVPN standard is RFC 7432.[2] With the advent of Virtual Extensible LAN (VXLAN) in the data center, EVPN was adopted as the solution for network virtualization in the data center. A new standard in the IETF, RFC 8365, explains some of the changes suggested for use within the data center. It is this standard that all routing suites, FRR or vendor-specific, support for use in the data center support today.

1 Flooding is defined as sending a packet out to all ports that carry the virtual network of the packet except the port on which it came in. We discuss this in more detail in Chapter 4.

2 RFC stands for "Request for Comments."

Architecture and Protocols for Traditional EVPN Deployment

Most mid- to large-sized enterprises have multiple border routers. The primary reason for this is reliability. But this can be for other reasons, too, such as their networks being spread across multiple geographically separated locations.

Each enterprise border router peers with a service provider edge (PE) router. They peer over eBGP to the PEs to advertise their locally attached L2 addresses. The relevant PEs (which belong to the same SP) peer with each other using iBGP to distribute these routes. The data traffic goes over this core public network encapsulated, typically MPLS-encapsulated. Figure 3-1 shows such a network along with the internal Border Gateway Protocol (iBGP) peering between PEs. CE is the Customer Edge router that wishes to use the VPN service. The edge networks are shown as two separate virtual networks from the PE's perspective, peering between PEs.

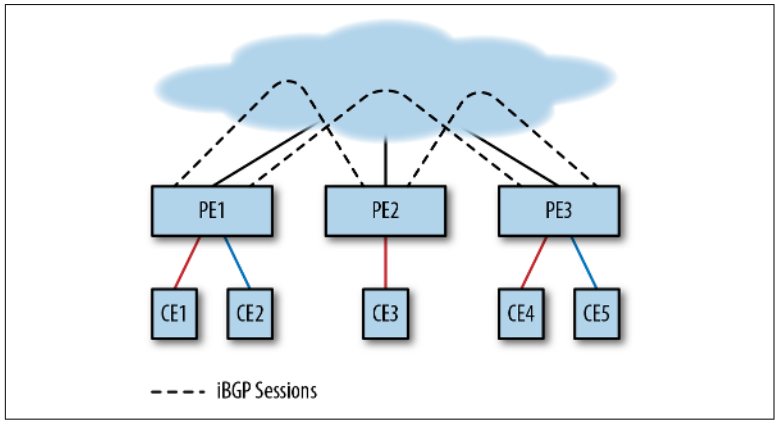

Figure 3-1. VPN setup in an SP world

The PEs usually belong to the same SP network, and reachability between the peers is established via an Interior Gateway Protocol (IGP) routing protocol such as Open Shortest Path First (OSPF) or IS-IS.

iBGP peering is full mesh; that is, every iBGP router is connected to every other iBGP router in the same administrative domain. This obviously does not scale when there are lots of routers. Thus, the most common alternative is to use something called Route Reflec-

tors (RRs). With RRs, every iBGP speaker peers with one or more RRs. Think of the RR as the hub, and the various iBGP peers as the spokes. An RR reflects only the BGP announcements (after computing best path) to its peers. It is rarely involved in the actual data path. So, the administrator chooses nodes that are centrally located in the network as RRs.

To summarize, EVPN deployments in the service provider world use iBGP, with a different underlying protocol providing connectivity between the iBGP peers. In other words, there are usually two separate protocols, one for the underlay and the other for the overlay.

EVPN in the Data Center

In the modern data center, the fundamental network topology is the *Clos topology* or the *leaf-spine network*. Figure 3-2 shows an example of a Clos network. In this topology, the leaf switch (also called Top-of-Rack [ToR]) is usually the VXLAN Tunnel End Point (VTEP), the edge of the virtual network. These correspond to the PEs in the traditional EVPN deployment. Therefore, if you assume the traditional model for EVPN deployment, there is an IGP between the routers in the Clos, and iBGP between the VTEPs for EVPN.

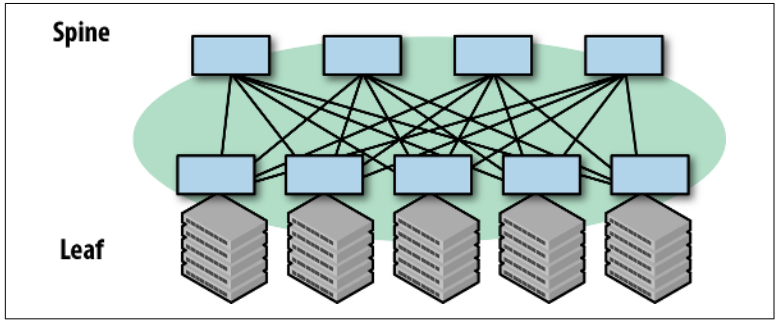

Figure 3-2. Example of a 2-tier Clos network

In a data center, OSPF is sometimes used as the routing protocol between the spine and leaf switches. In such a model, the SP model of using iBGP over OSPF makes sense. If your network is built with OSPF and you're comfortable with it, you can stick with the traditional model of deploying EVPN. However, the most common protocol I've encountered within the data center is eBGP. In other words, eBGP is the underlay protocol in the data center. A blind

adherence to the traditional EVPN deployment leads to using both eBGP and iBGP: eBGP for the underlay network, and iBGP for the overlay network. In a typical Clos network, because there are lots of leaves, the use of RRs is essential. In such a network, the spine switches seem a natural shoo-in for RRs. Therefore, traditional EVPN deployment translates to two physically adjacent BGP peers having two peering sessions: eBGP for non-EVPN addresses, and iBGP for EVPN addresses. Figure 3-3 illustrates this.

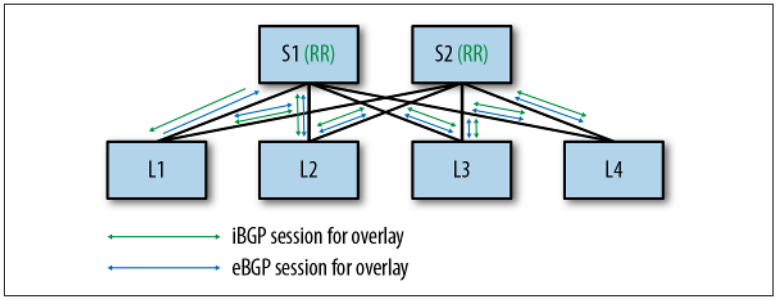

Figure 3-3. Traditional model of EVPN deployment in the data center

This use of a dual protocol feels needlessly complex. Another model I've seen is the use of separate eBGP sessions, one per address-family. In this model, there's a separate eBGP session for the underlay IPv4 addresses, and a second eBGP session for the overlay EVPN addresses. In my own personal opinion and after speaking with many BGP experts, this option is also redundant and unnecessary for a Clos topology.

Free Range Routing (FRR), the open source routing suite I use throughout this book, eliminates this need for dual peering sessions. A single eBGP session can carry both underlay and overlay peering information (Figure 3-4). In essence, this makes L2 reachability announcements just another address family advertisement[3]—not very different, say, from IPv6 advertisements. This combined with a few other sane defaults (which we discuss in "FRR Support for EVPN" on page 31) makes configuring EVPN with FRR trivial. A few other vendors support this simplified peering model, too, though not all of them do. FRR also supports the traditional model of deploying EVPN.

3 See "Address Family Indicator/Subsequent Address Family Indicator" on page 25 later in this chapter for more about address families.

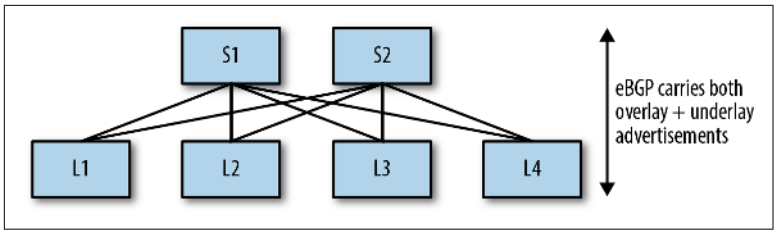

Figure 3-4. Simplified EVPN deployment model for the data center

Remember that this book is aimed at data centers. The SP model makes no sense in the data center world, and vice versa. In the traditional model, the eBGP peers belong to different organizations. Asking an eBGP peer of a different organization to handle VPN information is a gross violation of boundaries.

To summarize, EVPN in the data center uses a single eBGP session to advertise both L2 and VTEP reachability. As we shall see, the reliance of eBGP instead of iBGP has ripple effects across network configuration options.

BGP Constructs for Virtual Networks

In the previous section, we studied how to establish peering to exchange routing information. In this section, we look at the basic concepts necessary when building the information to be exchanged over this channel. The combination of these two sections helps in understanding how BGP assists in constructing virtual networks, EVPN or otherwise. I also introduce route types, the fundamental object EVPN uses to exchange information.

For a control protocol to exchange information about virtual networks, it needs to support three primary constructs:

- A way to identify the network address being exchanged (the role of AFI/SAFI)
- Identifying which virtual network an address belongs to (the role of RD and RT)
- Identifying what tunnel encapsulation method is used to build the virtual network overlay (indicated via the BGP Encapsula-

tion Extended Community used with all EVPN advertisements, see RFC 8365)

Finally, the specific virtual network overlay technology has to construct its message types to fit within the standards specified by BGP to exchange its specific information.

We study how BGP addresses each of these points in the next few subsections.

Address Family Indicator/Subsequent Address Family Indicator

BGP can advertise how to reach not just IP addresses, but also MPLS labels and MAC addresses. The basic standard that defines support for multiple kinds of addresses is RFC 4760. Each network protocol supported by BGP has its own identifier, called the Address Family Indicator (AFI). The AFI identifies the primary network protocol. For example, IPv4 and IPv6 each have their own AFI. However, even within an AFI, there is a need for further distinctions. For example, unicast and multicast reachability information is quite different. BGP distinguishes these cases by using separate Subsequent Address Family Indicator (SAFI) numbers for unicast and multicast addresses. All BGP configuration specific to a network protocol is grouped under an AFI/SAFI section.[4]

The AFI/SAFI list that is of interest to a BGP speaker will be advertised using BGP Capabilities in the BGP OPEN message. Two BGP peers will exchange information about a network address only if both sides advertise an interest in its AFI/SAFI. EVPN is designed as a SAFI family of the L2VPN AFI. Thus, in BGP lingo, EVPN's AFI/SAFI is l2vpn/evpn.

Route Distinguisher

As discussed in Chapter 2, virtual networks allow the reuse of an address. In other words, an address is unique only within a virtual network. A common, well understood example of this is the use of the 10.x.x.x subnet in IPv4. The 10.x address space is a private

4 BGP has roughly two sections in its configuration: the general part and an AFI/SAFI-specific part.

address space, so different organizations can reuse the address with impunity. Similarly, different virtual networks can reuse the same 10.x IPv4 address. This is true also for L2 addresses. So, we need a way in BGP to separate the advertisement of an address in one virtual network from the same address in a different virtual network. That is the job of a *Route Distinguisher* (RD).

When exchanging VPN addresses, BGP prepends an 8-byte RD to every address. This combination of RD + address makes the address globally unique. Section 4.2 of RFC 4364 defines RD, its formats, and its use. There are three different RD formats. The RD format used in EVPN is defined by RFC 7432. Figure 3-5 demonstrates the format.

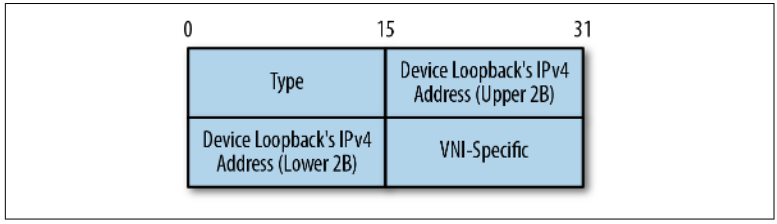

Figure 3-5. Format of RD used in EVPN

You might have spotted that two bytes isn't sufficient to encode the three bytes that identify the VNI in VXLAN. This is not considered an issue, because it is assumed that no VTEP will, in practice, host more than 64,000 VNIs. Most switching hardware doesn't support this many VNIs on a single device today. Even if they could or did, supporting this many VNIs on a single device is not considered acceptable because of the number of customers who'd be affected by a failure of the device. It is the combination of the router's IPv4 loopback address plus the VNI that makes the RD unique across the network. Thus, the value of the VNI-specific part of the RD is a device-local encoding of the VNI, not necessarily the absolute value of the VNI.

Because the router's loopback IP address is part of the RD, two nodes with the same virtual network will end up having different RDs. This is the expected behavior. See the section "RD, RT, and BGP Processing" on page 28 later in the chapter for a discussion of why this is the desired behavior.

Route Target

BGP advertisements carry *path attributes*, which you can think of as optional Post-it notes that add extra information about a network address. These Post-its further qualify the processing of a BGP UPDATE message. The Post-it notes are not in text and do not say things like "fragile" or "do not bend." Instead, they are carried as encoded bits. They carry information such as the next-hop IP address for a prefix, whether to propagate an advertisement, and so on. Path attributes take several forms, including well-known *attributes*, *communities*, and *extended communities*. This is not the venue to further describe these terms. If you're interested, you can find these details in several online and offline references, including standards documents.

In this section, we look at a specific path attribute called the *Route Target* (RT). An RT encodes the virtual network it represents. A BGP speaker advertising virtual networks and their addresses uses a specific RT called the *export RT*. A BGP speaker receiving and using the advertisement uses this RT to decide which local virtual network into which to add the routes. This is called the *import RT*. In a typical VPN configuration, the network administrator must configure both import and export RTs.

The definition and use of RT is in the standard RFC 4364, section 4.3.1. For more details, refer to that document.

The encoding of RT for EVPN over VXLAN is described in RFC 8365. Figure 3-6 presents this encoding.

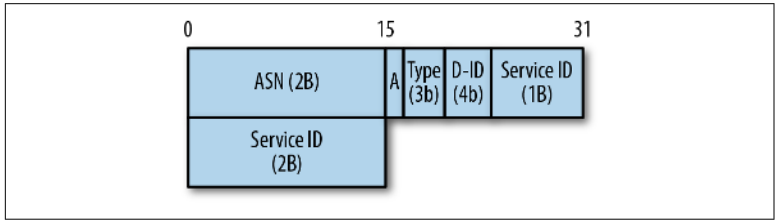

Figure 3-6. Structure of RT for EVPN with VXLAN

The different fields are as follows:

ASN
: The two-byte Autonomous System Number (ASN) of the BGP speaker advertising the address.

A
: A bit indicating whether the RT is autoderived or manually configured.

Type
: A three-bit field indicating the encapsulation used in EVPN. For VXLAN, it is 1, and for VLAN, it is 0.

Domain ID (D-ID)
: Four bits that are typically zero. In certain cases, if there is an overlap in the VXLAN numbering space, this field is used to qualify the administrative domain to which the VNI belongs.

Service ID
: Three bytes containing the virtual network identifier. For VXLAN, it is the three-byte VNI, for VLAN it is 12 bits (the lower 12 bits of the 3-byte field).

RD, RT, and BGP Processing

RD and RT both identify the virtual network from which a packet comes. To understand why you need both, consider the constraints BGP has to deal with. Let's see if an analogy helps.

Imagine Santa Claus is like BGP. Come Christmas, a lot of kids will be getting the exact same present, the viral toy for that year. Worse still for poor Santa, some kids will receive multiple copies of the same toy, thanks to their large, extended family. Santa has multiple responsibilities. First, he has to keep copies of this identical toy separate. This is the purpose of RD. Santa stamps each copy of the toy with its own RD. Santa's second responsibility is to not play favorite relative. He must not decide to (wittingly or not) give a child exactly a single copy of a toy or choose which copy from which relative a child receives. Why is this a risk? Because Santa is like BGP, he runs the best-path algorithm on each toy and picks only one. However, it is up to each kid to decide which copy of a toy they choose to keep. How does an excited youngster know which copy is from which relative? That is the job of the RT. In short, RD is Santa's way of keeping the toys separate, and RT is how the child knows who the toy is from.

Switching back to BGP, let's see how RD and RT affect BGP's advertisement processing. Every BGP implementation I know of maintains two kinds of routing tables: a global one and one per virtual

network. BGP runs the best-path algorithm on the global table to pick a single path to advertise for each prefix to its peers. Because the RD is unique to each originator, all copies of a route will be advertised to a neighbor. To install routes into a virtual network's routing table, BGP first uses the import RT clause to select specific candidate routes from the global table to import into this virtual network. Then, it runs the best-path algorithm again on the imported candidate routes, but this time within the context of the virtual network's routing table. If the same address is advertised with multiple RTs, the best-path algorithm selects the best one.

Route Types

In BGP, UPDATE messages carry reachability information. This reachability information is encoded in a specific structure called Network Layer Reachability Information (NLRI). For most AFI/SAFI combinations, the structure and content of the reachability information carried in an UPDATE message is the same. For example, an IPv4 Unicast UPDATE message carries the same kind of information: reachability about an IPv4 prefix.

