Juniper Networks EVPN Implementation for Next-Generation Data Center Architectures

Using Ethernet VPN to Address Evolving Data Center Requirements

Table of Contents

List of Figures

Executive Summary

Traditionally, data centers have used Layer 2 technologies such as Spanning Tree Protocol (STP), multichassis link aggregation group (MC-LAG), and Transparent Interconnection of Lots of Links (TRILL) for compute and storage connectivity. As the design of these data centers evolves to scale out multitenant networks, a new data center architecture is needed that decouples the underlay network from a tenant overlay network with technologies such as Virtual Extensible LAN (VXLAN). Using a Layer 3 IP-based underlay coupled with a VXLAN-EVPN overlay, data center and cloud operators can deploy much larger networks than are otherwise possible with traditional L2 Ethernet-based architectures. With overlays, endpoints (servers or virtual machines) can be placed anywhere in the network and remain connected to the same logical L2 network, enabling the virtual topology to be decoupled from the physical topology.

Introduction

The current data center network is coming under pressure due to a number of major trends[1]:

- Cloud-based resources and services are becoming an increasingly important part of the enterprise's IT strategy, requiring a high-performance network architecture that doesn't compromise security or performance.
- End users require anytime, anywhere access and high levels of responsiveness, which are becoming harder and harder to achieve with today's network architectures.

These trends are driving data center architects to reenvision the network with three key goals in mind:

- **Scalability**: Some enterprises are accommodating growth by increasing their use of cloud services, while others are deploying their own private and hybrid clouds. Service providers must grow rapidly to have sufficient capacity to meet demand. Today's networks are often too rigid and difficult to change to support the scalability needs of the large enterprise and service provider. New ways to scale the tenants are required in cloud data centers. One example of such a protocol is VXLAN, which scales the number of tenants in a cloud data center to 16 million by decoupling the tenants' state from the state of an underlying network by tunneling it over an underlay network.
- **Operational efficiency:** As enterprises expand their geographic reach, they face problems relating to physical distance between data centers and users, as well as shrinking maintenance windows due to around-the-clock operations. The new data center network must support application mobility, allowing network administrators to easily migrate applications within the data center and between data centers for business continuity, maintenance without downtime, and load balancing.
- **High performance:** End users often complain about poor response times and even outages of business-critical applications caused by bandwidth limitations and latency problems. The new data center needs technologies such as multipathing and control plane learning can optimize network traffic flows, rein in network faults, and ensure maximum utilization of bandwidth.

The key problem with today's network is that applications are tied to the physical network topology, which has a number of negative implications:

- Application scalability is hampered by the network's inability to scale.
- Applications cannot easily be moved within the data center or to other data centers.
- The rigid connection between applications and physical infrastructure makes it difficult to take advantage of cloud services.

Transforming the Data Center Network with VXLAN and EVPN

In the traditional data center, network architects use VLANs to create L2 logical networks that provide security by segregating users and applications, and they improve performance by limiting broadcast traffic.

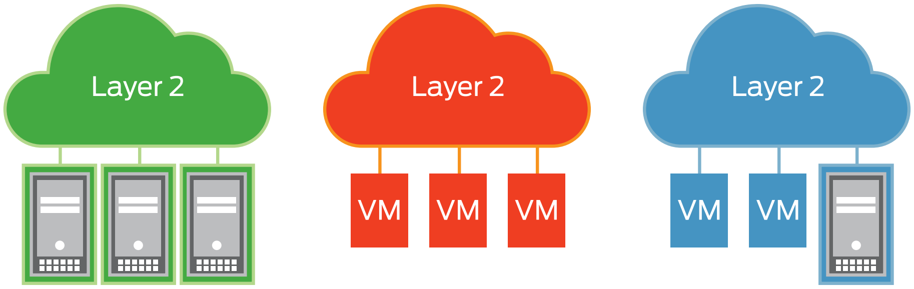


Figure 1: Layer 2 logical networks

[1] http://www.datacenterknowledge.com/archives/2014/12/22/dynamic-data-center-3-trends-driving-change-2015/

However, this architecture is difficult to scale. The VLAN specification (IEEE 802.1ad) provides a relatively small address space which results in a maximum number of 4,096 VLANs. There is a one-to-one mapping between VLANs and logical networks; therefore, the number of logical networks in the data center is also limited to 4,096. Multitenancy environments usually support a large number of users, each of whom may need multiple logical networks, so it's relatively easy to run up against this limit.

Another problem with the VLAN approach is that it constrains the movement of virtual machines (and thus the applications associated with those VMs) to the physical hardware environment hosting the VLANs. Moving an application to another location in the data center or to another data center is a cumbersome and error-prone process; in practice, most network administrators avoid doing so unless absolutely necessary.

VXLAN Overview

The VXLAN (standard IETF RFC7348) takes a major step in resolving these problems. VXLAN enables network administrators to create logical L2 networks across different L3 networks. VXLAN has a 24-bit Virtual Network ID (VNID) space, which allows for 16 million logical networks. Implemented in hardware, VXLAN supports transport of native Ethernet packets inside a tunnel encapsulation. VXLAN has become the de facto standard for overlays terminated on physical switches and is supported in Juniper Networks® QFX5100 and QFX10000 switches, EX9200 Ethernet Switches, and MX Series 3D Universal Edge Routers.

VXLAN overlays offer a number of benefits:

- Elimination of Spanning Tree Protocol (STP)
- Increased scalability
- Improved resiliency
- Fault containment

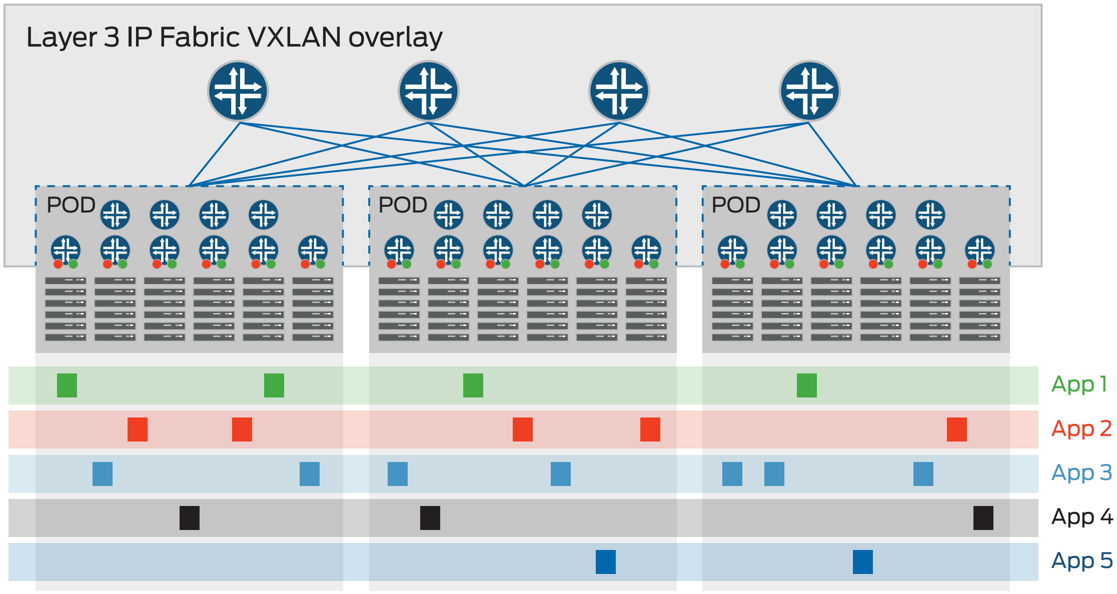


Figure 2: Application mobility across performance-optimized data centers (PODs)

Data Center Control Planes

The VXLAN abstraction does not change the flood and learn[2] behavior of the Ethernet protocol, which has inherent limitations in terms of scalability, efficiency, and utilization.

VXLAN can be deployed as a tunneling protocol across an L3 Clos data center without a control plane protocol. Two primary methods exist for doing this: VXLAN with a multicast-enabled underlay, and static unicast VXLAN tunnels. While both are viable options for eliminating L2 in an underlay, and with it L2 protocols, neither solves the inherent flood-and-learn problem, and both are difficult to scale to large multitenant environments.

The solution is to introduce a control plane to minimize flooding and facilitate learning. To facilitate learning, the control plane distributes end host information to Virtual Tunnel End Points (VTEPs) in the same segment.

[2] When a switch receives a broadcast or multicast frame or a unicast frame for which it lacks the destination MAC address and port, it "floods" the frame out all of its ports except the input port. A switch "learns" an incoming frame by adding the frame's source port and MAC address to the switch's MAC address table.

Multiprotocol BGP (MP-BGP) addresses the flood and learn problem. MP-BGP allows the network to carry both L2 media access control (MAC) and L3 IP information at the same time. Having the combined set of MAC and IP information available for forwarding decisions allows optimized routing and switching. This extension that allows BGP to transport L2 MAC and L3 IP information is called Ethernet VPN (EVPN).

EVPN solves the flood and learn problem. The emergence of VXLAN as the overlay protocol of choice for IP fabrics means that EVPN can use VXLAN for transport, breaking away from the traditional MPLS transport requirement. Furthermore, because it is based on standards[3], EVPN also complements software-defined networking (SDN) with its support on different controllers.

EVPN Overview

Control-based protocols like EVPN, virtual private LAN service (VPLS), and even L2VPN solve the legacy flood-and-learn problem. However, they have predominantly been MPLS driven. Given the advent of VXLAN as an overlay protocol of choice for IP fabrics, EVPN breaks away from the traditional MPLS transport requirement by using VXLAN as the transport. The next sections of this paper delve into the advantages of EVPN in data center deployments, the differences from MPLS-based EVPN, and deployment considerations.

EVPN's advantages, include:

Improved network efficiency

- Reduced unknown-unicast flooding due to control-plane MAC learning
- Reduced Address Resolution Protocol (ARP) flooding due to MAC-to-IP binding in control plane
- Multipath traffic over multiple spine switches (VXLAN entropy)
- Multipath traffic to active/active dual-homed server
- Distributed L3 gateway: Virtual Machine Traffic Optimization (VMTO)

Fast convergence

- Faster reconvergence when link to dual-homed server fails (aliasing)
- Faster reconvergence when a VM moves

Scalability

- Very scalable BGP-based control plane

Flexibility

- Easy integration with L3VPNs and L2VPNs for Data Center Interconnect (DCI)
- BGP-based control plane that provides ability to apply fine-grained policies

EVPN is the only completely standards-based solution that offers these benefits for a data center control plane protocol.

[3] The relevant EVPN standards include RFC 4364, BGP/MPLS IP Virtual Private Networks (VPNs); RFC 4761, Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling; and RFC 7432, BGP MPLS-Based Ethernet VPN

EVPN Concepts

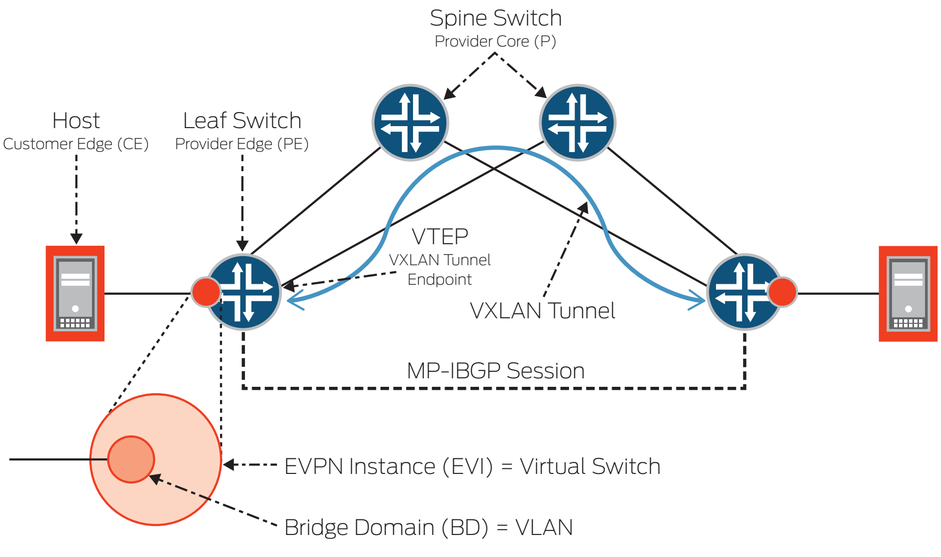


Figure 3: EVPN terminology

Figure 3 shows two leaf ("top of rack") switches in an L3 Clos topology. Between these two devices are N number of IP transport switches/routers, or "Provider Core" devices.

EVI = EVPN Instance spanning the provider edge (PE) devices participating in that EVPN.

MAC-VRF: A virtual routing and forwarding table for MAC addresses on a PE device. A unique route distinguisher (RD) is defined per MAC-VRF.

ES = Ethernet Segment. Each Ethernet Segment needs a unique identifier in an EVPN. When a customer site is connected to one or more PE devices via a set of Ethernet links, this set of Ethernet links constitutes an ES.

ESI = Ethernet Segment Identifier. For a multihomed site, each ES is identified by a unique non-zero identifier called an Ethernet Segment Identifier (ESI). In general, an Ethernet Segment should have a non-reserved ESI that is unique network wide (i.e., across all EVPN instances on all PE devices).

Attached to the leaf switches (or "provider edge"), are hosts (i.e., servers, storage, or any bare-metal device), which we refer to as customer edge (CE) devices.

Between leaf devices, we establish an MP-BGP session, which EVPN uses to distribute routes to be used in the overlay control protocol.

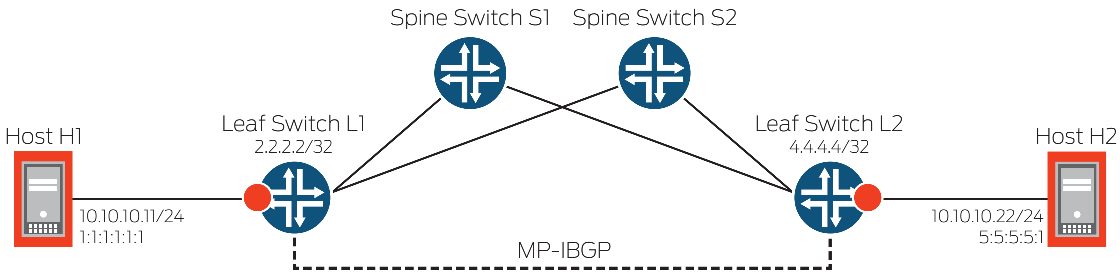
EVPN introduces the concept of Route Types. At the time of publication of this document, there are five.

- Route Type 1: Ethernet Auto-Discovery (AD) Route
  - These are advertised on a per EVI and per ESI basis. The Ethernet auto-discovery routes are required when a CE device is multihomed. When a CE device is single-homed, the ESI will be zero.
- Route Type 2: MAC/IP Advertisement Route
  - EVPN allows an end host's IP and MAC addresses to be advertised within the EVPN network layer reachability information (NLRI). This allows for control plane learning of end systems MAC addresses.
- Route Type 3: Inclusive Multicast Ethernet Tag Route
  - This route sets up a path for broadcast, unknown unicast, and multicast (BUM) traffic from a PE device to the remote PE device on a per VLAN, per ESI basis.
- Route Type 4: Ethernet Segment Route
  - ESI allows the CE device to be multihomed to two or more PE devices—in single/active or active/active mode. PE devices that are connected to the same Ethernet Segment will discover each other through the ES route.
- Route Type 5: IP Prefix Route
  - (Optional) This allows IP Prefix Route encoding for inter-subnet forwarding.

EVPN Remote MAC Learning

Once an EVPN MP-BGP session is established between two devices, there are various types of reachability information that the EVPN control plane will advertise.

The first we will discuss is an EVPN Type 2 route.



NLRI	Route Type	MAC/IP Advertisement Route (Type 2)
	Route Distinguisher (RD)	RD of red EVI on leaf switch L2
	Ethernet Segment Identifier (ESI)	0 (single homed host)
	Ethernet Tag ID	Global VXLAN VNID of bridge domain
	MAC Address	MAC address of host H2 (5:5:5:5:5:1)
	IP Address	IP address of host H2 (10.10.10.22) **
	MPLS label 1 + MPLS label 2	VNID
Next-hop		Loopback IP address of L2 (4.4.4.4)
Extended Community		Route-targets (Red)
Other attributes (Origin, AS-Path, Local-Pref, ...)		...

Figure 4: Remote learning (MAC/IP advertisement, EVPN Type 2 route)

In Figure 4, Leaf Switch L2 locally learns host H2's MAC address through traditional L2 learning. Optionally, it may also learn the IP-to-MAC binding through either Dynamic Host Configuration Protocol (DHCP) or ARP snooping.

In a traditional flood-and-learn network, Leaf Switch L1 would not learn the MAC address of H2 until either H2 has sent traffic to H1, or H1 has received BUM traffic from H2 (i.e., an ARP request). Until Leaf Switch L1 has knowledge of H2's MAC address, any traffic from H1 towards H2 will be flooded as unknown unicast across all leaf switches throughout the network in the same ES.

With EVPN, on the other hand, as soon as Leaf Switch L2 locally learns Host H2's MAC address, it immediately advertises this information via a Type 2 route, to all of its MP-BGP peers belonging to the same VXLAN VNID. This is one primary benefit of an EVPN control plane.

EVPN Server Multihoming

Server multihoming to redundant top-of-rack devices is a common requirement in data centers. Traditionally, this requirement required vendor proprietary solutions such as multichassis link aggregation (MLAG), multichassis link aggregation group (MC-LAG), Virtual Chassis port (VCP), stacking, and Virtual Chassis. While each solution has its merits, it does require the same vendor across these devices, and in the case of MLAG/MC-LAG, multihoming is limited to two PE devices.

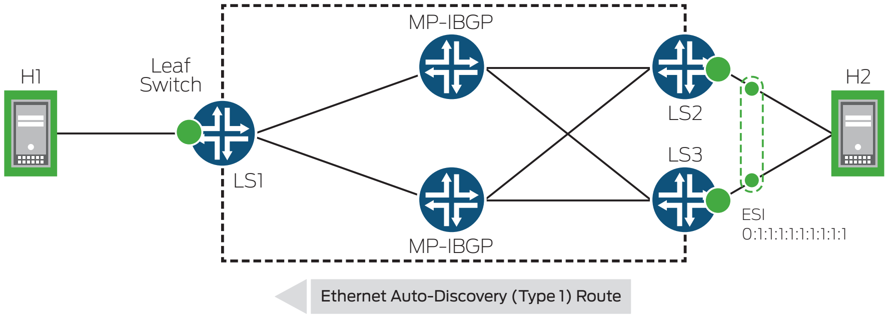
EVPN, on the other hand, is a standards-based multihoming solution, scales horizontally across any number of PE devices, and seamlessly integrates into a multivendor, L3 Clos fabric.

For EVPN server multihoming, a new type of route is required representing an ESI. This is an EVPN Type 1 route.

ETHERNET AUTO-DISCOVERY (TYPE 1) ROUTES

PER ETHERNET SEGMENT (ES): MULTIPATHING AND FAST CONVERGENCE

NLRI	Route Type	Ethernet Auto-Discovery Route (Type 1)
	Route Distinguisher (RD)	RD of EVI on leaf switch LS2 (contains IP of LS2)
	Ethernet Segment ID (ESI)	0:1:1:1:1:1:1:1:1:1
	Ethernet Tag ID	MAX-ET
	MPLS label	0
Extended communities		ESI Label Extended Community: · Single-Active Flag = false (0) · ESI Label = null
Next-hop		Loopback IP address of LS2
Other attributes (Origin, AS-Path Local-Pref, ...)		



