Nokia Service Router Linux

Release 24.3

EVPN-VXLAN Guide

3HE 20241 AAAA TQZZA
Edition: 01
March 2024

Table of contents

1 About this guide

This document describes basic configuration for EVPN Layer 2 and Layer 3 functionality on the Nokia Service Router Linux (SR Linux). It presents examples to configure and implement various protocols and services.

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how the router is configured.

Note:
This manual covers the current release and may also contain some content to be released in later maintenance loads. See the *SR Linux Release Notes* for information about features supported in each load.

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

1.1 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.2 Conventions

Nokia SR Linux documentation uses the following command conventions.

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- An open right-angle bracket indicates a progression of menu choices or simple command sequence (often selected from a user interface). Example: **start** > **connect to**.

- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([ ]) indicate optional elements.
- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.
- *Italic* type indicates a variable.

Generic IP addresses are used in examples. Replace these with the appropriate IP addresses used in the system.

2 What's new

Topic	Location
Support for route internal tags	Route internal tags (tag-sets in routing policies)

3 Overview

This chapter contains the following topics:

3.1 About EVPN

Ethernet Virtual Private Network (EVPN) is a technology that allows Layer 2 traffic to be bridged across an IP network. EVPN instances configured on Provider Edge (PE) routers function as virtual bridges, transporting traffic between Customer Edge (CE) devices at separate locations.

At a basic level, the PE routers exchange information about reachability, encapsulate Layer 2 traffic from CE devices, and forward it across the Layer 3 network. EVPN is a standard technology in multitenant data centers (DCs).

VXLAN is a means for segmenting a LAN at a scale required by service providers. With the prevalent use of VXLAN in multitenant DCs, the EVPN control plane was adapted for VXLAN tunnels in RFC 8365.

The SR Linux EVPN-VXLAN solution supports using Layer 2 Broadcast Domains (BDs) in multitenant DCs using EVPN for the control plane and VXLAN as the data plane.

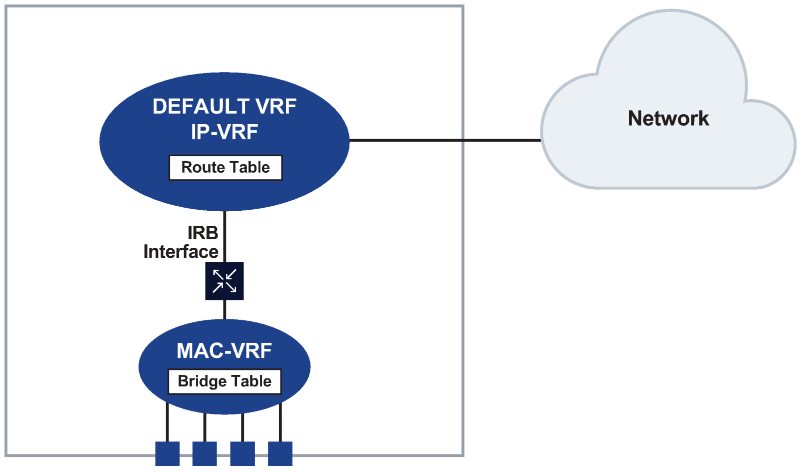
3.2 About Layer 2 services

Layer 2 services refers to the infrastructure implemented on SR Linux to support tunneling of Layer 2 traffic across an IP network, overlaying the Layer 2 network on top of the IP network.

To support this infrastructure, SR Linux uses a network instance of type MAC-VRF. The MAC-VRF network instance is associated with a network instance of type default or ip-vrf via an Integrated Routing and Bridging (IRB) interface.

The following figure shows the relationship between an IRB interface and MAC-VRF, and IP-VRF network-instance types.

Figure 1: MAC-VRF, IRB interface, and IP-VRF

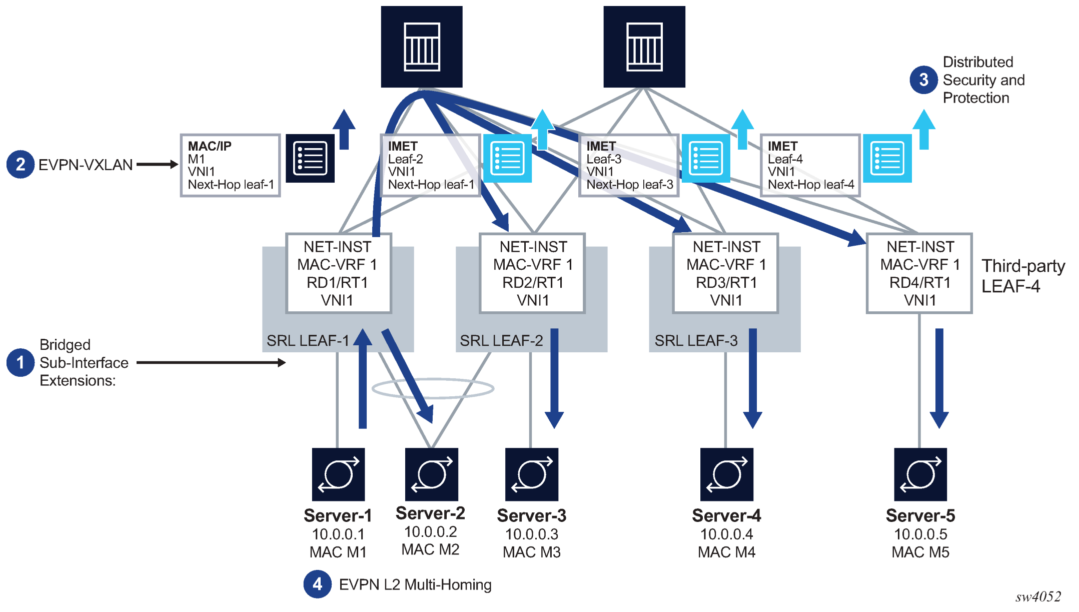


See Layer 2 services infrastructure for information about Layer 2 services on SR Linux, including configuring MAC-VRFs, IP-VRFs, and IRB interfaces.

3.3 About EVPN for VXLAN tunnels (Layer 2)

The primary usage for EVPN for VXLAN tunnels (Layer 2) is the extension of a BD in overlay multitenant DCs. This type of topology is shown in the following figure:

Figure 2: BD extension in overlay DCs


3 Distributed Security and Protection
2 EVPN-VXLAN
MAC/IP M1 VNI1 Next-Hop leaf-1
IMET Leaf-2 VNI1 Next-Hop leaf-1
IMET Leaf-3 VNI1 Next-Hop leaf-3
IMET Leaf-4 VNI1 Next-Hop leaf-4
NET-INST MAC-VRF 1 RD1/RT1 VNI1
SRL LEAF-1
NET-INST MAC-VRF 1 RD2/RT1 VNI1
SRL LEAF-2
NET-INST MAC-VRF 1 RD3/RT1 VNI1
SRL LEAF-3
NET-INST MAC-VRF 1 RD4/RT1 VNI1
Third-party LEAF-4
1 Bridged Sub-Interface Extensions:
Server-1 10.0.0.1 MAC M1
Server-2 10.0.0.2 MAC M2
Server-3 10.0.0.3 MAC M3
Server-4 10.0.0.4 MAC M4
Server-5 10.0.0.5 MAC M5
4 EVPN L2 Multi-Homing
sw4052

SR Linux features that support this topology fall into the following categories:

- bridged subinterface extensions, including:

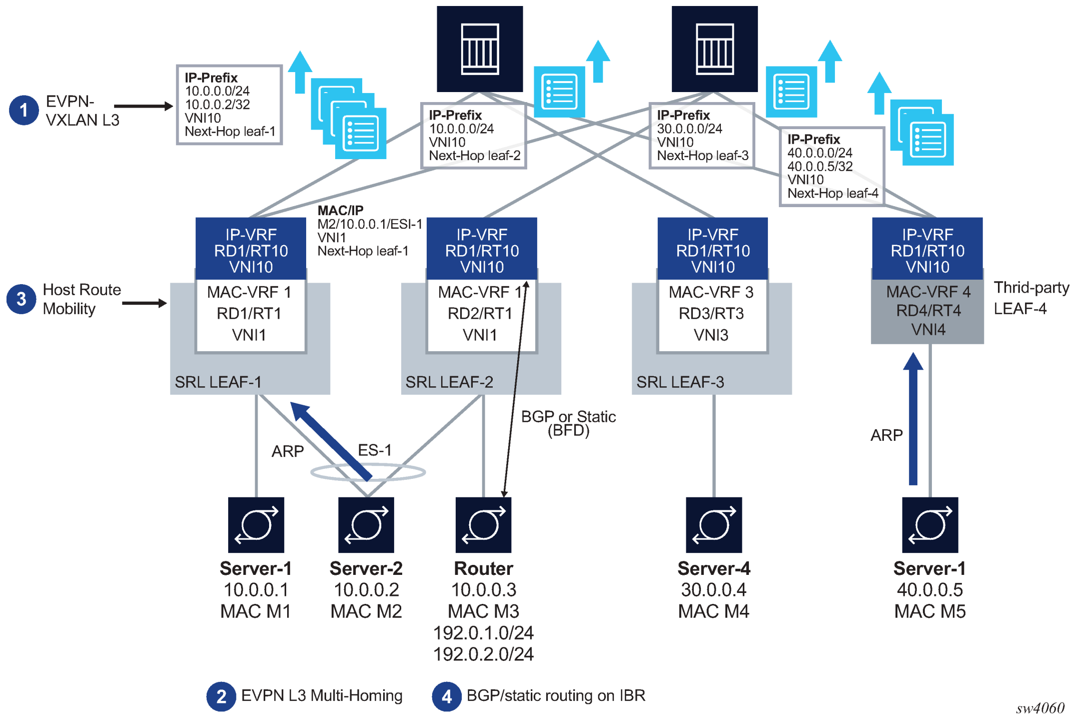
   - default subinterface, which captures untagged and non-explicitly configured VLAN-tagged frames on tagged subinterfaces
   - transparency of inner qtags not being used for service classification
- EVPN-VXLAN control and data plane extensions, as described in RFC 8365:
   - EVPN routes type MAC/IP and IMET
   - VXLANv4 model for MAC-VRFs
- distributed security and protection, including:
   - an extension to the MAC duplication mechanism that can be applied to MACs received from EVPN
   - protection of static MACs and learned-static MACs
- EVPN Layer 2 multihoming, including:
   - the Ethernet Segment (ES) model definition for all-active and single-active multihoming
   - interface-level reload-delay timers to avoid service impact when links recover
   - load-balancing and redundancy using aliasing

See EVPN for VXLAN tunnels for Layer 2 for information about the components of EVPN-VXLAN Layer 2 on SR Linux.

3.4 About EVPN for VXLAN tunnels (Layer 3)

The primary usage for EVPN for VXLAN tunnels (Layer 3) is inter-subnet-forwarding for unicast traffic within the same tenant infrastructure. This type of topology is shown in the following figure:

Figure 3: Inter-subnet forwarding with EVPN-VXLAN Layer 3



SR Linux features that support this topology fall into the following categories:

- EVPN-VXLAN Layer 3 control plane (RT5) and data plane, as described in RFC 9136
- EVPN Layer 3 multihoming on MAC-VRFs with IRB interfaces that use anycast gateway IP and MAC addresses in all leafs attached to the same BD
- host route mobility procedures to allow fast mobility of hosts between leaf nodes attached to the same BD

Other supported features include:

- interface-less (IFL) model interoperability with unnumbered interface-ful (IFF) model
- ECMP over EVPN
- support for interface-level OAM (ping) in anycast deployments

EVPN for VXLAN tunnels for Layer 3 describes the components of EVPN-VXLAN Layer 3 on SR Linux.

4 Layer 2 services infrastructure

This chapter describes the components of Layer 2 services on SR Linux. It contains the following topics:

- MAC-VRF network instance
- Interface extensions for Layer 2 services
- IRB interfaces
- Layer 2 services configuration
- Displaying bridge table information
- Deleting entries from the bridge table
- Storm control on bridged subinterfaces
- L2CP transparency
- Server aggregation configuration example

4.1 MAC-VRF network instance

The network-instance type MAC-VRF functions as a broadcast domain. Each MAC-VRF network instance builds a bridge table composed of MAC addresses that can be learned via the data path on network instance interfaces or via static configuration. You can configure the size of the bridge table for each MAC-VRF network instance, as well as the aging for dynamically learned MAC addresses and other parameters related to the bridge table.

The MAC-VRF network-instance type features a MAC duplication mechanism that monitors MAC address moves across network-instance interfaces and across interfaces.

4.1.1 MAC selection

Each MAC-VRF network instance builds a bridge table to forward Layer 2 frames based on a MAC address lookup. The SR Linux selects the MAC addresses to be sent for installation to the line card (XDP), based on the following priority:

1. local application MACs (for example, IRB interface MACs)
2. local static MACs
3. EVPN static MACs
4. local duplicate MACs
5. learned and EVPN-learned MACs

4.1.2 MAC duplication detection and actions

MAC duplication is the mechanism used by SR Linux for loop prevention. MAC duplication monitors MAC addresses that move between subinterfaces. It consists of detection, actions, and process restart.

4.1.2.1 MAC duplication detection

Detection of duplicate MAC addresses is necessary when extending broadcast domains to multiple leaf nodes. SR Linux supports a MAC duplication mechanism that monitors MAC address moves across network-instance interfaces.

A MAC address is considered a duplicate when its number of detected moves is greater than a configured threshold within a configured time frame where the moves are observed. When the threshold is exceeded, the system holds on to the prior local destination of the MAC and executes an action.

4.1.2.2 MAC duplication actions

The action taken on detecting one or more MAC addresses as duplicate on a subinterface can be configured for the MAC-VRF network instance or for the subinterface. The following are the configurable actions:

- **oper-down**

  When one or more duplicate MAC addresses are detected on the subinterface, the subinterface is brought operationally down.
- **blackhole**

  On detection of a duplicate mac on the subinterface, the mac is blackholed.
- **stop learning**

  On detection of a duplicate mac on the subinterface, the MAC address is no longer relearned on this or any subinterface. This is the default action for a MAC-VRF network instance.
- **use-network-instance-action**

  (Only available for subinterfaces) Use the action specified for the MAC-VRF network instance. This is the default action for a subinterface.

4.1.2.3 MAC duplication process restarts

When at least one duplicate MAC address is detected, the duplicate MAC addresses are visible in the state datastore and can be displayed with the **info from state mac-duplication duplicate-entries** CLI command. See Displaying bridge table information.

4.1.2.4 Configurable hold-down time

The **info from state mac-duplication duplicate-entries** command also displays the hold-down time for each duplicate MAC address. After the hold-down time expires for all of the duplicate MAC addresses for the subinterface, the oper-down or stop-learning action is cleared, and the subinterface is brought operationally up or starts learning again.

The hold-down time is configurable, ranging from 2 and 60 minutes. You can optionally specify **indefinite** for the hold-down time, which prevents the oper-down or stop-learning action from being cleared after a duplicate MAC address is detected; in this case, you can manually clear the oper-down or stop-learning action by changing the **mac-duplication** configuration or using the **tools network-instance bridge-table mac-duplication** command.

4.1.3 Bridge table configuration

The bridge table, its MAC address limit, and maximum number of entries can be configured on a per mac-vrf or per-subinterface basis.

When the size of the bridge table exceeds its maximum number of entries, the MAC addresses are removed in reverse order of the priority listed in MAC selection.

You can also configure aging for dynamically learned MAC addresses and other parameters related to the bridge table.

4.2 Interface extensions for Layer 2 services

To accommodate the Layer 2 services infrastructure, SR Linux interfaces support the following features:

- traffic classification and ingress or egress mapping actions
- subinterfaces of types routed and bridged

4.2.1 Traffic classification and ingress/egress mapping actions

On MAC-VRF network instances, traffic can be classified based on VLAN tagging. Interfaces where VLAN tagging is set to **false** or **true** can be used with MAC-VRF network instances.

A default subinterface can be specified, which captures untagged and non-explicitly configured VLAN-tagged frames in tagged subinterfaces.

Within a tagged interface, a default subinterface (where the **vlan-id** value is set to **any**) and an untagged subinterface can be configured. This type of configuration behaves as follows:

- the **vlan-id any** subinterface captures untagged and non-explicitly configured VLAN-tagged frames
- the untagged subinterface captures untagged and packets with tag0 as the outermost tag

When **vlan-id any** and untagged subinterfaces are configured on the same tagged interface, packets for unconfigured VLANs go to the **vlan-id any** subinterface and tag0 or untagged packets go to the untagged subinterface.

Classification is based on the following:

- all traffic for interfaces where VLAN tagging is set to false, regardless of existing VLAN tags
- single outermost tag for tagged interfaces where VLAN tagging is set to true. Only Ethertype 0x8100 is considered for tags; other Ethertypes are treated as payload

The following ingress and egress VLAN mapping actions are supported:

- at ingress, pop the single outermost tag (tagged interfaces) or perform no user-visible action (untagged interfaces)

- at egress, push a specified tag at the top of the stack (tagged interfaces) or perform no user-visible action (untagged interfaces)
- if the **vlan-id** value is set to **any** or the subinterface uses an **untagged** configuration, no tag is popped at ingress or pushed at egress

  There is one exception: on a subinterface that uses an **untagged** configuration, if a received packet has tag0 as its outermost tag, the subinterface pops tag0.

Dot1p is not supported.

4.2.2 Routed and bridged subinterfaces

SR Linux subinterfaces can be specified as type routed or bridged:

- routed subinterfaces can be assigned to a network instance of type mgmt, default, or ip-vrf
- bridged subinterfaces can be assigned to a network instance of type mac-vrf

Routed subinterfaces allow for configuration of IPv4 and IPv6 settings, and bridged subinterfaces allow for configuration of bridge table and VLAN ingress or egress mapping.

Bridged subinterfaces do not have MTU checks other than the interface-level MTU (port MTU) or the value set with the **l2-mtu** command. The IP MTU is only configurable on routed subinterfaces.

4.3 IRB interfaces

IRB interfaces enable inter-subnet forwarding. Network instances of type mac-vrf are associated with a network instance of type ip-vrf via an IRB interface. See Figure 1: MAC-VRF, IRB interface, and IP-VRF for an illustration of the relationship between MAC-VRF and IP-VRF network instances.

On SR Linux, IRB interfaces are named `irb`*N*, where *N* is an integer in the range of 0 to 255. Up to 4095 subinterfaces can be defined under an IRB interface. An IP-VRF network instance can have multiple IRB subinterfaces, while a MAC-VRF network instance can reference only one IRB subinterface.

IRB subinterfaces are type routed. They cannot be configured as type bridged.

IRB subinterfaces operate in the same way as other routed subinterfaces, including support for the following:

- IPv4 and IPv6 access control lists (ACLs)
- DSCP-based QoS (input and output classifiers and rewrite rules)
- static routes and BGP (IPv4 and IPv6 families)
- IP MTU (with the same range of valid values as Ethernet subinterfaces)
- all settings in the subinterface/ipv4 and subinterface/ipv6 containers; for IPv6, the IRB subinterface also gets an IPv6 link local address
- BFD
- subinterface statistics

IRB interfaces do not support sFlow, VLAN tagging, or interface statistics.

4.3.1 Using ACLs with IRB interfaces and Layer 2 subinterfaces

The following items apply when using access control lists (ACLs) with an IRB interface or Layer 2 subinterface:

- Input ACLs associated with Layer 2 subinterfaces match all the traffic entering the subinterface, including Layer 2 switched traffic or Layer 3 traffic forwarded to the IRB.
- Input ACLs associated with IRB subinterfaces only match Layer 3 traffic, that is, traffic with a MAC destination address (DA) matching the IRB MAC address.
- The same ACL can be attached to a Layer 2 subinterface and an IRB subinterface in the same service. In this case, there are two ACL instances: one for the IRB with higher priority and another one for the bridged traffic. Routed traffic matches the higher priority instance entries of the ACL.
- The same ACL can be attached to IRB subinterfaces and Layer 2 subinterfaces if both subinterface types belong to different services.
- On 7220 IXR-D1, D2, and D3 systems, egress ACLs, unlike ingress ACLs, cannot match both routed and switched traffic when a Layer 2 IP ACL is attached.
- Received traffic on a MAC-VRF is automatically discarded if the MAC source address (SA) matches the IRB MAC address, unless the MAC is an anycast gateway MAC.
- Packet capture filters show the Layer 2 subinterface for switched traffic and the IRB interface for routed traffic.

4.4 Layer 2 services configuration

The examples in this section show how to configure a MAC-VRF network instance, bridged interface, and IRB interface.

4.4.1 MAC-VRF network instance configuration example

The following example configures a MAC-VRF network instance and settings for the bridge table. The bridge table is set to a maximum of 500 entries. Learned MAC addresses are aged out of the bridge table after 600 seconds.

MAC duplication detection is configured so that a MAC address is considered a duplicate when its number of detected moves across network-instance interfaces is greater than three over a 5-minute interval. In this example, the MAC address is blackholed. After the hold-down time of 3 minutes, the MAC address is flushed from the bridge table, and the monitoring process for the MAC address is restarted.

The example includes configuration for a static MAC address in the bridge table.

The MAC-VRF network instance is associated with a bridged interface and an IRB interface.

```
--{ candidate shared default }--[  ]--
# info network-instance mac-vrf-1
    network-instance mac-vrf-1 {
        type mac-vrf
        admin-state enable
        description "Sample mac-vrf network instance"
        interface ethernet-1/1.1 {
        }
        interface irb1.1 {
        }
        bridge-table {
            mac-learning {
                admin-state enable
                aging {
                    admin-state enable
                    age-time 600
                }
            }
            mac-duplication {
                admin-state enable
                monitoring-window 5
                num-moves 3
                hold-down-time 3
                action blackhole
            }
            mac-limit {
                maximum-entries 500
            }
            static-mac {
                mac 00:00:5E:00:53:FF {
                }
            }
        }
    }
```

4.4.2 Bridged subinterface configuration example

The following example configures the bridged subinterface that is associated with the MAC-VRF in the previous example.

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1
    interface ethernet-1/1 {
        admin-state enable
        subinterface 1 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 10
                    }
                }
            }
        }
    }
```

The **vlan-id** value can be configured as a specific valid number or with the keyword **any**, which means any frame that does not hit the **vlan-id** configured in other subinterfaces of the same interface is classified in this subinterface.

In the following example, the **vlan encap untagged** setting is enabled for subinterface 1. This setting allows untagged frames to be captured on tagged interfaces.

For subinterface 2, the **vlan encap single-tagged vlan-id any** setting allows non-configured VLAN IDs and untagged traffic to be classified to this subinterface.

With the **vlan encap untagged** setting on one subinterface, and the **vlan encap single-tagged vlan-id any** setting on the other subinterface, traffic enters the appropriate subinterface; that is, traffic for unconfigured VLANs goes to subinterface 2, and tag0 or untagged traffic goes to subinterface 1.

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/2
    interface ethernet-1/2 {
        vlan-tagging true
        subinterface 1 {
            type bridged
            vlan {
                encap {
                    untagged {
                    }
                }
            }
        }
        subinterface 2 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id any
                    }
                }
            }
        }
    }
```

4.4.3 IRB interface configuration example

The following example configures an IRB interface. The IRB interface is operationally up when its admin-state is enabled, and its IRB subinterfaces are operationally up when associated with MAC-VRF and IP-VRF network instances. At least one IPv4 or IPv6 address must be configured for the IRB subinterface to be operationally up.

```
--{ candidate shared default }--[  ]--
# info interface irb1
    interface irb1 {
        description IRB_Interface
        admin-state enable
        subinterface 1 {
            admin-state enable
            ipv4 {
                admin-state enable
                address 192.168.1.1/24 {
                }
            }
        }
    }
```

4.5 Displaying bridge table information

Procedure

You can display information from the bridge table of a MAC-VRF network instance using **show** commands and the **info from state** command.

