Engineering
Simplicity

Network Configuration Example

Campus Fabric IP Clos Using Junos OS CLI Workflow

Published
2024-9-18

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

END USER LICENSE AGREEMENT

Table of Contents

CHAPTER 1 Campus Fabric IP Clos Using Junos OS CLI Workflow

About This Configuration Example

Scope

Use this network configuration example (NCE) for building Juniper's Campus Fabric IP Clos using the Junos OS CLI workflow.

The goal of this NCE is to permit communication with all end devices (desktops and APs) and permit those end devices to access hosts on the Internet through the WAN router. Note that all inter-VRF communication must go through the WAN device, as the WAN device is a security device that can secure this inter-VRF traffic.

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation.
Send your comments to design-center-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Technology Primer: Campus Fabric IP Clos

Use Case Overview

Enterprise networks are undergoing massive transitions to accommodate the growing demand for cloud-ready, scalable, and efficient networks and the plethora of Internet of Things (IoT) and mobile devices. As the number of devices grows, so does network complexity, with an ever-greater need for scalability, segmentation, and security. To meet these challenges, you need a network with Automation and Artificial Intelligence (AI) for operational simplification. IP Clos networks provide increased scalability and segmentation using a well-understood standards-based approach (EVPN-VXLAN with GBP).

Most traditional campus architectures use single-vendor, chassis-based technologies that work well in small, static campuses with few endpoints. However, they are too rigid to support the scalability and changing needs of modern large enterprises. Multi-Chassis Link Aggregation Group (MC-LAG) is a good example of a single-vendor technology that addresses the collapsed core deployment model. In this model, two chassis-based platforms are typically in the core of a customer's network; deployed to handle all Layer 2 and Layer 3 requirements while providing an active/backup resiliency environment. MC-LAG does not interoperate between vendors, creating lock-in, and is limited to two devices.

A Juniper Networks EVPN-VXLAN fabric is a highly scalable architecture that is simple, programmable, and built on a standards-based architecture (https://www.rfc-editor.org/rfc/rfc8365) that is common across campuses and data centers.

The Juniper campus architecture uses a Layer 3 IP-based underlay network and an EVPN-VXLAN overlay network. Broadcast, unknown unicast, and multicast (BUM) traffic, is handled natively by EVPN and eliminates the need for Spanning Tree Protocols (STP/RSTP). A flexible overlay network based on VXLAN tunnels combined with an EVPN control plane efficiently provides Layer 3 or Layer 2 connectivity. This architecture decouples the virtual topology from the physical topology, which improves network flexibility and simplifies network management. Endpoints that require Layer 2 adjacency, such as IoT devices, can be placed anywhere in the network and remain connected to the same logical Layer 2 network.

With an EVPN-VXLAN campus architecture, you can easily add core, distribution, and access layer devices as your business grows without a need for redesigning the network. EVPN-VXLAN is vendor-agnostic, so you can use the existing access layer infrastructure and gradually migrate to access layer switches that support EVPN-VXLAN capabilities once the Core and Distribution part of the network is deployed. Connectivity with legacy switches that do not support EVPN-VXLAN is accomplished with standards-based ESI-LAG. ESI-LAG utilizes standards-based Link Aggregation Control Protocol (LACP) to interconnect with legacy switches.

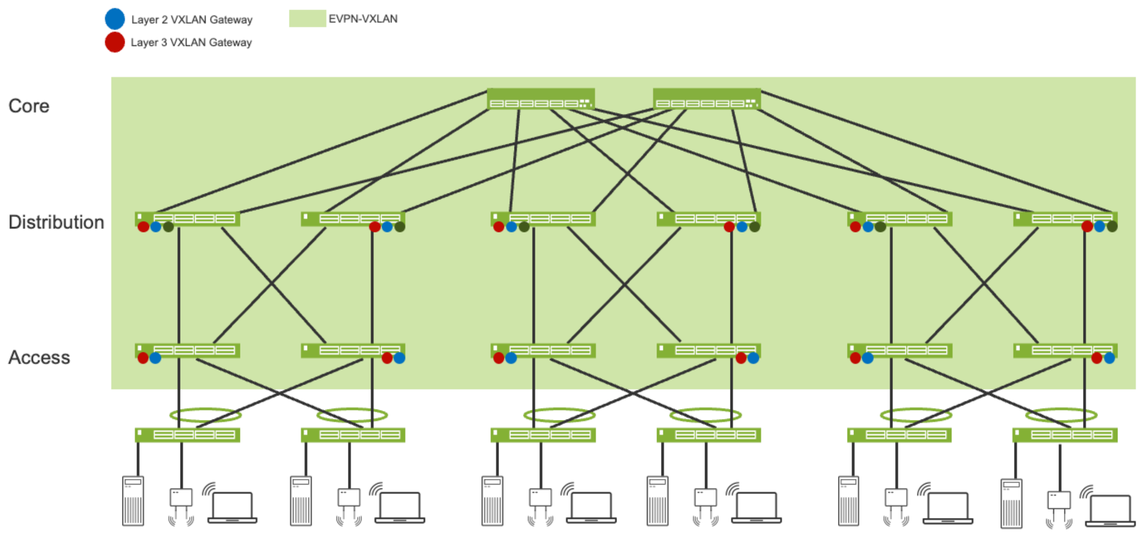
Benefits of Campus Fabric: IP Clos

With the increasing number of devices connecting to the network, you must scale your campus network rapidly without adding complexity. Many IoT devices have limited networking capabilities and require Layer 2 adjacency across buildings and campuses. Traditionally, this problem was solved by extending virtual LANs (VLANs) between endpoints using data plane-based flood and learning mechanisms inherent with Ethernet switching technologies. The traditional Ethernet switching approach is inefficient because it leverages broadcast and multicast technologies to announce Media Access Control (MAC) addresses. It is also difficult to manage because you need to manually configure VLANs to extend them to new network ports. This problem increases multi-fold when considering the explosive growth of mobile and IoT devices.

Campus fabrics have an underlay topology with a routing protocol that ensures loopback interface reachability between nodes. Devices participating in EVPN-VXLAN function as VXLAN tunnel endpoint (VTEP) that encapsulate and decapsulate the VXLAN traffic. VTEP represents the construct within the switching platform that originates and terminates VXLAN tunnels. In addition, these devices route and bridge packets in and out of VXLAN tunnels as required.

The campus fabric IP Clos extends the EVPN fabric to connect VLANs across multiple buildings or floors of a single building. This is done by stretching the Layer 2 VXLAN network with routing occurring in the access device instead of the core or distribution layers. IP Clos network encompasses the distribution, core, and access layers of your topology.

Figure 1: Campus Fabric IP Clos



An EVPN-VXLAN fabric solves the problems of previous architectures and provides the following benefits:

- Reduced flooding and learning—Control plane-based Layer 2 and Layer 3 learning reduce the flood and learn issues associated with data plane learning. Learning MAC addresses in the forwarding plane has an adverse impact on network performance as the number of endpoints grows. The EVPN control plane handles the exchange and learning of MAC addresses through eBGP routing, rather than a Layer 2 forwarding plane.
- Scalability—More efficient control plane based Layer 2 and Layer 3 learning. For example, in a Campus Fabric IP Clos, core switches only learn the access layer switches addresses instead of the device endpoint addresses.
- Consistency—A universal EVPN-VXLAN-based architecture across disparate campus and data center deployments enable a seamless end-to-end network for endpoints and applications.
- Group-Based Policies—With group-based policy (GBP), you can enable micro-segmentation with EVPN-VXLAN to provide traffic isolation within and between broadcast domains as well as simplify security policies across a Campus Fabric.
- Location-agnostic connectivity—The EVPN-VXLAN campus architecture provides a consistent endpoint experience no matter where the endpoint is located. Some endpoints require Layer 2 reachability, such as legacy building security systems or IoT devices. VXLAN overlay provides Layer 2 extension across campuses without any changes to the underlay network. Juniper uses optimal BGP timers between the adjacent layers of the Campus Fabric with Bidirectional Forwarding Detection (BFD) that supports fast convergence in case of a node or link failure and Equal cost multipath (ECMP). For more information, see https://www.juniper.net/documentation/us/en/software/junos/sampling-forwarding-monitoring/topics/concept/policy-configuring-per-packet-load-balancing.html

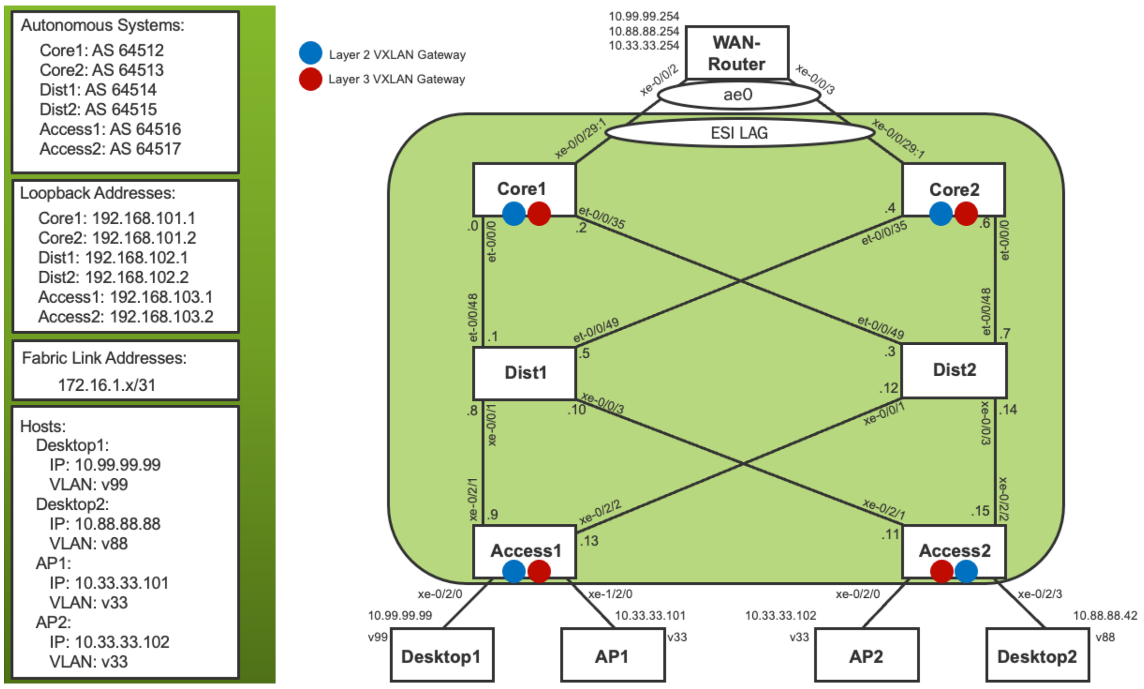
Campus Fabric IP Clos Components

This configuration example uses the following devices:

- Two EX4650 Switches as core devices, software version: Junos OS Release 21.4R1.12 or later
- Two QFX5120 Switches as distribution devices, software version: Junos OS Release 21.4R1.12 or later
- Two access layer EX4400 Switches, software version: Junos OS Release 22.1R1.10 or later
- One SRX345 WAN router, software version: 20.2R3-S2.5 or later
- Juniper Access Points
- Two Linux desktops that act as wired clients

NOTE: Advanced GBP microsegmentation features require Junos OS Release 22.4R1 and later.

Figure 2: Topology



Autonomous Systems:
Core1: AS 64512
Core2: AS 64513
Dist1: AS 64514
Dist2: AS 64515
Access1: AS 64516
Access2: AS 64517

Loopback Addresses:
Core1: 192.168.101.1
Core2: 192.168.101.2
Dist1: 192.168.102.1
Dist2: 192.168.102.2
Access1: 192.168.103.1
Access2: 192.168.103.2

Fabric Link Addresses:
172.16.1.x/31

Hosts:
Desktop1:
IP: 10.99.99.99
VLAN: v99
Desktop2:
IP: 10.88.88.88
VLAN: v88
AP1:
IP: 10.33.33.101
VLAN: v33
AP2:
IP: 10.33.33.102
VLAN: v33

By default, inter-VRF communications is not supported within the Campus Fabric. If inter-VRF communications is required, each VRF can include extra routes such as a Default Route that instructs the Campus Fabric to use an external router or firewall for further security inspection or routing capabilities. In this example, all traffic is trunked over the ESI-LAG and the SRX Series Firewall handles inter-VRF routing. See Figure 2.

SRX Series Firewall participates in the VLANs defined within the Campus Fabric and is the gateway of last resort for all traffic leaving the subnet. To this end, we must add a default route on the access switches to forward all traffic leaving a network, such as the 10.99.99.0/24 network, to utilize the next hop of the Juniper SRX Series Firewall, which for the 10.99.99.0/24 network is 10.99.99.254

CHAPTER 2 Configuring an EVPN-VXLAN Campus Fabric Using the Junos OS CLI

Overview

Configuring an EVPN-VXLAN network for the campus with the Junos OS CLI consists of the following main parts and its configuration:

- Underlay network
- Overlay network
- EVPN protocol
- Switch options
- VLAN/VNI.

Let’s discuss each of these parts in detail throughout this document.

The access devices act as traditional leafs, the Distribution devices act as Spines, and the Core devices act as border leafs of an EVPN-VXLAN network. Throughout this document, we’ll reference these devices by their names but keep the roles of these devices in mind.

Underlay Configuration

The purpose of the underlay network is to provide connectivity for the overlay network, specifically to the loopback address of the network devices. To this end, we are using eBGP to provide that connectivity. We need to configure eBGP on the EX and QFX Switches. Then, we must configure an export policy that is applied to the underlay BGP groups that match on the loopback interfaces and export them into the network. These loopback addresses are then used for the BGP peerings that are required for the overlay network. You can use another routing protocol, such as OSPF or IS-IS, for the underlay network if necessary. However, eBGP is a great choice for the underlay network as it provides increased flexibility and route filtering capabilities.

The following output shows the eBGP underlay group for Access1. The BGP group peers with two neighbors, Distribution1 and Distribution2. The peering addresses for these neighbors are the directly connected interface IP addresses, which are reachable as direct routes. This type of reachability means there is no need to have another routing protocol, such as static routing, to reach the eBGP peers.

Each peer has a different autonomous system (AS) that is different from the locally configured AS. This configuration is important as we’re using eBGP, which is typically used to peer with different service provider networks.

Finally, we use the `multipath multiple-as` parameter, as there are multiple equal-cost paths to get to destinations in the underlay network. However, this parameter only ensures that multiple next hops can be used in the routing table. We need to apply a load-balancing policy to the forwarding table to ensure that multiple next hops can be used in the forwarding table.

NOTE: A similar configuration is needed for the rest of the network devices. However, we do not show it here, as the only difference is the neighbor and AS configurations.

```
{master:0}[edit protocols bgp group underlay]
lab@access1# show
type external;
export loopbacks;
local-as 64516;
multipath {
    multiple-as;
}
neighbor 172.16.1.8 {
    peer-as 64514;
}
neighbor 172.16.1.12 {
    peer-as 64515;
}

{master:0}[edit protocols bgp group underlay]
lab@access1# top show policy-options policy-statement loopbacks
from interface lo0.0;
then accept;
```

The following output shows that the eBGP underlay peers are established and sharing routes:

```
{master:0}
lab@access1> show bgp summary

Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 1 Peers: 2 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                      10          8          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
172.16.1.8            64514        494        498       0       0     3:43:52
Establ
  inet.0: 4/5/5/0
172.16.1.12           64515        498        504       0       0     3:43:33
Establ
  inet.0: 4/5/5/0
```

The following output shows the route for the loopback address of Core1. The routing table shows two next hops, but the forwarding table only shows one next hop.

```
{master:0}
lab@access1> show route 192.168.101.1

inet.0: 15 destinations, 20 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.101.1/32   *[BGP/170] 00:36:40, localpref 100, from 172.16.1.8
                      AS path: 64514 64512 I, validation-state: unverified
                       to 172.16.1.8 via xe-0/2/1.0
                    >  to 172.16.1.12 via xe-0/2/2.0
                    [BGP/170] 00:36:40, localpref 100
                      AS path: 64515 64512 I, validation-state: unverified
                    >  to 172.16.1.12 via xe-0/2/2.0

{master:0}
lab@access1> show route forwarding-table destination 192.168.101.1
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
192.168.101.1/32   user     0 172.16.1.12        ucst     1946     8 xe-0/2/2.0
```

You can see the difference if we apply a load-balancing policy to the forwarding table and examine it. The forwarding table now has two next hops. Note that the routing table remains unchanged.

```
{master:0}[edit]
lab@access1# show policy-options policy-statement load-balance
term 1 {
    then {
        load-balance per-packet;
    }
}

{master:0}[edit]
lab@access1# show routing-options
router-id 192.168.103.1;
forwarding-table {
    export load-balance;
}

{master:0}[edit]
lab@access1# ...ng-table destination 192.168.101.1
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
192.168.101.1/32   user     0                    ulst   131070     4
                              172.16.1.8         ucst     1945     6 xe-0/2/1.0
                              172.16.1.12        ucst     1946     6 xe-0/2/2.0
```

Overlay Configuration

The overlay configuration enables EVPN routes sharing through BGP. This EVPN routes sharing is important as it's how the location of end hosts is distributed through the network. For the overlay network, we must use BGP. We can use eBGP or iBGP. In this example, we use eBGP for the overlay network.

The following example shows the overlay eBGP configuration for Distribution1:

```
{master:0}[edit protocols bgp group overlay]
lab@adistribution1# show
type external;
multihop;
local-address 192.168.102.1;
family evpn {
    signaling;
}
local-as 64514;
multipath {
    multiple-as;
}
neighbor 192.168.101.1 {
    peer-as 64512;
}
neighbor 192.168.101.2{
    peer-as 64513;
}
neighbor 192.168.102.2{
    peer-as 64515;
}
neighbor 192.168.103.1{
    peer-as 64516;
}
neighbor 192.168.103.2 {
    peer-as 64517;
}
```

In the previous configuration example, the configuration for the overlay BGP group is shown here. We have set the group to signal the EVPN protocol family and any parameters that are necessary for the BGP sessions. The example shows the eBGP configuration with `family evpn signaling`. Just as with the underlay configuration, we must specify the local AS and the neighbor parameters.

This configuration example is the same for Distribution2 too. The only difference is that the `local-as` and `local-address` parameters are set to the Distribution2 local AS and loopback address. The overlay eBGP configuration for the core and access devices are similar. The overlay configuration for Access1 is shown for reference.

```
{master:0}[edit protocols bgp group overlay]
lab@access1# show
type external;
multihop;
local-address 192.168.102.1;
family evpn {
    signaling;
}
local-as 64516;
multipath {
    multiple-as;
}
neighbor 192.168.101.1 {
    peer-as 64512;
}
neighbor 192.168.101.2 {
    peer-as 64513;
}
neighbor 192.168.102.1 {
    peer-as 64514;
}
neighbor 192.168.102.2 {
    peer-as 64515;
}
neighbor 192.168.103.2 {
    peer-as 64517;
}
```

The following output shows the overlay group BGP sessions for Distribution1. No EVPN routes are being shared yet; however, you can see that the EVPN family is negotiated for the sessions as the `bgp.evpn.0` table is present for the sessions. Although it is not shown, the same is true for the other devices.

```
{master:0}
lab@distrubtion1> show bgp summary group overlay

Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 2 Peers: 9 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.evpn.0
               0          0          0          0          0          0
inet.0
               8          8          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
192.168.101.1         65412        787        823       0       0     4:36:10 Establ
  bgp.evpn.0: 0/0/0/0
192.168.101.2         65413        781        815       0       0     4:35:03 Establ
  bgp.evpn.0: 0/0/0/0
192.168.102.2         65515        993        994       0       0     4:37:43 Establ
  bgp.evpn.0: 0/0/0/0
192.168.103.1         65516         44        110       0       1       14:01 Establ
  bgp.evpn.0: 0/0/0/0
192.168.103.2         65517         46        112       0       1       14:07 Establ
  bgp.evpn.0: 0/0/0/0
```

The following output from Access1 shows that it has established overlay eBGP sessions with all other devices. Again, no EVPN routes are being shared, but the `bgp.evpn.0` table is present for the sessions.

```
{master:0}
lab@access1> show bgp summary group overlay

Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 2 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.evpn.0
               0          0          0          0          0          0
inet.0
               10         8          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
192.168.101.1         65412        787        823       0       0     4:36:15 Establ
  bgp.evpn.0: 0/0/0/0
192.168.101.2         65413        781        815       0       0     4:35:10 Establ
  bgp.evpn.0: 0/0/0/0
192.168.102.1         65514        993        994       0       0     4:37:50 Establ
  bgp.evpn.0: 0/0/0/0
192.168.102.2         65515         44        110       0       1       14:05 Establ
  bgp.evpn.0: 0/0/0/0
192.168.103.2         65517         46        112       0       1       14:09 Establ
  bgp.evpn.0: 0/0/0/0
```

EVPN-VXLAN Configuration

To advertise EVPN routes, a device must have all the required parameters for that route. The parameters that must be included in an EVPN route advertisement include both EVPN and VXLAN information. The EVPN information that is required includes the encapsulation type, the target communities associated with locally configured VNIs, and the VNIs to which the device is connected to. The `[edit protocols evpn]` hierarchy defines the BGP target communities associated with each VNI, the encapsulation type, and the list of VNIs that are supported locally.

The `[edit switch-options]` hierarchy identifies the source address of the VTEP tunnel, the route distinguisher that must be added to advertised route prefixes, the target community that adds to routes leaving the local switching table, and the target community that identifies which routes received from remote BGP peers are to be imported into the local VXLAN switching table.

A `vrf-export` statement adds the associated VRF target community value to the BGP updates that are sent. The VRF import policy evaluates received BGP routes, and if the VRF target community in the VRF import policy matches a community value in the route, the information from the route is imported into the local EVPN route table, and subsequently, the VXLAN switching table.

The `vrf-target` statement under the `[edit switch-options]` automatically creates hidden VRF import and VRF export policies for the specified community value. If a `vrf-target` statement is included for a VNI in the VNI option hierarchy, the `vni-options vrf-target` statement overrides the `switch-options vrf-target` value for Type 2 and Type 3 EVPN routes. The `switch-options vrf-target` statement is still applied automatically to Type 1 EVPN routes.

NOTE: If you do not configure [`edit protocols evpn vni-options]`, then all EVPN routes are sent and accepted regardless of the remote VTEP VNI settings.

We must configure the leafs (Access1, Access2, Core1, and Core2) with the necessary EVPN protocol parameters and switch options. The spines (Distribution1 and Distribution2) do not need this configuration as they act simply as transport switches that the VXLAN tunnels traverse.

As isolation through VRFs is necessary, we configure a VRF for each tenant. To this end, we must configure many of the EVPN-VXLAN related parameters within the VRFs. Recall that all inter-VRF traffic must go through the firewall.

The following output shows the EVPN protocol and switch options configuration for Access1 and Core1. Both devices use VXLAN encapsulation and accept all VNIs. Also the loopback interface as the VTEP source and the same VRF target too. The only difference is with the route distinguisher, which uniquely marks any routes originating from a specific leaf. These parameters assign Layer 2 gateway to these devices.

The routing instance configuration is also shown in the following output. Specifying a VRF instance type with an associated VRF target allows routes to be shared between similar VRFs on the access devices. For example, the v33 routing instance is configured on both access devices with the `vrf-target target:65512:33` statement. This statement connects those two VRFs together and allows the automatic sharing of routes between them. The same is true for the v88 and v99 routing instances.

The default route that is configured in each routing instance that points towards the WAN router. This default route is the gateway of last resort and allows inter-VRF communication to go through the WAN router, which is an SRX Series Firewall, for security processing.

You'll also notice that each routing instance has an IRB interface in it. These IRB interfaces, which we'll show the configuration later in the document, are the default gateways for the connected hosts, such as Desktop1 or AP1. The access devices having the IRB interfaces here means that they act as Layer 3 gateways. Also, having the IRB interfaces on the access devices enables ERB for the campus fabric. If you want to enable CRB for the Campus Fabric, then the IRB interfaces need to be on the distribution devices.

In this configuration, there are two sets of route distinguishers and VRF target parameters, which creates confusion. The parameters under the switch options enable the VXLAN tunnels to form between fabric devices. In contrast, the parameters under the routing instances enable the VRFs on the different access devices to share routes with each other. When we look at the core devices configurations later, you'll see that the VRF targets there match what is on the access devices under the `switch-options` configuration.

NOTE: Access2 has a similar configuration, with the only difference there being the unique route distinguishers.

```
{master:0}[edit]
lab@access1# show protocols evpn
encapsulation vxlan;
extended-vni-list all;

{master:0}[edit]
lab@access1# show switch-options
vtep-source-interface lo0.0;
route-distinguisher 192.168.103.1:1;
vrf-target target:65512:1;

{master:0}[edit]
lab@access1# show routing-instances
v33 {
    instance-type vrf;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.33.33.254;
        }
        multipath;
    }
    interface irb.33;
    route-distinguisher 192.168.103.1:33;
    vrf-target target:65512:33;
    vrf-table-label;
}
v88 {
    instance-type vrf;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.88.88.254;
        }
        multipath;
    }
    interface irb.88;
    route-distinguisher 192.168.103.1:88;
    vrf-target target:65512:88;
    vrf-table-label;
}
v99 {
    instance-type vrf;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.99.99.254;
        }
        multipath;
    }
    interface irb.99;
    route-distinguisher 192.168.103.1:99;
    vrf-target target:65512:99;
    vrf-table-label;
}
```

Core1 and Core2 only act as Layer 3 gateways, so the EVPN-VXLAN configuration differs from Access1 and Access2.

NOTE: Core2 has a similar configuration, with the only difference there being the unique route distinguishers.

NOTE: We can place the previous configuration for Core1 and Core2 in the main routing instance. However, we recommend placing the configuration in a virtual switch routing instance is best practice as it allows you to add other virtual switches in the future.

```
{master:0}[edit routing-instances]
lab@core1# show
evpn_vs {
    instance-type virtual-switch;
    protocols {
        evpn {
            encapsulation vxlan;
            extended-vni-list all;
        }
    }
    vtep-source-interface lo0.0;
    interface ae0.0;
    route-distinguisher 192.168.101.1:1;
    vrf-target target:65512:1;
    vlans {
        v33 {
            vlan-id 33;
            vxlan {
                vni 5033;
            }
        }
        v88 {
            vlan-id 88;
            vxlan {
                vni 5088;
            }
        }
        v99 {
            vlan-id 99;
            vxlan {
                vni 5099;
            }
        }
    }
}
```

Now, we can look at the overlay eBGP sessions and see that some new tables are added.

The `bgp.evpn.0` table, which we discussed earlier, is the main EVPN routing table where all the EVPN routes are stored. This table contains the routes for all the EVPN instances configured on the device. The routes in this table are used to provide Layer 2 and Layer 3 VPN services in a VXLAN environment.

The `default-switch.evpn.0` table is specific to the default-switch routing instance, which is an automatically created instance for the default virtual network. It is used to store EVPN routes specific to the default-switch instance. In a typical EVPN-VXLAN deployment, you have multiple virtual networks and routing instances, each with its own routing table. With this example, we configured the default switch instance under `[switch-options]` as our network is not overly complex.

The `default_evpn.evpn.0` table is an internal table used for EVPN route processing. It is not directly related to a specific EVPN instance or virtual network. The double underscores indicate that it's an internal table. You generally don't need to interact with this table, as it is mainly used by the system for internal route handling and processing.

NOTE: No routes are being shared yet as we've to configure any VLANs or interfaces that belong to those VLANs on the access devices.

```
{master:0}
lab@access1> show bgp summary group overlay

Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 2 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.evpn.0
                       0          0          0          0          0          0
inet.0
                      10          8          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
192.168.101.1         64512      154         86       0       0       33:31 Establ
  bgp.evpn.0: 0/0/0/0
  default-switch.evpn.0: 0/0/0/0
  __default_evpn__.evpn.0: 0/0/0/0
192.168.101.2         64513      131         83       0       0       33:33 Establ
  bgp.evpn.0: 0/0/0/0
  default-switch.evpn.0: 0/0/0/0
  __default_evpn__.evpn.0: 0/0/0/0
```

```
192.168.102.1          64514      152         84       0       0       33:35 Establ
  bgp.evpn.0: 0/0/0/0
  default-switch.evpn.0: 0/0/0/0
  __default_evpn__.evpn.0: 0/0/0/0
192.168.102.2          64515      134         86       0       0       33:39 Establ
  bgp.evpn.0: 0/0/0/0
  default-switch.evpn.0: 0/0/0/0
  __default_evpn__.evpn.0: 0/0/0/0
192.168.103.2          64517      150         80       0       0       33:44 Establ
  bgp.evpn.0: 0/0/0/0
  default-switch.evpn.0: 0/0/0/0
  __default_evpn__.evpn.0: 0/0/0/0
```

Interface and VLAN Configuration

Now, we must configure the interfaces and VLANs on the access devices. Recall that the WAN router is going to provide inter-VLAN communication as well as Internet-based connectivity.

The VLAN-to-VXLAN VNI mapping is configured under the `[edit vlans]` hierarchy. Once a VLAN-to-VNI mapping is performed and a local interface is configured with the corresponding VLAN, the device creates a local VTEP interface in the local `:vxlan.inet.0` routing table. The local VNI is then advertised to the remote peers, and the local device is advertised as a Layer 2 gateway to reach the VNI.

The following configuration shows VLAN stanza for Acess1. This configuration snippet shows that VLAN v33 uses VLAN ID 33, which is mapped to VXLAN VNI 5033. Also, the irb.33 interface is the Layer 3 interface. The VXLAN configuration turns this leaf into a Layer 2 gateway, and the Layer 3 interface turns it into a Layer 3 gateway. The configuration for VLANs v88 and v99 is similar. Access2 has the exact same configuration.

```
{master:0}[edit]
lab@access2# show vlans
v33 {
    vlan-id 33;
    l3-interface irb.33;
    vxlan {
        vni 5033;
    }
}
v88 {
    vlan-id 88;
    l3-interface irb.88;
    vxlan {
        vni 5088;
    }
}
v99 {
    vlan-id 99;
    l3-interface irb.99;
    vxlan {
        vni 5099;
    }
}
```

NOTE: If you look at the topology (Figure 2), then you might notice that Access1 does not have a host that is a member of VLAN v88 and Access2 does not have a host that is a member of VLAN v99. Both devices have all three VLANs configured. It is best practice to configure the leafs in this manner as it enables hosts to move between leafs without having to reconfigure the VLANs on the leafs.

Let's talk about configuring the edge interfaces. First, the interfaces on the Access1 and Access2 leafs are not multihomed and connect directly to the end hosts. To that end, we must configure them as access interfaces with the correct VLANs. The following configuration shows the interfaces on Access1 configuration that point towards Desktop1 and AP1:

```
{master:0}[edit interfaces]
lab@access1# show xe-0/2/0
unit 0 {
    family ethernet-switching {
        vlan {
            members v99;
        }
    }
}

{master:0}[edit interfaces]
lab@access1# show xe-1/2/0
unit 0 {
    family ethernet-switching {
        vlan {
            members v33;
        }
    }
}
```

The edge interface configuration is shown for Access2 in the following output:

```
{master:0}[edit interfaces]
lab@access2# show xe-0/2/0
unit 0 {
   family ethernet-switching {
       vlan {
           members v33;
       }
    }
}

{master:0}[edit interfaces]
lab@access2# show xe-0/2/3
unit 0 {
    family ethernet-switching {
        vlan {
            members v88;
        }
    }
}
```

The edge interface configuration for the Core1 and Core2 border leafs differs as they are multihomed with the WAN router. This situation requires us to use a multihomed link aggregation group (LAG) interface where all links that connect VTEPs to a host are in a LAG bundle on both the host and the VTEPs.

LAGs are normally configured on a single device, and the Link Aggregation Control Protocol (LACP) manages the links in the bundle. The LAG interface terminates on three devices within the network; the two links start on the WAN router, then one terminates on Core1, and the other on Core2.

An EVPN Type 1 route, or Ethernet segment route, is advertised between Core1 and Core2 to indicate that they are connected to the same Ethernet segment. In addition, an autodiscovery route, or EVPN Type 4 route, is generated by each connected VTEP and advertised to remote VTEPs. The auto-discovery route is used in the election of a designated forwarder. The designated forwarder is the device connected to the ESI that is responsible for forwarding BUM traffic for the segment. A non-designated forwarder might forward unicast traffic to the Ethernet segment but blocks BUM traffic to avoid packet duplication. The auto-discovery route also indicates what forwarding mode is configured on the device, which in the case of Juniper Networks devices, is always active-active.

From the perspective of the WAN router, the LAG must terminate on the same remote device or must appear to terminate on the same remote device. To enable this functionality, the LACP system ID on Core1 and Core2 is configured with the same value. The LACP control packets sourced from Core1 and Core2 arrive on the WAN router and appear to originate from the same remote device.

MP-BGP enables Core1 and Core2 to manage the shared link properly. We configure Core1 and Core2 with LACP on the single physical link on each device toward the WAN router. The same ESI is assigned to the aggregated Ethernet interface on both devices. As long as the devices are connected to the VXLAN network and the underlay fabric, they maintain the LACP status to the WAN router as active. Suppose one of the core devices loses BGP connectivity to the fabric, therefore is disconnected from the fabric. In that case, the isolated VTEP sets the LACP status to standby, thereby blocking traffic between the host and VTEP.

To limit traffic loss and delay in the fabric network, when a remote VTEP advertises connectivity to an ESI is removed from the network due to failure or some other cause, the remote VTEP performs a mass withdrawal of routes associated with the remote VTEP. Forwarding next hops toward the withdrawn device are remapped to the remaining VTEPs that retain connectivity to the remote ESI.

The following outputs show the configuration of the EVPN-LAG for Core1 and Core2, each of which has a single physical link to the Ethernet segment. The ESI is automatically determined from the value that is set for the LACP system ID. This results in both the ESI and LACP system ID being synchronized on the two devices, as are the interface parameters.

The `all-active` parameter under the ESI configuration enables all links in the Ethernet segment to be active and forward traffic simultaneously. This configuration statement provides load balancing and redundancy in EVPN deployments. This parameter can be omitted from the configuration as it is the default behavior.

The `active` parameter under the `aggregated-ether-options` stanza indicates that the devices actively send LACP packets to negotiate and establish a link aggregation group with the peer device. This parameter is not the default setting here; we must include it.

Lastly, we must specify the number of aggregated Ethernet interfaces we want Junos OS to generate. Technically, we only need to set the `ethernet device-count` parameter to `1`, which generates the ae0 interface we use. Setting it to `2` generates the ae0 and ae1 interfaces, which is useful if we need to add another ESI LAG in the future.

The ae0 interface is configured with the Ethernet switching family that is in trunk mode. We add the v33, v88, and v99 VLANs to the interface. Also, the xe-0/0/29:1 interface is configured to participate as a child interface of ae0.

NOTE: The configuration for this step is the same for Core1 and Core2.

```
{master:0}[edit interfaces]
lab@core1# show xe-0/0/29:1
gigether-options {
    802.3ad ae0;
}

{master:0}[edit interfaces]
lab@core1# show ae0
esi {
    auto-derive {
        lacp;
    }
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 02:02:02:02:02:02;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members [ v99 v88 v33 ];
        }
    }
}

{master:0}[edit interfaces]
lab@core1# top show chassis aggregated-devices
ethernet {
    device-count 2;
}
```

WAN Router Configuration

The WAN router configuration is simple. The following output shows the configuration of the ae0 and underlying interfaces. This configuration is like what was configured on the core devices, with the only difference is this configuration lacks any ESI parameters, and ae0 is a Layer 3 interface split into different VLANs.

```
[edit interfaces]
lab@WAN-router# show
xe-0/0/2 {
    gigether-options {
        802.3ad ae0;
    }
}
xe-0/0/3 {
    gigether-options {
        802.3ad ae0;
    }
}
xe-0/0/6 {
    unit 0 {
        family inet {
            address 192.168.231.2/24;
        }
    }
}
ae0 {
    flexible-vlan-tagging;
    aggregated-ether-options {
        lacp {
            active;
        }
    }
    unit 33 {
        vlan-id 33;
        family inet {
            address 10.33.33.254/24;
        }
    }
    unit 88 {
        vlan-id 88;
        family inet {
            address 10.88.88.254/24;
        }
    }
    unit 99 {
        vlan-id 99;
        family inet {
            address 10.99.99.254/24;
        }
    }
}

...

[edit]
lab@WAN-router# show chassis aggregated-devices
ethernet {
    device-count 2;
}
```

The security policy configuration shows that hosts from the 10.33.33.0/24, 10.88.88.0/24, and 10.99.99.0/24 networks can access the Internet. Also, they are permitted to communicate with each other. These security policies basically permit all traffic. In a real-world scenario, inter-VLAN and potentially Internet-bound traffic must be much more restricted. Additionally, we can add advanced security to these policies, such as ATP Cloud or IPS, to provide additional security to these communications.

```
[edit]
lab@WAN-router# show security policies
global {
    policy Inet-access {
        match {
            source-address [ v88 v99 v33 ];
            destination-address any;
            application any;
            from-zone fabric;
            to-zone WAN;
        }
        then {
            permit;
        }
    }
    policy fab-fab-permit {
        match {
            source-address [ v88 v99 v33 ];
            destination-address [ v88 v99 v33 ];
            application any;
            from-zone fabric;
            to-zone fabric;
        }
        then {
            permit;
        }
    }
}
```

Lastly, the security zone configuration is shown. The three logical ae0 interfaces are placed in the fabric zone. The host inbound traffic configuration allows any inbound traffic. We want to be much more selective here in a real-world scenario.

```
[edit]
lab@WAN-router# show security zones
security-zone WAN {
    host-inbound-traffic {
        system-services {
            ping;
            dhcp;
        }
    }
    interfaces {
        xe-0/0/6.0;
    }
}
security-zone fabric {
    host-inbound-traffic {
        system-services {
            any-service;
        }
    }
    interfaces {
        ae0.88;
        ae0.99;
        ae0.33;
    }
}
```

CHAPTER 3 EVPN-VXLAN Campus Fabric Verification

Verification of the Campus Fabric IP Clos Deployment

For verification of the Campus Fabric IP Clos deployment, refer Figure 2.

We can use four hosts (Desktop1, Desktop2, AP1, and AP2) and an Internet host to validate the Campus Fabric. Let’s take a quick look to see if AP1 can connect internally and externally. Before we run our tests, let’s assume that another network administrator accidentally made a change that caused a problem. This problem gives us an opportunity to troubleshoot our Campus Fabric.

You can use a third-party tool, such as SecureCRT, to validate AP1’s connectivity:

```
[lab@AP1 ~]$ ip a | grep 10.33.33.
    inet 10.33.33.101/24 brd 10.33.33.255 scope global noprefixroute eth1

[lab@AP1 ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.33.33.1      0.0.0.0         UG    100    0        0 eth1
10.33.33.0      0.0.0.0         255.255.255.0   U     100    0        0 eth1

[lab@ AP1 ~]$ ping 10.33.33.1 -c 3
PING 10.33.33.1 (10.33.33.1) 56(84) bytes of data.
64 bytes from 10.33.33.1: icmp_seq=1 ttl=64 time=7.09 ms
64 bytes from 10.33.33.1: icmp_seq=2 ttl=64 time=9.63 ms
64 bytes from 10.33.33.1: icmp_seq=3 ttl=64 time=12.1 ms

--- 10.33.33.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 5ms
rtt min/avg/max/mdev = 7.087/9.620/12.140/2.062 ms
[lab@AP1 ~]$ ping 10.33.33.102 -c 3
PING 10.33.33.102 (10.33.33.102) 56(84) bytes of data.
64 bytes from 10.33.33.102: icmp_seq=1 ttl=64 time=0.469 ms
64 bytes from 10.33.33.102: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 10.33.33.102: icmp_seq=3 ttl=64 time=0.492 ms

--- 10.33.33.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 41ms
rtt min/avg/max/mdev = 0.469/0.481/0.492/0.020 ms

[lab@AP1 ~]$ ping 10.33.33.254 -c 3
PING 10.33.33.254 (10.33.33.254) 56(84) bytes of data.

--- 10.33.33.254 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 29ms

[lab@AP1 ~]$ ping 10.99.99.99 -c 3
PING 10.99.99.99 (10.99.99.99) 56(84) bytes of data.

--- 10.99.99.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 73ms

[lab@AP1 ~]$ ping 10.88.88.42 -c 3
PING 10.88.88.42 (10.88.88.42) 56(84) bytes of data.

--- 10.88.88.42 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 85ms
```

```
[lab@AP1 ~]$ ping 8.8.8.8 -c 3
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 92ms
```

Validation Steps

- Confirm local IP address (10.33.33.33.101) and default gateway (10.33.33.1) are configured on Desktop1.
- A successful ping of the default gateway (10.33.33.1) tells us AP1 can reach Access1.
- A failure to ping the WAN router (10.33.33.254) tells us AP1 can't reach the WAN router.
- A successful ping of AP2 (10.33.33.102) tells us AP1 can reach AP2, which is directly connected to Access2, through a VXLAN tunnel.
- A failure to ping Desktop1 (10.99.99.99) tells us AP1 can't reach Desktop1.
- A failure to ping Desktop2 (10.88.88.42) tells us AP1 can't reach Desktop2.
- A failure to ping an Internet host (8.8.8.8) tells us AP1 can't reach the Internet.

It looks like AP1 can reach devices in its own VRF and VLAN, except for the WAN router. It also can't reach anything outside of its VRF. Recall that all inter-VRF routing must go through the WAN router, so if AP1 can't reach the WAN router, it makes sense that it can't reach Desktop1 and Desktop2.

BGP Validation

Purpose

Verifying the state of BGP between adjacent layers is essential for EVPN-VXLAN to operate as expected. This network of point-to-point links between each layer supports the following:

- Load balancing using ECMP for greater resiliency and bandwidth efficiencies.
- BGP peering as well as loopback VXLAN reachability.

Without requiring verification at each layer, the focus can be on the distribution switches and their BGP relationships with the access and core switches. We can move to the next verification phase if both distribution switches have established BGP peering sessions with each adjacent layer.

Action

Verify that BGP sessions are established from the distribution switches with the access and core switches to ensure loopback reachability and load-balancing using ECMP.

Verification of BGP Peering

Distribution1:

```
{master:0}
lab@distrubtion1> show bgp summary

Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 2 Peers: 9 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.evpn.0
                      32         32          0          0          0          0
inet.0
                      20          8          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
172.16.1.0            64512         85         81       0       1       34:49 Establ
  inet.0: 2/5/5/0
172.16.1.4            64513         84         84       0       1       34:49 Establ
  inet.0: 2/5/5/0
172.16.1.9            64516         85         84       0       1       34:45 Establ
  inet.0: 2/5/5/0
172.16.1.11           64517         86         85       0       1       34:45 Establ
  inet.0: 2/5/5/0
192.168.101.1         65512         91         92       0       1       34:45 Establ
  bgp.evpn.0: 3/3/3/0
192.168.101.2         65513         82        100       0       1       34:41 Establ
  bgp.evpn.0: 3/3/3/0
192.168.102.2         65512         91         92       0       1       34:45 Establ
  bgp.evpn.0: 0/3/3/0
192.168.103.1         64516         84         98       0       1       34:47 Establ
  bgp.evpn.0: 13/13/13/0
192.168.103.2         64517         84         98       0       1       34:46 Establ
  bgp.evpn.0: 13/13/13/0
```

From the BGP sessions, we can see that the underlay peer relationships are established. This tells us the underlay links are attached to the correct devices and the links are up.

From the BGP summary, we can see that the underlay (172.16.1.X) peer relationships are established. This tells us the underlay links are attached to the correct devices and the links are up.

It also shows that the overlay (192.168.X.X) relationships are established and peering with the correct loopback addresses. This fact demonstrates loopback reachability.

It also shows that the overlay relationships are established and peering with the correct loopback addresses. This fact demonstrates loopback reachability. We can also see routes received; the time established is roughly equal, which looks good so far.

NOTE: If some of the BGP session do not establish correctly, go back and validate the correct links and IP addressing that you use for the BGP peering sessions.

Let's verify the routes are established to the Core and other devices across multiple paths. For example, Access1 and Access2 should leverage both paths through the distribution switches to access the loopback address of the core switches and each other.

Access1: Loopback Reachability to Core1 Through Distribution1 and Distribution2

```
{master:0}
lab@access1> show route forwarding-table destination 192.168.101.1
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
192.168.101.1/32   user     0                    ulst   131078     7
                              172.16.1.8         ucst     1945     6 xe-0/2/1.0
                              172.16.1.12        ucst     1946     6 xe-0/2/2.0
```

Access1: Loopback Reachability with Core2 Through Distribution1 and Distribution2

```
{master:0}
lab@access1> show route forwarding-table destination 192.168.101.2
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
192.168.101.2/32   user     0                    ulst   131078     7
                              172.16.1.8         ucst     1945     6 xe-0/2/1.0
                              172.16.1.12        ucst     1946     6 xe-0/2/2.0
```

Access1: Loopback Reachability with Access2 Through Distribution1 and Distribution2

```
{master:0}
lab@access1> show route forwarding-table destination 192.168.103.2
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
192.168.103.2/32   user     0                    ulst   131078     7
                              172.16.1.8         ucst     1945     6 xe-0/2/1.0
                              172.16.1.12        ucst     1946     6 xe-0/2/2.0
```

This can be repeated for Access2 and so forth to verify ECMP load balancing.

NOTE: At this point BGP underlay and overlay is operational through the verification of BGP between adjacent layers of the campus fabric and that routes are established to access, distribution, and core switches.

EVPN VXLAN Verification Between Access and Core Switches

Since AP1 can ping its default gateway, we can assume the Ethernet switching tables are correctly populated and VLAN and interface modes are correct. If pinging the default gateway failed, then we should troubleshoot underlay connectivity.

Verification of the EVPN Database on both Access Switches

You can view the entire database or search by MAC address.

```
{master:0}
lab@access1> show evpn database
Instance: default-switch
VLAN  DomainId  MAC address        Active source         Timestamp         IP address
     5033       00:50:56:be:ab:59  xe-1/2/0.0            May 25 22:31:25   10.33.33.101
     5033       00:50:56:be:f5:0b  192.168.103.2         May 25 22:37:54   10.33.33.102
     5033       02:05:86:8f:bd:00  irb.33                May 25 21:13:03   10.33.33.1
     5088       00:50:56:be:54:2a  192.168.103.2         May 25 22:37:54   10.88.88.42
     5088       02:05:86:8f:bd:00  irb.88                May 25 22:37:51   10.88.88.1
     5099       00:50:56:be:ed:96  xe-0/2/0.0          May 25 20:17:02   10.99.99.99
     5099       02:05:86:8f:bd:00  irb.99              May 25 21:13:03   10.99.99.1

{master:0}
lab@access1> show evpn database | match 00:50:56:be:ab:59
     5033       00:50:56:be:ab:59  xe-1/2/0.0          May 25 22:31:25   10.33.33.101
```

```
{master:0}
lab@access2> show evpn database
Instance: default-switch
VLAN  DomainId  MAC address        Active source         Timestamp         IP address
     5033       00:50:56:be:ab:59  192.168.103.1         May 25 22:37:54   10.33.33.101
     5033       00:50:56:be:f5:0b  xe-0/2/0.0            May 25 20:20:20   10.33.33.102
     5033       02:05:86:8f:bd:00  irb.33                May 25 21:20:57   10.33.33.1
     5088       00:50:56:be:54:2a  xe-0/2/3.0            May 25 20:20:54   10.88.88.42
     5088       02:05:86:8f:bd:00  irb.88                May 25 21:20:57   10.88.88.1
     5099       00:50:56:be:ed:96  192.168.103.1         May 25 22:37:54   10.99.99.99
     5099       02:05:86:8f:bd:00  irb.99                May 25 22:37:53   10.99.99.1

{master:0}
lab@access2> show evpn database | match 00:50:56:be:ab:59
     5033       00:50:56:be:ab:59  192.168.103.1         May 25 22:37:54   10.33.33.101
```

Both access switches have similar EVPN databases, which is expected. The entries for AP1 (10.33.33.101) and Desktop1 (10.99.99.99) are present in each access switch. These entries are learned locally or through the campus fabric as represented in the `Active source` output. There are no entries for the ESI LAG between the core switches and the WAN router. That's a point to keep in mind as we go.

The 10.33.33.101 IP address is associated with irb.33, and we see a VNI of 5033. Let us double-check VLAN-VNI mapping on Access and Core switches.

Access

```
{master:0}
lab@access1> show configuration vlans | display set | match 33
set vlans v33 vlan-id 33
set vlans v33 l3-interface irb.33
set vlans v33 vxlan vni 5033
```

Core

```
{master:0}
lab@core1> show configuration | display set | match 33
set routing-instances evpn_vs interface irb.33
set routing-instances evpn_vs vlans v33 vlan-id 33
set routing-instances evpn_vs vlans v33 l3-interface irb.33
set routing-instances evpn_vs vlans v33 vxlan vni 5033
```

Verification of VXLAN Tunneling Between Access and Core Devices

Access1:

The following output shows that Access1 has three VXLAN tunnels coming from the core devices and Access2. You can also see the associated VTEP interfaces and source IP addresses for the remote VXLAN tunnels. The output also tells us that Access1 is sourcing its VXLAN tunnel from the loopback interface and IP address.

```
{master:0}
lab@access1> show ethernet-switching vxlan-tunnel-end-point remote summary
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx    SVTEP-Mode    ELP-SVTEP-IP
<default>                 0   192.168.103.1    lo0.0    0
 RVTEP-IP        L2-RTT           IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP   Flags
 192.168.101.1  default-switch   571       vtep.32769   1843    RNVE
 192.168.101.2  default-switch   576       vtep.32771   1854    RNVE
 192.168.103.2  default-switch   575       vtep.32770   1851    RNVE
```

Access2:

The following output from Access2 shows similar information to what we saw on Access1. Three VXLAN tunnels from the core devices and Access2 are terminating on Access2.

```
{master:0}
lab@access2> show ethernet-switching vxlan-tunnel-end-point remote summary
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx    SVTEP-Mode    ELP-SVTEP-IP
<default>                 0   192.168.103.2    lo0.0    0
 RVTEP-IP        L2-RTT          IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP    Flags
 192.168.101.1  default-switch  708       vtep.32770   1855    RNVE
 192.168.103.1  default-switch  707       vtep.32769   1840    RNVE
 192.168.101.2  default-switch  709       vtep.32771   1856    RNVE
```

Verify AP1's MAC address is advertised through BGP.

```
{master:0}
lab@access1> show route advertising-protocol bgp 192.168.102.1 evpn-mac-address
00:50:56:be:ab:59 table bgp.evpn.0


bgp.evpn.0: 32 destinations, 51 routes (32 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  2:192.168.103.1:1::5033::00:50:56:be:ab:59/304 MAC/IP
*                         Self                         100        I
  2:192.168.103.1:1::5033::00:50:56:be:ab:59::10.33.33.101/304 MAC/IP
*                         Self                         100        I
```

And, is it is received on Core1.

```
{master:0}
lab@core1> show route receive-protocol bgp 192.168.102.1 evpn-mac-address
00:50:56:be:ab:59 table bgp.evpn.0


bgp.evpn.0: 32 destinations, 61 routes (32 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  2:192.168.103.1:1::5033::00:50:56:be:ab:59/304 MAC/IP
*                         192.168.103.1                100        I
  2:192.168.103.1:1::5033::00:50:56:be:ab:59::10.33.33.101/304 MAC/IP
*                         192.168.103.1                100        I
```

Let's check to see if Core1 has the AP1 MAC address. We can see that the MAC address for AP1 is present, but shouldn't we also see the entries for the .254 IP addresses for the three VLANs? Keep that in mind as we continue to troubleshoot.

```
{master:0}
lab@core1> show evpn database
Instance: evpn_vs
VLAN  DomainId  MAC address        Active source          Timestamp         IP address
     5033       00:50:56:be:ab:59  192.168.103.1          May 25 23:41:59  10.33.33.101
     5033       00:50:56:be:f5:0b  192.168.103.2          May 25 23:41:59  10.33.33.102
     5033       02:05:86:8f:bd:00  192.168.103.1          May 25 23:41:59  10.33.33.1
     5088       00:50:56:be:54:2a  192.168.103.2          May 25 23:41:59  10.88.88.42
     5088       02:05:86:8f:bd:00  192.168.103.1          May 25 23:41:59  10.88.88.1
     5099       00:50:56:be:ed:96  192.168.103.1          May 25 23:41:59  10.99.99.99
     5099      02:05:86:8f:bd:00  192.168.103.2         May 25 23:41:59  10.99.99.1

lab@core1> show evpn database | match 00:50:56:be:ab:59
     5033      00:50:56:be:ab:59  192.168.103.1         May 25 22:37:50  10.33.33.101
```

Let’s verify that the MAC address is mapped to the correct VTEP interface on Core1. You can also verify on any access switch.

```
{master:0}
lab@core1> show route forwarding-table family ethernet-switching extensive destination
00:50:56:be:ab:59
Routing table: evpn_vs.evpn-vxlan [Index 9]
Bridging domain: v33.evpn-vxlan [Index 3]
VPLS:
Enabled protocols: Bridging, ACKed by all peers, EVPN VXLAN,

Destination:  00:50:56:be:ab:59/48
  Learn VLAN: 0                       Route type: user
  Route reference: 0                  Route interface-index: 555
  Multicast RPF nh index: 0
  P2mpidx: 0
  IFL generation: 282                 Epoch: 0
  Sequence Number: 0                  Learn Mask:
0x400000000000000000000000000000000000000000
  L2 Flags: control_dyn
  Flags: sent to PFE
  Nexthop:
  Next-hop type: composite            Index: 1728     Reference: 10
  Next-hop type: indirect             Index: 2097151  Reference: 3
  Next-hop type: unilist              Index: 2097150  Reference: 6
  Nexthop: 172.16.1.1
  Next-hop type: unicast              Index: 1729     Reference: 6
  Next-hop interface: et-0/0/0.0      Weight: 0x0
  Nexthop: 172.16.1.3
  Next-hop type: unicast              Index: 1727     Reference: 6
  Next-hop interface: et-0/0/35.0     Weight: 0x0
```

Then, let’s use the following output to find which VTEP interface is associated with the 1728 index. The following output shows that the vtep.32769 interface is what we’re looking for.

```
{master:0}
lab@core1> show route forwarding-table family ethernet-switching | find evpn_vs.evpn-vxlan
Routing table: evpn_vs.evpn-vxlan
VPLS:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            perm     0                    dscd     1703     1
vtep.32769         intf     0                    comp     1728    10
vtep.32770         intf     0                    comp     1742     9
ae0.0              intf     0                    ucst     1747     2 ae0.0
```

Finally, check to see if the VTEP interface is up and passing traffic.

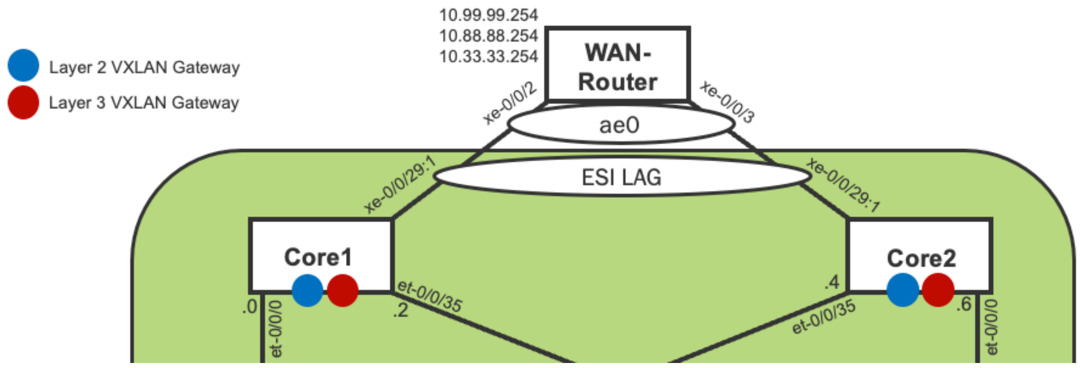
```
{master:0}
lab@core1> show interfaces vtep.32769
  Logical interface vtep.32769 (Index 555) (SNMP ifIndex 536)
    Flags: Up SNMP-Traps Encapsulation: ENET2
    VXLAN Endpoint Type: Remote, VXLAN Endpoint Address: 192.168.103.1, L2 Routing
Instance: evpn_vs, L3 Routing Instance: default
    Input packets : 0
    Output packets: 0
    Protocol eth-switch, MTU: Unlimited
      Flags: Trunk-Mode
```

The VTEP interface is up, but it isn’t passing traffic. We need to troubleshoot this problem further. Might be that we are looking in the wrong place. The previous outputs showed that we are missing information about the ESI LAG interface. With that in mind, let’s examine the connection between core devices and the WAN router.