Ethernet Auto-Discovery (Type 1) Route

Figure 5: EVPN Type 1 advertisement, ESI

In Figure 5, H2 is multihomed via a standard link aggregation group (LAG) to both LS2 and LS3 in the same L2 domain. Both LS2 and LS3 advertise direct reachability to this L2 segment, or ESI, via a Type1 route to LS1.

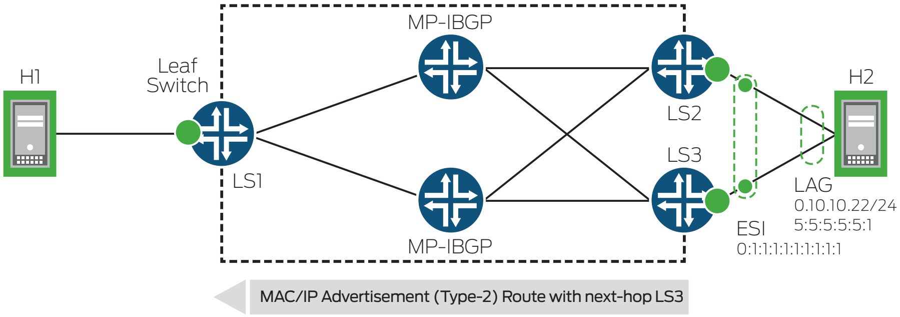
Type 1 routes do not advertise MAC address(es) learned on this ESI. For MAC reachability, a Type 2 route is required.

In the simplest case, we can assume that LS2 and LS3 have both learned H2's MAC address.

MAC / IP ADERTISEMENT (TYPE 2) ROUTES

REVISITED - FOR MULTIHOMED HOSTS

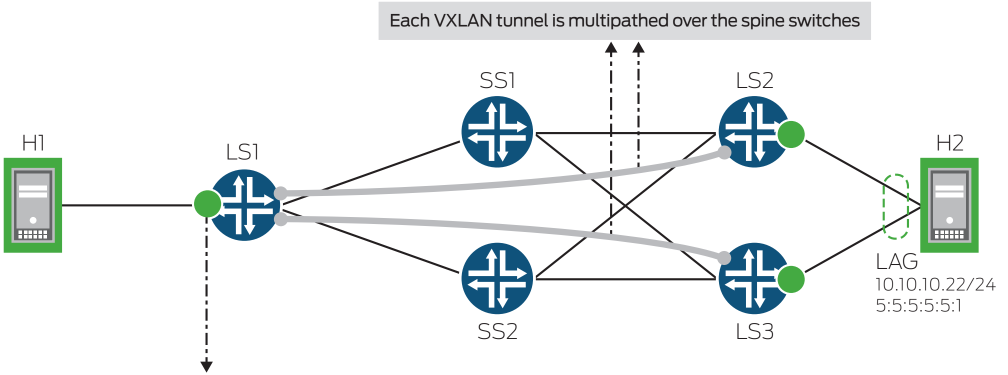
NLRI	Route Type	MAC/IP Advertisement Route (Type 2)
	Route Distinguisher (RD)	...
	Ethernet Segment Identifier (ESI)	0:1:1:1:1:1:1:1:1:1
	Ethernet Tag ID	VNID
	MAC Address	MAC address of host H2 (5:5:5:5:5:1)
	IP Address	IP address of host H2 (10.10.10.22)
	MPLS label 1 + MPLS label 2	VNID
Next-hop		Loopback of LS3
Other attributes (Origin, AS-Path, Local-Pref, ...)		...



MAC/IP Advertisement (Type-2) Route with next-hop LS3

Figure 6: EVPN Type 2 advertisement with associated ESI

In Figure 6, LS1 receives a Type 2 advertisement for H2's MAC from LS3, with associated ESI 0:1:1:1:1:1:1:1:1:1; it will similarly receive a Type 2 advertisement for H2 from LS2 on the same ESI (not shown), and therefore will know H2 is reachable via both peers.



Bridge Domain Forwarding Table	
Dest MAC	Next-hop
5:5:5:5:5:1	ECMP: VNID 1 over • ... VTEP1 • ... VTEP2

Figure 7: Multipathing from LS1 to H2 via LS2 and LS3

Figure 7 shows how LS1 will multipath via its VXLAN tunnels to both LS2 and LS3 to reach H2. A problem arises, however, when only one of the LS2/LS3 pairs has learned H2's MAC address (see Figure 8). For this scenario, we require EVPN aliasing in order to achieve multipathing (see Figure 9).

1. Host H2 sends all traffic over one LAG member to leaf switch LS2 (not to LS3)
2. Leaf switch LS3 does not learn the MAC of LH2
3. Leaf switch LS3 does not advertise a route for the MAC of H2
4. Leaf switch LS1 does not load-balance the traffic to H2 – it only sends traffic to LS2

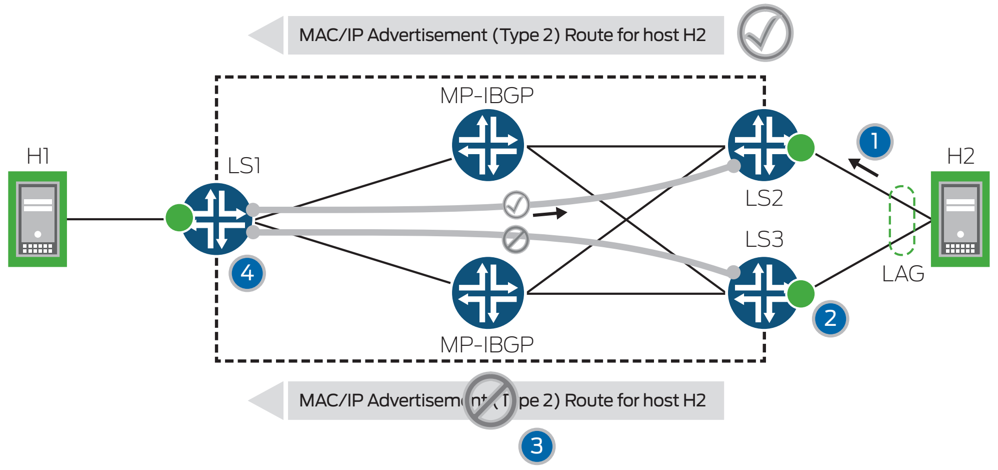

MAC/IP Advertisement (Type 2) Route for host H2
MP-IBGP
H1
LS1
LS2
1
H2
LS3
LAG
4
2
MP-IBGP
MAC/IP Advertisement (Type 2) Route for host H2
3

Figure 8: Multipathing failure from LS1 to H2 via LS2 and LS3, without aliasing

We revisit this scenario with aliasing in Figure 9 below.

1. LS1 receives Ethernet Auto-Discovery (type 1) Route from both LS2 and LS3
2. LS1 receives MAC/IP Advertisement (type 2) Route only from LS2
   LS1 knows on which ESI host H2 is located
   LS1 knows that the ESI on which H2 is located is reachable via LS2 and LS3
3. LS1 can ECMP traffic to host H2 over both VTEPs to LS2 and LS3

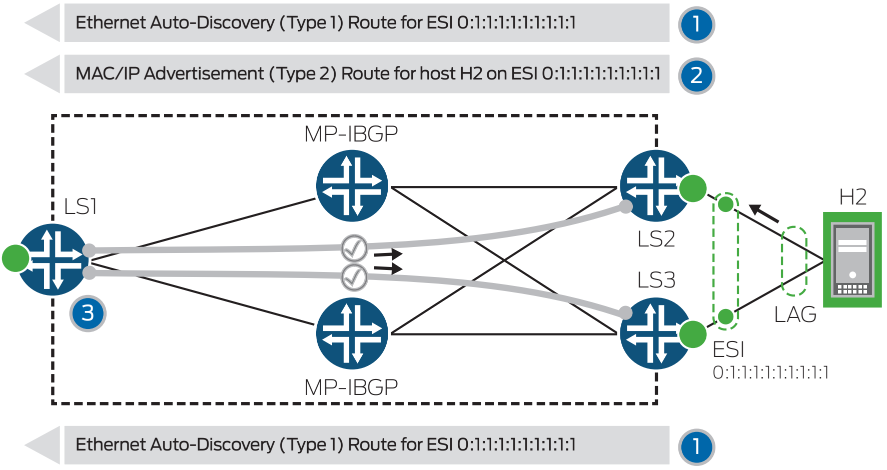

Ethernet Auto-Discovery (Type 1) Route for ESI 0:1:1:1:1:1:1:1:1:1
1
MAC/IP Advertisement (Type 2) Route for host H2 on ESI 0:1:1:1:1:1:1:1:1:1
2
MP-IBGP
H1
LS1
LS2
LS3
H2
3
LAG
ESI
0:1:1:1:1:1:1:1:1:1
MP-IBGP
Ethernet Auto-Discovery (Type 1) Route for ESI 0:1:1:1:1:1:1:1:1:1
1

Figure 9: Multipathing from LS1 to H2 via LS2 and LS3, with aliasing

Since H1 learns of H2's MAC address via a Type 2 route from LS2 with ESI 0:1:1:1:1:1:1:1:1:1, it can therefore determine that H2's MAC is also reachable via LS3 because of LS3's Type 1 advertisement of this same ESI.

EVPN Fast Convergence

In typical flood-and-learn protocols as well as legacy L2 control planes, a link failure with reachability to multiple, or hundreds, of MAC addresses can lead to slow convergence times.

Reconvergence after a failure is slow:

1. Connection between leaf switch LS3 and switch S goes down
2. Egress leaf switch LS3 withdraws routes for all hosts H2 ... H100 behind switch S
3. Ingress leaf switch LS1 removes all withdrawn routes from the forwarding table

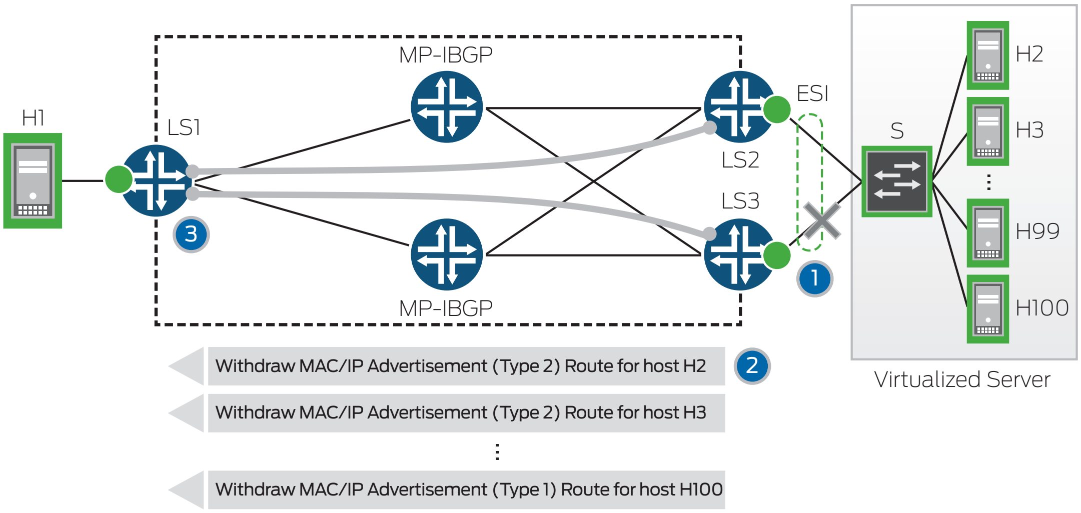

MP-IBGP
H1
LS1
3
LS2
LS3
ESI
1
S
H2
H3
⋮
H99
H100
Virtualized Server
MP-IBGP
Withdraw MAC/IP Advertisement (Type 2) Route for host H2
2
Withdraw MAC/IP Advertisement (Type 2) Route for host H3
⋮
Withdraw MAC/IP Advertisement (Type 1) Route for host H100

Figure 10: Slow convergence with individual MAC advertisements

In figure 10, we've introduced an intermediate multihomed L2 switch (S1) between LS2 and LS3 and hosts H2-H100. S1 might either be a physical switch, or more commonly might represent a virtual switch/router running on a hypervisor.

When LS3 loses its link to S1, it must withdraw 100 MAC address advertisements towards LS1. Until LS1 has received all 100 withdraws, it will continue to send traffic to these hosts towards LS3.

To solve this problem, EVPN introduces the concept of aliasing, depicted in Figure 11 below.

1. Connection between leaf switch LS3 and switch S goes down
2. Egress leaf switch LS3 withdraws route for ESI before withdrawing host routes
3. When LS1 sees that ESI is withdrawn, it removes all routes on that ESI from the FIB

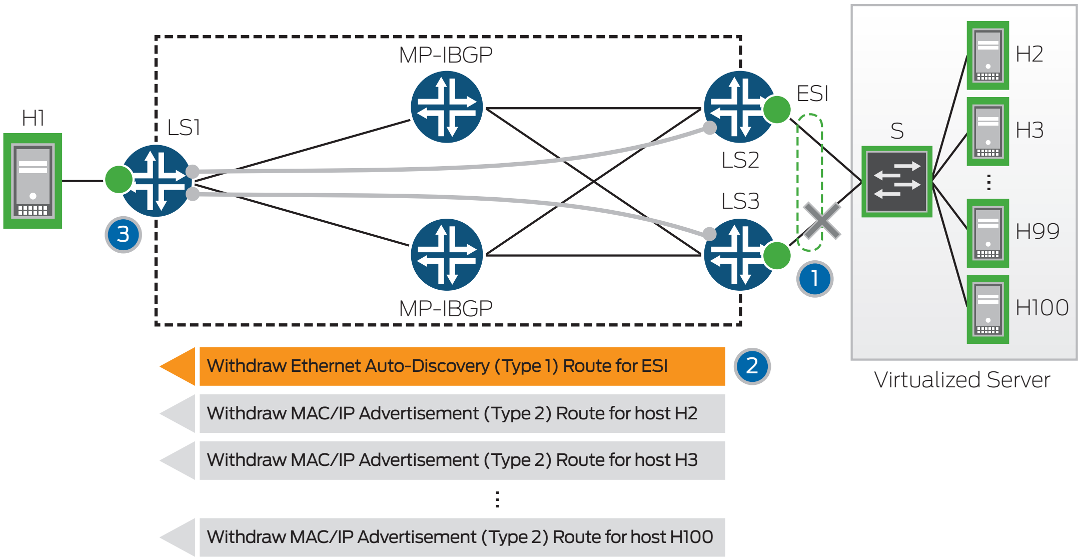

H1
LS1
3
MP-IBGP
MP-IBGP
ESI
LS2
LS3
1
S
H2
H3
⋮
H99
H100
Virtualized Server
Withdraw Ethernet Auto-Discovery (Type 1) Route for ESI
2
Withdraw MAC/IP Advertisement (Type 2) Route for host H2
Withdraw MAC/IP Advertisement (Type 2) Route for host H3
⋮
Withdraw MAC/IP Advertisement (Type 2) Route for host H100

Figure 11: Individual MAC advertisements using EVPN aliasing

With EVPN aliasing, LS3 will first withdraw its Type 1 ESI route before withdrawing 100 individual Type 2 routes.

H1, upon receiving this Type 1 withdraw, immediately purges all MAC addresses learned from LS3 on this ESI, which results in measurably improved convergence in the event of link failures.

EVPN Broadcast, Unknown Unicast and Multicast (BUM) Traffic Overview

EVPN has scalable mechanisms to forward broadcast, unknown unicast, and multicast (BUM) traffic.

Unknown unicast flooding in EVPN is mostly avoided due to BGP control plane propagation of MAC addresses. This is an inherent benefit of EVPN; however, a race condition might exist whereby a host on a PE device might send unicast traffic towards another host whose MAC address advertisement has not yet reached the PE device.

How EVPN forwards BUM traffic is a choice, and administrators have two general options. The first is to use a multicast underlay (also referred to as underlay replication). The second is to perform replication in the overlay (also referred to as ingress replication).

The differences between these two options are show in Figure 12.

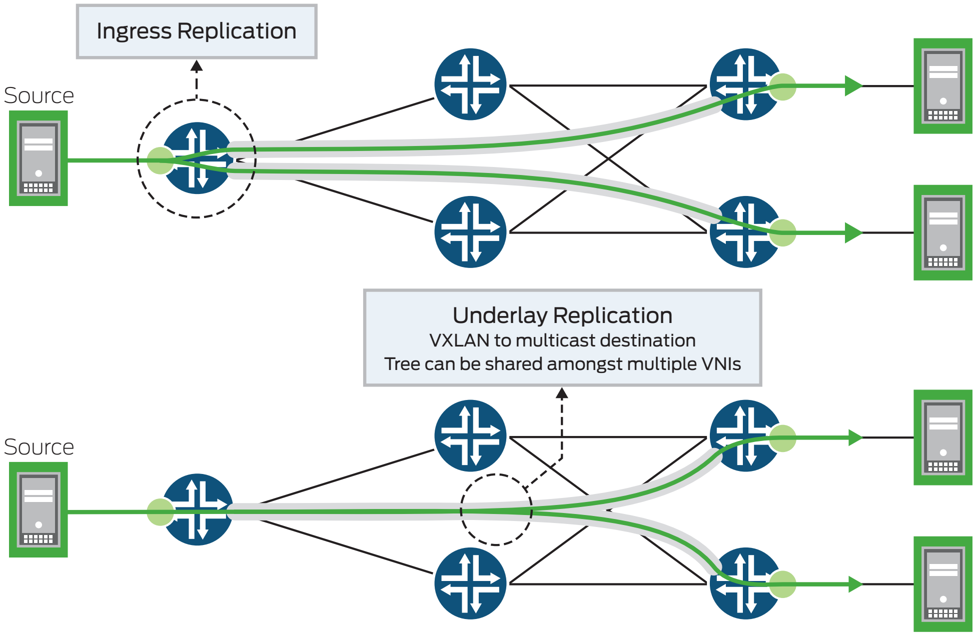

Ingress Replication
Source
Underlay Replication
VXLAN to multicast destination
Tree can be shared amongst multiple VNIs
Source

Figure 12: Ingress replication versus underlay replication

EVPN BUM Traffic—Underlay Replication

The advantage of replicating BUM traffic in the underlay lies in its efficiency. By using an underlay multicast tree, packet replication is pushed out towards the furthest point in the topology for which it is needed.

BUM traffic forwarding using an underlay can be as finely or as coarsely grained as the administrator wishes. Each VNI might have its own separate multicast group, all Virtual Network Identifiers (VNIs) might share the same multicast tree, or there could be any combination in between.

Exactly how EVPN knows to forward BUM traffic to a particular multicast group, as well as use a particular multicast protocol, is accomplished via a Type 3 route, the details of which are shown in Figure 13.

NLRI	Route Type	Inclusive Multicast Ethernet Tag Route (Type 3)	
	Route Distinguisher (RD)	...	
	Ethernet Tag ID	0	
	Originator IP Address	...	
Provider Multicast Service Interface (PMSI) Tunnel		Flags	0 (No leaf information required)
		Tunnel Type	Ingress replication, PIM-SSM, PIM-SM, BIDIR-PIM, ...
		MPLS Label	0 (Not used)
		Tunnel Identifier	Multicast Group IP Address Sender IP Address
Extended Communities		Route-Targets	
Other attributes (Origin, Nest-hop, AS-Path Local-Pref, ...)		...	