Example: Display summary information for MAC-VRF network instances

To display a summary of the bridge table contents for the MAC-VRF network instances configured on the system:

```
# show network-instance * bridge-table mac-table summary
-------------------------------------------------------------------
Network-Instance Bridge table summary
-------------------------------------------------------------------
-------------------------------------------------------------------
Name                            : mac-vrf-1
Irb mac                         :   1 Total   1 Active
Static macs                     :  19 Total  19 Active
Duplicate macs                  :  10 Total  10 Active
Learnt macs                     :  15 Total  14 Active
Total macs                      :  45 Total  44 Active
Maximum-Entries                 : 200
Warning Threshold Percentage    :  95% (190)
Clear Warning                   :  90% (180)
-------------------------------------------------------------------
Name                            : mac-vrf-2
Irb mac                         :   1 Total   1 Active
Static macs                     :   1 Total   1 Active
Duplicate macs                  :  10 Total  10 Active
Learnt macs                     :  15 Total  14 Active
Total macs                      :  27 Total  26 Active
Maximum-Entries                 : 200
Warning Threshold Percentage    :  95% (190)
Clear Warning                   :  90% (180)
-------------------------------------------------------------------
-------------------------------------------------------------------
Total Irb macs             : 2  Total 2 Active
Total Static macs          : 29 Total 29 Active
Total Duplicate macs       : 20 Total 20 Active
Total Learnt macs          : 30 Total 29 Active
Total Macs                 : 81 Total 80 Active
-------------------------------------------------------------------
```

Example: Display bridge table for a MAC-VRF network instance

To list the contents of the bridge table for a MAC-VRF network instance:

```
# show network-instance mac-vrf-1 bridge-table mac-table all
--------------------------------------------------------------------------------------
Mac-table of network instance mac-vrf-1
--------------------------------------------------------------------------------------
+------------------+--------------+----------+---------+------+-----+----------------+
| Mac              |Destination   |Dest-Index|Type     |Active|Aging|Last-update     |
+==================+==============+==========+=========+======+=====+================+
| 00:00:00:00:00:01|ethernet-1/1.1|65        |Learnt   |True  |256  |2020-2-3T3:37:26|
| 00:00:00:00:00:02|        irb1.1| 0        |Irb      |True  |N/A  |2019-2-1T3:37:26|
| 00:00:00:00:00:03|     blackhole| 0        |Duplicate|True  |N/A  |2019-2-1T3:37:26|
| 00:00:00:00:00:04|ethernet-1/1.2|66        |Learnt   |True  |256  |2019-2-1T3:37:26|
--------------------------------------------------------------------------------------
Total Irb macs             : 2  Total 2 Active
Total Static macs          : 0  Total 0 Active
Total Duplicate macs       : 1  Total 1 Active
Total Learnt macs          : 3  Total 3 Active
Total Macs                 : 6  Total 6 Active
-------------------------------------------------------------------------------------
```

Example: Display information about a MAC address

To display information about a specific MAC address in the bridge table:

```
# show network-instance * bridge-table mac-table mac 00:00:00:00:00:01
-----------------------------------------------------------------------------------
Mac-table of network instance mac-vrf-1
-----------------------------------------------------------------------------------
Mac                     : 00:00:00:00:00:01
Destination             : ethernet-1/1.1
Destination Index       : 65
Type                    : Learnt
Programming status      : Success | Failed | Pending
Aging                   : 250 seconds
Last update             : 2019-12-13T23:37:26.000
Duplicate Detect Time   : N/A
Hold-down-time-remaining: N/A
-----------------------------------------------------------------------------------
```

Example: Display duplicate MAC addresses

To display the duplicate MAC address entries in the bridge table:

```
# show network-instance * bridge-table mac-duplication duplicate-entries
-----------------------------------------------------------------------------------------
Mac-duplication in network instance mac-vrf-1
-----------------------------------------------------------------------------------------
Admin-state             : Enabled
Monitoring window       : 3 minutes
Number of moves allowed : 5
Hold-down-time          : 10 seconds
Action                  : Stop Learning
-----------------------------------------------------------------------------------------
Duplicate entries in network instance mac-vrf-1
-----------------------------------------------------------------------------------------
+------------------+----------------+----------+------------------------+----------------+
| Duplicate mac    | Destination    |Dest-Index| Detect-Time            | Hold down      |
|                  |                |          |                        | time remaining |
+==================+================+==========+========================+=================
| 00:00:00:00:00:01|ethernet-1/1.1  |65        |2019-12-13T23:37:26.000 | 6
| 00:00:00:00:00:02|ethernet-1/1.1  |65        |2019-12-13T23:37:26.000 | 6
| 00:00:00:00:00:03|ethernet-1/1.1  |65        |2019-12-13T23:37:26.000 | 6
-----------------------------------------------------------------------------------------
Total Duplicate macs       : 3  Total 3 Active
-----------------------------------------------------------------------------------------
```

Example: Use info from state command to display MAC address entries

You can display the duplicate, learned, or static MAC address entries in the bridge table using **info from state** commands. For example, the following command displays the duplicate MAC entries:

```
# info from state network-instance * bridge-table mac-duplication duplicate-entries mac *
 | as
 table
+--------------------+---------------------+---------+------------------------+-----------
----+
| Network-instance   | Duplicate-mac       | Dest-idx| Detect-time            | Hold-down-
time|
|                    |                     |         |                        | remaining
    |
+====================+=====================+=========+========================+===========
====+
| red                | 00:00:00:00:00:01   |       1 | 2019-12-13T23:37:26.000|
 10 |
| red                | 00:00:00:00:00:02   |       2 | 2019-12-13T23:37:26.000|
 20 |
| red                | 00:00:00:00:00:03   |       3 | 2019-12-13T23:37:26.000|
 90 |
| blue               | 00:00:00:00:00:04   |       4 | 2019-12-13T23:37:26.000|
 90 |
| blue               | 00:00:00:00:00:05   |       0 | 2019-12-13T23:37:26.000|
 90 |
+--------------------+---------------------+---------+------------------------+-----------
----+
```

Example: Display learned MAC addresses

The following command displays the learned MAC entries in the table:

```
# info from state network-instance * bridge-table mac-learning learnt-entries mac * | as
 table
+------------------+------------------+--------------+---------+------------------------+
| Network-instance | Learnt-mac       | Dest         | Aging   | Last-update            |
+==================+==================+==============+=========+========================+
| red              | 00:00:00:00:00:01|ethernet-1/1.1| 300     | 2019-12-13T23:37:26.000|
| red              | 00:00:00:00:00:02|ethernet-1/1.1| 212     | 2019-12-13T23:37:26.000|
| red              | 00:00:00:00:00:03|ethernet-1/1.2| 10      | 2019-12-13T23:37:26.000|
| blue             | 00:00:00:00:00:04|ethernet-1/1.3| 10      | 2019-12-13T23:37:26.000|
| blue             | 00:00:00:00:00:05|ethernet-1/1.4| 20      | 2019-12-13T23:37:26.000|
+------------------+------------------+--------------+---------+------------------------+
```

Example: Display static MAC entries

The following command displays the static MAC entries in the table:

```
# info from state network-instance * bridge-table static-mac mac * | as table
+------------------+------------------+--------------+
| Network-instance | Static-mac       | Dest         |
+==================+==================+==============+
| red              | 00:00:00:00:00:01|ethernet-1/1.1|
| red              | 00:00:00:00:00:02|        irb1.1|
| red              | 00:00:00:00:00:03|     blackhole|
| blue             | 00:00:00:00:00:04|ethernet-1/1.3|
| blue             | 00:00:00:00:00:05|ethernet-1/1.4|
+------------------+------------------+--------------+
```

4.6 Deleting entries from the bridge table

Procedure

The SR Linux includes commands for deleting duplicate or learned MAC entries from the bridge table. For a MAC-VRF or subinterface, you can delete all MAC entries, MAC entries with a blackhole destination, or a specific MAC entry.

Example: Clear blackhole MAC entries

The following example clears MAC entries in the bridge table for a MAC-VRF network instance that have a blackhole destination:

```
--{ candidate shared default }--[  ]--
# tools network-instance mac-vrf-1 bridge-table mac-duplication delete-blackhole-macs
```

Example: Delete a learned MAC address

The following example deletes a specified learned MAC address from the bridge table for a MAC-VRF network instance:

```
--{ candidate shared default }--[  ]--
# tools network-instance mac-vrf-1 bridge-table mac-learning delete-mac 00:00:00:00:00:04
```

Example: Clear duplicate MAC entries

The following example clears all duplicate MAC entries in the bridge table for a subinterface:

```
--{ candidate shared default }--[  ]--
# tools interface ethernet-1/1.1 bridge-table mac-duplication delete-all-macs
```

4.7 Storm control on bridged subinterfaces

Storm control allows you to rate-limit broadcast, unknown-unicast, and multicast (BUM) traffic on bridged subinterfaces. This feature measures the rate for the individual types of BUM traffic, and performs rate-limiting based on thresholds configured for each traffic type.

Storm control is configured as an interface-level policer. You set thresholds for each type of BUM traffic. You can configure the threshold as a percentage of interface bandwidth or as a rate in kb/s.

If `kbps` is specified as the unit type, the rate can be configured as a multiple of 64; for example, 64, 128, 192, and so on. If the rate is configured as any value in the 1 to 127 kb/s range, the effective rate is 64 kb/s, which is shown as the operational-rate in **info from state** output. Similarly, if the rate is configured as any value in the 128 to 191 kb/s range, the operational rate is 128 kb/s, and so on.

The default threshold is 100% of the interface bandwidth for the three traffic types. Specifying a threshold of 0 for a traffic type blocks all traffic of that type on the interface. When a storm control policer is set to 0, the first packet of a flow affected by the policer is forwarded, but subsequent packets are dropped.

Storm control policers can be configured for Ethernet interfaces only. The Ethernet interfaces can be members of a LAG, but storm control must be configured on each interface individually; aggregate rate-limiting for the LAG is not supported.

Statistics are not collected for the storm-control policers. However, it is possible to check discarded packets at the subinterface level. For example, if storm-control for broadcast traffic is configured on interface ethernet-1/1 at a rate of 50%, and broadcast traffic on the ethernet-1/1.1 subinterface exceeds 50% of the port capacity, statistics for the ethernet-1/1.1 subinterface statistics show ingress drops.

Storm control is supported on 7220 IXR-D1/D2/D3/D4/D5 platforms.

4.7.1 Configuring storm control for an interface

Procedure

To configure storm control, you set thresholds for each type of BUM traffic, either as a percentage of interface bandwidth or as a rate in `kbps`.

Example:

The following example configures storm control for an interface. The unit type is specified as `kbps`. Individual thresholds are set for each traffic type.

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1
    interface ethernet-1/1 {
        admin-state enable
        ethernet {
            storm-control {
                units kbps
                broadcast-rate 128
                multicast-rate 192
                unknown-unicast-rate 225
            }
        }
    }
```

4.7.2 Displaying storm control information

Procedure

You can display the configured rates for an interface using **info from state** and **show interface** commands.

Example: Display configured rates using info from state command

```
--{ * candidate shared default }--[  ]--
# info from state interface ethernet-1/1 ethernet storm-control
    interface ethernet-1/1 {
        ethernet {
            storm-control {
                units kbps
                broadcast-rate 128
                multicast-rate 172
                unknown-unicast-rate 225
                operational broadcast-rate 128
                operational multicast-rate 192
                operational unknown-unicast-rate 192
            }
        }
    }
```

Example: Display configured rates using show interface command

```
--{ * candidate shared default }--[  ]--
# show interface ethernet-1/1 detail
========================================================================
Interface: ethernet-1/1
------------------------------------------------------------------------
```

```
  Description     : dut2_noxia_1
  Oper state      : up
  Down reason     : N/A
  Last change     : 9h21m16s ago, 1 flaps since last clear
  Speed           : 100G
  Flow control    : Rx is disabled, Tx is disabled
  Loopback mode   : false
  MTU             : 9232
  VLAN tagging    : true
  Queues          : 8 output queues supported, 0 used since the last clear
  MAC address     : 00:01:02:FF:00:0C
  Last stats clear: never
-------------------------------------------------------------------------
Storm control for ethernet-1/1
-------------------------------------------------------------------------
  Broadcast Rate (kbps)                    : 128
  Multicast Rate (kbps)                    : 172
  Unknown Unicast (kbps)                   : 225
  Operational Broadcast Rate (kbps)        : 128
  Operational Multicast Rate (kbps)        : 192
  Operational Unknown Unicast Rate (kbps)  : 192
```

4.8 L2CP transparency

Layer 2 Control Protocol (L2CP) transparency refers to the ability to control whether L2CP frames are tunneled across a carrier Ethernet network. Tunneling in this context means injecting the L2CP frames into the regular Layer 2 datapath, as opposed to trapping them to the CPU or discarding them.

By default, SR Linux handles L2CP frames according to the default transparency action listed in Table 1: Default L2CP PDU actions for tagged and untagged frames.

Table 1: Default L2CP PDU actions for tagged and untagged frames

Layer 2 control protocol	L2CP destination address	Protocol identifier	Default transparency action
LLDP	01-80-c2-00-00-0e Only the above MAC address is identified as LLDP; other LLDP destination MAC addresses are not.	Ethertype: 0x88cc	Trap to CPU only if untagged, drop otherwise
LACP	01-80-c2-00-00-02	Ethertype: 0x8809 Subtype: 0x01	Trap to CPU only if untagged, drop otherwise
xSTP (STP/RSTP/ MSTP)	01-80-c2-00-00-00	Ethertype: any	Drop
PAUSE	01-80-c2-00-00-01	Ethertype: 0x8808	Drop
dot1x	01-80-c2-00-00-03	Ethertype: 0x888e	Drop
PTP	01-80-c2-00-00-0e	Ethertype: 0x88f7	Drop

Layer 2 control protocol	L2CP destination address	Protocol identifier	Default transparency action
E-LMI	01-80-c2-00-00-07	Ethertype: 0x88ee	Drop
GARP/MRP	01-80-c2-00-00-2x	Ethertype: any	Drop
LACP Marker-Protocol/ Link OAM 802.3ah/ ESMC	01-80-c2-00-00-02	Ethertype: 0x8809 All other unsupported subtypes	Drop

The L2CP transparency feature allows you to configure SR Linux to tunnel the following types of L2CP frames instead of performing the default transparency action, or you can configure SR Linux to tunnel all L2CP frames.

- LLDP
- LACP
- xSTP (STP/RSTP/MSTP)
- dot1x
- PTP

When tunneling is enabled, the following take place:

- Tagged and untagged L2CP frames are classified into a bridged subinterface based on their dot1q tag VLAN ID. If they are classified for forwarding into a MAC-VRF, they are flooded based on their MAC DA. If no bridged subinterface matches the incoming untagged or VLAN-ID tagged frames, they are discarded.
- L2CP transparency is supported on Ethernet interfaces only, with the processing and tunneling possible on bridged subinterfaces of a MAC-VRF. The system does not apply L2CP transparency rules to VXLAN tunnel interfaces, however it does apply them to Layer 3 subinterfaces in the default network instance.
- When an L2CP PDU is tunneled, the source MAC is learned. When the PDU is dropped, the source MAC is not learned.

To tunnel L2CP frames end-to-end between two EVPN-VXLAN leaf nodes, you must enable L2CP transparency on the ingress Layer 3 subinterfaces of the default network instance on the egress leaf node. Otherwise, even if L2CP transparency for a protocol is configured on the bridged subinterfaces of the ingress leaf, the system discards the L2CP frames at the ingress VXLAN tunnel of the egress leaf.

L2CP transparency is supported in configurations where tunneling for LLDP frames is used in the overlay (between PE and CE devices), and LLDP is used in the underlay (between spine and leaf nodes).

L2CP transparency is not supported on interfaces configured in breakout mode. For LAG interfaces, you must configure L2CP transparency on all member interfaces.

4.8.1 Configuring L2CP transparency

Procedure

You can enable tunneling for all L2CP protocol types listed in Table 1: Default L2CP PDU actions for tagged and untagged frames or you can enable tunneling individually for the following L2CP protocol types: LLDP, LACP xSTP (STP/RSTP/MSTP), dot1x, and PTP.

Example: Enable tunneling for all L2CP protocols

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1 ethernet l2cp-transparency
    interface ethernet-1/1 {
        ethernet {
            l2cp-transparency {
                tunnel-all-l2cp true
            }
        }
    }
```

When **tunnel-all-l2cp true** is configured, all L2CP frames coming into the interface, identified by MAC DA = 01:80:c2:00:00:0*x* or MAC DA = 01:80:c2:00:00:2*x* where *x* is any hex value, are tunneled.

The **tunnel-all-l2cp** command is not supported on interfaces where LLDP or LACP are enabled.

Example: Disable tunneling for all L2CP protocols

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1 ethernet l2cp-transparency
    interface ethernet-1/1 {
        ethernet {
            l2cp-transparency {
                tunnel-all-l2cp false
            }
        }
    }
```

When **tunnel-all-l2cp false** is configured, all L2CP frames not specifically enabled for tunneling are discarded.

Example: Enable tunneling for individual protocol types

You can enable tunneling individually for LLDP, LACP xSTP (STP/RSTP/MSTP), dot1x, and PTP L2CP protocol types. These L2CP frames are identified by the destination MAC address, Ethertype and sometimes slow-protocol subtype; see Table 1: Default L2CP PDU actions for tagged and untagged frames.

The following example enables tunneling for LLDP frames:

```
# info interface ethernet-1/1 ethernet l2cp-transparency
    interface ethernet-1/1 {
        ethernet {
            l2cp-transparency {
                lldp {
                    tunnel true
                }
            }
        }
    }
```

By default, all the **tunnel** commands are set to **false**, so all L2CP protocols are discarded, except for LLDP and LACP, for which a copy is trapped to the CPU (if untagged) and another copy is discarded.

Note: Only LLDP frames with MAC 01-80-c2-00-00-0e are tunneled. LLDP frames with MAC DA = 01-80-c2-00-00-00 or 01-80-c2-00-00-03 are not tunneled or trapped to the CPU.

Tunneling for LLDP frames is not supported on an interface where LLDP is enabled.

Tunneling for LACP frames is not supported on an interface where `aggregate-id` is configured or the interface is member of a LAG where LACP is enabled.

4.8.2 Displaying L2CP transparency information

Procedure

Each of the L2CP protocols that can be tunneled is associated with an L2CP transparency rule (`oper-rule`) that indicates the state of the actual rule installed in the datapath. This rule can be any of the following:

- `trap-to-cpu-untagged`
  This rule is used for LLDP and LACP frames if they are not configured to be tunneled. Untagged frames are trapped to the CPU; that is, a copy is trapped and sent to the CPM, while another copy is discarded. Tagged frames are discarded and not trapped.
- `drop-tagged-and-untagged`
  This rule is used for xSTP, dot1x and PTP frames if they are not configured for tunneling.
- `tunnel-tagged-and-untagged`
  This rule is used for any of the five L2CP protocols when they are configured to be tunneled.

You can display the L2CP transparency configuration and `oper-rule` for each L2CP protocol that can be tunneled.

Example: Display tunneling configuration and oper-rule for an interface

Use the **info from state** command for an interface to display its tunnelling configuration and `oper-rule` for each protocol. For example:

```
--{ candidate shared default }--[  ]--
# info from state interface ethernet-1/1 ethernet l2cp-transparency
    interface ethernet-1/1 {
        ethernet {
            l2cp-transparency {
                tunnel-all-l2cp false
                lldp {
                    tunnel false
                    oper-rule trap-to-cpu-untagged
                }
                lacp {
                    tunnel false
                    oper-rule trap-to-cpu-untagged
                }
                xstp {
                    tunnel false
                    oper-rule drop-tagged-and-untagged
                }
                dot1x {
                    tunnel false
                    oper-rule drop-tagged-and-untagged
                }
                ptp {
                    tunnel false
                    oper-rule drop-tagged-and-untagged
                }
            }
        }
    }
```

Example: Display the oper-rule for each protocol type

Use the **show interface detail** command to display the L2CP transparency rule (`oper-rule`) for each protocol type; for example:

```
--{ candidate shared default }--[  ]--
# show interface ethernet-1/1 detail
=============================================================================
Interface: ethernet-1/1
-----------------------------------------------------------------------------
  Description     : <None>
  Oper state      : up
  Down reason     : N/A
  Last change     : 2d22h27m45s ago, 1 flaps since last clear
  Speed           : 10G
  Flow control    : Rx is disabled, Tx is disabled
  MTU             : 9232
  VLAN tagging    : false
  Queues          : 8 output queues supported, 2 used since the last clear
  MAC address     : 00:01:01:FF:00:01
  Last stats clear: never
  Breakout mode   : false
-----------------------------------------------------------------------------
L2cp transparency rules for ethernet-1/1
-----------------------------------------------------------------------------
  Lldp               : trap-to-cpu-untagged
  Lacp               : trap-to-cpu-untagged
  xStp               : drop-tagged-and-untagged
  Dot1x              : drop-tagged-and-untagged
  Ptp                : tunnel-tagged-and-untagged
  Non-specified L2cp : drop-tagged-and-untagged
-----------------------------------------------------------------------------
```

4.8.3 Displaying L2CP transparency statistics

Procedure

To display L2CP transparency statistics, use the **info from state** command in candidate or running mode, or the **info** command in state mode.

Statistics are displayed for the following:

- total number of L2CP packets of any L2CP protocol on all interfaces, as well as the total number of discarded and tunneled L2CP packets
- total number of ingress tunneled and trapped-to-CPU packets per L2CP protocol at the system level. Individual statistics are displayed for LLDP, LACP, xSTP, dot1x, and PTP protocols if `tunnel-all-l2cp false` is configured

Example

```
--{ candidate shared default }--[  ]--
# info from state system l2cp-transparency l2cp-statistics
    system {
        l2cp-transparency {
            l2cp-statistics {
                total-in-packets 0
                total-in-discarded-packets 0
                total-in-tunneled-packets 0
                total-in-trap-to-cpu-packets 0
                last-clear "a day ago"
                lldp {
                    in-tunneled-packets 0
                    in-trap-to-cpu-packets 0
                }
                lacp {
                    in-tunneled-packets 0
                    in-trap-to-cpu-packets 0
                }
                xstp {
                    in-tunneled-packets 0
                    in-trap-to-cpu-packets 0
                }
                dot1x {
                    in-tunneled-packets 0
                    in-trap-to-cpu-packets 0
                }
                ptp {
                    in-tunneled-packets 0
                    in-trap-to-cpu-packets 0
                }
            }
        }
    }
```

Note:

- Discarded L2CP frames are counted as `in-discarded-packets` in interface-level statistics. They are not counted in subinterface statistics.
- Tunneled L2CP frames are counted in interface and subinterface statistics for outgoing interfaces in the appropriate counters.
- If **tunnel-all-l2cp true** is configured, all L2CP frames (including LLDP, LACP, xSTP, dot1x, and PTP) are counted only in `l2cp-statistics/total-in-tunneled-packets` regardless of protocol-specific configuration. In this case, the protocol-specific statistics are displayed as 0.

4.8.3.1 Clearing L2CP transparency statistics

Procedure

You can use a **tools** command to clear the total counters and the per-protocol counters for L2CP transparency statistics.

Example: Clear statistics counters for all L2CP protocols

```
--{ running }--[  ]--
```

```
# tools system l2cp-transparency l2cp-total-statistics clear
```

Example: Clear statistics counters for a specific L2CP protocol type

```
--{ running }--[  ]--
# tools system l2cp-transparency ptp clear
```

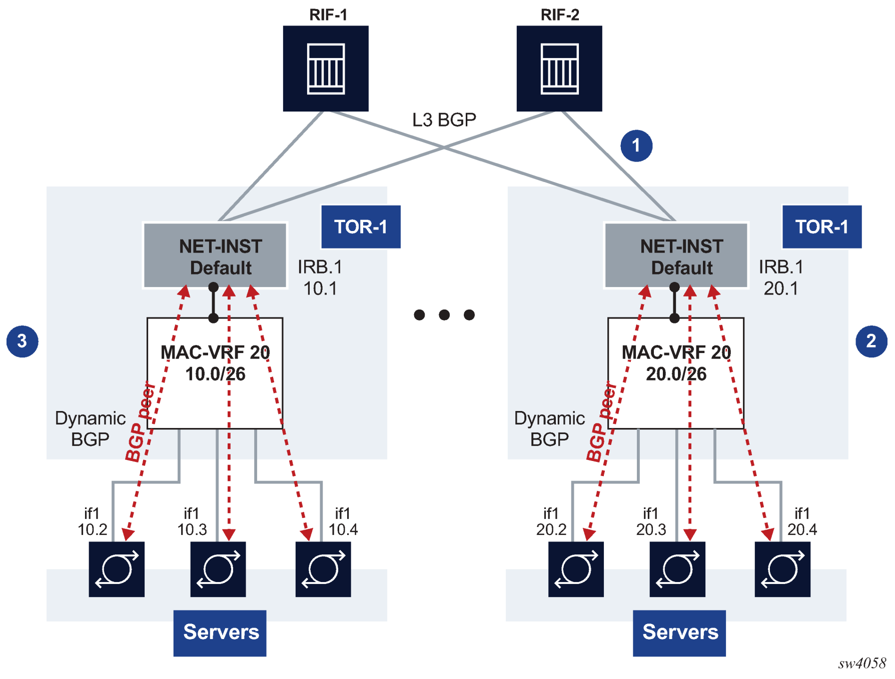
4.9 Server aggregation configuration example

Figure 4: Server aggregation example shows an example of using MAC-VRF network instances to aggregate servers into the same subnet.

In this example, Leaf-1 and Leaf-2 are configured with MAC-VRF instances that aggregate a group of servers. These servers are assigned IP addresses in the same subnet and are connected to the leaf default network instance by a single IRB subinterface. The servers use a PE-CE BGP session with the IRB IP address to exchange reachability.

Using the MAC-VRF with an IRB subinterface saves routed subinterfaces on the default network instance; only one routed subinterface is needed, as opposed to one per server.

Figure 4: Server aggregation example



In this example:

1. TORs peer (eBGP) to two or four RIFs.
2. MAC-VRF 20 is defined in TORs with an IRB interface with IPv4/IPv6 addresses. DHCP relay is supported on IRB subinterfaces.
3. The following Layer 2 features are implemented or loop and MAC duplication protection:
   - MAC duplication with oper-down or blackhole actions configured on the bridged subinterfaces

- storm control for BUM traffic on bridged subinterfaces

This example uses the following features:

- MAC-VRF with bridge subinterfaces and IRB subinterfaces to the default network instance
- PE-CE BGP sessions for IPv4 and IPv6 address families
- MAC duplication with oper-down or blackhole actions configured on the bridged subinterfaces
- storm control for BUM traffic on bridged subinterfaces

4.9.1 Configuration for server aggregation example

The following shows the configuration of Leaf-1 in Figure 4: Server aggregation example and its BGP session via IRB to server 1. Similar configuration is used for other servers and other TORs.

```
--{ [FACTORY] + candidate shared default }--[ interface * ]--
A:Leaf-1# info
    interface ethernet-1/1 {
        description tor1-server1
        vlan-tagging true
        subinterface 1 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id 100
                    }
                }
            }
        }
    }

// Configure an IRB interface and sub-interface that will connect
 the MAC-VRF to the existing default network-instance.

--{ [FACTORY] + candidate shared default }--[ interface irb* ]--
A:Leaf-1# info
    interface irb1 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                address 10.0.0.2/24 {
                }
            }
            ipv6 {
                admin-state enable
                address 2001:db8::2/64 {
                }
            }
        }
    }

// Configure the network-instance type mac-vrf
and associate the bridged and irb interfaces to it.

--{ [FACTORY] + candidate shared default }--[ network-instance MAC-VRF-1 ]--
A:Leaf-1# info
    type mac-vrf
    admin-state enable
    interface ethernet-1/1.1 {
    }
    interface irb1.1 {
    }

// Associate the same IRB interface to the network-
instance default and configure the BGP IPv4 and IPv6 neighbors to DUT1 and DUT3.