External Campus Fabric Connectivity Through the Border GW Core EX4650 Switches

Remember that you chose to deploy the Border GW capability on the EX4650 switches during the IP Clos deployment, represented below:

Figure 3: Layer 2 ESI-LAG Supporting Active-Active Load Balancing



Let's verify the Ethernet Segment Identifier (ESI) status on the core switches.

```
{master:0}
lab@core1> show lacp statistics interfaces
Aggregated interface: ae0
    LACP Statistics:       LACP Rx     LACP Tx    Unknown Rx   Illegal Rx
      xe-0/0/29:1                0           0            0            0

{master:0}
lab@core1> show lacp interfaces
Aggregated interface: ae0
    LACP state:       Role   Exp   Def  Dist  Col  Syn  Aggr  Timeout  Activity
      xe-0/0/29:1    Actor    No   Yes    No   No   No   Yes     Fast    Active
      xe-0/0/29:1  Partner    No   Yes    No   No   No   Yes     Fast   Passive
    LACP protocol:        Receive State  Transmit State          Mux State
      xe-0/0/29:1         Port disabled     No periodic           Detached
```

It looks like LACP is enabled on the ae0 interface, but the xe-0/0/29:1 interface appears to be disabled. Let's look at the xe-0/0/29:1 and ae0 interfaces.

```
{master:0}
lab@core1> show interfaces terse xe-0/0/29:1
Interface               Admin Link Proto    Local                 Remote
xe-0/0/29:1             down  down
xe-0/0/29:1.0           up    down aenet    --> ae0.0

{master:0}
lab@core1> show interfaces ae0 terse
Interface               Admin Link Proto    Local                 Remote
ae0                     down  down
ae0.0                   up    down eth-switch

{master:0}
lab@core1> show configuration interfaces xe-0/0/29:1
gigether-options {
    802.3ad ae0;
}

{master:0}
lab@core1> show configuration interfaces ae0
disable;
esi {
    auto-derive {
        lacp;
    }
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 02:02:02:02:02:02;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members [ v99 v88 v33 ];
        }
    }
}
```

Found it! It looks like the other network administrator disabled the ae0 interface for some reason. Might be that they were troubleshooting another potential problem and forgot to reenable it after they were done. Either way, we should definitely remove the `disable` statement from the ae0 interface configuration.

```
{master:0}
lab@core1> configure
Entering configuration mode

{master:0}[edit]
lab@core1# delete interfaces ae0 disable

{master:0}[edit]
lab@core1# show interfaces ae0
esi {
    auto-derive {
        lacp;
    }
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 02:02:02:02:02:02;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members [ v99 v88 v33 ];
        }
    }
}

{master:0}[edit]
lab@core1# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode

{master:0}
lab@core1>
```

NOTE: Although it's not shown, Core2 also had its ae0 interface disabled. The configuration was fixed and committed, same as we did for Core1.

Now, let's look at Core1 again. The following output shows that LACP packets are sent and received, and the xe-0/0/29:1 interface is functioning. The EVPN data not only shows the entry for AP1 but also shows that new entries are present for the .254 IP addresses for each of the three VLANs. Also, notice that the active source for these entries is the ESI value for the ESI LAG between the core devices and the WAN router.

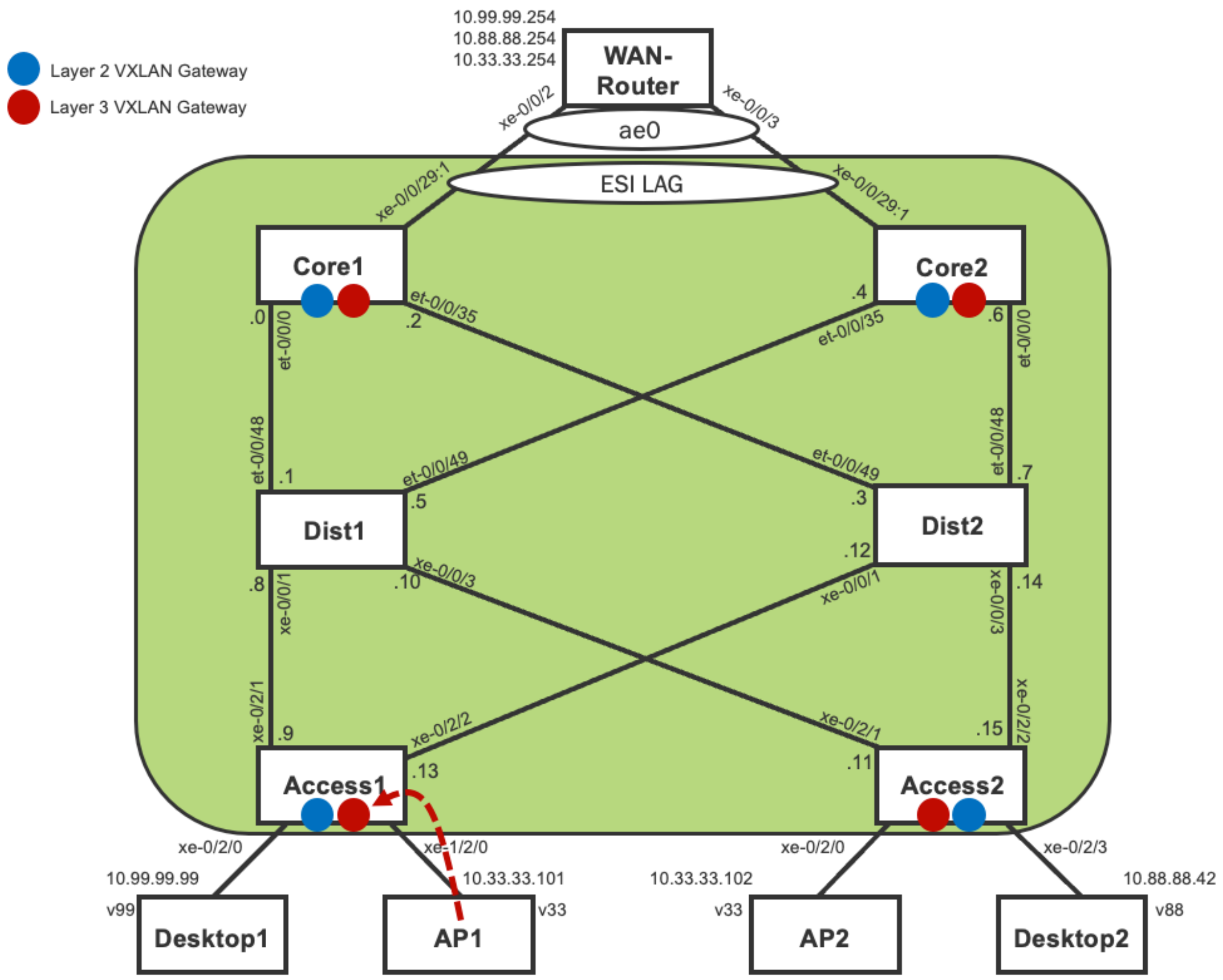
```
{master:0}
lab@core1> show lacp statistics interfaces
Aggregated interface: ae0
    LACP Statistics:       LACP Rx     LACP Tx   Unknown Rx   Illegal Rx
      xe-0/0/29:1              193         193            0            0

{master:0}
lab@core1> show lacp interfaces
Aggregated interface: ae0
    LACP state:       Role    Exp    Def  Dist  Col  Syn  Aggr  Timeout  Activity
      xe-0/0/29:1    Actor     No     No   Yes  Yes  Yes   Yes     Fast    Active
      xe-0/0/29:1  Partner     No     No   Yes  Yes  Yes   Yes     Fast    Active
    LACP protocol:        Receive State  Transmit State          Mux State
      xe-0/0/29:1               Current   Fast periodic Collecting distributing

{master:0}
lab@core1> show evpn database
Instance: evpn_vs
VLAN DomainId MAC address      Active source                  Timestamp        IP address
   5033       00:50:56:be:ab:59  192.168.103.1                  May 25 23:41  10.33.33.101
   5033       00:50:56:be:f5:0b  192.168.103.2                  May 25 23:41  10.33.33.102
   5033       02:05:86:8f:bd:00  192.168.103.1                  May 25 23:41  10.33.33.1
   5033       20:d8:0b:04:7f:40  01:02:02:02:02:02:02:00:01:00  May 25 23:44  10.33.33.254
   5088       00:50:56:be:54:2a  192.168.103.2                  May 25 23:41  10.88.88.42
   5088       02:05:86:8f:bd:00  192.168.103.1                  May 25 23:41  10.88.88.1
   5088       20:d8:0b:04:7f:40  01:02:02:02:02:02:02:00:01:00  May 25 23:44  10.88.88.254
   5099       00:50:56:be:ed:96  192.168.103.1                  May 25 23:41  10.99.99.99
   5099       02:05:86:8f:bd:00  192.168.103.2                  May 25 23:41  10.99.99.1
   5099       20:d8:0b:04:7f:40  01:02:02:02:02:02:02:00:01:00  May 25 23:44  10.99.99.254
```

Let's return to AP1 to see if it can send traffic across the fabric. It's also good to check whether AP1 can still ping its default gateway and AP2.

Let's start by pinging the default gateway.

```
[lab@AP1 ~]$ ping 10.33.33.1 -c 3
PING 10.33.33.1 (10.33.33.1) 56(84) bytes of data.
64 bytes from 10.33.33.1: icmp_seq=1 ttl=64 time=11.8 ms
64 bytes from 10.33.33.1: icmp_seq=2 ttl=64 time=3.45 ms
64 bytes from 10.33.33.1: icmp_seq=3 ttl=64 time=12.8 ms
--- 10.33.33.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 5ms
rtt min/avg/max/mdev = 3.447/9.348/12.830/4.196 ms
```

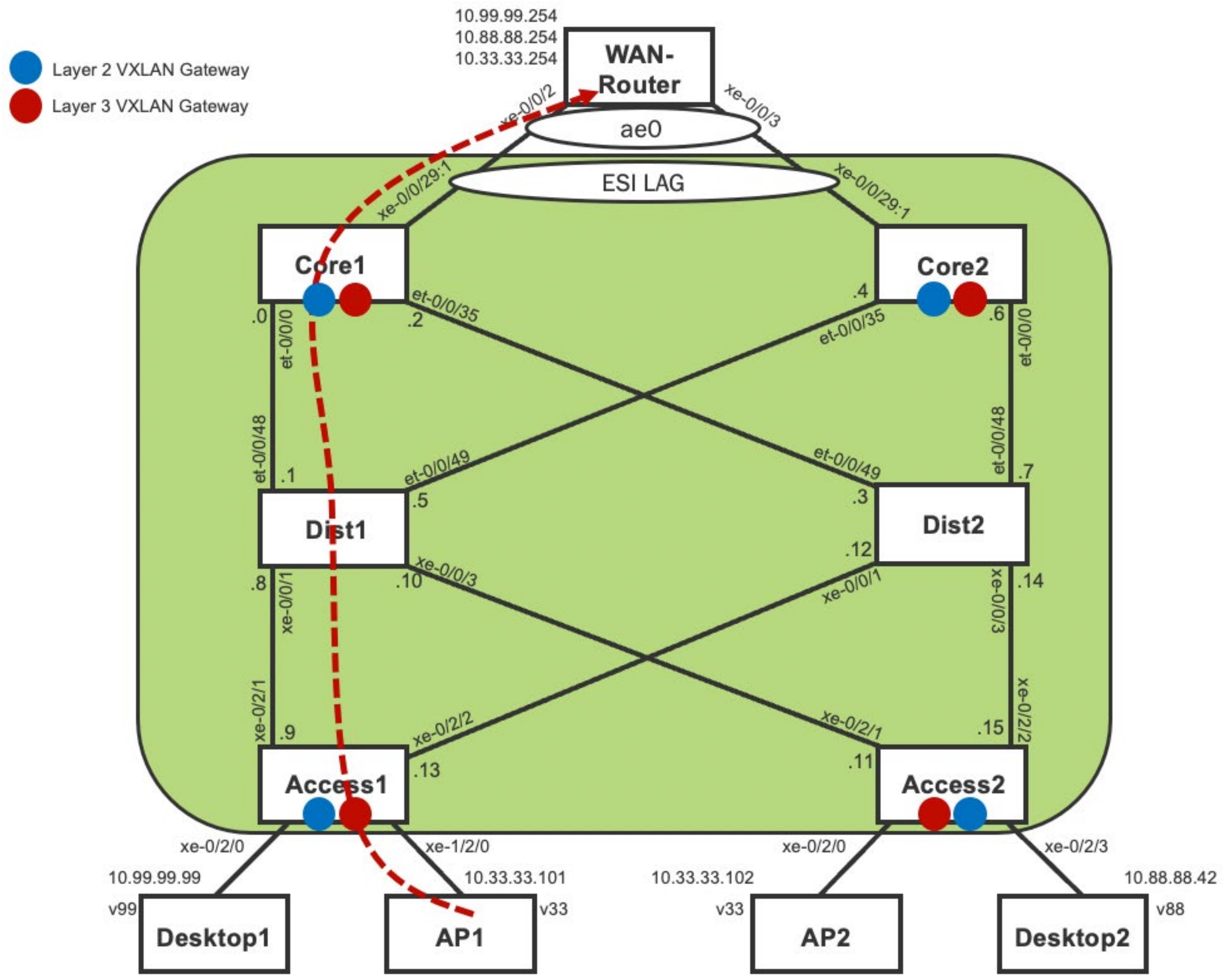
The following diagram shows how the traffic flows through the Campus Fabric when AP1 communicates with its default gateway (Access1).



Then, let's ping the WAN router.

```
[lab@AP1 ~]$ ping 10.33.33.254 -c 3
PING 10.33.33.254 (10.33.33.254) 56(84) bytes of data.
64 bytes from 10.33.33.254: icmp_seq=1 ttl=64 time=0.373 ms
64 bytes from 10.33.33.254: icmp_seq=2 ttl=64 time=0.298 ms
64 bytes from 10.33.33.254: icmp_seq=3 ttl=64 time=0.258 ms

--- 10.33.33.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 61ms
rtt min/avg/max/mdev = 0.258/0.309/0.373/0.051 ms
```

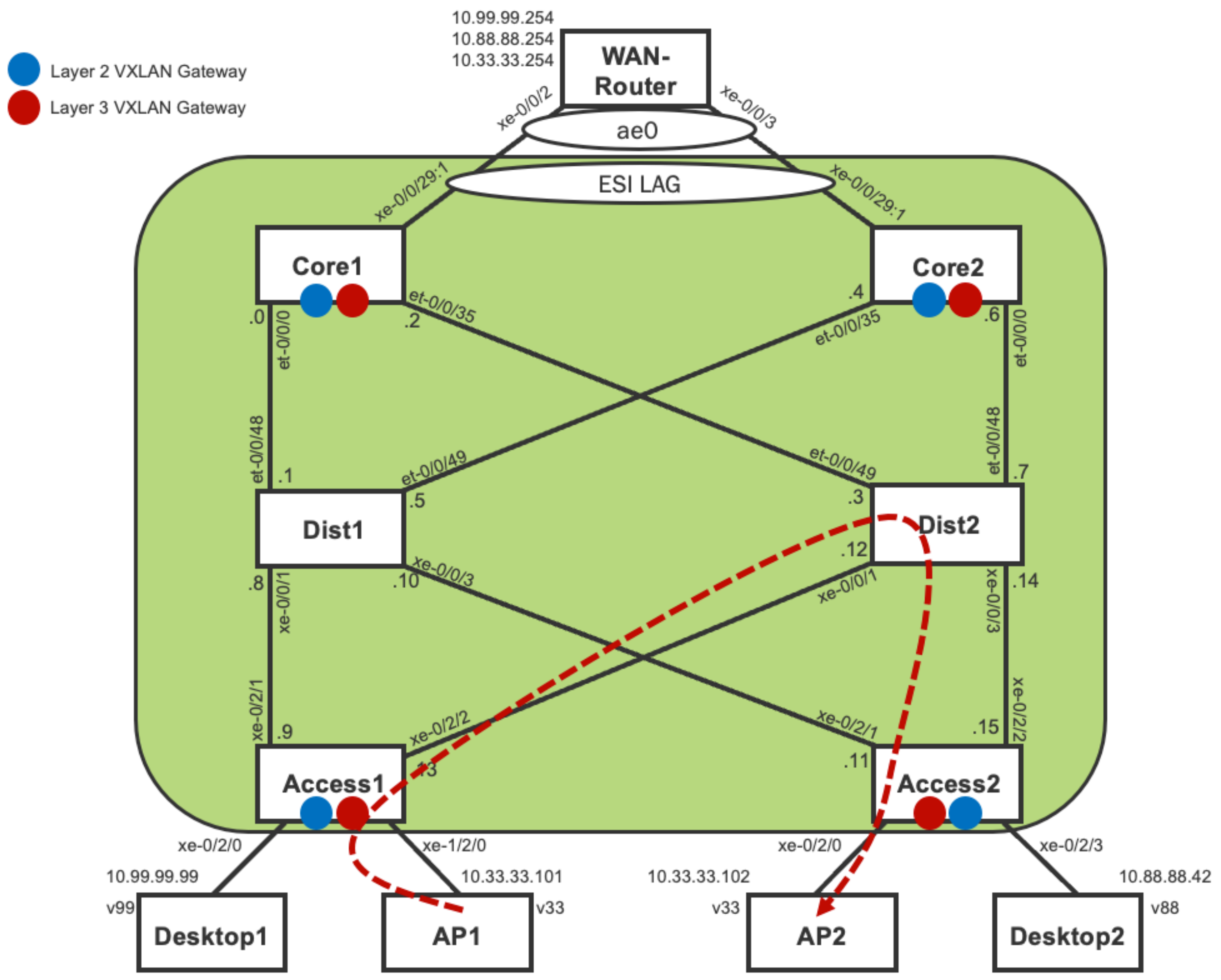
The following diagram shows how the traffic flows through the Campus Fabric when AP1 communicates with the WAN router.



Then, let's ping AP2.

```
[lab@AP1 ~]$ ping 10.33.33.102 -c 3
PING 10.33.33.102 (10.33.33.102) 56(84) bytes of data.
64 bytes from 10.33.33.102: icmp_seq=1 ttl=64 time=0.597 ms
64 bytes from 10.33.33.102: icmp_seq=2 ttl=64 time=0.490 ms
64 bytes from 10.33.33.102: icmp_seq=3 ttl=64 time=0.459 ms

--- 10.33.33.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 71ms
rtt min/avg/max/mdev = 0.459/0.515/0.597/0.061 ms
```

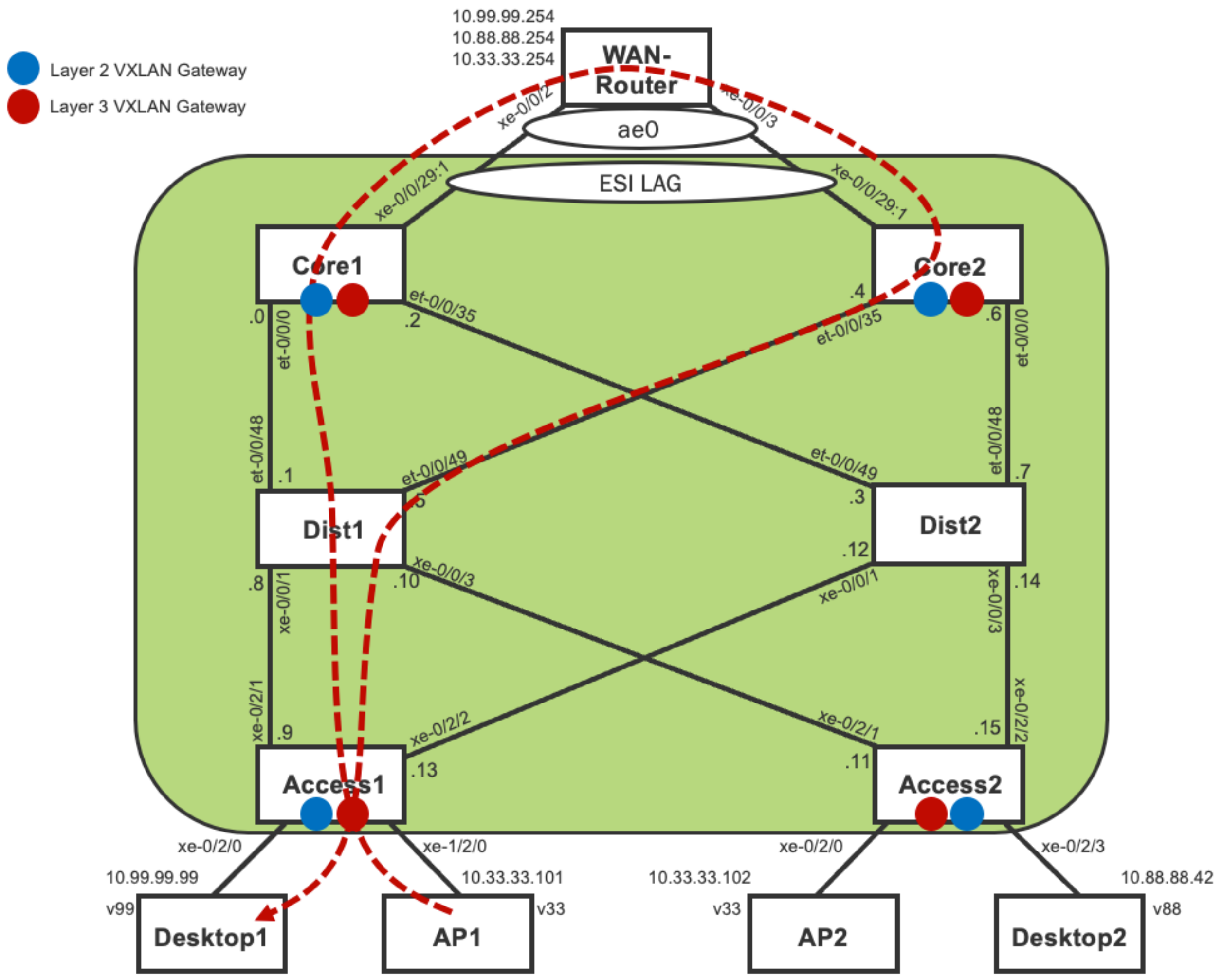
The following diagram shows how the traffic flows through the Campus Fabric when AP1 communicates with AP2.



Next, let’s ping Desktop1.

```
[lab@AP1 ~]$ ping 10.99.99.99 -c 3
PING 10.99.99.99 (10.99.99.99) 56(84) bytes of data.
64 bytes from 10.99.99.99: icmp_seq=1 ttl=62 time=0.610 ms
64 bytes from 10.99.99.99: icmp_seq=2 ttl=62 time=0.543 ms
64 bytes from 10.99.99.99: icmp_seq=3 ttl=62 time=0.517 ms

--- 10.99.99.99 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 56ms
rtt min/avg/max/mdev = 0.517/0.556/0.610/0.047 ms
```

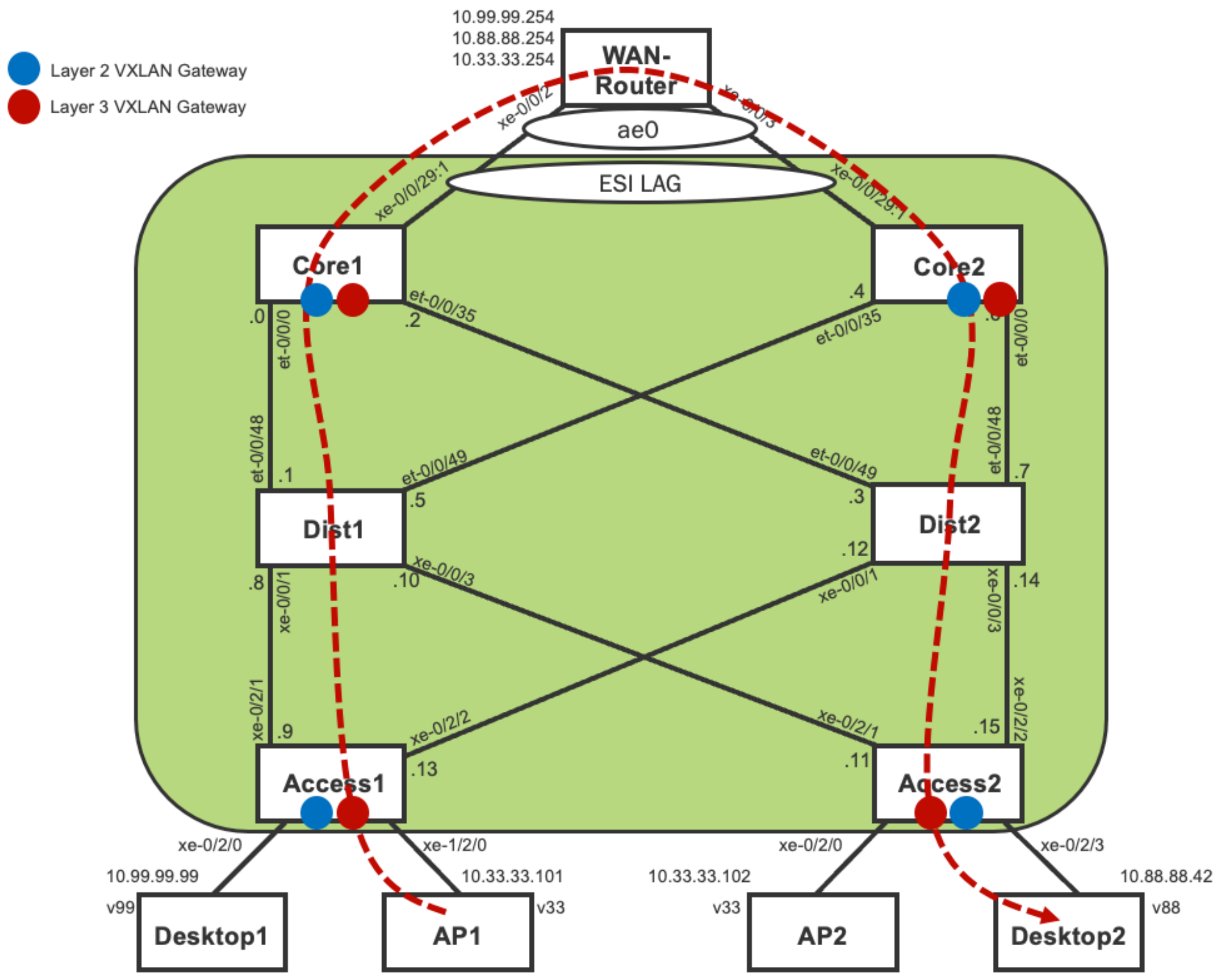
The following diagram shows how the traffic flows through the Campus Fabric when AP1 communicates with Desktop1.



Finally, let’s ping Desktop2.

```
[lab@AP1 ~]$ ping 10.88.88.42 -c 3
PING 10.88.88.42 (10.88.88.42) 56(84) bytes of data.
64 bytes from 10.88.88.42: icmp_seq=1 ttl=62 time=54.4 ms
64 bytes from 10.88.88.42: icmp_seq=2 ttl=62 time=0.599 ms
64 bytes from 10.88.88.42: icmp_seq=3 ttl=62 time=0.504 ms

--- 10.88.88.42 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 41ms
rtt min/avg/max/mdev = 0.504/18.491/54.372/25.371 ms
```

The following diagram shows how the traffic flows through the Campus Fabric when AP1 communicates with Desktop2.



It looks like AP1 can ping everything it needs to. The previous outputs confirm the following:

- A successful ping of the default gateway (10.33.33.1) tells us AP1 can reach Access1.
- A successful ping of the WAN router (10.33.33.254) tells us AP1 can reach the WAN router.
- A successful ping of AP2 (10.33.33.102) tells us AP1 can reach AP2.
- A successful ping of Desktop1 (10.99.99.99) tells us AP1 can reach Desktop1.
- A successful ping of Desktop2 (10.88.88.42) tells us AP1 can reach Desktop2.
- A successful ping of an Internet host (8.8.8.8) tells us AP1 can reach the Internet.

NOTE: Connectivity within and outside of the campus fabric is verified. End devices communicate with each through the fabric, each in an isolated VRF. Then, traffic is forwarded to the SRX Series Firewall through the ESI-LAG on the core devices when accessing services outside of the campus fabric. The campus fabric performs total isolation between VRFs by default while using the SRX Series Firewall to accept or discard inter-VRF communications.

NOTE: The configurations for the devices discussed in chapters 1 through 3 are found in Appendix A, which is at the end of this document

CHAPTER 4 Seamless EVPN-VXLAN Stitching

Seamless EVPN-VXLAN Stitching: Type-2 Stitching

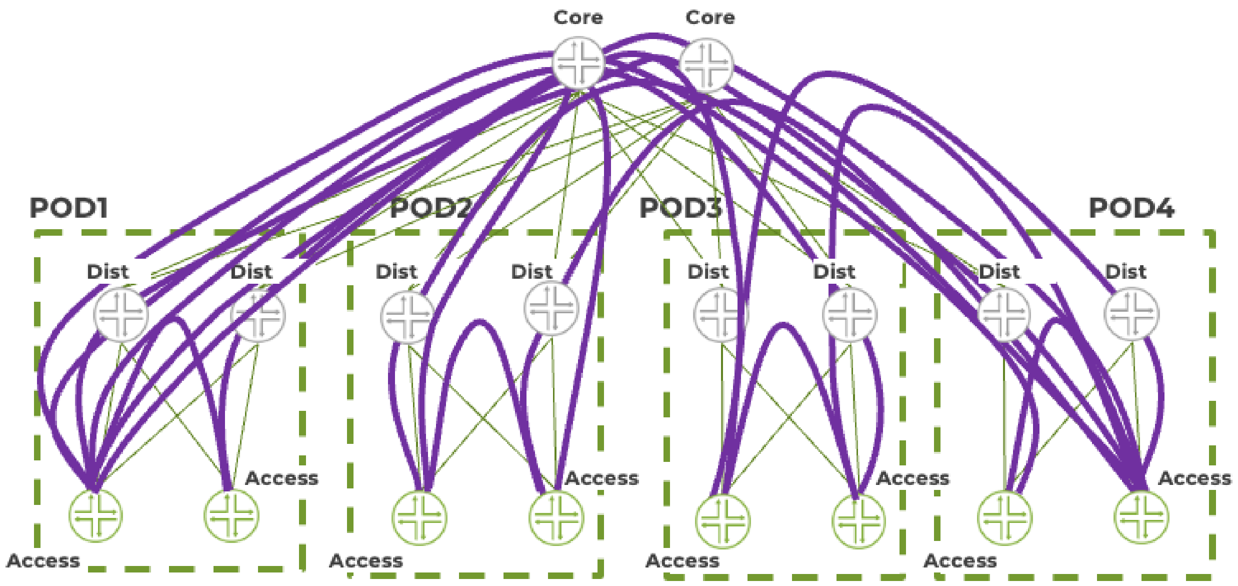
In modern data centers, three foundational elements are crucial: DC agility, a unified DC ecosystem, and engineering simplicity. While seamless EVPN-VXLAN stitching alone does not encompass all these elements, it effectively integrates with the unified DC ecosystem. Achieving agility is possible through software-based network provisioning rather than onsite engineering. This approach, supported by virtualized network models and enhanced with automation and analytics, enables efficient management and cost-effective operations.

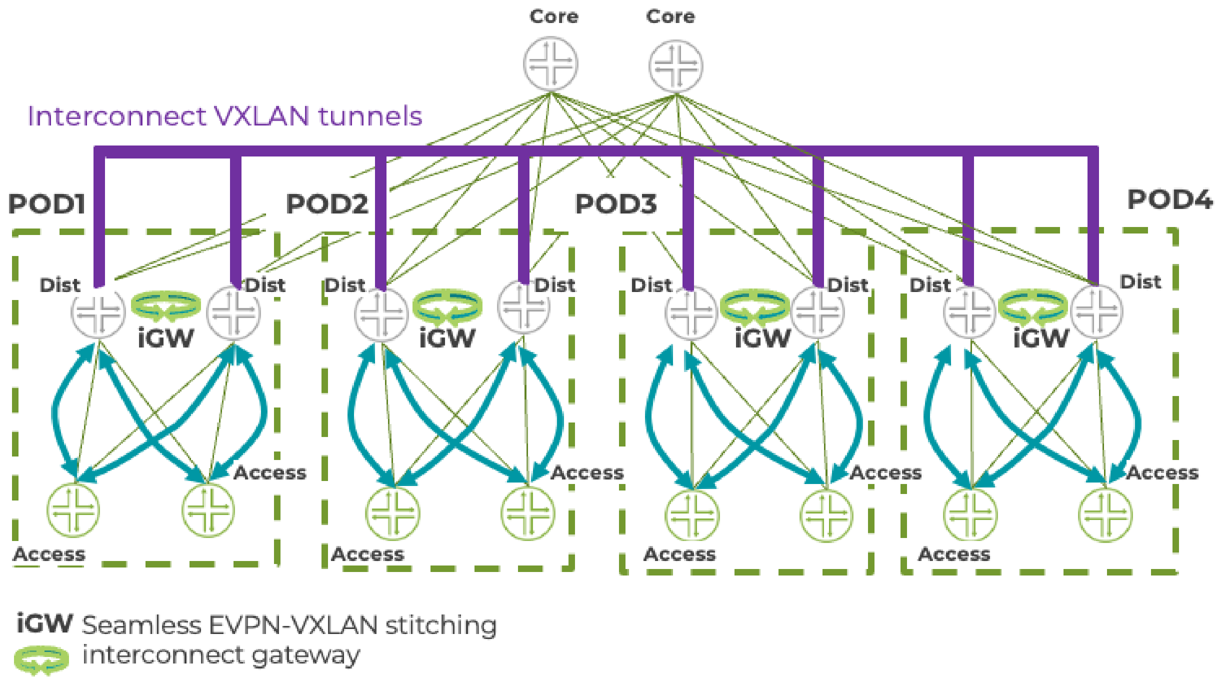
EVPN enhances virtualization in data centers, facilitating the management of tenants and services. Juniper's EVPN operates not just at the infrastructure level but also at the server/compute node level, ensuring end-to-end unified signalization.

Seamless EVPN-VXLAN stitching is primarily utilized in multi-pod data center fabrics and Data Center Interconnect (DCI). It allows for dividing a large data center fabric into smaller, manageable segments, streamlining provisioning tasks. For DCI, it offers controlled stretching of Layer 2 (L2) domains and optimized provisioning between sites, supporting multi-site solutions and large-scale data centers. Unlike the full mesh solutions used by many vendors, which can lead to scalability challenges, this approach offers better manageability.

The benefits of seamless EVPN-VXLAN stitching include easier architectural expansion in response to growing workloads. It optimizes pod interconnectivity, simplifying data center architecture. This method automatically creates VXLAN tunnels, easing fabric extensions. It also enhances multi-pod and DCI scaling control, reducing the total number of tunnels needed and improving Ethernet flooding efficiency. Furthermore, it offers advanced L2 virtualization capabilities, crucial for VNI translations during data center acquisitions between different organizations.

The following two graphics contrast the control tunnel utilization between the full mesh multi-POD model and the seamless EVPN-VXLAN stitching model in data centers. In a full mesh model, VXLAN tunnels terminate on each access device, passing through distribution and core devices as needed. With seamless EVPN-VXLAN stitching, new PODS (like POD3) are integrated into the existing network (POD1 and POD2) through interconnect tunnels. The access devices in POD3 establish VXLAN tunnels only with their own POD's distribution devices, serving as Interconnect Gateways (iGWs), avoiding direct tunnels with access devices in other PODs.





This approach streamlines operations. For instance, when adding POD4, the distribution devices are first connected to the interconnect VXLAN tunnels, followed by deploying access devices that form tunnels with their respective distribution iGWs, eliminating the need for direct inter-POD access device tunnels.

The seamless EVPN-VXLAN stitching solution significantly reduces the excessive VXLAN tunnels common in traditional EVPN-VXLAN setups. In full mesh models, each access device forms a VXLAN tunnel with every other access device, irrespective of POD location, leading to scalability issues with a growing number of access devices. Conversely, in the seamless model, adding a new access device, like in POD4, only necessitates VXLAN tunnels between the access device and the POD's core iGWs. This allows for simplified scheduling and control during expansion, as the new access device communicates with others in its POD through iGWs and utilizes established interconnect tunnels for cross-POD communication.

Ultimately, the seamless EVPN VXLAN stitching solution offers superior scalability and more controlled connectivity within the fabric, which is particularly beneficial for larger deployments.

In the realm of enterprise and campus network design, the strategic deployment of tunnels in VXLAN infrastructures plays a crucial role in enhancing connectivity and reducing complexity. This approach is particularly evident in the configuration of distribution and access switches within a network.

A key aspect of this design is that distribution switches are configured with tunnels to all other distribution switches within the network. This setup creates a robust mesh of connectivity, ensuring that each distribution switch can communicate directly with its peers. Such an arrangement is vital for maintaining high levels of redundancy and reliability in the network, as it allows for seamless data flow and efficient handling of traffic across different network segments.

In contrast, access switches, which connect end devices to the network, follow a more streamlined approach. These switches establish tunnels only to the distribution switches they are directly connected to. This method significantly reduces the complexity at the access layer, as each access switch manages a smaller number of tunnels. This focused connectivity not only simplifies the network architecture but also enhances performance by reducing the potential for bottlenecks and ensuring more efficient traffic handling.

Further optimization is achieved through Type-2 VXLAN to VXLAN stitching at the distribution layer. This technique plays a critical role in minimizing the number of tunnels required per access switch. By handling the stitching process at the distribution layer, the network can efficiently route traffic between different VXLAN segments without overburdening the access switches with numerous tunnels. This results in a more scalable and manageable network infrastructure, especially in environments with a large number of access switches.

Moreover, within a building, distribution switches are configured to form I-ESI (Interconnect Ethernet Segment Identifier) peers. This configuration is pivotal in creating a resilient and efficient network topology. By forming I-ESI peers, the distribution switches can work together more effectively, sharing information and balancing the load across the network. This setup not only improves the overall performance of the network but also enhances its fault tolerance, ensuring continuous operation even in the event of individual distribution switch failures.

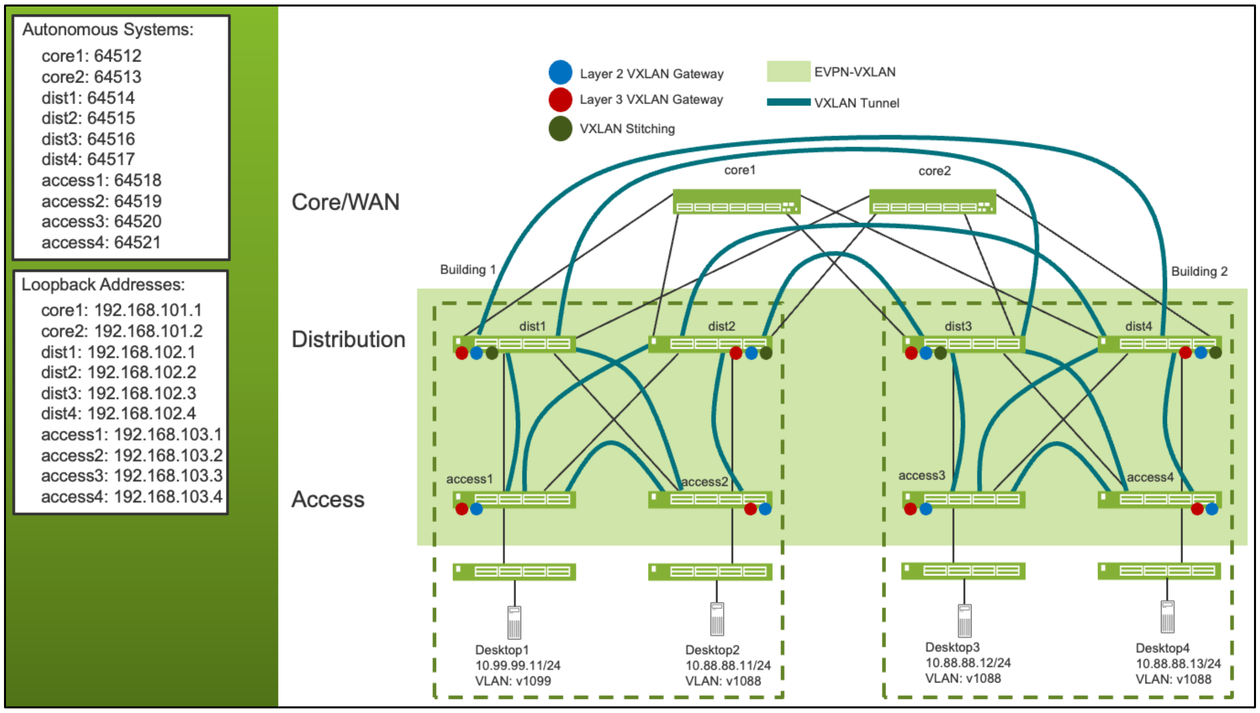
The following diagram shows tunnel formation from the first distribution switch in building one. The distribution switch forms interconnect VXLAN tunnels with all other distribution switches and normal VXLAN tunnels with the access switches in building one. The distribution switches in buildings two and three also form normal VXLAN tunnels with their respective access switches. This function reduces the number of VXLAN tunnels on one access switch from five tunnels to one.

Additional note how this topology uses the edge-routed bridging-based IP fabric as the Layer 3 and Layer 2 VXLAN gateways are found on the access and distribution switches. If this were a data center environment, then the access switches would be considered leafs and the distribution switches would be considered border leafs.

Seamless EVPN-VXLAN Stitching: Type-2 Stitching Configuration

NOTE: The configuration for EVPN-VXLAN Type-2 stitching is involved and complex. With this section of the guide, we’ll be showing the configuration from the perspective of a few devices. However, if you’re interested in the full configuration, please check out Appendix B.

The following topology is what we will be working with in this example. There are four access, four distribution, and two core devices. Each access device has a host attached to in and each host is in a specific VLAN, as shown in the topology.



Let's first have a look at the interface configurations for Dist1. Here we can see the fabric interfaces, the gigabit Ethernet interfaces, that connect Dist1 to the access devices in Building 1 and the core devices. Next, we have the loopback interface and the IRB interfaces. The loopback interface is important for overlay BGP peering and VXLAN tunnel establishment.

Note how the logical IRB interfaces are configured with unique IP addresses that are within the respective VLANs. When we look at the interface configurations for Access1, well see that it, and the rest of the access switches, are using an anycast IP address. The interface configurations for the other distribution switches are similar.

```
[edit interfaces]
root@dist1# show
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.1.1.1/31;
        }
    }
}
ge-0/0/2 {
    unit 0 {
        family inet {
            address 10.1.1.4/31;
        }
    }
}
ge-0/0/3 {
    unit 0 {
        family inet {
            address 10.1.3.0/31;
        }
    }
}
ge-0/0/4 {
    unit 0 {
        family inet {
            address 10.1.3.2/31;
        }
    }
}
irb {
    unit 1088 {
        family inet {
            address 10.88.88.5/24;
        }
    }
    unit 1099 {
        family inet {
            address 10.99.99.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 192.168.102.1/32;
        }
    }
}
```

The interface configurations for Access1 are similar to what we saw with Dist1. We have the fabric interfaces connect it to the distribution devices in building one. We also have the IRB interfaces that are used as default gateways for our locally attached hosts. Of interest is fact that the IRB interfaces are configured in an anycast manner. That is the other access devices use the same IP addressing and MAC addresses for the logical interfaces. This anycast configuration allows hosts to use any of the access devices, which is typically going to be the physically closest access device, as the default gateway. Lastly, the loopback address is important for overlay BGP peering and VXLAN tunnel establishment. One major difference is the ge-0/0/3 interface, which is the interface that connects to Desktop1 in our topology. Desktop1 is a member of VLAN v1099 and the VLAN is referenced with the interface here.

The other access devices have a similar interface configuration.

```
[edit interfaces]
root@access1# show
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.1.1.0/31;
        }
    }
}
ge-0/0/2 {
    unit 0 {
        family inet {
            address 10.1.1.2/31;
        }
    }
}
ge-0/0/3 {
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members v1099;
            }
        }
    }
}
irb {
    unit 1088 {
        family inet {
            address 10.88.88.1/24;
        }
        mac 50:04:00:04:07:01;
    }
    unit 1099 {
        family inet {
            address 10.99.99.1/24;
        }
        mac 50:04:00:04:07:00;
    }
}
lo0 {
    unit 0 {
        family inet {
            address 192.168.103.1/32;
        }
    }
}
```

The routing instance configuration on the distribution devices is where things start to get complicated. First, we use a MAC VRF instance instead of a simple VRF instance. A VRF instance would work if we were only concerned about Type-2 stitching, but as we'll be adding Type-5 stitching later, we'll need to use a MAC VRF instance here. The important parts of the configuration are called out inline with the following configuration snippet.

```
[edit routing-instances]
root@dist1# show
EVPN-VXLAN-1 {
    instance-type mac-vrf; << MAC VRF instance type
    protocols {
        evpn {
            encapsulation vxlan; << encapsulation must be set to VXLAN
            default-gateway no-gateway-community;
            extended-vni-list all; << allows all configured VNIs to be extended across the DCI
            interconnect {
                vrf-target target:1:123; << must be the same on all distribution devices
                route-distinguisher 192.168.100.1:111; << unique on all distribution devices
                esi {
                    00:11:12:13:14:15:16:17:18:11; << unique to bld1, different on bld2
                    all-active;
                }
                interconnected-vni-list all; << all VNIs are allowed over the interconnect
            }
            vni-options { << specific VNIs for this EVPN, should match on all devices
                vni 5010 {
                    vrf-target target:5010:1;
                }
                vni 5020 {
                    vrf-target target:5020:1;
                }
            }
        }
    }
    vtep-source-interface lo0.0; << source VXLAN tunnels from lo0.0
    service-type vlan-aware; << VLAN Aware service type
    route-distinguisher 192.168.102.1:1;
    vrf-target {
        target:65000:1; << the same on all devices
        auto;
    }
    vlans { << the same on all devices
        v1088 {
            vlan-id 1088;
            l3-interface irb.1088;
            vxlan {
                vni 5020;
            }
        }
        v1099 {
            vlan-id 1099;
            l3-interface irb.1099;
            vxlan {
                vni 5010;
            }
        }
    }
}
```

Before we look at the routing instance configuration for an access switch, let's look at the EVPN interconnect multihoming configuration. This configuration lets the distribution devices know about each other being interconnect gateways that are within the same building. What we see in the following configuration snippet Dist1 is peering with Dist2 in building one. Dist2 would have a similar configuration but it would peer with the loopback address of Dist1. Dist3 and Dist4 have a similar configuration for the interconnect multihoming gateways.

```
[edit protocols]
root@dist1# show evpn
interconnect-multihoming-peer-gateways 192.168.102.2;
```

The MAC VRF instance configuration for the access devices is simpler than what we saw with the distribution devices. The same parameters are present with the subtraction of the interconnect details. The important parts of the configuration are pointed out inline with the configuration snippet. This configuration is going to be almost exactly the same across all of the access switches in our topology.

```
[edit routing-instances]
root@access1# show
EVPN-VXLAN-1 {
    instance-type mac-vrf; << MAC VRF instance type
    protocols {
        evpn {
            encapsulation vxlan; << encapsulation must be set to VXLAN
            default-gateway no-gateway-community;
            extended-vni-list all;
            vni-options { << specific VNIs for this EVPN, should match on all devices
                vni 5010 {
                    vrf-target target:5010:1;
                }
                vni 5020 {
                    vrf-target target:5020:1;
                }
            }
        }
    }
    vtep-source-interface lo0.0; << source VXLAN tunnels from lo0.0
    service-type vlan-aware; << VLAN Aware service type
    interface ge-0/0/3.0; << interface that points towards connected end host
    route-distinguisher 192.168.103.1:1;
    vrf-target {
        target:65000:1; << the same on all devices
        auto;
    }
    vlans { << the same on all devices
        v1088 {
            vlan-id 1088;
            l3-interface irb.1088; << Default gateway for hosts in VLAN 1088
            vxlan {
                vni 5020;
            }
        }
        v1099 {
            vlan-id 1099;
            l3-interface irb.1099; << Default gateway for hosts in VLAN 1088
            vxlan {
                vni 5010;
            }
        }
    }
}
```

Now let's look at the BGP configurations on a distribution device. First up, let's look at the overlay configurations. The first overlay group is for the local building, Building 1, in the following output. This BGP group is fairly simple with support for EVPN routes and peering between the two local access devices. One important aspect to point out is the `vpn-apply-export` statement. This statement tells the local device to apply the user-defined or default export policies to the VPN routes before sending them out to BGP peers. This statement is vital for Type-2, and Type-5, stitching to function properly in our scenario. This BGP group should look similar on all distribution devices.

The second BGP group is the overlay group for the DCI. This overlay BGP group connects the local distribution device to the distribution devices in the remote building. As the first overlay BGP group, it is configured to support EVPN routes and uses the vpn-apply-export statement. Notice how the my-iDCI statement that we previously talked about is set as an export policy for this group. This export policy adds the iDCI community to any routes that are sent to the interconnect peers. The iDCI community contains the target:1:123 value, which recall is the VRF target of the interconnect DCI. This allows the routes to be accepted to the iDCI peers. This BGP group should look similar on all distribution devices.

```
[edit protocols bgp]
root@dist1# show
group overlay-bld-1 {
    type internal;
    multihop;
    local-address 192.168.102.1;
    family evpn {
        signaling;
    }
    cluster 10.1.1.1;
    local-as 65001;
    multipath;
    neighbor 192.168.103.1;
    neighbor 192.168.103.2;
    vpn-apply-export;
}
group overlay-dci {
    type external;
    multihop {
        no-nexthop-change;
    }
    local-address 192.168.102.1;
    family evpn {
        signaling;
    }
    export my-iDCI;
    local-as 64514;
    multipath {
        multiple-as;
    }
    neighbor 192.168.102.3 {
        peer-as 64516;
    }
    neighbor 192.168.102.4 {
        peer-as 64517;
    }
    vpn-apply-export;
}
```

Let’s next take a look at the interconnect DCI policy configuration on Dist1 that we saw in the BGP overlay interconnect group. Here we have a basic policy that matches on routes that have the iDCI community and accepts the traffic in the first term. Then, the second term rejects any other EVPN routes. Note how the iDCI community uses the interconnect VRF target value that was defined in the distribution interconnect parameters within the MAC VRF instance. The contents of this policy means that only routes with the interconnect VRF target are accepted. This policy is going to be exactly the same across all distribution devices.

```
[edit policy-options]
root@dist1# show policy-statement my-iDCI
term term1 {
    from community iDCI;
    then {
        accept;
    }
}
term term2 {
    from family evpn;
    then reject;
}

[edit policy-options]
root@dist1# show community iDCI
members target:1:123;
```

Next, let’s take a look at the underlay configuration for Dist1. The job of the underlay configuration is to simply get the loopback address of the fabric devices to where they need to go. In these underlay BGP groups we are peering with the IP address of the directly connected interfaces. Recall that the primary job of the underlay is to pass around the loopback addresses for the overlay, and the export-directs policy does just this for the underlay-core group. However, the `export-leafdirect` policy goes a little further. We’ll look at the `export-leafdirect` policy next. Note that the underlay BGP groups shown in the following output should be similar across all distribution devices.

```
[edit protocols bgp]
root@dist1# show
group underlay-bld-1 {
    type external;
    export export-leafdirect;
    local-as 64514;
    multipath {
        multiple-as;
    }
    neighbor 10.1.1.0 {
        peer-as 64518;
    }
    neighbor 10.1.1.5 {
        peer-as 64519;
    }
    vpn-apply-export;
}
group underlay-core {
    type external;
    export export-directs;
    local-as 64514;
    multipath {
        multiple-as;
    }
    neighbor 10.1.3.1 {
        peer-as 64512;
    }
    neighbor 10.1.3.3 {
        peer-as 64513;
    }
    vpn-apply-export;
}
```

The `export-leafsdirect` policy has a match criterion of `as-path-calc-length 1 orlower`. This means the policy will match routes that have an AS path length of 1 or lower. Essentially, it's looking for routes that are either directly connected or learned from a direct neighbor. In essence, this routing policy is designed to only accept routes that are directly connected or learned from a direct BGP neighbor. All other routes will be rejected. This kind of policy is often used in scenarios where a router should only advertise or accept routes from directly connected BGP neighbors and not routes that have traversed multiple autonomous systems, which might be used for controlling routing in a specific part of a network, like within a data center or a specific network segment.

This policy was necessary or Desktop1 and Desktop2 in our topology could not communicate with each other.

```
[edit policy-options]
root@dist1# show
policy-statement export-leafdirect {
    term term1 {
        from {
            as-path-calc-length 1 orlower;
        }
        then accept;
    }
    term term2 {
        then reject;
    }
}
```

The `export-directs` policy is straight forward as it simply export the routes associated with the loopback interfaces of the local device. Both of these policies are the same on all distribution devices.

```
[edit policy-options]
root@dist1# show
policy-statement export-directs {
    term 1 {
        from interface lo0.0;
        then accept;
    }
    term term2 {
        then reject;
    }
}
```

Next, let’s take a look at the BGP groups of the access devices. The following output shows the underlay BGP group for the Access1 device. This group peers with the Building 1 distribution devices and exports it’s loopback address to them with the export-directs policy. This export policy is exactly like the one we discussed in the previous output.

```
[edit protocols bgp]
root@access1# show
group underlay-bld-1 {
    type external;
    export export-directs;
    local-as 64518;
    neighbor 10.1.1.1 {
        peer-as 64514;
    }
    neighbor 10.1.1.3 {
        peer-as 64515;
    }
}
```

The following output shows the overlay BGP group for the Access1 device. This overlay group allows EVPN routes to be shared between access and distribution devices. These two BGP groups are similar across all access devices.

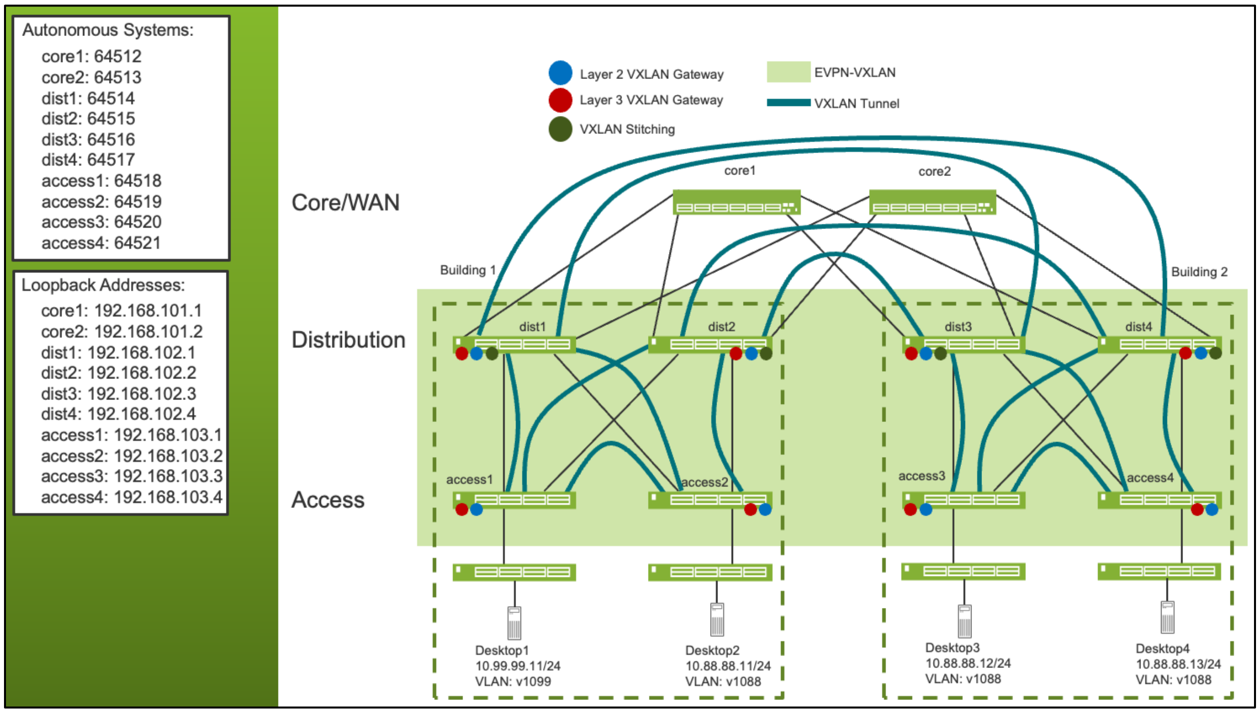
```
[edit protocols bgp]
root@access1# show
group overlay-bld-1 {
    type internal;
    multihop;
    local-address 192.168.103.1;
    family evpn {
        signaling;
    }
    local-as 65001;
    multipath {
        multiple-as;
    }
    neighbor 192.168.102.1;
    neighbor 192.168.102.2;
}
```

The following output shows the underlay BGP group for the Core1 device. This underlay group peers with the distribution devices and simply exports the local loopback address of the Core1 device and pass the loopback addresses of the distribution devices to each other. The under BGP configuration on the Core2 device is similar to what is shown in the following output.

```
[edit protocols bgp]
root@core1# show
group underlay-core {
    type external;
    export loopbacks;
    local-as 64512;
    neighbor 10.1.3.0 {
        peer-as 64514;
    }
    neighbor 10.1.3.4 {
        peer-as 64515;
    }
    neighbor 10.1.3.8 {
        peer-as 64516;
    }
    neighbor 10.1.3.12 {
        peer-as 64517;
    }
}
```

Seamless EVPN-VXLAN Stitching: Type-2 Stitching Validation

Now that everything for seamless EVPN-VXLAN Type-2 stitching is configured, we'll want to verify that the necessary routes are being distributed correctly. A good place to start is with the VXLAN tunnels. Let's look at the VXLAN tunnels from the perspective of Distribution1 and Access1.