Figure 13: EVPN Type 3 route

EVPN BUM Traffic—Ingress Replication

While ingress replication is not quite as efficient as using underlay replication, the benefit comes from eliminating the complexity of managing a multicast protocol on the underlay. Due to this reason, EVPN in data center deployments primarily tend to be those that use ingress replication.

In this method, the ingress PE device makes a unicast copy of data to all egress PE devices that need to receive this data. These copies will be individually forwarded on their respective unicast VXLAN tunnel(s).

EVPN Ingress Replication—Split Horizon and Designated Forwarders

When CE devices are only connected to a single PE device, EVPN's split horizon rules will suffice for loop prevention.

If a PE device receives a BUM packet from a local CE device, it will:

- Flood to local CE devices in the same VLAN
- Flood to remote PE devices in the same VLAN
- Not flood to originating CE device

If a PE device receives a BUM packet from a remote PE device, it will:

- Flood to local CE devices in the same VLAN
- Not flood to remote PE devices.

EVPN split horizon rules are depicted in Figure 14.

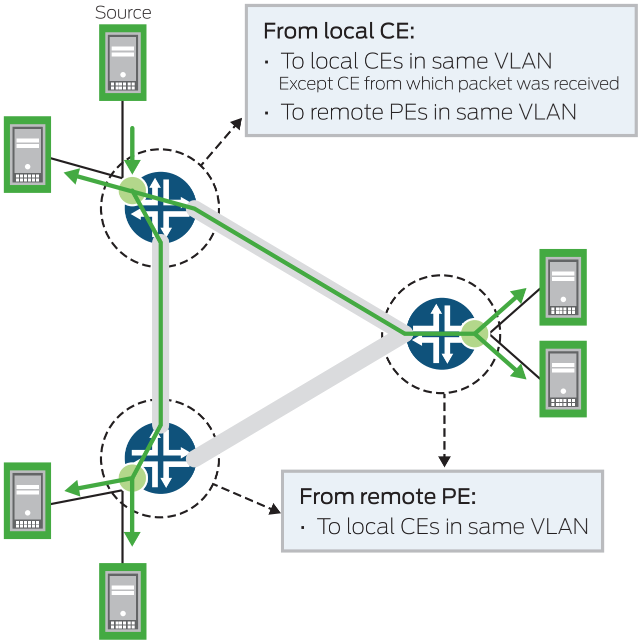

Source
From local CE:
· To local CEs in same VLAN
Except CE from which packet was received
· To remote PEs in same VLAN
From remote PE:
· To local CEs in same VLAN

Figure 14: EVPN split horizon

The concept of a designated forwarder is required when CE devices are multihomed to more than one PE device. Without a designated forwarder, multihomed hosts would receive duplicate packets.

Consider Figure 15, below.

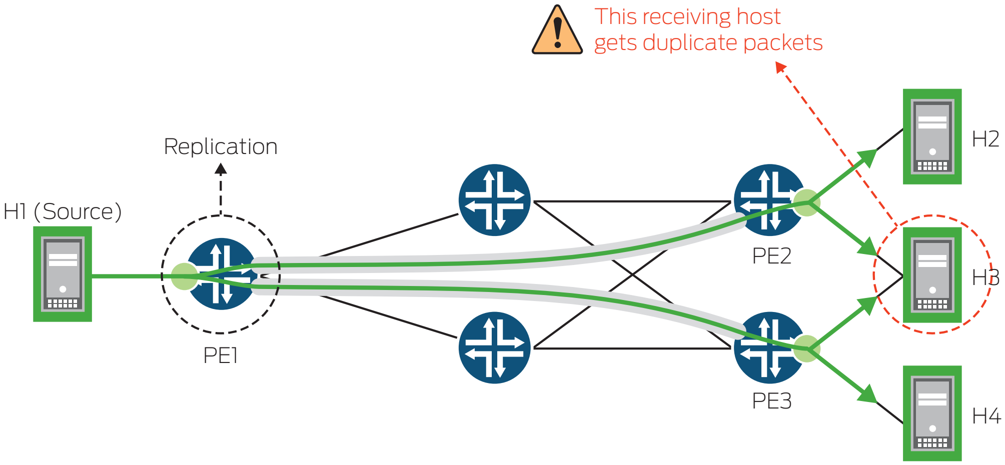


Figure 15: EVPN ingress replication and the need for a designated forwarder

If we only had split horizon rules, PE1 would replicate H1's BUM traffic on its tunnels to PE2 and PE3, which both have local hosts on the same VLAN.

PE2, following split horizon rules, would not replicate this BUM traffic back towards PE1, nor would it replicate towards PE1. It would, according to the same rules, forward to locally attached hosts H2 and H3 on the same VLAN.

Similarly, PE3 would not replicate H1's traffic back to the overlay, but would forward to its local hosts H3 and H4 on the same VLAN. This is where the problem arises, as H3 will receive duplicate traffic for H1 from both PE2 and PE3.

This introduces the requirement for a designated forwarder, shown in Figure 16.

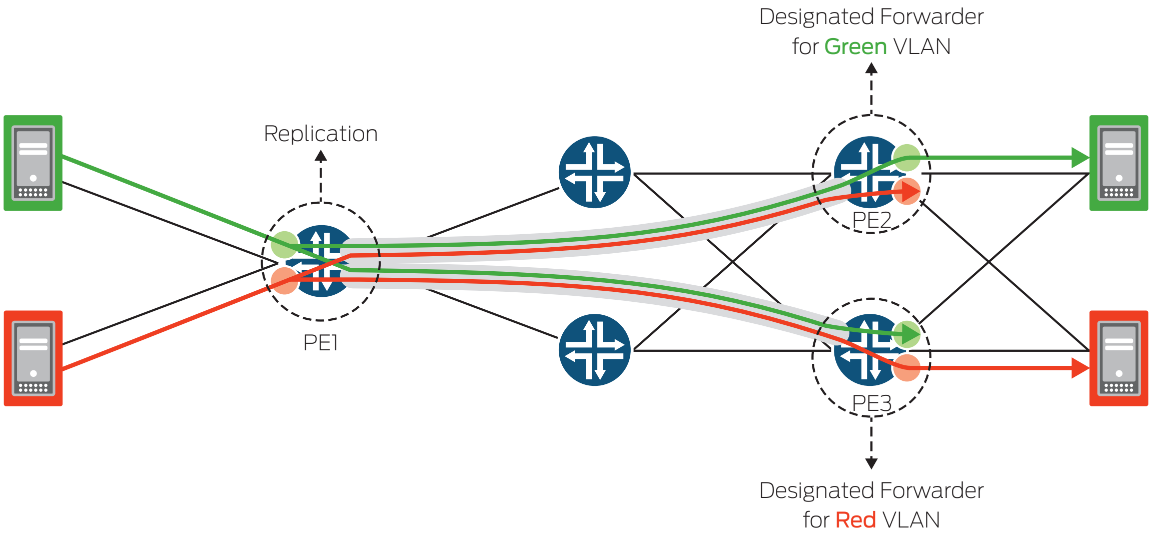
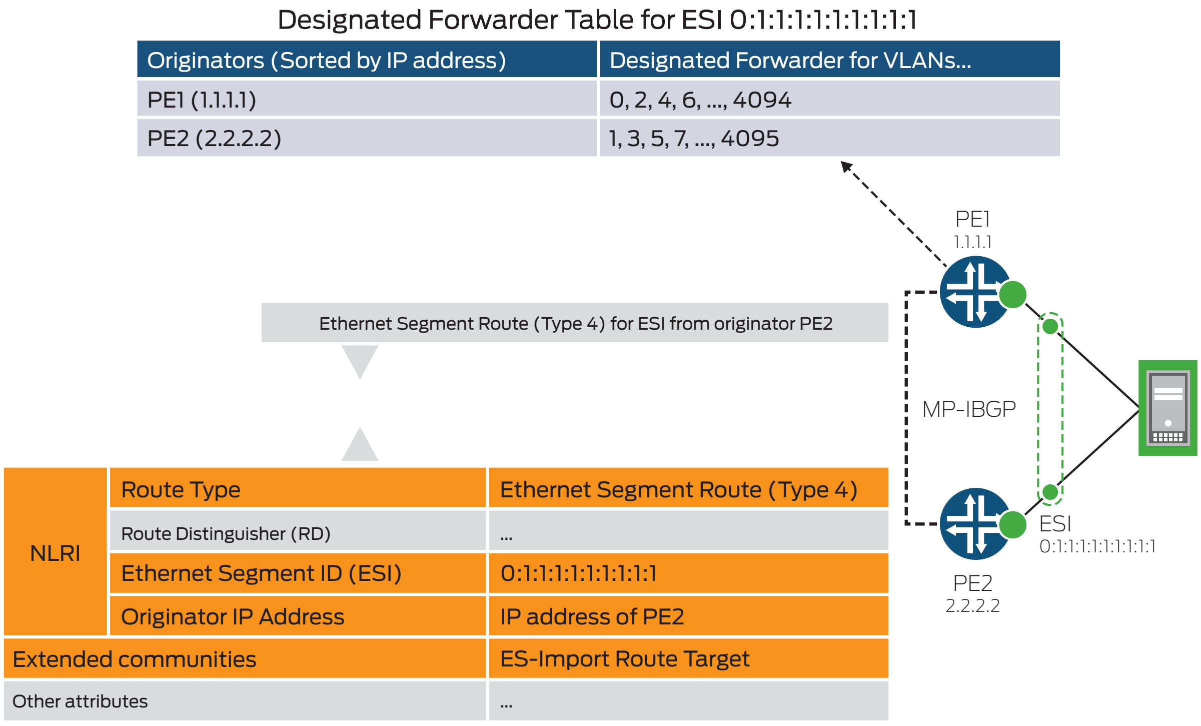


Figure 16: EVPN ingress replication and designated forwarders

PE2 becomes the designated forwarder for Green VLAN, while PE3 becomes the designated forwarder for Red VLAN. This load-balances designated forwarder duties and prevents duplicate packets from being sent to multihomed hosts.

Designated forwarders are chosen for ESI based on Type 4 route advertisements as shown in Figure 17.



Designated Forwarder Table for ESI 0:1:1:1:1:1:1:1:1:1

Originators (Sorted by IP address)	Designated Forwarder for VLANs...
PE1 (1.1.1.1)	0, 2, 4, 6, ..., 4094
PE2 (2.2.2.2)	1, 3, 5, 7, ..., 4095

Ethernet Segment Route (Type 4) for ESI from originator PE2

NLRI	Route Type	Ethernet Segment Route (Type 4)
	Route Distinguisher (RD)	...
	Ethernet Segment ID (ESI)	0:1:1:1:1:1:1:1:1:1
	Originator IP Address	IP address of PE2
Extended communities		ES-Import Route Target
Other attributes		...

Figure 17: ESI designated forwarders based on Type 4 advertisements

- When a PE device discovers the ESI of its attached Ethernet Segment, it advertises an ES route with the associated ES-Import extended community attribute to its MP-BGP peers.

The PE device then starts a timer (default value = 3 seconds) to allow the reception of Ethernet Segment routes from other PE devices, or nodes, connected to the same ES. This timer value should be the same across all PE devices connected to the same ES.

- When the timer expires, each PE device builds an ordered list of the addresses of all PE nodes, connected to the ES (including itself), in increasing numeric value. VLANs are then assigned designated forwarders, round-robin across all PE devices in numerical order, from each PE node advertising this ES, resulting in consistent agreement of designated forwarders across all PE nodes.

Even with both split horizon and designated forwarders, there exists one more case in which duplicate packets would be received by a multihomed CE device if there were not additional rules to follow. Consider Figure 18.

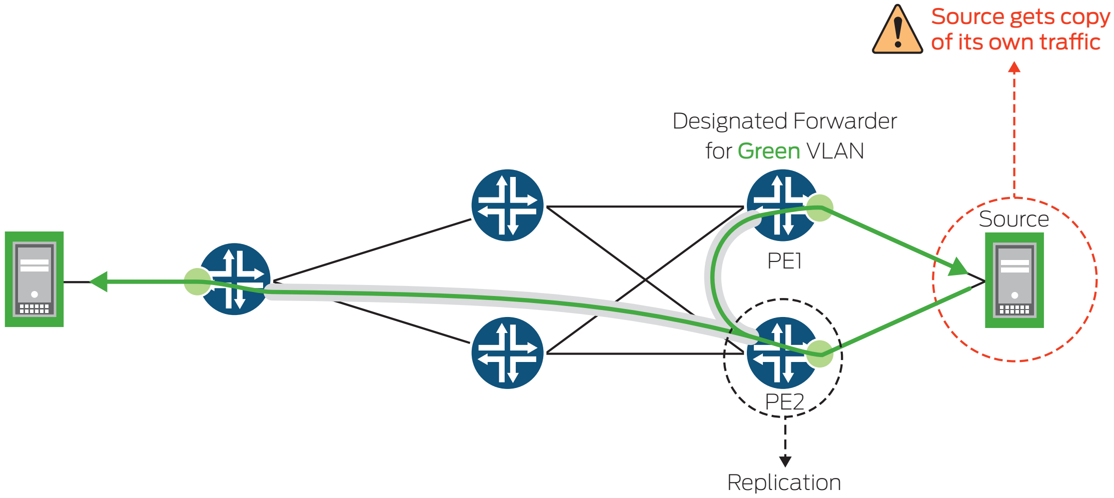


Figure 18: EVPN ingress replication—traffic looped back to source with designated forwarder

In this example, the source has sent BUM traffic out of its interface towards PE2, which is not the designated forwarder for this ES/VLAN.

Following split horizon rules, PE2 replicates towards PE1, which is the designated forwarder. Without additional rules, this designated forwarder would dutifully forward this BUM traffic back towards the source, which results in the source receiving its own traffic.

Finally, we introduce the concept of local bias, whose rules include the following:

- A PE device receiving BUM traffic from a locally attached CE device will forward to all local servers in the same Ethernet tag, and to all remote PE devices that belong to the same Ethernet tag.
- A PE device receiving BUM traffic from a peer PE device will:
  - Determine the VTEP source IP address of this BUM traffic.
  - Consult Type 4 (Ethernet Segment AD) routes from this source IP address.
  - If ESI is present in this Source IP's Type 4 routes, then it will drop this BUM traffic towards this particular ESI.
  - Otherwise it will forward to all local CE device interfaces on this same Ethernet tag.

EVPN MAC Mobility

One of the primary benefits EVPN provides, in enabling the stretch of L2 domains across an L3 fabric, is to allow for seamless migration of workloads.

In order to prevent race conditions which might exist with multiple rapid moves, EVPN introduces sequence numbering in its Type 2 routes.

Figure 19 shows a scenario in which Host H was originally attached to L1. It then moves to L2, and then again quickly moves to L4.

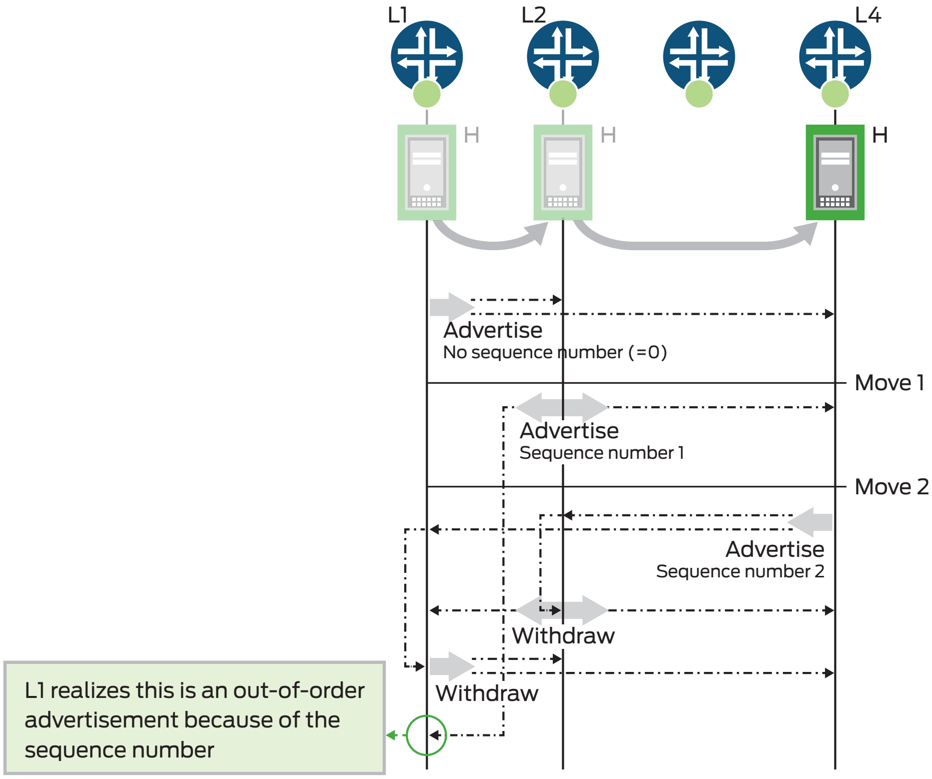


Figure 19: EVPN MAC mobility with multiple host moves

- In the original state, L1 has advertised the initial Host H Type 2 route with a sequence of zero, since Host H had not existed in the EVPN control plane previously.
- After Host H moves to L2, L2 advertises a new Host H Type 2 route with sequence 1. This advertisement reaches L4, but has not yet reached L1.
- Host H moves to L4, at which point L4 advertises a new Host H Type 2 route with sequence 2. L2 receives this updated route, realizes that Host H has moved, and sends a “withdraw” message for its sequence 1 route.
- L1 has received L4’s advertisement (sequence 2) as well as L2’s withdraw message (sequence 1). L1 sends its own withdraw message for its sequence 0 route.
- Lastly, L1 belatedly receives L2’s sequence 1 route. Since L1 has already learned a newer sequence 2 route, it can safely discard this outdated advertisement.

EVPN Distributed Default Gateways

EVPN offers a unique and scalable solution which allows for traditional IP gateways to be actively distributed across an arbitrary number of network elements. This is especially relevant in cloud environments where an L2 tenant might exist anywhere in the fabric.

Functionally, it would be desirable to place a distributed default gateway as close as possible to all hosts. This would present an architecture whereby the distributed default gateway resides on the leaf.

However, on Broadcom Trident2-powered platforms such as the QFX5100, VXLAN routing is not supported in a single hardware pass[4]. This then necessitates moving this VXLAN routing capability further up the L3 Clos tier, at the spine or the core.

Refer to the example depicted in Figure 20.