This is not the case with EVPN. There are disparate pieces of information to be exchanged. For example, the update can be reachability to a specific MAC address, or it could be reachability to an entire virtual network. Also, unlike IPv4 and IPv6, because EVPN has already consumed both an AFI and a SAFI, there is no way to separate information about unicast and multicast addresses. To accommodate these additional subdivisions, EVPN NLRI is further classified by a Route Type. Table 3-1 shows the different Route Types used in EVPN. The minimum required Route Types needed to operate an EVPN network are RT-2, RT-3, and RT-5. The rest are optional and dependent on the choices you make in building your network. We cover these in detail in subsequent chapters.

Table 3-1. EVPN Route Types

Route Type	What it carries	Primary use
Type 1	Ethernet Segment Auto Discovery	Used in the data center in support of multihomed endpoints
Type 2	MAC, VNI, IP	Advertises reachability to a specific MAC address, and optionally its IP address
Type 3	VNI/VTEP Association	Advertises reachability in a virtual network

Route Type	What it carries	Primary use
Type 4	Multicast Information	Used in the data center in support of multihomed endpoints, to ensure that only one of the VTEPs forwards multicast packets
Type 5	IP Prefix, L3 VNI	Advertises prefix (not /32 or /128), routes such as summarized routes in a virtual L3 network
Type 6	Multicast group membership info	Information about interested multicast groups derived from IGMP

Modifications to Support EVPN over eBGP

Using eBGP to exchange EVPN information requires some additional configuration. Although FRR itself does not require this additional configuration, other routing suites do. It is important for a network administrator to understand these knobs to operate a multivendor network.

Some nodes might require even more additional configuration. For example, Cisco configuration guides recommend using *disable-peer-as-check* option, as well. I don't describe such additional implementation-specific changes in this book.

Keeping the NEXT HOP Unmodified

By default, when advertising a prefix in eBGP, the sender changes the next hop for that prefix to its own address. For example, assume three eBGP peers in a line like this: A - B - C. When B receives a prefix advertisement from A, the advertisement includes A as the next hop for that prefix. When B sends that advertisement to C, it changes the next hop to B. In other words, C receives said prefix with B as the next hop, not A.

In the traditional model of EVPN, using iBGP, the peering is between A and C. Thus, the L2 reachability information received by C says A is the next hop to reach that L2 endpoint. Even when B functions as an RR between A and C, B does not change the next hop.

With EVPN, the L2 address is of interest only to A and C. As discussed in Chapter 2, B is part of the underlay and does not know or care about this address. So, when eBGP is used to also transmit EVPN information, B must not change the next hop for the L2 prefix advertisement it receives from A to itself before sending to C.

Most, if not all, BGP implementations support the ability to conditionally instruct BGP to not change the next hop for an eBGP session. For example, Cisco routers use route-maps as follows:

```
route-map foo permit 10
    set ip next-hop unchanged
```

In a Clos topology, configure this on at least all non-FRR spines. If you're using a host as VTEP and have EVPN configured on a host, configure this on such non-FRR leaves, as well.

Retaining Route Targets

eBGP semantics consider it acceptable to drop all prefixes with unused route targets. This optimization, like the default in the previous section, is also born out of the assumption that the VPN information exchange uses iBGP. If you're using iBGP, the reasoning goes, why should you bother keeping any unused information? With eBGP, the spines have no knowledge of the VNIs about which the VTEPs are communicating. But they cannot drop the prefix just because they're not using it; in other words, installing it in their forwarding tables. They need to distribute it to the other leaves (other than the one they received the original advertisement from).

Again, the configuration on a Cisco router is via the command `retain-route-target-all`. This is also added under the section of `address-family l2vpn evpn`.

FRR Support for EVPN

FRR supports the traditional SP model of configuring EVPN, as well as the simpler data center model. It uses profiles to set up defaults which make sense for each model. As such, the model assumes sane defaults for working in a data center so that the user configuration is minimal.

Automatic Propagation of NEXT HOP

For any EVPN advertisement carrying L2 information, FRR does not change the next hop when the BGP communication is over eBGP. It is also smart enough to function as a regular eBGP speaker for non-EVPN advertisements. In other words, it sets the next hop to itself for IPv4/IPv6 unicast AFI/SAFI advertisements, but does not do so for EVPN routes (specifically type 2 and 3 Route Types).

RT/RD Derivation

The EVPN standard specifies that the RT can be autoderived, if desired. Not all implementations support this model yet, but FRR does. It encodes RT as specified by RFC 8365. It assumes that the encapsulation is VXLAN. Most implementations require you to state this objective via a configuration such as `route-target import auto`. But FRR derives this automatically without requiring additional configuration.

FRR maintains a bitmap in which each bit represents a separate VNI. The VNI-specific two bytes in the RD come from the value of this bit position. FRR also allows an administrator to manually configure the RT for a specific virtual network, but this is not recommended because of the potential to make mistakes.

What Is Not Supported in FRR

FRR version 4.0.1, the latest release at the time of this writing, supports only Route Types 2, 3, and 5.

Summary

This chapter laid the groundwork for using BGP to advertise reachability to L2 addresses and virtual networks. Unlike its use in the SP world, EVPN in the data center can use eBGP for both the underlay and the overlay. By assuming sane defaults, FRR simplifies the configuration and eliminates needless clutter. We're now ready to understand the core of EVPN: how bridging and routing are implemented in EVPN.

CHAPTER 4

Bridging with Ethernet VPN

The first few chapters of this book provided the background and framework for Ethernet VPN (EVPN).

EVPN's primary function is to support virtual Layer 2 (L2) network overlays. Bridging is how packet forwarding works in L2 networks. Therefore, in this chapter our focus is on how bridging works in EVPN networks. This study includes understanding control-plane and data-plane semantics associated with bridging. The details include a study of choices when it comes to handling frames destined to multiple Network Virtualization Edges (NVEs), handling Media Access Control (MAC) movement, and L2 loops. We study how EVPN works with multiattached endpoints such as dual-attached servers, a common deployment model with EVPN. We conclude with a study of Address Resolution Protocol (ARP) suppression and the benefits it provides.

An Overview of Traditional Bridging

To understand how bridging works with EVPN, we begin with a quick study of how bridging works in a traditional 802.1Q bridge. IEEE 802.1Q defined the standard that describes how VLAN-aware bridges work, and so traditional bridging is also called *802.1Q bridging* or plain *.1Q bridging*. We also introduce the terminology commonly used in bridging. We then use this as the basis to describe how EVPN solves the same problem, albeit differently. A compare-and-contrast table at the end helps solidify these concepts further.

For the sake of this discussion, let us assume the network shown in Figure 4-1. The figure shows a simple Clos network with two spine switches labeled S1 and S2 and four leaf switches labeled L1 through L4. Hosts (or endpoints) A through F are attached to the leaves. There are two virtual networks, red and blue. In an 802.1Q bridge, these virtual networks will be VLANs. To simplify the mappings a reader has to maintain, we'll refer to the MAC address of A as MAC_A. Similarly, we'll assume that L1 connects to host A on $Port_A$.

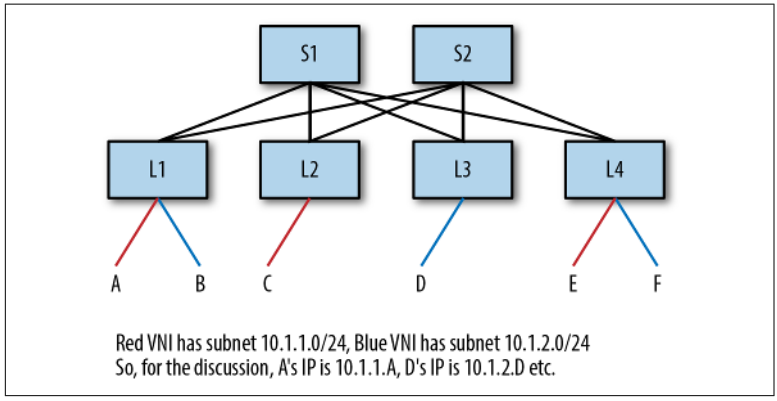

Figure 4-1. Sample topology

An 802.1Q bridge forwards a packet based on the VLAN and the destination MAC address of the packet. If a packet is received without a VLAN tag, the bridge port has a default VLAN tag associated with it. 802.1Q bridges use what is called "*flood-and-learn*" to populate the MAC forwarding table. What this means is that whenever a bridge receives a packet whose combination of VLAN and destination MAC address is not in the MAC forwarding table, the bridge sends the packet out every port that is in the same VLAN as the incoming packet. This is called *flooding*. However, the bridge filters out the ingress port from this list of ports the packet is flooded to. This filtering is called *self-forwarding check*. The other thing the bridge does is record the (VLAN, source MAC address) in the MAC forwarding table as being reachable on the ingress port. This is called *learning*. So, in Figure 4-1, when node L1 receives a packet from A destined to E, and it does not have an entry for (red VLAN, MAC_E) in its MAC forwarding table, it floods the packet out all ports which belong to the red VLAN. It filters out $Port_A$ from this list of ports. It also records that (red VLAN, MAC_A) is reachable via $Port_A$ in its MAC forwarding table.

This is done by every node in the packet's path as it inexorably moves from the source to the destination. Thus, spines are not relieved from any duties when it comes to packet forwarding and knowledge of virtual networks. Hence, VLAN is not an overlay network virtualization technology.

How does a bridge know which ports belong to a specific VLAN? By configuration. In Figure 4-1, all the links between the leaves and spine switches carry both the red and blue VLAN traffic. Such links are called *trunked links*.

A careful reader trying to map this logic with a packet trace in the topology of Figure 4-1 will realize that it is possible for a packet to loop in the network because of loops in the topology (consider the path L1–S1–L2–S2–L1, for example). To avoid this, a control protocol called the *Spanning Tree Protocol* (STP) is used to convert any topology into a loop-free topology by removing all redundant paths to a node. In other words, STP eliminates multipathing in a network.

Overview of Bridging with EVPN

Let us now reexamine bridging, but with EVPN. We will use the same communication example as in the previous section: host A wishes to correspond with host E. We will use Figure 4-1.

A few things to note to kick off the discussion. Packet forwarding between the leaves and spines happens via routing. For the sake of this discussion, assume that the control protocol used to set up the IP forwarding tables is BGP. This means all paths between the routers are used (i.e., multipathing works). Next, there are no VLANs on the links between the leaves and spine switches. We're using VXLAN as the network virtualization overlay protocol. Network virtualization begins at the leaves. Therefore, the leaves are the network virtualization edges (NVEs) (i.e., they encapsulate and decapsulate VXLAN packets). As NVEs, they need to have tunnel addresses—IP addresses that are used in the VXLAN tunnel header. We'll designate these addresses using the symbol VTEP: that is, L1's tunnel IP address is $VTEP_{L1}$, L4's is $VTEP_{L4}$, and so on. The VTEP IP address is typically the address assigned to the loopback device of the node. We will also enable 802.1Q traditional learning, but only on the edge ports. In Figure 4-1, the edge ports are all the end host facing ports. Now, let us begin.

To enable EVPN, we must first enable the Address Family Indicator (AFI)/Subsequent Address Family Indicator (SAFI) of EVPN, which is l2vpn/evpn in all the BGP peerings between the leaves and switches. We also enable the advertisement of information associated with each NVE's locally attached virtual networks (see Chapter 6 for specific configuration commands).

Each NVE first advertises the list of all its locally attached virtual networks. In the case of L1, these are the red and blue virtual networks, for L2 it is only the red network, and so on. At the end of this advertisement exchange, all NVEs know what virtual networks are of interest to all the other NVEs in the network. The MAC forwarding tables are empty, so there are no MAC addresses to advertise at this point. Figure 4-2 illustrates this exchange from the point of view of L1. In fact, all EVPN packet exchanges follow this same model. A leaf sends information to its BGP peers, which are the spines, and the spines send information to their peers, the leaves.

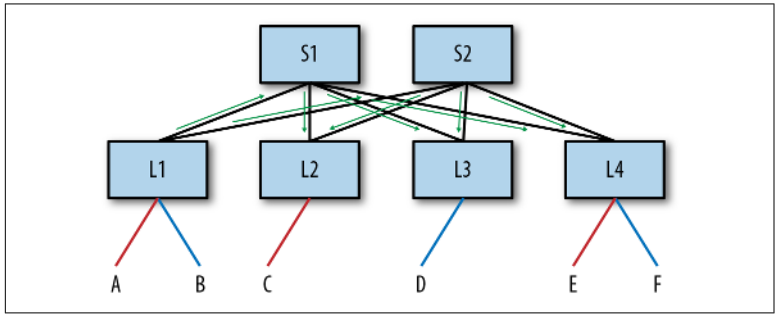

Figure 4-2. Illustrating a sample exchange of EVPN packet flow from the perspective of L1

Now A sends its packet to L1 destined to E. Figure 4-3 illustrates this packet flow.

1. The packet sent from A to L1 is no different in this case from traditional bridging. L1 receives this packet and, just as in traditional bridging, learns that MAC_A is reachable via $Port_A$. Now, because L1 does not have any information about MAC_E, it decides to flood the packet, just as in traditional bridging. Remember, S1 and S2 know nothing about virtual networks because this is a network virtualization overlay solution. So, for L1 to flood the packet, it must encapsulate the packet in a VXLAN tunnel header and send the packet to the NVEs that

have expressed an interest in the red virtual network. There are two main options for how this can be done. We discuss those in "Handling BUM Packets" on page 42.

2. For now, let's just assume that L1 sends VXLAN-encapsulated packets to L2 and L4 (L3 does not have a red virtual network). L1 computes the hash on the packet header fields of the original packet and sets this to be the UDP source port in the VXLAN tunnel header. The packets are then routed to L2 and L4. In Figure 4-3, the packet to L2 is routed via S2, and the packet to L4 is routed via S1.

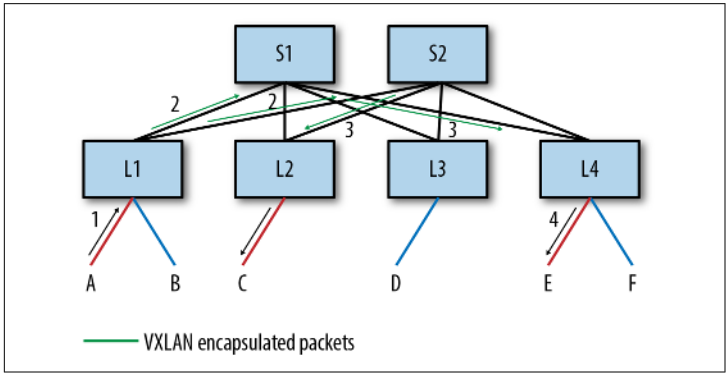

Figure 4-3. Flooding in an EVPN network, A → E

3. When these VXLAN-encapsulated packets reach L2 and L4, L2 and L4 each know that they are the end of the VXLAN tunnel because the destination IP address in the packet is their IP address and the User Datagram Protocol (UDP) destination port says that this is a VXLAN packet.
4. They decapsulate the packet and send the packet out of all the locally attached ports that are in the red virtual network. Neither L2 nor L4 attempt to send this packet in VXLAN-encapsulated form to any other node. Not flooding a VXLAN-encapsulated packet after decapsulation back into the VXLAN overlay is the equivalent of the self-forwarding check. In EVPN, it is referred to by its routing protocol name, *split-horizon check*.

Neither L2 nor L4 learn anything about MAC_A from this flooded packet. However, L1 has a new local entry in its MAC forwarding table. So L1 advertises the reachability to MAC_A in the red virtual network via a BGP EVPN message. Specifically, it uses a Route Type

2 (RT-2) message, which carries the {VNI, MAC} advertisement. The message says that MAC_A in the red virtual network is reachable via the VXLAN Tunnel End Point (VTEP) L1.[1] L1 delivers this information to its BGP peers, S1 and S2. S1 and S2 in turn deliver this message to their peers, L2, L3, and L4. L2 and L4 populate their MAC forwarding tables with information about MAC_A. They note that MAC_A is remote and reachable via the VTEP L1. L3 which has no red VNI simply stores this message (or can discard it). We discuss why L3 even gets this message in "Why Does NVE L3 Get an Advertisement for MAC_A?" on page 41. This BGP exchange is as illustrated earlier in Figure 4-2.

Figure 4-4 shows the packet flow of E's reply to A.

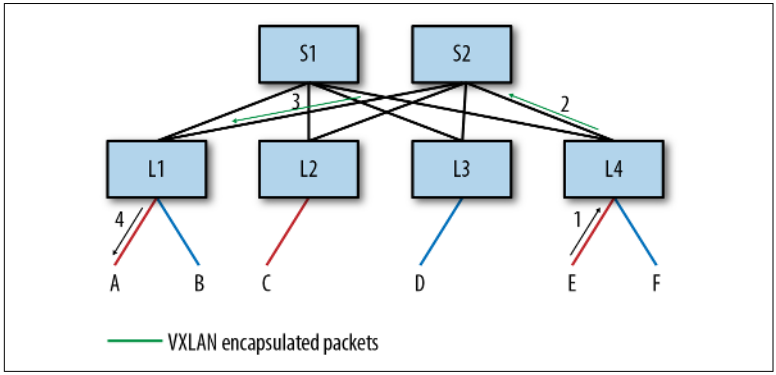

Figure 4-4. Packet flow from E to A with EVPN

1. E sets the destination MAC address of the packet to be MAC_A, and the source MAC to be MAC_E and sends the packet to L4. L4 receives this message on $Port_E$. L4 learns this MAC address to port association because it is received on a locally attached port. L4 knows that MAC_A is a remote MAC reachable via $VTEP_{L1}$ from the earlier EVPN message. It therefore VXLAN encapsulates the packet with the destination IP address of $VTEP_{L1}$, and the source IP address of $VTEP_{L4}$. It computes the hash on the packet header fields of the original packet and sets this to be the UDP source port in the VXLAN tunnel header. The packet is next routed to $VTEP_{L1}$ via the usual IP routing rules.