In the following output from Distribution1, we five VXLAN tunnels, Access1, Access2, Distribution2, Distribution3, and Distribution4. Take special note of the designation under the RVTEP-Mode field for each tunnel. The VXLAN tunnels from the access switches have RNVE, which is your standard VXLAN tunnel. The Distribution2 device has the I-ESI-Peer value as it is the local iESI peer for Distribution1 in Building 1. The Distribution2 and Distribution3 devices has the value of Wan-VTEP in this field as they are the remote iESI peers in Building 2.

Also of note, is the VNIs that each device is advertising, here we can see that each device is advertising that it is participating in VNIs 5010 and 5020.

This output should appear nearly the same on each distribution device.

```
root@dist1> show ethernet-switching vxlan-tunnel-end-point remote
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx    SVTEP-Mode    ELP-SVTEP-IP
<default>                 0   192.168.102.1    lo0.0    0
 RVTEP-IP         L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.103.1    EVPN-VXLAN-1             420       vtep.32770   1315    RNVE << Access1
    VNID          MC-Group-IP
    5010          0.0.0.0
    5020          0.0.0.0
 RVTEP-IP         L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.102.2    EVPN-VXLAN-1             419       vtep.32769   1310    I-ESI-Peer << Distribution2
```

```
    VNID           MC-Group-IP
    5010           0.0.0.0
    5020           0.0.0.0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.103.2     EVPN-VXLAN-1             423       vtep.32773   1328    RNVE << Access2
    VNID           MC-Group-IP
    5010           0.0.0.0
    5020           0.0.0.0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.102.3     EVPN-VXLAN-1             421       vtep.32771   1316    Wan-VTEP << Distribution3
    VNID           MC-Group-IP
    5010           0.0.0.0
    5020           0.0.0.0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.102.4     EVPN-VXLAN-1             422       vtep.32772   1321    Wan-VTEP << Distribution4
    VNID           MC-Group-IP
    5010           0.0.0.0
    5020           0.0.0.0
```

The following output shows the VXLAN tunnels from the perspective of Access1. Notice how Access1 only has regular VXLAN tunnels, as noted by the RNVE designation, from Distribution1, Distribution2, and Access2. Access1 doesn't have any tunnels coming from the devices in Building 2.

This output should appear nearly the same on each access device.

```
root@access1> show ethernet-switching vxlan-tunnel-end-point remote
Logical System Name       Id   SVTEP-IP         IFL    L3-Idx    SVTEP-Mode    ELP-SVTEP-IP
<default>                 0    192.168.103.1    lo0.0    0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.102.1     EVPN-VXLAN-1             554       vtep.32769   1963    RNVE << Distribution1
    VNID           MC-Group-IP
    5020           0.0.0.0
    5010           0.0.0.0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.102.2     EVPN-VXLAN-1             555       vtep.32770   1970    RNVE << Distribution2
    VNID           MC-Group-IP
    5020           0.0.0.0
    5010           0.0.0.0
 RVTEP-IP          L2-RTT                   IFL-Idx   Interface    NH-Id   RVTEP-Mode  ELP-IP
Flags
 192.168.103.2     EVPN-VXLAN-1             556       vtep.32771   1971    RNVE << Access2
    VNID           MC-Group-IP
    5020           0.0.0.0
    5010           0.0.0.0
```

The following command allows us to verify the DCI connection. The interconnect DCI parameters, as well as the EVPN route statistics, are shown.

The following output shows the interconnect route distinguisher, interconnect VRF import and export policies, as well as the interconnect VRF import and export route targets of spine distribution1. The interconnect VRF import and export policies were automatically generated based on the interconnect VRF target

```
root@dist1> show evpn instance dci EVPN-VXLAN-1
Instance: EVPN-VXLAN-1
  EVPN-Interconnect:
    Route-distinguisher: 192.168.100.1:111
    Vrf-import: [ __evpn-ic-import-EVPN-VXLAN-1-internal__ ]
    Vrf-export: [ __evpn-ic-export-EVPN-VXLAN-1-internal__ ]
    Vrf-import-target: [ target:1:123 ]
    Vrf-export-target: [ target:1:123 ]
  DCI route stats                      Local
    AD  route advertisements:                1
    IM  route advertisements:                2
    MAC route advertisements:               12
    MAC+IP route advertisements:            12
    ES  route advertisements:                0
    SG  Proxy route advertisements:          0
```

The following output lets you track where MAC addresses come from. The output shows, from the perspective of Distribution1, that the MAC addresses are coming from the remote interconnect ESI of 00:21:22:23:24:25:26:27:28:22, which is the interconnect ESI of Building 2. The output also shows that these MAC addresses are associated with the 5010 and 5020 VNIs.

```
root@dist1> show evpn database origin dci-remote instance EVPN-VXLAN-1
Instance: EVPN-VXLAN-1
VLAN  DomainId  MAC address        Active source                  Timestamp        IP address
     5010       2c:6b:f5:6c:43:f5  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:21  10.99.99.7
     5010       2c:6b:f5:6c:43:f6  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:14  10.99.99.8
     5010       52:54:00:9b:56:8e  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:21  10.99.99.12
     5020       2c:6b:f5:6c:43:f5  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:21  10.88.88.7
     5020       2c:6b:f5:6c:43:f6  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:14  10.88.88.8
     5020       52:54:00:e7:2c:74  00:21:22:23:24:25:26:27:28:22  Dec 07 20:13:21  10.88.88.12
```

The following output shows something similar to the previous output but from the perspective of Access1. Let's specifically look at the output Desktop4, which uses IP 10.88.88.12 and is in building2. The active source for this entry is 00:11:12:13:14:15:16:17:18:11, which is the iESI configured on Distribution1 and Distribution2. This output means that to reach Desktop4 from Access1, Access1 needs to send the traffic through the VXLAN tunnel that connects to Distribution1 or Distribution2. From there, the traffic is sent from one of the building1 distribution devices to one of the building2 distribution devices, and finally to Access4, which is directly connected to Desktop4.

Now, let's look at the entry for Desktop2, which uses IP 10.88.88.11. The active source is 192.168.103.2, which is the VTEP of Access2.

Also of interest, is the entry for Desktop1, which uses IP 10.99.99.11. The active source shows the local ge-0/0/3 interface, as Desktop is directly connected to Access1 on that interface.

```
root@access1> show evpn database instance EVPN-VXLAN-1
Instance: EVPN-VXLAN-1
VLAN  DomainId  MAC address        Active source                  Timestamp        IP address
     5010       2c:6b:f5:6c:43:f3  192.168.102.1                  Dec 07 20:13:17  10.99.99.5
     5010       2c:6b:f5:6c:43:f4  192.168.102.2                  Dec 07 20:13:19  10.99.99.6
     5010       2c:6b:f5:6c:43:f5  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:19  10.99.99.7
     5010       2c:6b:f5:6c:43:f6  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:19  10.99.99.8
     5010       50:04:00:04:07:00  irb.1099                       Dec 07 20:12:58  10.99.99.1
     5010       52:54:00:9b:56:8e  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:21  10.99.99.12
     5010       52:54:00:a1:e9:a8  ge-0/0/3.0                     Dec 07 20:12:58  10.99.99.11
     5020       2c:6b:f5:6c:43:f3  192.168.102.1                  Dec 07 20:13:17  10.88.88.5
     5020       2c:6b:f5:6c:43:f4  192.168.102.2                  Dec 07 20:13:19  10.88.88.6
     5020       2c:6b:f5:6c:43:f5  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:19  10.88.88.7
     5020       2c:6b:f5:6c:43:f6  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:19  10.88.88.8
     5020       50:04:00:04:07:01  irb.1088                       Dec 07 20:13:18  10.88.88.1
     5020       52:54:00:e7:2c:74  00:11:12:13:14:15:16:17:18:11  Dec 07 20:13:19  10.88.88.12
     5020       52:54:00:f5:6b:1f  192.168.103.2                  Dec 07 20:13:19  10.88.88.11
```

The following output shows the details of the MAC address of Desktop1 from the perspective of Distribution1. Remember that Desktop1 is located in the same building as Distribution1. Here we can see that a DCI route is created for this route and advertised across the interconnect to the distribution devices in Building 2.

```
root@dist1> show evpn database mac-address 52:54:00:a1:e9:a8 extensive
Instance: EVPN-VXLAN-1

VN Identifier: 5010, MAC address: 52:54:00:a1:e9:a8
  State: 0x0
  Source: 192.168.103.1, Rank: 1, Status: Active
    Mobility sequence number: 0 (minimum origin address 192.168.103.1)
    Timestamp: Dec 07 20:13:13.075200 (0x65722759)
    State: <Remote-To-Local-Adv-Done>
    MAC advertisement route status: Not created (no local state present)
    Interconn advertisement route status: DCI route created << remote DCI route is created for it
    IP address: 10.99.99.11
      Interconn advertisement route status: DCI route created << remote DCI route is created for it
    History db:
      Time                       Event
      Dec  7 20:13:13.074 2023   192.168.103.1 : Remote peer 192.168.103.1 created
      Dec  7 20:13:13.075 2023   192.168.103.1 : Created
      Dec  7 20:13:13.077 2023   Updating output state (change flags 0x1 <ESI-Added>)
      Dec  7 20:13:13.077 2023   Active ESI changing (not assigned -> 192.168.103.1)
      Dec  7 20:13:13.078 2023   192.168.103.1 : 10.99.99.11 Selected IRB interface nexthop
      Dec  7 20:13:13.078 2023   192.168.103.1 : 10.99.99.11 Reject remote ip host route 10.99.99.11
in L3 context master since no remote-ip-host-routes configured
```

Now let's look at a MAC address for a route that was learned from Building 2. The following output uses the MAC address for Desktop4, which uses IP 10.88.88.12. Here we see that the route comes from the distribution devices in Building 2.

```
root@dist1> show evpn database mac-address 52:54:00:e7:2c:74 extensive
Instance: EVPN-VXLAN-1

VN Identifier: 5020, MAC address: 52:54:00:e7:2c:74
  State: 0x0
  Source: 00:21:22:23:24:25:26:27:28:22, Rank: 1, Status: Active
    Remote origin: 192.168.102.3 << Comes from Distribution3
    Remote state: <Mac-Only-Adv Interconnect-DC>
    Remote origin: 192.168.102.4 << Comes from Distribution4
    Remote state: <Mac-Only-Adv Interconnect-DC>
    Mobility sequence number: 0 (minimum origin address 192.168.102.3)
    Timestamp: Dec 07 20:13:21.391018 (0x65722761)
    State: <Remote-To-Local-Adv-Done>
    MAC advertisement route status: Not created (no local state present)
    Interconn advertisement route status: DC route created << local DCI route is created for it
    IP address: 10.88.88.12
    Flags: <Interconnect-DC>
      Remote origin: 192.168.102.3 << Comes from Distribution3
      Remote state: <Sent-to-l2ald Interconnect-DC>
      Remote origin: 192.168.102.4 << Comes from Distribution4
      Remote state: <Sent-to-l2ald Interconnect-DC>
      Interconn advertisement route status: DC route created << local DC route is created for it
    History db:
      Time                       Event
      Dec  7 20:13:14.592 2023   00:21:22:23:24:25:26:27:28:22 : Remote peer 192.168.102.4 created
      Dec  7 20:13:14.592 2023   00:21:22:23:24:25:26:27:28:22 : Created
      Dec  7 20:13:14.594 2023   Updating output state (change flags 0x1 <ESI-Added>)
      Dec  7 20:13:14.594 2023   Active ESI changing (not assigned -> 00:21:22:23:24:25:26:27:28:22)
      Dec  7 20:13:14.595 2023   00:21:22:23:24:25:26:27:28:22 : 10.88.88.12 Selected IRB interface
nexthop
      Dec  7 20:13:14.595 2023   00:21:22:23:24:25:26:27:28:22 : 10.88.88.12 Reject remote ip host
route 10.88.88.12 in L3 context master since no remote-ip-host-routes configured
      Dec  7 20:13:16.278 2023   00:21:22:23:24:25:26:27:28:22 : Remote peer 192.168.102.3 created
      Dec  7 20:13:16.282 2023   00:21:22:23:24:25:26:27:28:22 : Updating output state (change flags
0x10080 <ESI-Peer-Added ESI-Remote-Peer-Com-Chg>)
      Dec  7 20:13:16.282 2023   00:21:22:23:24:25:26:27:28:22 : 10.88.88.12 Selected IRB interface
nexthop
      Dec  7 20:13:16.283 2023   00:21:22:23:24:25:26:27:28:22 : 10.88.88.12 Reject remote ip host
```

```
route 10.88.88.12 in L3 context master since no remote-ip-host-routes configured
```

Let’s take a closer look at the EVPN route for the MAC address that is associated with Desktop4. Here we can see that this is a Type-2 route that originates from Distribution3 (192.168.100.3), with VNI 5020. Also the additional important items in the output are shown, such as the interconnect VRF target and the iESI from Building 2.

```
root@dist1> show route table EVPN-VXLAN-1.evpn.0 evpn-mac-address 52:54:00:e7:2c:74 extensive

EVPN-VXLAN-1.evpn.0: 98 destinations, 194 routes (98 active, 0 holddown, 0 hidden)
2:192.168.100.3:333::5020::52:54:00:e7:2c:74/304 MAC/IP (3 entries, 1 announced)
        *BGP    Preference: 170/-101
                Route Distinguisher: 192.168.100.3:333
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7ca7214
                Next-hop reference count: 72
                Kernel Table Id: 0
                Source: 192.168.102.3
                Protocol next hop: 192.168.102.3 << Protocol next hop is Distribution3
                Label operation: Push 313
                Label TTL action: prop-ttl
                Load balance label: Label 313: None;
                Indirect next hop: 0x2 no-forward INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Secondary Active Ext>
                Peer AS: 64516
                Age: 1:22:16    Metric2: 0
                Validation State: unverified
                Task: BGP_64516_64514.192.168.102.3
                Announcement bits (1): 0-EVPN-VXLAN-1-evpn
                AS path: 64516 I
                Communities: target:1:123 encapsulation:vxlan(0x8) << Interconnect VRF target
                Import Accepted
                Route Label: 5020 << VNI
                ESI: 00:21:22:23:24:25:26:27:28:22 << iESI from Building 2
                Localpref: 100
                Router ID: 192.168.102.3
                Primary Routing Table: bgp.evpn.0
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.102.3 ResolvState: Resolved
                        Label operation: Push 313
                        Label TTL action: prop-ttl
                        Load balance label: Label 313: None;
                        Indirect next hop: 0x2 no-forward INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 2
                                Next hop type: Router
                                Next hop: 10.1.3.3 via ge-0/0/4.0
                                Session Id: 0
                                Next hop: 10.1.3.1 via ge-0/0/3.0
                                Session Id: 0
                                192.168.102.3/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 2
                                        Next hop type: Router
                                        Next hop: 10.1.3.3 via ge-0/0/4.0
                                        Session Id: 0
                                        Next hop: 10.1.3.1 via ge-0/0/3.0
                                        Session Id: 0
```

Seamless EVPN-VXLAN Stitching: Type-5 Stitching for Enhanced Scale and Efficiency

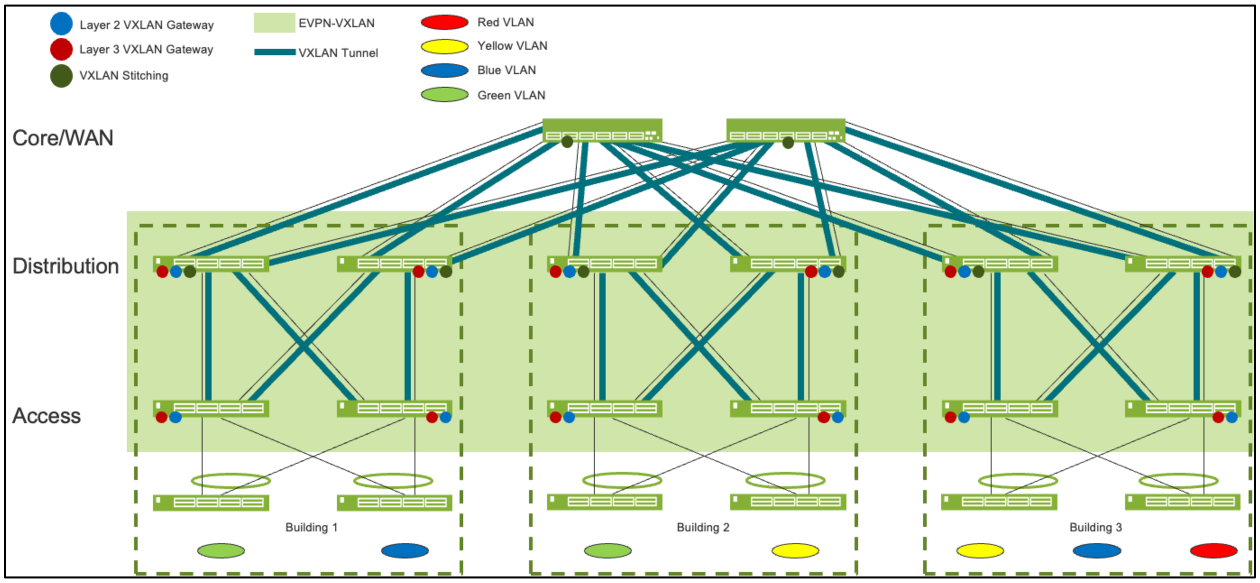
In the evolving landscape of network technology, the implementation of Type-5 stitching in EVPN-VXLAN infrastructures marks a significant advancement, particularly in addressing the challenges of scalability and efficiency. This approach is tailored to significantly enhance the capacity of the network fabric, allowing it to support over 250,000 host routes. Such an increase in scale is pivotal for large-scale networks, catering to extensive enterprise environments with multiple buildings or campuses.

One of the key challenges in expanding network scale is the limitation of Access devices, particularly EX devices, in handling a large number of host routes. These devices, while efficient in their designated roles, are not equipped to manage the heightened scale that comes with extensive network expansion. To mitigate this, the core network devices, specifically the MX series, are leveraged to handle the entire scale of the campus fabric. By centralizing the scale management at the core, the network achieves a more streamlined and efficient operation, ensuring that all buildings within the campus are adequately serviced.

A crucial aspect of this implementation is the strategic configuration of policies at the core. These policies are designed to allow host routes into specific buildings only when the corresponding VLAN exists. This selective approach ensures that network resources are optimized, and only necessary routes are maintained and propagated. It's a method that not only enhances efficiency but also reinforces network security by controlling route dissemination based on VLAN presence.

Furthermore, for VLANs that are not present in certain buildings, the network's core adopts a different strategy. Instead of broadcasting all host routes, the core limits its advertisements to only the aggregate prefix for these VLANs. This method significantly reduces the overhead and complexity in the network, ensuring that the routes are concise and relevant to the specific requirements of each building or network segment.

In conclusion, the adoption of Type-5 stitching in EVPN-VXLAN networks marks a substantial step forward in addressing the challenges of large-scale network management. By efficiently leveraging core devices for scale management and employing smart policy configurations, networks can achieve unprecedented levels of scalability and efficiency, essential for modern enterprise environments.

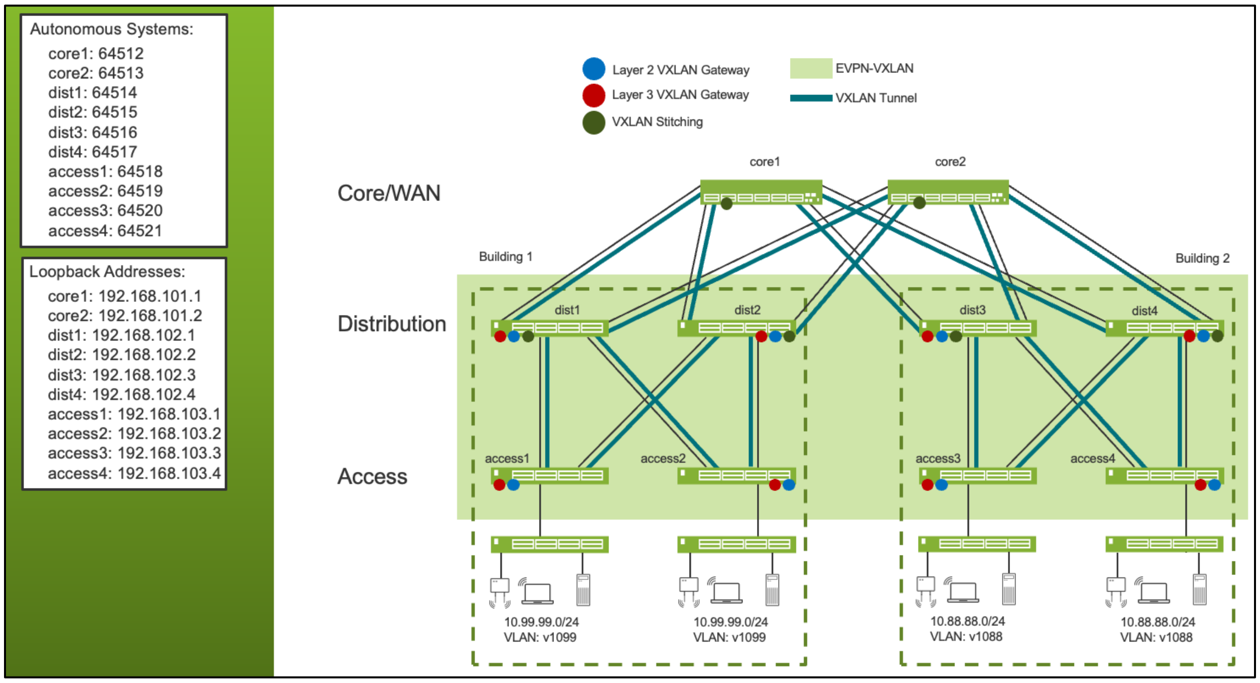


Seamless EVPN-VXLAN Stitching: Type-5 Stitching Configuration

NOTE: The configuration for EVPN-VXLAN Type-5 stitching is involved and complex. With this section of the guide, we'll be showing the configuration from the perspective of a few devices. However, if you're interested in the full configuration, please check out Appendix C.

To begin our seamless EVPN-VXLAN Type-5 stitching scenario, let's take a look at our topology. The topology has changed a bit as Desktop2 has changed to VLAN v1088 and is using IP 10.88.88.11. Desktop3 and Desktop4 are still part of VLAN v1088, but their IP addresses have changed to 10.88.88.12 and 10.88.88.13, respectively. Desktop1 hasn't changed anything as it remains in VLAN v1099 with IP 10.99.99.11.

This change is necessary as it allows us to test the Type-5 stitching correctly, as Building 1 has both VLANs and Building 2 only has VLAN v1088. This VLAN deployment means that host routes for both VLANs will be stored on the core devices, and Building 2 will only have host routes for VLAN v1088. Building 1 will have an aggregate route for the IP range in VLAN v1088 to be able to reach hosts in Building 2.



Let's first have a look at the interface configurations for Dist1. The interfaces are the same as the Type-2 stitching configuration, so we won't spend any additional time discussing them. They are included here so you can quickly reference them. Recall that the other distribution devices have a similar interface configuration.

```
[edit interfaces]
root@dist1# show
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.1.1.1/31;
        }
    }
}
ge-0/0/2 {
    unit 0 {
        family inet {
            address 10.1.1.4/31;
        }
    }
}
ge-0/0/3 {
    unit 0 {
        family inet {
            address 10.1.3.0/31;
        }
    }
}
ge-0/0/4 {
    unit 0 {
        family inet {
            address 10.1.3.2/31;
        }
    }
}
irb {
    unit 1088 {
        family inet {
            address 10.88.88.5/24;
        }
    }
    unit 1099 {
        family inet {
            address 10.99.99.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 192.168.102.1/32;
        }
    }
}
```

The Access1 interface configuration is the same as the Type-2 stitching configuration and Access2 is similar. However, Access3 and Access4 do not have the IRB configuration for the 10.99.99.0/24 network as there is only VLAN v1088 in Building 2.

```
[edit interfaces]
root@access1# show
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.1.1.0/31;
        }
    }
}
ge-0/0/2 {
    unit 0 {
        family inet {
            address 10.1.1.2/31;
        }
    }
}
ge-0/0/3 {
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members v1099;
            }
        }
    }
}
irb {
    unit 1088 {
        family inet {
            address 10.88.88.1/24;
        }
        mac 50:04:00:04:07:01;
    }
    unit 1099 {
        family inet {
            address 10.99.99.1/24;
        }
        mac 50:04:00:04:07:00;
    }
}
lo0 {
    unit 0 {
        family inet {
            address 192.168.103.1/32;
        }
    }
}
```

The MAC VRF instance is the same on the all distribution devices is exactly the same as what we saw with the Type-2 stitching configuration. Note that the interconnect part of the configuration is necessary so that Desktop1 (10.88.88.11) can communicate with Desktop3 (10.88.88.12) and Desktop4 (10.88.88.13) over Layer 2.

The MAC VRF instances on the distribution devices are similar.

```
[edit routing-instances]
root@dist1# show
EVPN-VXLAN-1 {
    instance-type mac-vrf; << MAC VRF instance type
    protocols {
        evpn {
            encapsulation vxlan; << encapsulation must be set to VXLAN
            default-gateway no-gateway-community;
            extended-vni-list all; << allows all configured VNIs to be extended across the DCI
            interconnect {
                vrf-target target:1:123; << must be the same on all distribution devices
                route-distinguisher 192.168.100.1:111; << unique on all distribution devices
                esi {
                    00:11:12:13:14:15:16:17:18:11; << unique to bld1, different on bld2
                    all-active;
                }
                interconnected-vni-list all; << all VNIs are allowed over the interconnect
            }
            vni-options { << specific VNIs for this EVPN, should match on all devices
                vni 5010 {
                    vrf-target target:5010:1;
                }
                vni 5020 {
                    vrf-target target:5020:1;
                }
            }
        }
    }
    vtep-source-interface lo0.0; << source VXLAN tunnels from lo0.0
    service-type vlan-aware; << VLAN Aware service type
    route-distinguisher 192.168.102.1:1;
    vrf-target {
        target:65000:1; << the same on all devices
        auto;
    }
    vlans { << the same on all devices
        v1088 {
            vlan-id 1088;
            l3-interface irb.1088;
            vxlan {
                vni 5020;
            }
        }
        v1099 {
            vlan-id 1099;
            l3-interface irb.1099;
            vxlan {
                vni 5010;
            }
        }
    }
}
```

The MAC VRF instance configuration for the access devices is similar to what we saw with the Type-2 stitching configuration. Access1 and Access2 have the same configuration that is shown below. However, Access3 and Access4 have similar configurations, but they are lacking VLAN v1099 as that VLAN is not part of Building 2.

```
[edit routing-instances]
root@access1# show
EVPN-VXLAN-1 {
    instance-type mac-vrf; << MAC VRF instance type
    protocols {
        evpn {
            encapsulation vxlan; << encapsulation must be set to VXLAN
            default-gateway no-gateway-community;
            extended-vni-list all;
            vni-options { << specific VNIs for this EVPN, should match on all devices
                vni 5010 {
                    vrf-target target:5010:1;
                }
                vni 5020 {
                    vrf-target target:5020:1;
                }
            }
        }
    }
    vtep-source-interface lo0.0; << source VXLAN tunnels from lo0.0
    service-type vlan-aware; << VLAN Aware service type
    interface ge-0/0/3.0; << interface that points towards connected end host
    route-distinguisher 192.168.103.1:1;
    vrf-target {
        target:65000:1; << the same on all devices
        auto;
    }
    vlans {
        v1088 {
            vlan-id 1088;
            l3-interface irb.1088; << Default gateway for hosts in VLAN 1088
            vxlan {
                vni 5020;
            }
        }
        v1099 {
            vlan-id 1099;
            l3-interface irb.1099; << Default gateway for hosts in VLAN 1088
            vxlan {
                vni 5010;
            }
        }
    }
}
```

The following output shows the Type-5 routing instance. Here we are using a VRF instance that allows us to configure Type-5 route advertisements. The IRB interfaces are placed in the routing instance so the associated direct routes can be advertised as Type-5 routes as well. The EVPN interconnect properties are now found here. We have the VRF import policy that will import the routes based on the interconnect community, which we will talk about shortly. Then, the interconnect route target must match on the distribution and core devices. The regular VRF route target must match on all access, distribution, and core devices. The IP prefix routes section describes how the routes will be advertised as Type-5 routes. We configure a matching VNI as well as specify an export policy.

```
[edit routing-instances T5]
root@dist1# show
instance-type vrf; << VRF instance type
routing-options {
    multipath; << multipath with auto-export of unicast routes
    auto-export {
        family inet {
            unicast;
        }
    }
}
protocols {
    evpn {
        interconnect { << Interconnect to the core devices
            vrf-import t5-import; << import policy based on a user-defined community
            vrf-target target:300:1; << IC VRF target that matches with the core devices
            route-distinguisher 192.168.102.1:555; << unique RD
        }
        ip-prefix-routes { << Type-5 route advertisements
            advertise direct-nexthop;
            encapsulation vxlan; << VXLAN encapsulation
            vni 50001; << Matching NVI with access, distribution, and core devices
            export EXPORT_CORE_ROUTES; << Policy that describes the routes to be exported as Type-5
        }
    }
}
vtep-source-interface lo0.0;
interface irb.1088; << IRB direct routes will be advertised as Type-5 routes
interface irb.1099; <<
route-distinguisher 192.168.102.1:55;
vrf-target target:100:556; << VRF target matches on all access, distribution, and core devices
```

The following policy is referenced in the previous output as the interconnect import policy. This policy matches routes based on the `t5-ic` community that has the value of target:200:1. The routes that will be tagged with this community are the host routes that will be received from the core devices.

```
[edit policy-options]
root@dist1# show policy-statement t5-import
term 1 {
    from community t5-ic;
    then accept;
}
term 2 {
    then reject;
}

[edit policy-options]
root@dist1# show community t5-ic
members target:200:1;
```

Now let’s look at the policy that is used to export routes as Type-5 routes to the core devices. The first two terms in the policy matches on host routes for VLANs v1088 and v1099. This is important as both VLANs are present in Building 1. This policy is the same on Distribution2. This policy on Distribution3 and Distribution4 is similar, but it does not have a term to export host routes for VLAN v1099 as it is not present in Building 2.

```
[edit policy-options]
root@dist1# show policy-statement EXPORT_CORE_ROUTES
term 1 {
    from {
        route-filter 10.99.0.0/16 orlonger;
    }
    then accept;
}
term 2 {
    from {
        route-filter 10.88.0.0/16 orlonger;
    }
    then accept;
}
term 3 {
    from {
        route-filter 0.0.0.0/0 upto /31 accept;
    }
    then reject;
}
```

Let’s move on to Access1 and examine the Type-5 instance there. The main difference between what we see here and what is seen on the distribution devices is the absence of the interconnect parameters.

This configuration is the same on Access2. The configuration lacks the IRB interface for VLAN v1099 on Access3 and Access4 as that VLAN is not present in building2.

```
[edit routing-instances T5]
root@access1# show
instance-type vrf; << VRF instance type
routing-options {
    multipath; << multipath with auto-export of unicast routes
    auto-export {
        family inet {
            unicast;
        }
    }
}
protocols {
    evpn {
        ip-prefix-routes { << Type-5 route advertisements
            advertise direct-nexthop;
            encapsulation vxlan;
            vni 50001; << Matching NVI with access, distribution, and core devices
            export EXPORT_HOST_ROUTES; << Policy that describes the routes to be exported as Type-5
        }
    }
}
vtep-source-interface lo0.0;
interface irb.1088; << IRB direct routes will be advertised as Type-5 routes
interface irb.1099; <<
route-distinguisher 192.168.103.1:55;
vrf-target target:100:556; << VRF target matches on all access, distribution, and core devices
```

The `EXPORT_HOST_ROUTES` policy enables the access devices to export into host routes they have towards the distribution devices. This policy only has one term that matches EVPN host routes. This policy is the same on all access devices.

```
[edit policy-options]
root@access1# show policy-statement EXPORT_HOST_ROUTES
```

```
term TERM_1 {
    from {
        protocol evpn;
        route-filter 0.0.0.0/0 prefix-length-range /32-/32;
    }
    then accept;
}
```

Let’s look at the T5 instances on the core devices. The following output shows the T5 instance on Core1. The EVPN details, interconnect and IP prefix routes, match up with what was seen on the distribution devices’ T5 instances. On particular note is the aggregate route configuration. There are two aggregate routes configured here for the v1088 and v1099 VLANs. These aggregate routes are important as the core devices will be advertising them through the interconnect to both buildings.

The T5 configuration for both core devices is similar

```
[edit policy-options]
root@core1# top show routing-instances
T5 {
    instance-type vrf;
    routing-options {
        aggregate {
            route 10.88.88.0/24; << Aggregate route that represents VLAN v1099
            route 10.99.99.0/24; << Aggregate route that represents VLAN v1088
        }
        multipath;
        auto-export {
            family inet {
                unicast;
            }
        }
    }
    protocols {
        evpn {
            interconnect {
                vrf-target target:300:1; << IC VRF target that matches with the distribution devices
                route-distinguisher 192.168.101.1:555;
            }
            ip-prefix-routes {
                advertise direct-nexthop;
                encapsulation vxlan;
                vni 50001; << VNI that matches with the distribution devices
            }
        }
    }
    vtep-source-interface lo0.0;
    route-distinguisher 192.168.101.1:55;
    vrf-target target:200:1; << VRF target matches on all access, distribution, and core devices
}
```

Now let’s look at the BGP configurations on a distribution device. The local overlay and DCI groups do not change from what we saw in the Type-2 stitching example. We’ve included them here for easy reference.

```
[edit protocols bgp]
root@dist1# show
group overlay-bld-1 {
    type internal;
    multihop;
    local-address 192.168.102.1;
    family evpn {
        signaling;
    }
    cluster 10.1.1.1;
    local-as 65001;
    multipath;
    neighbor 192.168.103.1;
    neighbor 192.168.103.2;
    vpn-apply-export;
}
group overlay-dci {
    type external;
    multihop {
        no-nexthop-change;
    }
    local-address 192.168.102.1;
    family evpn {
        signaling;
    }
    export my-iDCI;
    local-as 64514;
    multipath {
        multiple-as;
    }
    neighbor 192.168.102.3 {
        peer-as 64516;
    }
    neighbor 192.168.102.4 {
        peer-as 64517;
    }
    vpn-apply-export;
}
```

What is new is the iBGP overlay group that connects to the core devices. This iBGP overlay group is similar to the `overlay-dci` BGP group as it shares EVPN routes. The export policy exports EVPN routes based on the T5 community. We'll examine this policy next.

```
[edit protocols bgp]
root@dist1# show
group overlay-dis-to-core {
    type internal;
    multihop {
        no-nexthop-change;
    }
    local-address 192.168.102.1;
    family evpn {
        signaling;
    }
    export my-t5iDCI;
    local-as 65002;
    multipath {
        multiple-as;
    }
    neighbor 192.168.101.1;
    neighbor 192.168.101.2;
    vpn-apply-export;
}
```

Let's next take a look at the interconnect T5 DCI policy configuration on Dist1. Here we have a basic policy that matches on routes that have the T5 iDCI community and accepts the traffic in the first term. Then, the second term rejects any other EVPN routes. Note how the T5 iDCI community uses the T5 interconnect route target value that was defined in the distribution interconnect parameters within the T5 instance. The contents of this policy mean that only routes with the T5 interconnect route target are accepted. This policy is exactly the same across all distribution devices.

```
[edit policy-options]
root@dist1# show policy-statement my-t5iDCI
term 1 {
    from community t5-comm;
    then accept;
}
term 2 {
    from family evpn;
    then reject;
}

[edit policy-options]
root@dist1# show community t5-comm
members target:300:1;
```

NOTE: There is no change to the underlay BGP groups that we showed in the Type-2 stitching part on any of the access, distribution, and core devices. For that reason, we won't repeat their configurations here.

NOTE: There is no change to the overlay BGP groups that we showed in the Type-2 stitching part on any of the access devices. For that reason, we won't repeat their configurations here.

Let's look at the overlay iBGP group on the core devices. The following output shows the overlay group for Core1. It's similar to what the corresponding overlay BGP group looks like on the distribution devices, with there being a difference in the export policies. Each iBGP peer gets a specific export policy. The iBGP peers in Building 1 get a different export policy than the iBGP peers in Building 1. We'll look at these export policies next.

```
[edit protocols bgp]
root@core1# show
group overlay-dis-to-core {
    type internal;
    multihop {
        no-nexthop-change;
    }
    local-address 192.168.101.1;
    family evpn {
        signaling;
    }
    local-as 65002;
    multipath {
        multiple-as;
    }
    neighbor 192.168.102.1 {
        export from-bld2;
    }
    neighbor 192.168.102.2 {
        export from-bld2;
    }
    neighbor 192.168.102.3 {
        export from-bld1;
    }
    neighbor 192.168.102.4 {
        export from-bld1;
    }
    vpn-apply-export;
}
```

The following output shows the two export policies that are used in the core overlay iBGP group. The first policy is applied to the Distribution3 and Distribution4 BGP peers in the previous example. As Building 2 doesn't have VLAN v1099, there is no need to send host routes from the VLAN into Building 2. To this end, an aggregate route that represents the 10.99.99.0/24 route is sent to these iBGP peers.

Building 1 has hosts that are in VLANs v1099 and v1088, so the second policy matches any host routes in the 10.88.88.0/24 prefix and also any aggregate routes. Recall that the core devices have aggregate routes for the 10.99.99.0/24 and 10.88.88.0/24 prefixes. All other routes are rejected.

This policy is the same on both core devices.

```
[edit policy-options]
root@core1# show
policy-statement from-bld1 {
    term 1 {
        from protocol aggregate;
        then accept;
    }
    term 2 {
        from {
            route-filter 0.0.0.0/0 prefix-length-range /32-/32;
        }
        then reject;
    }
}
policy-statement from-bld2 {
    term 1 {
        from {
            route-filter 10.88.88.0/24 orlonger;
        }
        then accept;
    }
    term 2 {
        from protocol aggregate;
        then accept;
    }
    term 3 {
        from {
            route-filter 0.0.0.0/0 prefix-length-range /32-/32;
        }
        then reject;
    }
}
```

Seamless EVPN-VXLAN Stitching: Type-5 Host Route Suppression

EVPN Type 5 stitching allows seamless connectivity across data centers linked via a WAN. Gateway devices (GWs) play a crucial role, utilizing BGP EVPN signaling to redistribute routes and establish end-to-end forwarding paths.

To optimize resource usage, host routes (/32 IPv4, /128 IPv6) learned from the WAN are selectively redistributed to the local data center. Here's how it works:

1. Verification: The GW checks if the received host route belongs to a subnet within a Layer 2 stretched bridge domain in the local data center.
2. Suppression (if configured): If the `suppress-host-routes-from-dci-to-dc` parameter is enabled and the host route doesn't match a stretched subnet, redistribution to the data center is blocked.
3. Standard Redistribution: If suppression isn't active or the route matches a stretched subnet, standard EVPN Type 5 redistribution occurs.

The following output shows a configuration sample for the `suppress-host-routes-from-dci-to-dc` statement.

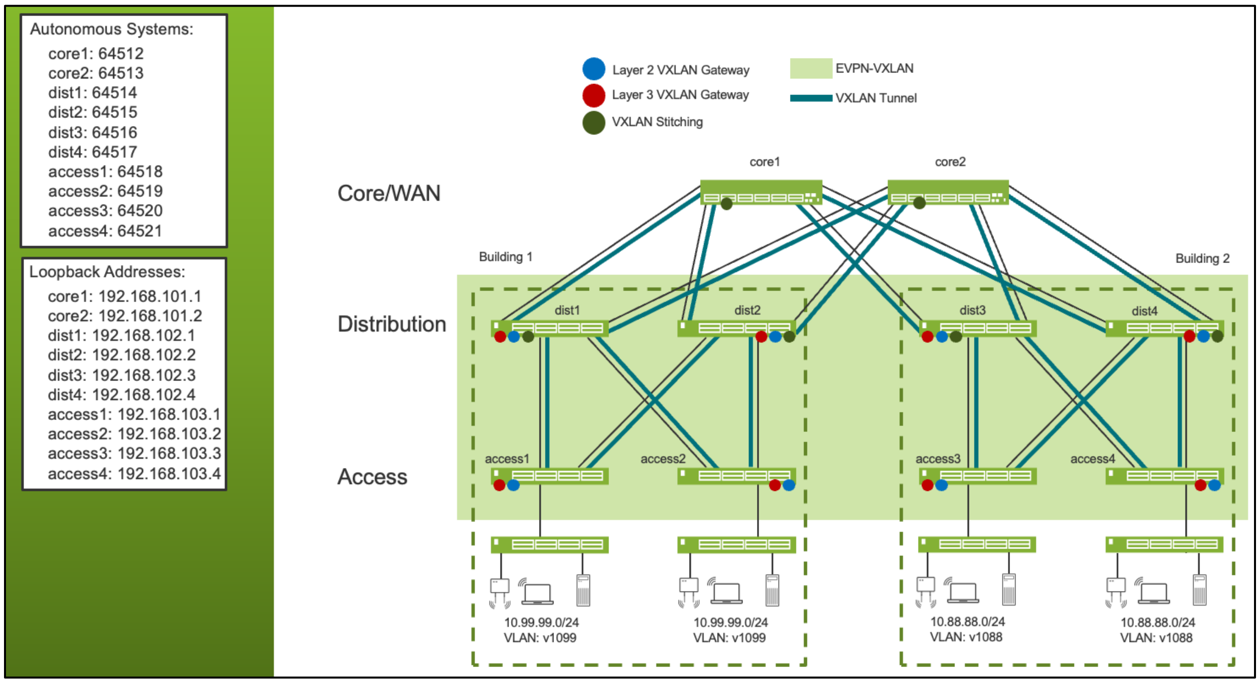
NOTE: Type 5 host route suppression is only shown here as an example. This configuration snippet is not part of this document's seamless EVPN-VXLAN Type 5 stitching use case.

```
user@dis1> show configuration routing-instances VRF-100 protocols evpn
interconnect {
    suppress-host-routes-from-dci-to-dc;
    vrf-target target:200:200;
    route-distinguisher 10.255.1.11:110;
}
ip-prefix-routes {
    advertise direct-nexthop;
    encapsulation vxlan;
    vni 9100;
    export t5-export;
}
```

This selective host route redistribution approach offers significant benefits for EVPN Type 5 stitching. It enhances scaling and performance within the data center by reducing unnecessary control plane information and forwarding table entries. Additionally, it conserves resources on data center leaf devices, resulting in a more efficient and streamlined network operation.

Seamless EVPN-VXLAN Stitching: Type-5 Stitching Validation

Now that everything for seamless EVPN-VXLAN Type-5 stitching is configured, we'll want to verify that the necessary routes are being distributed correctly. Let's track the Desktop1 host route (10.99.99.11/32) as it's passed from Building 1 to Building 2. Recall that VLAN v1099 is not present in Building 2. This concept means that we should only see an aggregate route the represents the 10.99.99.0/24 network in Building 2.



Autonomous Systems:
core1: 64512
core2: 64513
dist1: 64514
dist2: 64515
dist3: 64516
dist4: 64517
access1: 64518
access2: 64519
access3: 64520
access4: 64521

Loopback Addresses:
core1: 192.168.101.1
core2: 192.168.101.2
dist1: 192.168.102.1
dist2: 192.168.102.2
dist3: 192.168.102.3
dist4: 192.168.102.4
access1: 192.168.103.1
access2: 192.168.103.2
access3: 192.168.103.3
access4: 192.168.103.4

To begin, let's look for the Desktop1 host route on Access1. Here we can see that there is an EVPN route that uses the irb.1099 interface for the next hop. This output makes sense as Desktop1 is directly connected to Access1.

```
root@access1> show route 10.99.99.11 extensive expanded-nh

T5.inet.0: 14 destinations, 28 routes (14 active, 0 holddown, 0 hidden)

10.99.99.11/32 (1 entry, 1 announced)
        *EVPN   Preference: 7
                Next hop type: Interface, Next hop index: 0
                Address: 0x7cae494
                Next-hop reference count: 2
                Kernel Table Id: 0
                Next hop: via irb.1099, selected << irb.1099 next hop
                State: <Active Int Ext>
                Age: 2:11:28
                Validation State: unverified
                Task: EVPN-VXLAN-1-evpn
                Announcement bits (2): 0-Resolve tree 6 1-T5-EVPN-L3-context
                AS path: I
                Route-nexthop:
                Interface (0x7cae494)
                Thread: junos-main
```

Now, let's look for the Desktop1 host route on Distribution1. Here we can see that the host route is present and the Type-5 tunnel information shows a VTEP source of Distribution1 (192.168.102.1) and a destination of Access1 (192.168.103.1) and a VNI of 50001. With this route we also see the T5 instance route target of target:100:556.

```
root@dist1> show route 10.99.99.11 extensive expanded-nh

T5.inet.0: 11 destinations, 12 routes (11 active, 0 holddown, 0 hidden)

10.99.99.11/32 (1 entry, 1 announced)
Installed-nexthop:
Indr (0x7cadd94) 192.168.103.1
  Krt_inh (0x78ea2f0) Index:1048654 PNH: 192.168.103.1
    Tun-comp (0x7caec94) Index:921 VxLAN src 192.168.102.1 dest 192.168.103.1 encap vni 50001 decap
vni 50001 << Type 5 tunnel
TSI:
KRT in-kernel 10.99.99.11/32 -> {indirect(1048654)}
        *EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7cadd94
                Next-hop reference count: 2
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 617
                Next hop: 10.1.1.0 via ge-0/0/1.0, selected
                Session Id: 140
                Protocol next hop: 192.168.103.1
                Indirect next hop: 0x78ea2f0 1048654 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Active Int Ext VxlanLocalRT>
                Age: 2:13:47    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                Announcement bits (3): 0-Resolve tree 6 1-T5-EVPN-L3-context 2-KRT
                AS path: I
                Communities: target:100:556 << Access and Distribution T5 instance route target
                Route-nexthop:
                Indr (0x7cadd94) 192.168.103.1
                  Krt_inh (0x78ea2f0) Index:1048654
                    Router (0x7cb0814) Index:617 10.1.1.0
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.103.1 ResolvState: Resolved
```

```
                         Indirect next hop: 0x78ea2f0 1048654 INH Session ID: 0
                         Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                         Indirect path forwarding next hops: 1
                                 Next hop type: Router
                                 Next hop: 10.1.1.0 via ge-0/0/1.0
                                 Session Id: 140
                                 192.168.103.1/32 Originating RIB: inet.0
                                   Node path count: 1
                                   Forwarding nexthops: 1
                                         Next hop type: Router
                                         Next hop: 10.1.1.0 via ge-0/0/1.0
                                         Session Id: 140
```

Now, let’s look for the Desktop1 host route on core1. Here we can see that the host route is present and the Type-5 tunnel information shows a VTEP source of core1 (192.168.101.1) and a destination of distribution1 (192.168.101.1) distribution2 (192.168.101.2). The VNI of 50001 is also present. With this route, we also see the distribution interconnect T5 route target of target:300:1.

```
root@core1> show route 10.99.99.11 extensive expanded-nh

T5.inet.0: 12 destinations, 26 routes (12 active, 0 holddown, 0 hidden)

10.99.99.11/32 (3 entries, 2 announced) << Desktop1 host route
        State: <CalcForwarding>
Installed-nexthop:
List (0x7dd3df4) Index:1048603
  Indr (0x7cadd94) 192.168.102.1
    Krt_inh (0x78df868) Index:1048602 PNH: 192.168.102.1
      Tun-comp (0x7cada94) Index:721 VxLAN src 192.168.101.1 dest 192.168.102.1 encap vni 50001 decap
vni 50001 << Type 5 tunnel
  Indr (0x7ca6914) 192.168.102.2
    Krt_inh (0x78deed8) Index:1048599 PNH: 192.168.102.2
      Tun-comp (0x7cadf14) Index:719 VxLAN src 192.168.101.1 dest 192.168.102.2 encap vni 50001 decap
vni 50001 << Type 5 tunnel
TSI:
KRT in-kernel 10.99.99.11/32 -> {list:indirect(1048602), indirect(1048599)}
Aggregated into 10.99.99.0/24
        @EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7cadd94
                Next-hop reference count: 9
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 602
                Next hop: 10.1.3.0 via ge-0/0/1.0, selected
                Session Id: 140
                Protocol next hop: 192.168.102.1
                Indirect next hop: 0x78df868 1048602 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Active Int Ext VxlanLocalRT>
                Age: 2:29:01    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                Announcement bits (3): 0-Resolve tree 2 1-T5-EVPN-L3-context 3-Aggregate
                AS path: 65001 I
                Communities: target:300:1 << Distribution interconnect T5 route target
                Route-nexthop:
                Indr (0x7cadd94) 192.168.102.1
                  Krt_inh (0x78df868) Index:1048602
                    Router (0x7cad594) Index:602 10.1.3.0
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.102.1 ResolvState: Resolved
                        Indirect next hop: 0x78df868 1048602 INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 1
                                Next hop type: Router
                                Next hop: 10.1.3.0 via ge-0/0/1.0
                                Session Id: 140
                                192.168.102.1/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 1
                                        Next hop type: Router
                                        Next hop: 10.1.3.0 via ge-0/0/1.0
                                        Session Id: 140
          EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7ca6914
                Next-hop reference count: 9
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 603
                Next hop: 10.1.3.4 via ge-0/0/2.0, selected
                Session Id: 141
                Protocol next hop: 192.168.102.2
                Indirect next hop: 0x78deed8 1048599 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Int Ext VxlanLocalRT>
                Inactive reason: Nexthop address
                Age: 2:29:12    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                AS path: 65001 I
                Communities: target:300:1
                Route-nexthop:
                Indr (0x7ca6914) 192.168.102.2
                  Krt_inh (0x78deed8) Index:1048599
                    Router (0x7caeb14) Index:603 10.1.3.4
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.102.2 ResolvState: Resolved
                        Indirect next hop: 0x78deed8 1048599 INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 1
                                Next hop type: Router
                                Next hop: 10.1.3.4 via ge-0/0/2.0
                                Session Id: 141
                                192.168.102.2/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 1
                                        Next hop type: Router
                                        Next hop: 10.1.3.4 via ge-0/0/2.0
                                        Session Id: 141
         #Multipath Preference: 255
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7dd3d54
                Next-hop reference count: 2
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 602
                Next hop: 10.1.3.0 via ge-0/0/1.0, selected
                Session Id: 140
                Next hop type: Router, Next hop index: 603
                Next hop: 10.1.3.4 via ge-0/0/2.0
                Session Id: 141
                Protocol next hop: 192.168.102.1
                Indirect next hop: 0x78df868 1048602 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                Protocol next hop: 192.168.102.2
                Indirect next hop: 0x78deed8 1048599 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <ForwardingOnly Int Ext VxlanLocalRT>
                Inactive reason: Forwarding use only
                Age: 2:29:01    Metric2: 0
                Validation State: unverified
                Task: RT
```

```
                 Announcement bits (1): 2-KRT
                 AS path: 65001 I
                 Communities: target:300:1
                 Route-nexthop:
                 Indr (0x7dd3d54) 192.168.102.1
                   Krt_inh (0x78df868) Index:1048602
                     Router (0x7cad594) Index:602 10.1.3.0
                   Krt_inh (0x78deed8) Index:1048599
                     Router (0x7caeb14) Index:603 10.1.3.4
                 Thread: junos-main
```

Let’s take a look at the 10.99.99.0/24 aggregate route that we configured earlier on core1. The output shows that the route is active and has three contributing EVPN routes. Recall that we’ll be advertising this aggregate route to Building2.

```
root@core1> show route 10.99.99.0/24 protocol aggregate detail

T5.inet.0: 12 destinations, 26 routes (12 active, 0 holddown, 0 hidden)
10.99.99.0/24 (3 entries, 1 announced)
        *Aggregate Preference: 130
                Next hop type: Reject, Next hop index: 0
                Address: 0x7ca4614
                Next-hop reference count: 4
                Kernel Table Id: 0
                State: <Active Int Ext>
                Age: 2:39:34
                Validation State: unverified
                Task: Aggregate
                Announcement bits (3): 0-Resolve tree 2 1-T5-EVPN-L3-context 2-KRT
                AS path: {65001} I  (LocalAgg)
                Flags:                  Depth: 0        Active
                AS path list:
                AS path: I Refcount: 2
                AS path: 65001 I Refcount: 1
                Contributing Routes (3):
                        10.99.99.5/32 proto EVPN << contributing routes
                        10.99.99.6/32 proto EVPN << contributing routes
                        10.99.99.11/32 proto EVPN << contributing routes
                Thread: junos-main
```

The following output shows how core1 is advertising the EVPN route with the 10.99.99.0/24 route information to Distribution3 (192.168.102.3). In the output, we can see the route label matches VNI 50001 and the distribution interconnect T5 route target of target:300:1.

```
root@core1> show route advertising-protocol bgp 192.168.102.3 match-prefix
5:192.168.101.1:555::0::10.99.99.0::24/248 extensive table T5.evpn.0

T5.evpn.0: 36 destinations, 36 routes (36 active, 0 holddown, 0 hidden)
* 5:192.168.101.1:555::0::10.99.99.0::24/248 (1 entry, 1 announced)
 BGP group overlay-dis-to-core type Internal
     Route Distinguisher: 192.168.101.1:555
     Route Label: 50001
     Overlay gateway address: 0.0.0.0
     Nexthop: Self
     Flags: Nexthop Change
     Localpref: 100
     AS path: [65002] {65001} I  (LocalAgg)
     Communities: target:300:1 encapsulation:vxlan(0x8) router-mac:2c:6b:f5:4a:2b:f0
```

Now, let’s look for the Desktop1 host route on Distribution3. Here we can see that the host route is not present, but instead we have the 10.99.99.0/24 aggregate route. Recall that Building 2 doesn’t have any hosts in the 10.99.99.0/24 network, so there is no need to send all the host routes from that network into Building 2. The Type-5 tunnel information shows a VTEP source of Distribution3

(192.168.102.3) and a destination of core1 (192.168.101.1), and a VNI of 50001. With this route, we also see the interconnect T5 route target of target:200:1.

```
root@dist3> show route 10.99.99.11 extensive expanded-nh

T5.inet.0: 6 destinations, 7 routes (6 active, 0 holddown, 0 hidden)

10.99.99.0/24 (1 entry, 1 announced) << aggregate EVPN route from core
Installed-nexthop:
Indr (0x7cb0e14) 192.168.101.1
  Krt_inh (0x78de3b0) Index:1048653 PNH: 192.168.101.1
    Tun-comp (0x7cb0d14) Index:861 VxLAN src 192.168.102.3 dest 192.168.101.1 encap vni 50001 decap
vni 50001 << Type 5 tunnel
TSI:
KRT in-kernel 10.99.99.0/24 -> {indirect(1048653)}
        *EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7cb0e14
                Next-hop reference count: 3
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 841
                Next hop: 10.1.3.9 via ge-0/0/3.0, selected
                Session Id: 14f
                Protocol next hop: 192.168.101.1
                Indirect next hop: 0x78de3b0 1048653 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Active Int Ext>
                Age: 2:15:32    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                Announcement bits (3): 0-Resolve tree 6 1-T5-EVPN-L3-context 2-KRT
                AS path: {65001} I
                Aggregator: 65002 192.168.101.1
                Communities: target:200:1 << interconnect T5 route target
                Route-nexthop:
                Indr (0x7cb0e14) 192.168.101.1
                  Krt_inh (0x78de3b0) Index:1048653
                    Router (0x7cae994) Index:841 10.1.3.9
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.101.1 ResolvState: Resolved
                        Indirect next hop: 0x78de3b0 1048653 INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 1
                                Next hop type: Router
                                Next hop: 10.1.3.9 via ge-0/0/3.0
                                Session Id: 14f
                                192.168.101.1/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 1
                                        Next hop type: Router
                                        Next hop: 10.1.3.9 via ge-0/0/3.0
                                        Session Id: 14f
```

Lastly, let's look for the Desktop1 host route on Access3. Here we can see that the host route is present and the Type-5 tunnel information shows a VTEP source of Access3 (192.168.103.3) and a destination of distribution2 (192.168.101.3) distribution3 (192.168.101.3). The VNI of 50001 is also present.

```
root@access3> show route 10.99.99.11 extensive expanded-nh

T5.inet.0: 8 destinations, 14 routes (8 active, 0 holddown, 0 hidden)

10.99.99.0/24 (3 entries, 2 announced) << aggregate EVPN route
        State: <CalcForwarding>
Installed-nexthop:
```

```
List (0x7e075b4) Index:1048597
  Indr (0x7cae414) 192.168.102.3
    Krt_inh (0x78e8310) Index:1048593 PNH: 192.168.102.3
      Tun-comp (0x7cae114) Index:811 VxLAN src 192.168.103.3 dest 192.168.102.3 encap vni 50001 decap
vni 50001 << Type 5 tunnel
  Indr (0x7caf394) 192.168.102.4
    Krt_inh (0x78e8970) Index:1048595 PNH: 192.168.102.4
      Tun-comp (0x7caf314) Index:816 VxLAN src 192.168.103.3 dest 192.168.102.4 encap vni 50001 decap
vni 50001 << Type 5 tunnel
TSI:
KRT in-kernel 10.99.99.0/24 -> {list:indirect(1048593), indirect(1048595)}
        @EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7cae414
                Next-hop reference count: 6
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 612
                Next hop: 10.1.2.1 via ge-0/0/1.0, selected
                Session Id: 140
                Protocol next hop: 192.168.102.3
                Indirect next hop: 0x78e8310 1048593 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Active Int Ext>
                Age: 2:15:42    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                Announcement bits (1): 0-Resolve tree 6
                AS path: {65001} I
                Aggregator: 65002 192.168.101.1
                Route-nexthop:
                Indr (0x7cae414) 192.168.102.3
                  Krt_inh (0x78e8310) Index:1048593
                    Router (0x7cae614) Index:612 10.1.2.1
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.102.3 ResolvState: Resolved
                        Indirect next hop: 0x78e8310 1048593 INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 1
                                Next hop type: Router
                                Next hop: 10.1.2.1 via ge-0/0/1.0
                                Session Id: 140
                                192.168.102.3/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 1
                                        Next hop type: Router
                                        Next hop: 10.1.2.1 via ge-0/0/1.0
                                        Session Id: 140
         EVPN   Preference: 170/-101
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7caf394
                Next-hop reference count: 6
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 613
                Next hop: 10.1.2.3 via ge-0/0/2.0, selected
                Session Id: 141
                Protocol next hop: 192.168.102.4
                Indirect next hop: 0x78e8970 1048595 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <Int Ext>
                Inactive reason: Nexthop address
                Age: 2:15:44    Metric2: 0
                Validation State: unverified
                Task: T5-EVPN-L3-context
                AS path: {65001} I
                Aggregator: 65002 192.168.101.1
                Route-nexthop:
```

```
                Indr (0x7caf394) 192.168.102.4
                  Krt_inh (0x78e8970) Index:1048595
                    Router (0x7cadd94) Index:613 10.1.2.3
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 192.168.102.4 ResolvState: Resolved
                        Indirect next hop: 0x78e8970 1048595 INH Session ID: 0
                        Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                        Indirect path forwarding next hops: 1
                                Next hop type: Router
                                Next hop: 10.1.2.3 via ge-0/0/2.0
                                Session Id: 141
                                192.168.102.4/32 Originating RIB: inet.0
                                  Node path count: 1
                                  Forwarding nexthops: 1
                                        Next hop type: Router
                                        Next hop: 10.1.2.3 via ge-0/0/2.0
                                        Session Id: 141
        #Multipath Preference: 255
                Next hop type: Indirect, Next hop index: 0
                Address: 0x7dd1554
                Next-hop reference count: 2
                Kernel Table Id: 0
                Next hop type: Router, Next hop index: 612
                Next hop: 10.1.2.1 via ge-0/0/1.0
                Session Id: 140
                Next hop type: Router, Next hop index: 613
                Next hop: 10.1.2.3 via ge-0/0/2.0, selected
                Session Id: 141
                Protocol next hop: 192.168.102.3
                Indirect next hop: 0x78e8310 1048593 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                Protocol next hop: 192.168.102.4
                Indirect next hop: 0x78e8970 1048595 INH Session ID: 0
                Indirect next hop: INH non-key opaque: 0x0 INH key opaque: 0x0
                State: <ForwardingOnly Int Ext>
                Inactive reason: Forwarding use only
                Age: 2:15:42    Metric2: 0
                Validation State: unverified
                Task: RT
                Announcement bits (1): 2-KRT
                AS path: {65001} I
                Aggregator: 65002 192.168.101.1
                Route-nexthop:
                Indr (0x7dd1554) 192.168.102.3
                  Krt_inh (0x78e8310) Index:1048593
                    Router (0x7cae614) Index:612 10.1.2.1
                  Krt_inh (0x78e8970) Index:1048595
                    Router (0x7cadd94) Index:613 10.1.2.3
                Thread: junos-main
```

CHAPTER 5 EVPN Multicast

Multicast Functionality

In multicast network architectures, several components play pivotal roles. These encompass source devices, which introduce multicast traffic into the network, receiver devices that express interest in this traffic, and intermediary network devices that facilitate communication between the source and the receiver.