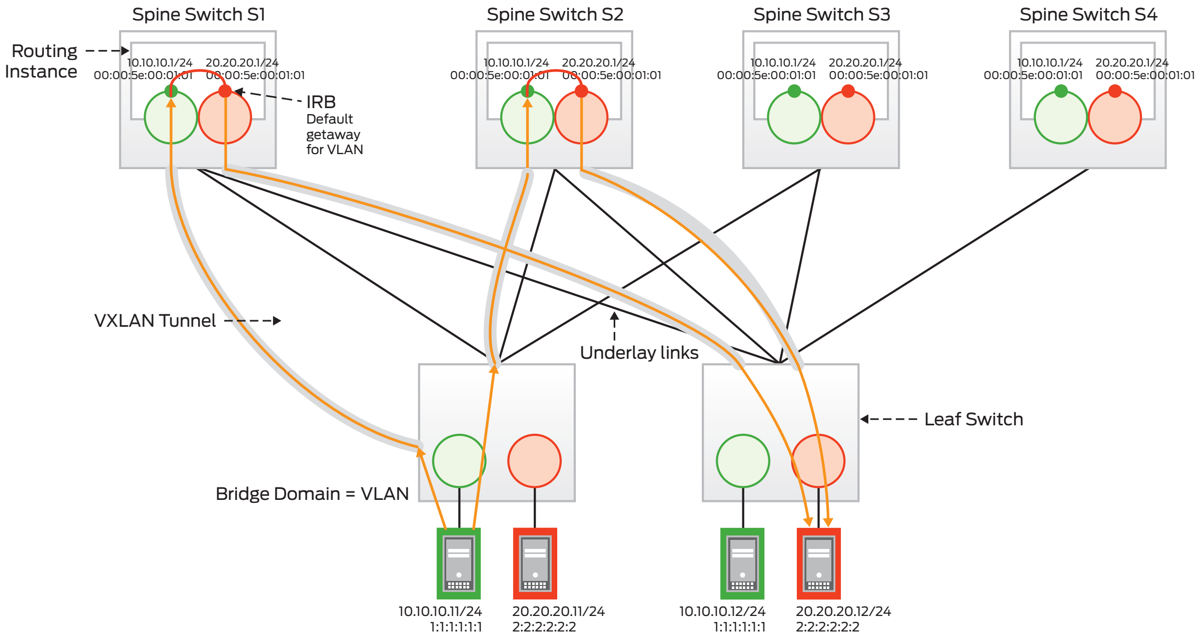


Figure 20: EVPN distributed default gateway in the spine

In this example, we have shown two leaf switches and four spines. Each spine has integrated routing and bridging (IRB) interface configured for the Green and Red VNIs. Furthermore, all spines share the same anycast IP and MAC for each IRB.

Green host 10.10.10.11 is sending two unique traffic flows to Red host 20.20.20.12 (for example, one SSH connection and another HTTPS connection). These two equal-cost multipath (ECMP) streams are load-balanced across two different anycast gateways. Additional ECMP paths will use all paths to all anycast gateways.

This active/active load balancing across an arbitrary number of anycast gateways is achieved using the same mechanism as CE device multihoming.

Conceptually, we can think of a multihomed "IRB" on every spine. Each spine will advertise this IRB's Type 1 ESI, as well as the same Type 2 MAC address. Remote PE devices will see equal-cost reachability to this same MAC and ESI across all spines. A simplified example with two spines, two IRBs, and two leaves is shown in Figure 21.

[4] Broadcom Trident2 platforms support routing with MPLS encapsulation. Thus, EVPN+MPLS encapsulation is a viable deployment on MPLS, which removes these VXLAN hardware restrictions.

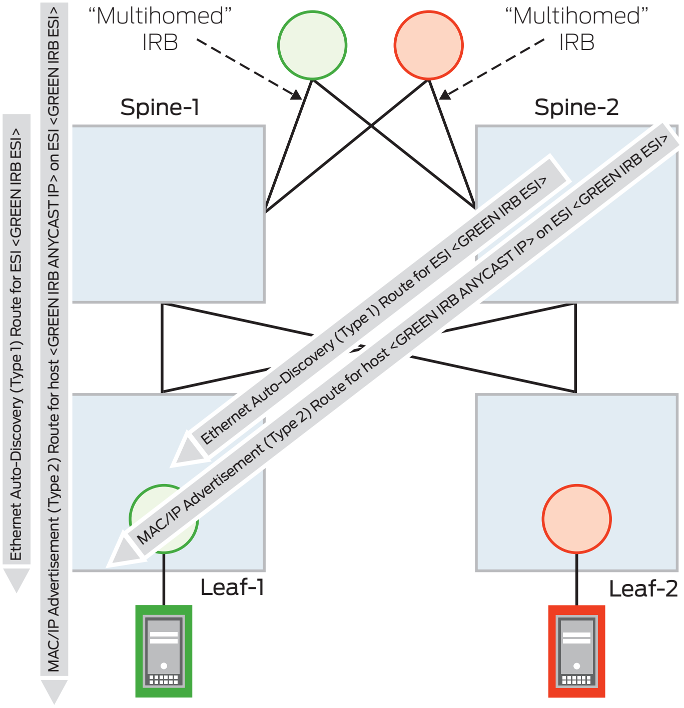

"Multihomed" IRB
"Multihomed" IRB
Spine-1
Spine-2
Leaf-1
Leaf-2
Ethernet Auto-Discovery (Type 1) Route for ESI <GREEN IRB ESI>
MAC/IP Advertisement (Type 2) Route for host <GREEN IRB ANYCAST IP> on ESI <GREEN IRB ESI>
Ethernet Auto-Discovery (Type 1) Route for ESI <GREEN IRB ESI>
MAC/IP Advertisement (Type 2) Route for host <GREEN IRB ANYCAST IP> on ESI <GREEN IRB ESI>

Figure 21: EVPN distributed default gateway route advertisements

We see Leaf-1 receiving a Type 1 ESI advertisement for the Green IRB from both Spine-1 and Spine-2. Leaf-1 will also receive a Type 2 advertisement for the Green IRB MAC from both Spine-1 and Spine-2.

Similarly, Leaf-1 will receive Type 1 and Type 2 routes for the Red IRB, and Leaf-2 will also receive these same Type 1 and Type 2 advertisements from Spine-1 and Spine-2 for both IRBs.

EVPN and VXLAN Configuration

The Underlay

Before configuring EVPN, the first consideration in designing the network involves the network underlay. While an L3 Clos fabric is the standard practice in large-scale data center deployments, there are many design choices.

Figure 22 depicts a large-scale 5-stage L3 Clos fabric, which is comprised of three different tiers. Within a POD, there are the leaf or "top-of-rack" switches, which are connected by the second tier spine layer. Multiple PODs are connected via a higher tier "core" or "fabric" layer. Although there are only three distinct tiers in the topology, this is commonly referred to as a 5-stage Clos fabric. A server in POD A that communicates with another server in POD B will traverse five network elements, thus the five stages.

Each of these layers can scale horizontally according to an organization's specific needs, and the oversubscription between tiers can also be managed according to an organization's requirements.

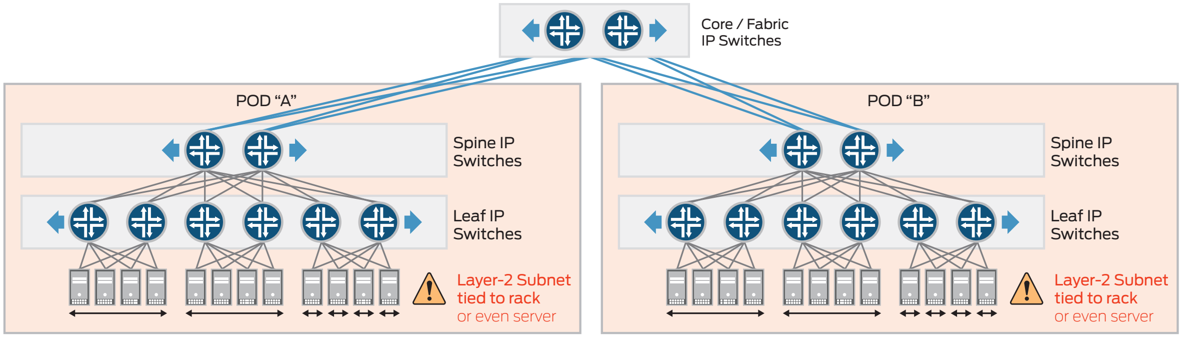


Figure 22: 5-stage L3 Clos fabric

Although the routing protocol, along with subsequent design and configurations, is an important choice when designing an L3 Clos topology, it is beyond the scope of this particular document. Smaller scale topologies might choose a Dijkstra-based protocol. Large-scale topologies, however, will typically use BGP.

Even BGP offers a number of subsequent design choices, also beyond this document's scope. Each tier might represent a single Autonomous System Number (ASN) (Figure 23), or each network device might belong in its own unique ASN (Figure 24).

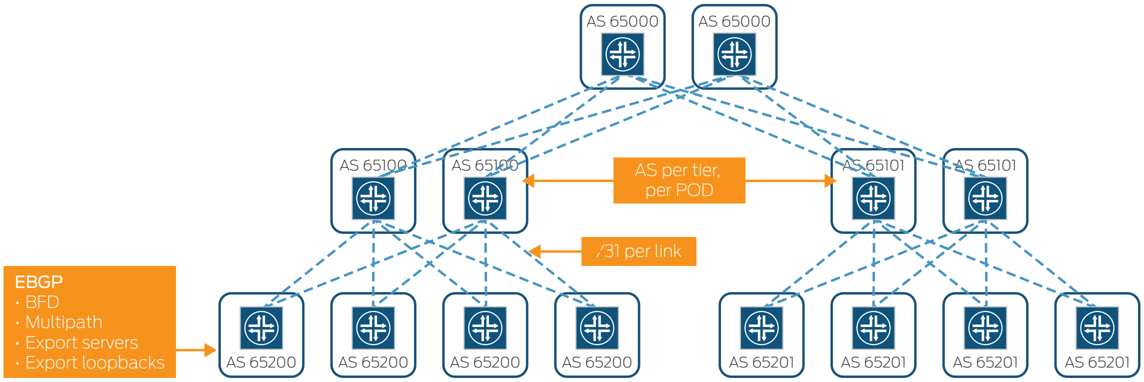


Figure 23: 5-stage L3 Clos fabric, unique ASN per tier, per POD

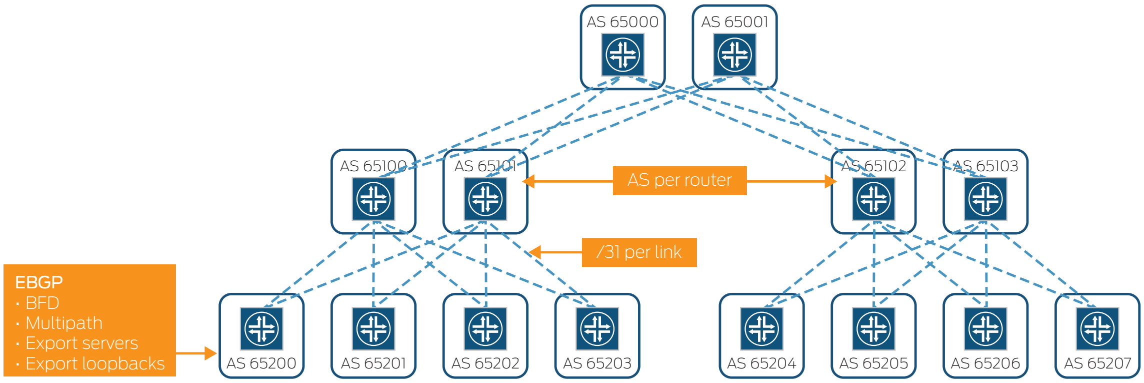


Figure 24: 5-stage L3 Clos fabric, unique ASN per device

Irrespective of the routing protocol and design chosen, the underlay must provide IP reachability of every network element's loopback address. This loopback address will be used to establish overlay BGP control plane connections, and will be the VTEP source (and destination) interfaces to build overlay VXLAN tunnels between devices.

For subsequent discussions, we will be referring to the topology depicted in Figure 25.

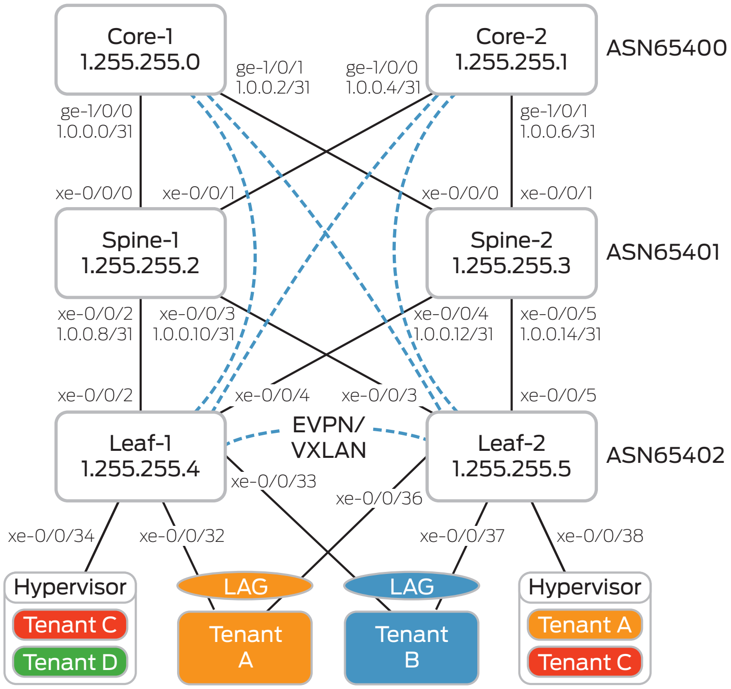


Figure 25: Example EVPN/VXLAN topology

In this example topology:

- Core-1 and Core-2 are Juniper Networks MX Series 3D Universal Edge Routers acting as IP gateways for the EVPN overlay.
- Spine-1 and Spine-2 are Juniper Networks QFX5100 switches acting as pure IP transport for the overlay.
- Leaf-1 and Leaf-2 are Juniper Networks QFX5100 switches, acting as PE devices in the EVPN topology.
- Each tier is configured with a single ASN.
- EVPN control plane connections include:
    - Leaf-1 <-> Leaf-2
    - Leaf-1 <-> Core-1
    - Leaf-1 <-> Core-2
    - Leaf-2 <-> Core-1
    - Leaf-2 <-> Core-2

First, we will step through the underlay external BGP (EBGP) configuration, starting from the leaf and working up towards the core.

Leaf-1

```
lab@leaf-1> show configuration routing-options
router-id 1.255.255.4;
autonomous-system 65402;
forwarding-table {
    export load-balance;
}

lab@leaf-1> show configuration policy-options policy-statement load-balance
term 1 {
    then {
        load-balance per-packet;
    }
}
```

The underlay BGP peering session to the spines introduces two important configurations. First, lo0 is exported and thus advertised into the underlay. Second, we configure family inet unicast loops 2. This is needed because of the design choice to reuse the same ASN within a tier.

```
lab@leaf-1> show configuration protocols bgp group underlay
type external;
advertise-peer-as;
family inet {
    unicast {
        loops 2;
    }
}
export lo0;
peer-as 65401;
multipath;
neighbor 1.0.0.8 {
    description spine-1;
}
neighbor 1.0.0.12 {
    description spine-2;
}
lab@leaf-1> show configuration policy-options policy-statement lo0
from {
    family inet;
    protocol direct;
    route-filter 0.0.0.0/0 prefix-length-range /32-/32;
}
then accept;
```

We will ignore the `advertise-peer-as` statement for now, and explain this when reviewing Spine-1's underlay configuration.

By default, a router will reject routes with its own ASN in any route updates it receives from its EBGP peer. However, Leaf-1 must learn Leaf-2's loopback in the underlay, for which the received AS_PATH will be 65401 65402.

This is shown in the output below:

```
lab@leaf-1> show route 1.255.255.5 detail table inet.0

inet.0: 13 destinations, 18 routes (13 active, 0 holddown, 0 hidden)
1.255.255.5/32 (2 entries, 1 announced)
        *BGP    Preference: 170/-101
                Next hop type: Router, Next hop index: 131070
                Address: 0x9760010
                Next-hop reference count: 84
                Source: 1.0.0.8
                Next hop: 1.0.0.8 via xe-0/0/2.0
                Session Id: 0x0
                Next hop: 1.0.0.12 via xe-0/0/4.0, selected
                Session Id: 0x0
                State: <Active Ext>
                Local AS: 65402 Peer AS: 65401
                Age: 2d 4:00:39
                Validation State: unverified
                Task: BGP_65401.1.0.0.8+61723
                Announcement bits (2): 0-KRT 2-Resolve tree 2
                AS path: 65401 65402 I (Looped: 65402)
                Accepted Multipath
                Localpref: 100
                Router ID: 1.255.255.2
         BGP    Preference: 170/-101
                Next hop type: Router, Next hop index: 1723
                Address: 0x95f8860
                Next-hop reference count: 7
                Source: 1.0.0.12
                Next hop: 1.0.0.12 via xe-0/0/4.0, selected
                Session Id: 0x0
                State: <NotBest Ext>
                Inactive reason: Not Best in its group - Active preferred
                Local AS: 65402 Peer AS: 65401
                Age: 2d 4:00:39
                Validation State: unverified
                Task: BGP_65401.1.0.0.12+60413
                AS path: 65401 65402 I (Looped: 65402)
                Accepted MultipathContrib
                Localpref: 100
                Router ID: 1.255.255.3

{master:0}
```

We observe that Leaf-1 learns Leaf-2's loopback over its EBGP sessions from Spine-1 and Spine-2. Without the loop configuration, this route would be discarded.

Spine-1

```
lab@spine-1> show configuration routing-options
router-id 1.255.255.2;
autonomous-system 65401;
forwarding-table {
    export load-balance;
    ecmp-fast-reroute;
}

lab@spine-1> show configuration policy-options policy-statement load-balance
term 1 {
    then {
        load-balance per-packet;
    }
}
```

```
lab@spine-1> show configuration protocols bgp group underlay-leaf
type external;
advertise-peer-as;
family inet {
    unicast {
        loops 2;
    }
}
export lo0;
peer-as 65402;
multipath;
neighbor 1.0.0.9 {
    description leaf-1;
}
neighbor 1.0.0.11 {
    description leaf-2;
}

lab@spine-1> show configuration protocols bgp group underlay-core
type external;
advertise-peer-as;
family inet {
    unicast {
        loops 2;
    }
}
export lo0;
peer-as 65400;
multipath;
neighbor 1.0.0.0 {
    description core-1;
}
neighbor 1.0.0.4 {
    description core-2;
}

lab@spine-1> show configuration policy-options policy-statement lo0
from {
    family inet;
    protocol direct;
    route-filter 0.0.0.0/0 prefix-length-range /32-/32;
}
then accept;
```

The configuration of underlay BGP on Spine-1 is similar to Leaf-1, but here we will explain the use of `advertise-peer-as` in `bgp group underlay-leaf`.

On Leaf-1, we needed to configure `family inet unicast loops 2` such that Leaf-1 will not reject EBGP-learned routes with its own ASN.

Spine-1 learns Leaf-2's loopback through its direct EBGP session with Leaf-2. However, EBGP rules dictate that Spine-1 should never re-advertise routes learned from a particular ASN back to that ASN. Therefore, Leaf-1, under normal configuration, would not receive Leaf-1's routes, regardless of the loops configuration.

Thus **`advertise-peer-as`**, as the astute reader might guess, allows Spine-1 to bypass this rule and re-advertise a route back to the same ASN, at which point the loops configuration is useful and needed to complete the desired route advertisement and learning behaviors.

Recall that **`advertise-peer-as`** is also configured on Leaf-1's EBGP configurations with Spine-1 and Spine-2. Strictly speaking, this is not required. Spine-1 does not need to know Spine-2's loopback and vice versa, since the spines do not participate in EVPN. From a troubleshooting and consistency perspective, the administrator might desire loopbacks to be reachable everywhere, thus the optional configuration on Leaf-1. The **`family inet unicast loops 2`** configuration on Spine-1 is therefore also optional.

Similarly, since Core-1 and Core-2 are not EVPN peered, strictly speaking, they also do not need reachability to the other's loopback. Thus, the **`advertise-peer-as`** configuration is optional on Spine-1 in `bgp group underlay-core`.