1 Note that we use VTEP and NVE interchangably.

2. Because L4 had learned via BGP that $VTEP_{L1}$ is reachable via both S1 and S2, it uses the hash of the packet (computed from the VXLAN header or the original, a matter of implementation) to pick either S1 or S2 and sends the packet to it. Let's assume it picks S2.[2]
3. S2, upon receiving this packet, routes it out $Port_{L1}$. When L1 receives this packet, it sees that the destination IP address is itself and strips off the VXLAN header. It looks for the destination MAC address on the exposed inner packet, which is MAC_A in the red virtual network.
4. The output of this lookup in the MAC forwarding table is $Port_A$, and so L1 forwards the packet out $Port_A$.
5. Finally, L4 advertises the reachability to MAC_E in the red virtual network to its BGP peers, S1 and S2. S1 and S2 in turn send the advertisement to their other BGP peers, L2, L3, and L4. L2 and L4 install the MAC entry for MAC_E as associated with $VTEP_{L4}$ in their MAC forwarding table (see "Why Does NVE L3 Get an Advertisement for MAC_A?" on page 41).

Figure 4-5 illustrates how the packet headers are added as the packet flows from E to A. Only relevant fields from the packet headers are shown; there are obviously a lot more fields in the IP and VXLAN headers.

2 You might notice that L4 routes the packet to A to a different spine from the spine on which it received the packet to E. This is just a coincidence resulting from the hashing decisions made locally at L4 (and at L1 for the packet to E).

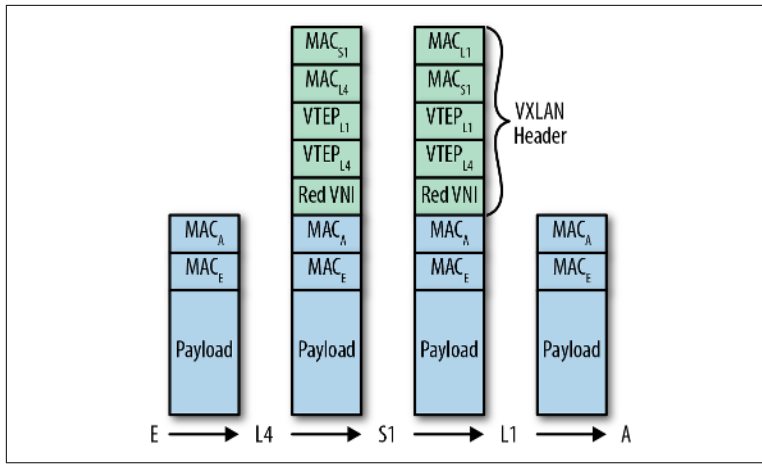

Figure 4-5. Packet headers in packet flow from E to A

Table 4-1 looks at how bridging works in both an 802.1Q bridge and an EVPN network.

Table 4-1. Comparing bridging in 802.1Q bridge and in EVPN network

Behavior	802.1Q Bridge	EVPN
Control protocol	STP	BGP EVPN
Knowing which edges are interested in a virtual network	No notion of remote edge; every hop along the way is aware of virtual network	Remote NVE interest in virtual network learned from BGP EVPN message
Behavior when destination is unknown	Flood along spanning tree, only on links carrying that virtual network	Flood to every VTEP interested in that virtual network[a]
Packet header carrying virtual network of the packet	802.1q VLAN tag	VXLAN header
MAC to port association of local end host	.1Q bridge learning	.1Q bridge learning
MAC to port association of remote end host	.1Q bridge learning	BGP EVPN messages
All links in network utilized	No	Yes
Path from A → E versus the path from E → A	Same path used in forward and reverse paths	Different paths may be used in the forward and reverse paths

[a] This is not entirely accurate, because there are choices. We study this in "Handling BUM Packets" on page 42

What Ifs

In this section, we look at the behavior when a slightly different sequence of events happens, sequences that you might already be pondering.

- *What happens if the BGP advertisement about MAC_A didn't get to L4 before the reply from E reaches L4?* In this situation, L4 behaves just as L1 did: it sends the packet to all the NVEs interested in the red virtual network.
- *What happens to the packet from E if, upon reaching L1, it does not know where A is?* Again, it behaves similarly to L4 in this case: it sends the packet out all locally attached ports in the red virtual network. In neither case (when the egress NVE is L1 or L4) does the packet ever get sent out a port with a VXLAN encapsulation due to the split-horizon check.

Why Does NVE L3 Get an Advertisement for MAC_A?

You might remember that the MAC advertisement about MAC_A and MAC_E was sent to L3, even though L3 had expressed no interest in the red virtual network. Why is that, and what does L3 do with this message?

This is an implementation detail, and it might be handled differently by various BGP implementations. Remember that the spine nodes, S1 and S2, are part of the underlay, and do not know anything about virtual networks. To ensure that L3 does not get the information, they would need to keep track of which BGP peers are interested in which virtual network. This is an unnecessary computation for them to perform. The implementation also scales much better if it does not need to keep track of virtual networks.

This implementation might have an additional benefit. If the red virtual network is attached to L3 at some point in the future, L3 is now interested in the red virtual network. If it already has the MAC advertisements for the red virtual network, packet flow between L2 and the other nodes can be much faster and result in less flooding. If L3 does not have the MAC advertisements, it must ask S1 and S2 to send it all the BGP EVPN updates again. It does this via a BGP

Route Refresh message.[3] This is somewhat inefficient. On the other hand, maintaining a copy of all MAC advertisements for networks in which it has no interest can cause bloated memory usage for a BGP process. In such a situation, it might be better to just drop the advertisement it receives rather than store it. At the time of this writing, FRR retains the MAC advertisements.

Handling BUM Packets

We were skimpy with the details on how flooding works in EVPN networks. You might also have noticed that two copies of the flood packet were sent, one to S1 and another to S2. How was this decided? We cover such details in this section.

The common terminology to refer to flooded packets is Broadcast, unknown Unicast, and unknown Multicast, or BUM, packets. EVPN provides two choices for packet forwarding of BUM packets: ingress replication and Layer 3 (L3) underlay multicast. In addition to this, some implementations might also provide the ability to drop all BUM packets. We discuss each in some more detail, including the trade-offs of each approach. This information helps administrators pick the right solution based on their enterprise needs instead of one dictated by the network vendor.

Ingress replication

In ingress replication, the ingress NVE sends multiple copies of a packet, one for each egress NVE interested in the virtual network. In our example, when A's packet to E is flooded, L1 makes two copies—one for L2 and one for L4—and sends them off one by one.

The primary benefit of this model is that it keeps the underlay simple. The underlay needs to provide only IP unicast routing to support network virtualization. Furthermore, it is the simplest model to configure: there is no additional configuration required. The replication list is automatically built from the BGP EVPN RT-3 (carrying the Virtual Network Identifiers [VNIs] of interest to a VTEP) messages without any further intervention from the user. This makes the solution also more robust, because the chances of human error are reduced significantly.

3 Not all implementations support Route Refresh, even though most popular implementations do, including FRR.

The primary disadvantage of this approach is that the replication bandwidth required from the underlay can be high, especially if there are lots of BUM packets. If the number of NVEs to replicate to is no more than 64 to 128 nodes[4] and the amount of BUM traffic is low, this approach works quite well. These two conditions are common enough that ingress replication is probably the most commonly deployed model for handling BUM packets in the data center. Even if the number of NVEs to replicate to is higher but the amount of traffic is low, this method works quite well. ARP suppression, which we discuss later in this chapter, helps reduce the most common source of BUM traffic.

L3 multicast support in the underlay network

The second approach to handling BUM packets is to use L3 multicast in the underlay network. By using multicast, the ingress NVE does not have to send a separate copy for each egress NVE. For the example described earlier, L1 would send a single copy to S1 and S1 would send multiple copies, one to each of the leaves. With just two egress NVEs in our example, the difference between the ingress replication and multicast approach is not apparent. If there were 20 egress NVEs in our example instead, L1 would send 20 copies to S1 (or divide that among S1 and S2, as described in the previous section). With L3 multicast, L1 would send a single copy or at most two copies (one each to S1 and S2), and they would replicate the packet to all the other relevant nodes.

The main advantage of this approach is that it is possible to handle a large volume of BUM packets or even well-known multicast packets efficiently. However, it is something of an administrative nightmare.

In this model, besides providing unicast routing support, the underlay must also provide multicast routing support. The most commonly used multicast routing protocol is called Protocol Independent Multicast (PIM). PIM is actually a family of protocols, providing for different ways to build a replication tree depending on the network topology and requirements. The most commonly deployed member of the PIM family is called PIM Sparse Mode

4 I picked 64 as a heuristic, there's not a lot of solid data behind this number. It seems to me when you're sending out 64 replicas for each BUM packet, and there are a lot of BUM packets, the amount of work the switching silicon has to do starts to add up.

(PIM-SM). However, the EVPN draft recommends using PIM Source-Specific Multicast (PIM-SSM) because it is more appropriate for this use case. Nonetheless, PIM-SSM is not commonly deployed in enterprise networks. On the other hand, PIM-SM requires additional protocols such as Session Discovery Protocol (SDP) for a reliable deployment. We do not have the space to get into explaining L3 multicast here. But deploying L3 multicast is sufficiently painful that everybody avoids it if possible.

Besides the complexity of enabling PIM and dealing with L3 multicast, this model has many other limitations. To ensure that for every virtual network only the associated NVEs get the packet, each virtual network must be in its own multicast group. But this would result in far too many multicast groups to scale well. So, the administrator must now manually map all of the virtual networks into a smaller number of multicast groups. This in turn leads to some NVEs receiving BUM packets for virtual networks in which they have no interest. Mapping a virtual network to a multicast group also adds significant configuration complexity. An administrator must configure the mapping of virtual network to multicast group on every single NVE. There is no simple way to ensure that this configuration is consistent and correct across all the NVEs.

Dropping BUM packets

BUM packets are considered by many network administrators to be a cheap way to launch a Distributed Denial-of-Service (DDoS) attack on the network. By sending packets to addresses that might never be seen by the network, less network bandwidth is available for legitimate traffic. Such packets can deluge end hosts in that virtual network and cause them to fail because the system is very busy coping with BUM packets.

Dropping BUM packets implies that after we hear from an endpoint, its MAC address is communicated to all the other nodes via BGP. Therefore, there is really no need to flood these packets. ARP/ND suppression, discussed later, also helps here.

If supported by an implementation, this option can be enabled. Some implementations allow for this to be enabled per virtual network or for all virtual networks.

The main disadvantage of this approach is possible communication breakdown with traffic meant for silent servers. For example, if E is

a print server and is sitting idle, its MAC address is not known to L4 and so cannot be advertised. If A wants to send a print job to that printer, it cannot because L1 does not know where E is. But silent servers are increasingly rare, so this might not be a problem in networks anymore.

Signaling which approach is employed

Because there are choices, each NVE must inform the other NVEs of what it can support. RT-3 EVPN messages can carry a BGP attribute called Provider Multicast Service Interface (PMSI), which identifies the kind of BUM packet handling supported by this device. The PMSI attribute is defined in a completely different standard (RFC 6514) from the usual EVPN standards. The values suggested by the EVPN draft for signaling the replication model are:

- 3 for PIM-SSM
- 4 for PIM-SM
- 5 for PIM-BiDir[5]
- 6 for Ingress Replication

Unfortunately, dropping BUM packets is a local configuration knob, not advertised to other neighbors.

Handling MAC Moves

Now consider what happens if the endpoint named A moves from behind L1 to behind L2. This usually happens if A is a virtual machine (VM). Most endpoints send out an ARP message when they move. This ARP message is sent to the broadcast address and is an ARP reply packet. It is not a reply sent in response to any request. Hence, this message is called a Gratuitous ARP (GARP). If A sends a GARP in conjunction with its move to behind L2, L2 learns that MAC_A is local to it now. It must update its MAC forwarding table entry to have MAC_A point out its local port rather than it be associated with $VTEP_{L1}$. Next, L2 must advertise its newly learned information to all the other VTEPs. It does so by sending a MAC advertisement via a BGP UPDATE (RT-2) message. When this update gets to L1, L1 faces a quandary. Its MAC forwarding table

5 PIM-BiDir or Bidirectional PIM is another variant of the PIM protocol.

says MAC_A is a locally attached MAC. In regular BGP updates, BGP always prefers the local information to a host that is remote. How do we fix this?

EVPN defines a new BGP extended community called MAC Mobility to help with this. Figure 4-6 shows the format of this extended community.

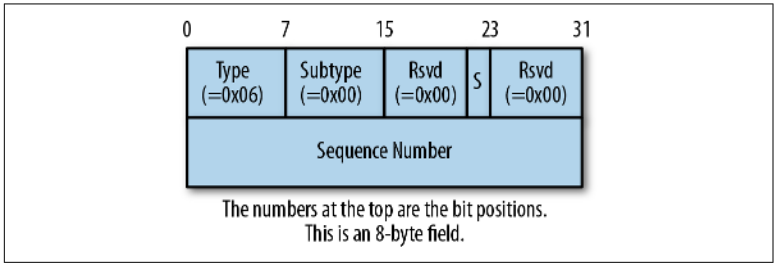

Figure 4-6. MAC Mobility extended community format

When L1 first advertised MAC_A in the red virtual network, MAC_A had not been advertised before. In such a situation, the advertisement is sent without tagging it with the MAC Mobility extended community. When L2 now wants to advertise about MAC_A, it tags the advertisement with the MAC Mobility extended community. The sequence number is set to 0. The presence of the MAC Mobility tag in the advertisement lets BGP prefer the remote advertisement over its local information. So L1 will update its MAC forwarding table entry as being behind $VTEP_{L2}$. L1 will also withdraw its MAC advertisement for MAC_A.

If at a later time, A moves back behind L1 (or moves somewhere else, say behind L4), L1 to which it is now local will send out a new advertisement, but with an incremented sequence number. If BGP receives an update with a MAC Mobility tag with a sequence number greater than its local copy, it prefers the new update. If multiple updates are received for a MAC update with the same sequence number but with different VTEPs, the advertisement from the VTEP with the lowest VTEP IP address is selected.

If a MAC address is static[6]—that is, it is never supposed to move—the very first time the MAC address is advertised, it must be tagged

6 Different implementations have different ways to signal a static MAC. In Linux, the netlink message has flags to do this.

with the MAC Mobility extended community with the "S" bit set to 1. When any VTEP receives a MAC advertisement with such a tag, it must ignore any locally detected changes in that MAC address in the associated virtual network.

Sometimes, messages indicating that a MAC address moved are incorrect. One reason is lax security in L2 networks, making it possible to spoof a MAC address and make it move from a safe host to a compromised host. Another reason for spurious moves is that a problem in a connected 802.1Q network can cause the STP to continuously update its tree. When the tree is updated, a MAC address might appear in a different location in the tree, making it look like the MAC address moved.

To handle all of these cases, if a VTEP detects that a MAC address is updating too frequently, it can stop sending or handling further updates to that MAC. It must also notify the administrator that this is happening. There is no clearly defined way to get out of this situation.

FRR's EVPN implementation supports MAC Mobility as defined by the standard, with the caveat that it does not support any way to handle excessive MAC updates specially.

Support for Dual-Attached Hosts

In enterprise networks, compute nodes are frequently attached to more than one switch. This is primarily done to ensure that a single link failure or a single switch failure doesn't leave a compute node disconnected from the network. This also allows upgrading of switches without taking down all the compute nodes associated with that switch during the upgrade window. As an enterprise network solution, EVPN is also frequently deployed with dual-attached hosts. We don't discuss a variation of the same idea: multisite attachments where geographically disparate data centers are hooked up via some form of a Wide-Area Network (WAN) connection. The problems associated with such sites are more intricate.[7]

Figure 4-7 shows the network we've used all along, but with the hosts dual-attached to L1 and L2. I've come across deployments in

7 And one we will not have the room to discuss in this book.

which some nodes are dual-attached and some are single attached. This topology is not uncommon when the deployment is hosting multiple workloads, such as a modern Hadoop cluster and a more traditional application or even OpenStack.