At the boundary of the multicast domain, the Internet Group Management Protocol (IGMP) and Multicast Listener Discover (MLD) are deployed. This protocol is instrumental in registering multicast sources and enabling receivers to solicit multicast streams from specific or broad-ranging multicast source points.

Within a LAN segment, a particular device is chosen as the designated router for that broadcast domain. This device's primary functions are to relay signals to the multicast network when a receiver no longer wishes to engage with a multicast stream and to authenticate or catalog interested receivers in its assigned broadcast domain.

The Protocol Independent Multicast (PIM) protocol is tasked with determining the optimal forwarding route across the routed network, achieved through a range of methods.

A common method involves directing all traffic originating from a source to a centralized point called a shared tree. Within this structure, a principal device is chosen to be the recipient of all source traffic. Furthermore, this device becomes the initial destination for all join requests for multicast streams when the particular source address is undisclosed. This central device is known as the rendezvous point (RP), acting as the nexus for all multicast sources and receivers, facilitating any receiver's participation in a multicast tree.

The term shortest-path tree (SPT) refers to an unobstructed routed trajectory from a source to its corresponding receiver. For the inception of an SPT, the receiver must know the exact source IP address, subsequently requesting traffic from that distinct source using an IGMP inquiry. Upon the reception of an IGMP query that specifies both a source and a group (S,G) pairing, a multicast forwarding tree emerges, adhering to the most direct route between the source's designated router and the receiver's equivalent.

The system leverages join and prune messages to meticulously manage multicast streams throughout the network. These messages are pivotal in activating or discontinuing multicast streams.

Using EVPN Multicast

Within a multicast framework, traffic is disseminated within a specified broadcast domain. Typically, this domain is confined to a singular switching realm, ensuring that multicast traffic is directed solely to remote receivers traversing a routed network. As per this architecture, multicast broadcast packets remain localized within the broadcast domain, concluding their journey at the designated router.

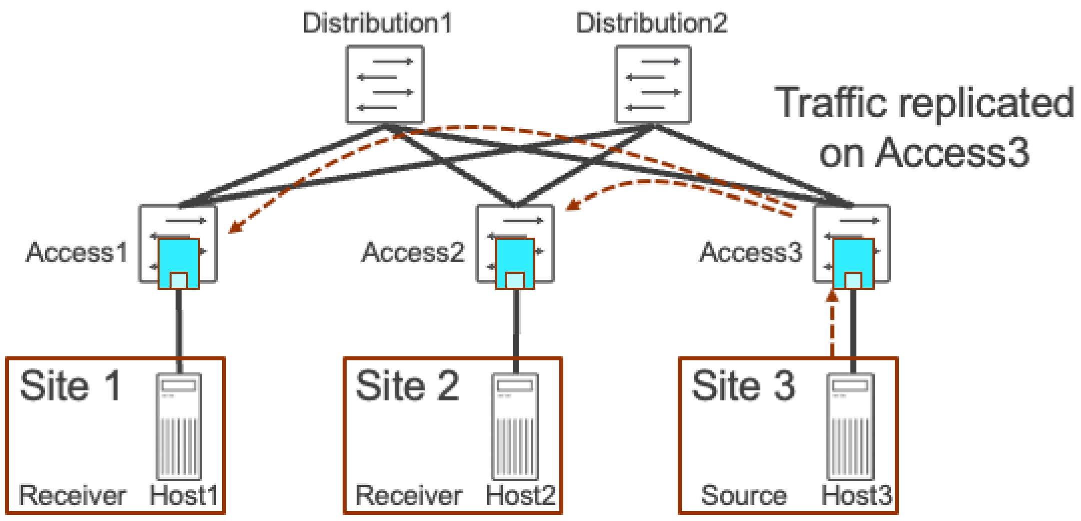
However, when utilizing an EVPN-VXLAN framework, the confines of a broadcast domain extend beyond a singular locale. Here, the EVPN-VXLAN domain can traverse a multitude of access and distribution devices, as well as various campus or data center networks. Consequently, VNIs situated in distant locations become an integral component of a singular broadcast domain. Because of this design, whenever a multicast source transmits traffic, it is forwarded throughout the broadcast domain.

Several methodologies exist to manage multicast traffic within an EVPN-VXLAN. An initial approach is to tunnel IGMP/MLD over the IP overlay. However, this method is less than ideal. It necessitates the establishment of a designated router at distant sites that lack a direct connection to the source or receiver. Such a setup results in suboptimal multicast stream replication while simultaneously segregating multicast operations from the EVPN control plane.

In response to these inefficiencies, three innovative EVPN route types dedicated to multicast were developed:

- The Type 6 route, termed the Selective Multicast Ethernet Tag Route (SMET), is deployed to relay IGMP join signals to distant VTEPs.
- The Type 7 route, classified as the IGMP join sync route, is designed for multihomed environments. In such settings, where a source or receiver is interconnected with a broadcast domain equipped with numerous routers functioning as exit nodes, it's imperative that join requests maintain synchronization across all potential periphery devices.
- Lastly, the Type 8 route, recognized as the IGMP leave sync route, is tailored for multihomed domains. In these scenarios, sources or receivers linked to a broadcast domain with an array of routers serving as egress points necessitate the synchronization of leave requests across all potential boundary apparatuses.

The following diagram shows how multicast traffic can be destined to multiple hosts. Traffic feeds must be replicated at points along the multicast tree where the forwarding path must divide. With EVPN multicast, multicast replication occurs on the source VTEP. The diagram shows that the source of Host3 is transmitting multicast traffic to Access3. It is the job of Access3 to replicate the traffic for every leaf device belonging to that bridge domain (BD). When you have only three access devices, it is not a big deal. But, if you have 200 access devices, suddenly, the workload of the originating access device and the bandwidth necessary becomes a problem.



Consider the following scenario. You have a campus fabric with 200 access devices and the necessary number of distribution devices to support the fabric. There are 40 multicast groups, and each group has around a constant 5 Mbps. Even though there are 200 access devices, only 20 access devices have interested multicast receivers attached to them. With that in mind, we'll use the following information to keep track of what's going on.

- Number of access devices: N = 200
- Number of multicast groups: M(G) = 40
- Traffic rate: R = 5 Mbps
- Number of access devices interested in the traffic: T = 20

The non-optimized multicast scenario that we described in the previous diagram is as follows:

- Distribution bandwidth consumption: N*MG*R (200*40*5) = 40 Gbps
- Replication load on the access devices: N*MG (200*40) = 8000 times
- Link bandwidth consumption between access and distribution: N*MG*R (200*40*5) = 40 Gbps

In this non-optimized multicast scenario, we have our 200 access devices times the 40 groups times the traffic rate of 5 Mbps, which gives us a distribution bandwidth consumption of 40 Gbps. The load on the source access device is the number of access devices multiplied by the number of groups, which works out to be 8000 times for replication. For the link bandwidth consumption between the access and the distribution layers, we have 200 access devices with 40 groups with a traffic rate of 5 Mbps, which results in 40 Gbps needed between access devices.

If we optimize this multicast scenario using Assisted Replication (AR) then things look a bit better:

- Distribution bandwidth consumption: N*MG*R (200*40*5) = 40 Gbps
- Replication load on the access devices: 1*MG (1*40) = 20 times
- Link bandwidth consumption between access and distribution: 1*MG*R (1*40*5) = 200 Mbps

If we break this scenario down, the distribution bandwidth consumption remains the same—200 access devices with 40 groups with 5 Mbps results in a total of 40 Gbps bandwidth consumption. However, the replication load on the source access device has dropped dramatically. Now the source access device only has to handle one replication per group, which results in 40 replications. The access and distribution layer bandwidth consumption is one replication per group at the rate of 5 Mbps or 200 Mbps

If we use SMET without Assisted Replication (AR) we get the following results:

- Distribution bandwidth consumption: T*MG*R (20*40*5) = 4 Gbps
- Replication load on the access devices: T*MG (20*40) = 800 times
- Link bandwidth consumption between access and distribution: T*MG*R (20*40*5) = 400 Mbps

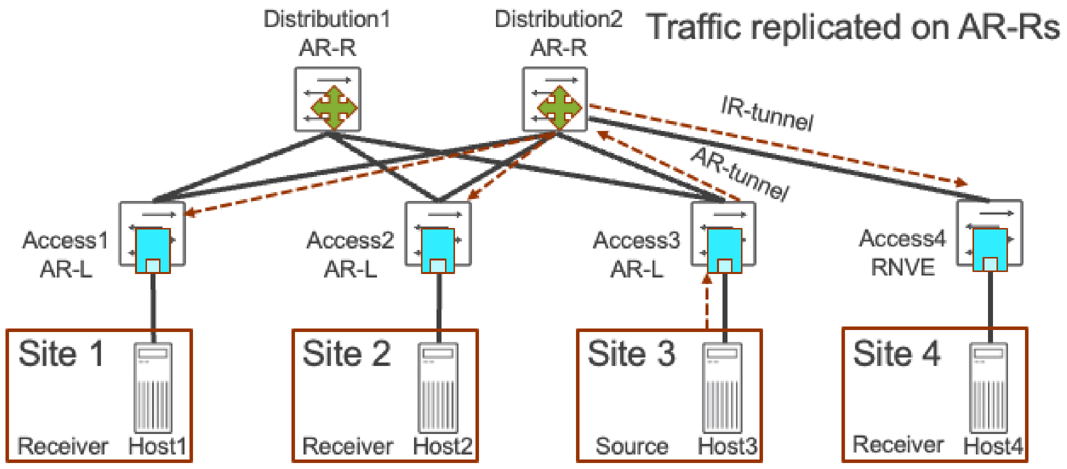
If we break this scenario down, we see that the distribution bandwidth consumption is 20 access devices multiplied by 40 groups multiplied by the rate of 5 Mbps, which results in 4 Gbps. The replication load on the source access device is now 20 multiplied by 40 groups for a total of 800 replications. The link bandwidth between the access and the distribution is 20 access devices multiplied by 40 groups multiplied by 5 Mbps, which results in 400 Mbps.

Now look at what happens when we combine SMET and AR.

- Distribution bandwidth consumption: T*MG*R (20*40*5) = 4 Gbps
- Replication load on the access devices: 1*MG (1*40) = 40 times
- Link bandwidth consumption between access and distribution: 1*MG*R (1*40*5) = 200 Mbps

If we break this scenario down, we see that the distribution bandwidth consumption stays the same as the previous example, as it's still at 4 Gbps. However, the replication load on the access devices is now one replication per group, or 40 replications. The link bandwidth consumption between the access and distribution is now one replication for 40 groups at a traffic rate of 5 Mbps, which results in 200 Mbps.

There are three components to assisted replication. There is the leaf, or access device, which is an AR-leaf (AR-L). The replicators are the AR-replicators (AR-Rs) and the regular network virtualization equipment (RNVE). The AR-L and RNVE devices are ToR devices, while the distribution devices act as the AR-Rs. Note that an RNVE is an access device that is not capable of working in an assisted replication environment.



EVPN-VXLAN sFlow Egress Multicast

The sFlow feature (RFC 3176) provides a robust and efficient network monitoring solution optimized for high-speed switched and routed networks. It is designed to deliver scalable visibility into network traffic patterns without performance degradation on network devices.

Core sFlow Concepts:

- Flow Agent: Embedded within network devices (switches, routers), the sFlow agent continuously samples network traffic and counters, generating sFlow datagrams.
- sFlow Collector: A centralized system responsible for receiving, aggregating, and analyzing sFlow datagrams to provide actionable insights into network behavior.
- Sampling Rate: Controls the frequency at which network packets are captured. Higher rates offer granular visibility but increase datagram volume.
- Polling Interval: Determines the frequency at which interface statistics and counters are sampled.

sFlow Sampling Mechanisms:

- Packet-Based Sampling:
  - Captures a predetermined proportion of packets (e.g., 1 out of 5000) traversing network interfaces.
  - Includes the first 128 bytes, encompassing Layer 2, 3, and 4 headers, plus application-layer headers when applicable.

    - Prioritizes depth over capturing every single flow, ensuring statistically representative traffic analysis for most network use cases.
- Time-Based Sampling:
    - Samples interface counters (e.g., bytes transmitted/received, errors) at regular intervals.
    - Essential for tracking network health metrics, performance indicators, and potential bottlenecks.

Leveraging sFlow for EVPN-VXLAN Multicast Traffic Monitoring:

QFX Series switches empower you to utilize sFlow technology for granular monitoring of known multicast traffic traversing EVPN-VXLAN overlays. This functionality is specifically designed to sample traffic ingressing via core-facing interfaces on an EVPN-VXLAN fabric and egressing through customer-facing ports. Importantly, note that sFlow sampling of multicast in this context is exclusively supported in the egress direction.

Configuration Considerations:

Interface Enablement: As with standard unicast traffic, enable sFlow on the relevant customer-facing egress interfaces.

Global Egress Multicast Sampling: Critically, you must explicitly enable egress multicast sampling by including the `egress-multicast enable` configuration option within the `[edit forwarding options sflow]` hierarchy level.

Replication Rate Control: Optionally fine-tune the replication rate of multicast traffic samples using the `egress-multicast max-replication-rate rate` option under the `[edit forwarding options sflow eggress-multicast]` hierarchy level.

Sampling Behavior & Rate Dynamics:

Interface Group Sampling:** When multiple egress interfaces with sFlow sampling are subscribed to the same multicast group and egress multicast sampling is enabled globally, a uniform sampling rate is enforced. This rate is determined by the most aggressive (lowest) sampling rate configured among the interfaces within the group.

Multi-Group Dynamics: A single interface belonging to multiple multicast groups will exhibit variable sampling rates. The rate applied for a specific group is guided by the most aggressive sampling rate configured among all interfaces participating in that group.

EVPN-VXLAN Architectural Support:

sFlow monitoring within EVPN-VXLAN environments is compatible with both Centrally-Routed Bridging (CRB) and Edge-Routed Bridging (ERB) architectures.

Best Practices and Caveats:

Balancing Granularity and Load: Carefully calibrate sampling and replication rates to achieve the desired level of visibility while minimizing the impact on switch resources and your sFlow collector.

Collector Considerations: As always, ensure your sFlow collector possesses sufficient capacity to ingest and process the anticipated volume of multicast sFlow datagrams.

EVPN OISM in the Campus Network

Optimized Inter-subnet Multicast (OISM) is a feature that enhances multicast traffic. It functions at both Layer 2 and Layer 3 in EVPN-VXLAN edge-routed bridging (ERB) overlay networks. While OISM can be applied to multicast traffic, it isn't suitable for broadcast or unknown unicast traffic.

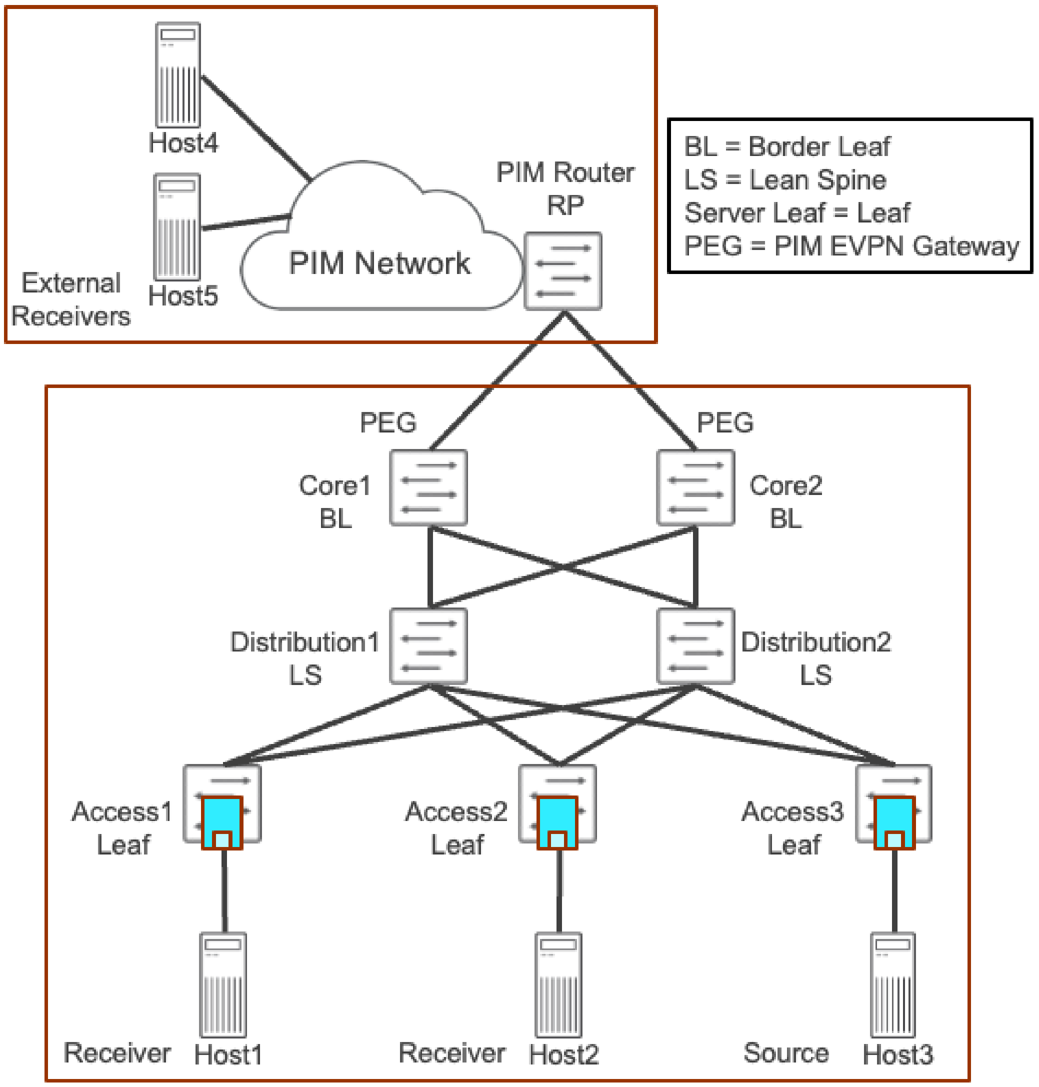
Within EVPN ERB overlay setups, leaf devices route traffic between BDs of tenants, meaning between VLANs. Activating OISM ensures that these leaf devices handle inter-subnet multicast traffic locally through IRB interfaces, relying on control plane multicast states. This local routing approach ensures that the receiver IRB interface doesn't send the routed multicast traffic into the EVPN core, thus reducing the traffic volume inside the EVPN core and eliminating traffic hairpinning.

Furthermore, OISM-enabled leaf devices strategically direct traffic into the EVPN core, targeting only other EVPN devices with relevant receivers. This selective forwarding further refines multicast traffic efficiency in the EVPN setup.

By enabling OISM, ERB overlay networks can adeptly manage multicast traffic between devices within and outside the EVPN structure. In contrast, without OISM, network architects would need to adopt the centrally routed bridging (CRB) overlay approach to accommodate multicast with external sources or recipients. OISM border leaf devices come with varied strategies to channel traffic to and from an external PIM realm. They either utilize integrated routing and bridging (IRB) interfaces or Layer 3 interfaces. Moreover, these border leaf units integrate a supplemental bridge domain (SBD) within the network to transmit traffic from external origins to receivers in the EVPN network.

OISM facilitates EVPN-VXLAN networks using the ERB overlay approach to handle multicast traffic from sources and receivers beyond the fabric boundaries. It also reduces multicast control messages and duplicated data packets within the EVPN fabric core, enhancing multicast efficiency in extensive designs.

The following graphic shows a simple EVPN-VXLAN ERB overlay fabric and the roles of the OISM device in the fabric.



Border leaf (BL):

- OISM leaf devices in the EVPN fabric underlay and overlay. Border leaf devices function as PIM EVPN gateways (PEGs), interconnecting the EVPN fabric to multicast devices (sources and receivers) outside the fabric in an external PIM domain.

Lean spine (LS):

- Spine devices in the underlay of the EVPN fabric. These devices usually operate as lean spines that support the EVPN underlay as IP transit devices. The lean spines might also act as route reflectors in the fabric.
- You configure OISM elements on the lean spine devices only in the following use cases:
  - The devices also serve as border devices for external multicast traffic. In this case, you configure the same OISM elements as you configure on border leaf devices.
  - The lean spine device serves as a standalone AR replicator when you integrate AR with OISM in the fabric. In this case, on the AR replicator spine device, you configure the same common OISM elements that you configure on the border

leaf and server leaf devices. You don't need to configure any of the PIM or external multicast elements specific to border leaf or server leaf devices.

Server leaf (Leaf):

- OISM leaf devices on the access side in the EVPN fabric underlay and overlay. Server leaf devices are often top-of-rack (ToR) switches. These devices connect the EVPN fabric to multicast sources and multicast receiver hosts on BDs or VLANs within the fabric.

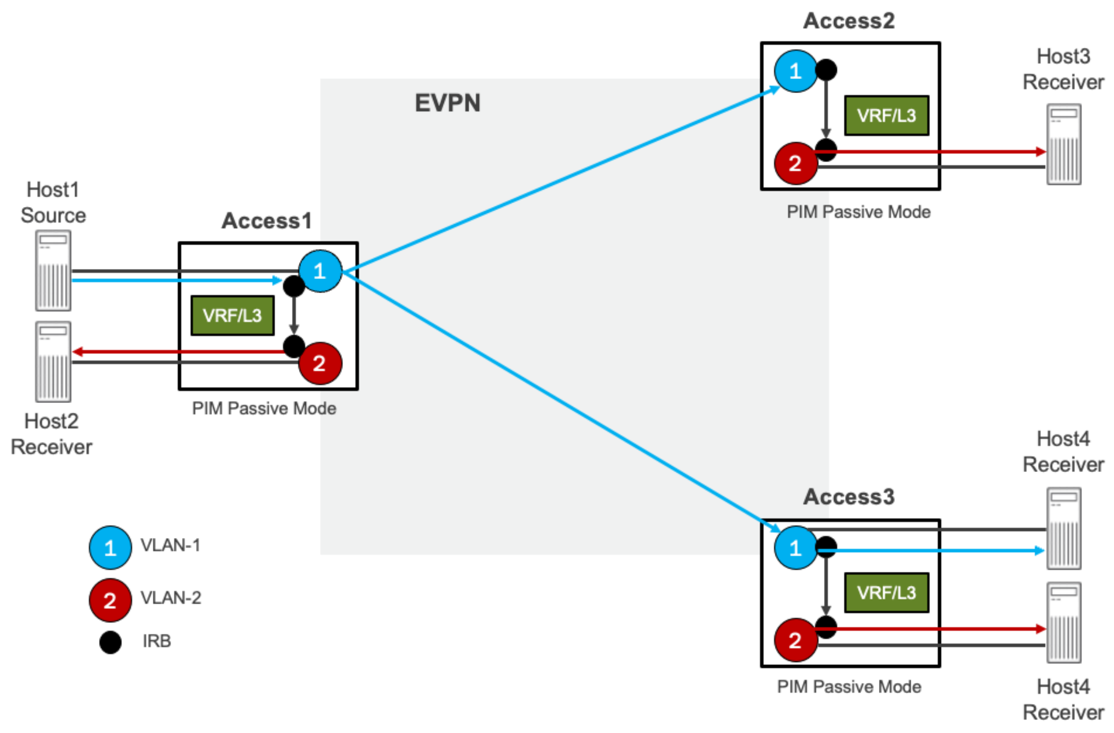
In the previous graphic, the OISM border leaf devices connect to multicast sources and receivers outside the EVPN fabric in a representative external PIM domain. The multicast devices in the external PIM domain follow standard PIM protocol procedures; their operation is not specific to OISM. External multicast traffic flows at Layer 3 through the PIM domain.

You can use OISM to route and to forward multicast traffic in an EVPN-VXLAN ERB overlay fabric between devices in the following use cases:

- Internal multicast sources and receivers
- Internal multicast sources and external multicast receivers
- External multicast sources and internal multicast receivers

How OSIM Works

The following graphic shows how local routing and forwarding works in general on OISM devices. As the figure shows, OISM local routing forwards the traffic on the source VLAN. Each leaf device routes the traffic locally to its receivers on other VLANS, which avoids hairpinning for inter-subnet routing on the same device.



In this case, the source traffic comes from Host1 on VLAN-1, the blue VLAN. Access devices use IRB interfaces and PIM in passive mode to route traffic between VLANs. With PIM in passive mode, leaf devices:

- Don’t become PIM neighbors with the other leaf devices.
- Act as a local PIM RP, create a local PIM state upon receiving IGMP or Multicast Listener Discovery (MLD) reports and avoid doing source registration.

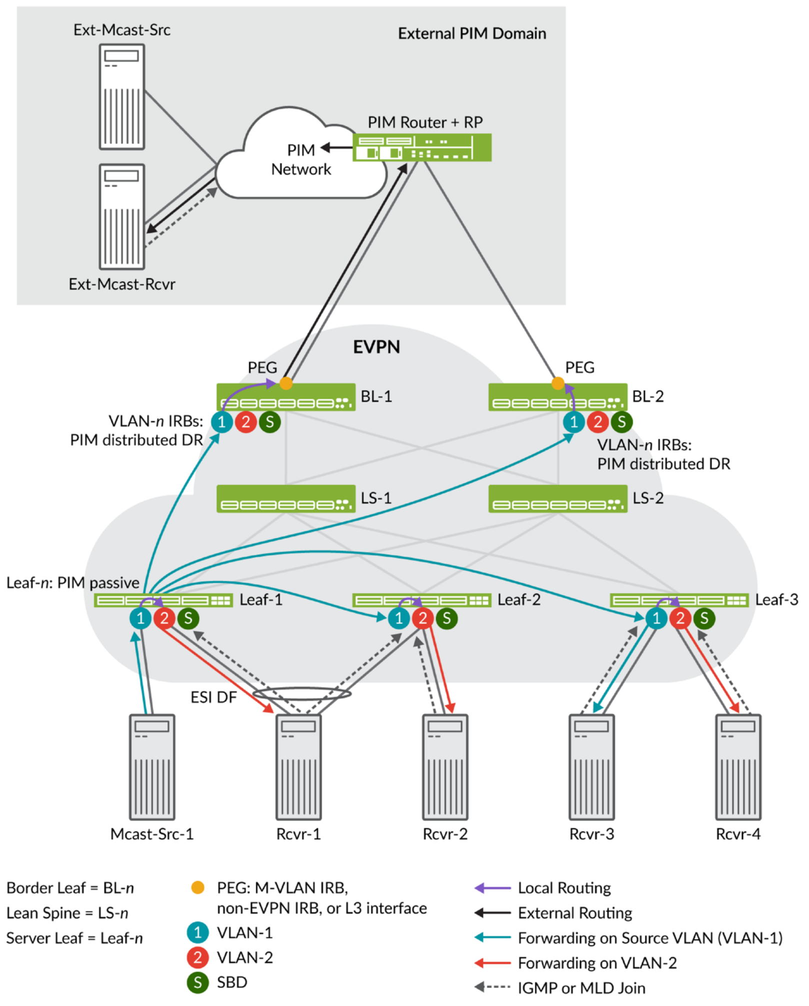
As a result, the server leaf devices forward and route multicast traffic within the fabric as follows to receivers interested in the multicast group:

- The ingress device (Access1) forwards the traffic on the source VLAN into the EVPN fabric toward the other leaf devices with interested receivers.
- All of the access devices (which act as server leafs) don't need to forward the traffic back into the EVPN core to another device that is a designated router. Server leaf devices can locally:
    - Forward the traffic on the source VLAN toward local interested receivers on the source VLAN.
    - Route the traffic from the source VLAN through the IRB interfaces toward local interested receivers in other VLANs.

OISM – Multicast Routing within a Fabric

When the multicast source is inside the EVPN fabric, the server leaf devices receive the multicast traffic on the source VLAN. Then they locally route or forward the traffic as we previously discussed.

The following graphic shows how OISM local routing and forwarding within an EVPN fabric in detail. The graphic also shows how local routing works with EVPN multihoming for a multicast receiver.



The multicast source, Mcast-Src-1, is single-homed to Leaf-1. The source VLAN is VLAN-1 (the blue VLAN). Multicast control and data traffic flow proceeds as follows:

- Receivers on all three server leaf devices sent IGMP or MLD reports (join messages) expressing interest in receiving the traffic for a multicast group.
- Leaf-1 forwards the traffic on the source VLAN to both Leaf-2 and Leaf-3 because both leaf devices have interested receivers. In this case, the receivers on Leaf-2 and Leaf-3 use single-homing.
- Leaf-2 and Leaf-3 forward or locally route the traffic to their interested receivers (Rcvr-2, Rcvr-3, and Rcvr-4).
- Rcvr-1 on VLAN-2 is multihomed to Leaf-1 and Leaf-2 in an EVPN ES. Rcvr-1 has expressed interest in receiving the multicast traffic, so:
    - Both server leaf devices, Leaf-1 and Leaf-2, receive the IGMP or MLD report.
    - Both Leaf-1 and Leaf-2 locally route the traffic from the source VLAN (VLAN-1) because each device has the PIM passive mode configuration.
    - However, because Leaf-1 is the DF for the EVPN ES, only Leaf-1 forwards the traffic to Rcvr-1.
- The border leaf devices receive multicast traffic through the EVPN fabric on the source VLAN. Note that the border leaf devices could have local receivers, although we don't

show that case. With local receivers, the device also locally routes or forwards the traffic to those receivers the same way the server leaf devices do.

- The graphic also shows that the border leaf devices locally route the traffic from the source VLAN toward any external multicast receivers in the external PIM domain.

OISM – Multicast Routing from Outside the Fabric

In the following graphic, we show the OISM use case where a multicast source inside the EVPN fabric sends multicast traffic to a receiver outside the fabric using either of the following methods for external multicast:

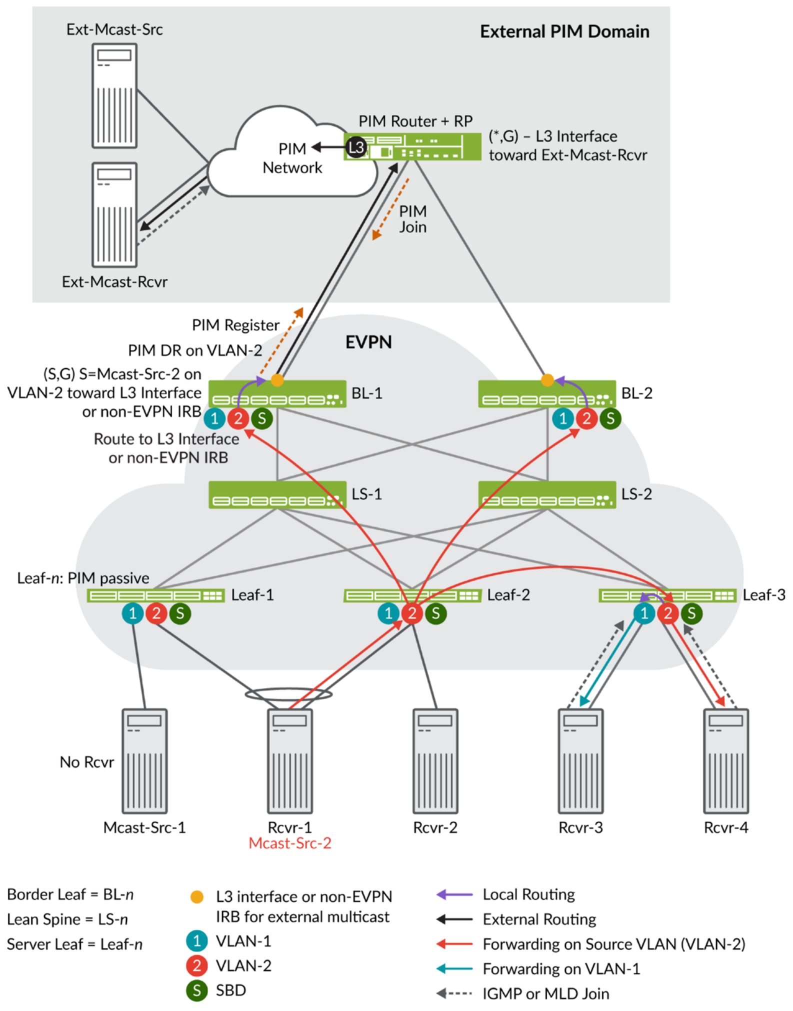
Classic Layer 3 interface external multicast method:

- On each border leaf device, you configure a classic Layer 3 interface with `family inet` that connects to the external PIM router. You assign an IP address to that interface on a different subnet than the Layer 3 interface subnets on other border leaf devices in the fabric.
- You enable PIM on the interface and include the interface in the tenant VRF instances that have multicast data receivers. This method differs from the M-VLAN IRB method because you don't extend this interface in the EVPN instance.

NOTE: The Layer 3 interface connection here can be an individual physical interface or an AE interface bundle that includes multiple physical Layer 3 interfaces.

Non-EVPN IRB external multicast method:

- On each border leaf device, you configure a unique extra VLAN that is only for external multicast. You also configure a corresponding Layer 3 IRB interface with an IP address that connects to the external PIM router. The extra VLAN ID can't be the same as the VLAN ID of the revenue BDs, SBD, or extra VLAN on any other border leaf device in the fabric. In addition, similar to the Layer 3 interface method, the non-EVPN IRB interfaces on different border leaf devices should connect to the PIM router on different subnets in the fabric.
- You enable PIM on the IRB interface and include the interface in the tenant VRF instances that have multicast data receivers. This method differs from the M-VLAN IRB method because you don't extend this VLAN or IRB interface in the EVPN instance.



In the graphic, we see that the internal source for a multicast group is Mcast-Src-2, which is multihomed to Leaf-1 and Leaf-2. The source sends the multicast traffic on VLAN-2, the red VLAN. The external receiver, Ext-Mcast-Rcvr, expresses interest in receiving the multicast traffic for that multicast group (sends a join message).

The border leaf devices in the OISM PEG role receive the multicast traffic on the source VLAN through the EVPN core. However, the main difference in this case is that the external multicast interfaces don't use EVPN signaling and don't share an ESI across the border leaf devices. The external multicast interfaces on each border leaf device are distinct, and each have Layer 3 reachability to the external PIM gateway router. The border leaf device that establishes the PIM join state replicates and sends the traffic on the Layer 3 interface or non-EVPN IRB interface to the external PIM domain with the external receiver.

NOTE: PEG border leafs only send multicast source traffic received on the revenue BDs to the external PIM domain. These devices don't forward the traffic back into the EVPN core toward the other border leafs.

EVPN OISM with Bridge Domain Not Everywhere (BDNE)

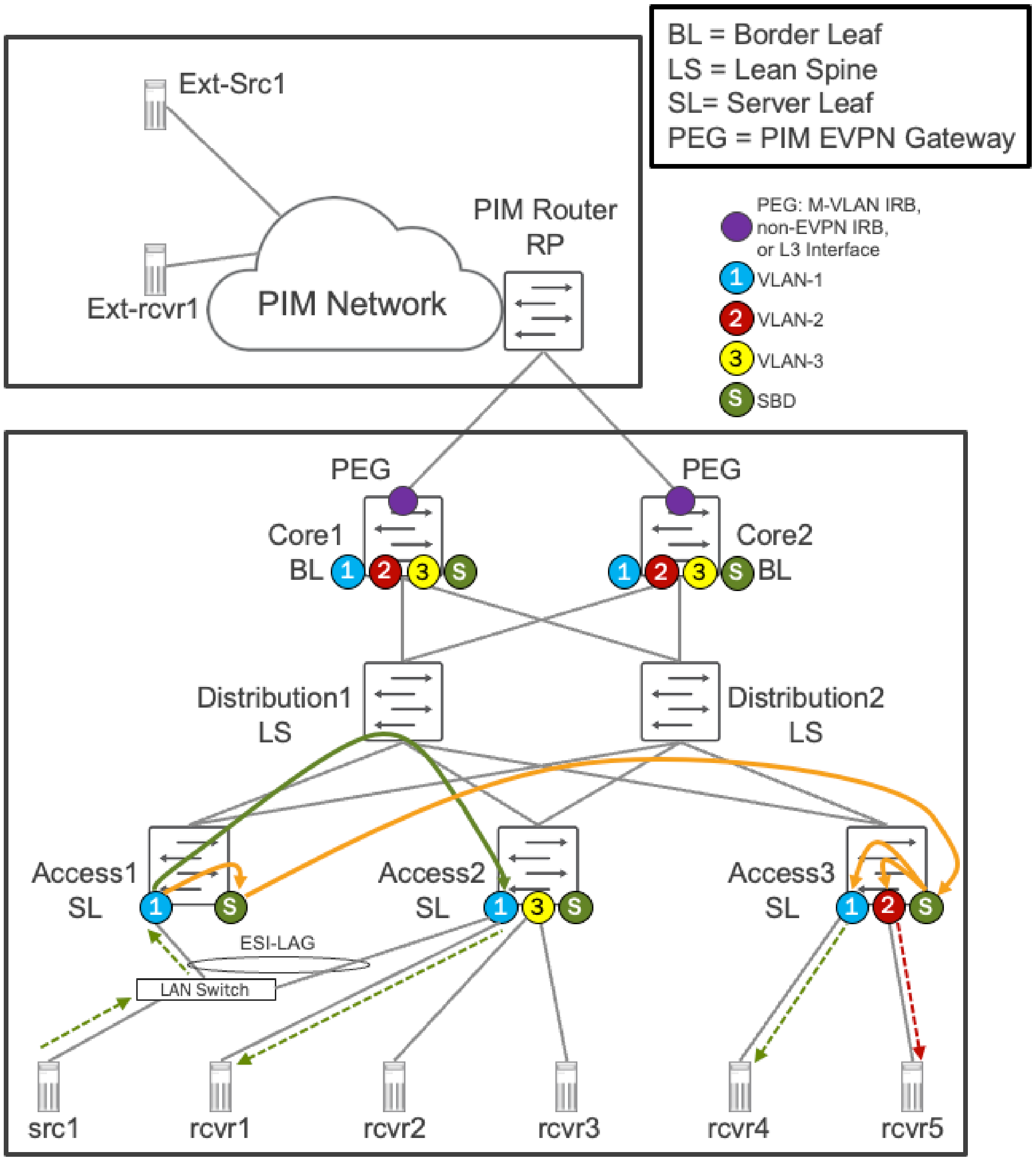
With OISM, multicast traffic is only sent to leafs or border leafs that have BDs with interested multicast receivers. When BDs are everywhere, you have uniformity with every VLAN present on every leaf in the EVPN. In this situation, OISM ensures efficient delivery of inter-subnet multicast traffic. Because each leaf knows about every BD, we do not need to do anything special as all BDs are known at every leaf. However, this situation does not scale well as the number of BDs increases, and the number of resources needed on the leafs also increases. Here, EVPN OISM BDNE comes to the rescue and addresses this scaling problem; however, OSIM is more difficult in the context of BDNE.

- **Unknown BDs**: If a leaf doesn't have a specific BD configured locally, it won't automatically know how to handle multicast traffic for that BD. We need more sophisticated control plane operations to let the leaf know where to send the multicast traffic or even to send it at all.
- **Type 7 Route Handling**: Type 7 routes (Inclusive Multicast Ethernet Tag routes) in EVPN signals interest in receiving multicast traffic for a particular BD. When not all leafs know about all BDs, handling these routes is more complex.
- **Increased Control Plan Overhead**: You can probably imagine there's more overhead in the control plane in this scenario. Leafs need to constantly update each other about which BDs they have and which multicast groups have interested receivers. This is important as it ensures that multicast traffic for unknown BDs is routed correctly.
- **Operational Complexity**: Troubleshooting and network operations can become more difficult. When all BDs are not everywhere, network engineers need to have a deeper understanding of the EVPN state on each leaf, the presence or absence of specific BDs, and how multicast traffic flows in this context.
- **Suboptimal Traffic Paths**: Multicast traffic can potentially take suboptimal paths through the network. If a leaf doesn't recognize a BD and needs to route multicast traffic for that BD based on control plane information, the chosen forwarding path might not be the most efficient path.

Given that all BDs are not present on each leaf in the EVPN, it's critical for leafs to know where they need to send multicast traffic for BDs that they don’t have locally. We can accomplish this through the EVPN control plan. Leafs share information about which BDs they have and the multicast groups that have interested receivers through enhanced Route to SBA Alone (Enh-RSA). This method allows a leaf to know where to send multicast traffic even if the destination BD isn’t locally configured.

The following graphic shows an EVPN network with three server leafs. The Access1 and Access2 devices are SLs that are multihomed with the multicast source src1. Access1 also only has the VLAN-1 BD and SBD. Recall that in this scenario, we are using EVPN OISM BDNE, so every leaf does not need to have every BD. Access2 has three multicast receivers (rcvr1, rcvr2, rcvr3) and three BDs (VLAN-1, VLAN-3, and SBD). In this scenario, only rcvr1 is interested in the multicast traffic from src1. Access3 has two multicast receivers (rcvr4 and rcvr5) and three BDs (VLAN-1, VLAN-2, and SBD). Here, both multicast receivers are interested in the multicast traffic from src1. The distribution devices act as lean spines, and the core devices, which are border leafs, do not come into play as we are not dealing with external multicast traffic.

As we have interested multicast receivers, multicast traffic from src1 is sent to Access1. Recall that Access1 and Access2 are multihomed; this means that the multicast traffic that is sent to rcvr1 is sent directly from VLAN-1 on Access1 to VLAN-1 on Access2. The SBD is not involved in forwarding this traffic as these two leafs are multihomed. The multicast traffic from src1 is not sent to rcvr2 and rcvr3 in VLAN-3 on Access2 as they are not interested in it. The multicast traffic is also forwarded to Access3 as there are two multicast receivers who are interested: rcvr4 and rcvr5. However, this traffic is first forwarded to VLAN-1 on Access1, then to the SBD on Access1, then to the SBD on Access3, and finally to the destination VLANs; VLAN-1 and VLAN-2 on Access3. From there, the multicast traffic is forwarded to the receivers.



NOTE: Enhanced OSIM is required for EVPN OISM BDNE to function properly.

Enhanced-Route-to-SBD-Alone (Enh-RSA)

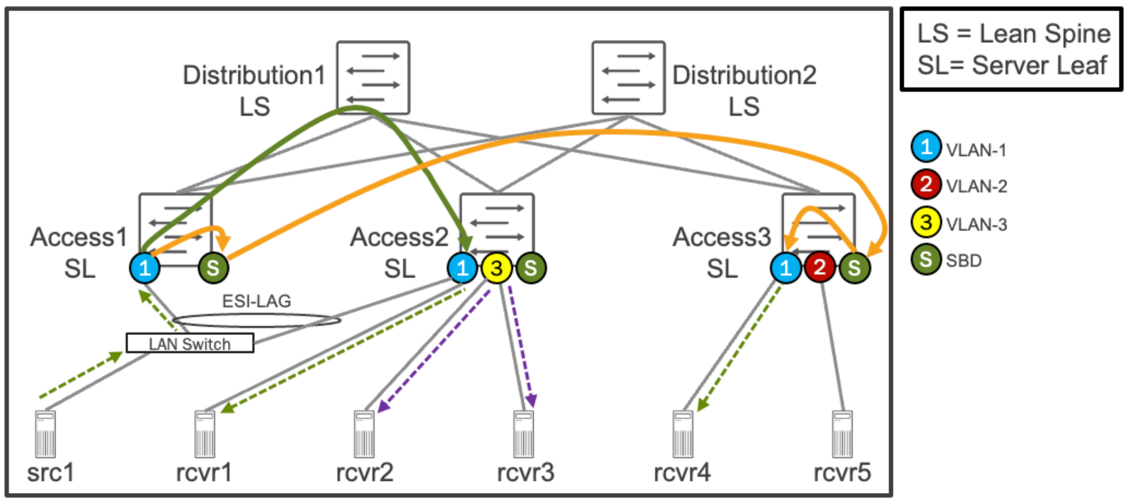
Enhanced-Route-to-SBD-Alone (Enh-RSA) is a traffic forwarding mechanism designed for efficiency and flexibility that is used in network architectures. At its core, Enh-RSA prioritizes direct communication between multihomed devices. If two devices are multihomed, Enh-RSA will establish a direct communication path between them using only the source VLAN.

However, when multihomed devices need to communicate with remote devices that are not multihomed, Enh-RSA leverages the SBD. The SBD acts as a centralized forwarding mechanism, ensuring traffic reaches its destination even if the remote device doesn't share the same source VLAN as the sender. Interestingly, Enh-RSA will still use the SBD even if a remote device happens to host the source VLAN – this maintains consistency in the routing process.

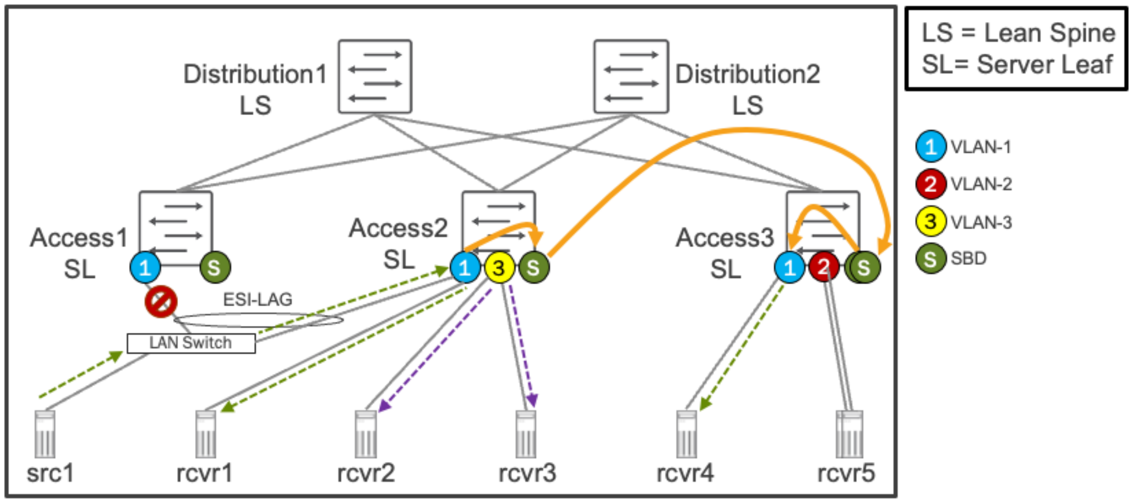
The following graphic shows a standard EVPN-VXLAN environment in which there is one multicast source, src1, and five multicast receivers. The rcvr1, rcvr2, rcvr3, and rcvr4 multicast receivers are interested in the src1 multicast traffic.

Currently, src1 has a default route that sends all traffic to the irb.vlan-1 interface. Note that the SLs Access1, Access2, and Access3 have an irb.vlan-1 interface as they both have VLAN-1 locally. Access2 additionally has the irb.vlan-3 interface as it locally has VLAN-3, and Access3 has the irb.vlan-2 interface as it locally has VLAN-2.

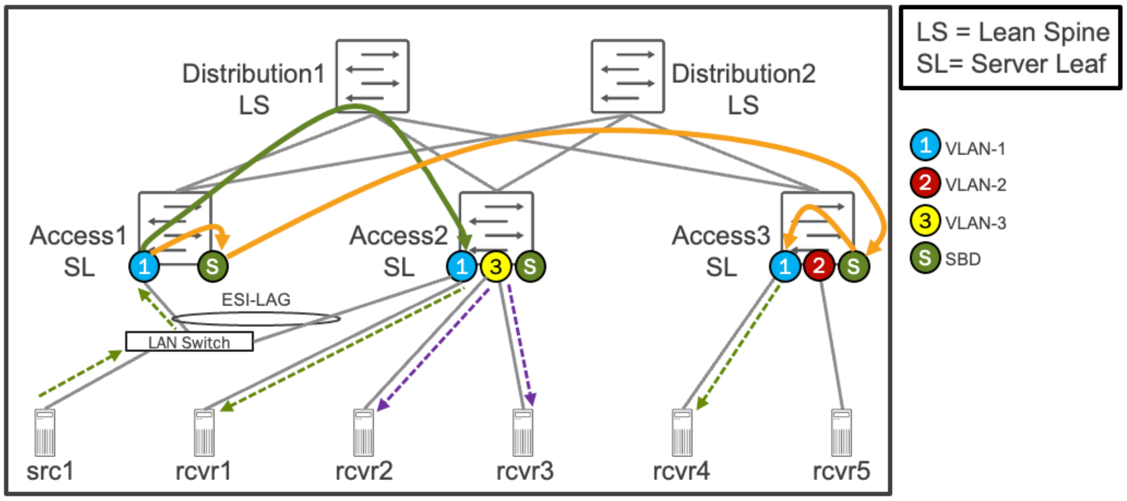
During normal operations, when there is no failure scenario, multicast traffic is sent to Access1 over the ESI-LAG. As there is a receiver, rcvr1, that is a member of VLAN-1, a multicast stream is sent to Access-2 over source BD, VLAN-1. On Access2, the multicast stream is replicated and sent to rcvr1, and the receivers, rcvr2 and rcvr3, in the VLAN-3 BD. It's important to point out here that the multicast traffic is not forwarded to Access2 over the SBD. Access3 also has an interested receiver, rcvr4, which is a member of the VLAN-1 BD. To get the multicast traffic to rcvr4, Access1 forwards the multicast traffic from VLAN-1 to the SBD on Access1. From there, the traffic is forwarded to the SBD on Access3 and locally sent to the VLAN-1 BD. Finally, it is forwarded to rcvr4.



Now, let's look at a scenario where the ESI-LAG member interface that is associated with Access1 goes down and back up, as shown in the following graphics.



When the ESI-LAG member on Access1 goes down, the traffic is sent from src1 directly to the IRB of VLAN-1 on Access2. Recall that Access2 was just previous receiving this traffic over the SBD. Then the traffic is locally forwarded to rcvr1, rcvr2, and rcvr3. The rcvr4 receiver is still interested in the multicast traffic, so it is forwarded from VLAN-1 to the SBD on Access2. Then, Access2 forwards it to the SBD on Access3, when then locally forwards it to VLAN-1.



When the ESI-LAG member interface on Access1 comes back up, traffic flows as it originally did. When traffic flips between the BDs, there will potentially be duplicates or brief packet loss. The reason for this is the flows must be updated with the correct next hop with the change. As you can imagine, this is a problem.

If we are using OSIM-BDNE, which is in turn uses enhanced-RSA to send multicast to a destination BD isn't locally configured, we can statically configure the multihomed peer to solve this problem making them peers for the purposes of enhanced-RSA. Doing this ensures that the traffic between the multihomed peers will always go over the source VLAN, no matter if one of the multihomed peers is up or down. This ensures that there isn't any need for the egress leaf to rely on incoming interface mismatches, thus avoiding the convergence problem.

The following output shows how to configure this feature for IPv4.

```
[edit]
user@Access1# set protocols evpn multihoming-peer-gateways <IPv4-gateway-
peer>
```

The following output shows how to configure this feature for IPv6.

```
[edit]
user@Access1# set protocols evpn multihoming-peer-v6-gateways <IPv6-gateway-
peer>
```

NOTE: The multihoming peer configuration must be completed on both peers.