Core-1

```
lab@core-1> show configuration routing-options
router-id 1.255.255.0;
autonomous-system 65400;
forwarding-table {
    export load-balance;
    ecmp-fast-reroute;
}
lab@core-1> show configuration policy-options policy-statement load-balance
term 1 {
    then {
        load-balance per-packet;
    }
}
lab@core-1> show configuration protocols bgp
group underlay {
    type external;
    advertise-peer-as;
    family inet {
        unicast {
            loops 2;
        }
    }
    export lo0;
    peer-as 65401;
    multipath;
    neighbor 1.0.0.1 {
        description spine-1;
    }
    neighbor 1.0.0.3 {
        description spine-2;
    }
}

lab@core-1> show configuration policy-options policy-statement lo0
from {
    family inet;
    protocol direct;
    route-filter 0.0.0.0/0 prefix-length-range /32-/32;
}
then accept;
```

Similar to the spines, Core-1 does not technically need to learn Core-2's loopback. As such, `family inet unicast loops 2` is optional on Core-1.

The only additional configuration shown on Core-1 is `forwarding-table ecmp-fast-reroute`. On MX Series routers[5], this additional configuration enables fast L3 reconvergence when multiple ECMP paths exist to a particular IP prefix.

The Overlay

Now that we've established reachability to loopback addresses in the underlay, we can configure the EVPN/VXLAN overlay.

There are two main components of EVPN configuration to consider:

1. Protocols bgp

   In this section, multicast BGP (MBGP) sessions with EVPN peers are configured with EVPN signaling.

2. Switch-level EVPN configuration

   QFX5100 offers a single logical switch, while the MX Series offers the option of multiple virtual switches. EVPN configurations between the two can then differ.

   Per switch, physical or virtual, there are EVPN-specific configurations required:

   - VTEP source interface (lo0.0)
   - Route distinguisher: The RD which MBGP will use to advertise EVPN routes
   - vrf-import: Defines which route targets will be imported into the switches' EVPN table
   - vrf-export: Defines the route target which will be used to advertise EVPN routes
   - protocols evpn
     - List of VNIs which are part of this switch domain
     - BUM forwarding method (ingress replication uses EVPN BUM traffic handling)
   - Vlan-level configuration
     - VNI to VLAN mapping
     - BUM forwarding method

On the QFX5100, the first hierarchy to consider is under `switch-options.`

```
lab@leaf-1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 1.255.255.4:1;
vrf-import vrf-imp;
vrf-target target:9999:9999;
```

The `vtep-source-interface` will always be lo0.0, which the reader will recall must be reachable via the underlay routing protocol.

The `route-distinguisher` must be unique, network-wide, across all switches (physical, and in the case of MX Series routers, virtual). Within Multiprotocol MGP (MP-MGP), this ensures that all route advertisements are globally unique.

The `vrf-target` on QFX5100 will, at a minimum, be the community with which the switch will send all ESI (Type 1) routes.

[5] This is also true with Juniper Networks PTX Series Packet Transport Routers

Lastly, `vrf-import vrf-imp` defines the target community list which will be imported into the `default-switch.evpn.0` instance from `bgp.evpn.0`.

```
lab@leaf-1> show configuration policy-options policy-statement vrf-imp
term t1 {
    from community com100;
    then accept;
}
term t2 {
    from community com200;
    then accept;
}
term t3 {
    from community com300;
    then accept;
}
term t4 {
    from community com400;
    then accept;
}
term t5 {
    then reject;
}
lab@leaf-1> show configuration policy-options | grep members
community com100 members target:1:100;
community com200 members target:1:200;
community com300 members target:1:300;
community com400 members target:1:400;
```

Next, we will move on to `protocols evpn` configuration.

```
lab@leaf-1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 1100 1200 1300 1400 ];
multicast-mode ingress-replication;
vni-routing-options {
    vni 1100 {
        vrf-target export target:1:100;
    }
    vni 1200 {
        vrf-target export target:1:200;
    }
    vni 1300 {
        vrf-target export target:1:300;
    }
    vni 1400 {
        vrf-target export target:1:400;
    }
}
```

The `extended-vni-list` is what establishes which VNIs will be part of the EVPN/VXLAN MP-BGP domain. The reader may recall previous discussions about BUM replication options available in EVPN. We are forgoing a multicast underlay in favor of EVPN and VXLAN ingress replication.

We configure different route targets (RTs) for each VNI instance under `vni-routing-options`. We will later observe Type 2 routes being exported with these RTs.

```
lab@leaf-1> show configuration vlans
v100 {
    vlan-id 100;
    vxlan {
        vni 1100;
        ingress-node-replication;
    }
}
v200 {
    vlan-id 200;
    vxlan {
        vni 1200;
        ingress-node-replication;
    }
}
v300 {
    vlan-id 300;
    vxlan {
        vni 1300;
        ingress-node-replication;
    }
}
v400 {
    vlan-id 400;
    vxlan {
        vni 1400;
        ingress-node-replication;
    }
}
```

Under the `vlans` stanza, we map locally significant vlan-ids to globally significant VNIs. Again, we configure ingress replication rather than relying on a multicast underlay.

Lastly, we configure the EVPN MP-BGP sessions.

```
lab@leaf-1> show configuration protocols bgp group EVPN_VXLAN_CORE
type external;
multihop {
    ttl 255;
    no-nexthop-change;
}
local-address 1.255.255.4;
family evpn {
    signaling;
}
```

```
peer-as 65400;
neighbor 1.255.255.0 {
    description core-1;
}
neighbor 1.255.255.1 {
    description core-2;
}

{master:0}
lab@leaf-1> show configuration protocols bgp group EVPN_VXLAN_LEAF
type internal;
local-address 1.255.255.4;
family evpn {
    signaling;
}
neighbor 1.255.255.5 {
    description leaf-2;
}
```

We establish that this is an EVPN MP-BGP session with `family evpn signaling`. And for an EBGP EVPN session, this will always be `multihop` given loopback peering.

Under Leaf-1's `EVPN_VXLAN_CORE` configuration, one might also notice `no-nexthop-change`. This is required in our specific design topology given a combination of factors.

First, this example topology uses a single ASN per tier. Thus the Leaf-1 to Leaf-2 EVPN MP-BGP session will be via internal BGP (IBGP). Second, Leaf-1 and Leaf-2 both peer via eBGP with each core.

If we refer to the single-homed Tenant_A CE device that is attached to Leaf-2 in Figure 25, let's consider the Type 2 route for Tenant_A that Core-1 will receive. Core-1 will receive Leaf-2's direct EBGP advertisement, with both BGP source and protocol-next-hop of Leaf-2's loopback address.

Leaf-2 will also advertise this same Type 2 route to Leaf-1. Leaf-1 will subsequently advertise this Type 2 route to Core-2, with BGP source and protocol-next-hop of Leaf-1's own loopback address, due to the fact that this route was learned from Leaf-2 via IBGP.

Core-1 now has two otherwise equal copies of the same Type 2 route. However, should Core-1 attempt to forward using the Type 2 route with Leaf-1's protocol-next-hop, this traffic will be black holed. Should Core-1 send this traffic across its VTEP to Leaf-1, Leaf-1 will successfully receive this traffic on its VTEP to Core-1. However, Leaf-1's reachability to this Tenant_C MAC is on its VTEP towards Leaf-2. It is at this point that the traffic will be dropped. EVPN split horizon rules dictate that traffic received on a PE-facing interface should never be re-forwarded out of another PE-facing interface (refer to "EVPN Ingress Replication—Split Horizon and Designated Forwarders").

The `no-nexthop-change` configuration is thus needed to preserve the original protocol-next-hop, such that Core-1 (or Core-2) send directly to the PE device to which a particular single-homed CE device is attached.

It is worth noting, however, that in real-world deployments, this is not likely to be required. This is due to several factors. If we were using a single autonomous system (AS) per PE device, then this IBGP/EBGP interaction would not exist, and this particular configuration would not be required. Furthermore, and perhaps more importantly, for medium to large-scale EVPN deployments, it is unlikely that EVPN BGP meshes will be configured, as this will quickly become unwieldy as the number of PE devices grows.

Instead, real-world deployments would deploy stand-alone MBGP EBGP route reflectors or route servers (such as the Juniper Networks Junos® OS Virtual Route Reflector feature), which would preserve the original protocol-next-hop. Each PE device would instead have a single MP-EBGP session to this route reflector (or redundant route reflectors).

Leaf-1 also has multihomed tenants which require additional interface-level configurations.

```
lab@leaf-1> show configuration interfaces xe-0/0/32
ether-options {
    802.3ad ae0;
}
```

```
{master:0}
lab@leaf-1> show configuration interfaces xe-0/0/33
ether-options {
    802.3ad ae1;
}

{master:0}
lab@leaf-1> show configuration interfaces ae0
esi {
    00:01:01:01:01:01:01:01:01:01;
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 00:00:00:01:01:01;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members v100;
        }
    }
}

{master:0}
lab@leaf-1> show configuration interfaces ae1
esi {
    00:02:02:02:02:02:02:02:02:02;
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 00:00:00:01:01:01;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members v200;
        }
    }
}
```

We have configured two Link Aggregation Control Protocol (LACP)-enabled aggregate-Ethernet (LAG) interfaces. The EVPN-specific configuration is under `esi`. We must first configure an end system identifier (ESI) value, which is globally unique across the entire EVPN domain. The `all-active` configuration stipulates that all PE routers to which this multihomed tenant is attached can forward traffic to/from the CE device, such that all CE router links are actively used.

Core-1

Most of Core-1's (MX Series) configuration takes place in the routing instances stanza. Here we have both virtual switches and virtual routers.

```
lab@core-1> show configuration routing-instances
VRF_Tenant_A {
    instance-type vrf;
    interface irb.1100;
    route-distinguisher 1.255.255.0:1100;
    vrf-target target:10:100;
}
VRF_Tenant_B {
    instance-type vrf;
    interface irb.1200;
    route-distinguisher 1.255.255.0:1200;
    vrf-target target:10:200;
}
VRF_Tenant_C {
    instance-type vrf;
    interface irb.1300;
    route-distinguisher 1.255.255.0:1300;
    vrf-target target:10:300;
}
VRF_Tenant_D {
    instance-type vrf;
    interface irb.1400;
    route-distinguisher 1.255.255.0:1400;
    vrf-target target:10:400;
}
VS_VLAN100 {
    vtep-source-interface lo0.0;
    instance-type virtual-switch;
    route-distinguisher 1.255.255.0:100;
    vrf-import VS_VLAN100_IMP;
    vrf-target target:1:100;
    protocols {
        evpn {
            encapsulation vxlan;
            extended-vni-list 1100;
            multicast-mode ingress-replication;
        }
    }
    bridge-domains {
        bd1100 {
            vlan-id 100;
            routing-interface irb.1100;
```

```
                    vxlan {
                        vni 1100;
                        ingress-node-replication;
                    }
                }
            }
        }
        VS_VLAN200 {
            vtep-source-interface lo0.0;
            instance-type virtual-switch;
            route-distinguisher 1.255.255.0:200;
            vrf-import VS_VLAN200_IMP;
            vrf-target target:1:200;
            protocols {
                evpn {
                    encapsulation vxlan;
                    extended-vni-list 1200;
                    multicast-mode ingress-replication;
                }
            }
            bridge-domains {
                bd1200 {
                    vlan-id 200;
                    routing-interface irb.1200;
                    vxlan {
                        vni 1200;
                        ingress-node-replication;
                    }
                }
            }
        }
        VS_VLAN300 {
            vtep-source-interface lo0.0;
            instance-type virtual-switch;
            route-distinguisher 1.255.255.0:300;
            vrf-import VS_VLAN300_IMP;
            vrf-target target:1:300;
            protocols {
                evpn {
                    encapsulation vxlan;
                    extended-vni-list 1300;
                    multicast-mode ingress-replication;
                }
            }
            bridge-domains {
                bd1300 {
                    vlan-id 300;
                    routing-interface irb.1300;
                    vxlan {
```

```
                vni 1300;
                ingress-node-replication;
            }
        }
    }
}
VS_VLAN400 {
    vtep-source-interface lo0.0;
    instance-type virtual-switch;
    route-distinguisher 1.255.255.0:400;
    vrf-import VS_VLAN400_IMP;
    vrf-target target:1:400;
    protocols {
        evpn {
            encapsulation vxlan;
            extended-vni-list 1400;
            multicast-mode ingress-replication;
        }
    }
    bridge-domains {
        bd1400 {
            vlan-id 400;
            routing-interface irb.1400;
            vxlan {
                vni 1400;
                ingress-node-replication;
            }
        }
    }
```

Let's first consider an individual virtual switch instance, of which we have four, one for each VNI. The configuration items on Core-1 under each virtual switch are similar to what was configured on Leaf-1 under the single `switch-options` hierarchy.

On VS_VLAN400, for example, we define the globally significant RD, the `vrf-target`, and the `vrf-import` list for this particular virtual switch. We specify VNI 1400 as belonging to this EVPN instance, specify ingress replication (instead of a multicast underlay), and we associate locally significant VLAN400 with globally significant VNI 1400.

Because Core-1 is acting as the default gateway for VNI 1400, we are also associating an IRB interface under the bridge domain.

Additionally, we are configuring IP VPN virtual routing and forwarding (VRF) tables for each tenant. This ensures L3 separation between tenants, and these IP VPN VRFs might be extended further across an MPLS-enabled WAN.

In our example, we have a one-to-one mapping between bridge domains and IP VPNs. This is not a restriction, however. Tenants might have multiple bridge domains (and multiple IRBs) associated with them, in which case these multiple IRBs can be configured in the same VRF instance, to allow L3 connectivity.

Notice that each virtual switch has a unique `vrf-import` policy configured. These individual policies are shown next.

```
lab@core-1> show configuration policy-options
policy-statement VS_VLAN100_IMP {
    term ESI {
        from community comm-leaf_esi;
```

```
            then accept;
        }
        term VS_VLAN100 {
            from community comm-VS_VLAN100;
            then accept;
        }
    }
    policy-statement VS_VLAN200_IMP {
        term ESI {
            from community comm-leaf_esi;
            then accept;
        }
        term VS_VLAN200 {
            from community comm-VS_VLAN200;
            then accept;
        }
    }
    policy-statement VS_VLAN300_IMP {
        term ESI {
            from community comm-leaf_esi;
            then accept;
        }
        term VS_VLAN300 {
            from community comm-VS_VLAN300;
            then accept;
        }
    }
    policy-statement VS_VLAN400_IMP {
        term ESI {
            from community comm-leaf_esi;
            then accept;
        }
        term VS_VLAN400 {
            from community comm-VS_VLAN400;
            then accept;
        }
    }
    policy-statement lo0 {
        from {
            family inet;
            protocol direct;
            route-filter 0.0.0.0/0 prefix-length-range /32-/32;
        }
        then accept;
    }
    policy-statement load-balance {
        term 1 {
            then {
                load-balance per-packet;
            }
        }
    }
    community comm-VS_VLAN100 members target:1:100;
    community comm-VS_VLAN200 members target:1:200;
    community comm-VS_VLAN300 members target:1:300;
    community comm-VS_VLAN400 members target:1:400;
    community comm-leaf_esi members target:9999:9999;
```

Each virtual switch-specific policy has two terms. The first will accept target:9999:9999, which is necessary to ensure that all virtual switches will import the Type 1 ESI routes from all leaves. The second will import Type 2 ESI routes from a specific RT for a single VNI.

This presents an interesting observation. In Figure 25, we notice that there are two multihomed tenants: Tenant_A and Tenant_B. Strictly speaking, VS_VLAN100 doesn't need to have knowledge of Tenant_B's ESI, and VS_VLAN200 doesn't need to know about Tenant_A's ESI.

Given that the `vrf-target` is a (virtual) switch-level configuration, and Leaf-1 supports a single switch instance, all ESIs will be in all VRFs. However, while Tenant_A might have a Type 1 route for a Tenant_B ESI, there will be Type 2 (MAC) routes associated with this ESI, and thus tenant isolation will remain intact.

Here is the IRB-level configuration:

```
lab@core-1> show configuration interfaces irb
unit 1100 {
    family inet {
        address 100.0.0.2/24 {
            virtual-gateway-address 100.0.0.1;
        }
    }
}
unit 1200 {
    family inet {
        address 200.0.0.2/24 {
            virtual-gateway-address 200.0.0.1;
        }
    }
}
unit 1300 {
    family inet {
        address 10.10.10.2/24 {
            virtual-gateway-address 10.10.10.1;
        }
    }
}
unit 1400 {
    family inet {
        address 10.10.10.2/24 {
            virtual-gateway-address 10.10.10.1;
        }
    }
}
```

We'll notice two things. First, because this is a multitenant environment, IP overlap between tenants is valid. Second, there is a `virtual-gateway-address` configured per IRB. This is a shared MAC address and IP address across Core-1 and Core-2.

Although our example is only using two gateway devices, it is important to note that this solution scales horizontally across any number of devices. It's thus entirely feasible that a deployment might have a dozen actively redundant gateways per IRB, all actively load-balancing traffic from each PE device. This is a clear advantage over traditional "multichassis LAG" deployments, which only scale to two devices. The scalability that EVPN provides is a requirement for medium or large-scale data centers.

Note: The EVPN anycast gateway with VXLAN feature is supported, at the time of this document's publication, on MX Series and QFX10000 platforms. Platforms based on Broadcom Trident2 chipsets, such as the QFX5100, cannot route between VXLANs natively. Future systems built on Broadcom Trident2+ will not have this restriction.

Lastly, MP-BGP sessions towards Leaf-1 and Leaf-2 need to be configured.

```
lab@core-1> show configuration protocols bgp group EVPN_VXLAN
type external;
local-address 1.255.255.0;
family evpn {
    signaling;
}
peer-as 65402;
multipath;
neighbor 1.255.255.4 {
    description leaf-1;
    multihop {
        ttl 255;
    }
}
neighbor 1.255.255.5 {
    description leaf-2;
    multihop {
        ttl 255;
    }
}
```

EVPN and VXLAN Troubleshooting

Once both the underlay and EVPN overlay are configured, there are a number of useful commands available for troubleshooting.

A valid EVPN route, once received, takes the following sequence from software until being programmed into hardware:

- bgp.evpn.0 = global EVPN routing table in the Junos OS routing protocol process (RPD)
- default.switch.evpn.0 (on QFX5100) = switch-level EVPN forwarding table in Junos OS RPD
- <virtual-switch-name>.evpn.0 (on MX) = virtual-switch level EVPN forwarding table in Junos OS RPD
- L2ALD (Layer-2 Address Learning Daemon)
  "show ethernet-switching-table" || "show bridge mac-table"
- Kernel
  "show route forwarding-table"
- Packet Forwarding Engine (PFE)
  from FPC vty: "show l2 manager mac-table"

We'll step through three examples: a single-homed CE device, multihomed CE device, and the anycast gateway.

Single-homed CE device

In our first example, we'll walk through MAC reachability to Tenant_C single-homed to Leaf-1.