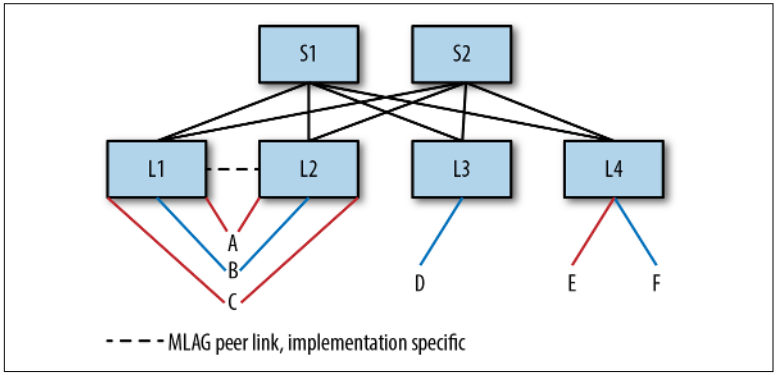

Figure 4-7. Dual-attached hosts in EVPN network

The following subsections address the problems of dual-connected hosts and how to address them. Following are the problems to be addressed:

- How are the dual-attached hosts connected to the switches?
- How is the dual-attached node seen by the other VTEPs? In other words, how do VTEPs L3 and L4 in Figure 4-7 view A, B, and C?
- What happens when some of the hosts lose their connection to one of the switches? For example, what happens if C loses its link to L1?
- How does packet delivery work for a multidestination frame, such as a BUM packet? Do A, B, and C get duplicates because L1 and L2 each deliver a copy? Duplicates can confuse certain applications if they're not running over a reliable stream protocol such as TCP.

The Host-Switch Interconnect

In the most common deployment, the host treats the two links as being in a bond (aka link aggregation). The two main advantages of a bond are that both links are usable at the same time, and that as a

bond, the link needs a single IP address instead of two. Using both links at the same time is also referred to as *active–active mode*. Creating a bond involves using a standard protocol called Link Aggregation Control Protocol (LACP). Using LACP ensures that the host is properly connected to the right devices. For example, if C accidentally hooks up a cable to L3 instead of L2, this is caught by LACP because C will discover that it is not communicating with the same entity on both the links. LACP supports bonding only if the links are connected to the same pair of devices on all links. LACP also catches problems such as unidirectional link failures. Some host vendors charge extra for enabling LACP, so many switch implementations support the notion of a *static bond*, in which the administrator configures the two ends of the link as being in a bond without using LACP.

Some customers use the dual-switch interconnect only to handle failure. In this case, one of the links is in standby mode. It comes alive only when the active link dies. This is called *NIC teaming* or *active-standby mode*. This mode makes some assumptions similar to the bond mode. It assumes that when the standby link comes alive, it has the same IP address as the recently deceased link. It also assumes that the default gateway is the same on this link as on the other link.

A third alternative is to treat each link as an independent link without LACP or bonding. In such a situation, the host can be made to use a single IP address, but this is a little more involved and is deployed in situations in which the host is either a router or a VTEP itself.

We'll discuss the case in which the links are bonded (with or without LACP) for the rest of this discussion, because that is the most common deployment.

VXLAN Model for Dual-Attached Hosts

Most packet switching silicon implementations as of this writing assume that a MAC address is behind a single VTEP. In Figure 4-7, there are two different switches associated with each of the dual-attached hosts. So how do the remote VTEPs (L3 and L4 in the figure) handle this case?

There are two possibilities: each VTEP has its own IP address or both VTEPs share a common IP address. The latter is the most com-

mon deployment: having the two VTEPs share the same IP address. The main reason for this is that the common implementation of a MAC forwarding table supports only a single port of exit. In traditional bridging, there was never a reason for a MAC address to have multiple ports of exit. The STP explicitly eliminates multiple paths to eliminate loops. Bonds are represented as a single logical port to avoid this problem. Additional logic after the logical port selection allowed the implementations to pick a single outgoing physical port to transmit the packet. At the time of this writing, the Linux kernel and most switching silicon that I'm aware of support the single common IP model. So even if you have a vendor that supports the alternate model, using the common IP address model ensures interoperability with other vendors.

So, in the example topology of Figure 4-7, L1 and L2 will both transmit packets for all dual-attached hosts with a source VTEP IP address that is common to them both. Most implementations verify that the switches have the same common IP address configured via a protocol such as Multichassis Link Aggregation (MLAG) Protocol,[8] described shortly. The network administrator must also ensure that this common IP address is advertised via BGP; otherwise, the other VTEPs will not know how to reach this VTEP IP address.

Switch Peering Solutions

Before we proceed to look at the other problems and how they are handled, we must first stop and examine the model adopted by the pair of switches to which the dual-attached hosts are connected. There are essentially two answers: MLAG and EVPN.

MLAG

The standard LACP does not support creating a bond when the links are split at one end across multiple devices. In other words, the standard does not support a model where dual-attached hosts (A, B, C in Figure 4-7) connect to talk to two different switches (L1 and L2). Therefore, every networking vendor has its own proprietary solution to provide that illusion. The generic name for this solution is MLAG, but each vendor has a brand name for its solution, and

8 I'm not aware of any implementation that uses the information from the EVPN messages to verify this.

the specific implementation and deployment details vary across implementations. For example, Cumulus and Arista require a separate peer link (shown as a dotted line between L1 and L2) to be present between the MLAG pair of switches. Cisco's NXOS does not require the peer link to be present.

A entire book can be written about MLAG. Here, we skip all the details and focus on how it solves the problems at hand in the next few sections.

EVPN

EVPN supports dual-attached devices natively. It calls them *multi-homed nodes*. The main details of this solution are described in section 8 of RFC 7432 and section 8 of RFC 8365. Primarily, EVPN uses Route Type 1 (RT-1) and Route Type 4 (RT-4) to handle multi-homed nodes. RT-1 tells the network which switches are attached to which common devices or Ethernet segments. An Ethernet segment in the case of a data center is defined as either the bridged network to which an NVE is connected or a bonded link. When connected to a bond, the RT-1 advertisement carries the LACP identifier of the remote node (i.e., the host in our case) as the Ethernet segment ID (ESI). When other NVEs receive the BGP updates of this RT-1 advertisement, they can determine which of their peers are attached to the same host.

RT-4 elects one of the peers as the designated forwarder for multi-destination frames. The RT-4 advertisement carries the mapping of the Ethernet segment to the router servicing the segment. From all the advertisements received for an Ethernet segment, each VTEP selects the segment with the lowest VTEP IP address as the designated forwarder for a virtual network. In this case, a common VTEP IP address is not required across the two peers.

Let's break this down using Figure 4-7. First, the standard allows L1 to be the designated forwarder for one set of nodes—say A and B—and L2 to be the designated forwarder for another set of nodes—say C. In Figure 4-7, each host carries only a single VLAN. But if the hosts supported multiple VLANs, the standard further allows L1 to be the designated forwarder for a node—say A—for one set of VLANs, and L2 as the designated forwarder for that node for a different set of VLANs.

Handling Link Failures

What happens if one of the hosts—say host C in Figure 4-7—loses a link to L1? The answer to this is also implementation specific. Even if the solution involves MLAG, two different implementations might do different things.

In the case of MLAG, using the peer link to reach the host via the other switch is the most common implementation. In our example, both L1 and L2 advertise reachability to C via a common VTEP IP. The underlay multipaths traffic addressed to the common VTEP IP between L1 and L2. When L1 loses its connectivity to C, it withdraws its advertisement of C. However, because L2's advertisement is still valid, L3 and L4 see that MAC_C is still reachable via the common VTEP IP. So, a packet to C can still end up at L1, even though the link between them is down and L1 cannot deliver the packet directly. In such cases, L1 decapsulates the packet and uses the peer link to send the packet unencapsulated to L2. L2 delivers the packet to C.

Is it not possible to avoid the use of the peer link, by, for instance, readvertising the MAC address of C and indicate that it is singly attached to L2? After all, both L1 and L2 have unique IP addresses that belong only to them. Alas, most packet-switching silicon support only a single IP address as the source of a VXLAN VNI, even though a VNI can have both dual-attached and singly attached hosts (A and C in our example). Thus, implementations cannot indicate that C is now attached only to L2's unique VTEP IP. So, even if a dual-attached host has lost a link to one of the VTEP peers, its address is announced using the common VTEP IP address; the peer link is used to deliver the packet to this host.

Without a peer link, most implementations will reencapsulate and send the packet to the other switch. In our example, L1 decapsulates the VXLAN packet, adds a new VXLAN header with the destination IP of L2, and sends the packet.

In EVPN multihoming implementations, the switch that lost the connectivity to the host will also withdraw reachability to the ESI identified by the LACP of the host. This is done for both RT-1 and RT-4 routes. The other switch eventually receives these withdrawals. On receiving the RT-4 withdrawal, the remaining switch appoints itself the designated forwarder for C. Receiving the RT-1 withdrawal tells the switch that the host is singly attached to it. The switch can-

not attempt to forward the packet to its peer when it loses connectivity to the host. In our example, when L2 receives the withdrawal originated by L1, it knows that if it loses the link to C, it cannot forward a packet to L1. However, reencapsulating a VXLAN packet without routing violates the split-horizon check. Thus, this model needs additional support from the underlying switching silicon.

As of version 4.0.1, FRR does not support the EVPN multihoming model. So, it relies on the MLAG solution with peer link discussed previously.

Duplicate Multidestination Frames

When the common VTEP IP model and ingress replication are used, only one of the pair of switches gets a packet. This is because the other VTEPs represent the pair with a single common VTEP IP. However, in the model without a shared common VTEP IP address, it is possible for both switches to get a copy of multidestination frames (BUM packets, for example).

To ensure that only one of the pair sends a packet to the end station, the two nodes use the RT-4 message to pick only one of them to deliver a multidestination frame in a virtual network. However, control protocol updates take time, and during the process of electing a designated forwarder or during transitions, the host can either receive no multidestination frames or receive duplicates. There's nothing that can be done about it.

ARP/ND Suppression

ARP requests and GARP packets are BUM packets because these packets are sent to the broadcast address. In the sample topology, if A had ARP'd for E's MAC address and obtained it, it seems unnecessary to have C's ARP request for E's MAC address behave the same way as A's request. L1 can cache E's ARP information and respond directly. This has the good effect of reducing BUM traffic in the network. Neighbor Discovery (ND) is the ARP equivalent for IPv6 addresses. As a result, ND would benefit from the same caching and response by L1. This function of caching a remote host's ARP/ND information and responding to ARP/ND requests for this information is *ARP/ND Suppression*.

ARP/ND suppression uses RT-2 messages to convey the IP address associated with a MAC address in a virtual network. It is up to the implementation to decide whether it wants to send separate MAC and MAC/IP advertisements or use a single message for both. FRR uses a single message update model when ARP/ND suppression is enabled.

Just as the MAC forwarding table marks when entries are learned via a control protocol, entries in the ARP cache (or IP neighbor table in the Linux kernel) are also marked as having been learned via a control protocol. The kernel does not attempt to run ARP refresh for such entries. The kernel also does not age out entries that have been marked as learned via a control protocol. The protocol is responsible for removing these entries' when the remote entries advertisements are withdrawn. FRR also removes them on graceful shutdown or on recovery from a crash.

This functionality must be enabled via configuration. Some implementations allow this configuration only if the NVE is also the gateway router for that virtual network. Cisco's is an example of such an implementation. Linux (and hence Cumulus) allows you to configure this independently of whether the NVE is the gateway router for that network.

Cumulus added support for ARP/ND suppression in the Linux kernel. This functionality is available starting from kernel version 4.14.

Summary

This chapter concerned itself with how bridging works with EVPN. In Chapter 5, we take up routing.

CHAPTER 5

Routing with Ethernet VPN

Ethernet VPN (EVPN) in the data center is the first Layer 2 (L2) Virtual Private Network (VPN) technology to support comprehensive routing as part of a complete solution. We start with a look at the use cases that motivate the need for EVPN to include routing. This sets the stage for the primary goal of this chapter: to understand the different routing models in EVPN. This helps you, as a network administrator or architect, choose the solution that fits your use case. It also helps you in vendor selection because not all vendors support all models or use cases. Finally, this understanding helps in configuring and troubleshooting routing with EVPN.

The chapter is easier if you understand the basics of routing. I do attempt to do a quick run through the basic logic of routing. But readers unfamiliar with routing will need to walk through this chapter more carefully.

The Case for Routing in EVPN

When virtual L2 network overlays and virtual Layer 3 (L3) network overlays were invented, they coexisted as neighbors. There was a clear fence that separated the two and neither attempted to cross it. This mimicked the nonvirtualized world. Both in the control protocols used and in the packet-forwarding behavior, the two realms differed. In the service provider (SP) world, two sites were connected via either a virtual L2 network or a virtual L3 network. Rarely, if ever, were they connected as both. So, what use cases drive the need for a comprehensive routing solution in EVPN?

Routing Use Cases in the Data Center

Consider an enterprise that wants to replace its VLAN-based infrastructure with one using VXLAN. It is possible that all the storage nodes are in a separate L2 network by themselves. Devices in other L2 networks can access the common storage only via routing. Another example is the classical three-tier application architecture in which the web tier, application tier, and database tier are isolated from one another by putting each in a separate virtual L2 network. A third common example that involves routing is communication with the outside world. What's common in all cases is that a single VTEP will need to bridge traffic between hosts in the same virtual L2 network, and route traffic between hosts in different virtual L2 networks.

To address these issues, the IETF (Internet Engineering Task Force) draft on integrated routing and bridging[1] was crafted. It specifies how routing works with EVPN in the data center.

Routing Models

Routing with EVPN isn't as simple as vanilla routing. To understand why this is so, and why we have anything like multiple routing models, is our first task. As a prelude, I'll devote the next two paragraphs to an abbreviated overview of routing.

An endpoint in an L2 network can talk to another endpoint in a different L2 network only via routing. The router, the device that performs routing, has an interface in each L2 network it routes between. This router interface is called a Switched VLAN Interface (SVI). An SVI has at least one IP address and at least one Media Access Control (MAC) address. It is common for endpoints to have a single route that points to the SVI's IP address as the next-hop IP address. Such a route has the IP address 0.0.0.0/0 (or ::/0 in IPv6) and is called the *default route*. A router routes packets received on a routed interface addressed to that interface's MAC address. Address Resolution Protocol (ARP) is used in IPv4 networks to get the MAC

1 If you're unfamiliar with the IETF world, drafts are not fully ratified standards (called RFC in IETF parlance) but can still be fairly complete, stable, and implemented by vendors.

address associated with an IP address. IPv6 networks have a similar mechanism called ND (Neighbor Discovery).

To determine how to route a packet, the router looks up the destination IP address of the packet in an IP forwarding table. The IP forwarding table is also called the Forwarding Information Base (FIB). The lookup involves two key fields: the Virtual Routing and Forwarding (VRF) associated with the packet and the destination IP address in the packet. The result is at least the outgoing interface. If the next hop is not the final destination, the result also contains the next hop's IP address. The router obtains the MAC address of this next hop IP address from the ARP (or ND) table. If the outgoing interface is an SVI, the router looks up this MAC address in the MAC table for that L2 network. This lookup yields the actual outgoing interface.

The silicon implementations of these three tables and the lookup pipeline will look different due to their need to sustain massive packet forwarding rates. For the purposes of this discussion, we'll stick with the logical model of these tables.

With that concise overview, let us turn back to understanding the problems that need to be addressed by EVPN routing. We'll use the sample topology shown in Figure 4-1. We'll assume the router's SVI is assigned the addresses 10.1.1.1/32 and 10.1.2.1/32 in the red and blue SVI respectively. A's IP address is IP_A, and F's IP_F.

Let's assume that A in the red network wants to talk to F in the blue network. Because F is in a different L2 network, A looks up the routing table to decide who the next hop for reaching F is. As we mentioned earlier, hosts usually have a single default route pointing to the first-hop router. The ARP table contains the MAC address for reaching the first-hop router. So, A sends a packet with the destination MAC address of the first-hop router's SVI. In traditional networks, a single router (or a pair of routers) is the first-hop router for an L2 network. In Figure 4-1, the red network is attached to leaves L1, L2, and L4. How do we select which of them is the first-hop router for the red network? If we pick a pair of them to be first-hop routers as in a traditional network, is there a single active router at any given time or can both of them be active at the same time? Can we have more than two active first-hop routers for a given subnet? Alternatively, can each leaf be the first-hop router for its local endpoints in the red network? In such a case, do we use a separate

router IP address for each leaf? What happens when a VM endpoint moves behind a different leaf at some point in the future? Can it treat the leaf it is now attached to as its first-hop router?

From a different angle, Virtual Extensible LAN (VXLAN) provides a virtual L2 overlay network. How can it transport routed packets? For the case of A talking to F, if L1 is the router, what Virtual Network Identifier (VNI) does L1 put on the packet it sends to L4? Further, after routing, does L1 address the packet to L4 or to F in the inner (or VXLAN payload) header? Before suggesting obvious answers, consider what needs to happen when C wants to send a packet to D. L2 does not even carry the blue network and so has no knowledge of the blue network.