EVPN-VXLAN sFLOW

The sFlow technology is a powerful monitoring tool designed for high-speed switched and routed networks. It strategically samples network packets and sends them via UDP datagrams to a centralized collector. By enabling sFlow on individual interfaces, you can monitor traffic at wire speed across your entire network. Note that sFlow requires interface-level configuration; there's no global 'enable all' option. Junos OS fully supports the sFlow standard (RFC 3176).

sFlow leverages two distinct sampling mechanisms:

- Packet-based sampling: This method selectively captures one packet out of a specified number on sFlow-enabled interfaces. Only the first 128 bytes of each packet are sent to the collector, including Ethernet, IP, transport layer headers, and potentially application-level headers. While occasional infrequent flows might be missed, this method provides a reliable overall picture of network activity over time. You configure packet-based sampling by setting a sample rate.
- Time-based sampling: This approach is all about statistics. sFlow gathers interface counters (e.g., Ethernet errors) at intervals you define on enabled interfaces. You control time-based sampling by setting a polling interval.

You can use sFlow to monitor known multicast traffic within EVPN-VXLAN environments with the following Juniper switches:

- EX4100
- EX4400
- EX4650
- QFX5110
- QFX5120
- QFX5130
- QFX5700

Specifically, this applies to traffic entering through core-facing interfaces and leaving through customer-facing ports. Remember, multicast sampling is currently only supported in the egress direction. Here's how to enable it:

- Interface Configuration: Enable sFlow on the desired customer-facing interface (egress direction), just as you would for standard unicast traffic.
- Global Configuration: Include the egress-multicast enable option within the `[edit forwarding options sflow]` hierarchy level.
- Optional: Replication Rate Limiting: To control the maximum replication rate for multicast traffic samples, use the **`egress-multicast max-replication-rate <rate>`** parameter under the `[edit forwarding options sflow eggress-multicast]` hierarchy.

It's important to note that when multiple sFlow-enabled interfaces subscribe to the same multicast group (and egress multicast sampling is active), they'll all be sampled at the same rate. This rate is the most aggressive (lowest sampling rate) configured among those interfaces.

If a single port belongs to multiple multicast groups, it can have different sampling rates. The rate for each group is determined by the most aggressive sampling rate among the ports within that group.

EVPN OISM BDNE Configuration and Verification Commands

To configure EVPN OISM BDNE you must use the `enhanced-oism` statement on all server and border leafs, as shown in the following output.

```
[edit]
user@Access1# set forwarding-options multicast-replication evpn irb ?
Possible completions:
  local-only           Multicast forward in local-only mode
  local-remote         Multicast forward in local-remote mode
  oism                 Optimized inter subnet multicast mode
  enhanced-oism        Enhanced OISM with vlans not being everywhere

forwarding-options {
    multicast-replication {
        evpn {
            irb enhanced-oism;
        }
    }
}
```

Once enhanced OISM has been configured, we can look following OISM output to see that enhanced OISM is configured. Display multicast source and group (S,G) information corresponding to Selective P-router Multicast Service Interface (S-PMSI) Auto-Discovery (A-D) routes in an EVPN-VXLAN network running enhanced optimized intersubnet multicast (OISM). Enhanced OISM devices use S-PMSI A-D routes to perform PIM source registration only for multicast sources inside the EVPN-VXLAN network.

With enhanced OISM, also called the asymmetric bridge domains design, the PEG devices receive traffic on the OISM supplemental bridge domain (SBD) from both external and internal multicast sources. Because OISM PEG devices should only perform PIM registration to the PIM rendezvous point (RP) for internal sources, the enhanced OISM design must distinguish between internal and external multicast sources. With enhanced OISM, the ingress leaf devices advertise S-PMSI A-D routes for internal multicast sources. The PEG devices then send a PIM register to the PIM RP only for the sources that correspond to received S-PMSI A-D routes.

```
user@Access1> show evpn oism spmsi-ad
Instance: evpn-vxlan-A
    VN Identifier: 901
        Group IP: 225.1.1.4, Source IP: 60.0.0.103


user@Access1> show evpn oism spmsi-ad extensive
Instance: evpn-vxlan-A
  VN Identifier: 901
    Group        Source        Local    Remote
    225.1.1.4    60.0.0.103     1        0

{master:0}
user@Access1> show evpn oism extensive
EVPN L3 context: DELTA
  OISM SBD interface: irb.901
  OISM Mode: Enhanced OISM

EVPN L3 context: GAMMA
  OISM SBD interface: irb.902
  OISM Mode: Enhanced OISM
```

The options for the `show evpn oism spmsi-ad` command are:

State	Description
Basic brief output (no extensive option)	Display summary output for all Layer 3 (L3) virtual routing and forwarding (VRF) instances and all EVPN instances.
`extensive`	(Optional) Display more detailed output.
`group` ***`multicast-group-address`***	(Optional) Display information for the specified multicast group.
`l3-context` ***`l3-vrf-name`***	(Optional) Display information only for the specified Layer 3 virtual routing and forwarding (VRF) routing instance. Without this option, this command displays information for all Layer 3 VRF routing instances where you enabled OISM.
`group` ***`multicast-group-address`***	(Optional) Display information for the specified multicast group.

The following output shows the prefixes that are associated with the enhanced OSIM from Access1 and Access2.

```
{master:0}
user@Access1> show route table default-switch.evpn.0 match-prefix 10*
extensive

default-switch.evpn.0: 698 destinations, 698 routes (698 active, 0 holddown,
0 hidden)
10:10.255.0.3:100::901::60.0.0.103::225.1.1.4::10.255.0.3/520 (1 entry, 1
announced)
        *EVPN   Preference: 170
                Next hop type: Indirect, Next hop index: 0
                Address: 0x77bd034
                Next-hop reference count: 202, key opaque handle: 0x0, non-
key opaque handle: 0x0
                Protocol next hop: 10.255.0.3
                Indirect next hop: 0x0 - INH Session ID: 0
                State: <Active Int Ext>
                Age: 2:11
                Validation State: unverified
                Task: default-switch-evpn
                Announcement bits (1): 2-rt-export
                AS path: I
                Communities: encapsulation:vxlan(0x8)
                Thread: junos-main

user@Access2> show route table default-switch.evpn.0 match-prefix 10*
extensive

evpn-vxlan-A.switch.evpn.0: 698 destinations, 698 routes (698 active, 0
holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10:10.255.0.3:100::901::60.0.0.103::225.1.1.4::10.255.0.3/520
                   *[BGP/170] 00:02:44, localpref 100, from 10.255.0.3
                      AS path: I, validation-state: unverified
                       to 1.0.4.2 via xe-0/0/0.0
                    >  to 2.0.4.2 via xe-0/0/1.0

default-switch.evpn.0: 698 destinations, 698 routes (698 active, 0 holddown,
0 hidden)
10:10.255.0.3:100::901::60.0.0.103::225.1.1.4::10.255.0.3/520 (1 entry, 1
announced)
        *BGP    Preference: 170/-101
                Route Distinguisher: 10.255.0.3:100
                Next hop type: Indirect, Next hop index: 0
                Address: 0x77c02f4
                Next-hop reference count: 32, key opaque handle: 0x0, non-key
opaque handle: 0x0
                Source: 10.255.0.3
                Protocol next hop: 10.255.0.3
                Indirect next hop: 0x2 no-forward INH Session ID: 0
                State: <Secondary Active Int Ext>
                Local AS:   100 Peer AS:   100
                Age: 2:58       Metric2: 2
                Validation State: unverified
                Task: BGP_100.10.255.0.3
                Announcement bits (1): 0-default-switch-evpn
                AS path: I
                Communities: target:100:100 encapsulation:vxlan(0x8)
                Import Accepted
                Localpref: 100
                Router ID: 10.255.0.3
                Primary Routing Table: bgp.evpn.0
                Thread: junos-main
                Indirect next hops: 1
                        Protocol next hop: 10.255.0.3 Metric: 2 ResolvState:
Resolved
                        Indirect next hop: 0x2 no-forward INH Session ID: 0
                        Indirect path forwarding next hops: 2
                                Next hop type: Router
                                Next hop: 1.0.4.2 via xe-0/0/0.0
                                Session Id: 0
                                Next hop: 2.0.4.2 via xe-0/0/1.0
                                Session Id: 0
                                10.255.0.3/32 Originating RIB: inet.0
                                  Metric: 2 Node path count: 1
                                  Forwarding nexthops: 2
                                        Next hop type: Router
                                        Next hop: 1.0.4.2 via xe-0/0/0.0
                                        Session Id: 0
                                        Next hop: 2.0.4.2 via xe-0/0/1.0
                                        Session Id: 0s
```

EVPN DCI Multicast with OISM BDNE

Data centers are special facilities where computer servers and networks are stored. They can be set up in many places, such as within a company, in the cloud, or even in different parts of the world. Companies can either set up their own data center or use a company that's an expert in managing them.

When a company has more than one data center, more than one campus, or a campus and a data center, that they must link together, the link is known as a Data Center Interconnect (DCI).

A DCI can work in two main ways: Layer 2 or Layer 3. Think of these layers as different methods of communication. A Layer 2 DCI is like a bridge that lets data flow between centers. On the other hand, a Layer 3 DCI works like a postal service, directing data where to go based on its address. There are many ways to set up these connections between data centers.

Big companies often have many data centers in different places. To connect these places, they look at the cost and available connection options. Companies control how data is directed, and sometimes they even control how data is transported between sites. A popular way to connect these centers is over the Internet. When using this, the company providing the service can either be unseen by the end user, like with a Layer 2 VPN, or it can show up as a network that reroutes data.

When deciding how to connect data centers, there are many choices to consider. It's essential to know the strengths and weaknesses of each method, as well as what your applications need.

For Layer 3 DCI only, the data center fabric can be EVPN-VXLAN. This DCI option passes EVPN Type-5 routes across the DCI. With this option, no Layer 2 connectivity is established across the DCI link. Only Layer 3 routing is performed across the DCI link.

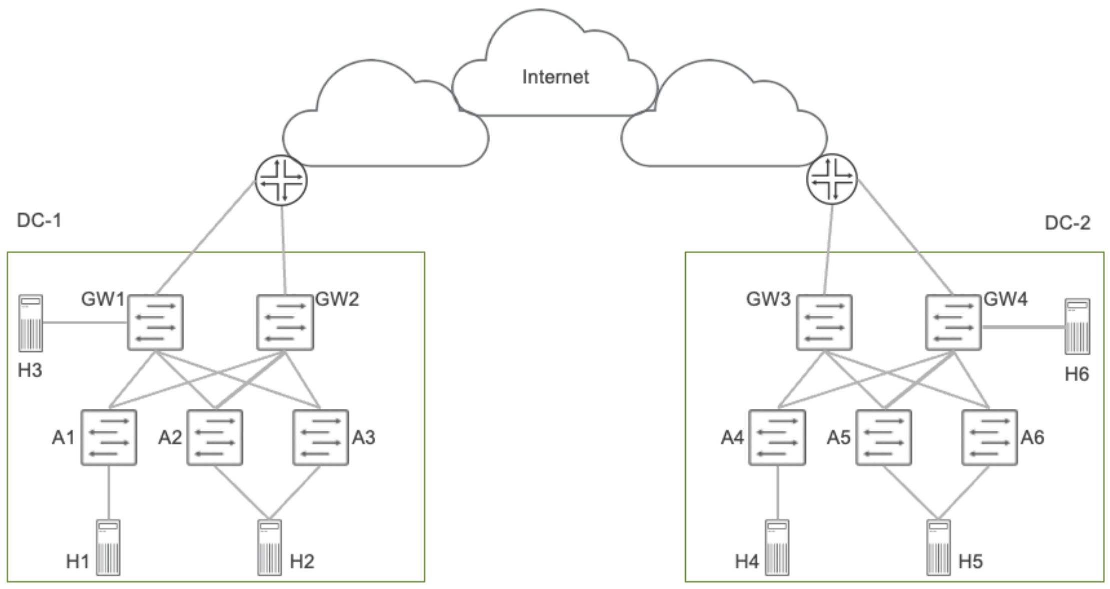
If Layer 2 and Layer 3 DCI are required, a hybrid of EVPN with Type-2 routes and EVPN with Type-5 routes can be used. EVPN Type-2 routes are used for a Layer 2 stretch across the DCI. EVPN Type-5 routes are used when Layer 3 DCI is required.

When using an MPLS backbone, and a data center is running EVPN-VXLAN, an IP VPN/MPLS DCI can be configured.

If Layer 2 and Layer 3 connectivity is desired across an MPLS backbone, a VLAN handoff from the data center fabric to an edge device, which is running VPLS, or IP VPN/MPLS, or EVPN/MPLS can be used.

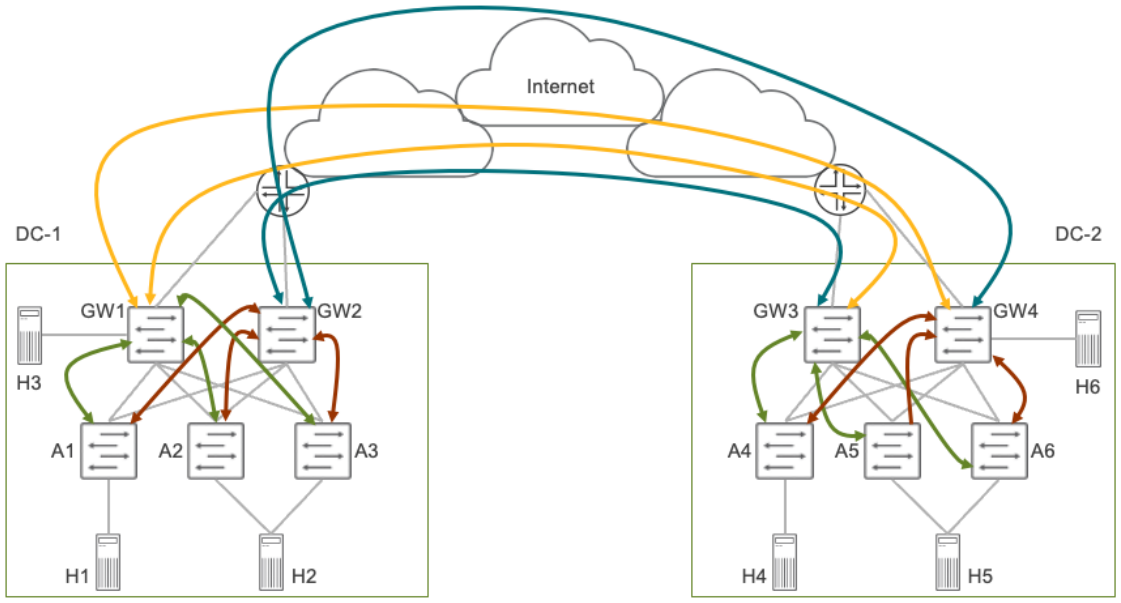
NOTE: Assisted Replication is not currently supported with EVPN DCI multicast.

The following graphic shows two Layer 3 data centers that have similar resources and are connected together over the Internet. To connect those resources, we can use EVPN-VXLAN.



The following graphic shows how VXLAN tunnels are formed between the different network devices in the two data centers. The leafs in a specific data center only form EVPN-VXLAN tunnels with spine devices in the same data center. Then, EVPN-VXLAN tunnels are formed between the spine devices in the different data centers over the Internet.

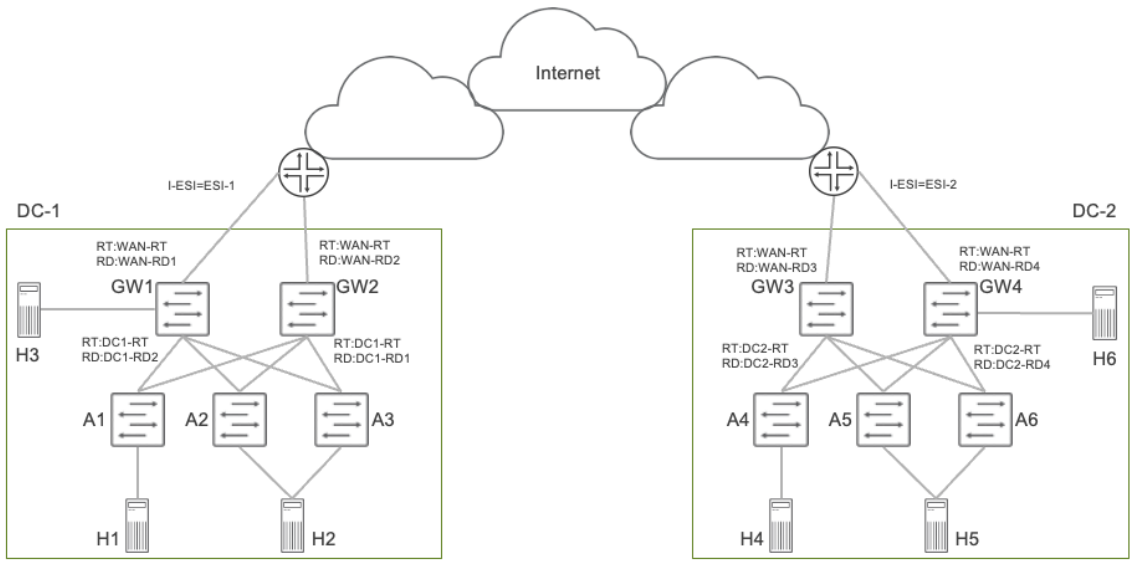
NOTE: The VXLAN tunnels that are shown in the graphic are related to DCI functionality. Additional VXLAN tunnels will form between leafs in a data center to facilitate traffic between hosts inside the same data center. These tunnels are not shown for the sake of brevity.



The following graphic shows how the route targets and route distinguishers work in the VXLAN tunnel scenario described in the previous graphic. The VXLAN tunnels inside each data center here use the same route target and different route distinguishers. For example, you can see in DC-1 that the route target is DC1-RT, and the route distinguishers are DC1-RD1 and DC1-RD2. We see something similar for DC-2.

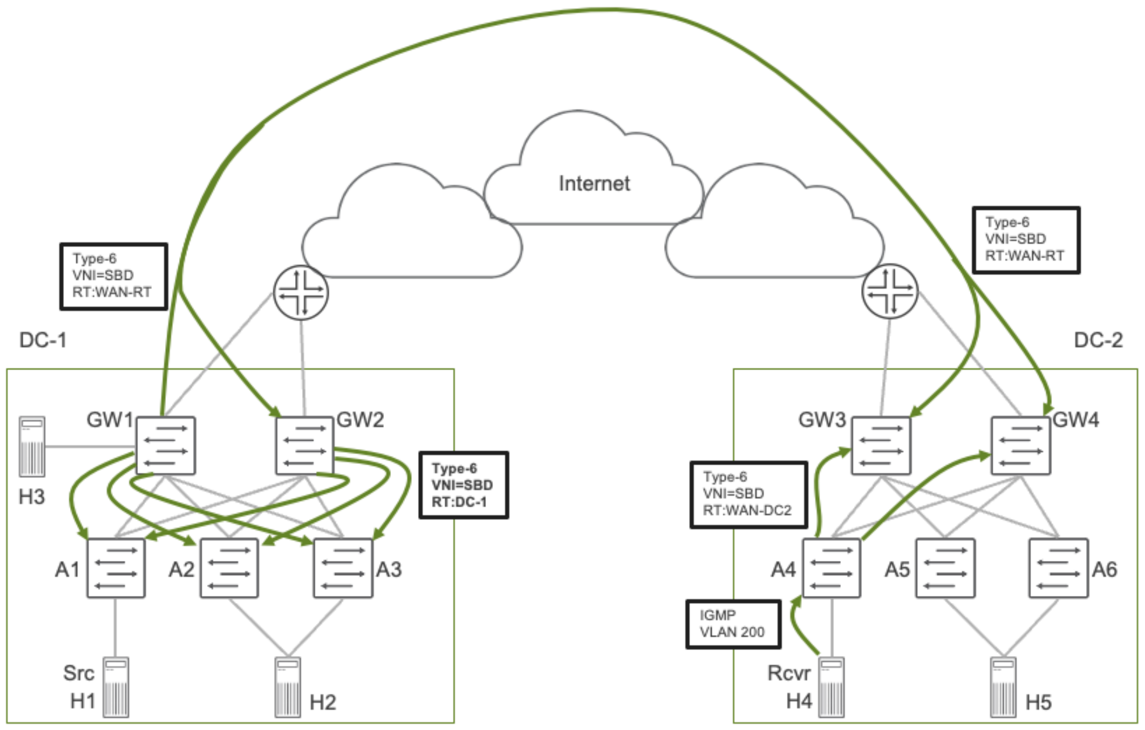
For the VXLAN tunnels of the DCI, the same route target is in use, WAN-RT, for both data centers with different route distinguishers.

NOTE: We are using simplified route targets and route distinguishers for our example here. In a real data center, you would need to use appropriate route targets and route distinguishers.



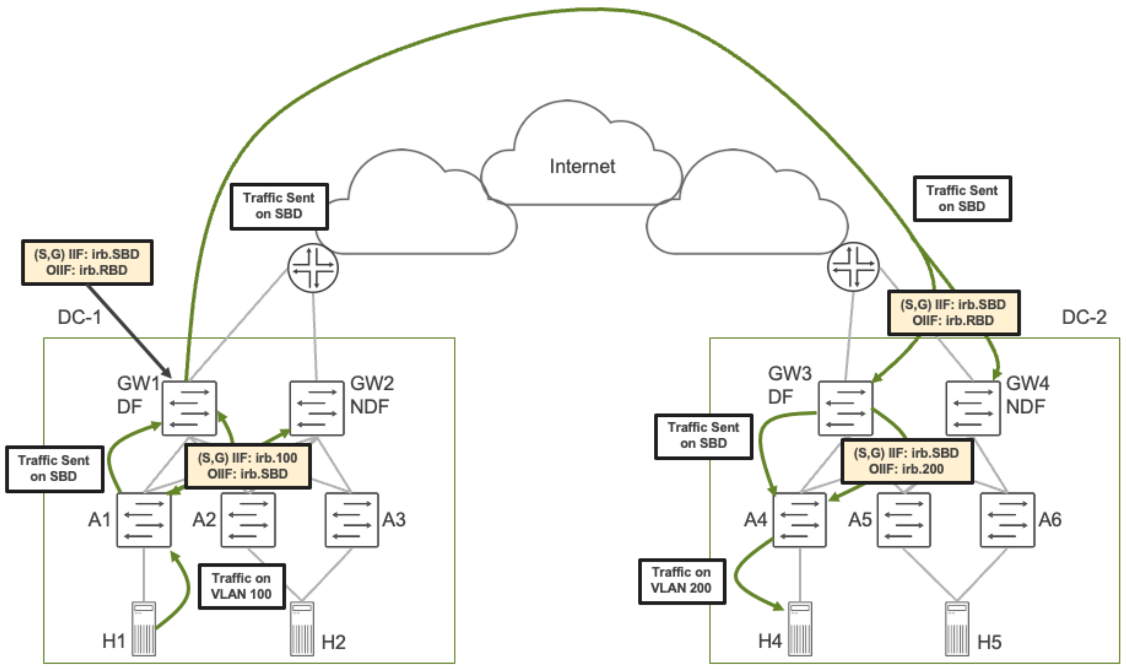
EVPN Control Plane Signaling with DCI Multicast

In the following graphic, host H1 acts as a source in VLAN 100, while host H4 serves as a receiver in VLAN 200. When H4 wants to join a multicast group, it sends an IGMP report to A4. Responding to this, A4 creates a special route, known as an EVPN Type-6 route, and tags it with an SBD VNI and DC2-RT. This route information is then shared with two gateways, GW3 and GW4. On receiving the EVPN Type-6 route, both GW3 and GW4 adjust it by replacing the route target with the WAN-RT route target, and then they send it out towards the WAN. This route information is picked up by two other gateways, GW1 and GW2, which replace the WAN-RT route target with a DC1-RT route target and then forward it toward the leafs in DC1. Finally, A1 in DC1 receives the EVPN Type-6 route with the DC1-RT route target and adds it to the appropriate tables.



EVPN DCI Multicast Packet Forwarding

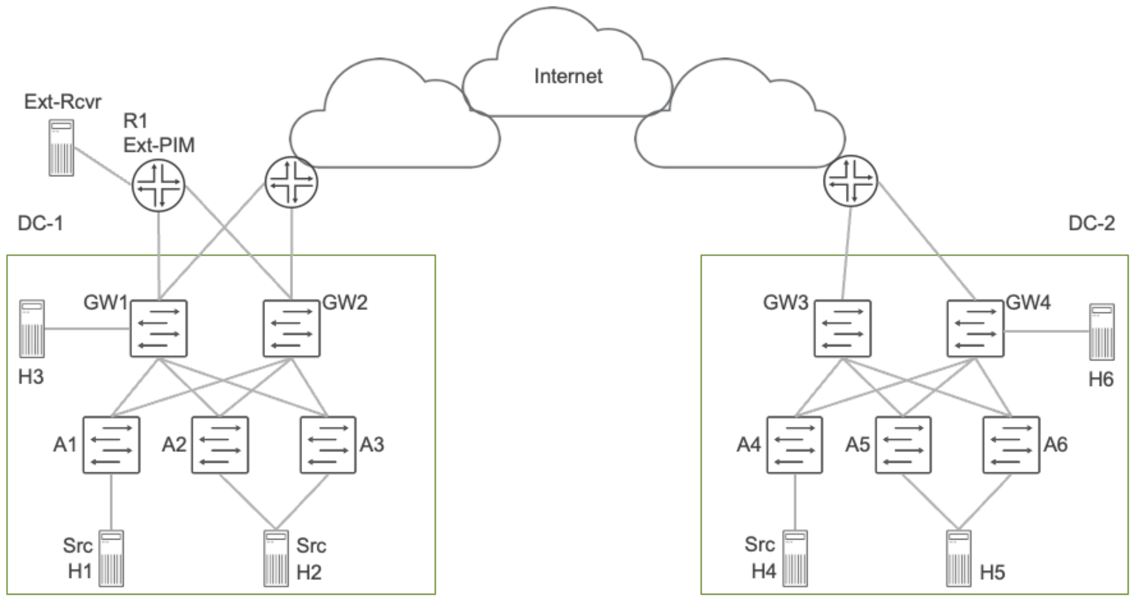
In the following graphic, host H1, which is in VLAN 100, sends multicast traffic, and host H4 from VLAN 200 is set to receive it. When H1 initiates the traffic, it directs the traffic to A1 on VLAN 100. A1, upon receiving the traffic, promptly relays it to two gateways, GW1 and GW2, using the SBD. Following this action, GW1 takes charge, as it is the DF (designated forwarder), and forwards the traffic to two other gateways, GW3 and GW4, still through the SBD. GW3, which is the DF, then forwards the traffic towards A4 through the same SBD. Once A4 receives the traffic on the SBD, it switches it over to VLAN 200 and forwards it to host H4.



EVPN DCI External Multicast Traffic Forwarding Flow

In the following graphic, R1 is an external PIM device that connects to two gateways, GW1 and GW2, through Layer 3 interfaces. These gateways, GW1 and GW2, are recipients of multicast information from an external multicast receiver known as Ext-Rcvr. Among these, GW1 holds a significant role as the PIM Designated Router (DR) on the irb.SBD interface. Because GW1 is elected as the PIM DR, it forwards the multicast information to several devices: A1, A2, A3, as well as the gateways GW3 and GW4, all through the SBD. Taking their tasks further, both GW1 and GW2 incorporate the PEG bit in the Multicast Flags (MF) community within the EVPN Type-3 route and set up an EVPN Type-6 (*,*) multicast route. They then disseminate the EVPN Type-3 route (with the PEG bit set in the MF community) and the EVPN Type-6 (*,*) route, ensuring it spreads within DC-1 and extends over the WAN to DC-2. As a result of this configuration, traffic from any source within DC-1 and DC-2 is directed to GW1 and GW2 using the SBD.

NOTE: GW2 refrains from forwarding the multicast traffic towards the WAN. This is because GW2 is an NDF (Non-Designated Forwarder), meaning it’s not the primary device chosen to forward this specific multicast traffic.



EVPN DCI External Multicast with Enhanced OISM

In the majority of data centers, not all BDs are hosted across every EVPN edge device in all of the data centers. This approach is lends well to using EVPN OISM BDNE. Remember that EVPN OSIM BDNE requires the use of Enhanced OISM. For all of this to work correctly, every EVPN enabled device across all data centers must use the `enhanced-oism` statement configured.

When it comes to EVPN DCI multicast, EVPN Type-6 routes are generated by the DCI Gateway when moving from one data center to the next, but this occurs exclusively on the SBD. This means that the multicast traffic switches between DCI Gateways over the WAN, but only on the SBD.

Using Enhanced OISM over the DCI has the following characteristics:

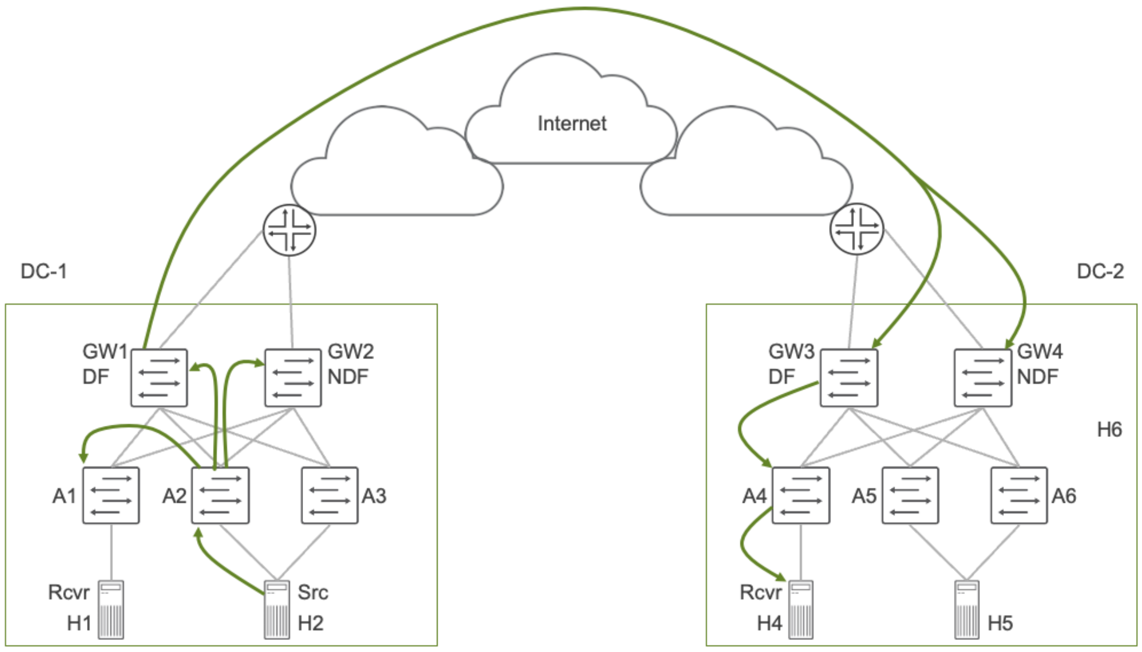
- EVPN Type-6 routes are exclusively announced to remote EVPN edge devices on the SBD.
- Multicast traffic circulates among the multihomed peers, but only over the source-VLAN.
- Furthermore, multicast traffic is dispatched to distant non-multihomed EVPN edge devices through the SBD, regardless of their association with the host source-VLAN.

Given these characteristics, it's evident that Enhanced OISM works perfectly with the demands and functions of EVPN DCI multicast support.

EVPN DCI Multicast Forwarding Scenario #1

In the following graphic, H2 acts as the multicast source, while both H1 and H4 are multicast receivers. When the traffic starts, A2 forwards the multicast traffic from a revenue BD to three devices: A1, GW1, and GW2, all through the SBD. GW1, notable for its role as the DF, then passes the multicast traffic onwards to both GW3 and GW4, still utilizing the SBD. From there, GW3, which is a DF, forwards the multicast traffic to A4, once again through the SBD. Finally, A4 forwards the traffic on a revenue BD to H4.

NOTE: GW2 refrains from forwarding the multicast traffic towards the WAN. This is because GW2 is an NDF (Non-Designated Forwarder), meaning it's not the primary device chosen to forward this specific multicast traffic.

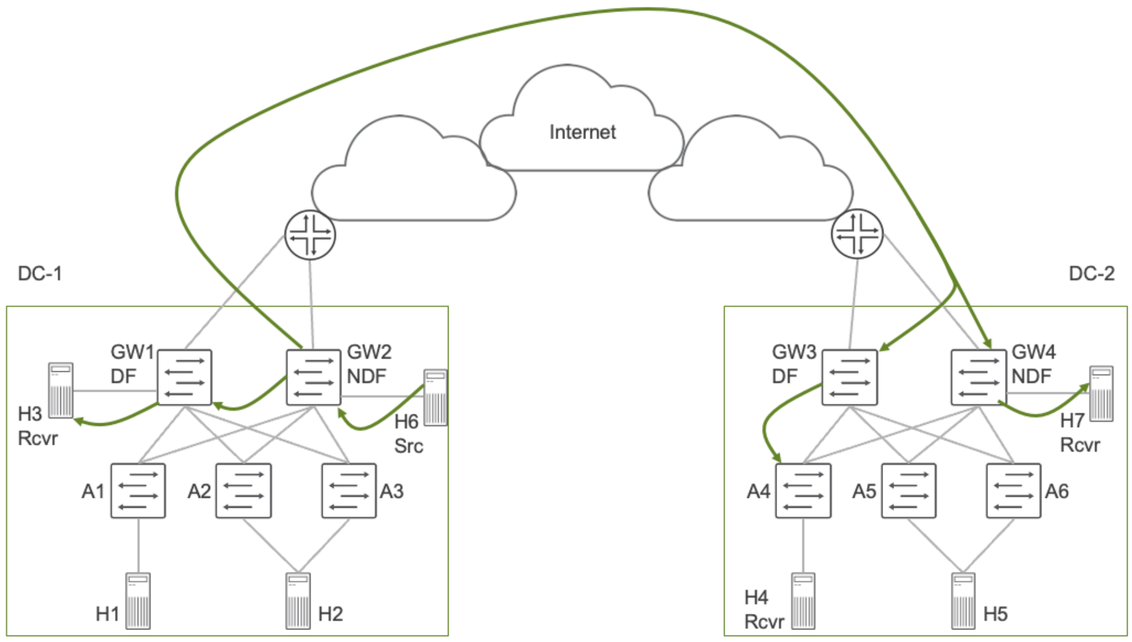


EVPN DCI Multicast Forwarding Scenario #2

In the following graphic, H6 is the multicast source, while H3, H4, and H7 take on the roles of multicast receivers. Even though it might come across as unusual, GW2, which is the NDF, handles the task of forwarding the multicast traffic. This exception is made because H6, the multicast source, is directly connected to GW2. Hence, GW2 forwards the traffic to GW1, GW3, and GW4 over the SBD. When GW1 receives this multicast traffic through the SBD, it promptly sends it to H3 using a revenue BD.

Moving on, GW3 forwards this multicast traffic to A4, again through the SBD. Similarly, GW4 receives this multicast traffic from GW2 through the SBD and is responsible for forwarding it to H7 on a revenue BD.

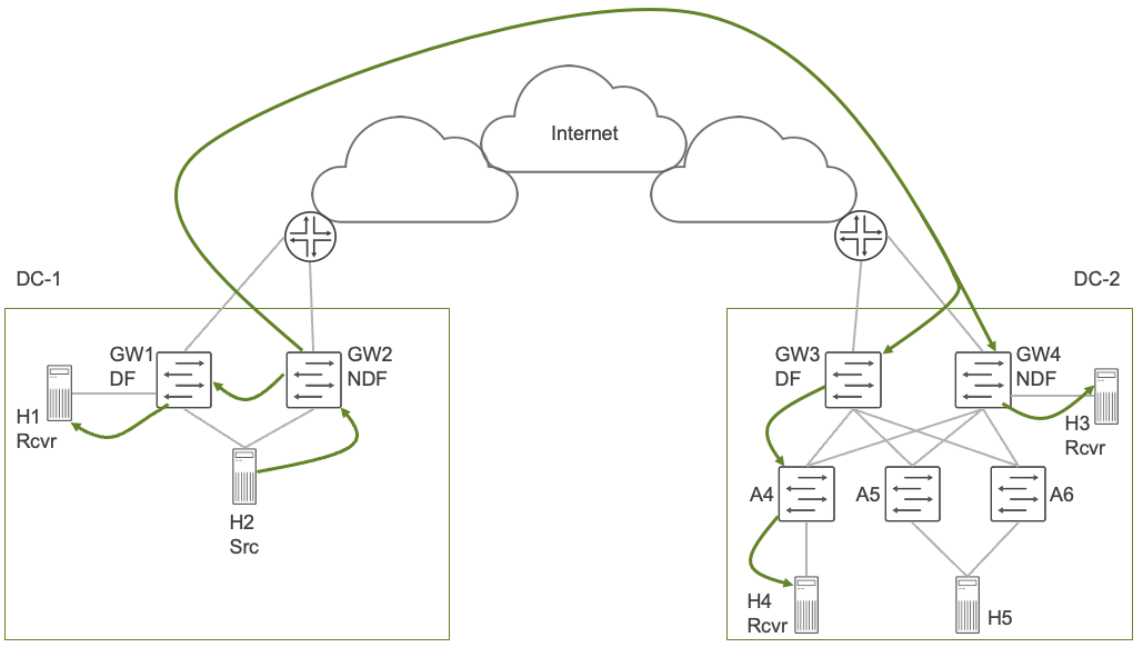
NOTE: Despite GW1 being labeled as the DF, it refrains from forwarding the traffic towards the WAN. In this specific setup, it's more efficient for GW2 to undertake that task, even though GW2 is an NDF.



EVPN DCI Multicast Forwarding Scenario #3

In the following graphic, H2 takes on the role of the multicast source, while H1, H3, and H4 are set up as multicast receivers. H2 sends the multicast traffic directly to GW2, which is an NDF. Due to GW1 and GW2 being multihomed with H2, GW2 passes the multicast traffic over to GW1 using a revenue BD. Additionally, GW2 forwards this multicast traffic from a revenue BD to both GW3 and GW4 over the WAN using the SBD.

Next, GW1 forwards the multicast traffic it received to H1 over a revenue BD. Interestingly, GW1 avoids forwarding the multicast traffic towards the WAN, given that the initial transmission was directly from H2 to GW2. Progressing further, GW3, recognized as the DF, forwards the multicast traffic to A1 through the SBD. Lastly, even though GW4 is an NDF, it actively forwards the multicast traffic from the SBD directly to H3, making use of a revenue BD.

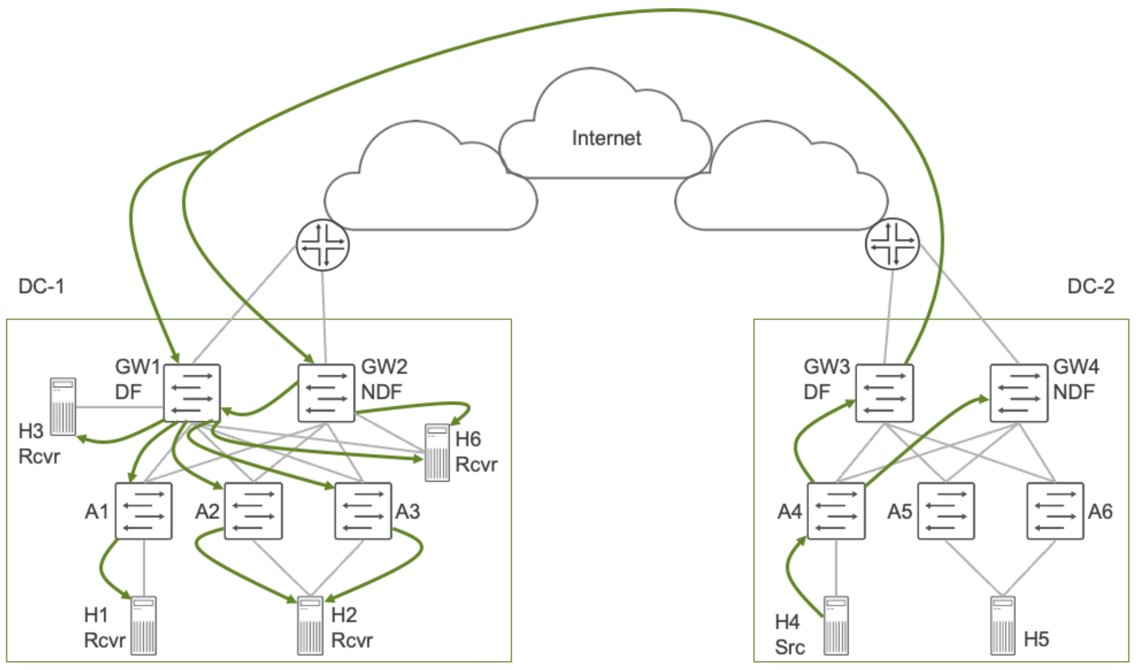


EVPN DCI Multicast Forwarding Scenario #4

In the following graphic, H4 is the multicast source, with H1, H2, H3, and H6 designated as the multicast receivers. When H4 initiates multicast traffic, A4 receives it and forwards the traffic from a revenue BD to both GW3 and GW4 using the SBD. Then, GW3 forwards this multicast traffic over the SBD across the WAN, to both GW1 and GW2.

Upon receipt, GW1 forwards this multicast traffic to A1, A2, and A3, all through the SBD. Due to A2 and A3 being multihomed with H2, either device has the capability to forward the multicast traffic from the SBD directly to H2, utilizing a revenue BD. In a similar stride, GW1 forwards the multicast traffic to H3, once again using a revenue BD. It's noteworthy that, thanks to the multihomed connection between GW1, GW2, and H6, the designated forwarder (DF) can efficiently forward the multicast traffic from the SBD straight to H6 in a revenue BD. Concluding the flow, A1 forwards the multicast traffic from the SBD to the revenue BD H1 is part of.

NOTE: Even though GW4 receives the multicast traffic from A4, it does not forward it across the WAN as it is an NDF.

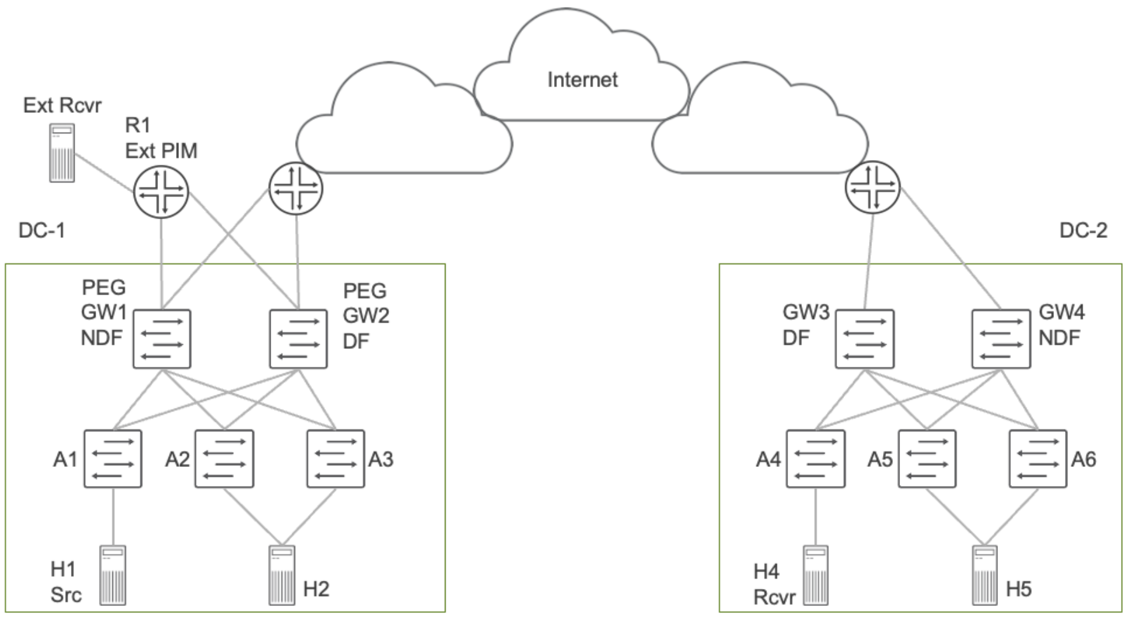


EVPN DCI Multicast PEG Placement Recommendations

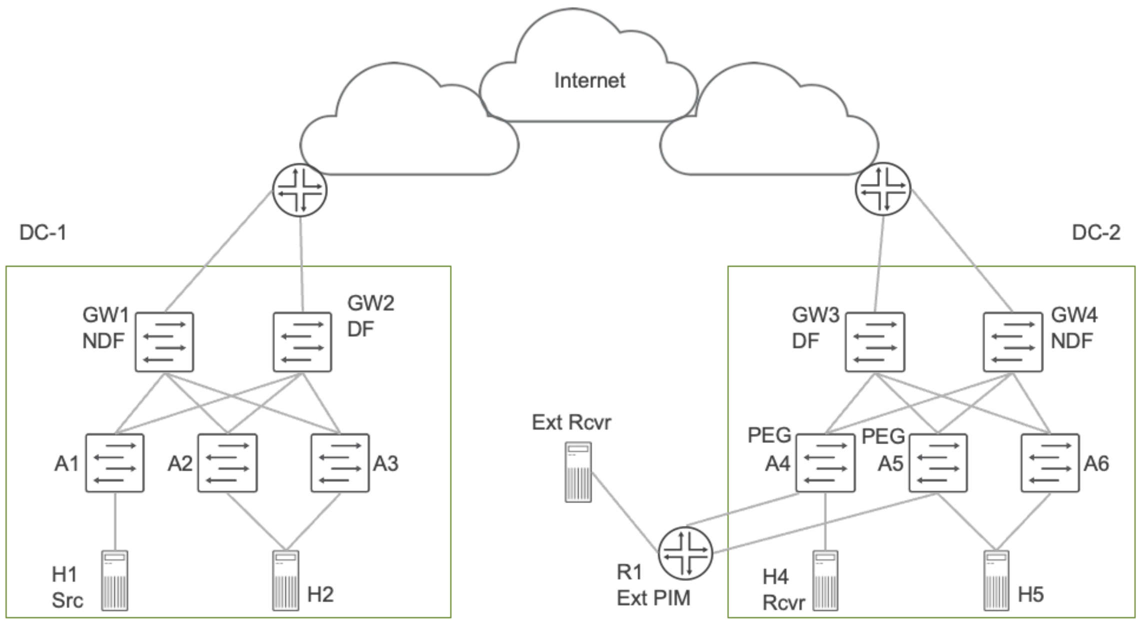
When aiming to establish PEG redundancy at the data center level, we should note that all the DCI gateways in both DCs are configured as PEGs. Under this configuration, PIM is enabled on the irb.sbd interface of these PEG devices. This leads these PEG devices to form PIM neighborships through the irb.sbd interfaces. In this situation, one PEG is chosen as the PIM DR. This elected PIM DR has the responsibility of forwarding the multicast traffic from external sources to the receivers situated within the DCs.

Each PEG device undertakes the task of broadcasting the EVPN Type-3 route, embedding the PEG bit within the MF community. They also advertise the EVPN Type-6 (*,*) route both within the DC and to other DCs. Consequently, any source traffic originating within a DC or from other DCs is directed towards the PEG devices using the SBD. Once the PEG devices receive this traffic, they efficiently route it to the external receivers.

To provide redundancy, we recommend that you configure two devices as the PEGs. The following graphic shows that the DCI gateways in the same data center, GW1 and GW2, as the PEGs.



The following graphic shows that the server leafs in the same data center, A4 and A5, are the PEGs.



We do not recommend the PEG functionality in the following scenarios.

- DCI GWs in different data centers.
- Leafs in different data centers.
- One DCI GW and one leaf in the same data center.
- One DCI GW and one leaf in different data centers.

NOTE: Existing rules of configuring PEG devices within a data center still apply for a DCI setup as well. For example, a PEG device and a non-PEG device cannot be multihomed peers.

EVPN DCI Multicast and Unicast Routing for Multicast RPF

For PEG devices, a unicast routing protocol is essential for the external interface that connects to the broader external environment. This can be OSPF or another unicast routing protocol that is compatible with PIM.

Specifically for the PEG devices, it is necessary for OSPF, or another PIM supported unicast routing protocol, to operate on the external interface that reaches out to the outside world. Furthermore, if you are using OSPF, you should enable it on the irb.sbd interface. In addition, to advertise or learn about the subnets of the BDs, OSPF should be set to passive mode on the IRB interfaces of the revenue BDs.

Similarly, for the server leaf devices, it's also essential to enable OSPF on the irb.sbd interfaces. And just like with the PEG devices, OSPF passive mode should be enabled on the IRB of the revenue BDs to enable the advertising and learning of the subnet addresses of these BDs.

NOTE: A unicast routing protocol that is supported by PIM is imperative for facilitating EVPN DCI with OISM BDNE, given that not all devices will be hosting every BD.

NOTE: A critical configuration step involves enabling the PIM `accept-remote-source` statement for the irb.sbd interfaces. Adding this configuration enables multicast RPF checks and multicast traffic acceptance.

For the DCI Gateway, regardless of whether it's set up as a PEG, the requirement for a unicast routing protocol remains consistent.

Within the Layer 3 VRF, OSPF needs to be up and running on the irb.sbd interface, and similarly, OSPF passive mode must be enabled on the IRB of the revenue BDs. As a result of this setup, a DCI Gateway can form an OSPF neighborship with DCI gateway devices from other data centers and also foster connections between DCI Gateway and server leafs devices inside a DC. In this scenario, configuring the PIM `accept-remote-source` statement for the irb.sbd interface is necessary. The following output is an example of this configuration.

```
[edit protocols pim]
user@Access1# set interface irb.sbd accept-remote-source
```

By following these steps, the Layer 3 multicast routes will be installed, allowing the traffic to be directed according to standard PIM protocol procedures.

CHAPTER 6 Multihoming EVPN Backup Liveness

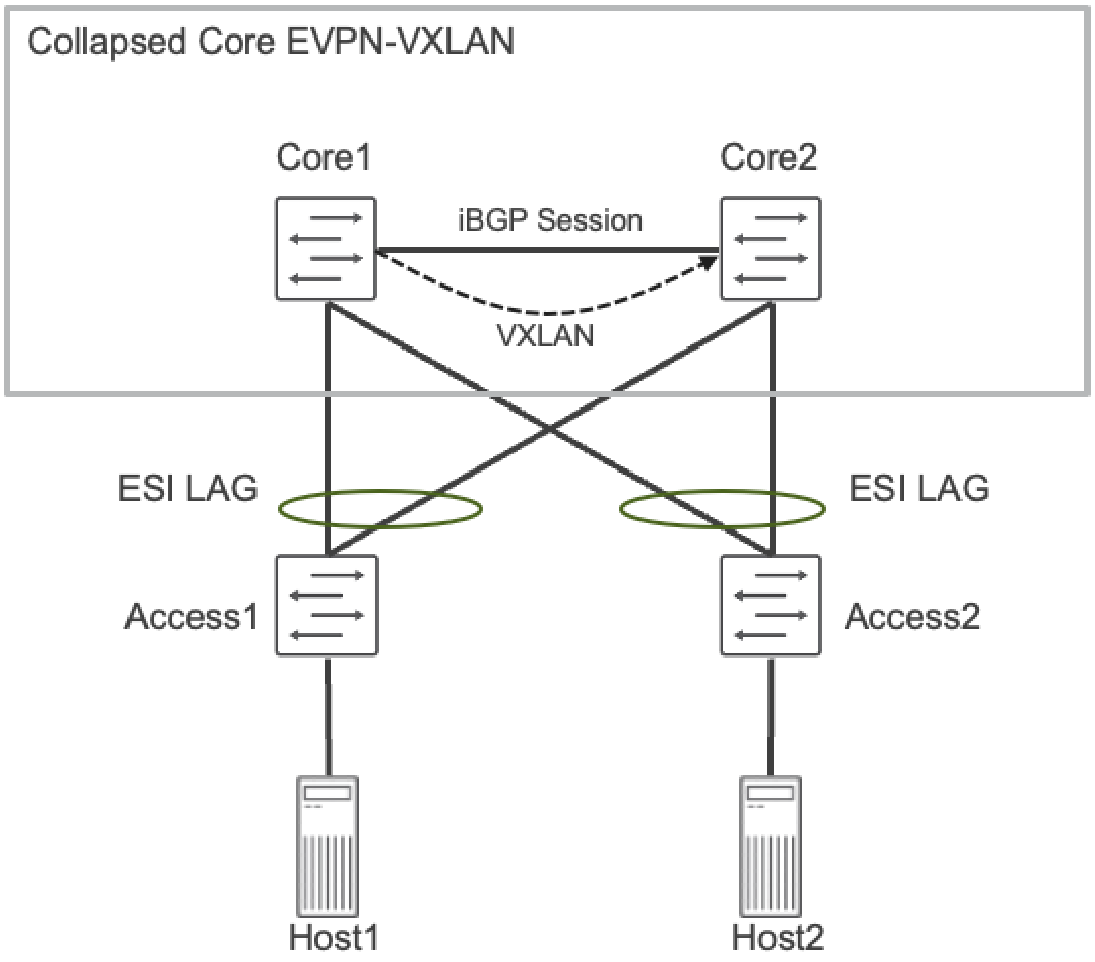
What is Multihoming EVPN Backup Liveness?

The following diagram shows a collapsed core EVPN-VXLAN campus network. In this network, the two core devices are running EVPN-VXLAN with an iBGP session sharing EVPN routes between them. Then, the core devices have ESI LAGs between them and the access switches.

By default, an EVPN network implements the core isolation feature. With this feature, if the core devices lose their iBGP session, LACP automatically brings down the ESI LAGs to ensure that data traffic is not sent toward the isolated core device. With this idea in mind, if the link between Core1 and Core2 goes down, the iBGP session also goes down. As the core isolation feature is enabled by default, LACP sets the access-facing interfaces of the core devices to standby mode, which results in traffic from both access devices being dropped. In this scenario, traffic in the campus network is essentially stopped, which, as you can guess, is not a great outcome.

In this scenario, we can turn off the core isolation feature, but doing so comes with some problems. If it’s turned off on both core devices and the iBGP session goes down, Layer 3 traffic forwarding can fail as ARP and ND resolution fails. In this situation, when Core1 generates an ARP or ND request, the reply might get load balanced to Core2. As the iBGP session is down, control plane sync doesn’t happen, and that means ARP or ND failures.

Alternatively, we can only disable core isolation on Core1. If the core is isolated in this situation, Core2 would bring down its access-facing interfaces, but Core1 would keep its access-facing interfaces up. As in the previous scenario, Layer 3 traffic to and from the access devices is dropped, as Core2 ends up bringing down its access-facing interfaces.



The solution to this problem is to track the liveness between EVPN peers, which is Core1 and Core2 in our scenario. The liveness detection is accomplished using Bidirectional Forwarding Detection (BFD). If the BFD session goes down, one of the core devices is set to take action to ensure traffic can be forwarded through the network. This BFD liveness session can run over standard interfaces, such as the interfaces between the core devices, or over the management interfaces of the core devices.

If the iBGP session between our two core devices goes down, the two core devices are isolated. If the liveness detection feature is enabled in this scenario, and the BFD session between the core devices stays up, the core device with the action configured brings its ESI LAG interfaces towards the access devices down. The core device that doesn't have the action configured leaves its ESI LAG interfaces in the Up state.

In the event that the iBGP session and the BFD session between the core devices go down, which indicates that the core device has failed, the core devices that have the configured action keep the ESI LAG interface up, and no action occurs on the other core device. When the BFD session comes back up, the core device that has the action configured will again bring down the interfaces associated with its ESI LAGs.

There are two options for the action; turning the interface lasers off and triggering node isolation. When the action is configured to turn the lasers off, the lasers are off for interfaces of the core devices ESI LAGs during core isolation. If the action is set to trigger node isolation, the ESI LAG interfaces in the link-down state.

The following table summarizes some common scenarios.

BGP state	Node liveness BFD state	On Core where action is configured	On Core where action is not configured	Comments
Transitions from Up to Down	Down	ESI-LAG remains Up	ESI-LAG remains up	For example, on Core1 both BGP and BFD are down. This means that Core2 is unavailable. Keep ESI LAG UP regardless of action configuration.
Transitions from Up to Down	Up	Bring ESI-LAG down	ESI-LAG remains up	BGP state is down but BFD is up. This is split-brain scenario and only one ESI LAG has to be up. Bring down one LAG where the action is configured.
Transitions from Down to Up	Down	If ESI LAG is down, bring it UP	ESI-LAG remains up	BGP status takes precedence.
Transitions from Down to Up	Up	If ESI LAG is down, bring it UP	ESI-LAG remains up	BGP status takes precedence.