The first step is to verify that the MAC address is learned locally on Leaf-1. The Type 2 route will only be generated by Leaf-1 after the MAC is first learned.

```
lab@leaf-1> show ethernet-switching table vlan-id 300
MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent 
static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC, O - 
ovsdb MAC)
Ethernet switching table : 5 entries, 5 learned
Routing instance : default-switch
   Vlan                MAC                 MAC      Logical            Active
   name                address             flags   interface           source
   v300                00:00:5e:00:01:01   DR       esi.1726
05:00:00:ff:78:00:00:05:14:00
   v300                00:21:59:c8:24:65   D        xe-0/0/34.0
   v300                00:21:59:c8:24:69   D        vtep.32771         1.255.255.5
   v300                40:a6:77:9a:43:f0   D        vtep.32769         1.255.255.0
   v300                40:a6:77:9a:47:f0   D        vtep.32770         1.255.255.1
```

From the output above, we can see that MAC `00:21:59:c8:24:65` is successfully learned towards the Tenant_C CE device on xe-0/0/34.0. And from the output below, we can see that we generate the Type 2 route to Core-1.

```
lab@leaf-1> show route advertising-protocol bgp 1.255.255.0 evpn-mac-address 
00:21:59:c8:24:65
bgp.evpn.0: 77 destinations, 77 routes (77 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop         MED     Lclpref    AS path
  2:1.255.255.4:1::1300::00:21:59:c8:24:65/304
*                          Self                                   I
  2:1.255.255.4:1::1400::00:21:59:c8:24:65/304
*                          Self                                   I

default-switch.evpn.0: 67 destinations, 67 routes (67 active, 0 holddown, 0 
hidden)
  Prefix                  Nexthop         MED     Lclpref    AS path
  2:1.255.255.4:1::1300::00:21:59:c8:24:65/304
*                         Self                                    I
  2:1.255.255.4:1::1400::00:21:59:c8:24:65/304
*                         Self                                    I

__default_evpn__.evpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
```

The reader will notice that the same MAC is advertised twice, with two different RTs. On xe-0/0/34, Tenant_C and Tenant_D are on the same physical server, running on different virtual machines. Tenant isolation is preserved by advertising these MAC addresses on different VNIs and RTs.

On Core-1, we'll see that this Type 2 route is received into bgp.evpn.0.

```
lab@core-1> show route receive-protocol bgp 1.255.255.4 evpn-mac-address
00:21:59:c8:24:65 extensive table bgp.evpn.0

<output omitted>

* 2:1.255.255.4:1::1300::00:21:59:c8:24:65/304 (2 entries, 0 announced)

     Import Accepted

     Route Distinguisher: 1.255.255.4:1

     Nexthop: 1.255.255.4

     AS path: 65402 I

     Communities: target:1:300 encapsulation0:0:0:0:vxlan


* 2:1.255.255.4:1::1400::00:21:59:c8:24:65/304 (2 entries, 0 announced)

     Import Accepted

     Route Distinguisher: 1.255.255.4:1

     Nexthop: 1.255.255.4

     AS path: 65402 I

     Communities: target:1:400 encapsulation0:0:0:0:vxlan
```

Again, we receive two Type 2 routes for `00:21:59:c8:24:65`, but with different RTs. The RD is from Leaf-1, set as 1.255.255.4:1.

On Core-1, we then check to see whether this Type 2 route has been successfully imported from the bgp.evpn.0 table into the EVPN switch instance.

First, we'll notice that in Tenant_C's virtual switch, we only see the Type 2 route which was advertised with target:1:1300, which is expected.

```
lab@core-1> show route table VS_VLAN300.evpn.0 evpn-mac-address
00:21:59:c8:24:65 | grep 00:21:59:c8:24:65

2:1.255.255.4:1::1300::00:21:59:c8:24:65/304

lab@core-1>
```

Here's a look at this Type 2 route in greater detail:

```
lab@core-1> show route table VS_VLAN300.evpn.0 evpn-mac-address 00:21:59:c8:24:65
extensive


<output omitted>

2:1.255.255.4:1::1300::00:21:59:c8:24:65/304 (2 entries, 1 announced)
   *BGP    Preference: 170/-101
```

```
       Route Distinguisher: 1.255.255.4:1
       Next hop type: Indirect
       Address: 0x2cf479c
       Next-hop reference count: 76
       Source: 1.255.255.4
       Protocol next hop: 1.255.255.4
       Indirect next hop: 0x2 no-forward INH Session ID: 0x0
       State: <Secondary Active Ext>
       Local AS: 65400 Peer AS: 65402
       Age: 4:47  Metric2: 0
       Validation State: unverified
       Task: BGP_65402.1.255.255.4+179
       Announcement bits (1): 0-VS_VLAN300-evpn
       AS path: 65402 I
       Communities: target:1:300 encapsulation0:0:0:0:vxlan
       Import Accepted
       Localpref: 100
       Router ID: 1.255.255.4
       Primary Routing Table bgp.evpn.0
       Indirect next hops: 1
               Protocol next hop: 1.255.255.4
               Indirect next hop: 0x2 no-forward INH Session ID: 0x0
               Indirect path forwarding next hops: 2
                       Next hop type: Router
                       Next hop: 1.0.0.1 via ge-1/0/0.0
                       Session Id: 0x140
                       Next hop: 1.0.0.3 via ge-1/0/1.0
                       Session Id: 0x141
                  1.255.255.4/32 Originating RIB: inet.0
                    Node path count: 1
                    Forwarding nexthops: 2
                     Nexthop: 1.0.0.1 via ge-1/0/0.0
BGP    Preference: 170/-101
       Route Distinguisher: 1.255.255.4:1
       Next hop type: Indirect
       Address: 0x2cf479c
       Next-hop reference count: 76
       Source: 1.255.255.5
       Protocol next hop: 1.255.255.4
       Indirect next hop: 0x2 no-forward INH Session ID: 0x0
       State: <Secondary NotBest Ext>
       Inactive reason: Not Best in its group - Router ID
       Local AS: 65400 Peer AS: 65402
       Age: 4:47  Metric2: 0
       Validation State: unverified
       Task: BGP_65402.1.255.255.5+61407
       AS path: 65402 I
       Communities: target:1:300 encapsulation0:0:0:0:vxlan
       Import Accepted
```

```
                Localpref: 100
                Router ID: 1.255.255.5
                Primary Routing Table bgp.evpn.0
                Indirect next hops: 1
                        Protocol next hop: 1.255.255.4
                        Indirect next hop: 0x2 no-forward INH Session ID: 0x0
                        Indirect path forwarding next hops: 2
                                Next hop type: Router
                                Next hop: 1.0.0.1 via ge-1/0/0.0
                                Session Id: 0x140
                                Next hop: 1.0.0.3 via ge-1/0/1.0
                                Session Id: 0x141
                           1.255.255.4/32 Originating RIB: inet.0
                             Node path count: 1
                             Forwarding nexthops: 2
                              Nexthop: 1.0.0.1 via ge-1/0/0.0
```

Notice that Core-1 receives two copies. The first is the direct advertisement from Leaf-1 (Source: 1.255.255.4). The second is the indirect advertisement from Leaf-1 -> Leaf-2 -> Core-1 (Source: 1.255.255.5). Since we have configured `no-nexthop-change` on Leaf-2, we preserve the correct protocol next-hop of 1.255.255.4.

The L2ALD copy can be verified as follows:

```
lab@core-1> show bridge mac-table instance VS_VLAN300

MAC flags       (S -static MAC, D -dynamic MAC, L -locally learned, C -Control MAC
    O -OVSDB MAC, SE -Statistics enabled, NM -Non configured MAC, R -Remote PE MAC)

Routing instance : VS_VLAN300
 Bridging domain : bd1300, VLAN : 300
   MAC                   MAC       Logical               Active
   address               flags     interface             source
   00:21:59:c8:24:65     D         vtep.32774            1.255.255.4
   00:21:59:c8:24:69     D         vtep.32783            1.255.255.5
```

We see in the above output that `00:21:59:c8:24:65` is reachable via the vtep.32774 to Leaf-1.

The kernel-level forwarding table can be queried using the command below:

```
lab@core-1> show route forwarding-table family bridge vpn VS_VLAN300
Routing table: VS_VLAN300.evpn-vxlan
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            perm     0                    dscd      540     1
vtep.32774         intf     0                    comp      699     7
vtep.32783         intf     0                    comp      714     5

Routing table: VS_VLAN300.evpn-vxlan
Bridging domain: bd1300.evpn-vxlan
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
00:21:59:c8:24:65/48 user     0                  comp      699     7
```

```
00:21:59:c8:24:69/48 user     0                  comp      714     5
0x30003/51         user     0                    comp      714     2
```

Tenant_C's MAC, `00:21:59:c8:24:65`,is reachable via index 699.

Index 699 (NH-Id) can also be correlated to the correct VNI 1300 and Remote VTEP-ID of 1.255.255.4:

```
lab@core-1> show l2-learning vxlan-tunnel-end-point remote
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx
<default>                 0   1.255.255.0      lo0.0    0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.4      351       697
    VNID          MC-Group-IP
    1100          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.5      356       711
    VNID          MC-Group-IP
    1100          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.4      352       698
    VNID          MC-Group-IP
    1200          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.5      355       710
    VNID          MC-Group-IP
    1200          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.4      353       699
    VNID          MC-Group-IP
    1300          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.5      357       714
    VNID          MC-Group-IP
    1300          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.4      354       700
    VNID          MC-Group-IP
    1400          0.0.0.0
 RVTEP-IP         IFL-Idx   NH-Id
 1.255.255.5      358       709
    VNID          MC-Group-IP
    1400          0.0.0.0
```

Lastly, we can also see that `00:21:59:c8:24:65` is successfully programmed in PFE hardware.

```
# show l2 manager mac-table
<output omitted>
route table name    : VS_VLAN300.7
```

```
  mac counters
    maximum   count
    0           2
  mac table information
  mac address        BD     learn  Entry  entry  hal     hardware info
                     Index  vlan   Flags  ifl    ifl    pfe  mask  ifl
  ----------------------------------------------------------------------
  00:21:59:c8:24:65  4      0     0x0014 vtep.32774 vtep.32774  0   -D    src
unknown dest vtep.32774
  00:21:59:c8:24:69  4      0     0x0014 vtep.32783 vtep.32783  0   -D    src
unknown dest vtep.32783
 Displayed 2 entries for routing instance VS_VLAN300.7
```

Multihomed CE device

In this example, we'll consider the multihomed Tenant_B CE device on Leaf-1 and Leaf-2. Since it is multihomed, Leaf-1 and Leaf-2 will be advertising both Type 1 and Type 2 reachability towards this CE device.

```
lab@leaf-1> show ethernet-switching table vlan-id 200

MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent
static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC, O -
ovsdb MAC)


Ethernet switching table : 4 entries, 4 learned
Routing instance : default-switch
   Vlan                MAC                 MAC     Logical             Active
   name                address             flags   interface           source
   v200                00:00:5e:00:01:01   DR       esi.1727
05:00:00:ff:78:00:00:04:b0:00
   v200                00:21:59:c8:24:64   DL       ae1.0
   v200                40:a6:77:9a:43:f0   D        vtep.32769         1.255.255.0
   v200                40:a6:77:9a:47:f0   D        vtep.32770         1.255.255.1

lab@leaf-2> show ethernet-switching table vlan-id 200

MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent
static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC, O -
ovsdb MAC)

Ethernet switching table : 5 entries, 5 learned
Routing instance : default-switch
   Vlan                MAC                 MAC     Logical                Active
   name                address             flags   interface              source
   v200                00:00:5e:00:01:01   DR       esi.1727
05:00:00:ff:78:00:00:04:b0:00
   v200                00:21:59:c8:24:41   DL       ae1.0
```

```
    v200                   00:21:59:c8:24:68    DL        ae1.0
    v200                   40:a6:77:9a:43:f0    D         vtep.32770          1.255.255.0
    v200                   40:a6:77:9a:47:f0    D         vtep.32769          1.255.255.1
```

In our example topology, `00:21:59:c8:24:64` represents the physical MAC of the Tenant _B network interface card (NIC) physically attached to Leaf-1. `00:21:59:c8:24:68` represents the physical MAC of the Tenant_B NIC physically attached to Leaf-2. `00:21:59:c8:24:41` is a virtual MAC that, at the time of the capture, was only thus far learned on Leaf-2.

Recall that ae1 belongs on ESI `00:02:02:02:02:02:02:02:02:02`.

```
lab@leaf-1> show configuration interfaces ae1 esi

00:02:02:02:02:02:02:02:02:02;

all-active;
```

On Core-1, we'll make a number of observations regarding Tenant_B's EVPN table.

```
lab@core-1> show route table VS_VLAN200.evpn.0


VS_VLAN200.evpn.0: 18 destinations, 31 routes (18 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1:1.255.255.4:0::010101010101010101::FFFF:FFFF/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.4:0::020202020202020202::FFFF:FFFF/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.4:1::010101010101010101::0/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.4:1::020202020202020202::0/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.5:0::010101010101010101::FFFF:FFFF/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.5:0::020202020202020202::FFFF:FFFF/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.5:1::010101010101010101::0/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
1:1.255.255.5:1::020202020202020202::0/304
                   *[BGP/170] 23:27:33, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 23:27:33, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
<output omitted>
2:1.255.255.4:1::1200::00:21:59:c8:24:64/304
                   *[BGP/170] 4d 10:47:09, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                    > to 1.0.0.1 via ge-1/0/0.0
                      to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 1d 00:44:04, localpref 100, from 1.255.255.5
```

```
                      AS path: 65402 I, validation-state: unverified
                    > to 1.0.0.1 via ge-1/0/0.0
                      to 1.0.0.3 via ge-1/0/1.0
2:1.255.255.5:1::1200::00:21:59:c8:24:41/304
                   *[BGP/170] 00:14:09, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                    > to 1.0.0.1 via ge-1/0/0.0
                      to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 00:14:09, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                    > to 1.0.0.1 via ge-1/0/0.0
                      to 1.0.0.3 via ge-1/0/1.0
2:1.255.255.5:1::1200::00:21:59:c8:24:68/304
                   *[BGP/170] 1d 00:44:04, localpref 100, from 1.255.255.4
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0
                    [BGP/170] 4d 10:47:09, localpref 100, from 1.255.255.5
                      AS path: 65402 I, validation-state: unverified
                      to 1.0.0.1 via ge-1/0/0.0
                    > to 1.0.0.3 via ge-1/0/1.0

<output omitted>
```

1. As previously explained, we will see all ESI routes (target:9999:9999) in this virtual switch instance, and we will only see Type 2 routes with target:1200.
2. For the Tenant_B CE device, we observe four different routes for ESI 00:02:02:02:02:02:02:02:02:02.
   a. `1:1.255.255.4:0::020202020202020202::FFFF:FFFF/304`
      i. This is the per Ethernet segment AD Type 1 EVPN route originated from Leaf-1. The RD is taken from global level routing-options.
      ii. Core-1 receives this Type 1 route, which is originated from Leaf-1, from both Leaf-1 and Leaf-2.
   b. `1:1.255.255.4:1::020202020202020202::0/304`
      i. This is the per-EVI AD Type 1 EVPN route. The RD is taken from the routing instance, or in the case of QFX5100, switch-options.
      ii. Core-1 receives this Type 1 route, which is originated from Leaf-1, from both Leaf-1 and Leaf-2.
   c. `1:1.255.255.5:0::020202020202020202::FFFF:FFFF/304`
      i. This is the per Ethernet segment AD Type 1 EVPN route originated from Leaf-2. The RD is taken from global level routing-options.
      ii. Core-1 receives this Type 1 route, which is originated from Leaf-2, from both Leaf-2 and Leaf-1.
   d. `1:1.255.255.5:1::020202020202020202::0/304`
      i. This is the per-EVI AD Type 1 EVPN route. The RD is taken from the routing instance, or in the case of QFX5100, switch-options.
      ii. Core-1 receives this Type 1 route, which is originated from Leaf-2, from both Leaf-2 and Leaf-1.
3. We receive Type 2 routes, originated as expected, for the two physical and one virtual MAC addresses belonging to Tenant_B CE device.

Core-1's L2ALD table can be seen below:

```
lab@core-1> show bridge mac-table instance VS_VLAN200
```

```
MAC flags       (S -static MAC, D -dynamic MAC, L -locally learned, C -Control MAC
    O -OVSDB MAC, SE -Statistics enabled, NM -Non configured MAC, R -Remote PE MAC)

Routing instance : VS_VLAN200
 Bridging domain : bd1200, VLAN : 200
   MAC                 MAC      Logical                Active
   address             flags   interface              source
   00:21:59:c8:24:41   DR       esi.703
00:02:02:02:02:02:02:02:02:02
   00:21:59:c8:24:64   DR       esi.703
00:02:02:02:02:02:02:02:02:02
   00:21:59:c8:24:68   DR       esi.703
00:02:02:02:02:02:02:02:02:02
```

Core-1's kernel forwarding table looks like this:

```
lab@core-1> show route forwarding-table vpn VS_VLAN200
Routing table: VS_VLAN200.evpn-vxlan
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            perm     0                    dscd      536     1
vtep.32774         intf     0                    comp      702     6
vtep.32778         intf     0                    comp      720     6

Routing table: VS_VLAN200.evpn-vxlan
Bridging domain: bd1200.evpn-vxlan
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
00:21:59:c8:24:41/48 user     0                  indr  1048579     4
                                                 comp      703     2
00:21:59:c8:24:64/48 user     0                  indr  1048579     4
                                                 comp      703     2
00:21:59:c8:24:68/48 user     0                  indr  1048579     4
                                                 comp      703     2
0x30002/51         user     0                    comp      714     2
```

We're still not quite able to determine what VTEP(s) are being used to forward to ESI 00:02:02:02:02:02:02:02:02:02. We then execute the following command:

```
lab@core-1> show l2-learning vxlan-tunnel-end-point esi
<output omitted>
ESI                           RTT                      VLNBH INH     ESI-IFL
LOC-IFL   #RVTEPs
00:01:01:01:01:01:01:01:01:01 VS_VLAN200               704   1048580 esi.704
2
    RVTEP-IP             RVTEP-IFL      VENH     MASK-ID   FLAGS
    1.255.255.5          vtep.32778     720      1         2
    1.255.255.4          vtep.32774     702      0         2
<output omitted>
```

Here we can observe that we are actively load-balancing on our VTEP interfaces to both Leaf-1 and Leaf-2 for MAC addresses on this ESI, which validates the all active configuration on Leaf-1 and Leaf-2.

The reference to "esi.703" is an internal "unilist" next-hop that is comprised of multiple, valid, individual ECMP next hops. The extra curious reader can correlate these individual next hops to the unilist by executing this shell level command:

```
lab@core-1> start shell command "nhinfo -di 703"
<output omitted>
NHs in list: 720, 702,
<output omitted>
```

Here, we can reference index 720 with vtep.32778 to Leaf-2 and index 702 with vtep.32774 to Leaf-1 from output above.

EVPN anycast gateway

Troubleshooting the EVPN anycast gateway is similar to the multihomed CE device scenario. In this section, we will troubleshoot from the leaf (QFX5100) perspective, and consider the anycast gateway on both Core-1 and Core-2 for VNI 1100 (VLAN100 on both Leaf-1 and Leaf-2).