The different routing models in EVPN arise from the different possible answers to these questions. When a single VTEP (or subset of VTEPs) is the first-hop router for a virtual network, the model is called *centralized routing*. If each VTEP is the first-hop router for its local endpoints on a virtual network, the model is called *distributed routing*. When L1 bridges the packet to F directly after routing and puts the F's VNI (blue) in that packet, it is called the *asymmetric model* of routing. When L1 routes the packet to L4 and puts in a new VNI, which represents the VRF and that's neither red nor blue, it is called the *symmetric model* of routing.

Wait a minute, you say. Isn't this all a bit too complicated? Can't you just pick one answer that works well and use that instead of trying to support all possible answers? The distributed, symmetric model makes the cut in most cases. Unfortunately, there are other factors at play such as interoperability, opposing viewpoints from vendors, and maybe your own specific need, that make understanding all of this useful and important.

Where Is the Routing Performed?

This section lays out the consequences of locating the routing function in an EVPN network. As described in the previous section, there are two possible ways to locate routing: centralized and distributed.

Centralized Routing

Centralized routing is the simplest model to understand between the EVPN models. It preserves the original model of a single first-hop routing endpoint (or a pair of endpoints) for an L2 network. A common way to deploy this model is shown in Figure 5-1. B1 and B2 are the border leaves and act as the centralized routers. If most of the traffic is bound outside the data center—that is, north–south—this deployment is efficient with optimal traffic flow. Another advantage of this model is that it is possible to deploy various services such as firewalls and load balancers off these routers.

VXLAN routing was not widely available from most merchant silicon chips[2] until 2017 or so. Thus, this topology provided a simple way to insert newer boxes that do VXLAN routing into an existing network without upgrading the entire network.

If you make the spine switches the border nodes, the spine switches must function as VTEPs as well.

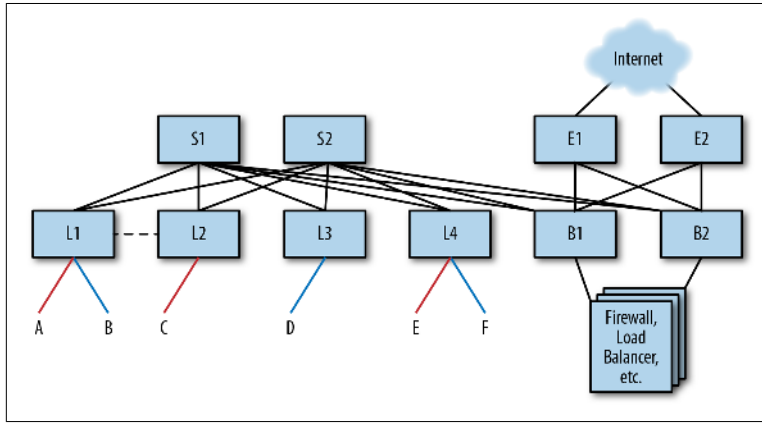

Figure 5-1. A popular topology for deploying centralized routing

2 If you're versed in this area, I mean VXLAN routing was not a single-pass operation through the switching chip.

The main disadvantages of this approach are scalability and potentially non-optimal traffic flow. The centralized routers need to scale to handle the load of ARP messages for the entire network. Using ARP/ND suppression will ease this burden.

If there is significant traffic between the endpoints inside this network (between A and F, for example), this model results in non-optimal traffic flow. The optimal path for the traffic from A to F is from L1 to L4 via one of the spines. With centralized routing, traffic from A to F flows from L1 to B1 (or B2) via one of the spines before flowing to L4 (via one of the spines again). These additional hops add latency and increase congestion in the network.

When both B1 and B2 act as centralized routers, there are two ways to deploy them. In the first, both routers are active for all VNIs. In the second, each is active for one subset of the VNIs and is configured to be on standby for the other VNIs. When both are active for all VNIs, you have to guard against the risk that any flooded packet results in duplicate routed packets. To understand this better, consider when A sends a packet to its first-hop router. If L1 does not know the router's MAC address, it will replicate the packet to all VTEPs interested in the red network (A is in the red network). If both B1 and B2 receive a copy, they will both route the packet, resulting in duplicate packets. Using a model such as the VTEP anycast IP address with ingress replication can ensure only one of them gets the packet. Alternatively, a protocol such as Virtual Router Redundancy Protocol (VRRP) can ensure that only one of them routes the packet for a network. A third way to ensure that only one of them routes a packet is to use the designated forwarder model described in the chapter on bridging.

My recommendation is to use the VTEP anycast IP model with ingress replication owing to its simplicity (no additional protocols and minimal configuration).

Distributed Routing

In distributed routing, every VTEP is the first-hop router for its locally attached networks. So, in Figure 4-1, L1 is the first-hop router for the red and blue networks, L2 is the first-hop router for the red network, and so on. L1 will route packets from A and B, L2 will route packets from C, and so on. This is the most commonly deployed model in the data center.

To ensure consistent access to the endpoints, all first-hop routers in a network have the same first-hop router address and router MAC address. For example, assuming that the router IP is 10.1.1.1 for the red network, L1, L2, and L4 are each configured with this address in the red network. This way a node moving from one leaf to another does not need to relearn its first-hop router MAC and IP addresses. Distributed routing does not need any additional protocols.

The main disadvantage of this approach is that it requires the administrator to rethink the ways services such as firewalls and load balancers are deployed. In the centralized routing model, services were located at the border leaves. If these services are needed only for traffic in and out of the data center, deploying these at the border leaves still works. But if the services are needed for traffic within the data center itself, the best way to deploy the services is on each host itself.

Another way to address the services problem is via the use of VRFs. Put each network that needs traffic to be validated in a different VRF. This isolates the network from any traffic external to itself. The routing path to this network can be via a firewall. By placing the firewall behind B1 and B2 as in Figure 5-1, we can reuse existing physical firewalls while still deploying distributed routing. If not all routed traffic needs to be validated via a firewall, this model works reasonably well. If B1 and B2 also host shared services such as storage nodes, traffic flow is also optimal. A real-world use case of this model is in the use of private clouds, when two separate tenant networks want to communicate with each other. Such access is over a public IP address rather than a private one, and that public IP address can be made reachable via the firewall.

You can deploy both distributed and centralized routing in the same data center.

How Routing Works in EVPN

In this section, we come to the heart of routing in EVPN. As described earlier, EVPN supports two routing models: asymmetric and symmetric.

Asymmetric Routing

In asymmetric routing, the ingress and egress VTEPs behave differently. The ingress VTEP, which is also the first-hop router, does a routing lookup to decide which is the egress VTEP. The inner packet is then addressed to the final destination. So, the ingress VTEP routes, whereas the egress VTEP only bridges. Hence, the name asymmetric routing. This model assumes that the destination network is local to the ingress VTEP; else it couldn't know the MAC address of the final destination.

Let's break this down by looking at a specific example. Using the network in Figure 4-1, let's study the packet flow when A sends a packet destined to F. Let's assume the distributed routing model.

1. A knows that F is in a different subnet. A knows this because it knows its own address is 10.1.1.A/24, whereas F's address is 10.1.2.F/24.[3] When A was assigned an IP address, it was also configured with a default route. The next hop for the default route was set to the first-hop router's IP address, 10.1.1.1/24 in this case. A sends a packet with the destination MAC address set to the MAC address assigned to IP address 10.1.1.1,[4] and the destination IP address set to 10.1.2.F.
2. L1 receives this packet. Because the destination MAC address is its router MAC address, L1 routes the packet.
    a. Because L1 also has endpoints in the blue network, the FIB lookup result is that the subnet 10.1.2.0/24 is local, reachable via the blue SVI.
    b. So L1 looks up its ARP table to see whether it knows how to reach F via the blue SVI. 10.1.2.F is present in its ARP table, because EVPN populated this information at an earlier time.
    c. L1 looks up F's MAC address and sees that F is behind VTEP L4.
    d. L1 looks up L4's IP address in its routing table and sees that it can reach L4 via either S1 or S2. It uses the packet hash to determine which of S1 and S2 to pick. Assume that it picks S1.

3 The third number in the dotted decimal address for F is different from A's.

4 A issues an ARP request for 10.1.1.1's MAC address if it doesn't already have it.

e. It changes the destination MAC address on the packet to MAC_F, and changes the source MAC address to be itself. It adds the VXLAN header with the destination IP address set to L4's VTEP IP address, the source IP address set to L1's VTEP IP address, the VNI set to blue (F's network), and the UDP source port based on the original packet's hash. Finally, it adds an outer Ethernet header with the destination MAC address to S1's MAC address.

f. L1 forwards the packet to S1.

3. S1 receives this packet. The destination IP address is that of L4, so S1 consults its IP FIB and sees that it knows how to reach L4. It changes the destination MAC address on the outer header to be L4's and sends the packet to L4 out $Port_{L4}$.

4. L4 receives the packet.

   a. Because the packet is destined to it both at the MAC and IP address layers and the packet is a VXLAN packet, L4 decapsulates the VXLAN header and saves the information that this packet belongs to the blue VNI.

   b. MAC_F is the destination MAC address in this decapsulated packet. So L4 looks up its MAC table for MAC_F in the blue VNI. It sees that MAC_F in the blue network is out $Port_F$

   c. L4 forwards the decapsulated packet to F.

Figure 5-2 illustrates the packet headers as the packet flows from A to F.

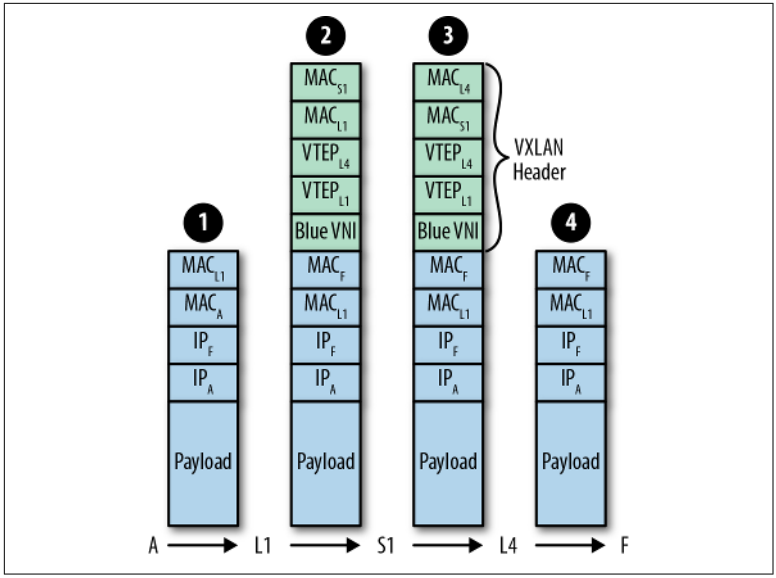

Figure 5-2. Packet headers in packet flow from A to F with asymmetric routing

The key part of the asymmetric routing all happened in step 2. L1 sets the VNI to be blue in the packet and the destination MAC address to be F's. After this, the behavior is no different from the VXLAN bridging model we studied in Chapter 4. The VXLAN packet is routed from L1 to L4, where the VXLAN header is stripped, and the original packet is bridged from L4 to F.

Consider what happens if C wants to send a packet to F. Step 1 is the same as in our previous example, except that the packet is accepted for processing by L2, not L1. Because we're assuming distributed routing, every first-hop router is configured to have the same router IP and MAC addresses for the relevant networks attached to it. L2 does not have the blue network and so it does not know what to do. Thus, asymmetric routing does not work if the destination subnet is not locally attached to the first-hop router.

The solution to this L2 problem is symmetric routing. We look at that in the next subsection and then compare asymmetric and symmetric routing.

Symmetric Routing

In symmetric routing, both the ingress and the egress VTEPs route the packet to its final destination.

Thus, packet forwarding in symmetric routing differs from asymmetric routing in three ways. First, the ingress VTEP routes the packet to the final destination with the egress VTEP as the next hop. Second, the egress VTEP, after decapsulating, does a routing lookup to decide the path to the final destination. Third, the VNI used to carry the packet between the ingress and egress VTEPs is an L3 or VRF VNI. This L3 VNI is different from the L2 VNI of the source or destination network. Table 5-1 highlights the differences between asymmetric and symmetric routing.

Table 5-1. Differences between asymmetric and symmetric routing packet forwarding

	Ingress VTEP behavior	Transit VNI	Egress VTEP behavior
Asymmetric routing	Post routing, packet is *bridged* to final destination via egress VTEP	Destination node's VNI	Decapsulate, then *bridge* packet to destination in the MAC header
Symmetric routing	Post routing, packet is *routed* to final destination via egress VTEP	L3 VNI	Decapsulate, then *route* packet to destination using the IP header

Figure 5-3 shows the packet headers for the packet flowing from A to F with symmetric routing for the network topology shown in Figure 4-1. The differences in the header from its asymmetric cousin are shown in bold.

Let us study once again the actions at each hop as the packet flows from A to F, but this time using symmetric routing.

1. The packet flow from A to L1 remains the same as in the asymmetric case. A knows that F is in a different subnet from it. A sends a packet to its first-hop router, 10.1.1.1, with the destination MAC address of 10.1.1.1 and the destination IP address of 10.1.2.F.
2. L1 receives this packet.
   a. Because the destination MAC is its router MAC address, L1 attempts to route the packet.

b. First, it determines the VRF associated with the red network (because A is on the red network). Let's call it the green VRF. L1 is also on the blue network, so its FIB lookup result is that the subnet 10.1.2.0/24 is local to it. However, because of symmetric routing, the routing table also contains a more specific route to 10.1.2.F/32 (we see how it knows in the next subsection). So, this route lookup wins over the subnet route. This route has the next-hop IP as the VTEP of L4, which is 10.1.2.1, reachable via the egress interface green SVI.

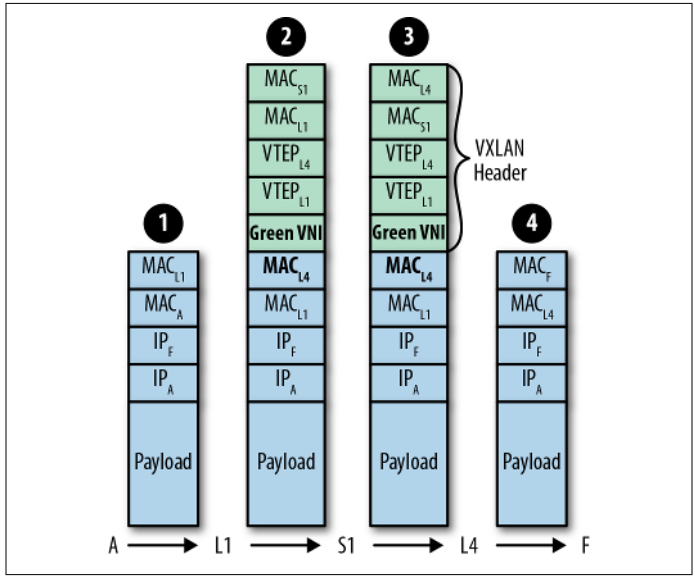

Figure 5-3. Packet headers in packet flow from A to F with symmetric routing

c. So, L1 looks up its ARP table to see whether it knows how to reach L4 via the green SVI.

d. It finds the MAC address associated with this entry (populated via EVPN at an earlier time). It then looks up the MAC table for this MAC address in the green network (a special L2 VNI created for the green VRF). L1 finds that this node is behind VTEP L4.

e. L1 looks up L4's IP address in its routing table (but in the default VRF this time) and sees that it can reach L4 via either S1 or S2. It uses the packet hash to determine which of S1 and S2 to pick. Assume that it picks S1.

f. It changes the destination MAC address on the packet to MAC_{L4} and changes the source MAC address to be its own MAC, MAC_{L1}. It adds the VXLAN header with the destination IP address set to L4's VTEP IP address, the VNI set to green, and the User Datagram Protocol (UDP) source port based on the original packet's hash. Finally, it adds an outer Ethernet header with the destination MAC address set to S1's MAC address.

g. L1 forwards the packet to S1.

3. S1 receives this packet. The destination IP address is that of L4, so S1 consults its IP FIB table and sees that it knows how to reach L4. It changes the destination MAC address on the outer header to be L4's and sends the packet to L4.
4. L4 receives the packet.
   a. The packet is destined to L4 at both at the MAC and IP address layers and this is a VXLAN packet. So L4 decapsulates the VXLAN header and saves the information that this packet belongs to the green VNI.
   b. The destination MAC address on the decapsulated packet is MAC_{L4}, so L4 attempts to route the packet.
   c. The green VNI maps to the green VRF. So L4 looks up 10.1.2.F in the green VRF in the FIB, which returns the result that the subnet is local on the blue SVI.
   d. L4 then looks up 10.1.2.F in the blue SVI in the ARP table and finds the associated MAC address, which is MAC_F. It now looks up the MAC address MAC_F in the blue network and finds that the egress port is $Port_F$.
   e. Thus, L4 forwards the decapsulated packet to F. F receives the packet.

If you've stayed with me so far, good. There are a lot of concepts to tease apart here and understand. To summarize the forwarding behavior with symmetric routing:

- The packet is routed twice in the overlay: once at the ingress VTEP and once at the egress VTEP.
- Because the egress VTEP is treated as a routed hop in the overlay, the model assumes that they share a common inter-VTEP VNI called the L3 VNI. In traditional non-EVPN routing, the

inter-router link is neither in the source network nor in the destination network. Symmetric routing mimics that behavior by putting the VXLAN tunnel connecting the ingress and egress VTEP in a new L3 VNI. This is shown in Figure 5-4.