--{ [FACTORY] + candidate shared default }--[ network-instance default ]--
A:Leaf-1# info
    type default
    admin-state enable
    router-id 2.2.2.2
    interface irb1.1 {
    }
    protocols {
        bgp {
            admin-state enable
            afi-safi ipv4-unicast {
                admin-state enable
            }
            autonomous-system 64502
            router-id 10.0.0.2
            ebgp-default-policy {
                import-reject-all false
            }
            failure-detection {
                enable-bfd true
                fast-failover true
            }
            group leaf {
                admin-state enable
                export-policy pass-all
                afi-safi ipv4-unicast {
                    admin-state enable
                }
                afi-safi ipv6-unicast {
                    admin-state enable
                }
                local-as as-number 64502 {
                }
                timers {
                    minimum-advertisement-interval 1
                }
            }
            afi-safi ipv4-unicast {
                admin-state enable
            }
            afi-safi ipv6-unicast {
                admin-state enable
            }
            neighbor 10.0.0.1 {
                peer-as 64501
                peer-group leaf
                transport {
                    local-address 10.0.0.2
                }
            }
            neighbor 2001:db8::1 {
                peer-as 64501
                peer-group leaf
                transport {
                    local-address 2001:db8::2
                }
            }
        }
    }
```

5 VXLAN v4

This chapter describes the implementation for VXLAN tunnels that use IPv4 in the underlay.

- VXLAN configuration
- VXLAN and ECMP
- VXLAN ACLs
- QoS for VXLAN tunnels
- VXLAN statistics collection

5.1 VXLAN configuration

VXLAN on SR Linux uses a tunnel model where VXLAN interfaces are bound to network instances, in the same way that subinterfaces are bound to network instances. Up to one VXLAN interface per network instance is supported.

Configuration of VXLAN on SR Linux is tied to EVPN. VXLAN configuration consists of the following steps:

1. Configure a tunnel interface and VXLAN interface.

   A tunnel interface for VXLAN is configured as `vxlan<N>`, where N can be 0 to 255.

   A VXLAN interface is configured under a tunnel interface. At a minimum, a VXLAN interface must have an index, type, and the ingress VXLAN network identifier (VNI).

   - The index can be a number in the range 0 to 4294967295.
   - The type can be bridged or routed and indicates whether the vxlan-interface can be linked to a MAC-VRF (bridged) or IP-VRF (routed).
   - The system looks for the ingress VNI in incoming VXLAN packets to classify them to this VXLAN interface and its network instance.

   Configuration of an explicit VXLAN interface egress source IP is not permitted, given that the data path supports one source tunnel IP address for all VXLAN interfaces. The source IP used in the VXLAN interfaces is the IPv4 address of sub-interface `system0.0` in the default network instance.

2. Associate the VXLAN interface to a network instance.

   A VXLAN interface can only be associated with one network instance, and a network instance can have only one VXLAN interface.

3. Associate the VXLAN interface to a BGP-EVPN instance.

   The VXLAN interface must be linked to a BGP-EVPN instance so that VXLAN destinations can be dynamically discovered and used to forward traffic.

The following configuration example shows these steps:

```
tunnel-interface vxlan1 {
  // (Step 1) Creation of the tunnel-interface and vxlan-interface
    vxlan-interface 1 {
        type bridged
        ingress {
            vni 1
        }
        egress {
            source-ip use-system-ipv4-address
        }
    }
}
network-instance blue {
    type mac-vrf
    admin-state enable
    description "network instance blue"
    interface ethernet-1/2.1 {
    }
    vxlan-interface vxlan1.1 {
  // (Step 2) Association of the vxlan-interface to the network-instance
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.1
  // (Step 3) Association of the vxlan-interface to the bgp-evpn instance
                evi 1
            }
        }
        bgp-vpn {
            bgp-instance 1 {
                route-distinguisher {
                    route-distinguisher 1.1.1.1:1
                }
                route-target {
                    export-rt target:1234:1
                    import-rt target:1234:1
                }
            }
        }
    }
}
```

When EVPN routes are received with VXLAN encapsulation, the SR Linux creates VXLAN Termination Endpoints (VTEPs) from the EVPN route next-hops, and each VTEP is allocated an index number (per source and destination tunnel IP addresses).

When a VTEP is created in the vxlan-tunnel table and a non-zero index allocated, a tunnel-table entry is also created for the tunnel in the tunnel-table.

If the next hop is not resolved to any route in the network-instance default route-table, the index in the vxlan-tunnel table shows as `0` for the VTEP, and no tunnel-table entry would be created in the tunnel-table for that VTEP.

The following example shows the created vxlan-tunnel entries and tunnel-table entries on reception of IMET routes from three different PEs.

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state tunnel vxlan-tunnel vtep *
    tunnel {
        vxlan-tunnel {
            vtep 10.22.22.2 {
                index 677716962894
 // index allocated per source and destination tunnel IP addresses.
 Index of 0 would mean that 10.22.22.2 is not resolved in the route-table
 and no tunnel-table entry is created.
```

```
                last-change "17 hours ago"
            }
            vtep 10.33.33.3 {
                index 677716962900
                last-change "17 hours ago"
            }
            vtep 10.44.44.4 {
                index 677716962897
                last-change "17 hours ago"
            }
        }
    }

--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance default tunnel-table ipv4
    network-instance default {
        tunnel-table {
            ipv4 {
                tunnel 10.22.22.2/32 type vxlan owner vxlan_mgr id 1 {
  // tunnel table entry for VTEP 10.22.22.2, created
 after the vxlan-tunnel vtep 10.22.22.2
                    next-hop-group 677716962900 // NHG-ID allocated by fib_mgr
                    metric 0
                    preference 0
                    last-app-update "17 hours ago"
                    vxlan {
                        destination-address 10.22.22.2
                        source-address 10.11.11.1
                        time-to-live 255
                    }
                }
                tunnel 10.33.33.3/32 type vxlan owner vxlan_mgr id 3 {
                    next-hop-group 677716962900
                    metric 0
                    preference 0
                    last-app-update "17 hours ago"
                    vxlan {
                        destination-address 10.33.33.3
                        source-address 10.11.11.1
                        time-to-live 255
                    }
                }
                tunnel 10.44.44.4/32 type vxlan owner vxlan_mgr id 2 {
                    next-hop-group 677716962897
                    metric 0
                    preference 0
                    last-app-update "17 hours ago"
                    vxlan {
                        destination-address 10.44.44.4
                        source-address 10.11.11.1
                        time-to-live 255
                    }
                }
            }
        }
    }

--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance default route-table next-hop-group 677716962900
    network-instance default {
        route-table {
            next-hop-group 677716962900 {
                next-hop 0 {
                    next-hop 677716962900
 // NH ID allocated by fib_mgr for the NHG-ID
                    active true
                }
            }
        }
    }

--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance default route-table next-hop 677716962900
    network-instance default {
        route-table {
            next-hop 677716962900 {
 // resolution of the NH ID
                type direct
                ip-address 10.1.2.2
                subinterface ethernet-1/1.1
            }
        }
    }
```

5.1.1 Source and destination VTEP addresses

In the network egress direction, the vxlan-interface/egress/source IP leaf determines the loopback interface that the system uses to source VXLAN packets (outer IP SA). The source IP used in the VXLAN interfaces is the IPv4 address of subinterface `system0.0` in the default network instance.

The egress VTEP (outer IP DA) is determined by EVPN and must be of the same family IPv4 as the configured source IP.

Only unicast VXLAN tunnels are supported (outer IP DA is always unicast), and ingress replication is used to deliver BUM frames to the remote VTEPs in the current release.

In the network ingress direction, the IP DA matches one of the local loopback IP addresses in the default network instance to move the packet to the VNI lookup stage (for loopback interfaces only, not other interfaces in the default network instance, such as IRB subinterfaces). The loopback IP address does not need to match the configured source IP address in the VXLAN interface.

The system can terminate any VXLAN packet with an outer destination IP matching a local loopback address, with no set restriction on the number of IPs.

5.1.2 Ingress or egress VNI

The configured ingress VNI determines the value used by the ingress lookup to find the network instance for a further MAC lookup. The egress VNI is specified by EVPN. For a MAC-VRF, only one egress VNI is supported. The system ignores the value of the I flag on reception. According to RFC 7348, the I flag must be set to 1. However, the system accepts VXLAN packets with I flag set to 0; the I flag is set to 1 on transmission.

5.1.3 VLAN tagging for VXLAN

Outer VLAN tagging is supported (one VLAN tag only), assuming that the egress subinterface in the default network instance uses VLAN tagging.

Inner VLAN tagging is transparent, and no specific handling is needed at network ingress for Layer 2 network instances. Inner VLAN tagging is not supported for VXLAN-originated traffic or VXLAN-terminated traffic in IP-VRF network instances that are BGP-EVPN IFL enabled.

5.1.4 Network instance and interface MTU

No specific MTU checks are done in network instances configured with VXLAN. Make the default network-instance interface MTU large enough to allow room for the VXLAN overhead. If the size of the egress VXLAN packets exceeds the IP MTU of the egress subinterface in the default network instance, the packets are still forwarded. No statistics are collected, other than those for forwarded packets.

IP MTU checks are used only for the overlay domain; that is, for interfaces doing inner packet modifications. IP MTU checks are not done for VXLAN-encapsulated packets on egress subinterfaces of the default network instance (which are in the underlay domain).

5.1.5 Fragmentation for VXLAN traffic

Fragmentation for VXLAN traffic is handled as follows:

- The Don't Fragment (DF) flag is set in the VXLAN outer IP header.
- The TTL of the VXLAN outer IP header is always 255.
- No reassembly is supported for VXLAN packets.

5.2 VXLAN and ECMP

Unicast traffic forwarded to VXLAN destinations can be load-balanced on network (underlay) ECMP links or overlay aliasing destinations.

Network LAGs, that is, LAG subinterfaces in the default network instance, are not supported when VXLAN is enabled on the platform. LAG access subinterfaces on MAC-VRFs are supported.

VXLAN-originated packets support double spraying based on overlay ECMP (or aliasing) and underlay ECMP on the default network instance.

Load-balancing is supported for the following:

- encapsulated Layer 2 unicast frames (coming from a subinterface within the same broadcast domain)
- Layer 3 frames coming from an IRB subinterface

For BUM frames, load balancing operates as follows:

- BUM supports spraying in access LAGs based on a hash. That is, BUM flows received from a VXLAN or a Layer 2 subinterface of a MAC-VRF are sprayed across egress access LAG links.
- BUM does not support spraying in underlay VXLAN next-hops. That is, BUM flows received from VXLAN or a Layer 2 subinterface of a MAC-VRF are sent to a single underlay subinterface.
- BUM VXLAN packets are sent to a single member of the next-hop group (NHG) associated with a specific VXLAN multicast destination. The chosen member is based on a hash of the NHG-ID of the VXLAN destination and the number of links in the NHG.

5.3 VXLAN ACLs

You can configure system filter ACLs to drop incoming VXLAN packets for reasons such as the following:

- the source IP is not recognized
- the destination IP is not an address to be used for termination
- the default destination UDP port is not being used

SR Linux supports logging VXLAN of packets dropped by ACL policies.

A system filter ACL is an IPv4 or IPv6 ACL that is evaluated before tunnel termination has occurred and before interface ACLs have been applied. A system filter can match and drop unauthorized VXLAN tunnel packets before they are decapsulated. When a system filter ACL is created, its rules are evaluated against all transit and terminating IPv4 or IPv6 traffic that is arriving on any subinterface of the router, regardless of where that traffic entered in terms of network instance, subinterface, and so on.

The system filter matches the outer header of tunneled packets; they do not filter the payload of VXLAN tunnels. If the system-filter does not drop the VXLAN-terminated packets, only egress IRB ACLs can match the inner packets. System filters can be applied only at ingress, not egress.

See the *SR Linux Configuration Basics Guide* for information about configuring system filter ACLs.

5.4 QoS for VXLAN tunnels

When the SR Linux receives a terminating VXLAN packet on a subinterface, it classifies the packet to one of eight forwarding classes and one of three drop probabilities (low, medium, or high). The classification is based on the following considerations:

- The outer IP header DSCP is not considered.
- If the payload packet is non-IP, the classified FC is fc0 and the classified drop probability is lowest.
- If the payload packet is IP, and there is a classifier policy referenced by the **qos classifiers vxlan-default** command, that policy is used to determine the FC and drop probability from the header fields of the payload packet.
- If the payload packet is IP, and there is no classifier policy referenced by the **qos classifiers vxlan-default** command, the default DSCP classifier policy is used to determine the FC and drop probability from the header fields of the payload packet.

When the SR Linux adds VXLAN encapsulation to a packet and forwards it out a subinterface, the inner header IP DSCP value is not modified if the payload packet is IP, even if the egress routed subinterface has a DSCP rewrite rule policy bound to it that matches the packet FC and drop probability. The outer header IP DSCP is set to a static value or copied from the inner header IP DSCP. However, this static or copied value is modified by the DSCP rewrite rule policy that is bound to the egress routed subinterface, if the rule policy exists.

Example:

You can specify a classifier policy that applies to ingress packets received from any remote VXLAN VTEP. The policy applies to payload packets after VXLAN decapsulation has been performed.

The following example specifies a VXLAN classifier policy:

```
--{ * candidate shared default }--[  ]--
```

```
# info qos
    qos {
        classifiers {
            vxlan-default p1 {
                traffic-class 1 {
                    forwarding-class fc0
                }
            }
        }
    }
```

See the *SR Linux Quality of Service Guide* for information on configuring QoS.

5.4.1 Configuring a VXLAN classifier policy

Procedure

You can specify a classifier policy that applies to ingress packets received from any remote VXLAN VTEP. The policy applies to payload packets after VXLAN decapsulation has been performed.

Example

The following example specifies a VXLAN classifier policy:

```
--{ * candidate shared default }--[  ]--
# info qos
    qos {
        classifiers {
            vxlan-default p1 {
                traffic-class 1 {
                    forwarding-class fc0
                }
            }
        }
    }
```

5.4.2 Using a DSCP rewrite-rule for VXLAN traffic

About this task

You can configure policies to modify the outer IP DSCP for VXLAN traffic as follows:

- **7220 IXR-D2/D2L/D3/D3L**

  On 7220 IXR-D2/D2L/D3/D3L, if you configure a DSCP rewrite rule policy on the egress routed subinterface, this same policy modifies the outer IP DSCP value for the VXLAN traffic also.

  If no DSCP rewrite policy is configured on the subinterface, then by default, the inner header IP DSCP value is not modified, and the outer header IP DSCP is copied from the inner header IP DSCP.

- **7220 IXR-D4/D5**

  On 7220 IXR-D4/D5, if a DSCP rewrite rule policy is applied to a subinterface, it has no effect on the VXLAN originated traffic. On these platforms, you must use the **qos rewrite-rules vxlan-outer-header-dscp-policy** command to explicitly associate a rewrite policy to the VXLAN originated traffic.

  If no VXLAN DSCP policy is configured on the subinterface, then by default, the inner header IP DSCP value is not modified, and the following platform-specific behavior applies:

- on 7220 IXR-D4: the outer header IP DSCP is copied from the inner header IP DSCP
- on 7220 IXR-D5: the outer header IP DSCP is marked 0

Procedure

On 7220 IXR D4/D5 systems, use the **qos rewrite-rules vxlan-outer-header-dscp-policy** command to apply a rewrite-rule policy for all VXLAN traffic, as shown in the following example:

Example: Apply a DSCP rewrite-rule for VXLAN traffic on 7220 IXR-D4/D5

```
--{ candidate shared default }--[  ]--
# info qos rewrite-rules vxlan-outer-header-dscp-policy
    qos {
        rewrite-rules {
            vxlan-outer-header-dscp-policy vxlan-rewrite-test
            dscp-policy vxlan-rewrite-test


            }
        }
```

5.5 VXLAN statistics collection

You can configure SR Linux to collect statistics for VXLAN tunnels. By default, statistics collection for VXLAN tunnels is disabled.

On 7220 IXR-D2/D3 devices, system resources allocated to statistics collection is distributed among Layer 3 or IRB subinterface statistics (if configured), Layer 2 subinterface statistics, and VXLAN statistics.

When Layer 3 or IRB subinterfaces are configured, enabling VXLAN statistics collection reduces the amount of resources available for collecting Layer 2 subinterface statistics:

- If VXLAN statistics collection is disabled (the default), resources are available for collecting statistics for all supported Layer 2 subinterfaces.
- If VXLAN statistics collection is enabled, resources are shared between Layer 2 subinterfaces and VXLAN tunnels, reducing the Layer 2 subinterface scale.

  If the maximum scale is required for Layer 2 subinterfaces, VXLAN statistics collection must be disabled. Disabling VXLAN statistics collection deallocates the resources for VXLAN statistics collection and allocates them to Layer 2 subinterface statistics collection. The VXLAN statistics are not cleared, however; to clear them, use the appropriate **tools** command. See Clearing VXLAN statistics.
- The amount of resources available for collecting Layer 3 or IRB subinterface statistics does not change if VXLAN statistics collection is enabled or disabled.

5.5.1 Enabling VXLAN statistics collection

Procedure

To enable VXLAN statistics collection, use the following command. Enabling VXLAN statistics collection disables and re-enables statistics collection on the subinterfaces. During this transition, statistics are not counted for the subinterfaces.

After enabling VXLAN statistics collection, if you attempt to configure more than the maximum number of Layer 2 subinterfaces supported with VXLAN statistics, an error message is displayed. If more than the maximum number of Layer 2 subinterfaces with VXLAN statistics are already configured in the system, VXLAN statistics collection cannot be enabled.

Example

```
--{ candidate shared default }--[  ]--
# info tunnel vxlan-tunnel statistics
    tunnel {
        vxlan-tunnel {
            statistics {
                admin-state enable
            }
        }
    }
```

5.5.2 Displaying VXLAN statistics

Procedure

To display statistics for all VXLAN tunnels or for a specified VTEP, use the **info from state** command in candidate or running mode, or the **info** command in state mode.

Example: Display statistics for all VXLAN tunnels

```
--{ running }--[  ]--
# info from state vxlan-tunnel statistics
    vxlan-tunnel {
        statistics {
            in-octets 7296882
            in-packets 83012
            in-discarded-packets 5
            out-octets 7297496
            out-packets 83007
            last-clear 2021-01-29T21:58:40.919Z
        }
    }
```

Example: Display statistics for a specified VTEP

```
--{ running }--[  ]--
# info from state vxlan-tunnel vtep 10.22.22.2
    vxlan-tunnel {
        vtep {
            address 10.22.22.2
            index 677716962894
            last-change 2021-01-29T21:52:34.151Z
            statistics {
                in-octets 7296882
                in-packets 83012
                in-discarded-packets 5
                out-octets 7297496
                out-packets 83007
                last-clear 2021-01-29T21:58:40.919Z
            }
        }
    }
```

5.5.3 Clearing VXLAN statistics

Procedure

You can clear the statistics for all VXLAN tunnels or for a specific VTEP.

Example: Clear statistics for all VXLAN tunnels

```
--{ running }--[  ]--
# tools vxlan-tunnel statistics clear
```

Example: Clear statistics for a specific VTEP

```
--{ running }--[  ]--
# tools vxlan-tunnel vtep 10.22.22.2 clear
```

6 EVPN for VXLAN tunnels for Layer 2

This chapter describes the components of EVPN-VXLAN Layer 2 on SR Linux.

- Basic configuration for EVPN-VXLAN Layer 2
- MAC duplication detection for Layer 2 loop prevention in EVPN
- EVPN Layer 2 multihoming
- Layer 2 proxy-ARP/ND

6.1 Basic configuration for EVPN-VXLAN Layer 2

Basic configuration of EVPN-VXLAN Layer 2 on SR Linux consists of the following:

- **VXLAN interface**

  Contains the ingress VNI of the incoming VXLAN packets associated with the vxlan-interface.
- **MAC-VRF network instance**

  Where the VXLAN interface is attached. Only one VXLAN interface can be attached to a MAC-VRF network instance.
- **BGP-EVPN**

  Enabled in the same MAC-VRF with a minimum configuration of the EVI and the network-instance VXLAN interface associated with it.

  The BGP instance under BGP-EVPN has an encapsulation-type leaf, which is VXLAN by default.

  For EVPN, this configuration determines that the BGP encapsulation extended community is advertised with value VXLAN and the value encoded in the label fields of the advertised NLRIs is a VNI.

  If the route-distinguisher or route-targets/policies are not configured, the required values are automatically derived from the configured EVI as follows:

  – The route-distinguisher is derived as `<ip-address:evi>`, where the `ip-address` is the IPv4 address of the default network instance subinterface system0.0.

  – The route-target is derived as `<asn:evi>`, where the `asn` is the autonomous system configured in the default network instance.

The following example shows a basic EVPN-VXLAN Layer 2 configuration consisting of a VXLAN interface, MAC-VRF network instance, and BGP-EVPN configuration:

```
--{ candidate shared default }--[ ]--
# info
...
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            type bridged
            ingress {
                vni 10
            }
            egress {
                source-ip use-system-ipv4-address
            }
        }
    }

// In the network-instance:

A:dut2#  network-instance blue
--{ candidate shared default }--[ network-instance blue ]--
# info
    type mac-vrf
    admin-state enable
    description "Blue network instance"
    interface ethernet-1/2.1 {
    }
    vxlan-interface vxlan1.1 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.1
                evi 10
            }
        }
        bgp-vpn {
            bgp-instance 1 {
     // rd and rt are auto-derived from evi if this context is not configured
                export-policy pol-def-1
                import-policy pol-def-1
                route-distinguisher {
                    route-distinguisher 64490:200
                }
                route-target {
                    export-rt target:64490:200
                    import-rt target:64490:100
                }
            }
        }
    }
```

6.1.1 EVPN Layer 2 basic routes

EVPN Layer 2 without multihoming includes the implementation of the BGP-EVPN address family and support for the following route types:

- EVPN MAC/IP route (or type 2, RT2)
- EVPN Inclusive Multicast Ethernet Tag route (IMET or type 3, RT3)

The MAC/IP route is used to convey the MAC and IP information of hosts connected to subinterfaces in the MAC-VRF. The IMET route is advertised as soon as BGP-EVPN is enabled in the MAC-VRF; it has the following purpose:

- auto-discovery of the remote VTEPs attached to the same EVI
- creation of a default flooding list in the MAC-VRF so that BUM frames are replicated

Advertisement of the MAC/IP and IMET routes is configured on a per-MAC-VRF basis. The following example shows the default setting **advertise true**, which advertises MAC/IP and IMET routes.

Note: Changing the setting of the **advertise** parameter and committing the change internally flaps the BGP instance.

```
--{ candidate shared default }--[ network-instance blue protocols bgp-evpn bgp-
instance 1 ]--
# info detail
    admin-state enable
    vxlan-interface vxlan1.1
    evi 1
    ecmp 1
    default-admin-tag 0
    routes {
        next-hop use-system-ipv4-address
        mac-ip {
            advertise true
        }
        inclusive-mcast {
            advertise true
        }
    }
```

6.1.2 Creation of VXLAN destinations based on received EVPN routes

The creation of VXLAN destinations of types unicast, unicast Ethernet segment (ES), and multicast for each VXLAN interface is driven by the reception of EVPN routes.

The created unicast, unicast ES, and multicast VXLAN destinations are visible in state. Each destination is allocated a system-wide unique destination index and is an internal next-hop group ID (NHG-ID). The destination indexes for the VXLAN destinations are shown in the following example for destination 10.22.22.4, `vni 1`:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state tunnel-interface vxlan1 vxlan-interface 1 bridge-table unicast-
destinations destination * vni *
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            bridge-table {
                unicast-destinations {
                    destination 10.44.44.4 vni 1 {
                        destination-index 677716962904 // destination index
                        statistics {
                        }
                        mac-table {
                            mac 00:00:00:01:01:04 {
                                type evpn-static
                                last-update "16 hours ago"
                            }
                        }
                    }
                }
            }
        }
    }
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance blue bridge-table mac-table mac 00:00:00:01:01:04
    network-instance blue {
        bridge-table {
            mac-table {
                mac 00:00:00:01:01:04 {
                    destination-type vxlan
                    destination-index 677716962904 // destination index
                    type evpn-static
                    last-update "16 hours ago"
                    destination "vxlan-interface:vxlan1.1 vtep:10.44.44.4 vni:1"
                }
            }
        }
    }
```

The following is an example of dynamically created multicast destinations for a VXLAN interface:

```
--{ [FACTORY] + candidate shared default }--[  ]--
A:dut1# info from state tunnel-interface vxlan1 vxlan-interface 1 bridge-
table multicast-destinations
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            bridge-table {
                multicast-destinations {
                    destination 40.1.1.2 vni 1 {
                        multicast-forwarding BUM
                        destination-index 46428593833
                    }
                    destination 40.1.1.3 vni 1 {
                        multicast-forwarding BUM
                        destination-index 46428593835
                    }
                    destination 40.1.1.4 vni 1 {
                        multicast-forwarding BUM
                        destination-index 46428593829
                    }
                }
            }
        }
    }
```

6.1.3 EVPN route selection

When a MAC is received from multiple sources, the route is selected based on the priority listed in MAC selection. Learned and EVPN-learned routes have equal priority; the latest received route is selected.

When multiple EVPN-learned MAC/IP routes arrive for the same MAC but with a different key (for example, two routes for MAC M1 with different route-distinguishers), a selection is made based on the following priority:

1. EVPN MACs with higher SEQ number
2. EVPN MACs with lower IP next-hop
3. EVPN MACs with lower Ethernet Tag
4. EVPN MACs with lower RD

6.1.4 BGP next hop configuration for EVPN routes

You can configure the BGP next hop to be used for the EVPN routes advertised for a network instance. This next hop is by default the IPv4 address configured in interface `system 0.0` of the default network instance. However, the next-hop address can be changed to any IPv4 address.

The system does not check that the configured IP address exists in the default network instance. Any valid IP address can be used as next hop of the EVPN routes advertised for the network instance, irrespective of its existence in any subinterface of the system. However, the receiver leaf nodes create their unicast, multicast and ES destinations to this advertised next-hop, so it is important that the configured next-hop is a valid IPv4 address that exists in the default network instance.

When the system or loopback interface configured for the BGP next-hop is administratively disabled, EVPN still advertises the routes, as long as a valid IP address is available for the next-hop. However, received traffic on that interface is dropped.

The following example configures a BGP next hop to be used for the EVPN routes advertised for a network instance.