Note: At the time of this document's publication, the initial release of EVPN+VXLAN on QFX5100 will load-balance anycast gateways *per VNI*. That is to say, for a given VNI, a QFX5100 leaf will forward traffic to a single VTEP. The immediate next release of EVPN+VXLAN on QFX5100 will duplicate the behavior seen on MX Series routers in the previous example for a multihomed CE device. Once implemented, the QFX5100 will then load-balance per *5-tuple flow* to different VTEPs (i.e., "cores") on the same VNI.

```
lab@leaf-1> show route receive-protocol bgp 1.255.255.0

inet.0: 13 destinations, 18 routes (13 active, 0 holddown, 0 hidden)

:vxlan.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)

bgp.evpn.0: 75 destinations, 123 routes (75 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop         MED     Lclpref    AS path
  1:1.255.255.0:0::050000ff780000044c00::FFFF:FFFF/304
*                          1.255.255.0                            65400 I
  1:1.255.255.0:0::050000ff78000004b000::FFFF:FFFF/304
*                          1.255.255.0                            65400 I
  1:1.255.255.0:0::050000ff780000051400::FFFF:FFFF/304
*                          1.255.255.0                            65400 I
  1:1.255.255.0:0::050000ff780000057800::FFFF:FFFF/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:300::1300::00:00:5e:00:01:01/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:300::1300::40:a6:77:9a:43:f0/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:100::1100::00:00:5e:00:01:01/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:100::1100::40:a6:77:9a:43:f0/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:400::1400::00:00:5e:00:01:01/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:400::1400::40:a6:77:9a:43:f0/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:200::1200::00:00:5e:00:01:01/304
*                          1.255.255.0                            65400 I
  2:1.255.255.0:200::1200::40:a6:77:9a:43:f0/304
```

```
*                         1.255.255.0                              65400 I
  2:1.255.255.0:300::1300::00:00:5e:00:01:01::10.10.10.1/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:300::1300::40:a6:77:9a:43:f0::10.10.10.2/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:100::1100::00:00:5e:00:01:01::100.0.0.1/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:100::1100::40:a6:77:9a:43:f0::100.0.0.2/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:400::1400::00:00:5e:00:01:01::10.10.10.1/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:400::1400::40:a6:77:9a:43:f0::10.10.10.2/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:200::1200::00:00:5e:00:01:01::200.0.0.1/304
*                         1.255.255.0                              65400 I
  2:1.255.255.0:200::1200::40:a6:77:9a:43:f0::200.0.0.2/304
*                         1.255.255.0                              65400 I
<output omitted>
```

On Leaf-1, we see in the output above that we receive Type 1 advertisements for the auto-generated ESIs for the anycast gateway. We also see the anycast and physical MAC addresses (a.b.c.2) per bridge domain.

We expect to see the same Type 1, same anycast Type 2, and different physical Type 2 (a.b.c.3) from Core-2:

```
lab@leaf-1> show route receive-protocol bgp 1.255.255.1

inet.0: 13 destinations, 18 routes (13 active, 0 holddown, 0 hidden)

:vxlan.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)

bgp.evpn.0: 75 destinations, 123 routes (75 active, 0 holddown, 0 hidden)
  Prefix                 Nexthop        MED     Lclpref    AS path
  1:1.255.255.1:0::050000ff780000044c00::FFFF:FFFF/304
*                         1.255.255.1                              65400 I
  1:1.255.255.1:0::050000ff78000004b000::FFFF:FFFF/304
*                         1.255.255.1                              65400 I
  1:1.255.255.1:0::050000ff780000051400::FFFF:FFFF/304
*                         1.255.255.1                              65400 I
  1:1.255.255.1:0::050000ff780000057800::FFFF:FFFF/304
*                         1.255.255.1                              65400 I
  2:1.255.255.1:300::1300::00:00:5e:00:01:01/304
*                         1.255.255.1                              65400 I
  2:1.255.255.1:300::1300::40:a6:77:9a:47:f0/304
*                         1.255.255.1                              65400 I
  2:1.255.255.1:100::1100::00:00:5e:00:01:01/304
*                         1.255.255.1                              65400 I
  2:1.255.255.1:100::1100::40:a6:77:9a:47:f0/304
*                         1.255.255.1                              65400 I
  2:1.255.255.1:400::1400::00:00:5e:00:01:01/304
*                         1.255.255.1                              65400 I
```

```
  2:1.255.255.1:400::1400::40:a6:77:9a:47:f0/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:200::1200::00:00:5e:00:01:01/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:200::1200::40:a6:77:9a:47:f0/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:300::1300::00:00:5e:00:01:01::10.10.10.1/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:300::1300::40:a6:77:9a:47:f0::10.10.10.3/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:100::1100::00:00:5e:00:01:01::100.0.0.1/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:100::1100::40:a6:77:9a:47:f0::100.0.0.3/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:400::1400::00:00:5e:00:01:01::10.10.10.1/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:400::1400::40:a6:77:9a:47:f0::10.10.10.3/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:200::1200::00:00:5e:00:01:01::200.0.0.1/304
*                         1.255.255.1                             65400 I
  2:1.255.255.1:200::1200::40:a6:77:9a:47:f0::200.0.0.3/304
<output omitted>
```

Similarly, if we look at `the default-switch.evpn.0` table on Leaf-1, we see for the ESI for VNI 1110:

lab@leaf-1> show route table default-switch.evpn.0 evpn-esi-value 05:00:00:ff:78:00:00:04:4c:00

```
default-switch.evpn.0: 66 destinations, 114 routes (66 active, 0 holddown, 0
hidden)
+ = Active Route, - = Last Active, * = Both

1:1.255.255.0:0::050000ff780000044c00::FFFF:FFFF/304
                   *[BGP/170] 00:23:37, localpref 100, from 1.255.255.0
                      AS path: 65400 I, validation-state: unverified
                    > to 1.0.0.8 via xe-0/0/2.0
                      to 1.0.0.12 via xe-0/0/4.0
                    [BGP/170] 00:09:29, localpref 100, from 1.255.255.5
                      AS path: 65400 I, validation-state: unverified
                    > to 1.0.0.8 via xe-0/0/2.0
                      to 1.0.0.12 via xe-0/0/4.0
1:1.255.255.1:0::050000ff780000044c00::FFFF:FFFF/304
                   *[BGP/170] 00:23:33, localpref 100, from 1.255.255.1
                      AS path: 65400 I, validation-state: unverified
                    > to 1.0.0.8 via xe-0/0/2.0
                      to 1.0.0.12 via xe-0/0/4.0
                    [BGP/170] 00:09:29, localpref 100, from 1.255.255.5
                      AS path: 65400 I, validation-state: unverified
                    > to 1.0.0.8 via xe-0/0/2.0
                      to 1.0.0.12 via xe-0/0/4.0
```

We receive two unique advertisements for this ESI—one from Core-1 and the other from Core-2, with corresponding RDs. Since Leaf-1 is IBGP peered with Leaf-2, we'll also see the extra copy for each route from Leaf-2, with the original protocol-next-hop preserved.

This is Leaf-1's L2ALD table:

```
lab@leaf-1> show ethernet-switching table vlan-id 100

MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent
static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC, O -
ovsdb MAC)

Ethernet switching table : 5 entries, 5 learned
Routing instance : default-switch
   Vlan                MAC                  MAC      Logical              Active
   name                address              flags   interface             source
   v100                00:00:5e:00:01:01    DR       esi.1727
05:00:00:ff:78:00:00:04:4c:00
   v100                00:21:59:c8:24:63    DL       ae0.0
   v100                00:21:59:c8:24:69    D        vtep.32771           1.255.255.5
   v100                40:a6:77:9a:43:f0    D        vtep.32769           1.255.255.0
   v100                40:a6:77:9a:47:f0    D        vtep.32770           1.255.255.1
```

And this is Leaf-1's kernel table:

```
lab@leaf-1> show route forwarding-table family ethernet-switching
<output omitted>

Routing table: default-switch.bridge
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            perm     0                    dscd     1688     1
vtep.32769         intf     0                    comp     1724    19
vtep.32770         intf     0                    comp     1737    15
vtep.32771         intf     0                    comp     1738     9
<output omitted>

Routing table: default-switch.bridge
Bridging domain: v100.bridge
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
00:00:5e:00:01:01/48 user     0                  comp     1724    19
<output omitted>
```

Here we see that Leaf-1 has chosen index 1724, correlating with vtep.32769, to forward all traffic for the anycast gateway on VNI 1100.

What is vtep.32769?

```
lab@leaf-1> show ethernet-switching vxlan-tunnel-end-point esi
<output omitted>
05:00:00:ff:78:00:00:04:4c:00 default-switch          1727  131074  esi.1727
2
```

```
RVTEP-IP            RVTEP-IFL      VENH     MASK-ID   FLAGS
1.255.255.1         vtep.32770     1737     1         2
1.255.255.0         vtep.32769     1724     0         2
```

Above, we can see that it is the VTEP to Core-1. This, again, is the behavior on QFX5100 in the initial EVPN+VXLAN Junos operating system release. A future release will load-balance on the unilist as observed on the MX Series routers in the previous example.

Full Configurations

Leaf-1

```
set system host-name leaf-1
set chassis aggregated-devices ethernet device-count 2
set interfaces xe-0/0/2 unit 0 family inet address 1.0.0.9/31
set interfaces xe-0/0/4 unit 0 family inet address 1.0.0.13/31
set interfaces xe-0/0/32 ether-options 802.3ad ae0
set interfaces xe-0/0/33 ether-options 802.3ad ae1
set interfaces xe-0/0/34 unit 0 family ethernet-switching interface-mode trunk
set interfaces xe-0/0/34 unit 0 family ethernet-switching vlan members v300
set interfaces xe-0/0/34 unit 0 family ethernet-switching vlan members v400
set interfaces ae0 esi 00:01:01:01:01:01:01:01:01:01
set interfaces ae0 esi all-active
set interfaces ae0 aggregated-ether-options lacp active
set interfaces ae0 aggregated-ether-options lacp system-id 00:00:00:01:01:01
set interfaces ae0 unit 0 family ethernet-switching interface-mode access
set interfaces ae0 unit 0 family ethernet-switching vlan members v100
set interfaces ae1 esi 00:02:02:02:02:02:02:02:02:02
set interfaces ae1 esi all-active
set interfaces ae1 aggregated-ether-options lacp active
set interfaces ae1 aggregated-ether-options lacp system-id 00:00:00:01:01:01
set interfaces ae1 unit 0 family ethernet-switching interface-mode access
set interfaces ae1 unit 0 family ethernet-switching vlan members v200
set interfaces lo0 unit 0 family inet address 1.255.255.4/32
set routing-options router-id 1.255.255.4
set routing-options autonomous-system 65402
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay type external
set protocols bgp group underlay advertise-peer-as
set protocols bgp group underlay family inet unicast loops 2
set protocols bgp group underlay export lo0
set protocols bgp group underlay peer-as 65401
set protocols bgp group underlay multipath
set protocols bgp group underlay neighbor 1.0.0.8 description spine-1
set protocols bgp group underlay neighbor 1.0.0.12 description spine-2
set protocols bgp group EVPN_VXLAN_CORE type external
set protocols bgp group EVPN_VXLAN_CORE multihop ttl 255
set protocols bgp group EVPN_VXLAN_CORE multihop no-nexthop-change
set protocols bgp group EVPN_VXLAN_CORE local-address 1.255.255.4
set protocols bgp group EVPN_VXLAN_CORE family evpn signaling
set protocols bgp group EVPN_VXLAN_CORE peer-as 65400
set protocols bgp group EVPN_VXLAN_CORE local-as 65403
set protocols bgp group EVPN_VXLAN_CORE neighbor 1.255.255.0 description core-1
set protocols bgp group EVPN_VXLAN_CORE neighbor 1.255.255.1 description core-2
set protocols bgp group EVPN_VXLAN_LEAF type internal
set protocols bgp group EVPN_VXLAN_LEAF local-address 1.255.255.4
set protocols bgp group EVPN_VXLAN_LEAF family evpn signaling
set protocols bgp group EVPN_VXLAN_LEAF export LEAF-PREPEND
set protocols bgp group EVPN_VXLAN_LEAF neighbor 1.255.255.5 description leaf-2
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 1100
set protocols evpn extended-vni-list 1200
set protocols evpn extended-vni-list 1300
set protocols evpn extended-vni-list 1400
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-routing-options vni 1100 vrf-target export target:1:100
set protocols evpn vni-routing-options vni 1200 vrf-target export target:1:200
set protocols evpn vni-routing-options vni 1300 vrf-target export target:1:300
set protocols evpn vni-routing-options vni 1400 vrf-target export target:1:400
set protocols l2-learning traceoptions file l2ald.log
set protocols l2-learning traceoptions file size 10m
set protocols l2-learning traceoptions level all
set protocols l2-learning traceoptions flag all
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement LEAF-PREPEND then as-path-prepend 65402
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-
range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-
packet
set policy-options policy-statement vrf-imp term t1 from community com100
set policy-options policy-statement vrf-imp term t1 then accept
set policy-options policy-statement vrf-imp term t2 from community com200
set policy-options policy-statement vrf-imp term t2 then accept
set policy-options policy-statement vrf-imp term t3 from community com300
set policy-options policy-statement vrf-imp term t3 then accept
set policy-options policy-statement vrf-imp term t4 from community com400
set policy-options policy-statement vrf-imp term t4 then accept
set policy-options policy-statement vrf-imp term t5 then reject
set policy-options community com100 members target:1:100
set policy-options community com200 members target:1:200
set policy-options community com300 members target:1:300
set policy-options community com400 members target:1:400
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 1.255.255.4:1
set switch-options vrf-import vrf-imp
set switch-options vrf-target target:9999:9999
set vlans v100 vlan-id 100
set vlans v100 vxlan vni 1100
set vlans v100 vxlan ingress-node-replication
set vlans v200 vlan-id 200
set vlans v200 vxlan vni 1200
set vlans v200 vxlan ingress-node-replication
set vlans v300 vlan-id 300
set vlans v300 vxlan vni 1300
set vlans v300 vxlan ingress-node-replication
set vlans v400 vlan-id 400
set vlans v400 vxlan vni 1400
set vlans v400 vxlan ingress-node-replication
```

Leaf-2

```
set system host-name leaf-2
set chassis aggregated-devices ethernet device-count 2
set interfaces xe-0/0/3 unit 0 family inet address 1.0.0.11/31
set interfaces xe-0/0/5 unit 0 family inet address 1.0.0.15/31
set interfaces xe-0/0/36 ether-options 802.3ad ae0
set interfaces xe-0/0/37 ether-options 802.3ad ae1
set interfaces xe-0/0/38 unit 0 family ethernet-switching interface-mode trunk
set interfaces xe-0/0/38 unit 0 family ethernet-switching vlan members v300
set interfaces xe-0/0/38 unit 0 family ethernet-switching vlan members v100
set interfaces ae0 esi 00:01:01:01:01:01:01:01:01:01
set interfaces ae0 esi all-active
set interfaces ae0 aggregated-ether-options lacp passive
set interfaces ae0 aggregated-ether-options lacp system-id 00:00:00:01:01:01
set interfaces ae0 unit 0 family ethernet-switching interface-mode access
set interfaces ae0 unit 0 family ethernet-switching vlan members v100
set interfaces ae1 esi 00:02:02:02:02:02:02:02:02:02
set interfaces ae1 esi all-active
set interfaces ae1 aggregated-ether-options lacp passive
set interfaces ae1 aggregated-ether-options lacp system-id 00:00:00:01:01:01
set interfaces ae1 unit 0 family ethernet-switching interface-mode access
set interfaces ae1 unit 0 family ethernet-switching vlan members v200
set interfaces lo0 unit 0 family inet address 1.255.255.5/32
set routing-options router-id 1.255.255.5
set routing-options autonomous-system 65402
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay type external
set protocols bgp group underlay advertise-peer-as
set protocols bgp group underlay family inet unicast loops 2
set protocols bgp group underlay export lo0
set protocols bgp group underlay peer-as 65401
set protocols bgp group underlay multipath
set protocols bgp group underlay neighbor 1.0.0.10 description spine-1
set protocols bgp group underlay neighbor 1.0.0.14 description spine-2
set protocols bgp group EVPN_VXLAN_CORE type external
set protocols bgp group EVPN_VXLAN_CORE multihop ttl 255
set protocols bgp group EVPN_VXLAN_CORE multihop no-nexthop-change
set protocols bgp group EVPN_VXLAN_CORE local-address 1.255.255.5
set protocols bgp group EVPN_VXLAN_CORE family evpn signaling
set protocols bgp group EVPN_VXLAN_CORE peer-as 65400
set protocols bgp group EVPN_VXLAN_CORE neighbor 1.255.255.0 description core-1
set protocols bgp group EVPN_VXLAN_CORE neighbor 1.255.255.1 description core-2
set protocols bgp group EVPN_VXLAN_LEAF type internal
set protocols bgp group EVPN_VXLAN_LEAF local-address 1.255.255.5
set protocols bgp group EVPN_VXLAN_LEAF family evpn signaling
set protocols bgp group EVPN_VXLAN_LEAF export LEAF-PREPEND
set protocols bgp group EVPN_VXLAN_LEAF neighbor 1.255.255.4 description leaf-1
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 1100
set protocols evpn extended-vni-list 1200
set protocols evpn extended-vni-list 1300
set protocols evpn extended-vni-list 1400
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-routing-options vni 1100 vrf-target export target:1:100
set protocols evpn vni-routing-options vni 1200 vrf-target export target:1:200
set protocols evpn vni-routing-options vni 1300 vrf-target export target:1:300
set protocols evpn vni-routing-options vni 1400 vrf-target export target:1:400
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement LEAF-PREPEND then as-path-prepend 65402
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-
range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-
packet
set policy-options policy-statement vrf-imp term t1 from community com100
set policy-options policy-statement vrf-imp term t1 then accept
set policy-options policy-statement vrf-imp term t2 from community com200
set policy-options policy-statement vrf-imp term t2 then accept
set policy-options policy-statement vrf-imp term t3 from community com300
set policy-options policy-statement vrf-imp term t3 then accept
set policy-options policy-statement vrf-imp term t4 from community com400
set policy-options policy-statement vrf-imp term t4 then accept
set policy-options policy-statement vrf-imp term t5 then reject
set policy-options community com100 members target:1:100
set policy-options community com200 members target:1:200
set policy-options community com300 members target:1:300
set policy-options community com400 members target:1:400
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 1.255.255.5:1
set switch-options vrf-import vrf-imp
set switch-options vrf-target target:9999:9999
set vlans v100 vlan-id 100
set vlans v100 vxlan vni 1100
set vlans v100 vxlan ingress-node-replication
set vlans v200 vlan-id 200
set vlans v200 vxlan vni 1200
set vlans v200 vxlan ingress-node-replication
set vlans v300 vlan-id 300
set vlans v300 vxlan vni 1300
set vlans v300 vxlan ingress-node-replication
set vlans v400 vlan-id 400
set vlans v400 vxlan vni 1400
set vlans v400 vxlan ingress-node-replication
```

Spine-1

```
set system host-name spine-1
set interfaces xe-0/0/0 unit 0 family inet address 1.0.0.1/31
set interfaces xe-0/0/1 unit 0 family inet address 1.0.0.5/31
set interfaces xe-0/0/2 unit 0 family inet address 1.0.0.8/31
set interfaces xe-0/0/3 unit 0 family inet address 1.0.0.10/31
set interfaces lo0 unit 0 family inet address 1.255.255.2/32
set routing-options router-id 1.255.255.2
set routing-options autonomous-system 65401
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay-leaf type external
set protocols bgp group underlay-leaf advertise-peer-as
set protocols bgp group underlay-leaf family inet unicast loops 2
set protocols bgp group underlay-leaf export lo0
set protocols bgp group underlay-leaf peer-as 65402
set protocols bgp group underlay-leaf multipath
set protocols bgp group underlay-leaf neighbor 1.0.0.9 description leaf-1
set protocols bgp group underlay-leaf neighbor 1.0.0.11 description leaf-2
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core advertise-peer-as
set protocols bgp group underlay-core family inet unicast loops 2
set protocols bgp group underlay-core export lo0
set protocols bgp group underlay-core peer-as 65400
set protocols bgp group underlay-core multipath
set protocols bgp group underlay-core neighbor 1.0.0.0 description core-1
set protocols bgp group underlay-core neighbor 1.0.0.4 description core-2
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-packet
```