- Symmetric routing requires all routing to take place in the context of a VRF. On ingress, the source VNI determines the VRF which also provides the L3 VNI to be used. On the egress VTEP, the L3 VNI in the packet determines the VRF to be used in the routing table lookup.

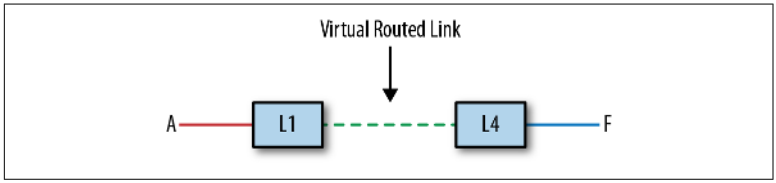

Figure 5-4. Model of inter-VTEP link in symmetric routing

Back to where asymmetric routing became flummoxed: if C wants to talk to F, symmetric routing supports it just fine. The ingress VTEP, L2, never needs to look up any information using the destination network of F (blue), and so the packet forwarding works just as it did on L1.

However, because symmetric routing expects the egress VTEP to perform routing, the centralized model does not work with symmetric routing; it works only with asymmetric routing.

VRFs in EVPN Routing

In EVPN, routing is assumed to occur within the context of a VRF. This is true independent of whether the model is symmetric or asymmetric. The underlay routing table is assumed to be in the default or global routing table, while the overlay routing table is assumed to be in a VRF-specific routing table. It is possible to have asymmetric routing work without the use of VRFs, as demonstrated by the example in "Asymmetric Routing" on page 62. But VRFs are necessary if the endpoints have to communicate to the external world, because RT-5 advertisements are involved. RT-5 advertisements always occur in the context of a VRF: the L3 VNI signaled in the advertisement. Thus, to preserve a uniform routing model, I strongly recommend always using VRFs with EVPN routing.

Summarized Route Announcements

So far, all the advertisements we've seen are fully qualified /32 routes. The MAC/IP (RT-2) advertisement does not support advertising summarized or prefix routes such as /16 or /24 routes. But the inability to advertise summarized routes affects the scalability of the solution. For example, how does routing work when host A needs to communicate with the outside world? What does L1's routing table look like to support this?

If you consider a network such as the one in Figure 5-1, the edge routers (E1 and E2) commonly advertise only the default route to B1 and B2. In just about every deployment, the spines and the leaves do not carry the routing table of the external world. They just carry the default route which gets them to B1 and B2 and from there to E1 and E2. The default route is 0.0.0.0/0, which has a non-/32 prefix (IPv6 has ::/0 as the default route). How is this advertised into the EVPN network of L1-L4?

A new route type, type 5 (RT-5) was introduced to support this use case. The draft standard on IP prefix advertisement defines the definition and use of this route type.

Advertising summarized routes is not done automatically; you need to configure it. We cover this in Chapter 6.

BGP Support for EVPN Routing

In this section, we go over the BGP support for EVPN routing. Although this knowledge might not be necessary for daily use, it is nevertheless useful in troubleshooting.

Asymmetric routing does not need any additional support in BGP. It uses the information in the RT-2 (Route Type 2) message. After an SVI is created on a VTEP and an IP address is assigned, the box is ready to perform asymmetric routing. If ARP/ND suppression had been enabled earlier, the neighbor table is already being populated. If ARP/ND suppression isn't enabled, an RT-2 advertisement is sent by a VTEP each time the IP neighbor table is updated by local endpoints' ARP requests.

Symmetric routing needs more love.

There are three additional pieces of information required for symmetric routing: the VNI to use between the ingress and egress

VTEPs, the next-hop IP address (which is the egress VTEP's IP address), and the router MAC address of the egress VTEP. These need to be conveyed in some BGP UPDATE message. The egress VTEP's IP address is always carried in the NEXTHOP attribute of the BGP advertisement. The RT-2 message has provisions to support carrying the other two pieces of information.

RT-2 supports carrying two VNIs (or Multiprotocol Label Switching [MPLS] labels in the original EVPN standard). With asymmetric routing, only one of the two VNI fields is used, to announce the L2 VNI. With symmetric routing, both VNI fields are used, one for L2 VNI and the other for the transit or L3 VNI.

The route also needs to carry the egress VTEP's MAC address, because this address is used as the destination MAC address on the inner packet. This MAC address is carried as a new BGP Extended Community in the advertisement. This new extended community is called Router MAC Extended Community.

Comparing Asymmetric and Symmetric Models

Figure 5-5 shows how the various fields of the VXLAN header are populated in the case of asymmetric and symmetric routing when L1 routes a packet destined to IP_F. The differences are shown in bold. This information is useful to a network administrator who needs to troubleshoot a problem in the EVPN network.

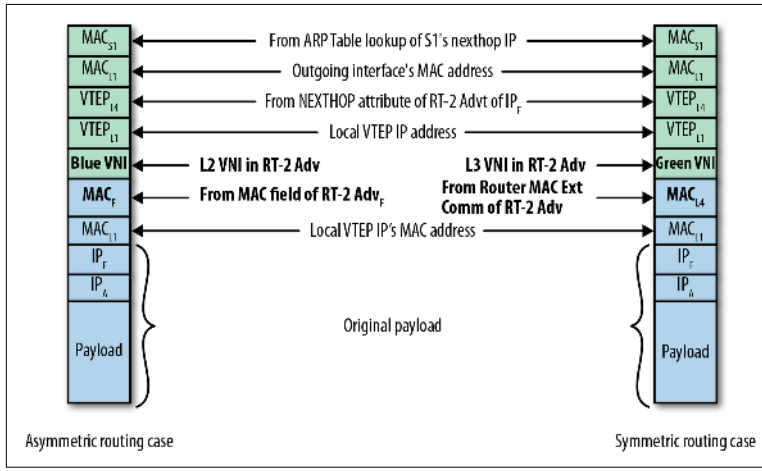

Figure 5-5. How the fields in the VXLAN packet are populated by L1 on routing IP_F

Another way to compare the two routing models is to look at how the forwarding tables are consumed in each model. In asymmetric routing, because the packet is bridged after routing—i.e., the destination network is local to the ingress VTEP—all the remote endpoints are present as ARP/ND entries while the routing table remains largely empty. Because all the remote endpoints are in the ARP/ND table, they're all present in the MAC forwarding table, as well.

In symmetric routing, for the destination networks that are local, the behavior is the same as asymmetric routing. For the destination networks that are not local, the routing table contains all the remote endpoints (as /32 routes) with the respective egress VTEP as the next hop. The only entry in the ARP/ND table for these remote entries is the egress VTEP out the VRF VNI. So, the MAC forwarding table only contains a single MAC address entry (that of the egress VTEP) for all the remote endpoints behind that egress VTEP. However, if a destination network is local, there is really no saving in the overall forwarding table space for that VNI with symmetric versus asymmetric routing.

From the perspective of VNIs, symmetric routing scales much better than asymmetric since each VTEP can only contain the VTEPs it cares about while allowing connectivity to non-local VNIs. On the other hand, if all VNIs are not present on all VTEPs, if VMs either move about or can be spun up or down, some entity has to ensure that the local VTEP will have the VNI associated with the VM as configured if it's not present. The same entity also has to ensure that a VNI is destroyed on a VTEP when the last VM using it shuts down. Unless you're operating a private cloud where the scale of VNIs matters, my recommendation is to always configure all the relevant VNIs on all the VTEPs. This limits the amount of state churn in the network when VMs spin up and down and eliminates the need for an orchestrator between the compute and the network to configure and delete VNIs on VM spin up and down.

From a configuration standpoint, symmetric routing requires additional configuration: for the L3 VNI. However, if external connectivity is required, then symmetric routing is required nonetheless, and so the configuration of the L3 VNI doesn't add much.

Another difference between asymmetric and symmetric routing models is in the use of VRFs. Asymmetric routing can function in

the same VRF as the underlay network, that is in the default VRF, if the user does not have a requirement for multiple VRFs. Symmetric routing requires the explicit configuration of an L3 or VRF VNI, even when a single VRF is all that is required. It cannot function in the default VRF. However, to keep the underlay and the overlay networks separate, it is better to use nondefault VRF even in the asymmetric routing case.

Vendor Support for EVPN Routing

Now that the theory of routing with EVPN has been made clear, we turn our attention to more pragmatic concerns: what models are supported by various vendors and merchant switching silicons.

There are two camps (we can debate over a beer whether they're dueling) among the vendors in support of the various models. Arista, Cisco, and Cumulus support the symmetric model, whereas Cumulus and Juniper support the asymmetric model.

There are quite a few options when it comes to switching silicon that supports VXLAN routing. Network administrators will need to validate for themselves whether the chip in the product they intend to buy supports VXLAN routing (the functionality is commonly referred to as Routing In and Out of Tunnels [RIOT]) in a single pass.

Summary

We explored how routing works in the context of EVPN. Reread this chapter to get more familiar with the various routing options. Armed with the theory of bridging and routing in EVPN, we next examine how to configure and administer EVPN networks.

CHAPTER 6

Configuring and Administering Ethernet VPN

All the theory and grounding of the past chapters are put to practical use in this chapter. We study Ethernet VPN (EVPN) configuration for the three most common deployment scenarios: *asymmetric centralized routing*, *asymmetric distributed routing*, and *symmetric routing*. In each case, we first present the final configuration for the open source routing suite, Free Range Routing (FRR). We then dissect the configurations so that we can learn how to put them to use in other scenarios.

The configuration with FRR is radically simpler than in other routing suites but interoperates with the more complex configuration of those other suites. To help users, we provide a sample cheat sheet at the end providing an FRR configuration along with its Cisco NX-OS equivalent.

Although we're focusing on EVPN, we also have to show the dataplane configuration because EVPN configuration reflects some of the aspects of that configuration. It includes Virtual Routing and Forwarding (VRF), Virtual Network Identifier (VNI), and Virtual Local-Area Network (VLAN)-to-VNI mappings. These parts are common to all networks and configuring them is vendor specific, so I limit the discussion to stating the logical pieces to configure. In some cases, I show snippets of configuration on a Debian Linux distribution. I assume use of the ifupdown2 package for network interface configuration.

It is not sufficient to configure a network. We also need to examine the running state and troubleshoot it. Toward that end, we examine various commands that display the running state of an EVPN. We end the chapter with deployment considerations for three-tier Clos networks. There is a companion GitHub repository that contains the configuration discussed, along with the ability to instantiate the network with VMs using Vagrant.

ifupdown2 is a backward-compatible successor to ifupdown, Debian's network interface configuration manager. ifupdown2's design lends itself to configuring routers and bridges. It also supports configuring new features such as Address Resolution Protocol (ARP)/Neighbor Discovery (ND) suppression. Ubiquitously used by Debian system administrators, ifupdown was designed for host networking and so does not scale when used with routers. ifupdown2 is an open source project with official packages for Ubuntu and Debian distributions.

The Sample Topology

Figure 6-1 presents the sample topology used in the rest of this chapter. The topology is a scaled-down version of the most common topology I have encountered in EVPN customer deployments. The number of VLANs deployed to servers are tens to a couple of thousands in real life, whereas we use only two. Similarly, there are many more leaves, spines, exit (or border) leaves, and so forth in real life. The routers are all assumed to be running Cumulus Linux (version 3.5.1 or greater) with FRR. The servers are all assumed to be running Ubuntu 16.04 LTS release.

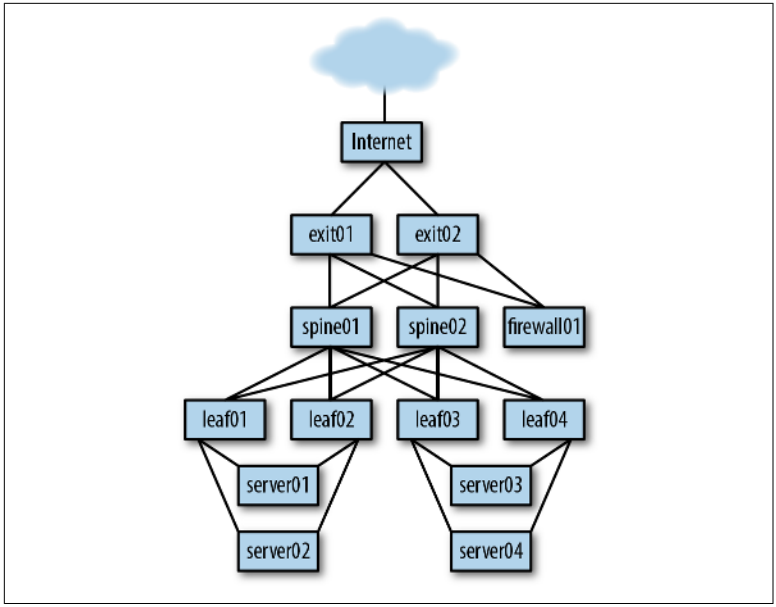

Figure 6-1. Sample network topology for EVPN configuration

We assume two virtual networks, *red* and *blue*, mapped to a single overlay VRF. Red is mapped to VLAN 3 and Blue to VLAN 4.[1] The even-numbered servers support VLAN 4, whereas the odd-numbered servers support VLAN 3. The VXLAN VNIs are identity mapped from their VLAN IDs; that is, *vni3* for VLAN 3 and *vni4* for VLAN 4. The Switched VLAN Interface (SVI) representing the routed interface is named after the VLAN to and from which it's routing. So *vlan3* and *vlan4* are the SVI names.

Each server is dual attached to two switches. The server interfaces are bonds (aka aggregated links or etherchannel). I'm assuming that the switches support a version of the Multichassis Link Aggregation (MLAG) protocol to provide the illusion of a single device to the servers. The two VTEPs of an MLAG pair use the same VTEP IP address.

I assume that ingress replication is used for multidestination frames, BUM (Broadcast, unknown unicast and multicast) packets. Unless

1 VLAN 1 is traditionally never used to carry real data, and I've skipped VLAN 2 to make the mapping between odd and even servers more natural.

the volume of multidestination frames is high enough in your networks to benefit from the use of underlay multicast, I recommend ingress replication for multidestination frames.

The connection to the external world is walled off by a firewall enforcing policy-based access. The firewall is assumed to be attached as an L3 hop connecting across VRFs. Most switching silicon implementations dictate that the VXLAN Tunnel End Point (VTEP) underlay be in the default VRF. So, access to the internet is in a VRF called *internet-vrf*. The server traffic is part of a separate VRF called *evpn-vrf* and has a VNI of 4001 and a corresponding VLAN of 4001. These nodes in the *evpn-vrf* can reach the external world via the firewall.

We allocate the loopback IP addresses for the routers from the 10.0.0.0/24 subnet. The subnet for VLAN 3 is 10.1.3.0/24, whereas that for VLAN 4 is 10.1.4.0/24. The gateway IP addresses for these subnets are 10.1.3.1 and 10.1.4.1, respectively. The server IP address is the server number plus 100. For example, server01 is in VLAN 3 with an IP address of 10.1.3.101, whereas server04 is on VLAN 4 with an IP address of 10.1.4.104.

The routers all have a separate out-of-band management network connected to their eth0 port. The addresses to this interface are doled out via a DHCP server on the management network. This DHCP server runs on a node called *oob-mgmt-server* in the Vagrant setup. All access to the individual nodes is via Secure Shell (SSH) from this *oob-mgmt-server* as user *cumulus*.

Configuration Cases

This chapter presents configurations for three cases. Table 6-1 shows the cases and why they are interesting.

Table 6-1. The configuration cases described in this book

Case	Reason it's interesting
Asymmetric centralized routing	Leaves run switching silicon that can only do VXLAN bridging, with exit leaves doing the routing
Asymmetric distributed routing	The model used by Juniper, and therefore valuable for a Juniper-interop scenario
Symmetric routing	The default EVPN routing model; also the model used to interoperate with equipment from Cisco and Arista

We discussed the differences between these cases in the Chapter 5. But let's do a quick recap of the three scenarios:

- Centralized routing requires only the exit leaves to know about routing. All traffic between VNIs goes to the exit leaves and back. This is clearly inefficient but doesn't matter if most of the traffic is outbound from the network, because the exit leaves are on the exit path.
- In asymmetric distributed routing, every leaf does the function of routing, making for more efficient traffic flow across VNIs.
- Symmetric distributed routing is what I consider the default routing model for its scalability. As discussed earlier, compared to the asymmetric distributed case, a packet makes two routing hops in the overlay between the source and the destination endpoints.

In distributed routing, the task of the exit leaves is limited to connecting to service boxes such as firewall and load balancing for all traffic between VRFs, including traffic in and out of the data center.

For each case, we present the configuration for the leaves and the exit leaves. The spines are part of the underlay, so their configuration is invariant across the cases. The server configuration and the firewall's routing configuration also remain the same across the different cases. The internet-facing router's configuration is complex and beyond the scope of this book. Its sample configuration in the GitHub repository only announces the default route via BGP.