```
--{ candidate shared default }--[ network-instance 1 protocols bgp-evpn bgp-
instance 1 ]--
# info
    routes {
        next-hop 1.1.1.1
        }
    }
```

6.2 MAC duplication detection for Layer 2 loop prevention in EVPN

MAC loop prevention in EVPN broadcast domains is based on the SR Linux MAC duplication feature (see MAC duplication detection and actions), but considers MACs that are learned via EVPN as well. The feature detects MAC duplication for MACs moving among bridge subinterfaces of the same MAC-VRF, as well as MACs moving between bridge subinterfaces and EVPN in the same MAC-VRF, but not for MACs moving from a VTEP to a different VTEP, via EVPN, in the same MAC-VRF.

Also, when a MAC is declared as duplicate, and the **blackhole** configuration option is added to the interface, then not only incoming frames on bridged subinterfaces are discarded if their MAC SA or DA match the blackhole MAC, but also frames encapsulated in VXLAN packets are discarded if their source MAC or destination MAC match the blackhole MAC in the mac-table.

When a MAC exceeds the allowed number of moves, the MAC is moved to a type duplicate irrespective of the type of move: EVPN-to-local, local-to-local, local-to-EVPN. The EVPN application receives an update that advertises the MAC with a higher sequence number, which may trigger the duplication in other nodes. The duplicate MAC can be overwritten by a higher priority type, or flushed by the **tools** command (see Deleting entries from the bridge table).

6.3 EVPN Layer 2 multihoming

SR Linux supports single-active multihoming and all-active multihoming, as defined in RFC 7432. The EVPN multihoming implementation uses the following SR Linux features:

- **system network instance**

  A system network instance container hosts the configuration and state of EVPN for multihoming.

- **BGP network instance**

  The ES model uses a BGP instance from where the RD/RT and export/import policies are taken to advertise and process the multihoming ES routes. Only one BGP instance is allowed, and all the ESes are configured under this BGP instance. The RD/RTs cannot be configured when the BGP instance is associated with the system network instance; however, the operational RD/RTs are still shown in state.

- **ESs**

  An ES has an admin-state (disabled by default) setting that must be toggled to change any of the parameters that affect the EVPN control plane. In particular, the ESes support the following:

  – general and per-ES boot and activation timers

  – manual 10-byte ESI configuration

  – all-active and single-active multihoming modes

  – DF Election algorithm type Default (modulo based) or type Preference

  – configuration of ES and AD per-ES routes next-hop, and ES route originating-IP per ES

  – an AD per ES route is advertised per MAC-VRF, where the route carries the network-instance RD and RT

  – association with an interface that can be of type Ethernet or LAG. When associated with a LAG, the LAG can be static or LACP-based. In case of LACP, the same **system-id**, **system-priority**, and **port-key** settings must be configured on all the nodes attached to the same ES.

- **aliasing load balancing**

  This hashing operation for aliasing load balancing uses the following hash fields in the incoming frames by default:

  – for IP traffic: IP DA and IP SA, Layer 4 source and destination ports, protocol, and VLAN ID

  – for Ethernet (non-IP) traffic: MAC DA and MAC SA, VLAN ID, and Ethertype

  For IPv6 addresses, 32-bit fields are generated by XORing and folding the 128-bit address. The packet fields are supplied as input to the hashing computation.

- **reload-delay timer**

  The reload-delay timer configures an interface to be shut down for a period of time following a node reboot or an IMM reset to avoid black-holing traffic.

6.3.1 EVPN Layer 2 multihoming procedures

EVPN relies on three different procedures to handle multihoming: DF election, split-horizon, and aliasing. DF election is relevant to single-active and all-active multihoming; split-horizon and aliasing are relevant only to all-active multihoming.

- **Designated Forwarder (DF) Election**

  The DF is the leaf that forwards BUM traffic in the ES. Only one DF can exist per ES at a time, and it is elected based on the exchange of ES routes (type 4) and the subsequent DF Election Algorithm (DF Election Alg).

  In single-active multihoming, the non-DF leafs bring down the subinterface associated with the ES.

  In all-active multihoming, the non-DF leafs do not forward BUM traffic received from remote EVPN PEs.

- **split-horizon**

  This is the mechanism by which BUM traffic received from a peer ES PE is filtered so that it is not looped back to the CE that first transmitted the frame. Local bias is applied in VXLAN services, as described in RFC 8365.

- **aliasing**

  This is the procedure by which PEs that are not attached to the ES can process non-zero ESI MAC/IP routes and AD routes and create ES destinations to which per-flow ECMP can be applied.

To support multihoming, EVPN-VXLAN supports two additional route types:

- **ES routes (type 4)**

  Used for ES discovery on all the leafs attached to the ES and DF Election.

  ES routes use an ES-import route target extended community, its value derived from the ESI, so that its distribution is limited to only the leafs that are attached to the ES.

  The ES route is advertised with the DF Election extended community, which indicates the intent to use a specific DF Election Alg and capabilities.

  On reception of the remote ES routes, each PE builds a DF candidate list based on the originator IP of the ES routes. Then, based on the agreed DF Election Alg, each PE elects one of the candidates as DF for each MAC-VRF where the ES is defined.

- **AD route (type 1)**

  Advertised to the leafs attached to an ES. There are two versions of AD routes:

  – **AD per-ES route**

    Used to advertise the multihoming mode (single-active or all-active) and the ESI label, which is not advertised or processed in case of VXLAN. Its withdrawal enables the mass withdrawal procedures in the remote PEs.

  – **AD per-EVI route**

    Used to advertise the availability of an ES in an EVI and its VNI. It is needed by the remote leafs for the aliasing procedures.

  Both versions of AD routes can influence the DF Election. Their withdrawal from a leaf results in removing that leaf from consideration for DF Election for the associated EVI, as long as **ac-df exclude** is configured. The AC-DF capability can be set to **exclude** only if the DF election algorithm type is set to **preference**.

6.3.2 EVPN-VXLAN local bias for all-active multihoming

Local bias for all-active multihoming is based on the following behavior at the ingress and egress leafs:

- At the ingress leaf, any BUM traffic received on an all-active multihoming LAG subinterface associated with an EVPN-VXLAN MAC-VRF is flooded to all local subinterfaces, irrespective of their DF or NDF status, and VXLAN tunnels.
- At the egress leaf, any BUM traffic received on a VXLAN subinterface associated with an EVPN-VXLAN MAC-VRF is flooded to single-homed subinterfaces and multihomed subinterfaces whose ES is not shared with the owner of the source VTEP if the leaf is DF for the ES.

In SR Linux, the local bias filtering entries on the egress leaf are added or removed based on the ES routes, and they are not modified by the removal of AD per EVI/ES routes. This configuration may cause blackholes in the multihomed CE for BUM traffic if the local subinterfaces are administratively disabled.

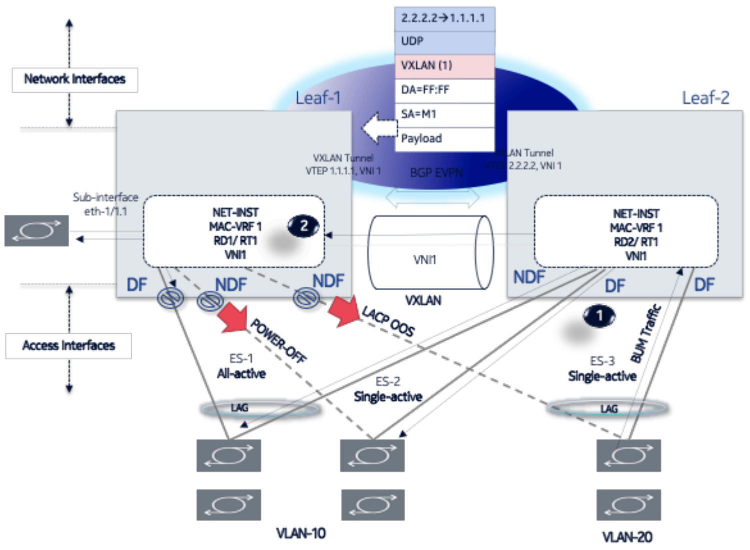
6.3.3 Single-active multihoming

Figure 5: EVPN Layer 2 single-homing configuration shows a single-active ES attached to two leaf nodes. In this configuration, the ES in single-active mode can be configured to do the following:

- associate with an Ethernet interface or a LAG interface, as all-active ESes
- coexist with all-active ESes on the same node, as well as in the same MAC-VRF service
- signal the non-DF state to the CE by using LACP out-of-sync signaling or power off

  Optionally, the ES can be configured not to signal to the CE. When the LACP sync flag or power off is used to signal the non-DF state to the CE or server, all of the subinterfaces are active on the same node; that is, load balancing is not per-service, but rather per-port. This mode of operation is known as EVPN multihoming port-active mode.
- connect to a CE that uses a single LAG to connect to the ES or separate LAG/ports per leaf in the ES

Figure 5: EVPN Layer 2 single-homing configuration



All peers in the ES must be configured with the same multihoming mode; if the nodes are not configured consistently, the `oper-multi-homing-mode` in state is single-active. From a hardware resource perspective, no local-bias-table entries are populated for ESes in single-active mode.

The following features work in conjunction with single-active mode:

- Preference-based DF election/non-revertive option configures ES peers to elect a DF based on a preference value, as well as an option to prevent traffic from reverting back to a former DF node.
- Attachment Circuit-influenced DF Election (AC-DF) allows the DF election candidate list for a network instance to be modified based on the presence of the AD per-EVI and per-ES routes.
- Standby LACP-based or power-off signaling configures how the node's non-DF state is signaled to the multihomed CE.

6.3.3.1 Preference-based DF election/non-revertive option

Preference-based DF election is defined in *draft-ietf-bess-evpn-pref-df* and specifies a way for the ES peers to elect a DF based on a preference value where highest preference-value wins. The *draft-ietf-bess-evpn-pref-df* document also defines a non-revertive mode, so that on recovery of a former DF node, traffic does not revert to the node. This mode is desirable in most cases to avoid double impact on the traffic (failure and recovery).

The configuration requires the command **df-election/algorithm/type preference** and the corresponding **df-election/algorithm/preference-alg/preference-value**. Optionally, you can set non-revertive mode to **true**. See EVPN multihoming configuration example.

Nokia recommends that all of the peers in the ES be configured with the same algorithm type. However, if that is not the case, all the peers fall back to the default algorithm or operational type.

6.3.3.2 Attachment Circuit-influenced DF Election (AC-DF)

Attachment Circuit-influenced DF Election (AC-DF) refers to the ability to modify the DF election candidate list for a network instance based on the presence of the AD per-EVI and per-ES routes. When AC-DF is enabled with the **ac-df include** command, a node cannot become DF if it has the ES subinterface in administratively disabled state. The AC-DF capability is defined in RFC 8584, and it is by default enabled.

Nokia recommends disabling the AC-DF capability, **ac-df exclude** command, when single-active multihoming is used and standby-signaling (**lacp power-off** command) signals the non-DF state to the multihomed CE or server. In this case, the same node must be DF for all the contained subinterfaces. Administratively disabling one subinterface does not cause a DF switchover for the network instance if **ac-df exclude** is configured.

The AC-DF capability is configured with the command **df-election algorithm preference-alg capabilities ac-df**; **include** is the default. See EVPN multihoming configuration example.

6.3.3.3 Standby LACP-based or power-off signaling

Standby LACP-based or power-off signaling is used for cases where the AC-DF capability is excluded, and the DF election port-active mode is configured.

When single-active multihoming is used and all subinterfaces on the node for the ES must be in DF or non-DF state, the multihomed CE does not send traffic to the non-DF node. SR Linux supports two ways of signaling the non-DF state to the multihomed CE: LACP standby or power-off.

Signaling the non-DF state is configured at the interface level, using the command **interface ethernet standby-signaling**, and must also be enabled for a specific ES using the **ethernet-segment df-election interface-standby-signaling-on-non-df** command. See EVPN multihoming configuration example.

The LACP signaling method is only available on LAG interfaces with LACP enabled. When the node is in the non-DF state, it uses an LACP out-of-sync notification (the sync bit is clear in the LACP PDUs) to signal the non-DF state to the CE. The CE then brings down LACP, and the system does not jump to the collecting-distributing state and neither does the peer (because of the out_of_sync state). After the port is out of standby mode, LACP needs to be re-established, after which the forwarding ports need to be programmed.

The power-off signaling is available on Ethernet and LAG interfaces. When the node is in non-DF state, the interface goes operationally down, and the lasers on the Ethernet interfaces (all members in case of a LAG) are turned off. This process brings the CE interface down and avoids any traffic on the link. The interfaces state show `oper-state down` and `oper-down-reason standby-signaling`.

6.3.4 Reload-delay timer

After the system boots, the reload-delay timer keeps an interface shut down with the laser off for a configured amount of time until connectivity with the rest of network is established. When applied to an access multihomed interface, typically an ES interface), this delay can prevent black-holing traffic coming from the multihomed server or CE.

In EVPN multihoming scenarios, if one leaf in the ES peer group is rebooting, coming up after an upgrade or a failure, it is important for the ES interface not to become active until after the node is ready to forward traffic to the core. If the ES interface comes up too quickly and the node has not programmed its forwarding tables yet, traffic from the server is black-holed. To prevent this from happening, you can configure a reload-delay timer on the ES interface so that the interface does not become active until after network connectivity is established.

When a reload-delay timer is configured, the interface port is shut down and the laser is turned off from the time that the system determines the interface state following a reboot or reload of the XDP process until the number of seconds specified in the reload-delay timer elapse.

The reload-delay timer is only supported on Ethernet interfaces that are not enabled with breakout mode. For a multihomed LAG interface, configure the reload-delay timer on all the interface members. The reload-delay timer can be from 1 to 86 400 seconds. There is no default value; if not configured for an interface, there is no reload-delay timer.

Only configure ES interfaces with a non-zero reload-delay timer. Do not configure a reload-delay timer on single-homed interfaces and network interfaces (used to forward VXLAN traffic).

The following example sets the reload-delay timer for an interface to 20 seconds. The timer starts following a system reboot or when the IMM is reconnected, and the system determines the interface state. During the timer period, the interface is deactivated and the port laser is inactive.

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1
    interface ethernet-1/1 {
        admin-state enable
        ethernet {
            reload-delay 20
        }
    }
```

When the reload-delay timer is running, the `port-oper-down-reason` for the port is shown as `interface-reload-timer-active`. The `reload-delay-expires` state indicates the amount of time remaining until the port becomes active. For example:

```
--{ * candidate shared default }--[  ]--
# info from state interface ethernet-1/1
    interface ethernet-1/1 {
        description eth_seg_1
        admin-state enable
        mtu 9232
        loopback-mode false
        ifindex 671742
        oper-state down
        oper-down-reason interface-reload-time-active
        last-change "51 seconds ago"
        linecard 1
        forwarding-complex 0
        vlan-tagging true
        ...
        ethernet {
            auto-negotiate false
            lacp-port-priority 32768
            port-speed 100G
            hw-mac-address 00:01:01:FF:00:15
            reload-delay 20
            reload-delay-expires "18 seconds from now"
            flow-control {
                receive false
                transmit false
            }
        }
    }
```

6.3.5 EVPN multihoming configuration example

The following is an example of a single-active multihoming configuration, including standby signaling power-off, AC-DF capability, and preference-based DF algorithm.

The following configures power-off signaling and the reload delay timer for an interface:

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/21 ethernet
    standby-signaling power-off // needed to signal non-DF state to the CE
    reload-delay 100 // upon reboot, this is required to avoid attracting traffic
from the multi-homed CE until the node is ready to forward.
// The time accounts for the time it takes all network protocols to be up
and forwarding entries ready.
```

The following configures DF election settings for the ES, including preference-based DF election and a preference value for the DF election alg. The **ac-df** setting is set to **exclude**, which disables the AC-DF capability. The **non-revertive** option is enabled, which prevents traffic from reverting back to a former DF node when the node reconnects to the network.

```
--{ * candidate shared default }--[ system network-instance protocols evpn ethernet-
segments bgp-instance 1 ethernet-segment eth_seg_1 ]--
# info
    admin-state enable
    esi 00:01:00:00:00:00:00:00:00:00
    interface ethernet-1/21
    multi-homing-mode single-active
    df-election {
        interface-standby-signaling-on-non-df { // presence container that enables
the standby-signaling for the ES
        }
        algorithm {
            type preference // enables the use of preference based DF election
            preference-alg {
                preference-value 100 // changes the default 32767 to a
specific value
                capabilities {
                    ac-df exclude // turns off the default ac-df capability
                    non-revertive true // enables the non-revertive mode
                }
            }
        }
    }
```

The following shows the state, and consequently the configuration, of an ES for single-active multihoming and indicates the default settings for the algorithm or operational type. Nokia recommends configuring all of the peers in the ES with the same algorithm type. However, if that is not the case, all the peers fall back to the default algorithm.

```
--{ * candidate shared default }--[ system network-instance protocols evpn ethernet-
segments bgp-instance 1 ethernet-segment eth_seg_1 ]--
# info
    admin-state enable
    oper-state up
    esi 00:01:00:00:00:00:00:00:00:00
    interface ethernet-1/21
    multi-homing-mode single-active
    oper-multi-homing-mode single-active // oper mode may be different if not all
the ES peers are configured in the same way
    df-election {
        interface-standby-signaling-on-non-df {
        }
        algorithm {
            type preference
            oper-type preference // if at least one peer in the ES is in type
default, all the peers will fall back to default
            preference-alg {
                preference-value 100
                capabilities {
                    ac-df exclude
                    non-revertive true
                }
            }
        }
    }
    routes {
        next-hop use-system-ipv4-address
        ethernet-segment {
            originating-ip use-system-ipv4-address
        }
    }
    association {
        network-instance blue {
            bgp-instance 1 {
                designated-forwarder-role-last-change "2 seconds ago"
                designated-forwarder-activation-start-time "2 seconds ago"
                designated-forwarder-activation-time 3
                computed-designated-forwarder-candidates {
                    designated-forwarder-candidate 40.1.1.1 {
                        add-time "2 seconds ago"
                        designated-forwarder true
                    }
                    designated-forwarder-candidate 40.1.1.2 {
                        add-time "2 minutes ago"
                        designated-forwarder false
                    }
                }
            }
        }
    }
}
```

To display information about the ES, use the **show system network-instance ethernet-segments** command. For example:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# show system network-instance ethernet-segments eth_seg_1
------------------------------------------------------------------------------------
eth_seg_1 is up, single-active
  ESI      : 00:01:00:00:00:00:00:00:00:00
  Alg      : preference
  Peers    : 40.1.1.2
  Interface: ethernet-1/21
  Network-instances:
     blue
      Candidates : 40.1.1.1 (DF), 40.1.1.2
      Interface : ethernet-1/21.1
------------------------------------------------------------------------------------
Summary
 1 Ethernet Segments Up
 0 Ethernet Segments Down
------------------------------------------------------------------------------------
```

The **detail** option displays more information about the ES. For example:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# show system network-instance ethernet-segments eth_seg_1 detail
====================================================================================
Ethernet Segment
====================================================================================
Name                 : eth_seg_1
40.1.1.1 (DF)
Admin State          : enable              Oper State        : up
ESI                  : 00:01:00:00:00:00:00:00:00:00
Multi-homing         : single-active          Oper Multi-homing : single-active
Interface            : ethernet-1/21
ES Activation Timer  : None
DF Election          : preference          Oper DF Election  : preference

Last Change          : 2021-04-06T08:49:44.017Z
====================================================================================
 MAC-VRF    Actv Timer Rem   DF
eth_seg_1   0                Yes
------------------------------------------------------------------------------------
DF Candidates
------------------------------------------------------------------------------------
Network-instance       ES Peers
blue                   40.1.1.1 (DF)
blue                   40.1.1.2
====================================================================================
```

On the DF node, the **info from state** command displays the following:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state interface ethernet-1/21 | grep oper
        oper-state up
            oper-state not-present
            oper-state up

--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance blue interface ethernet-1/21.1
    network-instance blue {
        interface ethernet-1/21.1 {
            oper-state up
            oper-mac-learning up
            index 6
            multicast-forwarding BUM
        }
    }
```

On the non-DF node, the **info from state** command displays the following:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state interface ethernet-1/21 | grep oper
        oper-state down
        oper-down-reason standby-signaling
            oper-state not-present
            oper-state down
            oper-down-reason port-down

--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance blue interface ethernet-1/21.1
    network-instance blue {
        interface ethernet-1/21.1 {
            oper-state down
            oper-down-reason subif-down
            oper-mac-learning up
            index 7
            multicast-forwarding none
        }
    }
```

6.4 Layer 2 proxy-ARP/ND

Proxy-Address Resolution Protocol (ARP) and proxy-neighbor discovery (ND) are Layer 2 functions supported on MAC-VRF network instances, specified in RFC 9161. Proxy-ARP/ND enables the learning of IP-to-MAC bindings so that leaf nodes can reply to ARP-requests or neighbor solicitations (NS) without having to flood those requests in the BD. The proxy-ARP/ND function supports ARP/ND flooding suppression and security protection against ARP/ND spoofing attacks in large BDs.

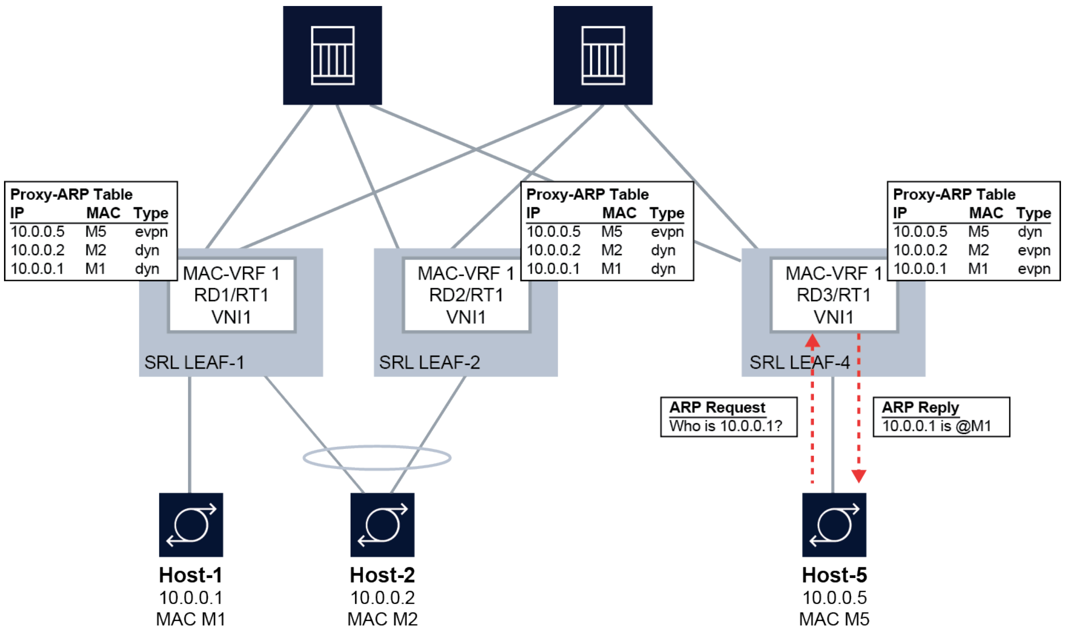
When proxy-ARP/ND is enabled for a MAC-VRF, a table is created that contains entries related to proxy-ARP/ND for the BD. Entries in the proxy-ARP/ND table can be of the following types:

- **dynamic**
  Dynamic IP-MAC entries are learned by snooping ARP and ND messages; requires enabling proxy-ARP/ND and enabling dynamic learning. See Dynamic learning for proxy-ARP/ND.
- **static**
  Static IP-MAC entries are manually provisioned in the proxy-ARP/ND table; requires enabling proxy-ARP/ND and configuring static entries. See Static proxy-ARP entries.
- **EVPN-learned**
  EVPN-learned IP-MAC entries are learned from information received in EVPN MAC/IP (type 2 or RT2) routes from remote PE devices; requires enabling proxy-ARP/ND and configuring **bgp-evpn** on the MAC-VRF. See EVPN learning for proxy-ARP/ND.
- **duplicate**
  Duplicate entries are identified as duplicates by the IP duplication detection procedure. You can configure the criteria that determines whether an entry is a duplicate and optionally inject anti-spoofing MACs in case of duplication. See Proxy-ARP/ND duplicate IP detection.

The proxy-ARP/ND table for the MAC-VRF has a default size of 250 entries. You can modify the table size. See Proxy-ARP/ND table.

The following figure shows how proxy-ARP/ND functions in a BD with an example showing proxy-ARP.

Figure 6: Layer 2 proxy-ARP example



Proxy-ARP Table (SRL LEAF-1)

IP	MAC	Type
10.0.0.5	M5	evpn
10.0.0.2	M2	dyn
10.0.0.1	M1	dyn

Proxy-ARP Table (SRL LEAF-2)

IP	MAC	Type
10.0.0.5	M5	evpn
10.0.0.2	M2	dyn
10.0.0.1	M1	dyn

Proxy-ARP Table (SRL LEAF-4)

IP	MAC	Type
10.0.0.5	M5	dyn
10.0.0.2	M2	evpn
10.0.0.1	M1	evpn

In this example, the SR Linux leaf nodes snoop and learn the local hosts and add dynamic IP-MAC entries for them in the proxy-ARP table for the MAC-VRF. The IP-MAC bindings for the local hosts are advertised in EVPN MAC/IP (type 2 or RT2) routes and installed as EVPN-learned IP-MAC entries in the proxy-ARP table on the remote SR Linux leaf nodes.

For example, SRL LEAF-1 dynamically learns the IP-MAC binding for Host-1 (10.0.0.1-M1) and adds it to the proxy-ARP table as a dynamically learned entry. The IP-MAC binding for Host-1 is advertised in an EVPN MAC/IP route. The remote SRL LEAF-4 installs the IP-MAC binding for Host-1 in its proxy-ARP table as an EVPN-learned entry. When Host-5 sends an ARP-request for 10.0.0.1, SRL LEAF-4 looks it up in the proxy-ARP table and replies with M1. Because Leaf-4 has the 10.0.0.1-M1 binding in its proxy-ARP table, it does not need to flood the request in the BD.

If the lookup is unsuccessful, the ARP-request is re-injected into the datapath and flooded in the BD. Alternatively, this flooding can be suppressed. See Configuring proxy-ARP/ND traffic flooding options.

See RFC 9161 for details about proxy-ARP/ND in EVPN deployments.

6.4.1 Dynamic learning for proxy-ARP/ND

When dynamic learning for proxy-ARP/ND is enabled, all frames coming into bridged subinterfaces of the MAC-VRF that have Ethertype 0x0806 (for proxy-ARP) or ICMPv6 ND (for proxy-ND) are sent to the CPM for learning. This includes ARP-request and ARP-reply (including gratuitous ARP) messages and neighbor solicitation (NS) and neighbor advertisement (NA) messages (for ND). Dynamic entries in the proxy-ARP/ND table are created from both message types. For proxy-ND, dynamic entries are created only from NA messages. Learning an entry is done irrespective of the MAC DA of the ARP or ND packet (unicast or broadcast).

The dynamically learned information in the table is based on the ARP/ND payload MAC SA and IP DA, and not the frame outer MAC SA, although they normally match. A valid MAC SA must be present for the entry to be learned.

In addition to learning the dynamic entry from the ARP-request, ARP-reply, or NA, the system re-injects and forwards messages as follows:

- For received ARP-request or NS messages, the system looks up the requested IP address, and if the lookup is successful, it sends an ARP-reply or NA with the MAC→IP information. If the lookup is not successful, the ARP-request/NS is re-injected and flooded based on the flood list and the configured flooding option, with source squelching. See Configuring proxy-ARP/ND traffic flooding options.

  The re-injected ARP-request or NS keeps all existing non-service-delimiting tags of the original frame. Unicast ARP-requests are replied to if there is an entry in the proxy table. If the lookup is not successful, the frame is forwarded to the MAC DA.
- For ARP-reply/NA messages, the MAC DA (unicast) is looked up in the MAC table. In case of a hit, the frame is re-injected and unicasted based on the MAC table information. If there is no hit, the frame is re-injected and flooded based on the flood list information (with source squelching). The re-injected ARP-reply/NA keeps all existing non-service-delimiting tags of the original frame.
- For ARP/ND frames that are sent to the CPM, the ARP reply/NA is always unicasted to the subinterface on which the ARP-request/NS arrived, even if the MAC itself has not yet been learned in the MAC table.

If a MAC ACL is bound to the ingress of a bridged subinterface (see the *SR Linux ACL and Policy-Based Routing Guide*) and proxy-ARP/ND is enabled in the network instance, the rules in the MAC ACL are applied before extraction to the CPM. Consequently, if the MAC ACL drops an ARP packet, it never reaches the CPM.

Disabling dynamic learning for proxy-ARP/ND causes the dynamically learned entries to be flushed from the proxy-ARP/ND table. You can also set an age timer (default disabled) for dynamic entries, after which they are flushed from the proxy table.

In addition, you can set a timer to refresh dynamic entries. At a configured interval (default never) the system generates ARP-requests/NS with the intent to refresh the proxy entry; if no response is received, another one is attempted.

6.4.1.1 Configuring dynamic learning for proxy-ARP/ND

Procedure

To configure dynamic learning for entries in the proxy-ARP table, enable proxy-ARP and dynamic learning for the MAC-VRF. The same commands exist under `proxy-nd`.

Example

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                admin-state enable
                dynamic-learning {
                    admin-state enable
                    age-time 600
                    send-refresh 200
                }
            }
        }
    }
```

This example also configures the **age-time** and **send-refresh** timers. By default, both timers are disabled.