Configuring and Verifying Multihoming EVPN Backup Liveness

The following configuration shows that node detection is configured for Core1. The node detection is occurring over the management interface em0 with the BFD parameters shown. The action that this node takes in the event of a failure is to turn the lasers off for the affected ESI LAG member interfaces.

```
user@Core1> show configuration protocols evpn
node-detection {
    next-hop em0.0;
    bfd-liveness-detection {
        minimum-interval 100;
        multiplier 3;
        neighbor 10.54.8.83;
    }
    action laser-off;
}
```

The following configuration is the same as the previous configuration, except for the action, which in this case is set to trigger node isolation instead.

```
user@Core1> show configuration protocols evpn
node-detection {
    next-hop em0.0;
    bfd-liveness-detection {
        minimum-interval 100;
        multiplier 3;
        neighbor 10.54.8.83;
    }
    trigger-node-isolation;
}
```

If the `laser-off` statement is used, then you must configure the following statement for the members of the ESI LAG.

```
[edit interfaces]
user@Core1# show
ge-0/0/0 {
    gigether-options {
        asynchronous-notification;
    }
}
```

If the `trigger-node-isolation` statement is used, then you must configure a node isolation profile with the link-down action to bring the necessary interfaces down on core isolation.

```
[edit]
user@Core1# show
protocols {
    network-isolation {
        group network-isolation-group-name {
          detection {
              hold-time {
                  up milliseconds;
              }
              service-tracking {
                     core-isolation;
              }
           }
           service-tracking-action {
              link-down;
           }
         }
      }
```

You can view the BFD from the core devices as shown in the following output.

```
user@Core1> show bfd session
                                               Detect   Transmit
Address               State     Interface      Time     Interval  Multiplier
10.54.8.83            Up        ge-0/0/3.0     0.300     0.100        3
```

When the action is set to `laser-off`, you can examine the syslog messages for state information.

```
user@Core1> show log messages | match isolat
Sep  6 17:48:55  Core1 18956[EVPN_CORE_ISOLATED]:
Sep  6 17:48:55  Core1 18956[EVPN_vmm _ISOLATED]:
Sep  6 17:48:59  Core1 18956[EVPN_CORE_ISOLATED]:
Sep  6 17:48:59  Core1 18956[EVPN_CORE_ISOLATED]:
Sep  6 17:49:55  Core1 18956[EVPN_CORE_ISOLATION_CLEARED]:
```

```
Sep  6 17:50:05  Core1 18956[EVPN_CORE_ISOLATION_CLEARED]:
Sep  6 17:50:05  Core1 18956[EVPN_CORE_ISOLATION_CLEARED]:
```

CHAPTER 7 EVPN-VXLAN L2 MAC Limit, MAC Move Limit, and Persistent MAC

EVPN-VXLAN MAC limits

One important part of managing a network is to set a MAC address learning limit for an EVPN routing instance. There are many reasons for doing this, including:

- **Security**: An attack known as MAC flooding can overload a switch with too many MAC addresses, causing it to behave unpredictably. By setting a limit, you can prevent too many MAC addresses from being learned.
- **Resource Management**: All network devices have finite resources. Too many MAC addresses can consume a device's memory. By setting a limit, you can make sure the device operates efficiently and does not run out of resources.
- **Predictable Performance**: Uncontrolled growth in MAC address tables can lead to unpredictable network behavior and performance issues. By setting a threshold, network administrators can maintain a predictable level of performance.
- **Operational Stability**: If the MAC address table becomes full, newer addresses might not be learned, or older addresses might be removed too soon. This can cause operational issues. Setting a limit can help maintain stability.
- **Isolation of Faults**: Sometimes, a sudden increase in MAC addresses can be a sign of a network issue, like a loop. By monitoring the MAC address table and setting limits, you can quickly identify and fix problems like these.
- **Fine-tuning Network Behavior**: In multi-tenant environments or when dealing with different service levels, you might want to allocate more resources to higher-tier services and fewer resources to lower-tier services. By setting MAC address limits, you can adjust the behavior and resource allocation of your network to different segments.

When configuring MAC limits for an EVPN routing instance on a Juniper device, it is important to ensure that the limits are suitable for the network's operational requirements and are updated regularly to adapt to any changes in network behavior or needs.

You configure MAC learning limits for EVPN at the `[edit routing-instances` *`routing-instance-name`* `protocols evpn]` or `[edit routing-instances` *`routing-instance-name`* `protocols evpn interface` *`interface-name`*`]` configuration levels using the `limit` statement. You can set the range from 1 to 131,017 MAC addresses.

EVPN-VXLAN MAC Move Limits

MAC move limits are a crucial aspect of network management and security. Essentially, EVPNs provide layer-2 connectivity over an IP network, enabling MAC addresses to shift as workloads move across different parts of the network.

A MAC move occurs when a switch detects the same MAC address on a different interface or port than where it first learned it. This may happen for valid reasons, such as VM migrations or device mobility, but it can also be caused by network problems or malicious activities.

Turning on MAC move limits provides security by limiting MAC moves helps protect against certain types of attacks, like MAC spoofing, where an attacker might try to associate their device with an already known MAC address in the network.

One way to improve network stability is to monitor the frequency of MAC moves, which can signal issues like network loops. By setting a threshold, potential problems can be identified quickly. Additionally, limiting MAC moves can improve device performance by reducing churn in the MAC address table and preventing performance degradation.

To set a MAC move limit, it's important to find a balance between security and flexibility. An overly strict limit might mistakenly flag legitimate MAC moves, like VM migrations, as security threats and even block them. On the other hand, a too-lenient limit may not provide the necessary level of security and stability.

You can configure MAC move limits you begin under the `[edit vlans vlan-name switch-options]` configuration hierarchy. From there, you can use the limit statement to specify how many times a MAC address can move limit for the VLAN and the actions to be taken when the limit is reached.

The actions that can be taken are as follows:

```
[edit vlans VLAN-NAME switch-options mac-move-limit]
user@Access1# set packet-action ?
Possible completions:
  drop                 Drop the packet and do not generate an alarm
  drop-and-log         Drop the packet and generate an alarm, an SNMP trap, or a system log entry.
  log                  Do not drop the packet but generate an alarm, an SNMP trap, or a system log entry.
  none                 Take no action
  shutdown             Disable the interface and generate an alarm, an SNMP trap, or a system log entry.
```

When EVPN-VXLAN is enabled, a VLAN can include local network ports and VTEP interfaces as members, allowing for various possibilities of MAC moves such as:

- A MAC move between network ports. By default, configured action will be applied on the port where the MAC is moved last. If action priority is configured on the ports, then action is applied on the port with less priority.
- A MAC move between VTEP ports. This mac move is ignored, and no action is taken.
- A MAC move between a network and a VTEP port. In this case, action will be applied always on the network port.

EVPN-VXLAN Persistent MAC Learning

Persistent MAC learning, also known as sticky MAC, is a port security feature that enables an interface to retain dynamically learned MAC addresses when the switch is restarted or if the interface goes down and is brought back online.

Persistent MAC address learning is disabled by default. You can enable persistent MAC address learning in conjunction with MAC limiting to restrict the number of persistent MAC addresses. You enable this feature on interfaces.

Configure persistent MAC learning on an interface to:

- Prevent traffic losses for trusted workstations and servers because the interface does not have to relearn the addresses from ingress traffic after a restart.
- Protect the switch against security attacks. Use persistent MAC learning in combination with MAC limiting to protect against attacks, such as Layer 2 denial-of-service (DoS)

attacks, overflow attacks on the Ethernet switching table, and DHCP starvation attacks, by limiting the MAC addresses allowed while still allowing the interface to dynamically learn a specified number of MAC addresses. The interface is secured because after the limit has been reached, additional devices cannot connect to the port.

By configuring persistent MAC learning along with MAC limiting, you enable interfaces to learn MAC addresses of trusted workstations and servers from the time when you connect the interface to your network until the limit for MAC addresses is reached, and ensure that after this limit is reached, new devices will not be allowed to connect to the interface even if the switch restarts. As an alternative to using persistent MAC learning with MAC limiting, you can statically configure each MAC address on each port or allow the port to continuously learn new MAC addresses after restarts or interface-down events. Allowing the port to continuously learn MAC addresses represents a security risk.

To configure persistent MAC learning, you must set a specific interface to use it under the `[edit switch-options]` hierarchy, as shown in the following output.

```
[edit switch-options]
user@Access1# set interface interface-name persistent-learning
```

With regards to EVPN-VXLAN, you can configure the persistent MAC learning feature on network interfaces, but you cannot configure it on VTEP interfaces. If you attempt to configure it on a VTEP interface, you cannot commit the configuration.

CHAPTER 8 Dynamic IPv6 Filters

Dynamic IPv6 Filters with 802.1x

On EX Series switches, firewall filters that you apply to interfaces enabled for 802.1X or MAC RADIUS authentication are dynamically combined with the per-user policies sent to the switch from the RADIUS server. The switch uses internal logic to dynamically combine the interface firewall filter with the user policies from the RADIUS server and create an individualized policy for each of the multiple users or nonresponsive hosts that are authenticated on the interface.

When the 802.1X configuration on an interface is set to multiple supplicant mode, the system dynamically combines interface firewall filter with the user policies sent to the switch from the RADIUS server during authentication and creates separate terms for each user. Because there are separate terms for each user authenticated on the interface, you can use counters to view the activities of individual users that are authenticated on the same interface.

Previously, this feature was only supported for as part of the match condition for destination IPv4 addresses. Now, IPv6 match support has been added and this enables action to be taken based on the destination IPv6 address. The filter can be installed as a filter-ID or switching-filter received from RADIUS server. The switching-filter will have counter similar to other match conditions. Also, both IPv4 and IPv6 match conditions can be specified within same filter.

The following example is for a switching-filter.

```
supp1 Auth-Type = EAP,Cleartext-Password := "supp"

        Juniper-Switching-Filter = "match dst 19.82.1.3 action deny",
        Juniper-Switching-Filter = "match dst
232:231:454:0a0:9b9:8c8:7d7:6e6/64 action deny"
```

The following example is for a filter-ID.

```
supp2 Auth-Type = EAP,Cleartext-Password := "supp"

              Filter-Id = "f1"
```

The following output show the Junos configuration for the filter-ID.

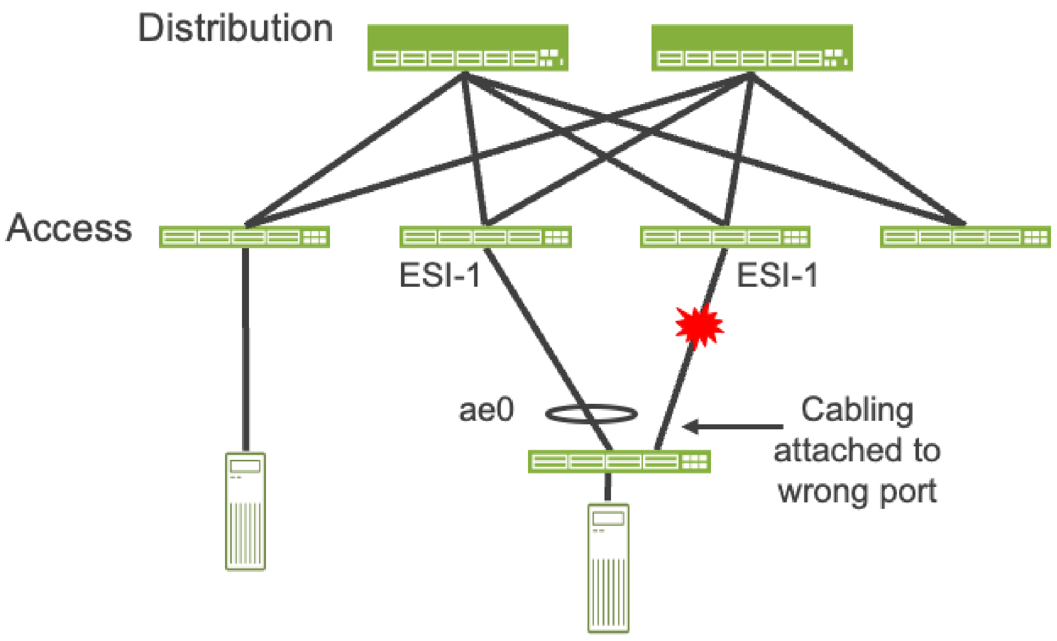
```
[edit]
user@Access1# set firewall family ethernet-switching filter f1 term t1 from
ip-version ipv6 ip6-destination-address 232:231:454:0a0:9b9:8c8:7d7:6e6/128

[edit]
user@Access1# set firewall family ethernet-switching filter f1 term t1 then
discard
```

NOTE: An IPv6 address can be specified in compressed or expanded format. For example; 21DA:00D3:0000:0000:0000:02AA:0008:9C5A can be compressed to 21DA:D3::2AA:8:9C5A.

CHAPTER 9 EVPN-VXLAN Lightweight Leaf to Server Loop Detection

Why is EVPN-VXLAN Lightweight Leaf to Server Loop Detection Needed?

The Ethernet VPN–Virtual Extensible LAN (EVPN-VXLAN) network has revolutionized data center topology by eliminating many of the challenges of the traditional three-stage setup. Specifically, it has significantly reduced the chances of Layer 2 loops by getting rid of east-west interfaces. However, it's worth noting that the possibility of Layer 2 loops still exists. This can happen due to incorrect cabling or configurations. For instance, in the implemented EVPN multihomed topology shown in the graphic, Leaf 3 has been misconfigured with ESI-2 instead of ESI-1. This creates an issue with the EVPN multihomed designator election and creates a loop.



EVPN-VXLAN lightweight leaf-to-server loop detection solves this potential loop problem in an EVPN environment. The loop-detect protocol is a component of connectivity fabric management.

Once it is configured, the interfaces periodically send multicast protocol data units (PDUs). If a loop detection-enabled interface receives the PDU, it detects a loop and triggers the configured action to break the loop. In the graphic on the slide, the configured action is to shut down the interface.

The loop-detect protocol is capable of identifying loops between two interfaces located in different ESIs. These loops are usually caused by miswiring or misconfiguration of fabric components or interfaces. Additionally, the protocol can also detect loops between two interfaces that have the same ESI. This type of loop is commonly caused by miswiring a third-party switch to the fabric.

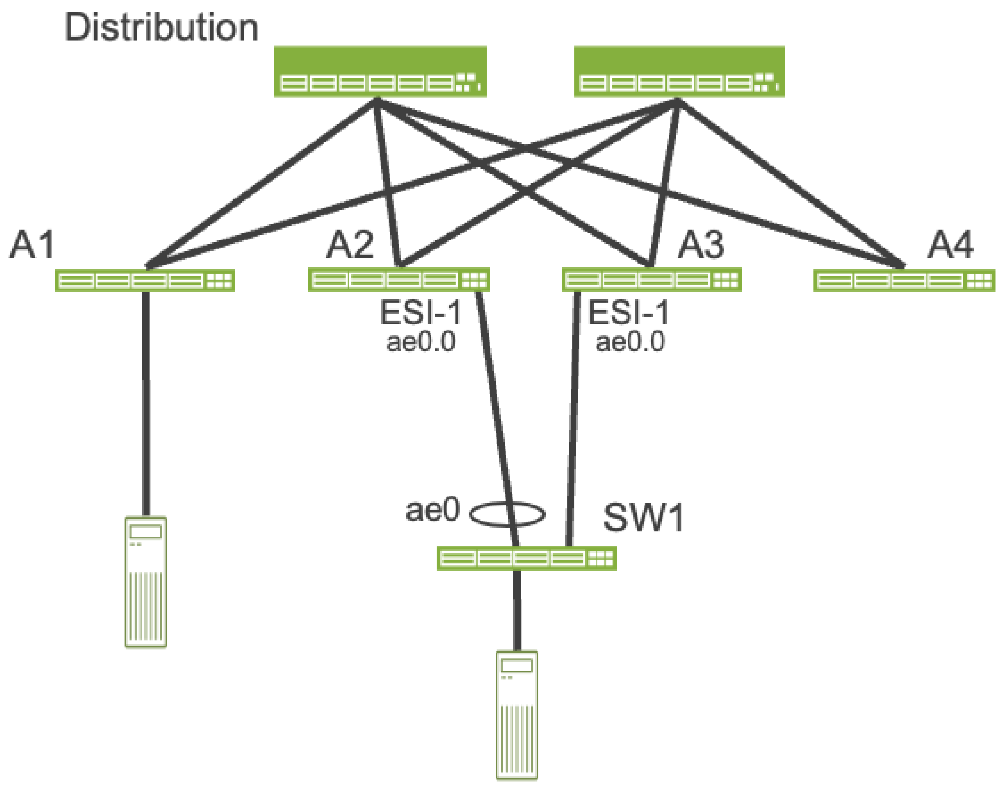
The Loop-detect feature, found in the protocol hierarchy, is a cost-effective solution that is more efficient than the spanning tree method. It is a part of Connectivity Fault Management (CFM), which adheres to the IEEE 802.1 AG standard for Operation, Administration, and Maintenance (OAM).

Ethernet networks may have one or more service instances that can experience connectivity faults, which can compromise the network. To prevent this, connectivity fault management monitors these networks and provides service-level monitoring through OAM technology. By implementing action policies, this monitoring can help lower operating expenses and give data centers a competitive edge.

Connectivity fault management helps monitor network problems. It uses a system called continuity check protocol to find and keep track of connections within groups of computers. Another tool it uses is the Link Trace protocol. This tool helps you figure out the route between two specific points in the network. When an administrator wants to check this route, they use a command called "traceroute" to send a message between those two points. The loop-detect protocol is a feature of CFM. Each of these features operates independently.

EVPN-VXLAN Lightweight Leaf to Server Loop Detection Configuration

The following graphic shows a use case for loop-detect. We have a three-stage data center topology with devices A2 and A3 configured for EVPN multihoming, and the links are servicing a downstream switch configured with an aggregate Ethernet interface. In our example, the SW1 device has the link to A3 that is connected to a port that is not configured as a member of AE0. This scenario can create a loop.



The following output shows the ae0 interface configuration for A2.

```
[edit]
user@A2# show interfaces ae0
esi {
    00:00:00:00:00:00:00:00:00:01;
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 00:00:00:00:00:01;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members 10;
        }
    }
}
```

The following output shows the ae0 interface configuration for A3.

```
[edit]
user@A3# show interfaces ae0
esi {
    00:00:00:00:00:00:00:00:00:01;
    all-active;
}
aggregated-ether-options {
    lacp {
        active;
        system-id 00:00:00:00:00:01;
    }
}
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members 10;
        }
    }
}
```

The following output shows the loop-detect protocol configuration.

NOTE: The loop-detect configuration for A3 is not shown, as it is exactly the same as what is shown here for A2.

```
[edit]
user@A2# show protocols loop-detect enhanced
interface ae0 {
    loop-detect-action interface-down;
    transmit-interval 1m;
    revert-interval 300;
    vlan-id 10;
}
```

The first option we see is the loop-detect action. The output shows that the action is to shut the interface down. This brings down the interface that detected the misconfiguration. Another option is to turn the laser of the physical interfaces off. To this end, the `laser-off` statement turns off the optics transmit laser on the device.

You have the ability to bring the affected interface back online using the `revert-interval` parameter. This parameter identifies, in seconds, the waiting period and then automatically brings the interface back up. If the loop-detect protocol shuts down an interface, and you do not specify a revert interval, the interface must be brought back up manually.

The `transmit-interval` parameter defines the frequency of the multicast transmissions. If no transmit interval is configured, the default is 1 second.

The `vlan-id` option is required for trunk interfaces or interfaces with enterprise style configuration. With this parameter, you specify the VLAN identifier for the interface within the range of 1 through 4094.

EVPN-VXLAN Lightweight Leaf to Server Loop Detection Verification

The following output shows loop detection working without any loop detected.

```
{master:0}
user@A2> show loop-detect enhanced interface
Interface :ae0.0
Vlan-id :10
ESI :00:00:00:00:00:00:00:00:00:01
Current status :Normal[Link Up]
Last loop-detect time :-
Receive statistics :0
Action configured :Interface-down
Action count :0
Transmit Interval :1s
Revert Interval :60s
```

The following output shows loop detection working when a loop is detected.

```
{master:0}
user@A2> show loop-detect enhanced interface
Interface :ae0.0
Vlan-id :10
ESI :00:00:00:00:00:00:00:00:00:01
Current status :Loop-detected
 Remote Host :A3
 Remote Chassis :94:f7:ad:94:dd:40
 Remote Interface :ae0
 Remote ESI :00:00:00:00:00:00:00:00:00:01
Last loop-detect time :Tue May 26 04:36:37 2020
Receive statistics :4
Action configured :Interface-down
Action count :1
Transmit Interval :1s
Revert Interval :60s
```

CHAPTER 10 EVPN-VXLAN Loop Prevention on Link Recovery

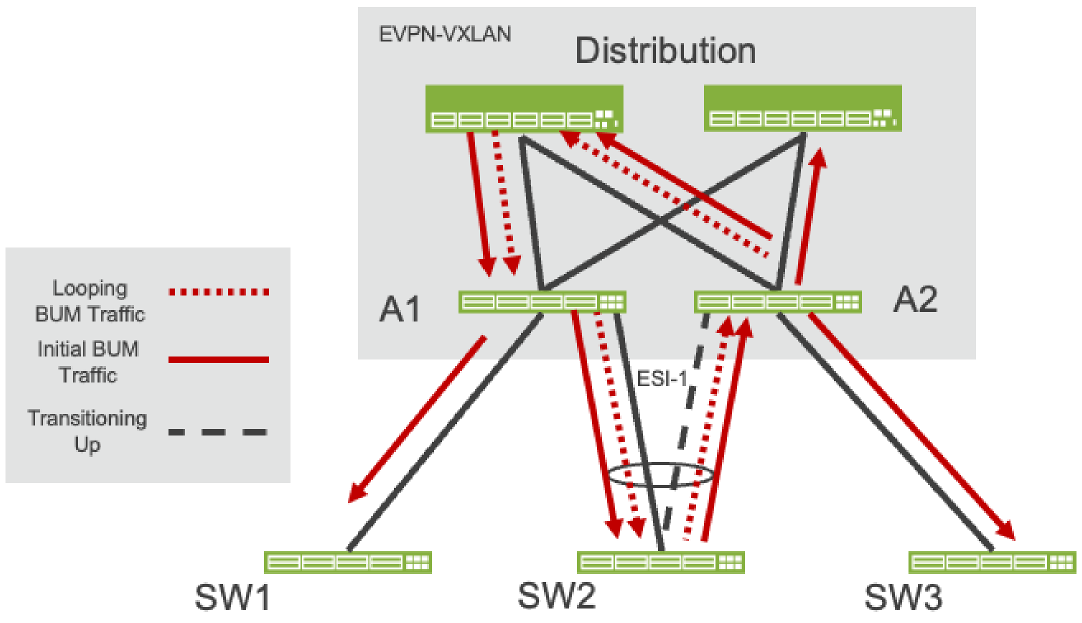
How Does a Loop Happen on Link Recovery?

In an EVPN-VXLAN multihoming environment, brief flood (Broadcast, Unknown unicast, Multicast) traffic surges can occur between the DF and non-DF routers when the edge link returns to an operational status. The EVPN-VXLAN framework includes a local filter to circumvent loop complications. However, there's a brief delay in activating this filter after the interface has been restored. During this interval, traffic is directed to the other multihoming router through the ESI LAG interface.

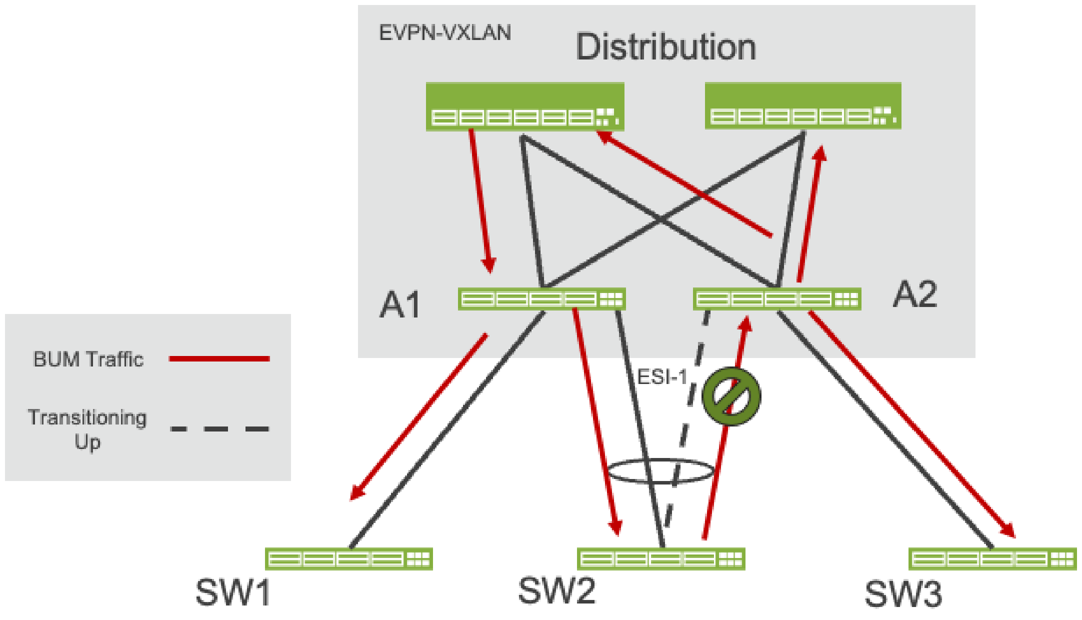
A potential solution to this challenge is to incorporate a brief recovery period for BUM traffic once the interface is restored. This means halting BUM traffic momentarily after interface restoration, ensuring the local filter has adequate time to activate.

In the following graphic, when the CE4-PE3 link is restored, a BUM traffic loop is initiated.

Shortly after the recovery of the A2-SW2 link, BUM traffic originating from SW2 travels through ESI-1. This traffic then chooses a path to A2 and subsequently reaches A1 through the network core. This process occurs before the split-horizon/local filtering for ESI-1 can be set on A1. As a result, a BUM traffic loop is created, with A1 directing the BUM traffic back towards SW2.



The problem can be addressed by delaying BUM traffic for a few seconds after the interface has been restored from the down state. When the A2-SW2 link is reinstated, A2 receives BUM traffic from the multihomed SW2. However, A2 temporarily blocks this incoming BUM traffic from SW2 for a short time.



Configuration and Verification

You can enable this feature by setting a blocking window duration in seconds with the **`set global-evpn-bum-blocking-hold-timer`** ***`seconds`*** command at the `[edit protocols l2-learning]` configuration hierarchy. In the absence of a specified timer setting, a default value of 5 seconds will be applied. The adjustable range for this setting spans from 2 to 60 seconds.

The following output shows that the feature is configured and working with a five-second hold timer.

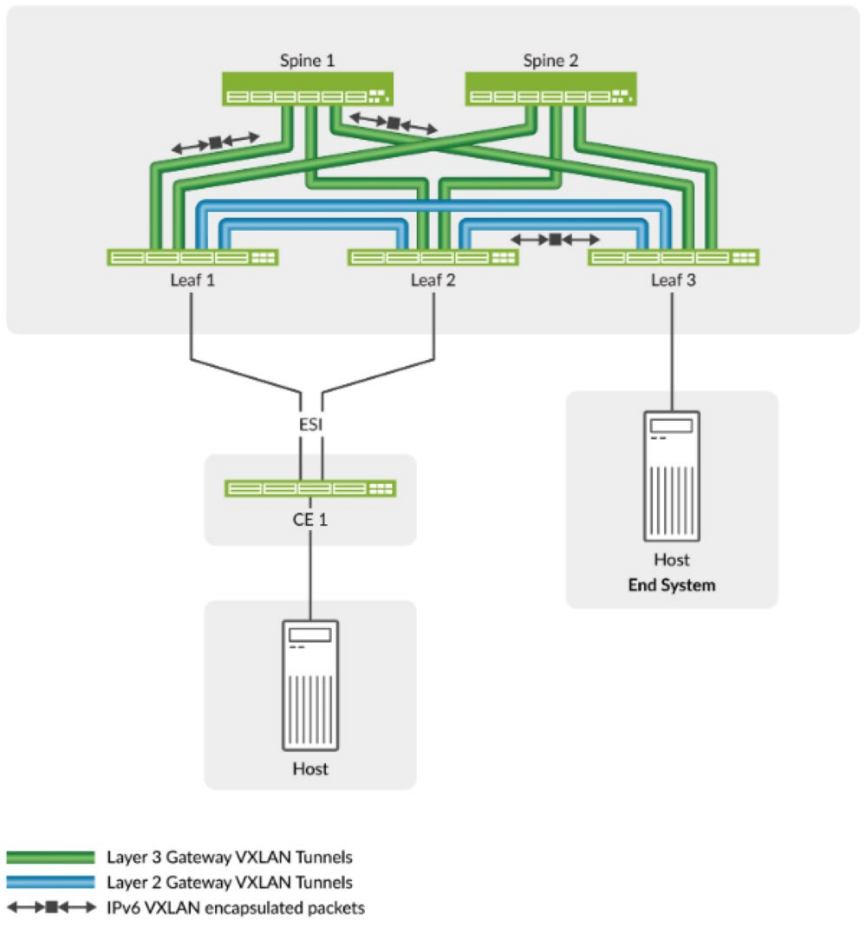
```
user@A2> show l2-learning global-information
   Global Configuration:

   MAC aging interval     : 300
   MAC learning           : Enabled
   MAC statistics         : Disabled
   MAC limit Count        : 393215
   MAC limit hit          : Disabled
   MAC packet action drop: Disabled
   MAC+IP aging interval : IPv4 - 1200 seconds
                           IPv6 - 1200 seconds
   MAC+IP limit Count     : 393215
   MAC+IP limit reached   : No
   LE  aging time         : 1200
   LE  BD aging time      : 1200
   MP discard notification interval: 60
   EVPN BUM block hold time    : 5 seconds
```

CHAPTER 11 EVPN-VXLAN with an IPv6 Underlay

IPv6 Underlay Support in EVPN-VXLAN Fabrics

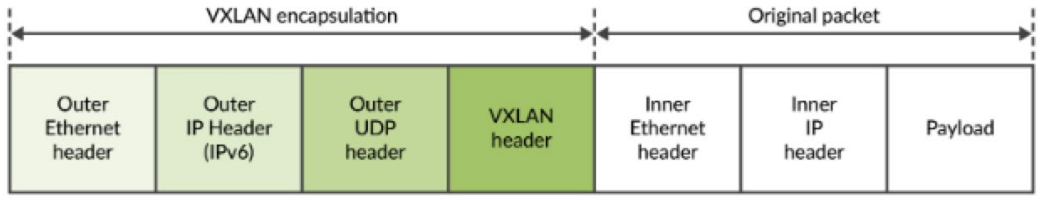
EVPNs link devices through Layer 2 virtual bridges. VXLANs create overlay tunnels, extending the Layer 2 connections across a Layer 3 network. In EVPN-VXLAN setups, a device, whether a leaf or spine, can serve as a VXLAN gateway on Layer 2, Layer 3, or both. While the underpinning network for this VXLAN overlay can be either an IPv4 or an IPv6 network. Utilizing an IPv6 underlay for VXLAN tunnel setups allows you to benefit from the enhanced address capacity and optimized packet handling provided by the IPv6 protocol. This section focuses on utilizing an IPv6 underlay as opposed to an IPv4 one.

In EVPN-VXLAN setups, a VXLAN overlay is configured on VXLAN gateway devices, known as VTEPs, operating at either Layer 2 or Layer 3. This VXLAN overlay forms virtual tunnels connecting VTEPs across the foundational IP network. On compatible devices, the IP foundation can be configured using IPv6 addressing to facilitate the VXLAN overlay tunnels. The following graphic shows an example of an EVPN-VXLAN network that uses an IPv6 underlay.



Spine 1
Spine 2
Leaf 1
Leaf 2
Leaf 3
ESI
CE 1
Host
End System
Host
Layer 3 Gateway VXLAN Tunnels
Layer 2 Gateway VXLAN Tunnels
IPv6 VXLAN encapsulated packets

When you use an IPv6 underlay, the VTEPs encapsulate VXLAN packets with an IPv6 outer header and tunnel the packets through an IPv6 underlay network. The following graphic shows the components of a VXLAN-encapsulated packet.



IPv6 Underlay Support in EVPN-VXLAN Fabrics Configuration

When configuring your EVPN-VXLAN fabric, consider these options and requirements:

- **Underlay Routing Protocols**: Your EVPN-VXLAN network can be set up using various underlay routing protocols that are compatible with IPv6 underlays, including eBGP, iBGP, and OSPFv3.
- **QFX Series Switches Support**: Within EVPN-VXLAN networks, QFX Series switches only support IPv6 underlays when using MAC-VRF routing instances. Ensure all MAC-VRF routing instances have EVPN instances with VXLAN encapsulation. Any routing instances mentioned in this guidance pertain exclusively to EVPN-VXLAN MAC-VRF instances.
- **Enabling IPv6 Underlay for VXLAN Tunneling**: Include the following components in your EVPN-VXLAN configuration:
    - *Loopback Interface Addressing*: Assign an IPv6 address to the loopback interface of the devices functioning as Layer 2 or Layer 3 VXLAN gateway VTEPs.
    - *Device Connectivity Configuration*: Set up the underlay and EVPN instance connectivity for the overlay using any of the routing protocols compatible with IPv6 addressing.
    - *VTEP Source Interface*: Within the EVPN routing instance, define the underlay VTEP source interface using the device's loopback IPv6 address. For a QFX Series switch, the command would be:

```
[edit]
user@Acess# set routing-instances instance-name vtep-source-interface lo0.0
inet6
```

    - *IPv6 Address Family and Router ID*: The underlay employs the IPv6 address family. IPv6 protocols also require a unique 32-bit router ID within the routing domain. This ID should be a 4-octet unsigned non-zero integer. Configure the router ID in dotted quad notation. The overlay, based on IPv6, will use the IPv6 loopback address for the VTEP's local address with the command:

```
[edit]
user@Acess# set routing-options router-id ID
```

CHAPTER 12 Services

Understanding DHCP Snooping

DHCP snooping allows a switch or router to monitor DHCP messages from untrusted devices connected to it. When activated on a VLAN, the system inspects DHCP messages from these untrusted hosts, extracting IP addresses and lease details. This data helps create and update the DHCP snooping database. Only hosts verified through this database can access the network.

DHCP dynamically assigns IP addresses, leasing them to devices so they can be repurposed when not in use. For hosts and end devices to acquire IP addresses through DHCP, they must interact with a DHCP server across the LAN.

By acting as a network security monitor, DHCP snooping keeps a record of legitimate IP addresses given to downstream devices by a trusted DHCP server (linked to a trusted network port).

By default, all switch trunk ports are considered trusted, while access ports are deemed untrusted concerning DHCP snooping.

Trusting an access port or deeming a trunk port untrusted can be achieved via the overrides configuration statement, selecting either the trusted or untrusted option.

Upon enabling DHCP snooping, server lease information is utilized to form the DHCP snooping table (or DHCP binding table). This table displays the current IP-MAC address relationships, lease duration, binding type, and related VLANs and interfaces.

The DHCP snooping table undergoes updates during:

- IP address release by a device (sending a DHCPRELEASE message), which leads to the removal of the related entry.
- A device's move from one VLAN to another. This typically requires a new IP address, prompting an update in the database.
- Expiry of the DHCP server-assigned lease time, resulting in entry deletion.
- Lease renewal by a device through a unicast DHCPREQUEST message, updating the lease duration in the database.
- If a device can't contact the original DHCP server, it sends a broadcast DHCPREQUEST and reconnects to the responding server. This causes a new IP to be assigned, and the DHCP snooping table is updated.
- If a device with a fixed IP from a DHCP server gets replaced by another device with a different MAC, the new IP-MAC binding remains until a DHCPACK message is received. Subsequently, the DHCP snooping table is updated with the new binding.

NOTE: By default, the IP-MAC bindings are lost when the switch is rebooted, and the DHCP clients (the network devices, or hosts) must reacquire bindings. However, you can configure the bindings to persist by setting the `dhcp-snooping-file` statement to store the database file either locally or remotely.

NOTE: You can configure the switch to snoop DHCP server responses only from specific VLANs. Doing this prevents spoofing of DHCP server messages.

Configuring DHCP Snooping

When leveraging the DHCP snooping functionality in Junos OS devices, it's crucial to grasp the conditions under which it's enabled.

In the Junos OS environment, DHCP snooping isn't a standalone feature. It's inherently tied to the configuration of DHCP security for a particular VLAN. Whenever DHCP security is set up for a VLAN, DHCP snooping is consequently activated for that VLAN to carry out its functions.

To clarify, activating DHCP security for a specific VLAN will automatically enable DHCP snooping for that VLAN.

In the context of Junos OS, DHCP snooping becomes active whenever the `dhcp-security` statement is set at the `[edit vlans` *`vlan-name`* `forwarding-options]` hierarchy level.

Before Junos OS Release 17.1, the system would automatically activate DHCP snooping if any of the subsequent port security features were configured within the `[edit vlans` *`vlan-name`* `forwarding-options]` hierarchy:

- Dynamic ARP Inspection (DAI)
- IP Source Guard
- DHCP Option 82
- Static IP

However, from Junos OS Release 17.1R1 onwards, it's possible to set up DHCP snooping or DHCPv6 snooping for a VLAN without having to enable other associated port security functionalities. To initiate DHCP snooping, use the `set vlans vlan-name forwarding-options dhcp-security` configuration statement as we previously mentioned.

NOTE: To disable DHCP snooping, you must delete the `dhcp-security` statement from the configuration. DHCP snooping is not disabled automatically when you disable other port security features.

The DHCP Snooping Process

NOTE: When DHCP snooping is enabled for a VLAN, all DHCP packets sent from network devices in that VLAN are subjected to DHCP snooping. The final IP-MAC binding occurs when the DHCP server sends a DHCPACK packet from the DHCP client.

DHCP snooping involves the subsequent steps:

1. A network device initiates the process by transmitting a DHCPDISCOVER packet to request an IP address.
2. This packet is then relayed by the switch to the DHCP server.
3. In response, the server dispatches a DHCPOFFER packet, proposing an IP address. If this packet originates from a trusted source or interface, the switch conveys it to the network device.
4. Accepting the offered IP, the network device sends out a DHCPREQUEST packet. At this point, the switch introduces a temporary IP-MAC binding in the DHCP snooping table. This entry remains provisional until a DHCPACK packet is confirmed from the server. In the interim, the proposed IP could potentially be allocated to another device.
5. The DHCP server then dispatches either a DHCPACK packet to finalize the IP address allocation or a DHCPNAK packet to decline the request.
6. Based on the received packet, the switch modifies its DHCP database:
   - Receiving a DHCPACK packet prompts the switch to refresh its database with the lease details for the specified IP-MAC binding.
   - If a DHCPNAK packet is received, the provisional entry is eradicated from the database.

NOTE: The DHCP database is updated only after the DHCPREQUEST packet is sent.

DHCPv6 Snooping

While DHCPv6 snooping mirrors the process of its IPv4 counterpart, the communication between the client and server uses distinct message names when assigning IPv6 addresses. The table below presents the equivalent DHCPv6 messages compared to DHCPv4.

Sent By	DHCPv6 Messages	DHCPv4 Equivalent Messages
Client	SOLICIT	DHCPDISCOVER
Server	ADVERTISE	DHCPOFFER
Client	REQUEST, RENEW, REBIND	DHCPREQUEST
Server	REPLY	DHCPACK/DHCPNAK
Client	RELEASE	DHCPRELEASE
Client	INFORMATION-REQUEST	DHCPINFORM
Client	DECLINE	DHCPDECLINE
Client	CONFIRM	none
Server	RECONFIGURE	DHCPFORCERENEW
Client	RELAY-FORW, RELAY-REPLY	none

DHCPv6 Rapid Commit

The DHCPv6 Rapid Commit feature offers a more efficient interaction between client and server. If both the server supports it and the client activates it, this function trims the communication from a four-step process to a simple two-step handshake.

Here's how the exchange operates with the Rapid Commit feature active:

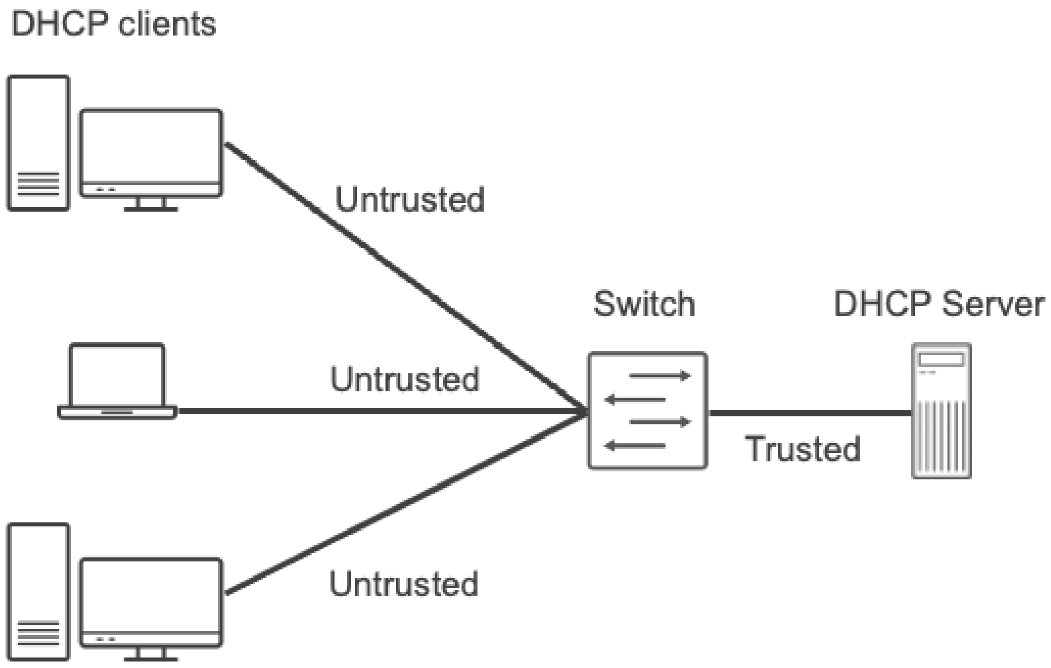
- The DHCPv6 client dispatches a SOLICIT message, signaling its preference for a swift allocation of address, prefix, and other configuration details.
- In turn, if the DHCPv6 server is equipped for quick allocation, it returns with a REPLY message, furnishing the designated IPv6 address, prefix, and additional configuration data.

DHCP Server Access

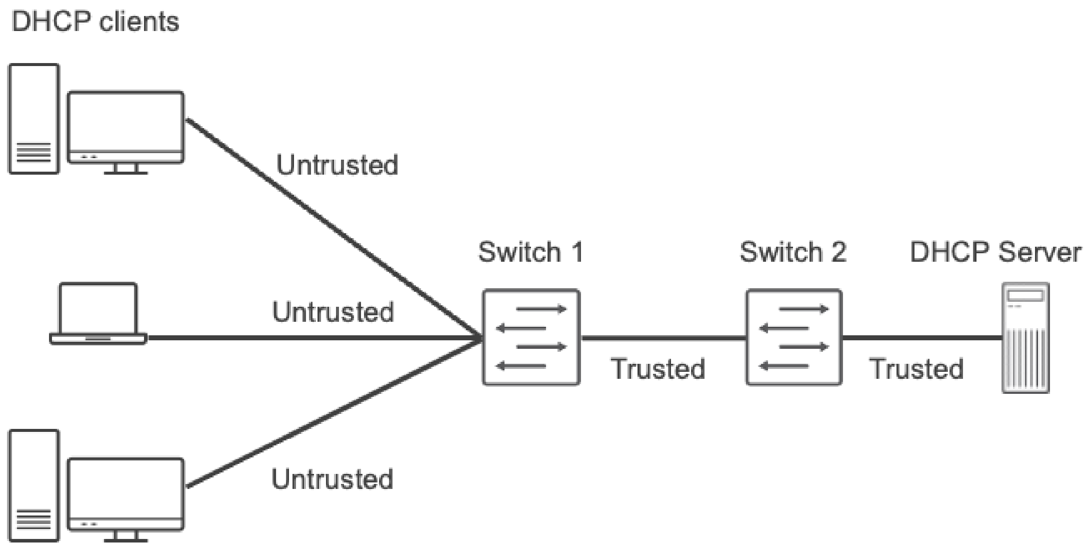
We can configure the switch's access to the DHCP server in three different ways:

- The switch, DHCP clients, and the DHCP server are all on the same VLAN.
- The switch acts as the DHCP server.
- The switch acts as a relay agent.

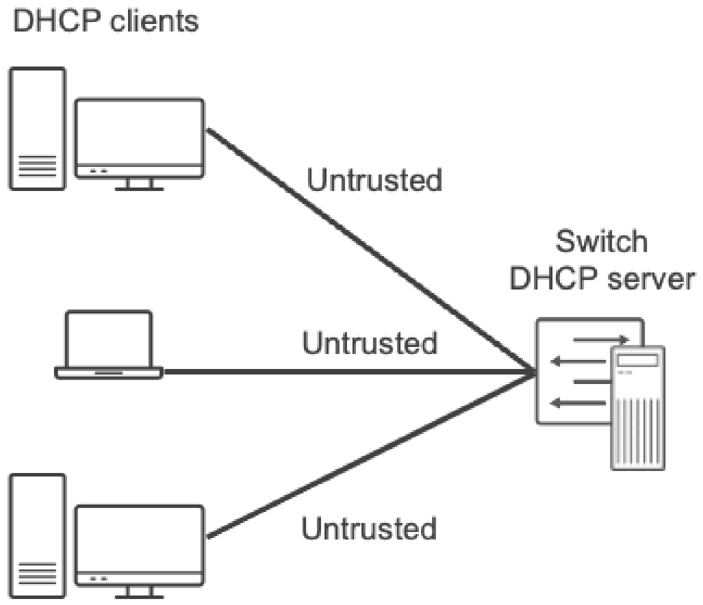
The following diagram depicts a setup where the server is linked directly to the same switch as the DHCP clients (devices seeking IP addresses from the server). In this configuration, DHCP snooping is activated on the VLAN to safeguard the non-trusted access ports, while the trunk port is inherently set as trusted.



The next diagram presents a slightly more complex arrangement. Here, the server is connected to a separate intermediary switch (Switch 2). This intermediary switch is then linked to Switch 1, to which the DHCP clients are connected, via a trunk port. In this setup, Switch 2 functions as a transit switch. Similar to the first setup, the VLAN on Switch 1 has DHCP snooping enabled to secure its non-trusted access ports, and its trunk port is by default labeled as trusted.



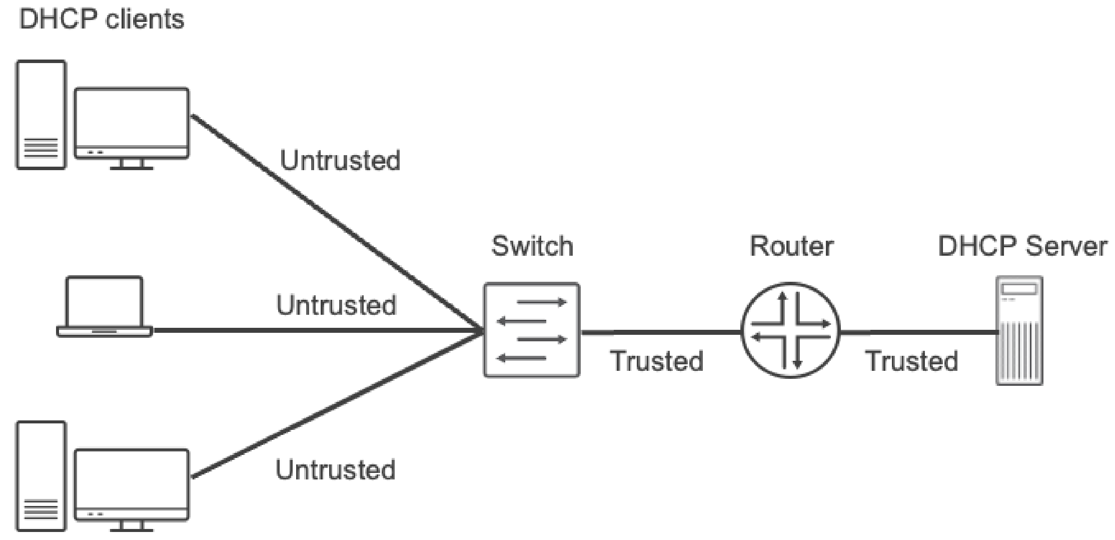
The switch can be set up to operate as an extended DHCP local server through specific configuration options. In the following, DHCP clients are connected to this extended DHCP local server via non-trusted access ports.



The following graphic shows a scenario of a DHCP server or its clients connect through a Layer 3 interface on a switch or router; the switch serves as a relay agent. These Layer 3 interfaces on the switch are set up as IRB interfaces. By default, trunk interfaces are considered trusted.

The switch plays the role of a relay agent in the following situations:

- The DHCP server and its clients are located in separate VLANs.
- The switch is linked to a router, which then connects to the DHCP server.



Understanding Dynamic ARP Inspection (DAI)

Dynamic ARP inspection (DAI) safeguards switches from ARP spoofing, also termed ARP poisoning.

DAI checks ARP packets within a LAN, utilizing the DHCP snooping database on the switch for validation. It compares ARP requests and responses to entries in this database, making decisions based on these comparisons. If an intruder attempts address spoofing with a counterfeit ARP packet, the switch contrasts the packet's address against the database entries. If the ARP packet's MAC or IP address doesn't coincide with a legitimate entry, the packet is discarded.

To transmit IP packets on a shared network, there's a need to associate an IP address with an Ethernet MAC address. This is achieved through ARP on Ethernet LANs. Switching devices hold this association in a cache consulted when directing packets. If there's no entry for the target device in the ARP cache, the host (being the DHCP client) broadcasts an ARP query and saves the response.

This method is a pathway for man-in-the-middle attacks. Here, an attacker dispatches an ARP packet, faking the MAC address of another LAN device. Consequently, instead of routing traffic to the right device, the switch sends it to the spoofed address. If this fake address belongs to the attacker, they seize all wrongly directed traffic. The outcome is misrouted traffic from the switch.

Gratuitous ARP is an ARP spoofing type, where a device seeks its own IP address. In normal scenarios, these messages suggest MAC address duplications. But they're also broadcast post any network interface card (NIC) change. Maliciously, attackers can reroute a device's traffic by sending deceptive ARP responses.

To combat MAC spoofing, switches scrutinize ARP replies with DAI.

- DAI assesses ARP exchanges on the LAN. It captures ARP packets at the access port, cross-referencing them with the DHCP snooping database. If there's no matching entry, the ARP packet gets discarded, preventing cache updates. Invalid IP packets are also dropped. Nonetheless, ARP probes bypass dynamic ARP inspection and are always relayed.
- In Junos OS for EX Series switches and the QFX Series, DAI is applied to ARP packets on access ports, as they're inherently untrusted. By contrast, trunk ports are trusted, so ARP packets evade DAI there.
- DAI is set per VLAN, not per port, and is deactivated by default. Setting a DHCP trusted port also trusts ARP packets on that port. ARP queries for the switching device are broadcast on its VLAN, with their responses undergoing DAI scrutiny.
- Regarding DAI, ARP packets are directed to the Packet Forwarding Engine. To avoid CPU strain, ARP packets for the Routing Engine are subject to rate limits.
- If the DHCP server becomes unavailable and an IP-MAC entry's lease for a valid ARP packet expires, the packet is obstructed.

Working with DAI

To enable DAI, you must turn it on for each VLAN as the following output shows.

```
[edit vlans vlan-name forwarding-options dhcp-security]
user@switch# set  arp-inspection
```

To verify that DAI is working, we can issue the `show dhcp-security arp inspection statistics` command, as the following output shows.

```
user@switch> show dhcp-security arp inspection statistics
Interface       Packets received  ARP inspection pass   ARP inspection fail
ge-0/0/1.0       7                 5                    2
ge-0/0/2.0      10                10                    0
ge-0/0/3.0      12                12                    0
```

The previous output shows the count of ARP packets, both received and examined, for each interface. It further details the number of packets that successfully underwent the inspection versus those that failed. The switch assesses ARP communications against the DHCP snooping database records. Should an ARP packet's MAC or IP address mismatch with a legitimate database entry, that packet gets discarded.

Working with IPv6 Neighbor Discovery Inspection

Nodes within an IPv6 environment use the Neighbor Discovery Protocol (NDP) to identify other nodes on the same link. While hosts deploy NDP to identify routers for packet forwarding, routers utilize it to broadcast their presence. Additionally, NDP helps maintain up-to-date path information to active neighbors, offering alternative paths if a router or its pathway breaks.

The foundation of the NDP method lies in the exchange of neighbor solicitation and advertisement messages. However, the lack of security in NDP messages leaves the messages vulnerable to spoofing attacks. In these attacks, a malicious node can redirect packets meant for legitimate nodes by either transmitting a neighbor solicitation with a faked source MAC or a neighbor advertisement with a counterfeit target MAC. Consequently, the false MAC address gets linked with an authentic IPv6 address by the receiving nodes.

To counteract NDP's vulnerabilities, IPv6 neighbor discovery inspection scrutinizes neighbor discovery messages, cross-referencing them with the DHCPv6 snooping table. Constructed by monitoring DHCPv6 communications, this table contains data like IPv6 address, MAC address, VLAN, and interface for every associated host within the VLAN. For packets received on untrusted interfaces, they're discarded unless their source details align with an entry in the DHCPv6 snooping table. This table can also be augmented with entries with the `static-ipv6` command.

NOTE: Neighbor discovery messages are always allowed on trusted interfaces.

Neighbor discovery inspection evaluates five distinct ICMPv6 messages: Router Solicitation, Router Advertisement, Neighbor Solicitation, Neighbor Advertisement, and Redirect. By filtering out unverifiable messages against the DHCPv6 snooping table, it prevents:

- **Cache Poisoning Attacks**: This is akin to ARP spoofing in IPv4. Attackers transmit unauthorized advertisements with their own MAC address linked to a legitimate IP. Once nodes update their neighbor caches with these malicious links, attackers can intercept traffic meant for genuine hosts.

- **Routing DoS Attacks**: By impersonating a router's address and sending a neighbor advertisement with a cleared router flag, attackers can mislead a host into believing its primary router is offline.

- **Redirect Attacks**: Typically, routers notify hosts of better routes using ICMPv6 redirect requests. Malicious actors can exploit this by diverting all traffic from the targeted host, similar to cache poisoning. The inspection ensures that only trusted routers send Router Redirect messages.

To turn on neighbor discovery inspection for a specific VLAN, you must use the neighbor-discovery-inspection statement, as the following output shows.

NOTE: DHCPv6 snooping is enabled automatically when neighbor discovery inspection is configured. There is no explicit configuration required for DHCPv6 snooping.

```
[edit vlans vlan-name forwarding-options dhcp-security]
user@switch# set neighbor-discovery-inspection
```

To verify that NDI is working, we can issue the `show dhcp-security neighbor-discovery-inspection statistics` command, as the following output shows.

```
user@switch> show dhcp-security neighbor-discovery-inspection statistics
Interface        ND Packets received  ND inspection pass   ND inspection fail
ge-0/0/1.0       7                    5                    2
ge-0/0/2.0       9                    9                    0
ge-0/0/3.0       9                    9                    0
```

Understanding IP Source Guard

Ethernet LAN switches can be targets of attacks where attackers spoof or falsify source IP or MAC addresses. The IP source guard is a security feature designed to combat these threats.

Access interface hosts can launch attacks by sending a flood of packets with counterfeit source IP and MAC addresses. When combined with methods like TCP SYN flood attacks, these can lead to denial-of-service (DoS) situations. In cases of IP or MAC address spoofing, pinpointing the source becomes a challenge for system admins. Attackers have the ability to spoof addresses within the same network or from different subnets.

IP source guard scrutinizes each packet originating from a host connected to an untrusted access interface. It validates the IP address, MAC address, VLAN, and associated interface of the host against the DHCP snooping database. Packets that don't align with a legitimate entry in this database are discarded, preventing their forwarding.

For VLANs, IP source guard specifically assesses packets coming from untrusted access interfaces. In general settings, access interfaces are labeled untrusted, while trunk interfaces are trusted. It's important to note that IP source guard does not evaluate packets from devices linked to trusted interfaces. This ensures a seamless connection for a DHCP server on such an interface to allocate dynamic IP addresses.

IP source guard derives its knowledge about the correlation between IP addresses and MAC addresses (known as IP-MAC binding) from the DHCP snooping table, which can be referred to as the DHCP binding table. This table gathers its data either dynamically via DHCP snooping or through manually configured static IP-to-MAC address links. To configure IP source guard, follow the below steps:

1. **VLAN Configuration**: IP source guard is set up on specific VLANs. Activating IP source guard on a VLAN prompts the switch to automatically turn on DHCP snooping for that VLAN.