Configuring the MTU

With VXLAN, as with other tunnels, it is important to set the correct Maximum Transmission Unit (MTU) across all the links. MTU determines the largest IP packet size acceptable on a link. With VXLAN, you must make an allowance to not exceed the MTU when the VXLAN tunnel header is added. 1,500 bytes is the most common MTU. However, it is common practice to set the MTU to much larger numbers for traffic inside the data center. The best practice recommendation is to set an MTU of 9,216 for the inter-switch links, and an MTU of 9,000 for the server-facing ports, which don't carry the VXLAN header.

In Linux, you can use the `ip route` command to set the MTU, but to have the setting stick across reboots, you must add the `mtu` line to the interface section in each host's */etc/network/interfaces* file[2] like this:

```
interface swp51
   mtu 9216
```

Under Cumulus Linux, you can use the Network Command Line Utility (NCLU) tool to set the MTU as follows:

```
net add interface swp51 mtu 9216
```

On a Cisco NX-OS or Arista box, use the command `mtu` (`mtu 9216` for example) under the appropriate interface subsection.

The End First: Complete FRR Configurations

In this section, we present the entire FRR configuration for each of the cases described in Table 6-1. The explanations of the relevant sections are provided later. The main reason for presenting it this way is because I posit that users will want to look at the sample configuration multiple times, but the explanations for the configurations themselves are not interesting after they are understood. My personal preference also is a factor in presenting configurations this way: I like to see the whole picture before breaking down the structure and understanding each piece.

The Invariants: Configuration for the Spines, Firewall, and Servers

The configuration for spine01 looks as follows:

```
router bgp 65020
    bgp router-id 10.0.0.21
    bgp bestpath as-path multipath-relax
    neighbor peer-group fabric
    neighbor fabric remote-as external
    neighbor swp1 interface peer-group fabric
    neighbor swp2 interface peer-group fabric
    neighbor swp3 interface peer-group fabric
    neighbor swp4 interface peer-group fabric
    neighbor swp5 interface peer-group fabric
    neighbor swp6 interface peer-group fabric
```

2 This file is processed by ifupdown and ifupdown2.

```
    address-family ipv4 unicast
        neighbor fabric activate
        redistribute connected route-map LOOPBACKS
    address-family l2vpn evpn
        neighbor fabric activate
!
route-map LOOPBACKS permit 10
    match interface lo
!
```

spine02 has the same configuration except for the `router-id`, which is set to its loopback IP address. There's nothing noteworthy in this configuration. Because we're using external Border Gateway Protocol (eBGP) to carry EVPN, we need to have the spines receive, process, and transmit the EVPN routes. So EVPN Address Family Indicator (AFI)/Subsequent Address Family Indicator (SAFI) is activated across all the spines' neighbors. What is shown is the entire FRR configuration. The interfaces configuration sets the MTU to 9216 on all its links and has an IP address assigned to the loopback interface.

The servers are running Ubuntu 16.04 with the interfaces `eth1` and `eth2` configured as a bond. The configuration of server01 looks as follows:

```
auto lo
iface lo inet loopback

auto eth1
iface eth1 inet manual
  bond-master uplink
  # Required for Vagrant
  post-up ip link set promisc on dev eth1

auto eth2
iface eth2 inet manual
  bond-master uplink
  # Required for Vagrant
  post-up ip link set promisc on dev eth2

auto uplink
iface uplink inet static
  mtu 9000
  bond-slaves none
  bond-mode 802.3ad
  bond-miimon 100
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy layer3+4
```

```
    address 10.1.3.101
    netmask 255.255.255.0
```

In Vagrant, it's important to set the interfaces in promiscuous mode to allow multicast and broadcast packets to be transmitted and received. Other than that, the rest of the configuration is quite straightforward. The configuration for the rest of the servers looks the same except for the IP address configured.

I assume that the firewall is running BGP, a common option. The firewall configuration looks as follows:

```
router bgp 65053
    bgp router-id 10.0.0.53
    bgp bestpath as-path multipath-relax
    neighbor fabric peer-group
    neighbor fabric remote-as external
    neighbor eth1.10 interface peer-group fabric
    neighbor eth1.11 interface peer-group fabric
    address-family ipv4 unicast
        neighbor fabric activate
        redistribute connected route-map LOOPBACKS
!
route-map LOOPBACKS permit 10
    match interface lo
!
```

The firewall in the sample topology is running Ubuntu 16.04 LTS, as well. FRR is installed on it via the deb from frrouting.org's GitHub repo. The internet-facing traffic routes are received via the `eth1.10` subinterface, whereas the internal network–facing traffic routes are received via the `eth1.11` subinterface. There is nothing really noteworthy in the configuration.

Centralized Routing

For the case of centralized routing, we assume that the exit leaves are the only nodes performing EVPN routing. The rest of the leaves are merely doing EVPN bridging. ARP/ND suppression is also enabled. The FRR configuration for `leaf01` is as follows:

```
router bgp 65011
    bgp router-id 10.0.0.11
    bgp bestpath as-path multipath-relax
    neighbor fabric peer-group
    neighbor fabric remote-as external
    neighbor swp51 interface peer-group fabric
    neighbor swp52 interface peer-group fabric
    address-family ipv4 unicast
```

```
        neighbor fabric activate
        redistribute connected route-map LOOPBACKS
    !
    address-family l2vpn evpn
        neighbor fabric activate
        advertise-all-vni
    !
!
route-map LOOPBACKS permit 10
    match interface lo
!
```

The other leaves have the exact same configuration except for a different Autonomous System Number (ASN) and `router-id`. If you want to use EVPN only for bridging and not for routing, this configuration is all you need for FRR. ARP/ND suppression is configured per VNI in the interfaces configuration file. We show this in "Configuring the Overlay: Interfaces" on page 90.

The exit leaves have a similar configuration, except that they also need to advertise themselves as the default gateways for VNIs 3 and 4. The exit leaves also must communicate with the internet edge router, internet, via the `internet-vrf` and the firewall. As described in "Underlay configuration of exit leaves" on page 86, BGP peering with the firewall must be in each VRF configured on the exit leaf. In our example, the exit leaves have three VRFs: the default VRF for the underlay, the `internet-vrf` for connecting to the external world, and the `evpn-vrf` for the internal EVPN network.

`exit01`'s FRR configuration looks as follows:

```
router bgp 65041
    bgp router-id 10.0.0.41
    bgp bestpath as-path multipath-relax
    neighbor fabric peer-group
    neighbor fabric remote-as external
    neighbor swp51 interface peer-group fabric
    neighbor swp52 interface peer-group fabric
    neighbor 10.253.0.1 interface remote-as external
    address-family ipv4 unicast
        neighbor fabric activate
        neighbor swp1.3 allowas-in 1
        redistribute connected route-map LOOPBACKS
    !
    address-family l2vpn evpn
        neighbor fabric activate
        advertise-all-vni
        advertise-default-gw
    !
```

```
!
route-map LOOPBACKS permit 10
    match interface lo
!
router bgp 65041 vrf internet-vrf
    bgp router-id 10.0.0.41
    bgp bestpath as-path multipath-relax
    neighbor internet peer-group
    neighbor internet remote-as external
    neighbor 10.253.1.1 interface peer-group internet
    neighbor swp44 interface peer-group internet
    address-family ipv4 unicast
        neighbor internet activate
        neighbor 10.253.1.1 allowas-in 1
        redistribute connected route-map INTERNET
!
route-map INTERNET permit 10
    match interface internet-vrf
!
```

We dissect this configuration in "Underlay configuration of exit leaves" on page 86.

Asymmetric Distributed Routing

In the case of asymmetric distributed routing, each of the leaves acts as a first-hop router for EVPN traffic. All leaves have the same SVI IP address and Media Access Control (MAC) address, as described in Chapter 5. Because SVI is enabled for the VLANs local to the router, there is no need to enable ARP/ND suppression.

The non-exit leaves' configuration is identical to the case with centralized routing. After you configure an SVI for a VNI/VLAN, asymmetric routing is automatically enabled.

The exit leaves have the most complex configuration in this setup. They're peering with the internet-facing router on the `internet-vrf`, with the underlay (spines) in the default VRF, and with the overlay (via the spines) in the `evpn-vrf`. In each of the VRFs, they're also peering with the firewall. Following are three separate BGP sections, one for each of these VRFs.

`exit01`'s configuration looks as follows:

```
vrf evpn-vrf
   vni 104001
!
! default VRF peering with the underlay and the firewall
```

```
!
router bgp 65041
    bgp router-id 10.0.0.41
    bgp bestpath as-path multipath-relax
    neighbor fabric peer-group
    neighbor fabric remote-as external
    neighbor swp51 interface peer-group fabric
    neighbor swp52 interface peer-group fabric
    neighbor swp1.2 interface remote-as external
    address-family ipv4 unicast
        neighbor fabric activate
        neighbor swp1.2 allowas-in 1
        redistribute connected route-map LOOPBACKS
    !
    address-family l2vpn evpn
        neighbor fabric activate
        advertise-all-vni
    !
!
route-map LOOPBACKS permit 10
    match interface lo
!
! evpn vrf peering to announce default route to internal net
!
router bgp 65041 vrf evpn-vrf
    bgp router-id 10.0.0.41
    neighbor swp1.3 interface remote-as external
    address-family ipv4 unicast
       network 10.1.3.0/24
       network 10.2.4.0/24
       neighbor swp1.3 allowas-in 1
    exit-address-family
    !
    ! This config ensures we advertise the default route
    ! as a type 5 route in EVPN in the main BGP instance.
    ! The firewall peering is to get the default route
    ! in the evpn-vrf from the internet-vrf. Firewall
    ! does not peer for l2vpn/evpn.
    !
    address-family l2vpn evpn
      advertise ipv4 unicast
!
! internet vrf peering to retrieve the default route from the
! internet facing router and give it to the firewall.
!
router bgp 65041 vrf internet-vrf
    bgp router-id 10.0.0.41
    bgp bestpath as-path multipath-relax
    neighbor internet peer-group
    neighbor internet remote-as external
    neighbor swp1.4 interface peer-group internet
```

```
    neighbor swp44 interface peer-group internet
    address-family ipv4 unicast
        neighbor internet activate
        neighbor swp1.4 allowas-in 1
        redistribute connected route-map INTERNET
    !
!
route-map INTERNET permit 10
    match interface internet-vrf
!
```

Symmetric Routing

In the case of symmetric routing, the exit leaves have the same configuration as in the previous section. It is in the configuration of VLANs and VNIs on an exit leaf that symmetric and asymmetric routing configurations differ. The non-exit leaf configuration is also different. The primary difference is that the leaves need to configure an additional L3 VNI in standards parlance. This VNI is what is transported as the VNI field between the ingress and egress VTEP. Most of the configuration for this L3 VNI (and its corresponding VLAN config) is in the interfaces configuration rather than in EVPN itself, except for one tiny section that marks the VNI for a VRF.

`leaf01`'s FRR configuration for symmetric routing looks as follows:

```
vrf evpn-vrf
      vni 104001
!
router bgp 65011
    bgp router-id 10.0.0.11
    bgp bestpath as-path multipath-relax
    neighbor fabric peer-group
    neighbor fabric remote-as external
    neighbor swp51 interface peer-group fabric
    neighbor swp52 interface peer-group fabric
    address-family ipv4 unicast
        neighbor fabric activate
        redistribute connected route-map LOOPBACKS
    !
    address-family l2vpn evpn
        neighbor fabric activate
        advertise-all-vni
    !
!
route-map LOOPBACKS permit 10
    match interface lo
!
```

Though strictly not required by EVPN, the current FRR implementation requires that the value of the L3 VNI for a given VRF be the same across all of the boxes.

Dissecting the Configuration

Broadly speaking, we can split EVPN configuration into two parts: configuring the underlay and configuring the overlay. Each of those parts can be broken up into interface configuration and routing configuration.

Configuring the Underlay

Configuring the underlay is essential to ensure that the VTEPs can reach one another. The configuration consists of assigning IP addresses to VTEPs and advertising those addresses via a routing protocol. In the underlay, every leaf with only singly attached hosts has a single IP address that can also be the loopback device's IP address. Spines need a single IP address. If you're using numbered interfaces, you'll need to configure per inter-switch link IP addresses, usually from the /31 or /30 subnet. Exit leaves can have a more complex role, so we examine their configuration in "Underlay configuration of exit leaves" on page 86.

We already discussed the constraints and choices for picking an underlay routing protocol. In the examples presented in this chapter, eBGP is the underlay routing protocol. More specifically, the use of unnumbered BGP leads to a trivial routing and interface configuration. If you're interested in knowing more about the use of BGP in the data center and unnumbered BGP, refer to the companion book, *BGP in the Data Center* (O'Reilly, 2017).

The interface configuration is as simple as this:

- Assign the node's unique IP address to the loopback device.
- Set the MTU on all interswitch links to carry jumbo frames, 9,216 bytes in size.

A sample Linux interfaces configuration for `leaf01` looks as follows:

```
auto all
iface lo inet loopback
  address 10.0.0.11/32
iface swp51
```

```
    mtu 9216
iface swp52
    mtu 9216
```

Here is the routing configuration using unnumbered BGP with FRR:

- Assign the router's ASN following principles described in the companion book, BGP in the Data Center.
- Assign the loopback IP address as the router ID.
- Define a peer-group to provide a template for all the neighbors. In the case of spines, the neighbors are all the leaves, including exit leaves. In the case of leaves, the neighbors are the spines.
- Define individual neighbor connections.
- Activate the advertisement of the IPv4 unicast address.
- Advertise locally connected routes, but only for the loopback device.

In our minimal network, each spine is connected to two exit leaves and four regular leaves, so our routing in "The Invariants: Configuration for the Spines, Firewall, and Servers" on page 78 lists six neighbors.

Underlay configuration of exit leaves

Exit leaves have a more complex configuration than the rest of the leaves. This is especially true when firewalls and load balancers are connected to them. I've assumed that the firewall and other services are attached to an exit leaf as a routed hop. The other model is as a bridged hop, also called a transparent firewall. The routed hop model is more popular and more appropriate for the use case we're discussing.

Figure 6-2 shows the logical traffic flow when the exit leaves function in this fashion. Every VRF that needs to reach destinations outside of itself is connected to the firewall via its own logical interface. On a firewall, traffic leaving a VRF will come in on that VRF's interface and, if permitted, go out the interface of the destination VRF back to the exit leaf. Traffic is routed to the firewall by placing the default route out of the VRF on the appropriate subinterface to the firewall. In our case, we have three VRFs: the default, the `evpn-vrf`, and the `internet-vrf`.

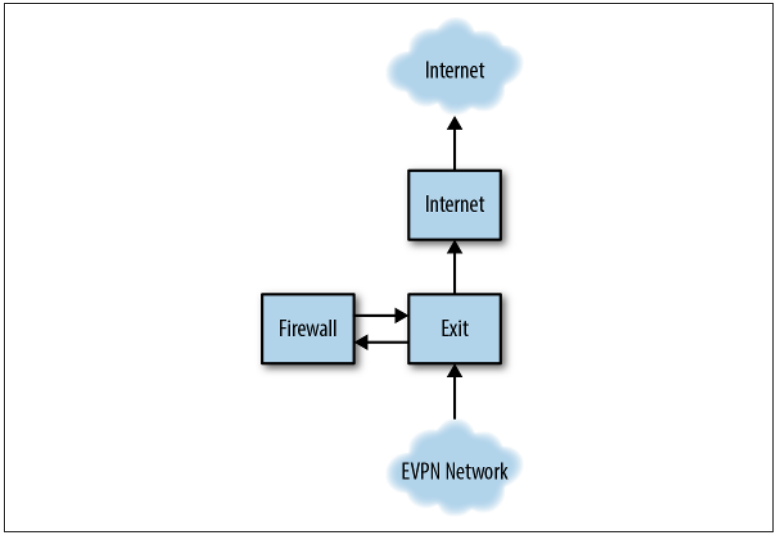

Figure 6-2. Traffic flow through exit leaves with a firewall

Therefore, the firewall is connected to each exit leaf via three logical interfaces. A VLAN subinterface is the most common logical interface used between the firewall and an exit leaf.

When the exit leaf is also a VTEP, due to a limitation in some of the switching silicon, packets coming out of a VXLAN tunnel cannot be routed without either putting the packet in another VXLAN tunnel or sending it out a bridged port. So, we define a bridge with a set of VLANs and SVIs, one for each VRF required on the exit leaf.

The interface configuration on the exit leaves consists of the following:

- Assigning the IP address to the loopback for basic reachability in the underlay
- Configuring VRFs, at least for the internet VRF, assuming the underlay is in the default VRF

A snippet of Linux interfaces configuration for these two looks as follows:

```
auto all
iface lo inet loopback
   address 10.0.0.41/32
iface internet-vrf
    vrf-table auto
```

```
iface swp44
    vrf internet-vrf
```

The BGP routing configuration consists of two sections, one for the default VRF and the other for the internet VRF.[3] The former is identified by the section "router bgp 65041," and the latter by the section "router bgp 65041 vrf internet-vrf."