- The **age-time** specifies in seconds the aging timer for each proxy-ARP/ND entry. When the aging expires, the entry is flushed. The age is reset when a new ARP/GARP/NA for the same IP-MAC binding is received.
- The **send-refresh** timer sends ARP-request/NS messages at the configured time so that the owner of the IP address can reply and therefore refresh its IP-MAC (proxy-ARP/ND) and MAC (FDB) entries.

6.4.2 Static proxy-ARP entries

You can configure static entries in the proxy-ARP/ND table. Static entries have higher priority than snooped dynamic and EVPN entries in the table.

A static proxy-ARP/ND entry requires a static or dynamic MAC entry in the MAC table to become active. The static entries are advertised in MAC/IP routes, with the MAC Mobility extended community following the information associated with the MAC entry (static bit or sequence number).

The system does not validate the MAC addresses configured in static entries.

When **proxy-arp** or **proxy-nd** is disabled, the configured static entries remain in the proxy table, unlike dynamically learned and EVPN-learned entries, which are flushed from the table.

6.4.2.1 Configuring static proxy-ARP entries

Procedure

To configure static proxy-ARP, add entries to the proxy-ARP table.

Example

The following example enables proxy-ARP and configures a static entry in the proxy-ARP table.

Note: The system does not validate MAC addresses specified in static entries.

The same commands exist under `proxy-nd`, although the configured IP address is an IPv6 address.

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                admin-state enable
                static-entries {
                    neighbor 101.1.1.1 {
                        link-layer-address 00:00:64:01:01:01
                    }
                }
            }
        }
    }
```

6.4.3 EVPN learning for proxy-ARP/ND

When proxy-ARP/ND is configured in an EVPN, the PE devices dynamically learn IP-MAC bindings for their local hosts and advertise them in EVPN MAC/IP routes to remote PE devices. The remote PE devices add these IP-MAC bindings to the proxy-ARP/ND table as EVPN-learned entries.

When a host sends an ARP-request or NS for a host on the remote side of the EVPN, if the PE device has the EVPN-learned IP-MAC binding in its proxy-ARP/ND table, it sends back an ARP-reply or NA with the remote host's MAC. If the IP-MAC binding does not exist in the PE device's proxy-ARP/ND table, the ARP-request or NS is flooded in the BD (ARP-request/NS flooding can be optionally disabled). See Figure 6: Layer 2 proxy-ARP example for an illustration of this process.

6.4.3.1 Configuring EVPN learning for proxy-ARP/ND

Procedure

To configure EVPN learning for proxy-ARP/ND, enable proxy-ARP/ND for the MAC-VRF and configure **bgp-evpn** to advertise the IP-MAC bindings in EVPN MAC/IP Advertisement routes and learn new IP-MAC bindings from the imported MAC/IP Advertisement routes.

For deployments that use a multihoming solution, **advertise-arp-nd-only-with-mac-table-entry** must be set to `true` to avoid issues with ESIs. If this setting is not configured, race conditions may result in MAC/IP routes advertised with ESI = 0 and later with non-zero ESIs.

Example: Advertise local MACs and local MAC-IP pairs

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                admin-state enable
                }
            }
        }
    }
```

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bgp-evpn bgp-instance 1 routes bridge-table
    network-instance MAC-VRF-1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        bridge-table {
                            mac-ip {
                                advertise true
                            }
                        }
                    }
                }
            }
        }
    }
```

Example: Advertise local MAC-IP pairs only with a local MAC in the MAC table

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                admin-state enable
                }
            }
        }
    }
```

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bgp-evpn bgp-instance 1 routes bridge-table
    network-instance MAC-VRF-1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        bridge-table {
                            mac-ip {
                                advertise-arp-nd-only-with-mac-table-entry true
                            }
                        }
                    }
                }
            }
        }
    }
```

Note: EVPN MAC/IP routes advertised for Layer 2 and Layer 3 differ in that Layer 3 ARP entries are always advertised in non-zero IP MAC/IP routes, while Layer 2 proxy-ARP/ND entries are advertised only when there is a local MAC in the MAC table, because the proxy-ARP/ND entry must be active.

Enabling **advertise-arp-nd-only-with-mac-table-entry** is recommended for Layer 2 proxy-ARP/ND multihoming deployments, because the MAC delete process is handled separately from the proxy-ARP/ND delete process. If the MAC delete process happens first, the proxy-ARP/ND entry is advertised with ES = 0 right away, which flushes the MAC on the ES peer.

6.4.4 Configuring proxy-ARP/ND traffic flooding options

Procedure

If a lookup in the proxy-ARP/ND table is unsuccessful, by default the system re-injects the ARP-request into the data path and floods it in the BD.

For proxy-ARP, you can configure the following options for how ARP frames are flooded into the EVPN network. The default for both of these options is **true**.

- `unknown-arp-req`

This option configures whether unknown broadcast ARP-requests are flooded to EVPN destinations. Unknown in this context means the lookup in the proxy-ARP/ND table was unsuccessful. Non-broadcast ARP-requests are not affected by this option.

- `gratuitous-arp`
  This option configures whether Gratuitous ARP (GARP) requests or replies are flooded to EVPN destinations. GARPs are ARP messages where the sender's IP address matches the target's IP address. Normally the MAC DA is a broadcast address.

For proxy-ND, you can configure the following options for how NS and NA frames are flooded into the EVPN network. The default for these options is **true**.

- `unknown-neighbor-advertise-host`
  This option configures whether to flood NA replies for type `host` into the EVPN network.
- `unknown-neighbor-advertise-router`
  This option configures whether to flood NA replies for type `router` into the EVPN network.
- `unknown-neighbor-solicitation`
  This option configures whether to flood NS messages (with source squelching) into the EVPN network

Example: Disable EVPN flooding for unknown broadcast ARP-requests and GARPs

The following example disables flooding to EVPN destinations for both unknown broadcast ARP-requests and GARPs:

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp evpn flood
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                evpn {
                    flood {
                        unknown-arp-req false
                        gratuitous-arp false
                    }
                }
            }
        }
    }
```

Example: Disable EVPN flooding for Neighbor Advertisements

The following example disables flooding to EVPN destinations for NA replies for type `router` and type `host`:

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-nd evpn flood
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-nd {
                evpn {
                    flood {
                        unknown-neighbor-advertise-router false
                        unknown-neighbor-advertise-host false
                    }
                }
            }
        }
    }
```

6.4.5 Proxy-ARP/ND duplicate IP detection

Proxy-ARP/ND duplicate IP detection is a security mechanism described in RFC 9161 to detect ARP/ND-spoofing attacks. In an ARP/ND-spoofing attack, an attacker sends false ARP/ND messages into a BD, with the goal of associating the attacker's IP address with a target host and directing traffic for that host to the attacker.

The proxy-ARP/ND duplicate IP detection feature monitors changes to active entries in the proxy-ARP/ND table. When an IP move occurs, for example, IP1→MAC1 is replaced by IP1→MAC2 in the table, a **monitoring-window** timer is started for a default 3 minutes. If a specified number of IP moves (default 5) is detected before the **monitoring-window** timer expires, the IP is considered to be a duplicate.

When the system detects an IP move in the proxy table, for example, IP1→MAC1 changing to IP1→MAC2, it places the IP1→MAC2 proxy table entry in pending-confirmation state for a maximum of 30 seconds. During the pending-confirmation period, the ARP/ND entry is inactive, and a confirm-message is unicast to MAC1. If no reply from MAC1 is received during the pending-confirmation period, the IP1→MAC2 entry is confirmed as legitimate and becomes active. If a reply from MAC1 is received, then MAC2 is sent a confirm-message. If MAC2 replies, an additional confirm-message is sent to MAC1. If both MAC1 and MAC2 keep replying to the confirm-messages, it triggers the duplicate IP detection procedure for IP1, because the number of IP moves exceeds the maximum allowed during the monitoring window.

When an IP is detected as a duplicate, the proxy table cannot be updated with new dynamic or EVPN-learned entries for the same IP, although you can configure a static entry for the IP. The duplicate IP is subject to this restriction until a configured **hold-down-time** expires, of which the default is 9 minutes, after which the entry for the IP is removed from the proxy table, and the monitoring process for the IP is restarted.

Anti-Spoofing MAC

You can configure an Anti-Spoofing MAC (AS-MAC). If an AS-MAC is configured, the system associates the duplicate IP with the AS-MAC in the proxy-ARP/ND table. A GARP or unsolicited-NA message with IP1→AS-MAC is sent to the local CEs, and an EVPN MAC/IP route with IP1→AS-MAC is sent to the remote PEs. This process updates the ARP/ND caches on the CEs in the BD so that CEs in the BD use the AS-MAC as MAC DA when sending traffic to IP1.

If you configure the `static-blackhole true` option, the AS-MAC is installed in the MAC table as a blackhole MAC, which discards incoming frames with a MAC source or destination matching the AS-MAC.

6.4.5.1 Configuring proxy-ARP/ND duplicate IP detection

Procedure

To configure proxy-ARP/ND duplicate IP detection, set the following options:

- `monitoring-window`
  This is the number of minutes the system monitors a proxy-ARP/ND table entry following an IP move (default 3 minutes).
- `num-moves`
  This is the maximum number of IP moves a proxy-ARP/ND table entry can have during the `monitoring-window` before the IP is considered duplicate (default 5 IP moves).
- `hold-down-time`

This is the number of minutes from the time an IP is declared duplicate to the time the IP is removed from the proxy-ARP/ND table (default 9 minutes).

- `anti-spoof-mac`
  If configured, this replaces the MAC of the duplicate IP in the proxy-ARP/ND table. The AS-MAC is advertised in EVPN to remote PEs.
- `static-blackhole`
  If this option is set to `true`, this installs the AS-MAC in the MAC table as a blackhole MAC, causing incoming frames with a MAC source or destination matching the AS-MAC to be discarded.

Example: Configure proxy-ARP duplicate IP detection

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp ip-duplication
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                ip-duplication {
                    monitoring-window 5
                    num-moves 7
                    hold-down-time 10
                    anti-spoof-mac 00:CA:FE:CA:FE:08
                    static-blackhole true
                }
            }
        }
    }
```

Example: Configure proxy-ND duplicate IP detection

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-nd ip-duplication
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-nd {
                ip-duplication {
                    monitoring-window 5
                    num-moves 7
                    hold-down-time 10
                    anti-spoof-mac 00:CA:FE:CA:FE:08
                    static-blackhole true
                }
            }
        }
    }
```

6.4.5.2 Configuring processing for proxy-ARP/ND probe packets

Procedure

ARP Request and NS packets that have 0.0.0.0 or :: as the sender's IP address are used as probes by the IPv4/IPv6 duplicate address detection (DAD) function. Proxy-ARP/ND identifies these probe packets and can either process them locally, or forward them into the MAC-VRF (flooding them into the MAC-VRF) so that they reach all the hosts in the BD. You can configure whether the probe packets are processed locally or flooded to the MAC-VRF.

Example: Configure flooding for proxy-ARP probe packets

The following example configures local processing for ARP probe messages. When set to `false`, ARP probe messages are flooded to the remote nodes if unknown-ARP-requests are configured to be flooded.

By default, local processing is set to `true`, which causes ARP probe messages used by the hosts for DAD to be processed, replied if a proxy-ARP entry is hit, or reinjected into the data path.

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1
    network-instance MAC-VRF-1 {
        type mac-vrf
        bridge-table {
            proxy-arp {
                process-arp-probes false
            }
        }
    }
```

Example: Configure flooding for Neighbor Solicitation DAD messages

The following example configures local processing for NS DAD messages. When set to `false`, NS DAD messages are flooded to the remote nodes if `unknown-neighbor-solicitation` is configured, so that unknown NS messages are flooded.

By default, local processing is set to `true`, which causes NS DAD messages used by the hosts for DAD to be processed, replied if a proxy-ND entry is hit, or reinjected into the data path.

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-nd {
                process-dad-neighbor-solicitations false
            }
        }
    }
```

6.4.5.3 Displaying proxy-ARP/ND duplicate IP detection information

Procedure

You can use a **show** command to display information about duplicate IPs detected in the proxy-ARP table. A similar **show** command exists for proxy-ND.

Example

To display duplicate IP information from the proxy-ARP table for all MAC-VRFs:

```
--{ running }--[  ]--
# show network-instance * bridge-table proxy-arp ip-duplication duplicate-entries
---------------------------------------------------------------------------------------------
IP-duplication in network instance mac-vrf-1
---------------------------------------------------------------------------------------------
Monitoring window       : 3 minutes
Number of moves allowed : 5
Hold-down-time          : 10 seconds
```

```
Anti-Spoof-MAC          : 00:DE:AD:00:00:01 (Static-blackhole)
---------------------------------------------------------------------------------------------
Duplicate entries in network instance mac-vrf-1
----------------------------------------------------------------------------------------------
+--------------+-----------------------------+---------------------------+------------------+
|  Neighbor    |     MAC Address             | Detect Time               | Hold down time   |
|              |                             |                           | remaining        |
+==============+=============================+===========================+==================+
| 10.10.10.1   |  00:DE:AD:00:00:01          | 2021-12-11T12:48:24.000Z  | 10               |
| 10.10.10.2   |  00:DE:AD:00:00:01          | 2021-12-11T12:48:24.000Z  | 9                |
| 10.10.10.3   |  00:DE:AD:00:00:01          | 2021-12-11T12:48:24.000Z  | 10               |
+--------------+-----------------------------+---------------------------+------------------+
---------------------------------------------------------------------------------------------
IP-duplication in network instance mac-vrf-2
---------------------------------------------------------------------------------------------
...
---------------------------------------------------------------------------------------------
Total Duplicate IPs       : 4  Total 4 Active
---------------------------------------------------------------------------------------------
```

6.4.6 Proxy-ARP/ND table

When you enable proxy-ARP/ND for a MAC-VRF, this creates a table containing proxy-ARP/ND entries learned dynamically by snooping ARP/ND messages, configured statically, or learned from EVPN MAC/IP routes from remote PE nodes. By default, this table can contain up to 250 entries. You can configure the size of this table to be from 1 to 8192 entries.

The system generates a log event when the size of the table reaches 90% of the maximum size and when the table reaches 95% of the maximum size. When the table reaches 100% of its maximum size, entries for an IP can be replaced, that is, a different MAC can be learned and added for the IP, but no new IP entries can be added to the table, regardless of the type (dynamic, static, or EVPN-learned).

6.4.6.1 Configuring the proxy-ARP/ND table size

Procedure

By default, the proxy-ARP or proxy-ND table can contain up to 250 entries of all types (dynamic, static, EVPN, duplicate). You can increase or decrease the maximum size of the table. If you configure the table size to be lower than the number of entries it currently contains, the system stops and restarts the proxy-ARP/ND application, causing the non-static entries to be flushed from the table.

Example: Configure the proxy-ARP table size

The following example configures the size of the proxy-ARP table for a MAC-VRF:

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp table-size
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                table-size 125
            }
        }
    }
```

Example: Configure the proxy-ND table size

The following example configures the size of the proxy-ND table for a MAC-VRF:

```
--{ candidate shared default }--[  ]--
# info network-instance MAC-VRF-1 bridge-table proxy-arp table-size
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-nd {
                table-size 125
            }
        }
    }
```

6.4.6.2 Clearing entries from the proxy-ARP/ND table

Procedure

You can use **tools** commands to clear dynamic and duplicate entries from the proxy-ARP table. You can clear all dynamic and duplicate entries or you can clear specific entries.

Example: Clear dynamic entries from the proxy-ARP table

To clear dynamic entries from the proxy-ARP table:

```
--{ running }--[  ]--
# tools network-instance MAC-VRF-1 bridge-table proxy-arp dynamic delete-all
# tools network-instance MAC-VRF-1 bridge-table proxy-arp dynamic entry 10.10.10.1 delete-
ip
```

Example: Clear duplicate entries from the proxy-ARP table

To clear duplicate entries from the proxy-ARP table:

```
--{ running }--[  ]--
# tools network-instance MAC-VRF-1 bridge-table proxy-arp duplicate delete-all
# tools network-instance MAC-VRF-1 bridge-table proxy-arp duplicate entry 10.10.10.1
 delete-ip
```

Example: Clear dynamic entries from the proxy-ND table

To clear dynamic entries from the proxy-ND table:

```
--{ running }--[  ]--
# tools network-instance MAC-VRF-1 bridge-table proxy-nd dynamic delete-all
# tools network-instance MAC-VRF-1 bridge-table proxy-nd dynamic entry 10.10.10.1 delete-
ip
```

Example: Clear duplicate entries from the proxy-ND table

To clear duplicate entries from the proxy-ND table:

```
--{ running }--[  ]--
# tools network-instance MAC-VRF-1 bridge-table proxy-nd duplicate delete-all
# tools network-instance MAC-VRF-1 bridge-table proxy-nd duplicate entry 10.10.10.1
 delete-ip
```

6.4.7 EVPN ARP/ND extended community flags

RFC 9047 defines an extended community (EC) for EVPN MAC/IP advertisement routes that carry flags related to ARP or ND. This EC includes flags that can indicate to a remote PE router whether an ARP/ND entry learned via EVPN belongs to a host or a router, or if the address is an anycast address. Information from the EC flags can allow remote PE routers configured for proxy ARP/ND to reply to local ARP or ND requests with correct information.

On SR Linux, the RFC 9047 EC is not carried in EVPN MAC/IP advertisement routes by default. You can configure SR Linux to advertise the RFC 9047 EC along with the MAC/IP routes advertised for local static, dynamic, and duplicate proxy-ARP/ND entries. When you enable SR Linux to advertise the RFC 9047 EC, you can also configure how the flags are advertised in EVPN MAC/IP routes and how the flags are set in replies to NS messages for EVPN entries.

This feature is not supported along with IRB interfaces.

The feature must be configured consistently on all PEs attached to the same BD, and in particular attached to the same ES and BD.

At the BGP level, only one EVPN ARP/ND EC is expected to be received along with an EVPN MAC/IP advertisement route. If more than one EVPN ARP/ND EC is received, the router considers only the first one on the list for processing purposes on the PE.

At the EVPN level, the flags are ignored if they come in a MAC/IP route without IP.

RFC 9047 extended community flags

RFC 9047 defines the following flags for the EVPN ARP/ND extended community. The flags field is the third octet of the EC.

- **router flag (R flag) (bit 7 in the flags field)**
  If set, the R flag indicates the IPv6→MAC pair in the MAC/IP advertisement route belongs to an IPv6 router; if not set, the IPv6→MAC pair belongs to a host.
- **override flag (O flag) (bit 6 in the flags field)**
  The O flag is normally set to 1 by the egress router when advertising IPv6→MAC pairs, set to 0 when IPv6 anycast is enabled for the BD or interface, and ignored when the received MAC/IP advertisement route belongs to an IPv4→MAC pair.

  Note: Anycast IPv6 addresses are not supported in SR Linux proxy-ND tables.

  The ingress PE installs the ND entry with the received O flag and always uses this O flag value when replying to an NS message for the IPv6 address.

  The O flag is conveyed in EVPN advertisements and advertised based on the received NA messages, and is installed in the proxy-ND entries. However, other than transferring the information between ND and EVPN, the O flag does not change any of the selection of ND entries in proxy-ND.
- **immutable flag (I flag) (bit 4 in the flags field)**
  If set, the I flag indicates the IP→MAC pair in the MAC/IP advertisement route is a configured ARP/ND entry; the IP address in the EVPN MAC/IP advertisement route can only be bound together with the MAC address specified in the same route.

  The I flag applies to IPv6 and IPv4 entries and is set in MAC/IP advertisements for static proxy-ARP/ND entries.

6.4.7.1 Enabling advertisement of the RFC 9047 extended community

When you enable advertisement of the RFC 9047 extended community, the EC is advertised along with the MAC/IP routes advertised for local static, dynamic, and duplicate proxy-ARP/ND entries. Enabling this feature also allows you to control how the EC is processed and how ARP/ND entries are selected based on the I flag.

The following example enables SR Linux to advertise the RFC 9047 EC:

```
--{ candidate shared default }--[  ]--
# info network-instance m1 protocols bgp-evpn bgp-instance 1 routes bridge-table
    network-instance m1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        bridge-table {
                            mac-ip {
                                advertise-arp-nd-extended-community true
                            }
                        }
                    }
                }
            }
        }
    }
```

By default, the `advertise-arp-nd-extended-community` command is set to `false`. When this command is set to `true`, the R, I, and O flags are advertised and processed as described in the following sections:

- Advertising and processing for the R flag
- Advertising and processing for the I flag
- Advertising and processing for the O flag

Advertising and processing for the R flag

After you enable SR Linux to advertise the RFC 9047 EC, you can control the advertisement of proxy-ND EVPN entries and the reply to NS messages received for those EVPN entries by configuring the `advertise-neighbor-type` command. For example:

```
--{ candidate shared default }--[  ]--
# info network-instance m1 bridge-table
    network-instance m1 {
        bridge-table {
            proxy-nd {
                evpn {
                    advertise-neighbor-type router-host
                }
            }
        }
    }
```

The `advertise-neighbor-type` command has three options: `host`, `router`, and `router-host`. The effect this command has on how SR Linux processes the R flag for static, dynamic, EVPN, and duplicate proxy-ND entries is described in Table 2: Effect the advertise-neighbor-type setting has on proxy ND entries.

Note the following:

- System-generated unsolicited NA messages for duplicate entries carry R = 0 or R = 1, and are flooded to EVPN if allowed by the configured flooding options.
- The R flag in the RFC 9047 EC is ignored for proxy-ARP entries.

Table 2: Effect the advertise-neighbor-type setting has on proxy ND entries

For this proxy-ND entry type	The advertise-neighbor-type setting has this effect
Static proxy-ND entries	Static proxy-ND entries are added with an R = 1 flag if configured with `neighbor <ipv6-address> type router`. NS messages received for the entry are replied to with the configured R flag. The entries are advertised in EVPN if the command `advertise-neighbor-type` is configured. If the configuration for an entry is changed from `host` to `router`, or from `router` to `host`, an unsolicited NA message is triggered containing the new flag. This unsolicited NA message is or isn't flooded into EVPN based on the configured flooding options.
Dynamic proxy-ND entries	When `proxy-nd/dynamic-learning/admin-state enable` is configured, the router learns dynamic entries from NA messages; the corresponding R flag is also learned and added to the proxy-ND table. Subsequent NS messages received for the entry are replied to with the corresponding R flag. The learned dynamic entries are advertised in EVPN depending configured `advertise-neighbor-type` setting: • `router` advertises dynamic and static entries for which R = 1 • `host` advertises dynamic and static entries for which R = 0 • `router-host` advertises dynamic and static entries for which R = 0 or R = 1 The `router-host` option is available only if `advertise-arp-nd-extended-community true` is configured. The R flag of the entry is then propagated in the R bit of the extended community for the route. • the default value is `advertise-neighbor-type router`
EVPN proxy-ND entries	The setting for `advertise-neighbor-type` controls the proxy-ND dynamic/static/duplicate entries that are advertised in EVPN: • `router` only advertises in EVPN dynamic entries learned with R = 1, or static entries configured as `router` • `host` only advertises in EVPN dynamic entries learned with R = 0, or static entries configured as `host`

For this proxy-ND entry type	The advertise-neighbor-type setting has this effect
	• `router-host` advertises in EVPN dynamic and static entries irrespective of the router flag, and with the correct R flag in each case. For an EVPN entry, the command adds the Router flag depending on the R flag coming in the RFC 9047 EC. • irrespective of the configured option, the absence of the RFC 9047 EC in a received MAC/IP route creates a proxy-ND entry of type R = 1 (router) The setting for `advertise-neighbor-type` must be consistent on all nodes for the same service.
Duplicate proxy-ND entries	Duplicate proxy-ND entries are treated as host or router entries as far as EVPN advertisement and response to NS messages is concerned. If `advertise-neighbor-type host` is configured, the duplicate entry is treated as host. If `advertise-neighbor-type router` or `router-host` is configured, the duplicate entry is treated as router.

Advertising and processing for the I flag

When the `advertise-arp-nd-extended-community true` command is configured, SR Linux advertises the I flag as follows:

- any static proxy-ARP/ND entry is advertised with I = 1
- duplicate entries with AS-MAC are advertised with I = 1 (in addition to O = 1 and R = 0 or 1, based on configuration)
- setting of the I flag is independent of the Static bit associated with the FDB entry; the I flag is only used with proxy-ARP/ND advertisements

On reception, the I flag is processed as follows:

When SR Linux receives an EVPN MAC/IP advertisement route containing an IP→MAC and I = 1, it installs the IP→MAC entry in the proxy-ARP/ND table as an immutable binding, overriding any existing non-immutable binding for the same IP→MAC. MAC mobility does not consider the I flag.

For multiple routes to the same IP, proxy-ARP/ND selection operates according the following rules:

- local immutable ARP/ND entries (static) are preferred first
- EVPN immutable ARP/ND entries are preferred next
- regular existing ARP/ND selection applies third

If SR Linux receives multiple EVPN MAC/IP advertisement routes with the I flag set to 1 with the same IP but a different MAC address, a route is selected using these selection rules.

If SR Linux originates an EVPN MAC/IP advertisement route containing an IP→MAC and I = 1, it also originates the route with the MAC mobility EC sticky/static flag set if the MAC is static. In this case, the MAC→IP binding is immutable, and it cannot move. If an update for the same immutable and static IP→MAC is received from a different PE, one of the two routes is selected.

There are no changes for generation of confirmation messages or GARP or unsolicited-NA messages when a new entry is learned.

Advertising and processing for the O flag

When the `advertise-arp-nd-extended-community true` command is configured, SR Linux advertises the O flag as follows:

- For dynamic entries, the setting for the O flag is taken from what was learned and added to the proxy-ND table.
- For static and duplicate entries, O = 1.