Spine-2

```
set system host-name spine-2
set interfaces xe-0/0/0 unit 0 family inet address 1.0.0.3/31
set interfaces xe-0/0/1 unit 0 family inet address 1.0.0.7/31
set interfaces xe-0/0/4 unit 0 family inet address 1.0.0.12/31
set interfaces xe-0/0/5 unit 0 family inet address 1.0.0.14/31
set interfaces lo0 unit 0 family inet address 1.255.255.3/32
set routing-options router-id 1.255.255.3
set routing-options autonomous-system 65401
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay-leaf type external
set protocols bgp group underlay-leaf advertise-peer-as
set protocols bgp group underlay-leaf family inet unicast loops 2
set protocols bgp group underlay-leaf export lo0
set protocols bgp group underlay-leaf peer-as 65402
set protocols bgp group underlay-leaf multipath
set protocols bgp group underlay-leaf neighbor 1.0.0.13 description leaf-1
set protocols bgp group underlay-leaf neighbor 1.0.0.15 description leaf-2
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core advertise-peer-as
set protocols bgp group underlay-core family inet unicast loops 2
set protocols bgp group underlay-core export lo0
set protocols bgp group underlay-core peer-as 65400
set protocols bgp group underlay-core multipath
set protocols bgp group underlay-core neighbor 1.0.0.2 description core-1
set protocols bgp group underlay-core neighbor 1.0.0.6 description core-2
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-
range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-
packet
```

Core-1

```
set system host-name core-1
set interfaces ge-1/0/0 unit 0 family inet address 1.0.0.0/31
set interfaces ge-1/0/1 unit 0 family inet address 1.0.0.2/31
set interfaces irb unit 1100 family inet address 100.0.0.2/24 virtual-gateway-
address 100.0.0.1
set interfaces irb unit 1200 family inet address 200.0.0.2/24 virtual-gateway-
address 200.0.0.1
set interfaces irb unit 1300 family inet address 10.10.10.2/24 virtual-gateway-
address 10.10.10.1
set interfaces irb unit 1400 family inet address 10.10.10.2/24 virtual-gateway-
address 10.10.10.1
```

```
set interfaces lo0 unit 0 family inet address 1.255.255.0/32
set routing-options router-id 1.255.255.0
set routing-options autonomous-system 65400
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay type external
set protocols bgp group underlay advertise-peer-as
set protocols bgp group underlay family inet unicast loops 2
set protocols bgp group underlay export lo0
set protocols bgp group underlay peer-as 65401
set protocols bgp group underlay multipath
set protocols bgp group underlay neighbor 1.0.0.1 description spine-1
set protocols bgp group underlay neighbor 1.0.0.3 description spine-2
set protocols bgp group EVPN_VXLAN type external
set protocols bgp group EVPN_VXLAN local-address 1.255.255.0
set protocols bgp group EVPN_VXLAN family evpn signaling
set protocols bgp group EVPN_VXLAN peer-as 65402
set protocols bgp group EVPN_VXLAN multipath
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.4 description leaf-1
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.4 multihop ttl 255
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.5 description leaf-2
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.5 multihop ttl 255
set protocols bgp group EVPN_VXLAN_TEMP type external
set protocols bgp group EVPN_VXLAN_TEMP local-address 1.255.255.0
set protocols bgp group EVPN_VXLAN_TEMP family evpn signaling
set protocols bgp group EVPN_VXLAN_TEMP peer-as 65403
set protocols bgp group EVPN_VXLAN_TEMP multipath
set protocols bgp group EVPN_VXLAN_TEMP neighbor 1.255.255.4 description leaf-1
set protocols bgp group EVPN_VXLAN_TEMP neighbor 1.255.255.4 multihop ttl 255
set protocols bgp group EVPN_VXLAN_TEMP neighbor 1.255.255.5 description leaf-2
set protocols bgp group EVPN_VXLAN_TEMP neighbor 1.255.255.5 multihop ttl 255
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement VS_VLAN100_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN100_IMP term ESI then accept
set policy-options policy-statement VS_VLAN100_IMP term VS_VLAN100 from community
comm-VS_VLAN100
set policy-options policy-statement VS_VLAN100_IMP term VS_VLAN100 then accept
set policy-options policy-statement VS_VLAN200_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN200_IMP term ESI then accept
set policy-options policy-statement VS_VLAN200_IMP term VS_VLAN200 from community
comm-VS_VLAN200
set policy-options policy-statement VS_VLAN200_IMP term VS_VLAN200 then accept
set policy-options policy-statement VS_VLAN300_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN300_IMP term ESI then accept
set policy-options policy-statement VS_VLAN300_IMP term VS_VLAN300 from community
```

```
comm-VS_VLAN300
set policy-options policy-statement VS_VLAN300_IMP term VS_VLAN300 then accept
set policy-options policy-statement VS_VLAN400_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN400_IMP term ESI then accept
set policy-options policy-statement VS_VLAN400_IMP term VS_VLAN400 from community
comm-VS_VLAN400
set policy-options policy-statement VS_VLAN400_IMP term VS_VLAN400 then accept
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-
range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-
packet
set policy-options community comm-VS_VLAN100 members target:1:100
set policy-options community comm-VS_VLAN200 members target:1:200
set policy-options community comm-VS_VLAN300 members target:1:300
set policy-options community comm-VS_VLAN400 members target:1:400
set policy-options community comm-leaf_esi members target:9999:9999
set routing-instances VRF_Tenant_A instance-type vrf
set routing-instances VRF_Tenant_A interface irb.1100
set routing-instances VRF_Tenant_A route-distinguisher 1.255.255.0:1100
set routing-instances VRF_Tenant_A vrf-target target:10:100
set routing-instances VRF_Tenant_A routing-options auto-export
set routing-instances VRF_Tenant_B instance-type vrf
set routing-instances VRF_Tenant_B interface irb.1200
set routing-instances VRF_Tenant_B route-distinguisher 1.255.255.0:1200
set routing-instances VRF_Tenant_B vrf-target target:10:200
set routing-instances VRF_Tenant_C instance-type vrf
set routing-instances VRF_Tenant_C interface irb.1300
set routing-instances VRF_Tenant_C route-distinguisher 1.255.255.0:1300
set routing-instances VRF_Tenant_C vrf-target target:10:300
set routing-instances VRF_Tenant_D instance-type vrf
set routing-instances VRF_Tenant_D interface irb.1400
set routing-instances VRF_Tenant_D route-distinguisher 1.255.255.0:1400
set routing-instances VRF_Tenant_D vrf-target target:10:400
set routing-instances VS_VLAN100 vtep-source-interface lo0.0
set routing-instances VS_VLAN100 instance-type virtual-switch
set routing-instances VS_VLAN100 route-distinguisher 1.255.255.0:100
set routing-instances VS_VLAN100 vrf-import VS_VLAN100_IMP
set routing-instances VS_VLAN100 vrf-target target:1:100
set routing-instances VS_VLAN100 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN100 protocols evpn extended-vni-list 1100
set routing-instances VS_VLAN100 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN100 bridge-domains bd1100 vlan-id 100
set routing-instances VS_VLAN100 bridge-domains bd1100 routing-interface irb.1100
set routing-instances VS_VLAN100 bridge-domains bd1100 vxlan vni 1100
set routing-instances VS_VLAN100 bridge-domains bd1100 vxlan ingress-node-
```

```
replication
set routing-instances VS_VLAN200 vtep-source-interface lo0.0
set routing-instances VS_VLAN200 instance-type virtual-switch
set routing-instances VS_VLAN200 route-distinguisher 1.255.255.0:200
set routing-instances VS_VLAN200 vrf-import VS_VLAN200_IMP
set routing-instances VS_VLAN200 vrf-target target:1:200
set routing-instances VS_VLAN200 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN200 protocols evpn extended-vni-list 1200
set routing-instances VS_VLAN200 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN200 bridge-domains bd1200 vlan-id 200
set routing-instances VS_VLAN200 bridge-domains bd1200 routing-interface irb.1200
set routing-instances VS_VLAN200 bridge-domains bd1200 vxlan vni 1200
set routing-instances VS_VLAN200 bridge-domains bd1200 vxlan ingress-node-replication
set routing-instances VS_VLAN300 vtep-source-interface lo0.0
set routing-instances VS_VLAN300 instance-type virtual-switch
set routing-instances VS_VLAN300 route-distinguisher 1.255.255.0:300
set routing-instances VS_VLAN300 vrf-import VS_VLAN300_IMP
set routing-instances VS_VLAN300 vrf-target target:1:300
set routing-instances VS_VLAN300 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN300 protocols evpn extended-vni-list 1300
set routing-instances VS_VLAN300 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN300 bridge-domains bd1300 vlan-id 300
set routing-instances VS_VLAN300 bridge-domains bd1300 routing-interface irb.1300
set routing-instances VS_VLAN300 bridge-domains bd1300 vxlan vni 1300
set routing-instances VS_VLAN300 bridge-domains bd1300 vxlan ingress-node-replication
set routing-instances VS_VLAN400 vtep-source-interface lo0.0
set routing-instances VS_VLAN400 instance-type virtual-switch
set routing-instances VS_VLAN400 route-distinguisher 1.255.255.0:400
set routing-instances VS_VLAN400 vrf-import VS_VLAN400_IMP
set routing-instances VS_VLAN400 vrf-target target:1:400
set routing-instances VS_VLAN400 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN400 protocols evpn extended-vni-list 1400
set routing-instances VS_VLAN400 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN400 bridge-domains bd1400 vlan-id 400
set routing-instances VS_VLAN400 bridge-domains bd1400 routing-interface irb.1400
set routing-instances VS_VLAN400 bridge-domains bd1400 vxlan vni 1400
set routing-instances VS_VLAN400 bridge-domains bd1400 vxlan ingress-node-replication
```

Core-2

```
set system host-name core-2
set chassis network-services enhanced-ip
set interfaces ge-1/0/0 unit 0 family inet address 1.0.0.4/31
set interfaces ge-1/0/1 unit 0 family inet address 1.0.0.6/31
set interfaces irb unit 1100 family inet address 100.0.0.3/24 virtual-gateway-address 100.0.0.1
set interfaces irb unit 1200 family inet address 200.0.0.3/24 virtual-gateway-
```

```
address 200.0.0.1
set interfaces irb unit 1300 family inet address 10.10.10.3/24 virtual-gateway-
address 10.10.10.1
set interfaces irb unit 1400 family inet address 10.10.10.3/24 virtual-gateway-
address 10.10.10.1
set interfaces lo0 unit 0 family inet address 1.255.255.1/32
set routing-options router-id 1.255.255.1
set routing-options autonomous-system 65400
set routing-options forwarding-table export load-balance
set routing-options forwarding-table ecmp-fast-reroute
set protocols bgp group underlay type external
set protocols bgp group underlay advertise-peer-as
set protocols bgp group underlay family inet unicast loops 2
set protocols bgp group underlay export lo0
set protocols bgp group underlay peer-as 65401
set protocols bgp group underlay multipath
set protocols bgp group underlay neighbor 1.0.0.5 description spine-1
set protocols bgp group underlay neighbor 1.0.0.7 description spine-2
set protocols bgp group EVPN_VXLAN type external
set protocols bgp group EVPN_VXLAN local-address 1.255.255.1
set protocols bgp group EVPN_VXLAN family evpn signaling
set protocols bgp group EVPN_VXLAN peer-as 65402
set protocols bgp group EVPN_VXLAN multipath
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.4 description leaf-1
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.4 multihop ttl 255
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.5 description leaf-2
set protocols bgp group EVPN_VXLAN neighbor 1.255.255.5 multihop ttl 255
set protocols lldp port-id-subtype interface-name
set protocols lldp interface all
set policy-options policy-statement VS_VLAN100_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN100_IMP term ESI then accept
set policy-options policy-statement VS_VLAN100_IMP term VS_VLAN100 from community
comm-VS_VLAN100
set policy-options policy-statement VS_VLAN100_IMP term VS_VLAN100 then accept
set policy-options policy-statement VS_VLAN200_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN200_IMP term ESI then accept
set policy-options policy-statement VS_VLAN200_IMP term VS_VLAN200 from community
comm-VS_VLAN200
set policy-options policy-statement VS_VLAN200_IMP term VS_VLAN200 then accept
set policy-options policy-statement VS_VLAN300_IMP term ESI from community comm-
leaf_esi
set policy-options policy-statement VS_VLAN300_IMP term ESI then accept
set policy-options policy-statement VS_VLAN300_IMP term VS_VLAN300 from community
comm-VS_VLAN300
set policy-options policy-statement VS_VLAN300_IMP term VS_VLAN300 then accept
set policy-options policy-statement VS_VLAN400_IMP term ESI from community comm-
leaf_esi
```

```
set policy-options policy-statement VS_VLAN400_IMP term ESI then accept
set policy-options policy-statement VS_VLAN400_IMP term VS_VLAN400 from community
comm-VS_VLAN400
set policy-options policy-statement VS_VLAN400_IMP term VS_VLAN400 then accept
set policy-options policy-statement lo0 from family inet
set policy-options policy-statement lo0 from protocol direct
set policy-options policy-statement lo0 from route-filter 0.0.0.0/0 prefix-length-
range /32-/32
set policy-options policy-statement lo0 then accept
set policy-options policy-statement load-balance term 1 then load-balance per-
packet
set policy-options community comm-VS_VLAN100 members target:1:100
set policy-options community comm-VS_VLAN200 members target:1:200
set policy-options community comm-VS_VLAN300 members target:1:300
set policy-options community comm-VS_VLAN400 members target:1:400
set policy-options community comm-leaf_esi members target:9999:9999
set routing-instances VRF_Tenant_A instance-type vrf
set routing-instances VRF_Tenant_A interface irb.1100
set routing-instances VRF_Tenant_A route-distinguisher 1.255.255.1:1100
set routing-instances VRF_Tenant_A vrf-target target:10:100
set routing-instances VRF_Tenant_A routing-options auto-export
set routing-instances VRF_Tenant_B instance-type vrf
set routing-instances VRF_Tenant_B interface irb.1200
set routing-instances VRF_Tenant_B route-distinguisher 1.255.255.1:1200
set routing-instances VRF_Tenant_B vrf-target target:10:200
set routing-instances VRF_Tenant_C instance-type vrf
set routing-instances VRF_Tenant_C interface irb.1300
set routing-instances VRF_Tenant_C route-distinguisher 1.255.255.1:1300
set routing-instances VRF_Tenant_C vrf-target target:10:300
set routing-instances VRF_Tenant_D instance-type vrf
set routing-instances VRF_Tenant_D interface irb.1400
set routing-instances VRF_Tenant_D route-distinguisher 1.255.255.1:1400
set routing-instances VRF_Tenant_D vrf-target target:10:400
set routing-instances VS_VLAN100 vtep-source-interface lo0.0
set routing-instances VS_VLAN100 instance-type virtual-switch
set routing-instances VS_VLAN100 route-distinguisher 1.255.255.1:100
set routing-instances VS_VLAN100 vrf-import VS_VLAN100_IMP
set routing-instances VS_VLAN100 vrf-target target:1:100
set routing-instances VS_VLAN100 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN100 protocols evpn extended-vni-list 1100
set routing-instances VS_VLAN100 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN100 protocols evpn default-gateway no-gateway-
community
set routing-instances VS_VLAN100 bridge-domains bd1100 vlan-id 100
set routing-instances VS_VLAN100 bridge-domains bd1100 routing-interface irb.1100
set routing-instances VS_VLAN100 bridge-domains bd1100 vxlan vni 1100
set routing-instances VS_VLAN100 bridge-domains bd1100 vxlan ingress-node-
replication
set routing-instances VS_VLAN200 vtep-source-interface lo0.0
set routing-instances VS_VLAN200 instance-type virtual-switch
set routing-instances VS_VLAN200 route-distinguisher 1.255.255.1:200
set routing-instances VS_VLAN200 vrf-import VS_VLAN200_IMP
set routing-instances VS_VLAN200 vrf-target target:1:200
set routing-instances VS_VLAN200 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN200 protocols evpn extended-vni-list 1200
set routing-instances VS_VLAN200 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN200 bridge-domains bd1200 vlan-id 200
set routing-instances VS_VLAN200 bridge-domains bd1200 routing-interface irb.1200
set routing-instances VS_VLAN200 bridge-domains bd1200 vxlan vni 1200
set routing-instances VS_VLAN200 bridge-domains bd1200 vxlan ingress-node-replication
set routing-instances VS_VLAN300 vtep-source-interface lo0.0
set routing-instances VS_VLAN300 instance-type virtual-switch
set routing-instances VS_VLAN300 route-distinguisher 1.255.255.1:300
set routing-instances VS_VLAN300 vrf-import VS_VLAN300_IMP
set routing-instances VS_VLAN300 vrf-target target:1:300
set routing-instances VS_VLAN300 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN300 protocols evpn extended-vni-list 1300
set routing-instances VS_VLAN300 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN300 bridge-domains bd1300 vlan-id 300
set routing-instances VS_VLAN300 bridge-domains bd1300 routing-interface irb.1300
set routing-instances VS_VLAN300 bridge-domains bd1300 vxlan vni 1300
set routing-instances VS_VLAN300 bridge-domains bd1300 vxlan ingress-node-replication
set routing-instances VS_VLAN400 vtep-source-interface lo0.0
set routing-instances VS_VLAN400 instance-type virtual-switch
set routing-instances VS_VLAN400 route-distinguisher 1.255.255.1:400
set routing-instances VS_VLAN400 vrf-import VS_VLAN400_IMP
set routing-instances VS_VLAN400 vrf-target target:1:400
set routing-instances VS_VLAN400 protocols evpn encapsulation vxlan
set routing-instances VS_VLAN400 protocols evpn extended-vni-list 1400
set routing-instances VS_VLAN400 protocols evpn multicast-mode ingress-replication
set routing-instances VS_VLAN400 bridge-domains bd1400 vlan-id 400
set routing-instances VS_VLAN400 bridge-domains bd1400 routing-interface irb.1400
set routing-instances VS_VLAN400 bridge-domains bd1400 vxlan vni 1400
set routing-instances VS_VLAN400 bridge-domains bd1400 vxlan ingress-node-replication
```

Conclusion

Cloud-based resources are becoming an increasingly large part of the enterprise's IT strategy, requiring a network architecture that can accommodate cloud-based services without compromising security or performance. Also, the demands of data center users for anytime, anywhere access and high levels of responsiveness are becoming harder and harder to achieve with today's network architectures.

Ethernet VPN (EVPN) is an efficient and scalable way to build and interconnect data center networks. For the first time, without stitching multiple control planes together, customers can build end-to-end networks with a unified control plane in a scalable manner due to EVPN's ability to work across multiple data planes (VXLAN and MPLS). EVPN also comprehensively solves some of the challenges associated with workload mobility, optimized routing, as well as network efficiencies with multipathing. With a robust BGP/EVPN implementation on all platforms—QFX Series switches, EX Series switches, and MX Series routers—Juniper is uniquely positioned to bring EVPN technology to its full potential by providing optimized, seamless, and standards-compliant L2 or L3 connectivity, both within and across today's evolving data centers.

About Juniper Networks

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

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

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