The default VRF configuration is the same as that of non-exit leaves, except that there is an additional neighbor, the firewall. This default VRF will also be where we configure the overlay, but more on that later. The internet VRF configuration contains two neighbors: one to the internet-facing router, and the other to the firewall.

One important piece to discuss is the use of a new BGP option, `allowas-in 1`. Recall that there are multiple connections between the firewall and an exit leaf, one in each VRF. However, the firewall itself is VRF-unaware and merely sees multiple BGP sessions. When the firewall reflects back the routes learned from a neighbor in one VRF to the neighbor in the other VRF, the exit leaf's BGP rejects these routes due to BGP's ASPATH loop detection. To understand this better, consider the ASPATH on a route received by the firewall. Let's say the ASPATH for this route is *<65041, 65020, 65011>*.[4] When the firewall reflects this route back to the exit leaf via the session on another subinterface, the ASPATH will be *<65053, 65041, 65020, 65011>*. Because `exit01`'s ASN (65041) is already present in this ASPATH, `exit01` treats this as an indication of a routing loop and drops the update. If you enable debugging on `exit01`, you'll see the following message in the logs, for example:

```
2018-04-13T06:19:04.101100+00:00 exit01 bgpd[4112]: swp1.3 rcvd UPDATE about
10.0.0.12/32 -- DENIED due to: as-path contains our own AS;
2018-04-13T06:19:04.101380+00:00 exit01 bgpd[4112]: swp1.3 rcvd UPDATE w/ attr: ,
origin ?, mp_nexthop
fe80::4638:39ff:fe00:4a(fe80::4638:39ff:fe00:4a)(fe80::4638:39ff:fe00:4a), path
65530 65041 65020 65013
```

The option `allowas-in 1` tells BGP to ignore a single occurrence of its own ASN in the ASPATH in the ASPATH loop detection. Because we want this specific configuration only with the firewall

3 The third section is for the overlay, and we discuss that later.

4 See the ASNs in the configuration to understand that this is a route advertised by `leaf01`.

peering, we use this specifically only with the firewall sessions and not against the peer-group as a whole.

Configuring the Overlay: FRR

Like configuring the underlay, configuring the overlay has two parts: the interface-specific part and the EVPN part. In FRR, assuming sane defaults, the EVPN configuration looks incredibly simple, especially when compared to other routing stack implementations. For everything but Route Type 5 routes (RT-5), here is the entire configuration for symmetric and asymmetric distributed routing configuration for leaves, both exit and regular:

```
address-family l2vpn evpn
   neighbor fabric activate
   advertise-all-vni
```

The `advertise-all-vni` keyword tells BGP to advertise the locally attached VNIs, their MACs, and their ARP/ND entries (if ARP/ND suppression is enabled). As discussed in Chapter 2, all the Route Distinguisher (RD) and Route Target (RT) stuff in most other routing suites' configurations is unnecessary in the case of FRR.

The symmetric routing configuration adds an additional interface section in FRR to map the VRF name to the L3 VNI. From our configuration quoted in the previous section, here is that section:

```
vrf evpn-vrf
   vni 104001
```

In case of centralized routing, the entire EVPN configuration on the exit leaf (or any other leaf that performs the task of the centralized router) looks as follows:

```
address-family l2vpn evpn
   neighbor fabric activate
   advertise-all-vni
   advertise-default-gw
```

The keyword `advertise-default-gw` advertises the router's MAC and IP as a RT-2 advertisement along with the Default Gateway extended community, as described in Chapter 5. In the case of central routing, the exit leaf will also have all the VNIs that need routing instantiated as locally attached VLANs/VNIs even though there might be no endpoints other than the SVI in those networks.

Announcing RT-5 routes is a little less intuitive. In FRR, IPv4 prefix routes in a VRF are typically announced via the BGP instance associated with that VRF. So, we configure RT-5 advertisements also via the BGP instance associated with the VRF. Because this is a signal to the EVPN machinery, as well, the configuration is under the EVPN AFI/SAFI. Here is that configuration, extracted from the configuration of `exit01` in the previous section:

```
router bgp 65041 vrf evpn-vrf
…
    address-family l2vpn evpn
        advertise ipv4 unicast
    !
```

This section indicates that BGP must announce the routes known to this VRF as EVPN RT-5 routes in the BGP instance configured in the default VRF section.

To help understand this better, let's provide some context. The BGP configuration model for a VRF has two possible modes of configuration. In the first, it follows the MPLS/L3VPN model, where the core is VPN-unaware and all VRF configuration is in a single BGP section. In the second model, the configuration follows the non-L3VPN, VRF only model, where there's an explicit and separate section for every VRF that uses BGP. Because FRR doesn't support MPLS/L3VPN as of this writing (version 4.0.1), its only option was to follow the latter model.

The spines are part of the underlay and need just to receive, process, and advertise EVPN routes without installing them in the forwarding tables. For them, therefore, the only configuration required is to activate the EVPN AFI/SAFI. This configuration, in its entirety, is as follows:

```
address-family l2vpn evpn
   neighbor fabric activate
```

Configuring the Overlay: Interfaces

Configuring VXLAN on the leaves consists of the following parts:

- Assign the VTEP IP address. This is the same as the loopback IP address if the hosts are singly attached; otherwise, it is the shared VTEP IP address with its peer.
- Define the VLANs connected to the server ports.

- Define the VXLANs that map to these VLANs.
- Define an SVI for each of the VLANs. Because we're using distributed routing, every leaf hosting a VLAN/VNI uses the same gateway IP address and MAC address.
- If the leaf is connected to dual-attached servers, assign a common VTEP IP address to this leaf and its peer. The IP address is attached to the loopback interface. Set this IP address as the VTEP IP address.
- Create as many VRFs as required. If you're using symmetric routing, create a VNI/VLAN pair for each VRF.
- Assign the SVIs to the relevant VRFs.

A snippet of the Linux interfaces configuration for these functions looks like this:

```
auto all
iface lo inet loopback
    address 10.0.0.0.11/32
    clagd-vxlan-anycast-ip 10.0.0.112
# vlan3 is the SVI for VLAN 3
iface vlan3
    address 10.1.3.11/24
    address-virtual 44:39:39:ff:00:13 10.1.3.1/24
    vlan-id 3
    vlan-raw-device bridge
    vrf vrf1
# This is the definition of VNI 3 corresponding to VLAN 3
iface vni3
  mtu 9000
  vxlan-id 3
  vxlan-local-tunnelip 10.0.0.11
  bridge-access 3
  bridge-learning off
iface bridge
  bridge-vlan-aware yes
  # bridge-ports includes all ports related to VxLAN and CLAG.
  bridge-ports bond01 bond02 peerlink vni3 vni4
  bridge-vids 3-4
```

In case of symmetric routing, creating the VRF and the VLAN/VNI pair looks as follows:

```
iface evpn-vrf
   vrf-table auto

iface vlan4001
  hwaddress 44:39:39:FF:40:94
```

```
    vlan-id 4001
    vlan-raw-device bridge
    vrf evpn-vrf

iface vxlan4001
    vxlan-id 4001
    vxlan-local-tunnelip 10.0.0.11
    bridge-learning off
    bridge-access 4001
```

Also add `vxlan4001` to the list of bridge-ports under the `iface bridge` paragraph.

ARP/ND suppression is not required if you're routing at a VTEP. In case of centralized routing, ARP/ND suppression is enabled per VNI. The snippet for a single VNI looks as follows:

```
# This is the definition of VNI 3 corresponding to VLAN 3
iface vni3
    mtu 9000
    vxlan-id 3
    vxlan-local-tunnelip 10.0.0.11
    bridge-access 3
    bridge-learning off
    bridge-arp-nd-suppress on
```

Examining an EVPN Network

We've discussed the configuration of EVPN networks so far. Let us now turn to looking at some useful commands to examine the running state of a router. You can run all FRR commands from within the FRR shell, invoked by `sudo vtysh` or by executing each individual command with `sudo vtysh -c "`*`command`*`"` where *`command`* is the command. The benefit of executing commands outside the `vtysh` shell is that you can then use the full war chest of Linux tools to make sense of the information.

Show Running Configuration

The first useful command is to show the running configuration via the command `show run bgp`. Following are some key points to look for in this output:

- Is the ASN correct?
- Is the `router-id` correct?
- Is the underlay VTEP IP address advertised?

- Is EVPN address family activated for advertisements?
- If this is a leaf, is the `advertise-all-vni` keyword present?
- If this is a spine, is the `advertise-all-vni` keyword missing?
- If this is an exit leaf with centralized routing, is the keyword `advertise-default-gw` present?
- If you're advertising default route, is `ipv4 unicast` enabled under `address-family l2vpn`?
- If this is symmetric mode, are the VRF, the L3 VNI, and its corresponding VXLAN defined?

Show BGP Summary

The next useful command is `show bgp ipv4 unicast summary`. This lists all the neighbors in the underlay. Here are some key points to look for in this output:

- If this is a leaf, do all spines show up in the output?
- If this is a spine, do all leaves show up in the output?
- For each neighbor shown, is the State/PfxRcd field showing a non-zero count?

Thanks to FRR's use of the hostname BGP option, the neighbors are listed not merely by interface name or IP address, but also by hostname.

The command `show bgp l2vpn evpn summary` lists all the neighbors in the overlay. Look for similar key points as listed for the underlay. It is correct for the firewall and internet-facing router to not show up as a neighbor for EVPN.

Show EVPN VNIs and VTEPs

Next, let us verify that we can see all the VNIs and the number of VTEPs associated with each of the VNIs. This is done via the command `show evpn vni`. Some key points to look for in this output include the following:

- Are all the VNIs present? You can use the command `show evpn` to get the counts of L2 and L3 VNIs.

- Are the VNIs in the relevant VRFs?
- Are the number of VTEPs associated with each VNI correct? L3 VNIs will not have any VTEPs.

Identify Which VTEP Advertised a MAC Address

Use the command `show evpn mac vni <vnid> mac <macaddr>` to identify which VTEP advertised a MAC address. On a Linux system, you can also run `bridge fdb show | grep <macaddr>` to identify the remote VTEP for the MAC specified by *<macaddr>*.

Comparing FRR and Cisco EVPN Configurations

As an aid in case you cannot use FRR but want to understand how to map the FRR configuration to Cisco's, I've attached some mappings of FRR to Cisco's NXOS recommended equivalents in Figure 6-3.

```
neighbor swp1 interface remote-as external      ! Leaf config. Spine config is different
                                                ! Underlay neighbor configuration to get
                                                ! loopback addr of neighbor
                                                neighbor 169.254.254.1 remote-as 65020
                                                    address-family ipv4 unicast
                                                      allowas-in
                                                      disable-peer-as-check
                                                ! overlay neighbor configuration
                                                neighbor 10.0.0.11 remote-as 65020
                                                   update-source loopback0
                                                   ebgp-multihop 3
                                                     allowas-in 1
                                                     send-community extended
                                                   address-family l2vpn
                                                     allowas-in 1
                                                     send-community extended
                                    ____________

address-family l2vpn evpn                       evpn
  neighbor swp1 activate                          ! The following VNI section has to
  advertise-all-vni                               ! be repeated for all configured VNIs
                                                  vni 3 l2
                                                    rd auto
                                                    route-target import auto
                                                    route-target export auto
                                    ____________

vrf evpn-vrf                                    vrf context evpn-vrf
  vni 104001                                      vni 104001
                                                  rd auto
                                                    address-family ipv4 unicast
                                                      route-target import 65535:101 evpn
                                                      route-target export 65535:101 evpn
                                                      route-target import 65535:101
                                                      route-target export 65535:101
```

Figure 6-3. Comparing FRR and Cisco configuration snippets

Considerations for Deploying EVPN in Large Networks

As we've learned, except for RT-5, routes in an EVPN network are largely /32 routes. In a two-tier Clos network, with about 100,000 to 120,000[5] /32 entries, this doesn't seem so bad. However, as networks become larger and we move into three-tier Clos networks, the lack of scale begins to affect the deployment. The lack of scale affects not just how big the forwarding tables become, but also the number of nodes to which ingress replication needs to replicate and the total number of VNIs in the system. All this negatively affects the robustness of the entire system. If a systemic failure can take down the entire network or render it less adaptable than you need it to be, you must reconsider the design. For these reasons, I don't think a two-tier Clos with more than 128 leaves can be robust. For scales beyond this, I highly recommend a three-tier Clos design.

A three-tier Clos network that provides merely L3 connectivity cannot be directly adapted to an EVPN network. The primary issue is the interaction between the connections of pods to the interpod spine layer and VTEP behavior. Figure 6-4 illustrates a traditional pod-based three-tier Clos design.

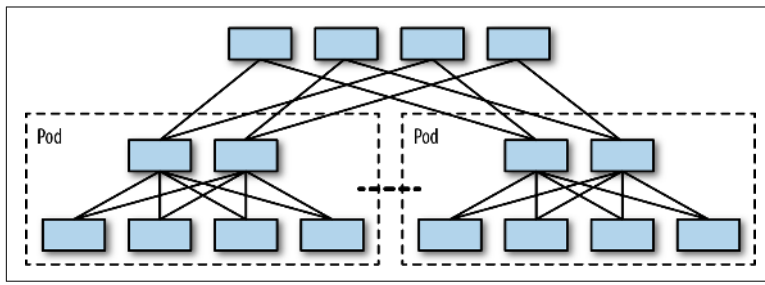

Figure 6-4. Example of a traditional pod-based three-tier Clos topology

To scale, routers in large networks typically reduce the number of routes they advertise by bundling multiple hosts by their high-order bits; this procedure is called *summarization*. With EVPN, the primary question is where do we summarize? We cannot summarize at

5 This number is more my comfort zone for a single-failure domain in enterprises rather than based on a hardware or software limitation; it is not vendor specific.

the spine layer within each pod.[6] In a pure L3 Clos, each rack already summarizes all locally attached subnets, because a subnet can be attached only to a single leaf (or pair of leaves). Making the interpod spine the point of summarization defeats the primary design of a Clos network which is to scale out the network, to spread the load to the edges, not suck it up into the center of the network. The scale and complexity of design and functionality of the interpod spines becomes quite large. Furthermore, providing services such as firewall for traffic in and out of pods becomes difficult in this design.

To this end, I think adding pod exit leaves and using these exit leaves as the points of summarization and locations to hook up pod-level services such as firewalls makes more sense. Each pod is now quite self-contained, as shown in Figure 6-5.

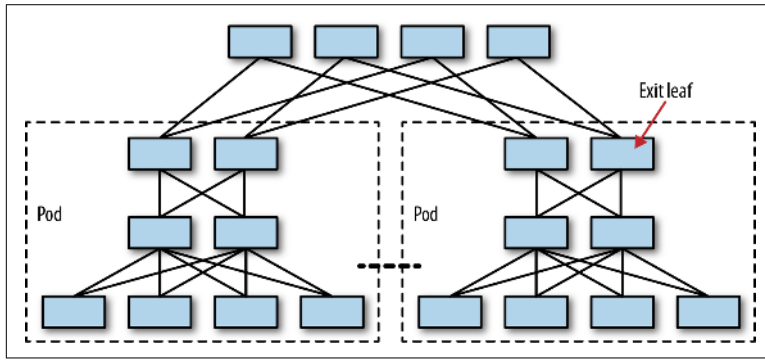

Figure 6-5. Sample of an EVPN-friendly three-tier Clos design

Let me first say that if you might think this is a four-tier Clos network, it is not. I've just switched the position of the exit leaves from being next to the other leaves to being on top, mostly to fit onto the screen.

In this network, the exit leaves primarily summarize the prefix routes specific to the pod, advertise it to the interpod spines, and advertise a default route to the pod. The exit leaves receive prefix routes of the other pods from the interpod spine.

If certain VNIs are stretched across the pods, the leaves within a pod can replicate just to the exit leaves, which can then replicate to the

6 See the companion volume, *BGP in the Data Center*, for details.

exit leaves of the relevant pods carrying that VNI, and the exit leaves can replicate internally to the leaves in their pod. This achieves good scaling but requires the switching silicon to handle the model of switching VXLAN tunnels. Those versed in the art will realize this replication model resembles H-VPLSs. You're not wrong, and so it requires the switching silicon to implement some form of hierarchical split-horizon checking to ensure that there are no loops created by switching tunnels. As far as I know, most existing merchant silicon cannot handle this correctly. But I suspect we'll see this functionality soon enough.

My recommendation is to avoid stretched VNIs to the extent possible.

Summary

We learned how to configure and administer EVPN networks in this chapter. Specifically, I hope the radical simplicity of FRR's implementation redundant sets the standard for how to configure EVPN networks and that other routing suites adopt the same or a similar model. Human error is one of the two biggest causes of network failure. Automation, if not backed by the right tooling, might only amplify the effect of human errors. So, having a simple configuration—so simple that the entire configuration can be eyeballed for errors—goes a long way toward making a robust network. There's a lot more that can be written about this specific topic, especially troubleshooting, but space constraints prevent me from getting into this with any amount of detail. The GitHub repository hopefully provides fertile ground for users to play with the network and understand how to manage EVPN networks.

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.