On reception, the O flag is processed as follows:

- The O flag is stored in the proxy-ND table, and is used when replying to a received NS message.
- For link-local neighbor advertisements, generated for solicitations to the local chassis MAC, NA messages with O = 0 are not learned.
- If an EVPN route is received without the RFC 9047 EC, the entry is installed with the default value of O = 1.
- For solicited or unsolicited NA messages, the O flag is O = 1 or O = 0, depending on the flag in the proxy-ND entry that was previously learned.

6.4.8 Displaying proxy-ARP/ND information

Procedure

You can use **show** commands to display information about the contents of the proxy-ARP table. Similar **show** commands exist for proxy-ND.

Example: Display the entire proxy-ARP table

To display all entries in the proxy-ARP table for a MAC-VRF:

```
--{ candidate shared default }--[  ]--
# show network-instance MAC-VRF-1 bridge-table proxy-arp all
-------------------------------------------------------------------------------------------------
Proxy-ARP table of network instance MAC-VRF-1
-------------------------------------------------------------------------------------------------
+-------------+--------------------+------------+---------+--------+-------------------------+
| Neighbor    | MAC Address        |   Type     | State   | Aging  | Last Update             |
+=============+====================+============+=========+========+=========================+
| 10.10.10.1  | 00:CA:FE:CA:FE:01  |  dynamic   | active  | 300    | 2021-12-11T12:48:24.000Z |
| 10.10.10.2  | 00:CA:FE:CA:FE:02  |  evpn      | active  | N/A    | 2021-12-11T12:48:24.000Z |
| 10.10.10.3  | 00:CA:FE:CA:FE:03  |  duplicate | active  | N/A    | 2021-12-11T12:48:24.000Z |
+-------------+--------------------+------------+---------+--------+-------------------------+
Total Static Neighbors     :    0 Total    0 Active
Total Dynamic Neighbors    :    1 Total    1 Active
Total Evpn Neighbors       :    1 Total    1 Active
Total Duplicate Neighbors  :    1 Total    1 Active
Total Neighbors            :    3 Total    3 Active
-------------------------------------------------------------------------------------------------
```

The output of this command indicates the total number of entries of each type, as well as the number that are active (replies are sent for received ARP-requests). For an entry to be considered active in the proxy-ARP table, it must have a corresponding entry in the MAC table of the type listed in the following table:

Table 3: MAC table entry types required for proxy-ARP table entry types to be active

Proxy-ARP table entry type	MAC table entry type
Dynamic	Learned
Dynamic	Static
Dynamic	EVPN
Static	Learned
Static	Static
EVPN	EVPN (irrespective of the ESI)
EVPN	Static or dynamic matching the EVPN ESI
Duplicate	–

If a proxy-ARP table entry has a corresponding entry in the MAC table of a type not listed in this table, then the proxy-ARP table entry is considered inactive, so no replies are sent for received ARP-requests. If the MAC-VRF is not active, its proxy-ARP table entries are not active as well.

Example: Display a specific entry in the proxy-ARP table

```
--{ candidate shared default }--[  ]--
# show network-instance MAC-VRF-1 bridge-table proxy-arp neighbor 10.10.10.1
----------------------------------------------------------------------------
Proxy-ARP table of network instance MAC-VRF-1
----------------------------------------------------------------------------
Neighbor                 : 10.10.10.1
MAC Address              : 00:CA:FE:CA:FE:01
Type                     : dynamic
Programming Status       : Success
Aging                    : 300
Last Update              : 2021-12-11T12:48:24.000Z
Duplicate Detect time    : N/A
Hold down time remaining: N/A
----------------------------------------------------------------------------
```

Example: Display a summary of the proxy-ARP table

```
--{ candidate shared default }--[  ]--
# show network-instance * bridge-table proxy-arp summary
------------------------------------------------------
Network Instance Proxy-ARP Table Summary
------------------------------------------------------
Network Instance: MAC-VRF-1
Total Static Neighbors     :    0 Total    0 Active
Total Dynamic Neighbors    :    1 Total    1 Active
Total Evpn Neighbors       :    1 Total    1 Active
Total Duplicate Neighbors  :    1 Total    1 Active
Total Neighbors            :    3 Total    3 Active
Maximum Entries  : 250
Warning Threshold: 95% (237)
Clear Warning    : 90% (225)
-------------------------------------------------------
```

```
Network Instance: MAC-VRF-2
Total Static Neighbors     :    1 Total    1 Active
Total Dynamic Neighbors    :    1 Total    1 Active
Total Evpn Neighbors       :    0 Total    0 Active
Total Duplicate Neighbors  :    0 Total    0 Active
Total Neighbors            :    2 Total    2 Active
Maximum Entries  : 250
Warning Threshold: 95% (237)
Clear Warning    : 90% (225)
--------------------------------------------------------
--------------------------------------------------------
Total Static Neighbors     :    1 Total    1 Active
Total Dynamic Neighbors    :    2 Total    2 Active
Total Evpn Neighbors       :    1 Total    1 Active
Total Duplicate Neighbors  :    1 Total    1 Active
Total Neighbors            :    5 Total    5 Active
---------------------------------------------------------
```

6.4.9 Displaying proxy-ARP/ND statistics

Procedure

To display proxy-ARP statistics, use the **info from state** command in candidate or running mode or the **info** command in state mode. You can display system-wide statistics or statistics for a specific MAC-VRF network instance.

Example: Display proxy-ARP statistics

The following example displays proxy-ARP statistics for a MAC-VRF network instance:

```
--{ candidate shared default }--[  ]--
# info from state network-instance MAC-VRF-1 bridge-table proxy-arp statistics
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-arp {
                statistics {
                    total-entries 0
                    active-entries 0
                    in-active-entries 0
                    pending-entries 0
                    neighbor-origin {
                        origin
                        total-entries 0
                        active-entries 0
                        in-active-entries 0
                        pending-entries 0
                    }
                }
            }
        }
```

Example: Display proxy-ND statistics

The following example displays proxy-ND statistics for a MAC-VRF network instance:

```
--{ candidate shared default }--[  ]--
# info from state network-instance MAC-VRF-1 bridge-table proxy-nd statistics
    network-instance MAC-VRF-1 {
        bridge-table {
            proxy-nd {
                statistics {
                    total-entries 0
                    active-entries 0
                    in-active-entries 0
                    pending-entries 0
                    neighbor-origin {
                        origin
                        total-entries 0
                        active-entries 0
                        in-active-entries 0
                        pending-entries 0
                    }
                }
            }
        }
```

7 EVPN for VXLAN tunnels for Layer 3

This chapter describes the components of EVPN-VXLAN Layer 3 on SR Linux.

7.1 EVPN Layer 3 basic configuration

The basic EVPN Layer 3 configuration model builds on the model for EVPN routes described in EVPN for VXLAN tunnels for Layer 2, extending it with an additional route type to support inter-subnet forwarding: EVPN IP prefix route (or type 5, RT5).

The EVPN IP prefix route conveys IP prefixes of any length and family that need to be installed in the ip-vrfs of remote leaf nodes. The EVPN Layer 3 configuration model has two modes of operation:

- **asymmetric IRB**

  This is a basic mode of operation EVPN Layer 3 using IRB interfaces. The term asymmetric refers to how there are more lookups performed at the ingress leaf than at the egress leaf.

  While the asymmetric model allows inter-subnet-forwarding in EVPN-VXLAN networks in a very simple way, it requires the instantiation of all the MAC-VRFs of all the tenant subnets on all the leafs attached to the tenant. Because all the MAC-VRFs of the tenant are instantiated, FDB and ARP entries are consumed for all the hosts in all the leafs of the tenant.

  These scale implications may make the symmetric model a better choice for data center deployment.

- **symmetric IRB**

  The term symmetric refers to how MAC and IP lookups are needed at ingress, and MAC and IP lookups are performed at egress. As opposed to asymmetric, symmetric IRB implies the same number of lookups at ingress and egress.

  SR Linux support for symmetric IRB includes the prefix routing model using RT5s as in RFC 9136, including the IFL IP-VRF-to-IP-VRF model (IFL model) :

Compared to the asymmetric model, the symmetric model scales better because hosts' ARP and FDB entries are installed only on the directly attached leafs and not on all the leafs of the tenant.

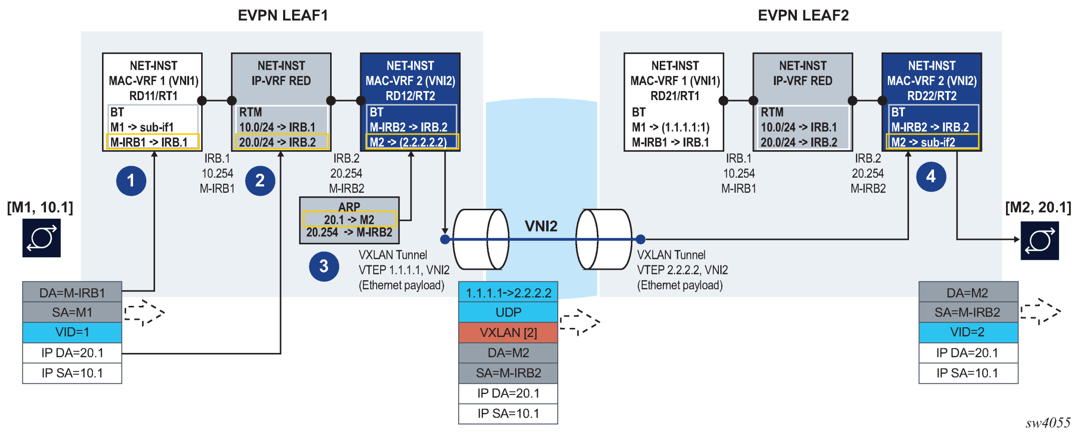
The following sections show asymmetric and symmetric IFL forwarding configurations.

7.1.1 Asymmetric IRB

The asymmetric IRB model is the basic Layer 3 forwarding model when the IP-VRF (or default network-instance) interfaces are all IRB-based. The asymmetric model assumes that all the subnets of a tenant are local in the IP-VRF/default route table, so there is no need to advertise EVPN RT5 routes.

The following figure shows the asymmetric IRB model.

Figure 7: EVPN-VXLAN Layer 3 asymmetric forwarding



In this example, the host with IP address 10.1 (abbreviation of 10.0.0.1) sends a unicast packet with destination 20.1 (a host in a different subnet and remote leaf). Because the IP DA is in a remote subnet, the MAC DA is resolved to the local default gateway MAC, M-IRB1. The frame is classified for MAC lookup on MAC-VRF 1, and the result is IRB.1, which indicates that an IP DA lookup is required in IP-VRF red.

An IP DA longest-prefix match in the route table yields IRB.2, a local IRB interface, so an ARP and MAC DA lookup are required in the corresponding IRB interface and MAC-VRF bridge table.

The ARP lookup yields M2 on MAC-VRF 2, and the M2 lookup yields VXLAN destination [VTEP:VNI]=[2.2.2.2:2]. The routed packet is encapsulated with the corresponding inner MAC header and VXLAN encapsulation before being sent to the wire.

In the asymmetric IRB model, if the ingress leaf routes the traffic via the IRB to a local subnet, and the destination MAC is aliased to multiple leaf nodes in the same ES destination, SR Linux can do load balancing on a per-flow basis.

EVPN Leaf 1 in Figure 7: EVPN-VXLAN Layer 3 asymmetric forwarding has the following configuration:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info
    interface ethernet-1/2 {
        admin-state enable
        vlan-tagging true
        subinterface 1 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1
                    }
                }
            }
        }
    }
    interface irb0 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                address 10.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
        subinterface 2 {
            ipv4 {
                admin-state enable
                address 20.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
    }
    network-instance ip-vrf-red {
        type ip-vrf
        interface irb0.1 {
        }
        interface irb0.2 {
        }
    }
    network-instance mac-vrf-1 {
        type mac-vrf
        admin-state enable
        interface ethernet-1/2.1 {
        }
        interface irb0.1 {
        }
        vxlan-interface vxlan1.1 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.1
                    evi 1
                }
            }
            bgp-vpn {
            }
        }
    }
    network-instance mac-vrf-2 {
        type mac-vrf
        admin-state enable
        interface irb0.2 {
        }
        vxlan-interface vxlan1.2 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.2
                    evi 2
                }
            }
            bgp-vpn {
            }
        }
    }
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            type bridged
            ingress {
                vni 1
            }
        }
        vxlan-interface 2 {
            type bridged
            ingress {
                vni 2
            }
        }
    }
```

EVPN Leaf 2 in Figure 7: EVPN-VXLAN Layer 3 asymmetric forwarding has the following configuration:

```
--{ [FACTORY] + candidate shared default }--[  ]--
A:LEAF2# info
    interface ethernet-1/12 {
        admin-state enable
        vlan-tagging true
        subinterface 1 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 2
                    }
                }
            }
        }
    }
    interface irb0 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                address 10.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
        subinterface 2 {
            ipv4 {
                admin-state enable
                address 20.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
    }
    network-instance ip-vrf-red {
        type ip-vrf
        interface irb0.1 {
        }
        interface irb0.2 {
        }
    }
    network-instance mac-vrf-1 {
        type mac-vrf
        admin-state enable
        interface irb0.1 {
        }
        vxlan-interface vxlan1.1 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.1
                    evi 1
                }
            }
            bgp-vpn {
            }
        }
    }
    network-instance mac-vrf-2 {
        type mac-vrf
        admin-state enable
        interface irb0.2 {
        }
        vxlan-interface vxlan1.2 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.2
                    evi 2
                }
            }
            bgp-vpn {
            }
        }
    }
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            type bridged
            ingress {
                vni 1
            }
        }
        vxlan-interface 2 {
            type bridged
            ingress {
                vni 2
            }
        }
    }
```

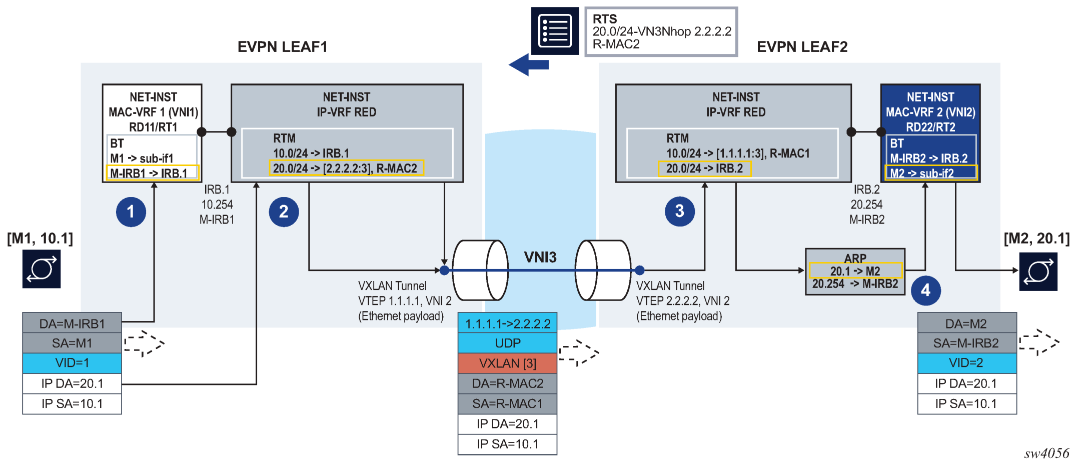
7.1.2 Symmetric IRB interface-less ip-vrf-to-ip-vrf model

SR Linux support for symmetric IRB is based on the prefix routing model using RT5s, and implements the EVPN interface-less (EVPN IFL) IP-VRF-to-IP-VRF model.

In the EVPN IFL model, all interface and local routes (static, ARP-ND, BGP, and so on) are automatically advertised in RT5s without the need for any export policy. Interface host and broadcast addresses are not advertised. On the ingress PE, RT5s are installed in the route table as indirect with owner BGP-EVPN.

The following figure shows the forwarding for symmetric IRB.

Figure 8: EVPN-VXLAN Layer 3 symmetric forwarding



As in the asymmetric model, the frame is classified for bridge-table lookup on the MAC-VRF and processed for routing in IP-VRF red.

In contrast to the asymmetric model, a longest prefix match does not yield a local subnet, but a remote subnet reachable via [VTEP:VNI]=[2.2.2.2:3] and inner MAC DA R-MAC2. SR Linux supports the EVPN interface-less (EVPN IFL) model, so that information is found in the IP-VRF route-table directly; a route lookup on the IP-VRF red route-table yields a VXLAN tunnel and VNI.

- Packets are encapsulated with an inner Ethernet header and the VXLAN tunnel encapsulation.
- The inner Ethernet header uses the system-mac as MAC SA, and the MAC advertised along with the received RT5 as MAC DA. No VLAN tag is transmitted or received in this inner Ethernet header.

At the egress PE, the packet is classified for an IP lookup on the IP-VRF red (the inner Ethernet header is ignored).

The inner and outer IP headers are updated as follows:

- inner IP header TTL is decremented
- outer IP header TTL is set to 255
- outer DSCP value is marked as described in QoS for VXLAN tunnels
- no IP MTU check is performed before or after encapsulation

Because SR Linux supports EVPN IFL, the IP lookup in the IP-VRF red route-table yields a local IRB interface.

Subsequent ARP and MAC lookups provide the information to send the routed frame to subinterface 2.

EVPN Leaf 1 in Figure 8: EVPN-VXLAN Layer 3 symmetric forwarding has the following configuration:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info
    interface ethernet-1/2 {
        admin-state enable
        vlan-tagging true
        subinterface 1 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1
                    }
                }
            }
        }
    }
    interface irb0 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                address 10.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
    }
    network-instance ip-vrf-red {
        type ip-vrf
        interface irb0.1 {
        }
        vxlan-interface vxlan1.3 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.3
                    evi 3
                }
            }
            bgp-vpn {
            }
        }
    }
    network-instance mac-vrf-1 {
        type mac-vrf
        admin-state enable
        interface ethernet-1/2.1 {
        }
        interface irb0.1 {
        }
        vxlan-interface vxlan1.1 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.1
                    evi 1
                }
            }
            bgp-vpn {
            }
        }
    }
    tunnel-interface vxlan1 {
        vxlan-interface 1 {
            type bridged
            ingress {
                vni 1
            }
        }
        vxlan-interface 3 {
            type routed
            ingress {
                vni 3
            }
        }
    }
```

EVPN Leaf 2 in Figure 8: EVPN-VXLAN Layer 3 symmetric forwarding has the following configuration:

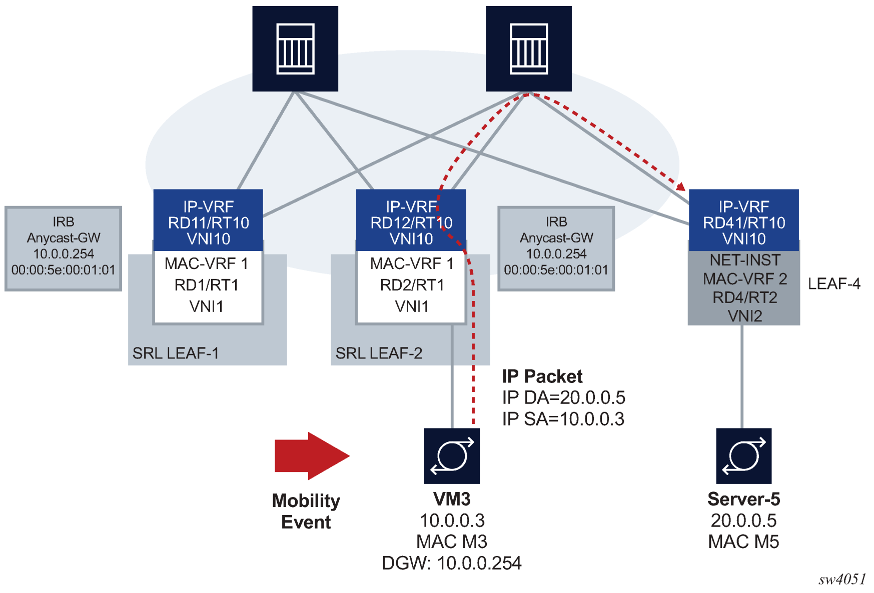
```
--{ [FACTORY] + candidate shared default }--[  ]--
# info
    interface ethernet-1/12 {
        admin-state enable
        vlan-tagging true
        subinterface 2 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 2
                    }
                }
            }
        }
    }
    interface irb0 {
        subinterface 2 {
            ipv4 {
                admin-state enable
                address 20.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
    }
    network-instance ip-vrf-red {
        type ip-vrf
        interface irb0.2 {
        }
        vxlan-interface vxlan1.3 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.3
                    evi 3
                }
            }
            bgp-vpn {
            }
        }
    }
    network-instance mac-vrf-2 {
        type mac-vrf
        admin-state enable
        interface ethernet-1/12.2 {
        }
        interface irb0.2 {
        }
        vxlan-interface vxlan1.2 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.2
                    evi 2
                }
            }
            bgp-vpn {
            }
        }
    }
    tunnel-interface vxlan1 {
        vxlan-interface 2 {
            type bridged
            ingress {
                vni 2
            }
        }
        vxlan-interface 3 {
            type routed
            ingress {
                vni 3
            }
        }
    }
```

7.2 Anycast gateways

Anycast gateways (anycast-GWs) are a common way to configure IRB subinterfaces in DC leaf nodes. Configuring anycast-GW IRB subinterfaces on all leaf nodes of the same BD avoids tromboning for upstream traffic from hosts moving between leaf nodes.

The following figure shows an example anycast-GW IRB configuration.

Figure 9: Anycast-GW IRB configuration



Anycast-GWs allow configuration of the same IP and MAC addresses on the IRB interfaces of all leaf switches attached to the same BD; for example, SRL LEAF-1 and SRL LEAF-2 in the preceding figure. This feature optimizes the south-north forwarding because a host's default gateway always belongs to the connected leaf, irrespective of the host moving between leaf switches; for example VM3 moving from SRL LEAF-1 to SRL LEAF-2 in the figure.

When an IRB subinterface is configured as an anycast-GW, it must have one IP address configured as **anycast-gw**. The subinterface may or may not have other non-anycast-GW IP addresses configured.

To simplify provisioning, an option to automatically derive the anycast-GW MAC is supported, as described in RFC 9135. The auto-derivation uses a virtual-router-id similar to MAC auto-derivation in RFC 5798 (VRRP). Anycast-GWs use a default virtual-router-id of 01, if not explicitly configured. Because only one anycast-gw-mac per IRB sub-interface is supported, the anycast-gw-mac for IPv4 and IPv6 is the same in the IRB sub-interface.

The following is an example configuration for an anycast-GW subinterface:

```
// Configuration Example of an anycast-gw IRB sub-interface

[interface irb1 subinterface 1 ]
A:leaf-1/2# info
  ipv4 {
    admin-state enable
    address 10.0.0.254/24 {
      primary true
      anycast-gw true
    }
  }
  anycast-gw {
    virtual-router-id 2
  }

// State Example of an anycast-gw IRB sub-interface

[interface irb1 subinterface 1 ]
A:leaf-1/2# info from state
```

```
  ipv4 {
    address 10.0.0.254/24 {
      primary true
      anycast-gw true
    }
  }
  anycast-gw {
    virtual-router-id 2
    anycast-gw-mac 00:00:5e:00:01:02
    anycast-gw-mac-origin auto-derived
  }
```

The **anycast-gw true** command designates the associated IP address as an anycast-GW address of the subinterface and associates the IP address with the **anycast-gw-mac** address in the same sub-interface. ARP requests or NSs received for the anycast-GW IP address are replied using the **anycast-gw-mac** address, as opposed to the regular system-derived MAC address. Similarly, CPM-originated GARPs or unsolicited NAs sent for the anycast-GW IP address use the **anycast-gw-mac** address as well. Packets routed to the IRB use the **anycast-gw-mac** as the SA in Ethernet header.

All IP addresses of the IRB subinterface and their associated MACs are advertised in MAC/IP routes with the static flag set. The non-anycast-GW IPs are advertised along with the interface hardware MAC address, and the anycast-GW IP addresses along with the anycast-gw-mac address.

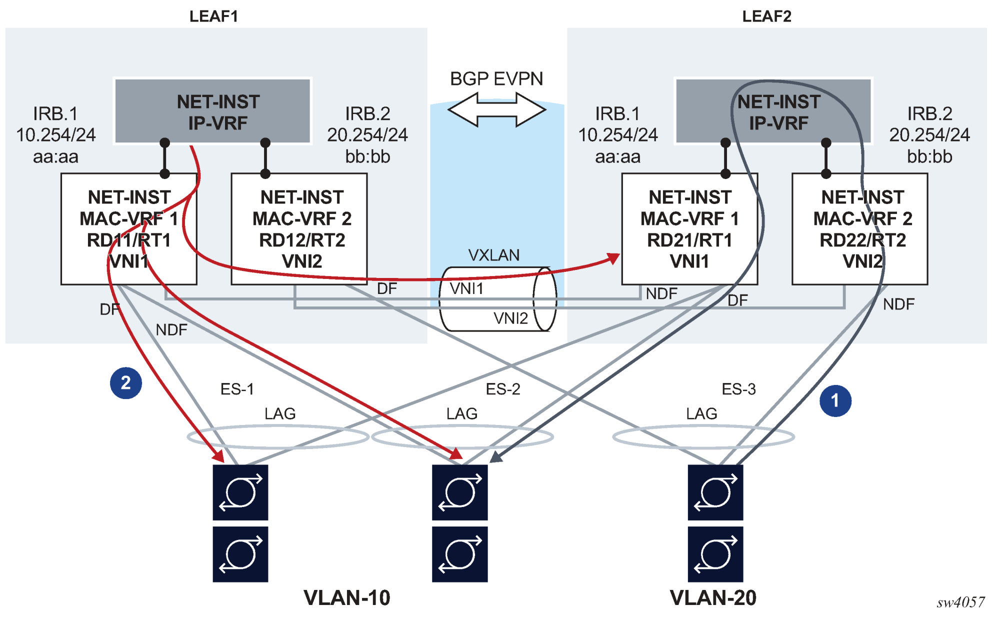
In addition, the **anycast-gw true** command makes the system skip the ARP/ND duplicate-address-detection procedures for the anycast-GW IP address.

7.3 EVPN Layer 3 multihoming and anycast gateways

In an EVPN Layer 3 scenario, all IRB interfaces facing the hosts must have the same IP address and MAC; that is, an anycast-GW configuration. This avoids inefficiencies for all-active multihoming or speeds up convergence for host mobility.