2. **Support for IPv6**: For switches equipped with DHCPv6 snooping, there's also support for IPv6 source guard. Activating either IP source guard or IPv6 source guard on a VLAN will automatically kickstart both DHCP snooping and DHCPv6 snooping for that VLAN.

3. **VLAN Set-Up Requirement**: Before you can get IP source guard or IPv6 source guard running on a VLAN, it's essential that the VLAN is already set up. For guidance on this, refer to your switch's documentation that covers basic bridging and VLAN configuration.

4. **Interface Trust Levels**: The application of IP source guard and IPv6 source guard is restricted to untrusted interfaces. By default, access interfaces are tagged as untrusted.

5. **Compatibility with 802.1X Authentication**: IP source guard and IPv6 source guard can be paired with 802.1X user authentication. This can function in single supplicant, single-secure supplicant, or multiple supplicant modes.

```
[edit vlans vlan-name forwarding-options dhcp-security]
user@switch# set ip-source-guard
```

To enable IP source guard for a specific VLAN, you simply need to use the `ip-source-guard` statement, as the following output shows.

To enable IPv6 source guard for a specific VLAN, you simply need to use the `ipv6-source-guard` statement, as the following output shows.

```
[edit vlans vlan-name forwarding-options dhcp-security]
user@switch# set ipv6-source-guard
```

Understanding IPv6 Router Advertisement Guard

IPv6 Router Advertisement Guard Overview

In IPv6 networks, router advertisement (RA) messages are how routers announce their presence and provide configuration parameters to neighboring nodes. This system is vital for automatic IPv6 configuration. Unfortunately, RA messages are inherently unsecure, making them vulnerable to spoofing attacks and misconfigurations. RA guard is a security feature designed to mitigate these risks as it prevents rogue or misconfigured routers from disrupting your IPv6 network by sending malicious RAs.

RA guard works by filtering RA messages based on configurable policies. These policies can check factors such as:

- Source MAC address
- Source IPv6 address and prefix
- Hop-count limit
- Router preference priority
- Managed configuration flag
- Other configuration flag

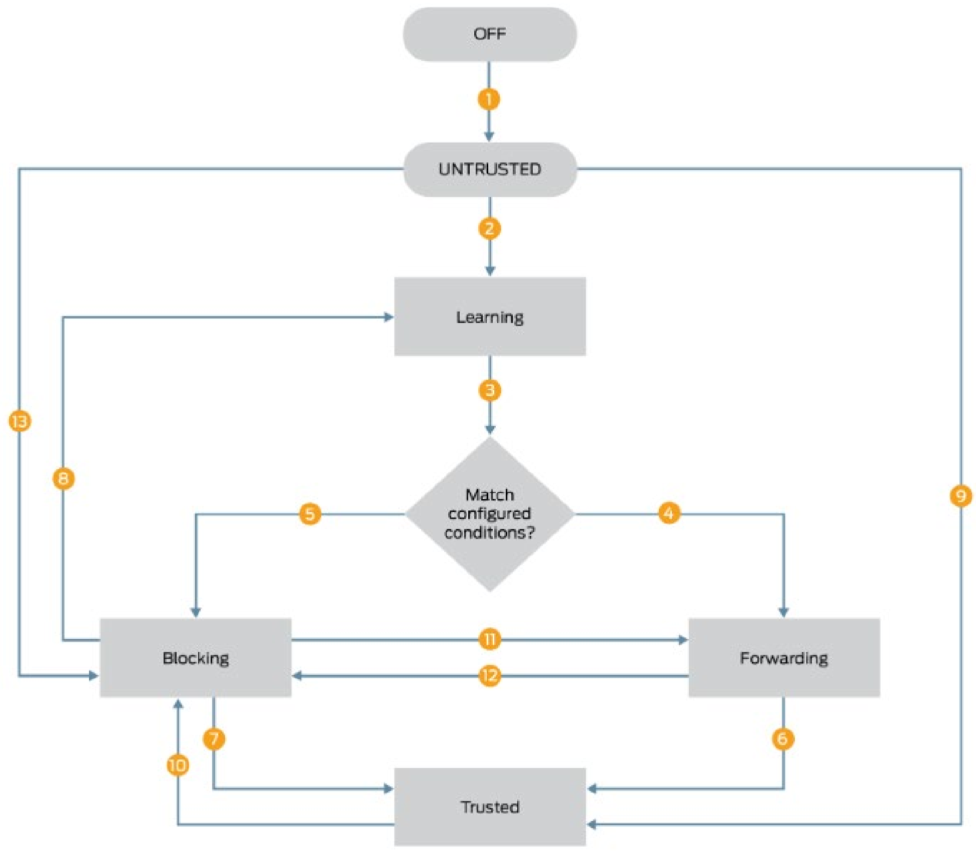
RA guard operates in either stateless or stateful mode. In stateless mode, the default state, RA messages received on each interface, are examined and filtered based on the attached policy. Interfaces can also be manually set to "trusted" (forward all RAs) or *blocked* (drop all RAs).

With the stateful mode, interfaces begin in a *learning* state, monitoring RA messages to establish a baseline of legitimate routers. After the learning period, valid sources transition to a *forwarding* state, while interfaces without valid RAs become *blocked*.

The following table summarizes the states of IPv6 RA guard in stateless and stateful modes.

State	Description	Mode
Off	The interface operates as if RA guard is not available	Stateless/Stateful
Untrusted	The interface forwards ingress RA messages if received RA messages are validated against the configured policy rules; otherwise, it drops the RA message. Untrusted state is the default state of an interface enabled for RA guard.	Stateless/Stateful
Blocked	The interface blocks ingress RA messages.	Stateless/Stateful
Forwarding	The interface forwards ingress RA messages if received RA messages are validated against the configured policy rules; otherwise, it drops the RA messages.	Stateful
Learning	The switch actively acquires information about the IPv6 routing device connected to the interface. The learning process takes place over a predefined period of time. At the end of the learning period, interfaces attached to legitimate senders of RA messages transition dynamically to the *forwarding* state, in which RA messages are forwarded if they can be validated against a policy. Interfaces that do not receive valid RA messages during the learning period transition dynamically to the *blocked* state, in which all ingress RA messages are dropped.	Stateful
Trusted	The interface forwards all RA messages directly, without validating them against the policy.	Stateless/Stateful

The following flow chart shows the transition of states when RA guard is enabled.



1. When RA guard is enabled on an interface, it moves to the untrusted state from the off state. The untrusted state is the default state of an interface that is enabled for RA guard.

2. When the command requesting the learning state is issued, the interface is moved from the off state to the learning state.

3. RA messages received during the learning state are compared to the configured policy.

4. If RA messages are validated against the configured policy, the interface moves to the forwarding state.

5. If RA messages are not validated against the configured policy, the interface moves to the blocked state.

6. If `mark-interface trust` is configured on the validated interface, then it moves from the forwarding state to the trusted state.

7. If `mark-interface trust` is configured on the blocked interface, then it moves from the blocked state to the trusted state.

8. If learning is requested on a blocked interface, then the interface moves from the blocked state to the learning state.

9. If an interface in the default untrusted state is configured as `mark-interface trust`, it moves directly to the trusted state. In this case a policy cannot be applied on that interface.

10. If the `mark-interface trust` configuration is deleted, and no valid RAs are received on the interface, then the interface moves to the blocked state.

11. If the command requesting the forwarding state is issued, then the interface moves directly from the blocked to the forwarding state.

12. If the command requesting the blocking state is issued, then the interface moves directly from the forwarding state to the blocked state.

13. If an interface in the default untrusted state is configured as `mark-interface block`, it moves directly to the blocked state. In this case, a policy cannot be applied to that interface.

Configuring Stateful IPv6 Router Advertisement Guard

Stateful IPv6 RA Guard provides enhanced security by intelligently filtering Router Advertisement (RA) messages. Here's how it works:

- Learning Phase: The switch begins in a learning state, monitoring RA messages on its interfaces. It builds a database of legitimate RA sources based on a configured policy.
- State Transitions:
    - Interfaces without valid RA sources during the learning phase are automatically transitioned to a blocking state, dropping all future RA messages.
    - Interfaces with confirmed valid RA sources move to a forwarding state, allowing RA messages that align with the policy.
- Manual Overrides: You can manually set an interface to blocking or forwarding states, overriding the automatic process.

Important Notes:

- Policies are Key: Before enabling stateful RA Guard, you must define a policy. This policy dictates what constitutes a valid RA message.
- Address Lists: If your policy checks source addresses or prefixes, you need to create an address list prior to configuring the policy.
    - Options
        - **source-mac-address-list** - Enables verification of the sender's MAC address in inspected messages from the access list.
        - **source-ip-address-list** - Enables verification of the sender's IPv6 address in inspected messages from the access list.
        - **prefix-list-name** - Enables verification of the advertised prefixes in inspected messages from the prefix list.

Configuring Stateful Interface IPv6 Router Advertisement Guard

You can enable stateful RA guard on an interface. First, configure a policy to validate incoming RA messages during the learning period. After you apply an RA guard policy to an interface, enable RA guard on the corresponding VLAN.

1. Configure a prefix list or a MAC list:

NOTE: The prefix list can be individual IPv6 address or IPv6 subnets.

```
[edit policy-options]
user@switch# set prefix-list <list-name> <prefixes-or-subnets>

[edit policy-options]
user@switch# set mac-list <list-name> <MAC-addresses>
```

2. Create the RA guard policy with the accept or discard action:

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> accept match-list source-mac-address-
list <MAC-list>

[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> accept match-list source-ip-address-
list <prefix-list>

[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> accept match-list prefix-list-name
<prefix-list>
```

OR

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> discard match-list source-mac-address-
list <MAC-list>

[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> discard match-list source-ip-address-
list <prefix-list>

[edit forwarding-options access-security router-advertisement-guard]
user@switch# set policy <policy-name> discard match-list prefix-list-name
<prefix-list>
```

NOTE: You can reference one list or more in the RA guard policy.

3. Apply a policy to an interface:

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set interface <interface-name> policy <policy-name>
```

4. Configure the stateful parameter on the interface:

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set interface <interface-name> policy <policy-name> stateful
```

5. You can enable stateful RA guard on a per-VLAN basis or for all VLANs. You must first configure a policy, which used to validate incoming RA messages during the learning state.

   To enable stateful RA guard for a specific VLAN:

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set vlans <VLAN-name> policy <policy-name> stateful
```

To enable stateful RA guard for all VLANs.

```
[edit forwarding-options access-security router-advertisement-guard]
user@switch# set vlan all policy <policy-name> stateful
```

NOTE: If a policy has been configured for a specific VLAN, that policy takes priority over the policy applied globally to all VLANs.

When stateful RA guard is first enabled, the default state is off. An interface in the off state operates as if RA guard is not available. To transition an interface to the learning state, you must request learning on the interface. An interface in the learning state actively acquires information from the RA messages that it receives.

To configure stateful RA guard learning on an interface:

1. Request learning on the interface with the learning period:

```
user@switch> request access-security router-advertisement-guard-learn
interface <interface-name> duration <seconds>
```

2. Configure the action to take on ingress RA messages received during the learning period. To forward RA messages received during the learning period, configure forwarding on the interface.
   a. To forward RA messages during the learning period:

```
user@switch> request access-security router-advertisement-guard-learn
interface <interface-name> duration <seconds> forward
```

   b. To block RA messages during the learning period:

```
user@switch> request access-security router-advertisement-guard-learn
```

```
interface <interface-name> duration <seconds> block
```

An interface in the forwarding state accepts ingress RA messages that can be validated against the configured policy and forwards them to their destination. An interface can dynamically transition to the forwarding state directly from the learning state, or the forwarding state can be statically configured on the interface.

```
user@switch> request access-security router-advertisement-guard-forward
interface <interface-name>
```

To configure the forwarding state on an interface:

An interface in the blocking state blocks ingress RA messages. An interface can dynamically transition to the blocking state directly from the learning state, or the blocking state can be statically configured on the interface. An interface that has been statically configured to be in the blocking state will remain in the blocking state until another state is configured on that interface.

To configure the blocking state on an interface:

```
user@switch> request access-security router-advertisement-guard-block
interface <interface-name>
```

Script level Allow-Transients

The `allow-transients` statement in Junos OS commit scripts enables transient configuration changes to be committed. By default, transient changes are disabled. If you omit the `allow-transients` statement and an enabled script generates transient changes, the CLI displays an error message, and the commit operation fails. To enable transient changes, include the following configuration:

```
[edit]
user@switch# set system scripts commit allow-transients
```

This setting allows for flexibility when handling configuration changes, especially in scenarios where temporary adjustments are necessary but should not be permanently committed. However, the `allow-transients` statement previously referenced is a global statement that is applied to all scripts on the Junos device. To address this problem, you can apply the `allow-transients` statement at the script level. For example:

```
[edit]
user@switch# set system scripts commit file <script-file-name> allow-
transients
```

NOTE: Transient changes are not persistent across reboots.

OpenConfig Interface Defaults

To ensure comprehensive configuration output, we must include attributes from the OpenConfig interfaces model with default values, even if only a non-default leaf is explicitly configured. SLAX scripts are used for this task.

If you configure the id leaf under the */interfaces/*interface/config container (and it has no default value), the SLAX script should also add configuration for any default-bearing leaves like loopback-mode, enabled, and TPID.

How It Works:

- *Default Source*: The */usr/libdata/cscript/import/openconfig-defaults.xml file* contains default values extracted from the OpenConfig model.
- *SLAX Logic*: The SLAX script reads this XML file and uses it to identify and add the configuration for default-bearing attributes whenever a related container is modified, ensuring complete output.

Supported Default OpenConfig Interface Module Values

XPath	Default Value
/interfaces/interface/config/loopback-mode	false
/interfaces/interface/config/enabled	true
/interfaces/interface/config/tpid	TPID_0X8100
/interfaces/interface/hold-time/config/up	0
/interfaces/interface/hold-time/config/down	0
/interfaces/interface/ethernet/config/auto-negotiate	true
/interfaces/interface/ethernet/config/enable-flow-control	false
/lacp/interfaces/interface/config/interval	SLOW
/lacp/interfaces/interface/config/lacp-mode	ACTIVE
/interfaces/interface/subinterfaces/subinterface/config/enabled	true
/interfaces/interface/subinterfaces/subinterface/ipv4/unnumbered/config/enabled	false
/interfaces/interface/subinterfaces/subinterface/ipv6/unnumbered/config/enabled	false

CHAPTER 13 EVPN-VXLAN Overlay Ping and Traceroute

Understanding Overlay Ping and Traceroute Functionality

Overlay networks, especially in virtualized environments, present challenges when attempting to diagnose connectivity issues. Traditional tools like ping and traceroute are limited in their capability to accurately report on the status of such networks.

In the realm of virtualized overlay networks, existing ping and traceroute methods fall short. While they can check connectivity between two endpoints in the physical network, they don't offer insights into the overlay network's state. To illustrate, imagine using a standard ping on a Juniper Networks device acting as a Virtual Tunnel Endpoint (VTEP) to ping another Juniper VTEP within a Virtual Extensible LAN (VXLAN) overlay. The ping might suggest a successful connection between the VTEPs even if an endpoint (like a physical server hosting applications) remains unreachable.

To make these tools functional in overlay networks:

- The ping and traceroute packets (commonly grouped as Operations, Administration, and Management or OAM packets) must be wrapped with the same VXLAN UDP headers as data packets in the overlay segment.
- This approach guarantees that transit nodes handle OAM packets just like they would a standard data packet within that overlay segment.
- Thus, if any data flow experiences connectivity issues, the corresponding overlay OAM packet will also encounter the same problems, providing a true reflection of the network's health.

Here are some considerations when you use overlay tools:

- The tools are designed exclusively for VXLAN tunnels.
- When deploying the overlay ping feature, ensure both the sending and receiving VTEPs in the overlay network are Juniper Networks devices that support the new ping and traceroute functions.

The mechanics of overlay ping and traceroute are described below:

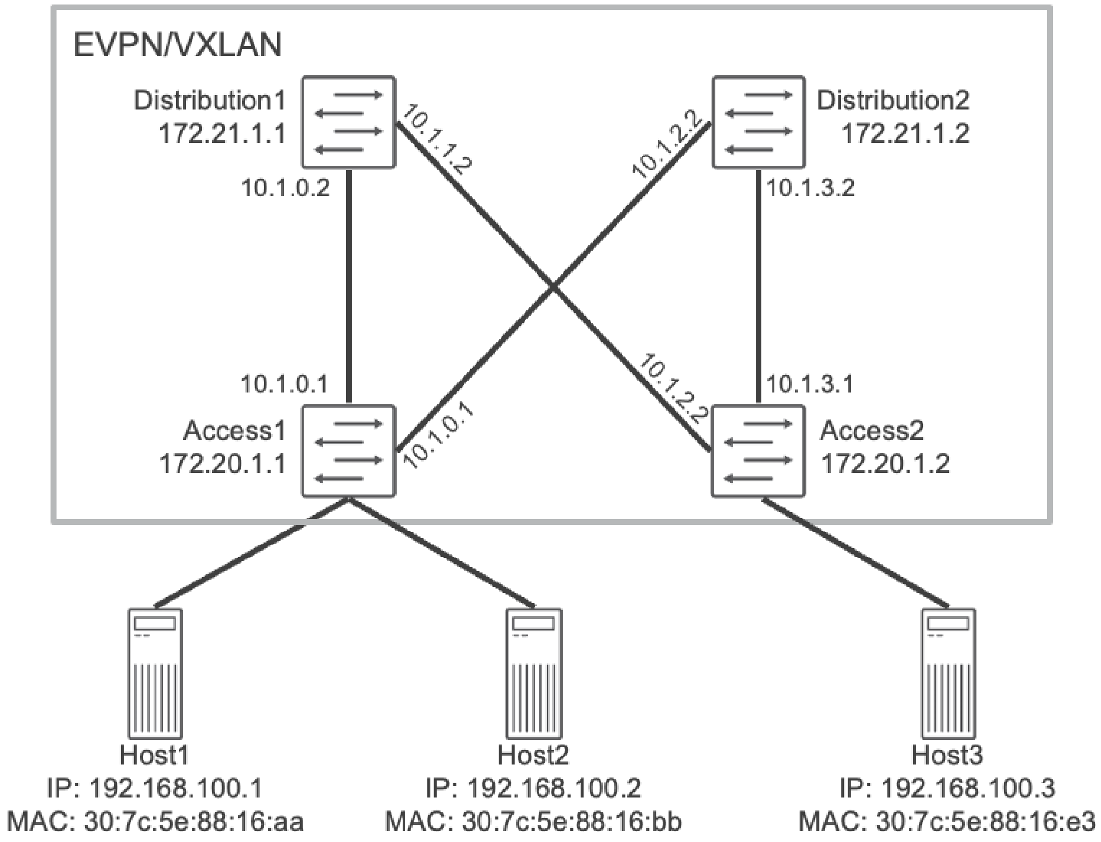
- **Encapsulation**: Overlay ping and traceroute packets are transported using the User Datagram Protocol (UDP). These are encapsulated within the VXLAN header.
- **VTEP Role**: Virtual Tunnel Endpoints (VTEPs) are the gateways of overlay tunnels. They take on the responsibility of sending and receiving overlay Operations, Administration, and Management (OAM) packets.
- **Device Compatibility**: The overlay ping and traceroute features function effectively only when both the sending and receiving VTEPs are Juniper Networks devices, and within the context of VXLAN overlay networks.

The features of overlay ping and traceroute are described below:

- **Enhanced Validation**: The overlay ping tool goes beyond just the data plane. It also verifies the MAC and IP addresses of the VTEPs. This is a departure from the traditional IP ping, where the target simply responds to the echo request without considering the overlay segment's specifics.
- **Route Tracing with Timestamps**: As the traceroute function maps out the path within the VXLAN overlay, Juniper devices that support the overlay traceroute feature also add timestamps to their responses. This is invaluable for diagnosing latency or understanding path dynamics. However, third-party devices, or even Juniper devices not equipped with overlay traceroute support, won't offer this timestamp data.

Using Overlay Ping and Traceroute

To ensure the functionality of ping and traceroute in a VXLAN overlay, these packets, often called Operations, Administration, and Management (OAM) packets, should be encapsulated with the same VXLAN UDP headers as the data packets in the potentially problematic VXLAN segment. If a connectivity issue does occur, both the overlay OAM packet and the data packet would be affected similarly. The following graphic shows an EVPN-VXLAN network that has three physical servers (Host1, Host2, and Host3) that have applications that run directly on them. The applications on Host1 and Host2 need to communicate with the applications on Host3. These servers are on the same subnet, so the communication between the applications occurs at the Layer 2 level, and VXLAN tunnels are used to transport their data packets over a Layer 3 network.



In this topology, there are two QFX5100 switches that function as VTEPs, which are Access1 and Access2. Access1 initiates and terminates VXLAN tunnels for Host1 and Host2, and Access2 does the same for Host3. Access1 and Access2 use VXLAN VNI 100.

In this VXLAN overlay network topology, a communication issue arises between Host1 and Host2. To troubleshoot the issue with this data flow, we can initiate the ping overlay and traceroute overlay commands on Access1 and specify Access2 as the destination VTEP.

Verifying That VXLAN 100 Is Configured on Access2

First, let's verify that a VXLAN with a VNI of 100 is configured on Access2. You can use either overlay ping or traceroute to check this. First, let's do an overlay ping.

```
user@Access1> ping overlay tunnel-type vxlan vni 100 tunnel-src 172.20.1.1 tunnel-
dst 172.20.1.2 count 5
ping-overlay protocol vxlan

        vni 100
        tunnel src ip 172.20.1.1
        tunnel dst ip 172.20.1.2
        mac address 00:00:00:00:00:00
        count 5
        ttl 255

                     WARNING: following hash-parameters are missing -
                hash computation may not succeed

                end-host smac
```

```
                 end-host dmac
                 end-host src ip
                 end-host dst ip
                 end-host vlan
                 end-host input interface
                 end-host protocol
                 end-host l4-src-port
                 end-host l4-dst-port

Request for seq 1, to 172.20.1.2, at 09-24 22:03:16 PDT.033 msecs

Response for seq 1, from 172.20.1.2, at 09-24 22:03:16 PDT.036 msecs, rtt 10 msecs

  Overlay-segment  not present at RVTEP 172.20.1.2


Request for seq 2, to 172.20.1.2, at 09-24 22:03:16 PDT.044 msecs

Response for seq 2, from 172.20.1.2, at 09-24 22:03:16 PDT.046 msecs, rtt 10 msecs

  Overlay-segment  not present at RVTEP 172.20.1.2


Request for seq 3, to 172.20.1.2, at 09-24 22:03:16 PDT.054 msecs

Response for seq 3, from 172.20.1.2, at 09-24 22:03:16 PDT.057 msecs, rtt 10 msecs

  Overlay-segment  not present at RVTEP 172.20.1.2


Request for seq 4, to 172.20.1.2, at 09-24 22:03:16 PDT.065 msecs

Response for seq 4, from 172.20.1.2, at 09-24 22:03:16 PDT.069 msecs, rtt 10 msecs

  Overlay-segment  not present at RVTEP 172.20.1.2

Request for seq 5, to 172.20.1.2, at 09-24 22:03:16 PDT.076 msecs

Response for seq 5, from 172.20.1.2, at 09-24 22:03:16 PDT.079 msecs, rtt 10 msecs

  Overlay-segment  not present at RVTEP 172.20.1.2
```

Next, let’s do an overlay traceroute.

```
user@Access1> traceroute overlay tunnel-type vxlan vni 100 tunnel-src 172.20.1.1 tunnel-dst
172.20.1.2
traceroute-overlay protocol vxlan

        vni 100
        tunnel src ip 172.20.1.1
        tunnel dst ip 172.20.1.2
        mac address 00:00:00:00:00:00
        ttl 255

                     WARNING: following hash-parameters are missing -
                hash computation may not succeed

                end-host smac
                end-host dmac
                end-host src ip
                end-host dst ip
                end-host vlan
                end-host input interface
                end-host protocol
                end-host l4-src-port
                end-host l4-dst-port

ttl   Address      Sender Timestamp               Receiver Timestamp           Response Time
  1   172.21.1.1   09-25 00:51:10 PDT.599 msecs             *                   10 msecs
  2   172.20.1.2   09-25 00:51:10 PDT.621 msecs   09-25 00:51:10 PDT.635 msecs   21 msecs

   Overlay-segment not present at RVTEP 172.20.1.2
```

The overlay ping output tells us the following:

- Access1 sent five ping requests to Access2, and Access2 responded to each request.
- Access2 indicated that the VNI of 100 is not configured (`Overlay-segment not present at RVTEP 172.20.1.2`) and included this information in its response to Access1.

The overlay traceroute output tells us the following:

- Upon receiving an overlay traceroute packet with a time-to-live (TTL) value of 1 hop, the Distribution1 device responds to Access1.
- Upon receiving an overlay traceroute packet with a TTL value of 2 hops, Access2 responds to Access1.
- Access2 indicates that the VNI of 100 is not configured (`Overlay-segment not present at RVTEP 172.20.1.2`) and included this information in its response to Access1.

NOTE: The asterisk (*) in the `Receiver Timestamp` column of the overlay traceroute output indicates that the Distribution1 devices that received the overlay traceroute packet is not a Juniper Networks device, or it is a Juniper Networks device that does not support overlay traceroute

Based on the results from the overlay ping and traceroute, it appears that VNI 100 is missing. To fix this problem, you must configure VNI 100 on Access2. To do this action, we must navigate to the `[edit vlans` *`vlan-id`* `vxlan]` section and apply the `vni 100` configuration statement. After making the changes, it is best to run the ping overlay and traceroute overlay commands again to confirm that VNI 100 is properly recognized.

Verifying That the MAC Address of the Destination Endpoint is on Access2

Next, we'll make sure that the MAC address (30:7c:53:88:16:cc) of Host3, the target endpoint, is listed in the forwarding table of Access2. For this verification, you can utilize either the overlay ping or traceroute command. Let's start by doing an overlay ping on Access1.

```
user@Access1> ping overlay tunnel-type vxlan vni 100 tunnel-src 172.20.1.1 tunnel-
dst 172.20.1.2 mac 30:7c:5e:88:16:cc count 5
ping-overlay protocol vxlan

        vni 100
        tunnel src ip 172.20.1.1
        tunnel dst ip 172.20.1.2
        mac address 30:7c:5e:88:16:cc
        count 5
        ttl 255

                     WARNING: following hash-parameters are missing -
                hash computation may not succeed

                end-host smac
                end-host dmac
                end-host src ip
                end-host dst ip
                end-host vlan
                end-host input interface
                end-host protocol
                end-host l4-src-port
                end-host l4-dst-port

Request for seq 1, to 172.20.1.2, at 09-24 23:53:54 PDT.089 msecs

Response for seq 1, from 172.20.1.2, at 09-24 23:53:54 PDT.089 msecs, rtt 6 msecs

  Overlay-segment present at RVTEP 172.20.1.2

      End-System Not Present


Request for seq 2, to 172.20.1.2, at 09-24 23:53:54 PDT.096 msecs

Response for seq 2, from 172.20.1.2, at 09-24 23:53:54 PDT.100 msecs, rtt 10 msecs

  Overlay-segment present at RVTEP 172.20.1.2

      End-System Not Present


Request for seq 3, to 172.20.1.2, at 09-24 23:53:54 PDT.107 msecs

Response for seq 3, from 172.20.1.2, at 09-24 23:53:54 PDT.111 msecs, rtt 10 msecs

  Overlay-segment present at RVTEP 172.20.1.2

      End-System Not Present

Request for seq 4, to 172.20.1.2, at 09-24 23:53:54 PDT.118 msecs

Response for seq 4, from 172.20.1.2, at 09-24 23:53:54 PDT.122 msecs, rtt 11 msecs

  Overlay-segment present at RVTEP 172.20.1.2

      End-System Not Present


Request for seq 5, to 172.20.1.2, at 09-24 23:53:54 PDT.129 msecs

Response for seq 5, from 172.20.1.2, at 09-24 23:53:54 PDT.133 msecs, rtt 10 msecs

  Overlay-segment present at RVTEP 172.20.1.2

      End-System Not Present
```

Next, let's do an overlay traceroute on Access1.

```
user@Access1> traceroute overlay tunnel-type vxlan vni 100 tunnel-src 172.25.1.1 tunnel-dst
172.25.1.2 mac 30:7c:5e:88:16:cc
traceroute-overlay protocol vxlan
```

```
        vni 100
        tunnel src ip 172.25.1.1
        tunnel dst ip 172.25.1.2
        mac address 30:7c:5e:88:16:cc
        ttl 255


                     WARNING: following hash-parameters are missing -
                hash computation may not succeed

                end-host smac
                end-host dmac
                end-host src ip
                end-host dst ip
                end-host vlan
                end-host input interface
                end-host protocol
                end-host l4-src-port
                end-host l4-dst-port

ttl  Address       Sender Timestamp   Receiver    Timestamp                    Response Time
  1   172.21.1.1   09-25 00:56:17 PDT.663 msecs     *                             10 msecs
  2   172.25.1.2   09-25 00:56:17 PDT.684 msecs   09-25 00:56:17 PDT.689 msecs    11 msecs

  Overlay-segment present at RVTEP 172.25.1.2

      End-System Not Present
```

The overlay ping output tells us the following:

- Access1 transmitted five ping requests to Access2, with Access2 providing a response to each.
- Access2 confirmed the configuration of VNI 100 (noted as `Overlay-segment present at RVTEP 172.25.1.2`). However, the MAC address for Host3 was absent from the forwarding table, denoted as `End-System Not Present`. This data was incorporated in Access2's response to Access1.

The overlay traceroute output tells us the following:

- When an overlay traceroute packet with a TTL value for 1 hop is received, the Layer 3 router sends a response to Access1.
- When a packet with a TTL value for 2 hops is received, Access2 offers a response to Access1.
- Similar to the ping output, Access2 confirmed the VNI 100 configuration (shown as `Overlay-segment present at RVTEP 172.25.1.2`), but noted that the MAC address Host3 wasn't in the forwarding table (`End-System Not Present`). This detail was incorporated in Access2's response to Access1.

Based on the results from both the overlay ping and traceroute, it's evident that Access2 does not recognize the MAC address of Host3. Consequently, a detailed examination is required to ascertain why this MAC address isn't listed in Access2's forwarding table.

Verifying the Data Flow

Ensure a smooth data transfer between Host1 and Host3. The networking components facilitating this transfer comprise Access1, Distribution1, and Access2.

Begin with the overlay ping for initial checks. Should any issues surface in the overlay ping findings, proceed with the overlay traceroute to pinpoint the exact segment encountering the problem.

For both overlay ping and traceroute, we need to use hash parameters to show information about the involved devices. Doing so enables the device to compute a VXLAN UDP header source port hash. This hash is integrated into the VXLAN UDP header for both overlay ping and traceroute packets. By including the computed hash in the VXLAN UDP header, these packets mimic the data packets in the flow, which aids in achieving more precise ping and traceroute outcomes. Let's start by doing an overlay ping on Access1.

NOTE: When using the hash parameters, we recommend specifying a value for each parameter. The exception to this guideline is the `hash-vlan` parameter, which you do not have to use if the source endpoint is not a member of a VLAN. This practice ensures that the overlay ping and traceroute processes are successful and that the output for each command is accurate. If you do not specify a value for one or more of the hash parameters, the system sends an OAM request that might include incorrect hash values and generates a warning message.

```
user@Access1> ping overlay tunnel-type vxlan vni 100 tunnel-src 172.25.1.1 tunnel-
dst 172.25.1.2 mac 30:7c:5e:88:16:cc count 5 hash-source-mac 30:7c:5e:88:16:aa hash-
destination-mac 30:7c:5e:88:16:cc hash-source-address 192.168.100.1 hash-
destination-address 192.168.100.3 hash-vlan 150 hash-input-interface xe-0/0/2 hash-
protocol 17 hash-source-port 4456 hash-destination-port 4540
ping-overlay protocol vxlan

        vni 100
        tunnel src ip 172.25.1.1
        tunnel dst ip 172.25.1.2
        mac address 30:7c:5e:88:16:cc
        count 5
        ttl 255

        hash-parameters:
                input-ifd-idx 653
                end-host smac 30:7c:5e:88:16:aa
                end-host dmac 30:7c:5e:88:16:cc
                end-host src ip 192.168.100.1
                end-host dst ip 192.168.100.3
                end-host protocol 17
                end-host l4-src-port 4456
                end-host l4-dst-port 4540
                end-host vlan 150

Request for seq 1, to 172.25.1.2, at 09-24 19:15:33 PDT.352 msecs

Request for seq 2, to 172.25.1.2, at 09-24 19:15:33 PDT.363 msecs

Request for seq 3, to 172.25.1.2, at 09-24 19:15:33 PDT.374 msecs

Request for seq 4, to 172.25.1.2, at 09-24 19:15:33 PDT.385 msecs

Request for seq 5, to 172.25.1.2, at 09-24 19:15:33 PDT.396 msecs
```

Next, let’s do an overlay traceroute on Access1.

```
user@Access1> traceroute overlay tunnel-type vxlan vni 100 tunnel-src 172.20.1.1 tunnel-
dst 192.0.2.20 mac 30:7c:5e:88:16:cc hash-source-mac 30:7c:5e:88:16:aa hash-destination-
mac 30:7c:5e:88:16:cc hash-source-address 192.168.100.1 hash-destination-address
192.168.100.3 hash-vlan 150 hash-input-interface xe-0/0/2 hash-protocol 17 hash-source-
port 4456 hash-destination-port 4540
traceroute-overlay protocol vxlan

        vni 100
        tunnel src ip 172.20.1.1
        tunnel dst ip 172.20.1.2
        mac address 30:7c:5e:88:16:cc
        ttl 255

        hash-parameters:
                input-ifd-idx 653
                end-host smac 30:7c:5e:88:16:aa
                end-host dmac 30:7c:5e:88:16:cc
                end-host src ip 192.168.100.1
                end-host dst ip 192.168.100.3
                end-host protocol 17
                end-host l4-src-port 4456
                end-host l4-dst-port 4540
                end-host vlan 150

ttl  Address       Sender Timestamp             Receiver       Timestamp     Response Time
  1   172.21.1.1   09-25 00:56:17 PDT.663 msecs      *                      10 msecs
```

The overlay ping output shows that Access1 transmitted five ping requests to Access2, yet Access2 did not return any responses. This absence of replies from Access2 suggests connectivity issues either on the route between Access1 and Distribution1 or between Distribution1 and Access2.

To pinpoint where the problem resides, the overlay traceroute tool is employed. The results from the overlay traceroute reveal:

- When the Layer 3 router receives an overlay traceroute packet with a TTL value set to 1 hop, it sends a response to Access1.
- This confirms the operational status of the connection between Access1 and the Layer 3 router.

Based on the overlay traceroute results, which suggest a connectivity problem between Distribution1 and Access2, it's essential to conduct a deeper examination of this segment to pinpoint the root cause of the disruption.

CHAPTER 14 Firewall Filter Flexible Match Conditions

Understanding Firewall Filter Flexible Match Conditions

Standard firewall filter match conditions vary based on the traffic's protocol family. For example, the terms available for bridge protocol traffic are different from those available for the inet or inet6 protocol families. The fields available for matching within each protocol family are, however, fixed or pre-defined. This means that filters can match on patterns within those pre-defined fields only.

Using flexible match conditions, firewall filters can be constructed that start the match at layer-2, layer-3, layer-4, or payload locations. From there, additional offset criteria can be specified thereby enabling pattern matches at custom, user-defined locations within a packet.

Flexible match filter terms are applied to interfaces as either input or output filters just as any other firewall filter terms. Flexible match filter terms can also be created as templates at the `[edit firewall]` hierarchy level. These templates can then be referenced within a flexible match term.

The `flexible-match` statement is configured at the `[edit firewall]` hierarchy level. It is used to define flexible match templates. `[edit firewall family [inet|inet6|bridge|ethernet-switching|ccc|vpls] filter <filter-name> term <term-name> from]` hierarchy.

Flexible match filter terms are constructed by giving a start location or anchor point within the packet. The start locations can be any of: layer-2, layer-3, layer-4, or payload, depending on the protocol family in use. You use these available start locations as the match-start locations for the flexible match filter terms. From these start locations, specific byte and bit offsets can be utilized to enable the filter to match patterns at very specific locations within the packet. These fields are called user-defined fields (UDFs). UDFs give you the ability to extract arbitrary bytes within the first 128 bytes of the packet. These extracted bytes can then be used as filtering fields in an ingress filter.

You can use UDFs with Layer 2 and Layer 3 headers for non-encapsulated packets. You can also use UDFs for tunnel-encapsulated Layer 3, UDP, VXLAN, and inner Layer 2 and Layer 3 headers. Tunnel-encapsulated packets are divided into multiple abstract packet types, and each abstract packet type will determine the base start offset, which determines the start point of the packet header from where UDF bytes can be extracted. Each packet category has a packet type, packet subtype and base offset. By providing the offset and the number of bytes to be extracted from the base start of the abstract packet type.

EVPN-VXLAN Examples of Firewall Filter Flexible Match Conditions

The following example matches on the inner source IP address of 10.1.1.1. To determine the prefix value we took the IP address (10.1.1.1) converted it to hex (0x0a010101), and then converted it to decimal (167837953).

Here's a breakdown of the terms:

- `match-start udp-vxlan`: Instructs the filter to look for packets encapsulated within the VXLAN protocol.
- `byte-offset 46`: Jumps 46 bytes into the packet.
- `bit-length 32`: Designates that the next 32 bits (4 bytes) should be considered for the match.
- `prefix 167837953`: The decimal value that the extracted MAC address must match against.

```
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask match-start udp-
vxlan
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask byte-offset 42
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask bit-length 32
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask prefix 167837953
```

The following example matches on the inner destination IP address of 10.10.10.1. To determine the prefix value we took the IP address (10.1.1.1) converted it to hex (0x0a0a0a01), and then converted it to decimal (168430081).

Here's a breakdown of the terms:

- `match-start udp-vxlan`: Instructs the filter to look for packets encapsulated within the VXLAN protocol.
- `byte-offset 46`: Jumps 46 bytes into the packet.
- `bit-length 32`: Designates that the next 32 bits (4 bytes) should be considered for the match.
- `prefix 168430081`: The decimal value that the extracted MAC address must match against.

```
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask match-start udp-
vxlan
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask byte-offset 46
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask bit-length 32
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask prefix 168430081
```

The following example matches on the inner source MAC address of 54:4b:8c:19:bf:80. To determine the prefix value we take the MAC address, remove the colons (544b8c19bf80) covert the hex value to decimal (92683449778048)

Here's a breakdown of the terms:

- `match-start udp-vxlan`: Instructs the filter to look for packets encapsulated within the VXLAN protocol.
- `byte-offset 22`: Jumps 22 bytes into the packet.
- `bit-length 32`: Designates that the next 32 bits (4 bytes) should be considered for the match.
- `prefix 92683449778048`: The decimal value that the extracted MAC address must match against.

```
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask match-start udp-
vxlan
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask byte-offset 22
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask bit-length 32
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask prefix
92683449778048
```

Here's a breakdown of the terms:

The following example matches on the inner destination MAC address of 54:4b:8c:19:bf:80. To determine the prefix value we take the MAC address, remove the colons (544b8c19bf80) covert the hex value to decimal (92683449778048)

- `match-start udp-vxlan`: Instructs the filter to look for packets encapsulated within the VXLAN protocol.
- `byte-offset 16`: Jumps 16 bytes into the packet.
- `bit-length 32`: Designates that the next 32 bits (4 bytes) should be considered for the match.

- `prefix 92683449778048`: The decimal value that the extracted MAC address must match against.

```
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask match-start udp-
vxlan
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask byte-offset 16
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask bit-length 32
set firewall family inet filter udf_vxlan term t1 from flexible-match-mask prefix
92683449778048
```

CHAPTER 15 Thresholds for Capacity Planning

Thresholds for Capacity Planning Overview

Hardware resources have finite capacities, and monitoring their utilization is crucial for preventing problems like traffic loss due to overload. This monitoring allows for proactive management – you can identify when resources are nearing their limits and take action before issues arise. By setting capacity thresholds, you can trigger notifications when resources are under strain, enabling effective management. There are two primary monitoring methods: periodic polling (where resources are checked at configured intervals) and on-change monitoring (where resources actively send updates when their utilization changes). Junos specifically supports periodic monitoring. Collected resource monitoring data can be integrated into familiar tools like the CLI or JTI for easy visualization.

You can monitor the following hardware resources:

HW Resource Name	Description	Associated Features
ecmp-group	Equal Cost Multi Path Group HW Table	Layer 3
ecmp-member	Equal Cost Multi Path Member HW Table	Layer 3
efp	TCAM: Egress Field Processor HW Table	Firewall
egress-l3-interface	Egress Layer 3 Interface HW Table	Layer 3
host-ipv4	IPv4 Host Routes HW Table	Layer 3
host-ipv6	IPv6 Host Routes HW Table	Layer 3
ifp	TCAM: Ingress Field Processor HW Table	Firewall
l3-next-hop	Layer 3 Next Hop HW Table	Layer 3
lpm-ipv4	Longest Prefix Match IPv4 32 bits wide HW Table	Layer 3
lpm-ipv6-128	Longest Prefix Match IPv6 128 bits wide HW Table	Layer 3
lpm-ipv6-64	Longest Prefix Match IPv6 64 bits wide HW Table	Layer 3
mac	Layer 2 Entry Table HW Table	Layer 2
mpls-ingress	Multiprotocol Label Switching Ingress HW Table	MPLS
mpls-swap	Multiprotocol Label Switching Swap HW Table	MPLS
tunnel	IP/Layer 3 tunnel HW Table	Layer 3
vfp	TCAM: VLAN Field Processor HW Table	Firewall
vport	Virtual Port HW Table	VXLAN

HW Resource Name	QFX5120-32C Max Entries	QFX5120-48T Max Entries	QFX5120-48Y Max Entries	QFX5120-48YM Max Entries	EX4650 Max Entries	EX4100-48MP Max Entries	EX4400-24MP Max Entries
ecmp-group	4096	4096	4096	4096	4096	256	1024
ecmp-member	32768	32768	32768	32768	32768	1024	4096
efp	2048	2048	2048	2048	2048	1024	1024
egress-l3-interface	16384	16384	16384	16384	16384	2048	8192
host-ipv4	Depends on UFT - 212992	Depends on UFT - 212992	Depends on UFT - 212992	Depends on UFT - 212992	Depends on UFT - 212992	Depends on UFT - 32768	Depends on UFT - 81920
host-ipv6	Depends on UFT - 106496	Depends on UFT - 106496	Depends on UFT - 106496	Depends on UFT - 106496	Depends on UFT - 106496	Depends on UFT - 16384	Depends on UFT - 40960
ifp	18432	18432	18432	18432	18432	Depends on UFT – 36864	18432
l3-next-hop	65536	65536	65536	65536	65536	32768	24576
lpm-ipv4	Depends on UFT - 175104	Depends on UFT - 175104	Depends on UFT - 175104	Depends on UFT - 175104	Depends on UFT - 175104	Depends on UFT – 28672	Depends on UFT – 64512
lpm-ipv6-128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT – 1024	Depends on UFT – 17408
lpm-ipv6-64	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT - 48128	Depends on UFT – 12288	Depends on UFT – 17408
mac	Depends on UFT – 288k	Depends on UFT – 288k	Depends on UFT – 288k	Depends on UFT – 288k	Depends on UFT – 288k	Depends on UFT – 64k	Depends on UFT – 112k
mpls-ingress	16384	16384	16384	16384	16384		
mpls-swap	16384	16384	16384	16384	16384		
tunnel	4096	4096	4096	4096	4096	1024	4096
vfp	1024	1024	1024	1024	1024	1024	2048
vport	16384	16384	16384	16384	16384	1024	7168

NOTE: Depends on UFT implies that the maximum entries of the hardware resource depend on the UFT profile that is configured on the box. In this table we are capturing the maximum entry out of all the possible UFT profiles.

Thresholds for Capacity Planning Configuration

To configure hardware resource monitoring, you can use the following command. You can configure a single resource list with multiple resources or you can monitor all of the resources using the `all-resources` statement.

```
set system packet-forwarding-options hw-resource-monitor resource-list <resource-list-name>
resource-names <l3-next-hop host-ipv4 efp> | all-resources
```

These CLI configuration parameters allow you to link configured resource lists with monitor profiles. Note that while a single monitor profile can be applied to multiple resource lists, a resource list can only be associated with one monitor profile at a time. Also, you must configure a monitor profile before mapping it to a resource list.

```
set system packet-forwarding-options hw-resource-monitor resource-list <resource-list-name>
monitor-profile <monitor-profile-name>
```

Optionally, you can configure the `lower-threshold` parameter. When the threshold value is reached, a minor notification is raised. The default value for this parameter is `50`.

```
set system packet-forwarding-options hw-resource-monitor monitor-profile <monitor-profile-
name> lower-threshold <1-100>
```

Optionally, you can configure the `upper-threshold` parameter. When the threshold value is reached, a major notification is raised. The default value for this parameter is `90`.

```
set system packet-forwarding-options hw-resource-monitor monitor-profile <monitor-profile-
name> upper-threshold <1-100>
```

Optionally, you can configure the `notification type` parameter. The default value for this parameter is `none`.

```
set system packet-forwarding-options hw-resource-monitor monitor-profile <monitor-profile-
name> notification type <syslog/alarms/none>
```

Optionally, you can configure polling interval for the applicable resources by using the `polling-interval` parameter. The default value of `1` second is applied only when thresholds are configured but no polling interval has been explicitly configured.

```
set system packet-forwarding-options hw-resource-monitor polling-interval <10-86400000
milliseconds>
```

The following output shows that two resources are monitored: `L3 Next Hop` and `Host table (IPv4)`. In the output, we can see the maximum capacity and current utilization for the system. We can also see that the `L3 Next Hop` resource has a low current utilization of 30%, which is below the lower threshold. However, the `Host table (IPv4)` has a current utilization of 70%, which is the upper threshold for the resource. This results in a minor and major alert being recorded in the syslog for this resource. Not alarms have been raised for the `L3 Next Hop` resource as it has met neither of the thresholds.

```
user@router> show system packet-forwarding-options hw-resource-monitor utilization-info

Resource Name    | Max Capacity | Current Utilization | Lower Threshold % | Upper Threshold % | Capacity/Health Band | Notification type
----------------------------------------------------------------------------------------------------------------------------------------
L3 Next Hop           12345              30%                  50%                80%                   Green              Alarm
Host table (IPv4)     28664              70%                  40%                70%                   Red                Syslog
```

CHAPTER 16 In-Band ZTP Management

In-Band ZTP Management Overview

Today, campus fabrics are managed by the cloud; we need to use each managed switch with an out-of-band interface and provide a separate switch infrastructure just to enable the fabric switches to connect to the cloud and then be managed from there. Typically, campus fabric has a hierarchical and multi-layered topology, where each layer of devices is spanned across the building. In a distributed campus environment where a single access switch may be dedicated to a floor, this is overkill, and the need to provide a separate infrastructure to manage peer switches in a closet is an idea that doesn't work.

Junos in-band ZTP is designed to solve the above problem and facilitates automated management of in-band devices. This solution works based on the event mechanism, and those events are part of any used case-specific policies. This solution facilitates newly connected devices to connect to the cloud by using existing telemetry data to detect the state and interface of the peer devices and apply event policies to the system. It helps campus fabric in the provisioning of a top-down approach, where *top* stands for upper layer devices and *down* stands for downstream devices, as each layer of devices that are provisioned automatically facilitates the provisioning of its downstream devices.

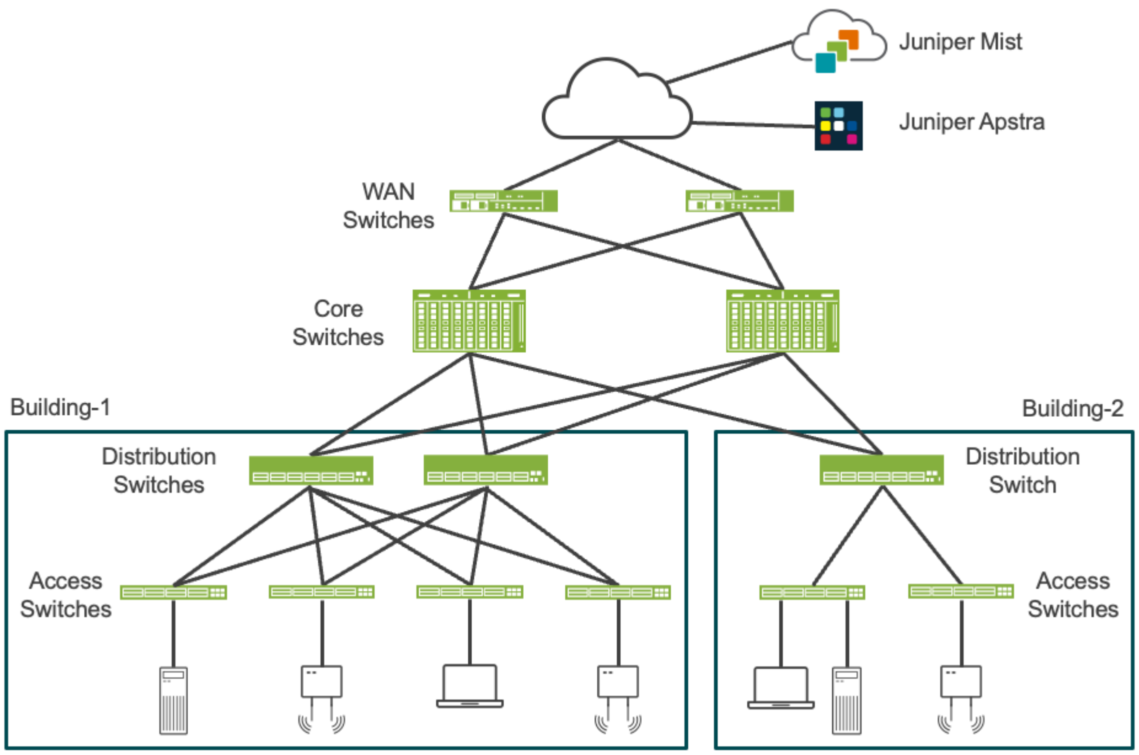
Traditional cloud-managed campus fabrics can be cumbersome. They often require a separate out-of-band network infrastructure just for management purposes. This adds complexity, especially in distributed campus environments where switches may be dedicated to individual floors. Junos in-band ZTP solves this problem by enabling in-band device management. It leverages existing telemetry data and event-driven policies to streamline provisioning and configuration. Here's how it works:

- *No More Separate Management Network*: Newly connected devices gain cloud access using the existing network infrastructure.
- *Top-Down Provisioning*: Junos in-band ZTP automates configuration in a hierarchical fashion. As each layer of devices is provisioned, it triggers the provisioning of its downstream devices.

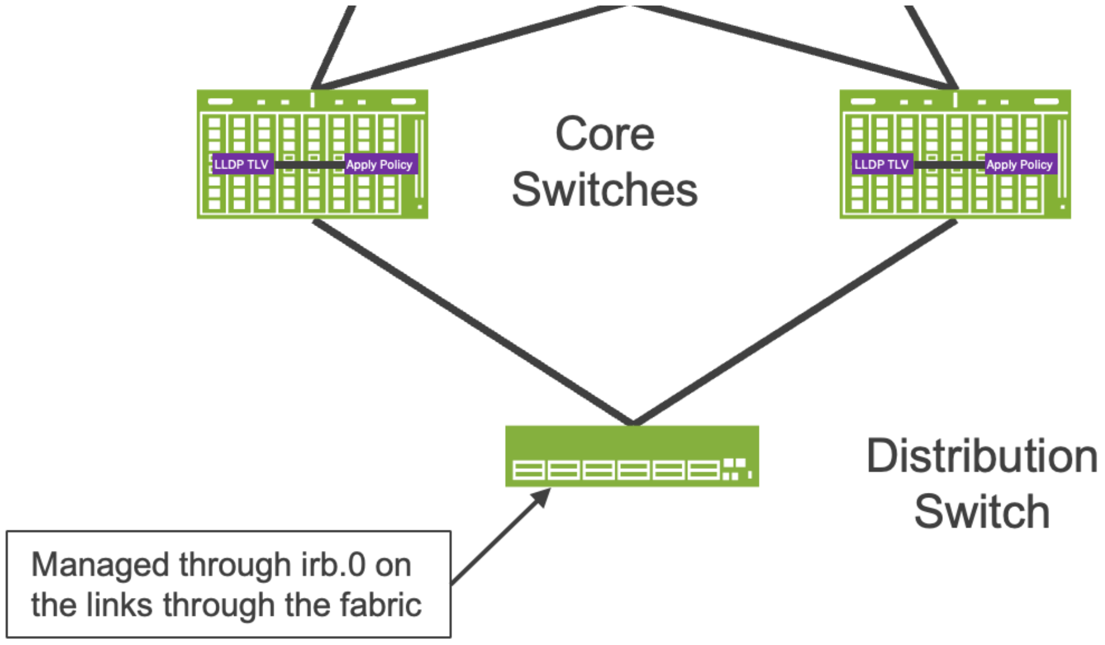
The Campus Fabric Network

A typical campus fabric network includes WAN routers, core, distribution, and access switches. A limited number of core switches, already provisioned and cloud-connected through the WAN routers, form the network's backbone. Juniper controllers (Mist or Apstra) run in the cloud.

Distribution switches are deployed with redundancy per building and are connected to the core. Access switches reside in closets on each building floor and are linked to the distribution layer. Finally, access points provide wireless coverage throughout the floors and are connected to the access switches.



The following topology displays two core switches, and then a new distribution switch is added.



When adding a new distribution switch, the following steps occur:

- Core switches are already provisioned and are connected to cloud services.

- Core switches have the set system services in-band-ztp configuration parameter to invoke the autonomous python daemon running which starts monitoring peers on down-stream interfaces with LLDP telemetry data (TLV JUNIPER CLOUD CONNECT).
- The distribution switch has the Day-0 or factory default configuration and is connected to core switches.
- The core switches detect the new distribution switch on the down-stream interfaces through telemetry data (LLDP TLV JUNIPER CLOUD CONNECT)
- The Junos autonomous python daemon converts dynamic LAG to static LAG if the downstream port is a Layer 3 aggregated Ethernet interface.
- The Junos autonomous Python daemon internally commits the local DHCP server static binding configuration to serve directly connected clients only:
    - The local DHCP server pool is for directly connected clients.
    - The local DHCP server pool is configured per interface.
    - The local DHCP server pool has a single IP address for the static binding.
    - The local DHCP server pool uses the local system’s DNS server details.
    - The local DHCP server pool uses the local interface network address as the gateway router IP.
- The distribution switch is able to get Layer 3 connectivity and on-boarded to the Juniper Mist cloud.
- The Juniper Mist cloud pushes any relevant Day-1 configuration to the distribution switch.

CHAPTER 17 Targeted Broadcasts and EVPN-VXLAN

Targeted Broadcasts Overview

Targeted broadcasts allow you to send Layer 3 broadcast packets from one subnet to a specific target subnet. This is useful for remote administration tasks like backups and Wake-on-LAN (WOL), and it even works with virtual routing and forwarding (VRF) instances. Targeted broadcasts work by:

- Routing: Targeted broadcast packets are routed like normal unicast packets until they reach the target subnet.
- On the Target Subnet: If targeted broadcast is enabled on the receiving router, the packets are either forwarded to the network interface (LAN), the routing engine, or both.
- Broadcast: The packets are then converted to standard broadcast packets and sent to all hosts on the target subnet.

Traditional broadcasts are limited to their originating subnet. Targeted broadcasts overcome this limitation, enabling remote administration across network segments. You can configure targeted broadcasts on the egress interface of a router or switch, with options to control whether packets are also forwarded to the routing engine.

In traditional environments, Layer 2 broadcast domains are isolated. This is beneficial for security and traffic management but poses a problem when you need to reach devices across subnets for tasks like Wake-on-LAN or remote administration. EVPN-VXLAN, while simplifying network configuration, maintains this broadcast isolation.

Targeted Broadcasts in EVPN-VXLAN: How It Works

Encapsulation: A node originating a targeted broadcast encapsulates the Layer 3 broadcast packet within a VXLAN frame. This frame includes specific EVPN control plane information (like the VNI) to identify the target network segment.

- Routing: The VXLAN encapsulated packet is routed like standard EVPN traffic, potentially traversing multiple network hops to reach the destination subnet.
- Destination Node: When the packet arrives at a VTEP (VXLAN Tunnel Endpoint) on the target subnet, it recognizes the targeted broadcast intent.
- Decapsulation & Handling: The VTEP decapsulates the packet, revealing the original Layer 3 broadcast. Now, the packet can either:
    - Be forwarded to the local network interface (LAN) for standard Layer 2 broadcasting within the subnet.
    - Be sent to the routing engine for potential further processing depending on your targeted broadcast configuration.

The EVPN control plane is crucial for distributing information about the target broadcast segment and enabling VTEPs to handle the encapsulated packets correctly. Also, targeted broadcast options on the originating device's egress interface will still apply (forward to interface only, or forward to the interface and the routing engine).

Targeted Broadcasts Configuration

Targeted broadcasts are configured for the egress interface with one of the following options:

- Option 1: *Forward and Send to Routing Engine*: Broadcast packets are sent out the network interface and a copy is forwarded to the Routing Engine for potential further processing.
- Option 2: *Forward to Interface Only*: Broadcast packets are sent exclusively to the network interface.

To configure targeted broadcast in an EVPN-VXLAN environment, you must configure a VLANs IRB interface with the `targeted-broadcast` statement, as shown in the following examples.