The use of anycast-GW along with all-active multihoming is shown in the following figure:

Figure 10: EVPN-VXLAN Layer 3 model – multihoming and anycast gateway



In this example:

1. Routed unicast traffic is always routed to the directly connected leaf (no tromboning).
2. BUM traffic sent from the IRB interface is sent to all DF and NDF subinterfaces, similar to BUM entering a subinterface.

   This applies to:

   - system-generated unknown or bcast
   - ARP requests and GARPs
   - unicast with unknown MAC DA

When a host connected to ES-3 sends a unicast flow to be routed in the IP-VRF, the flow must be routed in the leaf receiving the traffic, irrespective of the server hashing the flow to Leaf-1 or Leaf-2. To do this, the host is configured with only one default gateway, 20.254/24. When the host ARPs for it, it does not matter if the ARP request is sent to Leaf-1 or Leaf-2. Either leaf replies with the same anycast-GW MAC, and when receiving the traffic either leaf can route the packet.

This scenario is supported on IP-VRF network instances and the default network instance.

Note:
When IRB subinterfaces are attached to MAC-VRF network instances with all-active multihoming ESs, set the `arp timeout` / `neighbor-discovery stale-time` settings on the IRB subinterface to a value that is 30 seconds lower than the `age-time` configured in the MAC-VRF. This setting avoids transient packet loss situations triggered by the MAC address of an active ARP/ND entry being removed from the MAC table.

7.4 EVPN Layer 3 host route mobility

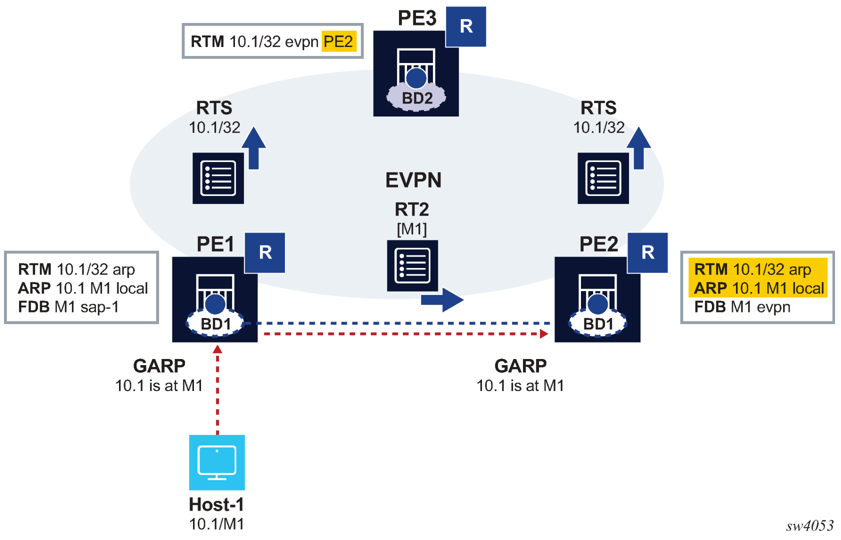
EVPN host route mobility refers to the procedures that allow the following:

- learning ARP/ND entries out of unsolicited messages from hosts
- generating host routes out of those ARP/ND entries
- refreshing the entries when mobility events occur within the same BD

EVPN host route mobility is part of basic EVPN Layer 3 functionality as defined in RFC 9135.

The following figure EVPN host route mobility.

Figure 11: EVPN host route mobility



In Figure 11: EVPN host route mobility a host is attached to PE1, so all traffic from PE3 to that host must be forwarded directly to PE1. When the host moves to PE2, all the tables must be immediately updated so that PE3 sends the traffic to PE2. EVPN host route mobility works by doing the following:

1. snooping and learning ARP/ND entries for hosts upon receiving unsolicited GARP or NA messages
2. creating host routes out of those dynamic ARP/ND entries. These routes only exist in the control plane and are not installed in the forwarding plane to avoid FIB exhaustion with too many /32 or /128 host routes.
3. advertising the locally learned ARP/ND entries in MAC/IP routes so that ARP/ND caches can be synchronized across leaf nodes
4. advertising the host routes as IP prefix routes
5. triggering ARP/ND refresh messages for changes in the ARP/ND table or MAC table for a specific IP, which allows updating the tables without depending on the ARP/ND aging timers (which can be hours long)

The following configuration enables an anycast-GW IRB subinterface to support mobility procedures:

```
// Example of the configuration of host route mobility features

--{ candidate shared default }--[ interface irb1 subinterface 1 ]--
A:-# info
```

```
    ipv4 {
        admin-state enable
        address 10.0.0.254/24 {
            anycast-gw true
            primary
        }
        arp {
            learn-unsolicited true
            host-route {
                populate static
                populate dynamic
            }
            evpn {
                advertise static
                advertise dynamic
            }
        }
    }
    ipv6 {
        admin-state enable
        address 200::254/64 {
            anycast-gw true
            primary
        }
        neighbor-discovery {
            learn-unsolicited true
            host-route {
                populate static
                populate dynamic
            }
            evpn {
                advertise static
                advertise dynamic
            }
        }
    }
    anycast-gw {
    }
--{ candidate shared default }--[ interface irb1 subinterface 1 ]--
# info from state
    admin-state enable
    ip-mtu 1500
    name irb1.1
    ifindex 1082146818
    oper-state up
    last-change "a minute ago"
    ipv4 {
        allow-directed-broadcast false
        address 10.0.0.254/24 {
            anycast-gw true
            origin static
        }
        arp {
            duplicate-address-detection true
            timeout 14400
            learn-unsolicited true
            host-route {
                populate static
                populate dynamic
                }
            }
        }
    }
    ipv6 {
        address 200::254/64 {
            anycast-gw true
            origin static
            status unknown
        }
        address fe80::201:1ff:feff:42/64 {
            origin link-layer
            status unknown
        }
        neighbor-discovery {
            duplicate-address-detection true
            reachable-time 30
            stale-time 14400
            learn-unsolicited both
            host-route {
                populate static
                populate dynamic
            }
        }
        router-advertisement {
            router-role {
                current-hop-limit 64
                managed-configuration-flag false
                other-configuration-flag false
                max-advertisement-interval 600
                min-advertisement-interval 200
                reachable-time 0
                retransmit-time 0
                router-lifetime 1800
            }
        }
    }
    anycast-gw {
        virtual-router-id 1
        anycast-gw-mac 00:00:5E:00:01:01
        anycast-gw-mac-origin vrid-auto-derived
    }
...
```

In this configuration, when `learn-unsolicited` is set to `true`, the node processes all solicited and unsolicited ARP/ND flooded messages received on subinterfaces (no VXLAN) and learns the corresponding ARP/ND entries as dynamic. By default, this setting is `false`, so only solicited entries are learned by default.

The advertisement of EVPN MAC/IP routes for the learned ARP entries must be enabled/disabled by configuration; it is disabled by default. In the example above, this is configured with the `advertise dynamic` and `advertise static` settings.

The creation of host routes in the IP-VRF route table out of the dynamic or static ARP entries can be enabled/disabled by configuration; it is disabled by default. In the example above, this is configured with the `host-route populate dynamic` and `host-route populate static` settings.

By default, the ARP/ND host routes are not installed in the data path. You can configure `host-route populate dynamic datapath-programming true` to change this default behavior so that the ARP/ND host routes are installed in the data path. This configuration is required when the ARP/ND host routes are leaked. See "Network instance route leaking" in the *SR Linux Configuration Basics Guide*.

The dynamic ARP entries are refreshed without any extra configuration. The system sends ARP requests for the dynamic entries to make sure the hosts are still alive and connected.

7.5 EVPN IFL interoperability with EVPN IFF

By default, the SR Linux EVPN IFL (interface-less) model, described in Symmetric IRB interface-less ip-vrf-to-ip-vrf model, does not interoperate with the EVPN IFF (interface-ful) model, as supported on Nuage WBX devices. However, it is possible to configure the SR Linux EVPN IFL model to interoperate with the EVPN IFF model.

To do this, configure the **advertise-gateway-mac** command for the IP-VRF network instance. When this command is configured, the node advertises a MAC/IP route using the following:

- gateway-mac for the IP-VRF (that is, the system-mac)
- RD/RT, next-hop, and VNI of the IP-VRF where the command is configured
- null IP address, ESI or Ethernet Tag ID

Nuage WBX devices support two EVPN Layer 3 IPv6 modes: IFF unnumbered and IFF numbered. The SR Linux interoperability mode enabled by the **advertise-gateway-mac** command only works with Nuage WBX devices that use the EVPN IFF unnumbered model. This is because the EVPN IFL and EVPN IFF unnumbered models both use the same format in the IP prefix route, and they differ only in the additional MAC/IP route for the gateway-mac. The EVPN IFL and EVPN IFF numbered models have different IP prefix route formats, so they cannot interoperate.

The following example enables interoperability with the Nuage EVPN IFF unnumbered model:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info from state network-instance protocols bgp-vpn
  bgp-evpn {
      bgp-instance 1 {
          admin-state enable
          vxlan-interface vxlan1.2
          routes {
              route-table {
                  mac-ip {
                      advertise-gateway-mac true
                  }
              }
          }
      }
  }
```

7.6 VIP discovery for redundant servers

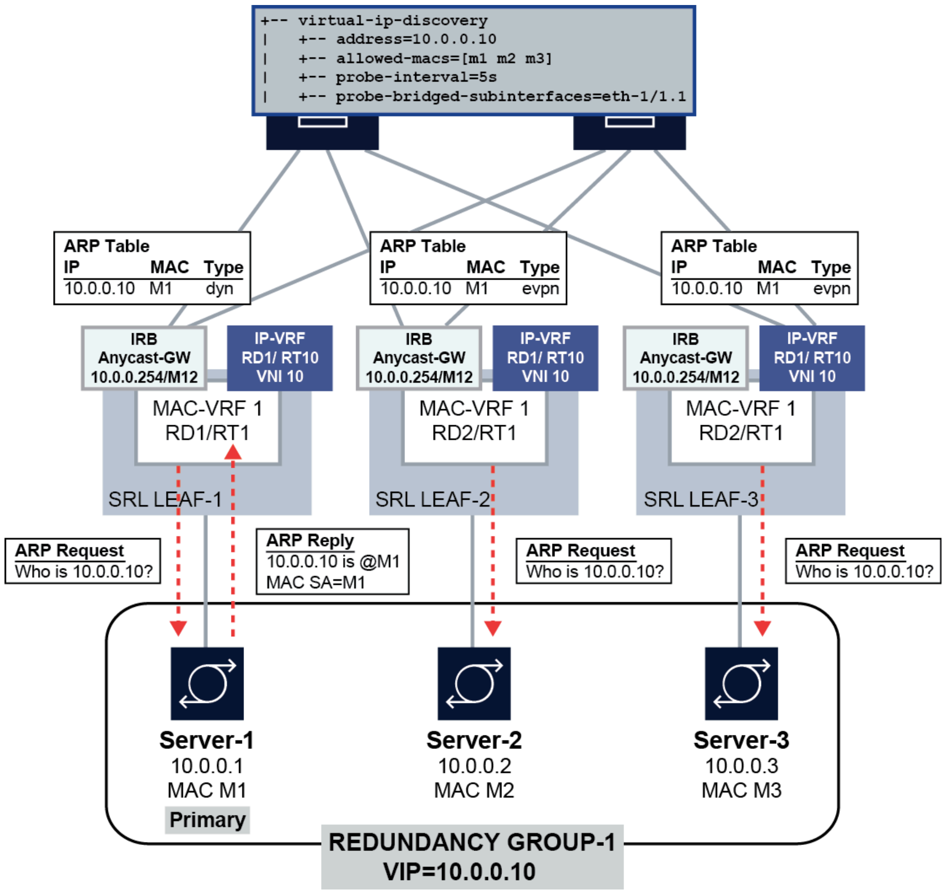
In data centers, it is common to have clusters of servers sharing the same IP address and working in an active-standby mode, so that only one of the servers in the cluster is active at a time. This shared IP address is known as a virtual IP (VIP) address. SR Linux can discover which server in the cluster owns the VIP address.

ARP requests (NS for IPv6) from a leaf node or gratuitous ARP (unsolicited NA for IPv6) from a server are used to discover which host owns the VIP. Among the servers sharing the VIP, only the active one either sends a gratuitous ARP (or unsolicited NA) or replies to the ARP request (NS) from the leaf node. If a server does not send a gratuitous ARP, you can optionally configure SR Linux to send ARP requests periodically at a specified probe interval.

The leaf nodes create entries in their ARP tables that map the VIP to the MAC address of the active server. You can optionally configure a list of allowed MAC addresses so that the ARP/ND entry for the VIP is created only if the reply from the server comes from one of the MACs on the allowed list.

The following figure shows a configuration where three SR Linux devices in a MAC-VRF are connected to a redundant server group.

Figure 12: VIP discovery for the active server in a server group



```
+-- virtual-ip-discovery
|   +-- address=10.0.0.10
|   +-- allowed-macs=[m1 m2 m3]
|   +-- probe-interval=5s
|   +-- probe-bridged-subinterfaces=eth-1/1.1
```

ARP Table		
IP	MAC	Type
10.0.0.10	M1	dyn

ARP Table		
IP	MAC	Type
10.0.0.10	M1	evpn

ARP Table		
IP	MAC	Type
10.0.0.10	M1	evpn

In this configuration, the SR Linux devices (SRL LEAF-1/2/3) start sending ARP requests for the VIP 10.10.10.10 on the ethernet 1/1.1 subinterface every 5 seconds until the ARP entry for the VIP/VMAC is learned. The ARP entry is created only if the MAC address matches one of the allowed MACs: M1, M2, or M3.

Initially, Server-1 is the primary server in the server group, and it responds to the ARP request from Leaf-1, causing the server's MAC address M1 to be added to the ARP table on the SR Linux devices for VIP 10.10.10.10. If Server-2 (MAC M2) becomes the primary server, it sends out a gratuitous ARP message or responds to an ARP request from LEAF-2, which triggers an update of the ARP table on Leaf-2 and a MAC/IP route update with 10.10.10.10/M2. When Leaf-1 receives the update, it updates its ARP table immediately with the new entry.

VIP discovery is supported on single-homed and multihomed subinterfaces. For all-active multihoming, and based on the local-bias behavior, the ARP/ND probes are sent to all ES subinterfaces irrespective of their DF state. When the active server replies, the MAC/IP route is synchronized in the other ES peers based on the functionality described in EVPN Layer 3 host route mobility.

7.6.1 Configuring VIP discovery

Procedure

To configure VIP discovery for an IRB interface, you specify the bridged subinterfaces that can be sent probe messages (ARP requests) for the VIP, optionally a probe interval that determines how often the probe messages are sent, and optionally a list of allowed MAC addresses that indicate the MAC addresses of the servers in the group.

Example: Configure VIP discovery for IPv4

```
--{ candidate shared default }--[  ]--
# info interface irb1 subinterface 1 ipv4 arp
 interface irb1 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                arp {
                    virtual-ipv4-discovery {
                        address 10.10.10.10 {
                            probe-bridged-subinterfaces [
                                ethernet-1/1.1
                            ]
                            probe-interval 5
                            allowed-macs [
                                00:14:9C:78:E2:E1
                                00:14:9C:78:E2:E2
                                00:14:9C:78:E2:E3
                            ]
                        }
                    }
                }
            }
        }
    }
```

Example: Configure VIP discovery for IPv6

```
--{ candidate shared default }--[  ]--
# info interface irb1 subinterface 1 ipv6 neighbor-discovery
    interface irb1 {
        subinterface 1 {
            ipv6 {
                admin-state enable
                neighbor-discovery {
                    virtual-ipv6-discovery {
                        address 2001::10:10:10:10 {
                            probe-interval 5
                            allowed-macs [
                                00:14:9C:78:E2:E1
                                00:14:9C:78:E2:E2
                                00:14:9C:78:E2:E3
                            ]
                            probe-bridged-subinterfaces [
                                ethernet-1/1.1
                            ]
                        }
                    }
                }
            }
        }
    }
```

In this configuration, SR Linux starts sending ARP requests for VIP 10.10.10.10 (**address** parameter) on the `ethernet 1/1.1` subinterface (**probe-bridged-subinterfaces** parameter) every 5 seconds (**probe-interval** parameter) so that the ARP entry for the VIP/VMAC is learned. The ARP entry is created only if the MAC address matches one of the MACs in the allowed list (**allowed-macs** parameter).

Up to 10 subinterfaces can be specified in the **probe-bridged-subinterfaces** list. The probe messages are sent only to the subinterfaces in this list. If no subinterfaces are specified in the list, no probe messages are sent.

The default **probe-interval** is 0, meaning that periodic probe messages are not sent by default. However regardless of the **probe-interval** setting, a probe message is sent to the subinterfaces in the **probe-bridged-subinterfaces** list whenever the following occur:

- the VIP is configured
- when the IRB interface becomes operationally up (when the prefix becomes preferred)
- when the ARP/ND entry corresponding to the VIP moves from type dynamic to type EVPN, or from type dynamic to type dynamic (changing to a different MAC)
- when subinterfaces in the **probe-bridged-subinterfaces** list become operationally up
- when the arpnd_mgr application restarts
- when the **allowed-macs** list, **probe-interval**, or **probe-bridged-subinterfaces** list changes

When the **probe-interval** is set to a non-zero value, SR Linux keeps probing for the VIP continuously, even after the ARP/ND entry for the VIP is created.

Up to 10 MAC addresses can be specified in the **allowed-macs** list. The ARP/ND entry for the VIP is created only if the resolving MAC address corresponds to one of the MAC addresses in the list. If no MAC addresses are specified in the list, any resolving MAC is valid for the creation of the ARP/ND entry. When a MAC address included in the **allowed-macs** list is used in an existing ARP/ND entry, and the MAC is removed from the list, the ARP/ND entry is deleted.

7.6.2 Displaying VIP discovery information

Procedure

You can display the number of probe packets sent by the SR Linux device using the **info from state** command. Statistics are displayed for probe packets on individual VIPs and the total number of probe packets for all VIPs configured for the subinterface.

Example

```
--{ candidate shared default }--[  ]--
# info from state interface irb1 subinterface 1 ipv4 arp
    interface irb1 {
        subinterface 1 {
            ipv4 {
                arp {
                    virtual-ipv4-discovery {
                        address 10.10.10.10 {
                            probe-bridged-subinterfaces [
                                ethernet-1/1.1
                            ]
                            probe-interval 5
                            allowed-macs [
                                00:14:9C:78:E2:E1
                                00:14:9C:78:E2:E2
                                00:14:9C:78:E2:E3
                            ]
                            statistics {
                                out-probe-packets 100
                            }
                        statistics {
                            out-total-probe-packets 100
                        }
                    }
                }
            }
        }
    }
```

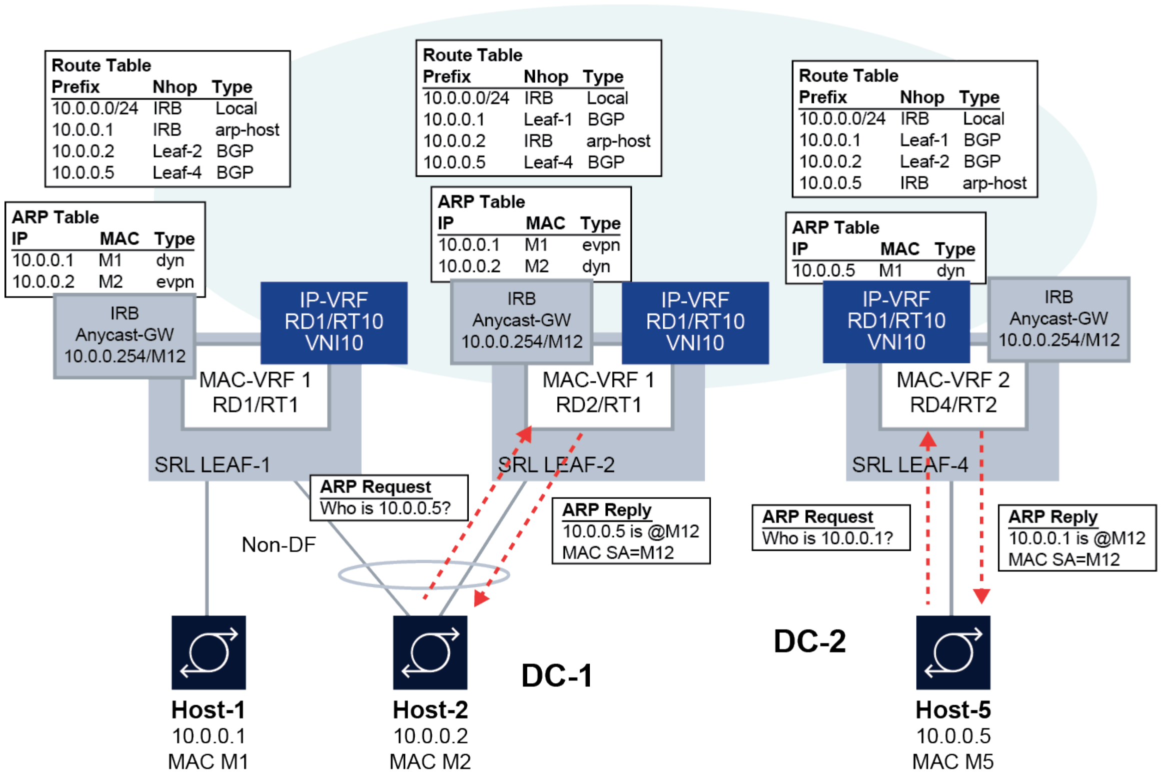
7.7 Layer 3 proxy-ARP/ND

Layer 3 proxy-ARP/ND configures the router to reply with its own MAC to ARP-requests and NS destined for any host. This feature is intended for the following use cases:

- Deployments that use a Layer 3 multihoming solution, but are unable to run EVPN, ESs or VXLAN.
- Virtual subnet scenarios, where leaf nodes are attached to different broadcast domains (BDs), but the hosts connected to them are part of the same subnet. RFC 7814 describes this type of solution.

The following figure shows an example virtual subnet configuration that uses this feature.

Figure 13: Layer 3 Proxy ARP/ND



In this example, SR Linux Leaf-1/Leaf-2 and SR Linux-4 are attached to different BDs; however, the hosts connected to them are part of the same subnet. This type of configuration is typically used in scenarios where the BDs are kept deliberately small, while hosts of the same subnet can be part of more than one of the BDs. This is the case described in RFC 7814.

Layer 3 proxy-ARP/ND triggers the router to reply to any ARP-request or NS from hosts, so that the traffic is attracted to the IRB subinterface and forwarded by the router to a destination on the same subinterface from which the request came or a different subinterface or route.

Layer 3 proxy-ARP/ND is intended to be used along with the `learn-unsolicited` and `host-route populate` features, so that ARP/ND entries are generated and host routes created out of them and advertised by the routing protocol enabled in the routing network instance (default or IP-VRF).

This feature is supported on IRB and other Layer 3 subinterfaces. EVPN may or may not be enabled on the attached MAC-VRF. The subinterfaces can be attached to an IP-VRF or the default network instance.

Note: The virtual subnet expansion enabled by this feature only works for unicast IP traffic with TTL > 1 between hosts in different BDs.

7.7.1 Layer 3 proxy-ARP

When Layer 3 proxy-ARP is enabled, all ARP-requests are copied to the CPM and replied using the `anycast-gw` MAC, if configured; otherwise, the interface `hw-mac-address` MAC is used. This process is irrespective of the target IP address of the ARP-request being in the ARP table or in the route-table.

The system does not reply to gratuitous ARP (GARP) requests, where the ARP Sender Protocol Address and ARP Target Protocol Address are both set to the IP address of the cache entry to be updated.

When Layer 3 proxy-ARP is enabled, the router replies to any ARP-request unconditionally if received over a bridged subinterface. If the ARP-request is received over VXLAN, the router replies only if the ARP-request is targeted to its own IP; if it is targeted to a non-local IP on the IRB, the ARP-request is flooded to local bridged subinterfaces, changing the source MAC and IP with the local ones.

If the original ARP-request came on a bridged subinterface, and the target IP has not been learned on the subinterface yet, the router sends an ARP-request with its own source information to get the target IP learned as soon as possible.

The enabled proxy-ARP behavior attracts all traffic from all the hosts attached to the IRB subinterface that sent an ARP-request for the destination IP address. As a result of this:

- If the received packets have an IP DA with an ARP entry on the same subinterface over which the ARP-request was received, the router forwards the packets to the next-hop indicated by the ARP entry.
- If the received packets have an IP DA with an ARP entry on a different subinterface, the router forwards the packets to the next-hop indicated by the ARP entry.
- If the received packets have an IP DA with a matching route in the route-table, the router forwards the packets based on the route-table entry.
- Otherwise, the attracted traffic is dropped.

The Layer 3 proxy-ARP feature does not require a route lookup before replying to the received ARP-requests. The received ARP-requests are only copied to CPM; they are not flooded to other objects in the MAC-VRF.

The ICMP redirects generated by XDP CPM are suppressed for IP packets targeting an IP address within the same subnet.

Received ARP-replies with a DA not equal to the local MAC, if any are received, are forwarded based on a mac-table lookup.

Note: The system verifies the source IP and replies regardless of the target IP (it floods to bridged subinterfaces if received on VXLAN). The command `learn-unsolicited` is required to create an ARP entry.

7.7.1.1 Configuring Layer 3 proxy-ARP

Procedure

To configure Layer 3 proxy-ARP, enable it on a subinterface.

Example

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv4 arp proxy-arp
    interface ethernet-1/1 {
        subinterface 1 {
            ipv4 {
                admin-state enable
                arp {
                    proxy-arp true
                }
            }
        }
    }
```

7.7.2 Layer 3 proxy-ND

When Layer 3 proxy-ND is enabled, it functions similarly to Layer 3 proxy-ARP. All NS messages sent to the Solicited-Node multicast address `[ff02::1:ff][low-3-bytes-unicast-ip]` are copied to CPM, and all of the NS messages are replied using the `anycast-gw` MAC, if configured (or interface `hw-mac-address` otherwise), without any route lookup.

Received NS messages are only copied to CPM and not flooded in the BD. In this way, the feature provides an implicit ND flood suppression.

The **learn-unsolicited** command is not needed to learn a neighbor from an NS, because it is basic ND behavior to create a neighbor for the SA of an NS if there is a reply and it is a valid neighbor.

7.7.2.1 Configuring Layer 3 proxy-ND

Procedure

To configure Layer 3 proxy-ND, enable it on a subinterface.