```
[edit interfaces irb]
user@Access1# set unit 100 family inet targeted-broadcast

[edit interfaces irb]
user@Access1# set unit 100 family inet targeted-broadcast forward-and-send-to-re

[edit interfaces irb]
user@Access1# set unit 100 family inet targeted-broadcast forward-only
```

CHAPTER 18 Dynamic VXLAN Members for 802.1X EVPN-VXLAN

Dynamic VXLAN Members for 802.1X EVPN-VXLAN Overview

When using the 802.1X protocol, access interfaces can be dynamically assigned to different bridge domains (BDs) based on RADIUS server authentication. This dynamic assignment is handled by the Layer 2 address learning daemon (l2ald), but the Routing Process Daemon (RPD) currently relies on static configuration for its interface-to-BD mapping.

RPD needs to know which Layer 2 interfaces are 'UP' within a BD or have an associated IRB interface to correctly generate EVPN Type-3 Inclusive Multicast Ethernet Tag Route. With 802.1X, interfaces might not have pre-configured associations with all EVPN VXLAN BDs. This lack of information hinders RPD's ability to create the necessary multicast routes.

To solve this, RPD needs the capability to dynamically learn interface-to-BD mappings. This will ensure it can accurately generate Type-3 IM routes for EVPN VXLAN extended BDs, even in environments with 802.1X authentication.

Consider the following scenario where the xe-0/0/13 interface has be set to the v200 VLAN through the configuration.

```
set interfaces xe-0/0/13 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/13 unit 0 family ethernet-switching vlan members v200
set vlans v500 vlan-id 500
set vlans v500 vxlan vni 500
set vlans v200 vlan-id 200
set vlans v200 vxlan vni 200
set protocols dot1x authenticator interface xe-0/0/13.0
```

Then, we have a scenario the interface-to-BD mapping changes and the xe-0/0/13 has its VLAN membership associated with VLAN v500 dynamically by 802.1X. In this process, Junos does not delete the v200 VLAN membership, and only adds the VLAN v500 membership. Then, a Type-3 IM route is advertised for VLANs v200 and v500.

In this scenario, you can also statically configure the dynamic VLAN (v500) on the xe-0/0/13 interface, which in turn deletes the dynamic interface-to-BD mapping. Here there is no change to the Type-3 IM route for VLAN v500, but the Type-3 IM route for v200 is removed.

```
delete interfaces xe-0/0/13 unit 0 family ethernet-switching vlan members vlan200
set interfaces xe-0/0/13 unit 0 family ethernet-switching vlan members vlan v500
```

Additional Dynamic VXLAN Members for 802.1X EVPN-VXLAN Scenarios

In another scenario, the RADIUS server authenticates xe-0/0/13 and changes the VLAN associated with the interface to v600, which is no part of the VXLAN extended BD. Here the dynamic interface-to-BD mapping (xe-0/0/13 and v600) is created, but is not part of the VXLAN extended BD. Because of this, there is no dynamic interface-to-BD mapping add message sent to the RPD.

We can also have coexistence to have the same statically configured and dynamic VLAN on the same interface. In this scenario, the xe-0/0/13 access interface is configured with VLAN v200 and 802.1X also dynamically maps it to VLAN v200. In this scenario, the configured VLAN takes precedence and a dynamic interface-to-BD message is not sent to the RPD. Removing VLAN configuration from an interface triggers a chain of events. First, Junos deletes the static interface-to-BD mapping and clears associated MAC addresses. Consequently, the RPD withdraws Type-3 IM routes if there aren't any remaining 'UP' Layer 2 interfaces or an IRB interface in that bridge domain. Simultaneously, the 802.1X protocol cleans up client sessions linked to the interface-to-BD. If traffic continues to flow, it prompts a new client authentication, which signals 802.1X to instruct Junos to create a dynamic interface-to-BD. This leads to Junos reestablishing the interface-to-BD, notifying RPD, and ultimately RPD regenerating the necessary Type-3 IM routes for the BD.

When a RADIUS server deauthenticates an interface or the 802.1X client disconnects, the response differs based on the 802.1X supplicant mode:

- *Single-Supplicant/Single-Secure*: This mode authenticates only the first connected user. If that user disconnects or their MAC address ages out, all learned MAC addresses on the bridge domain are flushed. The 802.1X protocol removes the dynamic VLAN association from the interface. Consequently, Junos deletes the dynamic interface-to-BD mapping, leading RPD to withdraw the Type-3 IM route for that BD (assuming no other Layer 2 interfaces or IRB interfaces are in the 'Up' state).
- *Multiple-Supplicant/Multiple-Secure*: Each user is authenticated individually. In this mode, the dynamic VLAN association remains as long as at least one authenticated user is present. Only when all users on the BD disconnect or their MACs are flushed does 802.1X trigger VLAN disassociation. This prompts Junos to delete the dynamic interface-to-BD mapping and, as before, results in RPD withdrawing the Type-3 IM route if no alternative Layer 2 or IRB interfaces remain active.

When a Change of Authentication (CoA) reassigns an interface's dynamic VLAN membership, the following occurs:

- *Single-Secure/Single-Supplicant:* Junos deletes the existing Interface-to-BD mapping and notifies the RPD. Junos then creates a new dynamic interface-to-BD mapping and informs RPD.
- *Multiple Supplicant*: Junos creates a new dynamic interface-to-BD mapping for the updated VLAN and sends the info to RPD.

In both scenarios, RPD generates a Type-3 IM route for the newly created dynamic interface-to-BD mappings.

APPENDIX A EVPN-VXLAN Configurations

Configuration of the Underlay IP Fabric

This section displays the configuration output for the IP Fabric underlay on the core, distribution, and access switches using BGP.

Core1 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces et-0/0/0 unit 0 family inet address 172.16.1.0/31
set interfaces et-0/0/35 unit 0 family inet address 172.16.1.2/31
```

2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.101.1/32
set routing-options router-id 192.168.101.1
```

3. Per-packet load-balancing.
```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64512
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.1 peer-as 64514
set protocols bgp group underlay neighbor 172.16.1.3 peer-as 64515
```

Core2 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces et-0/0/0 unit 0 family inet address 172.16.1.6/31
set interfaces et-0/0/35 unit 0 family inet address 172.16.1.4/31
```

2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.101.2/32
set routing-options router-id 192.168.101.2
```

3. Per-packet load-balancing.
```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64513
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.5 peer-as 64514
set protocols bgp group underlay neighbor 172.16.1.7 peer-as 64515
```

Dist1 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces et-0/0/48 unit 0 family inet address 172.16.1.1/31
set interfaces et-0/0/49 unit 0 family inet address 172.16.1.5/31
```
Access Interfaces:
```
set interfaces xe-0/0/1 unit 0 family inet address 172.16.1.8/31
set interfaces xe-0/0/3 unit 0 family inet address 172.16.1.10/31
```

2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.102.1/32
set routing-options router-id 192.168.102.1
```

3. Per-packet load-balancing.
```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two core switches and two access switches.
```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64514
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.0 peer-as 64512
set protocols bgp group underlay neighbor 172.16.1.4 peer-as 64513
set protocols bgp group underlay neighbor 172.16.1.9 peer-as 64516
set protocols bgp group underlay neighbor 172.16.1.11 peer-as 64517
```

Dist2 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces et-0/0/48 unit 0 family inet address 172.16.1.7/31
set interfaces et-0/0/49 unit 0 family inet address 172.16.1.3/31
```
Access Interfaces:
```
set interfaces xe-0/0/1 unit 0 family inet address 172.16.1.12/31
set interfaces xe-0/0/3 unit 0 family inet address 172.16.1.14/31
```

2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.102.2/32
set routing-options router-id 192.168.102.2
```

3. Per-packet load-balancing.
```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two core switches and two access switches.
```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64515
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.2 peer-as 64512
set protocols bgp group underlay neighbor 172.16.1.6 peer-as 64513
set protocols bgp group underlay neighbor 172.16.1.13 peer-as 64516
set protocols bgp group underlay neighbor 172.16.1.15 peer-as 64517
```

Access1 Configuration

1. Interconnects with the two distribution switches.

```
set interfaces xe-0/2/1 unit 0 family inet address 172.16.1.9/31
set interfaces xe-0/2/2 unit 0 family inet address 172.16.1.13/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.103.1/32
set routing-options router-id 192.168.103.1
```

3. Per-packet load-balancing.

```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two distribution switches.

```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64516
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.8 peer-as 64514
set protocols bgp group underlay neighbor 172.16.1.12 peer-as 64515
```

Access2 Configuration

1. Interconnects with the two distribution switches.

```
set interfaces xe-0/2/1 unit 0 family inet address 172.16.1.11/31
set interfaces xe-0/2/2 unit 0 family inet address 172.16.1.15/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.103.2/32
set routing-options router-id 192.168.103.2
```

3. Per-packet load-balancing.

```
set policy-options policy-statement load-balance term 1 then load-balance
per-packet
set routing-options forwarding-table export load-balance
```

4. BGP underlay network with the two distribution switches.

```
set protocols bgp group underlay type external
set protocols bgp group underlay export loopbacks
set protocols bgp group underlay local-as 64517
set protocols bgp group underlay multipath multiple-as
set protocols bgp group underlay neighbor 172.16.1.10 peer-as 64514
set protocols bgp group underlay neighbor 172.16.1.14 peer-as 64515
```

Configuration of the EVPN VXLAN Overlay and Virtual Networks

This section displays the configuration output for the EVPN VXLAN Overlay on the core, distribution, and access switches using BGP.

There are Layer 3 IRB interfaces on the Access layer (if the Routed at Distribution option was chosen during the initial phases of the campus fabric build, the Layer 3 IRB interfaces are on the distribution switches)

VXLAN tunneling is enabled, VLAN to VXLAN mapping, and MP BGP configuration snippets such as `vrf-targets` on the Access layer switches. The core switches have VXLAN tunneling and VLAN to VXLAN mapping.

Core1 Configuration

1. BGP overlay peering with the two distribution switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.101.1
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64512
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.2 peer-as 64513
set protocols bgp group overlay neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay neighbor 192.168.103.1 peer-as 64516
set protocols bgp group overlay neighbor 192.168.103.2 peer-as 64517
```

2. Switch options that define the VRF targets and the source loopback interface used for VXLAN.

```
set routing-instances evpn_vs vtep-source-interface lo0.0
set routing-instances evpn_vs route-distinguisher 192.168.101.1:1
set routing-instances evpn_vs vrf-target target:65512:1
```

3. VXLAN encapsulation.

```
set routing-instances evpn_vs protocols evpn encapsulation vxlan
set routing-instances evpn_vs protocols evpn extended-vni-list all
```

4. VRFs that are used for traffic isolation.

```
set routing-instances evpn_vs instance-type virtual-switch
set routing-instances evpn_vs protocols evpn encapsulation vxlan
set routing-instances evpn_vs protocols evpn extended-vni-list all
set routing-instances evpn_vs vtep-source-interface lo0.0
set routing-instances evpn_vs interface ae0.0
set routing-instances evpn_vs interface irb.33
set routing-instances evpn_vs interface irb.88
set routing-instances evpn_vs interface irb.99
set routing-instances evpn_vs route-distinguisher 192.168.101.1:1
set routing-instances evpn_vs vrf-target target:65512:1
```

5. VLAN to VXLAN mapping.

```
set routing-instances evpn_vs vlans v33 vlan-id 33
set routing-instances evpn_vs vlans v33 l3-interface irb.33
set routing-instances evpn_vs vlans v33 vxlan vni 5033
set routing-instances evpn_vs vlans v88 vlan-id 88
set routing-instances evpn_vs vlans v88 l3-interface irb.88
set routing-instances evpn_vs vlans v88 vxlan vni 5088
set routing-instances evpn_vs vlans v99 vlan-id 99
set routing-instances evpn_vs vlans v99 l3-interface irb.99
set routing-instances evpn_vs vlans v99 vxlan vni 5099
```

6. Layer 3 IRB interface enablement with anycast addressing.

```
set interfaces irb unit 33 family inet address 10.33.33.1/24
set interfaces irb unit 33 mac 02:05:86:8f:bd:00
set interfaces irb unit 88 family inet address 10.88.88.1/24
set interfaces irb unit 88 mac 02:05:86:8f:bd:00
set interfaces irb unit 99 family inet address 10.99.99.1/24
set interfaces irb unit 99 mac 02:05:86:8f:bd:00
```

Core2 Configuration

1. BGP overlay peering with the two distribution switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.101.2
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64513
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.1 peer-as 64512
set protocols bgp group overlay neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay neighbor 192.168.103.1 peer-as 64516
set protocols bgp group overlay neighbor 192.168.103.2 peer-as 64517
```

2. Switch options that define vrf-targets and the source loopback interface used for VXLAN.

```
set routing-instances evpn_vs vtep-source-interface lo0.0
set routing-instances evpn_vs route-distinguisher 192.168.101.2:1
set routing-instances evpn_vs vrf-target target:65512:1
```

3. VXLAN encapsulation.

```
Set routing-instances evpn_vs protocols evpn encapsulation vxlan
set routing-instances evpn_vs protocols evpn extended-vni-list all
```

4. VRFs that are used for traffic isolation.

```
set routing-instances evpn_vs instance-type virtual-switch
set routing-instances evpn_vs protocols evpn encapsulation vxlan
set routing-instances evpn_vs protocols evpn extended-vni-list all
set routing-instances evpn_vs vtep-source-interface lo0.0
set routing-instances evpn_vs interface ae0.0
set routing-instances evpn_vs interface irb.33
set routing-instances evpn_vs interface irb.88
set routing-instances evpn_vs interface irb.99
set routing-instances evpn_vs route-distinguisher 192.168.101.2:1
set routing-instances evpn_vs vrf-target target:65512:1
```

5. VLAN to VXLAN mapping.

```
set routing-instances evpn_vs vlans v33 vlan-id 33
set routing-instances evpn_vs vlans v33 l3-interface irb.33
set routing-instances evpn_vs vlans v33 vxlan vni 5033
set routing-instances evpn_vs vlans v88 vlan-id 88
set routing-instances evpn_vs vlans v88 l3-interface irb.88
set routing-instances evpn_vs vlans v88 vxlan vni 5088
set routing-instances evpn_vs vlans v99 vlan-id 99
set routing-instances evpn_vs vlans v99 l3-interface irb.99
set routing-instances evpn_vs vlans v99 vxlan vni 5099
```

6. Layer 3 IRB interface enablement with anycast addressing.

```
set interfaces irb unit 33 family inet address 10.33.33.1/24
set interfaces irb unit 33 mac 02:05:86:8f:bd:00
set interfaces irb unit 88 family inet address 10.88.88.1/24
set interfaces irb unit 88 mac 02:05:86:8f:bd:00
set interfaces irb unit 99 family inet address 10.99.99.1/24
set interfaces irb unit 99 mac 02:05:86:8f:bd:00
```

Dist1 Configuration

1. BGP Overlay peering with the two core switches and the two access switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.102.1
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64514
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.1 peer-as 64512
set protocols bgp group overlay neighbor 192.168.101.2 peer-as 64513
set protocols bgp group overlay neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay neighbor 192.168.103.1 peer-as 64516
set protocols bgp group overlay neighbor 192.168.103.2 peer-as 64517
```

Dist2 Configuration

1. BGP Overlay peering with the two core switches and the two access switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.102.2
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64515
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.1 peer-as 64512
set protocols bgp group overlay neighbor 192.168.101.2 peer-as 64513
set protocols bgp group overlay neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay neighbor 192.168.103.1 peer-as 64516
set protocols bgp group overlay neighbor 192.168.103.2 peer-as 64517
```

Access1 Configuration

1. BGP Overlay peering with the two distribution switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.103.1
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64516
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.1 peer-as 64512
set protocols bgp group overlay neighbor 192.168.101.2 peer-as 64513
set protocols bgp group overlay neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay neighbor 192.168.103.2 peer-as 64517
```

2. Switch options that define VRF targets and the source loopback interface used for VXLAN.

```
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 192.168.103.1:1
set switch-options vrf-target target:65512:1
```

3. VXLAN encapsulation.

```
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list all
```

4. VRFs that are used for traffic isolation..

```
set routing-instances v33 instance-type vrf
set routing-instances v33 routing-options static route 0.0.0.0/0 next-hop
10.33.33.254
set routing-instances v33 routing-options multipath
set routing-instances v33 interface irb.33
set routing-instances v33 route-distinguisher 192.168.103.1:33
set routing-instances v33 vrf-target target:65512:33
set routing-instances v33 vrf-table-label
set routing-instances v88 instance-type vrf
set routing-instances v88 routing-options static route 0.0.0.0/0 next-hop
10.88.88.254
set routing-instances v88 routing-options multipath
set routing-instances v88 interface irb.88
set routing-instances v88 route-distinguisher 192.168.103.1:88
set routing-instances v88 vrf-target target:65512:88
set routing-instances v88 vrf-table-label
set routing-instances v99 instance-type vrf
set routing-instances v99 routing-options static route 0.0.0.0/0 next-hop
10.99.99.254
set routing-instances v99 routing-options multipath
set routing-instances v99 interface irb.99
set routing-instances v99 route-distinguisher 192.168.103.1:99
set routing-instances v99 vrf-target target:65512:99
set routing-instances v99 vrf-table-label
```

5. VLAN to VXLAN mapping.

```
set vlans v33 vlan-id 33
set vlans v33 l3-interface irb.33
set vlans v33 vxlan vni 5033
set vlans v88 vlan-id 88
set vlans v88 l3-interface irb.88
set vlans v88 vxlan vni 5088
set vlans v99 vlan-id 99
set vlans v99 l3-interface irb.99
set vlans v99 vxlan vni 5099
```

6. Layer 3 IRB interface enablement with anycast addressing.

```
set interfaces irb unit 33 family inet address 10.33.33.1/24
set interfaces irb unit 33 mac 02:05:86:8f:bd:00
set interfaces irb unit 88 family inet address 10.88.88.1/24
set interfaces irb unit 88 mac 02:05:86:8f:bd:00
set interfaces irb unit 99 family inet address 10.99.99.1/24
set interfaces irb unit 99 mac 02:05:86:8f:bd:00
```

Access2 Configuration

1. BGP Overlay peering with the two distribution switches.

```
set protocols bgp group overlay type external
set protocols bgp group overlay multihop
set protocols bgp group overlay local-address 192.168.103.2
set protocols bgp group overlay family evpn signaling
set protocols bgp group overlay local-as 64517
set protocols bgp group overlay multipath multiple-as
set protocols bgp group overlay neighbor 192.168.101.1 peer-as 64512
set protocols bgp group overlay neighbor 192.168.101.2 peer-as 64513
set protocols bgp group overlay neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay neighbor 192.168.103.1 peer-as 64516
```

2. Switch options that define vrf-targets and the source loopback interface used for VXLAN.

```
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 192.168.103.2:1
set switch-options vrf-target target:65512:1
```

3. VXLAN encapsulation.

```
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list all
```

4. VRFs that are used for traffic isolation.

```
set routing-instances v33 instance-type vrf
set routing-instances v33 routing-options static route 0.0.0.0/0 next-hop
10.33.33.254
set routing-instances v33 routing-options multipath
set routing-instances v33 interface irb.33
set routing-instances v33 route-distinguisher 192.168.103.2:33
set routing-instances v33 vrf-target target:65512:33
set routing-instances v33 vrf-table-label
set routing-instances v88 instance-type vrf
set routing-instances v88 routing-options static route 0.0.0.0/0 next-hop
10.88.88.254
set routing-instances v88 routing-options multipath
set routing-instances v88 interface irb.88
set routing-instances v88 route-distinguisher 192.168.103.2:88
set routing-instances v88 vrf-target target:65512:88
set routing-instances v88 vrf-table-label
set routing-instances v99 instance-type vrf
set routing-instances v99 routing-options static route 0.0.0.0/0 next-hop
10.99.99.254
set routing-instances v99 routing-options multipath
set routing-instances v99 interface irb.99
set routing-instances v99 route-distinguisher 192.168.103.2:99
set routing-instances v99 vrf-target target:65512:99
set routing-instances v99 vrf-table-label
```

5. VLAN to VXLAN mapping.

```
set vlans v33 vlan-id 33
set vlans v33 l3-interface irb.33
set vlans v33 vxlan vni 5033
set vlans v88 vlan-id 88
set vlans v88 l3-interface irb.88
set vlans v88 vxlan vni 5088
set vlans v99 vlan-id 99
set vlans v99 l3-interface irb.99
set vlans v99 vxlan vni 5099
```

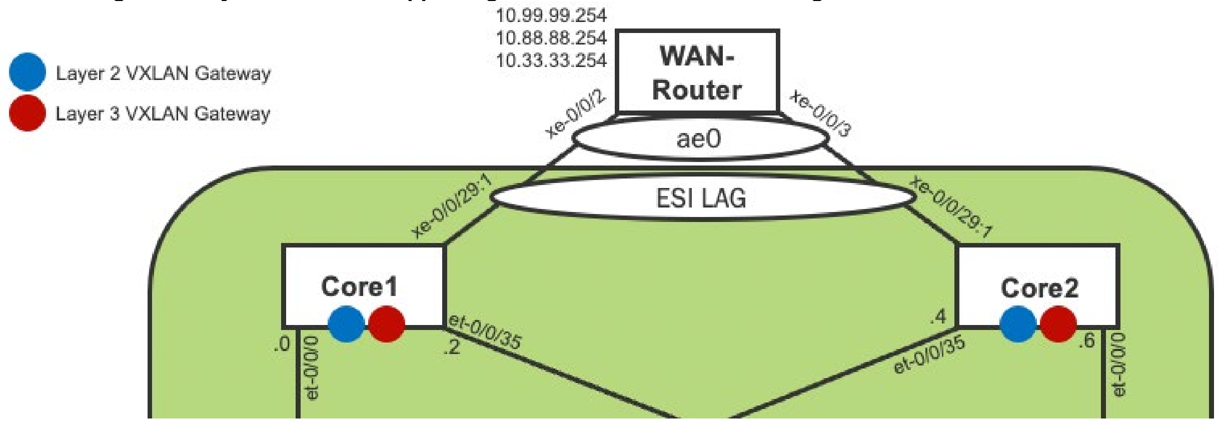
6. Layer 3 IRB interface enablement with anycast addressing.

```
set interfaces irb unit 33 family inet address 10.33.33.1/24
set interfaces irb unit 33 mac 02:05:86:8f:bd:00
set interfaces irb unit 88 family inet address 10.88.88.1/24
set interfaces irb unit 88 mac 02:05:86:8f:bd:00
set interfaces irb unit 99 family inet address 10.99.99.1/24
set interfaces irb unit 99 mac 02:05:86:8f:bd:00
```

Configuration of the Layer 2 ESI-LAG Between the Core Switches and SRX Series Firewall

This section displays the configuration output for the enablement of the Layer 2 ESI LAGs between the core switches and SRX Series Firewall. From the perspective of the SRX Series Firewall, the Ethernet bundle that is configured on the SRX Series Firewall views the ESI-LAG as a single MAC address with the same LACP system ID. This enables load hashing between the core switches and SRX Series Firewall without requiring L2 loop free detection protocols such as RSTP.

Figure 4: Layer 2 ESI-LAG Supporting Active-Active Load Balancing



Core1 Configuration

1. Interface association with the newly created Ethernet bundle that includes ESI and LACP configuration.

```
set interfaces xe-0/0/29:1 gigether-options 802.3ad ae0
set interfaces ae0 esi auto-derive lacp
set interfaces ae0 esi all-active
set interfaces ae0 aggregated-ether-options lacp active
set interfaces ae0 aggregated-ether-options lacp system-id 02:02:02:02:02:02
set interfaces ae0 unit 0 family ethernet-switching interface-mode trunk
set interfaces ae0 unit 0 family ethernet-switching vlan members v99
set interfaces ae0 unit 0 family ethernet-switching vlan members v88
set interfaces ae0 unit 0 family ethernet-switching vlan members v33
```

Core2 Configuration

1. Interface association with the newly created Ethernet bundle that includes ESI and LACP configuration.

```
set interfaces xe-0/0/29:1 gigether-options 802.3ad ae0
set interfaces ae0 esi auto-derive lacp
set interfaces ae0 esi all-active
set interfaces ae0 aggregated-ether-options lacp active
set interfaces ae0 aggregated-ether-options lacp system-id 02:02:02:02:02:02
set interfaces ae0 unit 0 family ethernet-switching interface-mode trunk
set interfaces ae0 unit 0 family ethernet-switching vlan members v99
set interfaces ae0 unit 0 family ethernet-switching vlan members v88
set interfaces ae0 unit 0 family ethernet-switching vlan members v33
```

SRX Series Firewall Configuration

1. Interface association with newly created Ethernet bundle and LACP configuration.

```
set interfaces xe-0/0/2 gigether-options 802.3ad ae0
set interfaces xe-0/0/3 gigether-options 802.3ad ae0
set interfaces ae0 flexible-vlan-tagging
set interfaces ae0 aggregated-ether-options lacp active
set interfaces ae0 unit 33 vlan-id 33
set interfaces ae0 unit 33 family inet address 10.33.33.254/24
set interfaces ae0 unit 88 vlan-id 88
set interfaces ae0 unit 88 family inet address 10.88.88.254/24
set interfaces ae0 unit 99 vlan-id 99
set interfaces ae0 unit 99 family inet address 10.99.99.254/24
```

APPENDIX B Seamless EVPN-VXLAN Type-2 Stitching

Configuration of the Underlay IP Fabric

This section displays the configuration output for the IP Fabric underlay on the core, distribution, and access switches using BGP.

Core1 Configuration

1. Interconnects with the four distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.3.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.3.5/31
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.9/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.13/31
```
2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.101.1/32
set routing-options router-id 192.168.101.1
```
3. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set policy-options policy-statement lb term 1 then accept
```
4. BGP underlay network with the four distribution switches.
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export loopbacks
set protocols bgp group underlay-core local-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.0 peer-as 64514
set protocols bgp group underlay-core neighbor 10.1.3.4 peer-as 64515
set protocols bgp group underlay-core neighbor 10.1.3.8 peer-as 64516
set protocols bgp group underlay-core neighbor 10.1.3.12 peer-as 64517
```
5. The loopbacks policy
```
set policy-options policy-statement loopbacks term 1 from interface lo0.0
set policy-options policy-statement loopbacks term 1 then community add core1
set policy-options policy-statement loopbacks term 1 then accept
```

Core2 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.3.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.3.7/31
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.11/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.15/31
```
2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.101.2/32
set routing-options router-id 192.168.101.2
```
3. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set policy-options policy-statement lb term 1 then accept
```
4. BGP underlay network with the four distribution switches.
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export loopbacks
set protocols bgp group underlay-core local-as 64513
set protocols bgp group underlay-core neighbor 10.1.3.2 peer-as 64514
set protocols bgp group underlay-core neighbor 10.1.3.6 peer-as 64515
set protocols bgp group underlay-core neighbor 10.1.3.10 peer-as 64516
set protocols bgp group underlay-core neighbor 10.1.3.14 peer-as 64517
```

5. The loopbacks policy

```
set policy-options policy-statement loopbacks term 1 from interface lo0.0
set policy-options policy-statement loopbacks term 1 then community add core1
set policy-options policy-statement loopbacks term 1 then accept
```

Dist1 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.0/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.2/31
```

Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.4/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.1/32
set routing-options router-id 192.168.102.1
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building1:
```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-leafdirect
set protocols bgp group underlay-bld-1 local-as 64514
set protocols bgp group underlay-bld-1 multipath multiple-as
set protocols bgp group underlay-bld-1 neighbor 10.1.1.0 peer-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.5 peer-as 64519
```

Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64514
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.1 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.3 peer-as 64513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist2 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.4/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.6/31
```

Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.7/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.2/32
set routing-options router-id 192.168.102.2
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building1:
```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-leafdirect
set protocols bgp group underlay-bld-1 local-as 64515
set protocols bgp group underlay-bld-1 multipath multiple-as
set protocols bgp group underlay-bld-1 neighbor 10.1.1.2 peer-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.6 peer-as 64519
```

Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64515
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.5 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.7 peer-as 6451364513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist3 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.8/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.10/31
```

Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.4/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.3/32
set routing-options router-id 192.168.102.3
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building2:
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-leafdirect
set protocols bgp group underlay-bld-2 local-as 64516
set protocols bgp group underlay-bld-2 multipath multiple-as
set protocols bgp group underlay-bld-2 neighbor 10.1.2.0 peer-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.5 peer-as 64521
```

Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64516
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.9 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.11 peer-as 64513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist4 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.12/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.14/31
```

Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.7/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.4/32
set routing-options router-id 192.168.102.3
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building2:
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-leafdirect
set protocols bgp group underlay-bld-2 local-as 64517
set protocols bgp group underlay-bld-2 multipath multiple-as
set protocols bgp group underlay-bld-2 neighbor 10.1.2.2 peer-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.6 peer-as 64521
```

Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64517
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.13 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.15 peer-as 64513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Access1 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.0/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.2/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1099
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.1/32
set routing-options router-id 192.168.103.1
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-directs
set protocols bgp group underlay-bld-1 local-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.1 peer-as 64514
set protocols bgp group underlay-bld-1 neighbor 10.1.1.3 peer-as 64515
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access2 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.5/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.6/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.2/32
set routing-options router-id 192.168.103.2
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-directs
set protocols bgp group underlay-bld-1 local-as 64519
set protocols bgp group underlay-bld-1 neighbor 10.1.1.4 peer-as 64514
set protocols bgp group underlay-bld-1 neighbor 10.1.1.7 peer-as 64515
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access3 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.0/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.2/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.3/32
set routing-options router-id 192.168.103.3
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-directs
set protocols bgp group underlay-bld-2 local-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.1 peer-as 64516
set protocols bgp group underlay-bld-2 neighbor 10.1.2.3 peer-as 64517
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access4 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.5/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.6/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.4/32
set routing-options router-id 192.168.103.4
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-directs
set protocols bgp group underlay-bld-2 local-as 64521
set protocols bgp group underlay-bld-2 neighbor 10.1.2.4 peer-as 64516
set protocols bgp group underlay-bld-2 neighbor 10.1.2.7 peer-as 64517
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Configuration of the EVPN VXLAN Overlay

This section displays the configuration output for the EVPN VXLAN Overlay on the distribution and access switches using BGP.

Dist1 Configuration

1. BGP Overlay peering with the two access switches.
```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.102.1
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 cluster 10.1.1.1
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath
set protocols bgp group overlay-bld-1 neighbor 192.168.103.1
set protocols bgp group overlay-bld-1 neighbor 192.168.103.2
set protocols bgp group overlay-bld-1 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.
```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.1
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64514
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.3 peer-as 64516
set protocols bgp group overlay-dci neighbor 192.168.102.4 peer-as 64517
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

Dist2 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.102.2
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 cluster 10.1.1.1
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath
set protocols bgp group overlay-bld-1 neighbor 192.168.103.1
set protocols bgp group overlay-bld-1 neighbor 192.168.103.2
set protocols bgp group overlay-bld-1 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.2
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64515
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.3 peer-as 64516
set protocols bgp group overlay-dci neighbor 192.168.102.4 peer-as 64517
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

Dist3 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-2 type internal
set protocols bgp group overlay-bld-2 multihop
set protocols bgp group overlay-bld-2 local-address 192.168.102.3
set protocols bgp group overlay-bld-2 family evpn signaling
set protocols bgp group overlay-bld-2 cluster 10.1.1.2
set protocols bgp group overlay-bld-2 local-as 65002
set protocols bgp group overlay-bld-2 multipath
set protocols bgp group overlay-bld-2 neighbor 192.168.103.4
set protocols bgp group overlay-bld-2 neighbor 192.168.103.3
set protocols bgp group overlay-bld-2 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.3
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64516
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay-dci neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

Dist4 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-2 type internal
set protocols bgp group overlay-bld-2 multihop
set protocols bgp group overlay-bld-2 local-address 192.168.102.4
set protocols bgp group overlay-bld-2 family evpn signaling
set protocols bgp group overlay-bld-2 cluster 10.1.1.2
set protocols bgp group overlay-bld-2 local-as 65002
set protocols bgp group overlay-bld-2 multipath
set protocols bgp group overlay-bld-2 neighbor 192.168.103.3
set protocols bgp group overlay-bld-2 neighbor 192.168.103.4
set protocols bgp group overlay-bld-2 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.4
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64517
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay-dci neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

Access1 Configuration

1. BGP Overlay peering with the two distribution switches in Building 1.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.1
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.1
set protocols bgp group overlay-bld-1 neighbor 192.168.102.2
```

Access2 Configuration

1. BGP Overlay peering with the two distribution switches in Building 1.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.2
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.1
set protocols bgp group overlay-bld-1 neighbor 192.168.102.2
```

Access3 Configuration

1. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.3
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65002
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.3
set protocols bgp group overlay-bld-1 neighbor 192.168.102.4
```

Access4 Configuration

1. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.4
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65002
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.3
set protocols bgp group overlay-bld-1 neighbor 192.168.102.4
```

Configuration of the MAC VRF Instances

This section displays the configuration output for the EVPN-VXLAN MAC VRF instances.

Distribution1 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.1:111
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:11:12:13:14:15:16:17:18:11
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
```

```
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.1:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Distribution2 Configuration

2. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.2:222
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:11:12:13:14:15:16:17:18:11
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.2:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Distribution3 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.3:333
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:21:22:23:24:25:26:27:28:22
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
```

```
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.3:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Distribution4 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.4:444
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:21:22:23:24:25:26:27:28:22
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.4:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Access1 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.1:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Access2 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.2:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Access3 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.3:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Access4 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.4:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

APPENDIX C Seamless EVPN-VXLAN Type-5 Stitching

Configuration of the Underlay IP Fabric

This section displays the configuration output for the IP Fabric underlay on the core, distribution, and access switches using BGP.

Core1 Configuration

1. Interconnects with the four distribution switches.

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.3.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.3.5/31
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.9/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.13/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.101.1/32
set routing-options router-id 192.168.101.1
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set policy-options policy-statement lb term 1 then accept
```

4. BGP underlay network with the four distribution switches.

```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export loopbacks
set protocols bgp group underlay-core local-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.0 peer-as 64514
set protocols bgp group underlay-core neighbor 10.1.3.4 peer-as 64515
set protocols bgp group underlay-core neighbor 10.1.3.8 peer-as 64516
set protocols bgp group underlay-core neighbor 10.1.3.12 peer-as 64517
```

5. The loopbacks policy

```
set policy-options policy-statement loopbacks term 1 from interface lo0.0
set policy-options policy-statement loopbacks term 1 then community add core1
set policy-options policy-statement loopbacks term 1 then accept
```

Core2 Configuration

1. Interconnects with the two distribution switches.

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.3.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.3.7/31
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.11/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.15/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.101.2/32
set routing-options router-id 192.168.101.2
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set policy-options policy-statement lb term 1 then accept
```

4. BGP underlay network with the four distribution switches.

```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export loopbacks
set protocols bgp group underlay-core local-as 64513
set protocols bgp group underlay-core neighbor 10.1.3.2 peer-as 64514
set protocols bgp group underlay-core neighbor 10.1.3.6 peer-as 64515
set protocols bgp group underlay-core neighbor 10.1.3.10 peer-as 64516
set protocols bgp group underlay-core neighbor 10.1.3.14 peer-as 64517
```

5. The loopbacks policy

```
set policy-options policy-statement loopbacks term 1 from interface lo0.0
set policy-options policy-statement loopbacks term 1 then community add core1
set policy-options policy-statement loopbacks term 1 then accept
```

Dist1 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:

```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.0/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.2/31
```

Access Interfaces:

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.4/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.1/32
set routing-options router-id 192.168.102.1
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building1:

```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-leafdirect
set protocols bgp group underlay-bld-1 local-as 64514
set protocols bgp group underlay-bld-1 multipath multiple-as
set protocols bgp group underlay-bld-1 neighbor 10.1.1.0 peer-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.5 peer-as 64519
```

Core:

```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64514
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.1 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.3 peer-as 64513
```

5. Export policies

Building1:

```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
```

```
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:

```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist2 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:

```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.4/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.6/31
```

Access Interfaces:

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.7/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.2/32
set routing-options router-id 192.168.102.2
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building1:

```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-leafdirect
set protocols bgp group underlay-bld-1 local-as 64515
set protocols bgp group underlay-bld-1 multipath multiple-as
set protocols bgp group underlay-bld-1 neighbor 10.1.1.2 peer-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.6 peer-as 64519
```

Core:

```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64515
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.5 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.7 peer-as 6451364513
```

5. Export policies

Building1:

```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:

```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist3 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.8/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.10/31
```
Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.1/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.4/31
```

2. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.102.3/32
set routing-options router-id 192.168.102.3
```

3. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building2:
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-leafdirect
set protocols bgp group underlay-bld-2 local-as 64516
set protocols bgp group underlay-bld-2 multipath multiple-as
set protocols bgp group underlay-bld-2 neighbor 10.1.2.0 peer-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.5 peer-as 64521
```
Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64516
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.9 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.11 peer-as 64513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```
Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Dist4 Configuration

1. Interconnects with the two core switches and the two access switches.

Core Interfaces:
```
set interfaces ge-0/0/3 unit 0 family inet address 10.1.3.12/31
set interfaces ge-0/0/4 unit 0 family inet address 10.1.3.14/31
```
Access Interfaces:
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.3/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.7/31
```

2. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.102.4/32
set routing-options router-id 192.168.102.3
```

3. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

4. BGP underlay network with the two core switches and two access switches.

Building2:
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-leafdirect
set protocols bgp group underlay-bld-2 local-as 64517
set protocols bgp group underlay-bld-2 multipath multiple-as
set protocols bgp group underlay-bld-2 neighbor 10.1.2.2 peer-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.6 peer-as 64521
```

Core:
```
set protocols bgp group underlay-core type external
set protocols bgp group underlay-core export export-directs
set protocols bgp group underlay-core local-as 64517
set protocols bgp group underlay-core multipath multiple-as
set protocols bgp group underlay-core neighbor 10.1.3.13 peer-as 64512
set protocols bgp group underlay-core neighbor 10.1.3.15 peer-as 64513
```

5. Export policies

Building1:
```
set policy-options policy-statement export-leafdirect term block-default from
route-filter 0.0.0.0/0 exact
set policy-options policy-statement export-leafdirect term block-default then
reject
set policy-options policy-statement export-leafdirect term term1 from as-
path-calc-length 1 orlower
set policy-options policy-statement export-leafdirect term term1 then accept
set policy-options policy-statement export-leafdirect term term2 then reject
```

Core:
```
set policy-options policy-statement export-directs term 1 from interface
lo0.0
set policy-options policy-statement export-directs term 1 then accept
set policy-options policy-statement export-directs term term2 then reject
```

Access1 Configuration

1. Interconnects with the two distribution switches.

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.0/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.2/31
```

2. Interface pointing to Desktop1.

```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1099
```

3. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.103.1/32
set routing-options router-id 192.168.103.1
```

4. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.

```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-directs
set protocols bgp group underlay-bld-1 local-as 64518
set protocols bgp group underlay-bld-1 neighbor 10.1.1.1 peer-as 64514
set protocols bgp group underlay-bld-1 neighbor 10.1.1.3 peer-as 64515
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access2 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.1.5/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.1.6/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.2/32
set routing-options router-id 192.168.103.2
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-1 type external
set protocols bgp group underlay-bld-1 export export-directs
set protocols bgp group underlay-bld-1 local-as 64519
set protocols bgp group underlay-bld-1 neighbor 10.1.1.4 peer-as 64514
set protocols bgp group underlay-bld-1 neighbor 10.1.1.7 peer-as 64515
```

6. The export policy.
```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access3 Configuration

1. Interconnects with the two distribution switches.
```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.0/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.2/31
```

2. Interface pointing to Desktop1.
```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.
```
set interfaces lo0 unit 0 family inet address 192.168.103.3/32
set routing-options router-id 192.168.103.3
```

4. Per-packet load-balancing.
```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.
```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-directs
set protocols bgp group underlay-bld-2 local-as 64520
set protocols bgp group underlay-bld-2 neighbor 10.1.2.1 peer-as 64516
set protocols bgp group underlay-bld-2 neighbor 10.1.2.3 peer-as 64517
```

6. The export policy.

```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Access4 Configuration

1. Interconnects with the two distribution switches.

```
set interfaces ge-0/0/1 unit 0 family inet address 10.1.2.5/31
set interfaces ge-0/0/2 unit 0 family inet address 10.1.2.6/31
```

2. Interface pointing to Desktop1.

```
set interfaces ge-0/0/3 unit 0 family ethernet-switching interface-mode trunk
set interfaces ge-0/0/3 unit 0 family ethernet-switching vlan members v1088
```

3. Loopback interface and router ID.

```
set interfaces lo0 unit 0 family inet address 192.168.103.4/32
set routing-options router-id 192.168.103.4
```

4. Per-packet load-balancing.

```
set policy-options policy-statement lb term 1 then load-balance per-flow
set routing-options forwarding-table export lb
```

5. BGP underlay network with the two distribution switches.

```
set protocols bgp group underlay-bld-2 type external
set protocols bgp group underlay-bld-2 export export-directs
set protocols bgp group underlay-bld-2 local-as 64521
set protocols bgp group underlay-bld-2 neighbor 10.1.2.4 peer-as 64516
set protocols bgp group underlay-bld-2 neighbor 10.1.2.7 peer-as 64517
```

6. The export policy.

```
set policy-options policy-statement export-directs term term1 from interface
lo0.0
set policy-options policy-statement export-directs term term1 then accept
```

Configuration of the EVPN VXLAN Overlay

This section displays the configuration output for the EVPN VXLAN Overlay on the distribution and access switches using BGP.

Core1 Configuration

1. BGP Overlay peering with the four distribution switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.101.1
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.1 export
from-bld2
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.2 export
from-bld2
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.3 export
from-bld1
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.4 export
from-bld1
```

```
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

2. The export policies.

```
set policy-options policy-statement from-bld1 term 1 from protocol aggregate
set policy-options policy-statement from-bld1 term 1 then accept
set policy-options policy-statement from-bld1 term 2 from route-filter
0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement from-bld1 term 2 then reject
set policy-options policy-statement from-bld2 term 1 from route-filter
10.88.88.0/24 orlonger
set policy-options policy-statement from-bld2 term 1 then accept
set policy-options policy-statement from-bld2 term 2 from protocol aggregate
set policy-options policy-statement from-bld2 term 2 then accept
set policy-options policy-statement from-bld2 term 3 from route-filter
0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement from-bld2 term 3 then reject
```

Core2 Configuration

1. BGP Overlay peering with the four distribution switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.101.2
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.1 export
from-bld2
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.2 export
from-bld2
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.3 export
from-bld1
set protocols bgp group overlay-dis-to-core neighbor 192.168.102.4 export
from-bld1
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

2. The export policies.

```
set policy-options policy-statement from-bld1 term 1 from protocol aggregate
set policy-options policy-statement from-bld1 term 1 then accept
set policy-options policy-statement from-bld1 term 2 from route-filter
0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement from-bld1 term 2 then reject
set policy-options policy-statement from-bld2 term 1 from route-filter
10.88.88.0/24 orlonger
set policy-options policy-statement from-bld2 term 1 then accept
set policy-options policy-statement from-bld2 term 2 from protocol aggregate
set policy-options policy-statement from-bld2 term 2 then accept
set policy-options policy-statement from-bld2 term 3 from route-filter
0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement from-bld2 term 3 then reject
```

Dist1 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.102.1
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 cluster 10.1.1.1
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath
set protocols bgp group overlay-bld-1 neighbor 192.168.103.1
set protocols bgp group overlay-bld-1 neighbor 192.168.103.2
set protocols bgp group overlay-bld-1 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.1
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64514
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.3 peer-as 64516
set protocols bgp group overlay-dci neighbor 192.168.102.4 peer-as 64517
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

4. BGP Overlay peering with the two core switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.102.1
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core export my-t5iDCI
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.1
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.2
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

5. BGP Overlay my-t5iDCI policy.

```
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
```

Dist2 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.102.2
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 cluster 10.1.1.1
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath
set protocols bgp group overlay-bld-1 neighbor 192.168.103.1
set protocols bgp group overlay-bld-1 neighbor 192.168.103.2
set protocols bgp group overlay-bld-1 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.2
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64515
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.3 peer-as 64516
set protocols bgp group overlay-dci neighbor 192.168.102.4 peer-as 64517
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

4. BGP Overlay peering with the two core switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.102.2
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core export my-t5iDCI
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.1
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.2
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

5. BGP Overlay my-t5iDCI policy.

```
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
```

Dist3 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-2 type internal
set protocols bgp group overlay-bld-2 multihop
set protocols bgp group overlay-bld-2 local-address 192.168.102.3
set protocols bgp group overlay-bld-2 family evpn signaling
set protocols bgp group overlay-bld-2 cluster 10.1.1.2
set protocols bgp group overlay-bld-2 local-as 65002
set protocols bgp group overlay-bld-2 multipath
set protocols bgp group overlay-bld-2 neighbor 192.168.103.4
set protocols bgp group overlay-bld-2 neighbor 192.168.103.3
set protocols bgp group overlay-bld-2 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.3
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64516
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay-dci neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

4. BGP Overlay peering with the two core switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.102.3
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core export my-t5iDCI
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.1
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.2
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

5. BGP Overlay my-t5iDCI policy.

```
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
```

Dist4 Configuration

1. BGP Overlay peering with the two access switches.

```
set protocols bgp group overlay-bld-2 type internal
set protocols bgp group overlay-bld-2 multihop
set protocols bgp group overlay-bld-2 local-address 192.168.102.4
set protocols bgp group overlay-bld-2 family evpn signaling
set protocols bgp group overlay-bld-2 cluster 10.1.1.2
set protocols bgp group overlay-bld-2 local-as 65002
set protocols bgp group overlay-bld-2 multipath
set protocols bgp group overlay-bld-2 neighbor 192.168.103.3
set protocols bgp group overlay-bld-2 neighbor 192.168.103.4
set protocols bgp group overlay-bld-2 vpn-apply-export
```

2. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-dci type external
set protocols bgp group overlay-dci multihop no-nexthop-change
set protocols bgp group overlay-dci local-address 192.168.102.4
set protocols bgp group overlay-dci family evpn signaling
set protocols bgp group overlay-dci export my-iDCI
set protocols bgp group overlay-dci local-as 64517
set protocols bgp group overlay-dci multipath multiple-as
set protocols bgp group overlay-dci neighbor 192.168.102.1 peer-as 64514
set protocols bgp group overlay-dci neighbor 192.168.102.2 peer-as 64515
set protocols bgp group overlay-dci vpn-apply-export
```

3. BGP Overlay my-iDCI policy.

```
set policy-options policy-statement my-iDCI term term1 from community iDCI
set policy-options policy-statement my-iDCI term term1 then accept
set policy-options policy-statement my-iDCI term term2 from family evpn
set policy-options policy-statement my-iDCI term term2 then reject
set policy-options community iDCI members target:1:123
```

4. BGP Overlay peering with the two core switches.

```
set protocols bgp group overlay-dis-to-core type internal
set protocols bgp group overlay-dis-to-core multihop no-nexthop-change
set protocols bgp group overlay-dis-to-core local-address 192.168.102.4
set protocols bgp group overlay-dis-to-core family evpn signaling
set protocols bgp group overlay-dis-to-core export my-t5iDCI
set protocols bgp group overlay-dis-to-core local-as 65002
set protocols bgp group overlay-dis-to-core multipath multiple-as
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.1
set protocols bgp group overlay-dis-to-core neighbor 192.168.101.2
set protocols bgp group overlay-dis-to-core vpn-apply-export
```

5. BGP Overlay my-t5iDCI policy.

```
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
```

Access1 Configuration

1. BGP Overlay peering with the two distribution switches in Building 1.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.1
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.1
set protocols bgp group overlay-bld-1 neighbor 192.168.102.2
```

Access2 Configuration

1. BGP Overlay peering with the two distribution switches in Building 1.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.2
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65001
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.1
set protocols bgp group overlay-bld-1 neighbor 192.168.102.2
```

Access3 Configuration

1. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.3
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65002
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.3
set protocols bgp group overlay-bld-1 neighbor 192.168.102.4
```

Access4 Configuration

1. BGP Overlay peering with the two distribution switches in Building 2.

```
set protocols bgp group overlay-bld-1 type internal
set protocols bgp group overlay-bld-1 multihop
set protocols bgp group overlay-bld-1 local-address 192.168.103.4
set protocols bgp group overlay-bld-1 family evpn signaling
set protocols bgp group overlay-bld-1 local-as 65002
set protocols bgp group overlay-bld-1 multipath multiple-as
set protocols bgp group overlay-bld-1 neighbor 192.168.102.3
set protocols bgp group overlay-bld-1 neighbor 192.168.102.4
```

Configuration of the MAC VRF Instances

This section displays the configuration output for the EVPN-VXLAN MAC VRF instances.

Distribution1 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.1:111
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:11:12:13:14:15:16:17:18:11
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.1:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Distribution2 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.2:222
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:11:12:13:14:15:16:17:18:11
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.2:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Distribution3 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.3:333
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:21:22:23:24:25:26:27:28:22
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.3:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Distribution4 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect vrf-target
target:1:123
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect route-
distinguisher 192.168.100.4:444
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi
00:21:22:23:24:25:26:27:28:22
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect esi all-active
set routing-instances EVPN-VXLAN-1 protocols evpn interconnect
interconnected-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.102.4:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Access1 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.1:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Access2 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.2:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
set routing-instances EVPN-VXLAN-1 vlans v1099 vlan-id 1099
set routing-instances EVPN-VXLAN-1 vlans v1099 l3-interface irb.1099
set routing-instances EVPN-VXLAN-1 vlans v1099 vxlan vni 5010
```

Access3 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.3:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Access4 Configuration

1. EVPN-VXLAN MAC VRF configuration

```
set routing-instances EVPN-VXLAN-1 instance-type mac-vrf
set routing-instances EVPN-VXLAN-1 protocols evpn encapsulation vxlan
set routing-instances EVPN-VXLAN-1 protocols evpn default-gateway no-gateway-
community
set routing-instances EVPN-VXLAN-1 protocols evpn extended-vni-list all
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5010 vrf-
target target:5010:1
set routing-instances EVPN-VXLAN-1 protocols evpn vni-options vni 5020 vrf-
target target:5020:1
set routing-instances EVPN-VXLAN-1 vtep-source-interface lo0.0
set routing-instances EVPN-VXLAN-1 service-type vlan-aware
set routing-instances EVPN-VXLAN-1 interface ge-0/0/3.0
set routing-instances EVPN-VXLAN-1 route-distinguisher 192.168.103.4:1
set routing-instances EVPN-VXLAN-1 vrf-target target:65000:1
set routing-instances EVPN-VXLAN-1 vrf-target auto
set routing-instances EVPN-VXLAN-1 vlans v1088 vlan-id 1088
set routing-instances EVPN-VXLAN-1 vlans v1088 l3-interface irb.1088
set routing-instances EVPN-VXLAN-1 vlans v1088 vxlan vni 5020
```

Configuration of the Type-5 VRF Instances

This section displays the configuration output for the EVPN-VXLAN MAC VRF instances.

Core1 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options aggregate route 10.88.88.0/24
set routing-instances T5 routing-options aggregate route 10.99.99.0/24
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.101.1:555
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 route-distinguisher 192.168.101.1:55
set routing-instances T5 vrf-target target:200:1
```

Core2 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options aggregate route 10.88.88.0/24
set routing-instances T5 routing-options aggregate route 10.99.99.0/24
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.101.1:555
```

```
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 route-distinguisher 192.168.101.1:55
set routing-instances T5 vrf-target target:200:1
```

Distribution1 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-import t5-import
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.102.1:555
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_CORE_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.102.1:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement t5-import term 1 from community t5-ic
set policy-options policy-statement t5-import term 1 then accept
set policy-options policy-statement t5-import term 2 then reject
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
set policy-options community t5-ic members target:200:1
```

3. IRB interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.5/24
set interfaces irb unit 1099 family inet address 10.99.99.5/24
```

Distribution2 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-import t5-import
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.102.2:555
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_CORE_ROUTES
```

```
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.102.2:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement t5-import term 1 from community t5-ic
set policy-options policy-statement t5-import term 1 then accept
set policy-options policy-statement t5-import term 2 then reject
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
set policy-options community t5-ic members target:200:1
```

3. IRB interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.6/24
set interfaces irb unit 1099 family inet address 10.99.99.6/24
```

Distribution3 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-import t5-import
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.102.3:555
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_CORE_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.102.3:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement t5-import term 1 from community t5-ic
set policy-options policy-statement t5-import term 1 then accept
set policy-options policy-statement t5-import term 2 then reject
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
set policy-options community t5-ic members target:200:1
```

3. IRB interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.7/24
set interfaces irb unit 1099 family inet address 10.99.99.7/24
```

Distribution4 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn interconnect vrf-import t5-import
set routing-instances T5 protocols evpn interconnect vrf-target target:300:1
set routing-instances T5 protocols evpn interconnect route-distinguisher
192.168.102.4:555
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_CORE_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.102.4:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement t5-import term 1 from community t5-ic
set policy-options policy-statement t5-import term 1 then accept
set policy-options policy-statement t5-import term 2 then reject
set policy-options policy-statement my-t5iDCI term 1 from community t5-comm
set policy-options policy-statement my-t5iDCI term 1 then accept
set policy-options policy-statement my-t5iDCI term 2 from family evpn
set policy-options policy-statement my-t5iDCI term 2 then reject
set policy-options community t5-comm members target:300:1
set policy-options community t5-ic members target:200:1
```

3. IRB interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.8/24
set interfaces irb unit 1099 family inet address 10.99.99.8/24
```

Access1 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_HOST_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.103.1:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
protocol evpn
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
route-filter 0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 then
accept
```

3. IRB anycast interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.1/24
set interfaces irb unit 1088 mac 50:04:00:04:07:f1
set interfaces irb unit 1099 family inet address 10.99.99.1/24
set interfaces irb unit 1099 mac 50:04:00:04:07:f0
```

Access2 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_HOST_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 interface irb.1099
set routing-instances T5 route-distinguisher 192.168.103.2:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
protocol evpn
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
route-filter 0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 then
accept
```

3. IRB anycast interfaces

```
set interfaces irb unit 1088 family inet address 10.88.88.1/24
set interfaces irb unit 1088 mac 50:04:00:04:07:f1
set interfaces irb unit 1099 family inet address 10.99.99.1/24
set interfaces irb unit 1099 mac 50:04:00:04:07:f0
```

Access3 Configuration

1. EVPN-VXLAN Type-5 VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_HOST_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 route-distinguisher 192.168.103.2:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
protocol evpn
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
route-filter 0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 then
accept
```

3. IRB anycast interface

```
set interfaces irb unit 1088 family inet address 10.88.88.1/24
set interfaces irb unit 1088 mac 50:04:00:04:07:f1
```

Access4 Configuration

1. EVPN-VXLAN Type VRF configuration

```
set routing-instances T5 instance-type vrf
set routing-instances T5 routing-options multipath
set routing-instances T5 routing-options auto-export family inet unicast
set routing-instances T5 protocols evpn ip-prefix-routes advertise direct-
nexthop
set routing-instances T5 protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances T5 protocols evpn ip-prefix-routes vni 50001
set routing-instances T5 protocols evpn ip-prefix-routes export
EXPORT_HOST_ROUTES
set routing-instances T5 vtep-source-interface lo0.0
set routing-instances T5 interface irb.1088
set routing-instances T5 route-distinguisher 192.168.103.4:55
set routing-instances T5 vrf-target target:100:556
```

2. Routing policies

```
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
protocol evpn
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 from
route-filter 0.0.0.0/0 prefix-length-range /32-/32
set policy-options policy-statement EXPORT_HOST_ROUTES term TERM_1 then
accept
```

3. IRB anycast interface

```
set interfaces irb unit 1088 family inet address 10.88.88.1/24
set interfaces irb unit 1088 mac 50:04:00:04:07:f1
```