Example

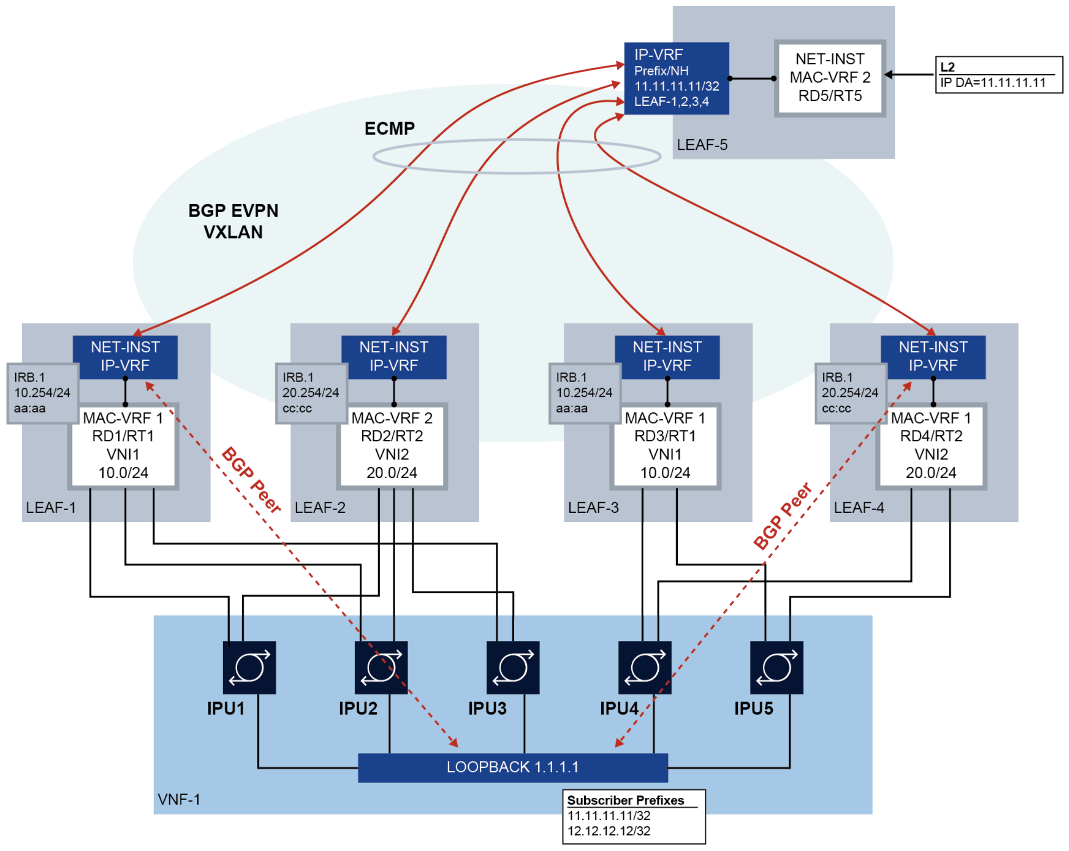
```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv6 neighbor-discovery
    interface ethernet-1/1 {
        subinterface 1 {
            ipv6 {
                neighbor-discovery {
                    proxy-nd true
                }
            }
        }
    }
```

7.8 EVPN IP aliasing for BGP peering to a VNF loopback address

As defined in *draft-sajassi-bess-evpn-ip-aliasing*, IP aliasing can be used with EVPN ESs to load-balance traffic to a host connected to multiple leaf nodes attached to the same IP-VRF, even if some of the leaves do not advertise reachability to that host.

SR Linux supports using IP aliasing for BGP peering where leaf switches are multihomed to a single virtual network function (VNF) loopback address. The following figure shows this type of configuration.

Figure 14: BGP peering to VNF loopback



In this configuration, leaf nodes LEAF-1 through LEAF-5 are connected to an EVPN-VXLAN Layer 3 tenant domain, and four of these leaf nodes are multihomed to the same VNF. This VNF has five IPUs, similar to line cards, that are each dual-homed to two leaf nodes. Each IPU has two Layer 3 links that are connected to the leaf switches.

MAC-VRF 1 and MAC-VRF 2 are different Broadcast Domains (BDs) and have no Layer 2 EVPN-VXLAN connectivity between them.

The VNF is attached to a pair of subscriber prefixes, 11.11.11.11/32 and 12.12.12.12/32, that must be advertised to the leaf nodes, peering from the IP-VRF IRB interfaces, so the leaf nodes can distribute the reachability to the rest of the network.

In this example, the VNF can run PE-CE BGP from loopback interface 1.1.1.1/32, but supports a maximum of two BGP sessions.

In this configuration, IP aliasing on SR Linux allows the following:

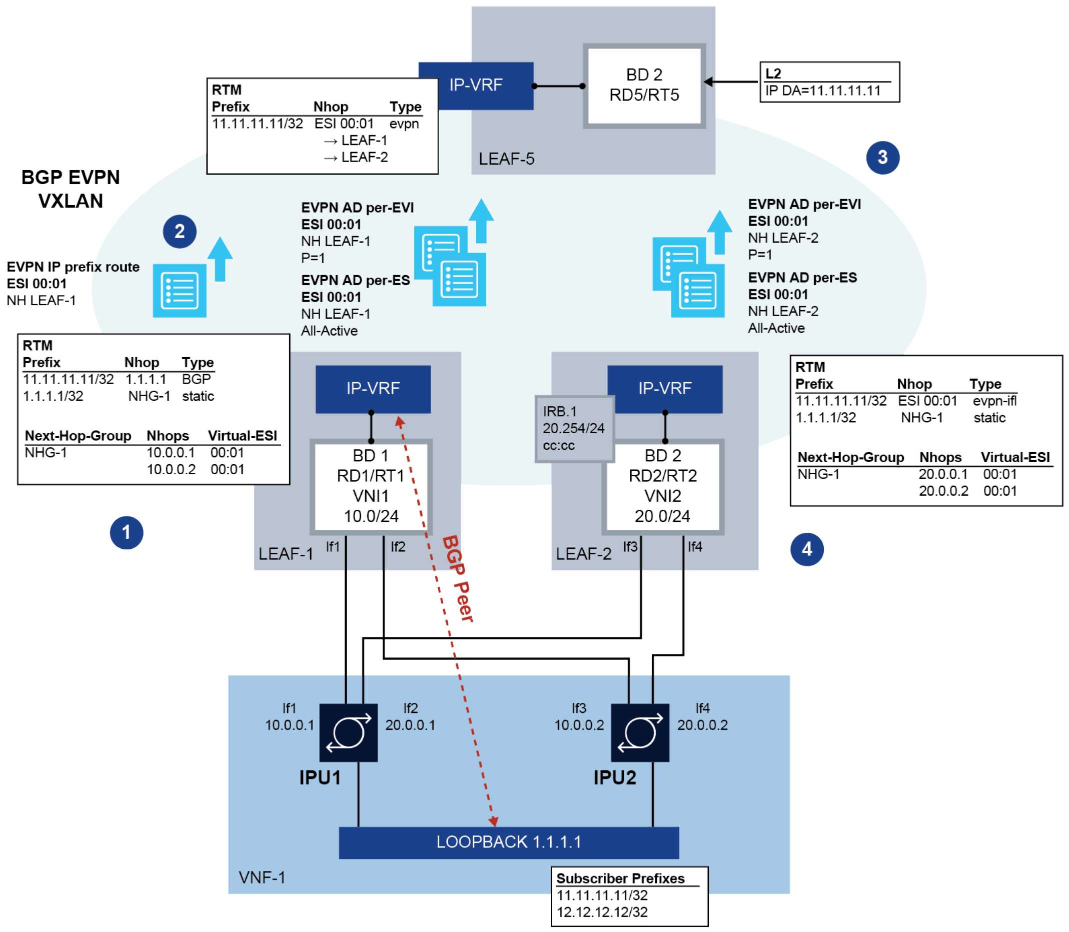
- Full traffic spraying in both upstream and downstream directions, even though the number of BGP sessions is limited to two, and the VNF connects to four leaf nodes.
- No tromboning for downstream traffic routed to a leaf node that does not have a BGP session with the VNF. The local leaf must always route locally to the VNF, unless the VNF loopback becomes unreachable.

  For example, when LEAF-5 sends traffic with destination 11.11.11.11 to LEAF-3, LEAF-3 needs to route locally to its local MAC-VRF 1, even if it has an RT5 in its route table pointing at LEAF-1 or LEAF-4.

EVPN IP aliasing on SR Linux

The following figure shows how IP aliasing on SR Linux works for BGP peering to a VNF loopback. To simplify the explanation, only two leaf nodes are shown; however, multiple leaf nodes can be multihomed to the same VNF loopback.

Figure 15: EVPN IP aliasing on SR Linux



RTM

Prefix	Nhop	Type
11.11.11.11/32	ESI 00:01 → LEAF-1 → LEAF-2	evpn

RTM

Prefix	Nhop	Type
11.11.11.11/32	1.1.1.1	BGP
1.1.1.1/32	NHG-1	static

Next-Hop-Group	Nhops	Virtual-ESI
NHG-1	10.0.0.1	00:01
	10.0.0.2	00:01

RTM

Prefix	Nhop	Type
11.11.11.11/32	ESI 00:01	evpn-ifl
1.1.1.1/32	NHG-1	static

Next-Hop-Group	Nhops	Virtual-ESI
NHG-1	20.0.0.1	00:01
	20.0.0.2	00:01

Subscriber Prefixes
11.11.11.11/32
12.12.12.12/32

In this configuration, BGP peering to the loopback on the VNF works as follows. The numbers represent the steps depicted in the figure.

1. The VNF's loopback address (1.1.1.1) is associated with virtual Layer 3 ES ES-1 on both LEAF-1 and LEAF-2, working in all-active mode. Static routes to 1.1.1.1/32 are configured on both leaf switches. These static routes can run BFD to the VNF, if required, to speed up fault detection.
   After ES-1 is enabled, and as soon as 1.1.1.1 has a route (of any mask length) in the IP-VRF's route table, LEAF-1 and LEAF-2 advertise an EVPN ES route, as well as EVPN Auto Discovery (AD) per ES and per EVI routes for ES-1. The AD routes are advertised based on the presence of a route for 1.1.1.1 in the route table. In this example, the route for 1.1.1.1/32 is statically configured in the route table, but any non-EVPN IP prefix route that provides reachability to 1.1.1.1 would trigger the advertisement of the AD routes.

   If the ES ES-1 oper-state is up and a route for 1.1.1.1 is active in the route table, each leaf advertises:

   - AD per-ES route for ES-1, indicating all-active mode

- AD per-EVI route for ES-1, indicating P = 1 (primary state is set), with route distinguisher and route target of the IP-VRF

  The AD per-EVI routes contain two flags, P (primary) and B (backup), which provide an indication for the remote nodes to know whether they can send traffic to the owners of the routes. The remote nodes send traffic for the ES to all the nodes advertising themselves as primary (P = 1), and not to nodes signaling P = 0, B = 1, unless there are no primary nodes left in the ES.

  For all-active multihoming mode, there is no DF election in the context of the ES.

2. Any prefixes that are resolved to the next-hop associated with ESI-1 are advertised as EVPN IP prefix routes with ESI 00:01 (representation of the ESI for ES-1). In the example in Figure 15: EVPN IP aliasing on SR Linux, prefix 11.11.11.11/32 is advertised as an EVPN IP prefix route with ESI 00:01.
3. LEAF-5 identifies the EVPN IP prefix route with ESI 00:01 and installs the route in the IP-VRF route table with as many next-hops as received AD routes with ESI 00:01 and the primary flag set (P = 1). That is, the IP prefix route with ESI 00:01 is recursively resolved to the next hops that advertise the AD per-EVI (with P = 1) and per-ES routes for 00:01.
   If the virtual Layer 3 ES is configured as single-active, a DF Election among all the leaf nodes attached to the same ES determines which leaf nodes advertise themselves as primary for the ES.
4. LEAF-2 identifies the EVPN IP prefix route with ESI 00:01 and next-hop LEAF-1 and resolves the route recursively to the next-hop associated with ES-1.
   If the static route associated with ES-1 goes down, the leaf withdraws the associated Layer 3 AD per-EVI/ES routes to avoid attracting traffic from LEAF-5.

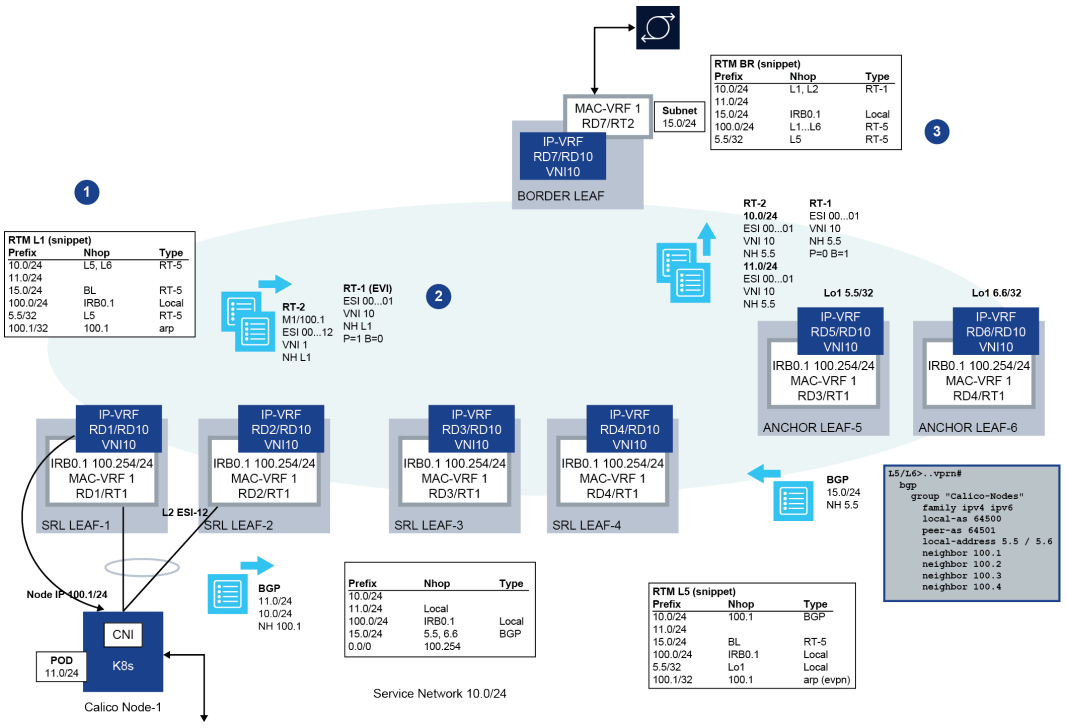
Centralized routing model and PE-CE routes resolved over EVPN-IFL

SR Linux supports PE-CE sessions over VXLAN and PE-CE next-hop resolution via EVPN-IFL, which, along with Layer 3 ES, enables support for the centralized routing model, shown in Figure 16: Centralized routing model.

This model uses BGP PE-CE peering on CEs to a pair of centralized anchor leaf nodes, as opposed to BGP PE-CE sessions to the directly connected leaf nodes. This is possible because BGP PE-CE route resolution via EVPN IFL routes is unblocked, which is necessary when the BGP session is not established via a local subinterface.

Having all the BGP PE-CE sessions centralized in a pair of anchor leaf nodes simplifies the provisioning of BGP not only on the rest of the leaf nodes, but also on the CEs connected to the leaf nodes.

Figure 16: Centralized routing model



RTM L1 (snippet)

Prefix	Nhop	Type
10.0/24	L5, L6	RT-5
11.0/24		
15.0/24	BL	RT-5
100.0/24	IRB0.1	Local
5.5/32	L5	RT-5
100.1/32	100.1	arp

RTM BR (snippet)

Prefix	Nhop	Type
10.0/24	L1, L2	RT-1
11.0/24		
15.0/24	IRB0.1	Local
100.0/24	L1...L6	RT-5
5.5/32	L5	RT-5

Prefix	Nhop	Type
10.0/24		
11.0/24	Local	
100.0/24	IRB0.1	Local
15.0/24	5.5, 6.6	BGP
0.0/0	100.254	

RTM L5 (snippet)

Prefix	Nhop	Type
10.0/24	100.1	BGP
11.0/24		
15.0/24	BL	RT-5
100.0/24	IRB0.1	Local
5.5/32	Lo1	Local
100.1/32	100.1	arp (evpn)

```
L5/L6>..vprn#
  bgp
    group "Calico-Nodes"
      family ipv4 ipv6
      local-as 64500
      peer-as 64501
      local-address 5.5 / 5.6
      neighbor 100.1
      neighbor 100.2
      neighbor 100.3
      neighbor 100.4
```

In this example, the centralized routing model is used between all the Kubernetes (K8s) node IPs and the IP-VRF loopbacks on anchor leaf nodes; for example, between Node IP 100.1/24 (Node-1) and 5.5/32 in ANCHOR LEAF-5. Although only one Kubernetes node is depicted in the figure, multiple nodes may be attached to the leaf nodes throughout the data center. By using this model, the BGP configuration is simplified, especially on the Kubernetes nodes, where they can all use the same configuration with a peer to the anchor leaf nodes.

All the BGP PE-CE routes advertised by the CEs are re-advertised by the anchor leaf nodes using EVPN IP prefix routes. For example, prefixes 10.0/24 and 11.0/24 are advertised by Node-1 and re-advertised by LEAF-5 in IP prefix routes. IP Aliasing solves the issue of tromboning the traffic with destination 10.0 or 11.0 from the BORDER LEAF to LEAF-1 or LEAF-2 via anchor LEAF-5. IP Aliasing works as follows (where the numbers represent the steps depicted in Figure 16: Centralized routing model):

1. LEAF-1, LEAF-2, LEAF-5 and LEAF-6 are all configured with the virtual Layer 3 ES ES-1, which uses ESI 00..01, is associated with Node-1 IP 100.1, and is configured as single-active. Node-1 advertises BGP PE-CE routes for 10.0/24 and 11.0/24 to LEAF-5.
2. Because LEAF-1 and LEAF-2 have 100.1 in their route table as a local route and they are configured as primary nodes for the ES, they advertise AD per-EVI routes for ES-1 with P = 1. Anchor LEAF-5 and LEAF-6 are configured as non-primary, so even if they have 100.1 in their route table, they advertise their AD per-EVI routes with P = 0 and B = 1 as backup nodes for ES-1. When for example LEAF-5 receives the BGP PE-CE routes from Node-1, it re-advertises them in EVPN IP prefix routes with ESI 00..01.This is because the BGP PE-CE routes' next-hop matches the IP address in ES-1.
3. BORDER LEAF imports the EVPN IP prefix routes with ESI 00..01 and recursively resolves them to the next-hops of all the AD per-EVI routes advertised as primary (P = 1); that is, LEAF-1 and LEAF-2. When receiving traffic to 10.0/24 or 11.0/24, BORDER LEAF load balances the traffic to only LEAF-1 and LEAF-2, as opposed to the anchor leaf nodes.

7.8.1 Configuring Layer 3 Ethernet Segments for IP aliasing

To configure a Layer 3 Ethernet Segment for IP aliasing, you specify the Layer 3 next hop to be associated with an Ethernet Segment (ES). The AD per-ES/EVI routes for the ES are advertised when the specified next-hop address is active in the route table of the IP-VRF with the configured EVI.

The following configures two examples of Layer 3 Ethernet Segments, which are associated with an IPv4 and an IPv6 next-hop, respectively.

```
A:Leaf-1# info from running system network-instance protocols evpn ethernet-segments
  system {
    network-instance {
      protocols {
        evpn {
          ethernet-segments {
            timers {
              boot-timer 180
              activation-timer 3
            }
            bgp-instance 1 {
              ethernet-segment es_v4_1 {
                type virtual
                admin-state enable
                esi 00:bc:de:00:00:00:00:01:00:04
                multi-homing-mode single-active
                next-hop 6.0.1.254 {
                  evi 1 {
                  }
                }
                df-election {
                  algorithm {
                    type preference
                    preference-alg {
                      preference-value 2
                      capabilities {
                        ac-df exclude
                        non-revertive false
                      }
                    }
                  }
                }
                routes {
                  next-hop use-system-ipv4-address
                  ethernet-segment {
                    originating-ip use-system-ipv4-address
                  }
                }
              }
              ethernet-segment es_v6_1 {
                type virtual
                admin-state enable
                esi 00:bc:de:00:00:00:00:01:00:06
                multi-homing-mode single-active
                next-hop 6:0:1::254 {
                  evi 1 {
                  }
                }
                df-election {
                  algorithm {
                    type preference
                    preference-alg {
                      preference-value 2
                      capabilities {
                        ac-df exclude
                        non-revertive false
                      }
                    }
                  }
                }
                routes {
                  next-hop use-system-ipv4-address
                  ethernet-segment {
                    originating-ip use-system-ipv4-address
                  }
                }
              }
            }
          }
        }
      }
    }
  }
```

Layer 3 ESs must be configured as type `virtual` and must be associated with a next-hop and EVI. When the next-hop IP address is active in an IP-VRF identified by the configured EVI, the corresponding AD-per-ES and per-EVI routes for the ES are advertised.

If the Layer 3 ES is configured as `multi-homing-mode all-active`, the DF election algorithm configuration is irrelevant, because there is no DF and all the nodes are primary (they all advertise P = 1 in their AD per-EVI routes). However, if configured as `multi-homing-mode single-active`, a DF election is computed among all the nodes advertising the same ESI in the ES route. In this case, the `default`, `preference`, or `manual` algorithms can be used. The manual DF election simply controls whether the node is primary or not by configuration.

For example, Leaf-1 is configured as primary, so it advertises an AD-per-EVI route with P = 1 irrespective of the other leaf nodes in the ES:

```
--{ +* candidate shared default }--[ system network-instance protocols evpn ethernet-segments
 bgp-instance 1 ethernet-segment L3-ES-1 ]--
A:Leaf-1# info
  type virtual
  admin-state enable
  esi 01:20:20:20:01:00:00:00:00:00
  multi-homing-mode single-active
  next-hop 20.20.20.1 {
    evi 2 {
    }
  }
  df-election {
    algorithm {
      type manual
      manual-alg {
        primary-evi-range 2 {
          end-evi 2
        }
      }
    }
  }
```

However, Leaf-2 in the following example is not configured as primary, so it advertises an AD-per-EVI route with P = 0 and B = 1. Therefore, it does not attract traffic to the ES as long as there are primary nodes in the ES.

```
--{ +* candidate shared default }--[ system network-instance protocols evpn ethernet-segments
 bgp-instance 1 ethernet-segment L3-ES-1 ]--
A:Leaf-2# info
  type virtual
  admin-state enable
  esi 01:20:20:20:01:00:00:00:00:00
  multi-homing-mode single-active
  next-hop 20.20.20.1 {
    evi 2 {
    }
  }
  df-election {
    algorithm {
      type manual
      manual-alg {
      }
    }
  }
```

The following example shows the configuration of LEAF-1 and LEAF-2 in the Figure 14: BGP peering to VNF loopback figure. LEAF-5 is configured without any ES and with an IP-VRF that has BGP-EVPN enabled and **ecmp** greater than or equal to 2, so that it can load-balance traffic to the two leaf nodes attached to the ES.

```
# LEAF-1 and LEAF-2 configuration of the Ethernet Segment
--{ + candidate shared default }--[ system network-instance protocols evpn ]--
A:leaf1/leaf2# info
  ethernet-segments {
    bgp-instance 1 {
      ethernet-segment L3-ES-1 {
        type virtual
        admin-state enable
        esi 01:01:00:00:00:00:00:00:00:00
        multi-homing-mode all-active
        next-hop 1.1.1.1 {
          evi 2 {
          }
        }
      }
    }
  }

# LEAF-1 IP-VRF configuration, including BGP session to the VNF
--{ + candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf1# info
  type ip-vrf
  admin-state enable
  interface irb0.1 {
  }
  interface lo1.1 {
  }
  vxlan-interface vxlan1.2 {
  }
  protocols {
    bgp-evpn {
      bgp-instance 1 {
        admin-state enable
        vxlan-interface vxlan1.2
        evi 2
        ecmp 4
      }
    }
    bgp {
        autonomous-system 64501
        router-id 10.0.0.1
        group PE-CE {
        export-policy all
        import-policy all
        peer-as 64500
      }
        afi-safi ipv4-unicast {
            admin-state enable
        }
      neighbor 1.1.1.1 {
        peer-group PE-CE
        multihop {
          admin-state enable
          maximum-hops 10
        }
        transport {
          local-address 10.0.0.1
        }
      }
    }
    bgp-vpn {
      bgp-instance 1 {
        route-distinguisher {
          rd 1.1.1.1:2
        }
        route-target {
          export-rt target:64500:2
          import-rt target:64500:2
        }
      }
    }
  static-routes {
    route 1.1.1.1/32 {
      next-hop-group NHG-1
    }
  }
  next-hop-groups {
    group NHG-1 {
      nexthop 1 {
        ip-address 10.0.0.1
      }
      nexthop 2 {
        ip-address 10.0.0.2
      }
    }
  }

# LEAF-2 IP-VRF configuration
--{ + candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf2# info
  type ip-vrf
  interface irb0.1 {
  }
  vxlan-interface vxlan1.2 {
  }
  protocols {
    bgp-evpn {
      bgp-instance 1 {
        vxlan-interface vxlan1.2
        evi 2
        ecmp 4
      }
    }
    bgp-vpn {
      bgp-instance 1 {
        route-distinguisher {
          rd 2.2.2.2:2
        }
        route-target {
          export-rt target:64500:2
          import-rt target:64500:2
        }
      }
    }
  static-routes {
    route 1.1.1.1/32 {
      next-hop-group NHG-1
    }
  }
  next-hop-groups {
    group NHG-1 {
      nexthop 1 {
        ip-address 20.0.0.1
      }
      nexthop 2 {
        ip-address 20.0.0.1
      }
    }
  }
}
```

The following shows the route table for one of the subscriber prefixes (11.11.11.11/32) that are advertised from the VNF with a next-hop matching the Layer 3 ES next-hop. For LEAF-1, the route is learned via BGP PE-CE.

```
# LEAF-1 route table entry
--{ + candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf1# info from state route-table ipv4-unicast route 11.11.11.11/32 id 0 route-type bgp
 route-owner bgp_mgr
  route-table {
    ipv4-unicast {
      route 11.11.11.11/32 id 0 route-type bgp route-owner bgp_mgr {
        metric 0
        preference 170
        active true
        last-app-update 2022-10-19T10:26:51.020Z
        next-hop-group 423577430891
        next-hop-group-network-instance ip-vrf-2
        resilient-hash false
        fib-programming {
          last-successful-operation-type add
          last-successful-operation-timestamp 2022-10-19T10:26:51.021Z
          pending-operation-type none
          last-failed-operation-type none
        }
      }
    }
  }
```

For LEAF-2, the route is learned via EVPN; however, the next-hop is resolved locally to the next-hop-group associated with the route for 1.1.1.1 (the ES next-hop IP address).

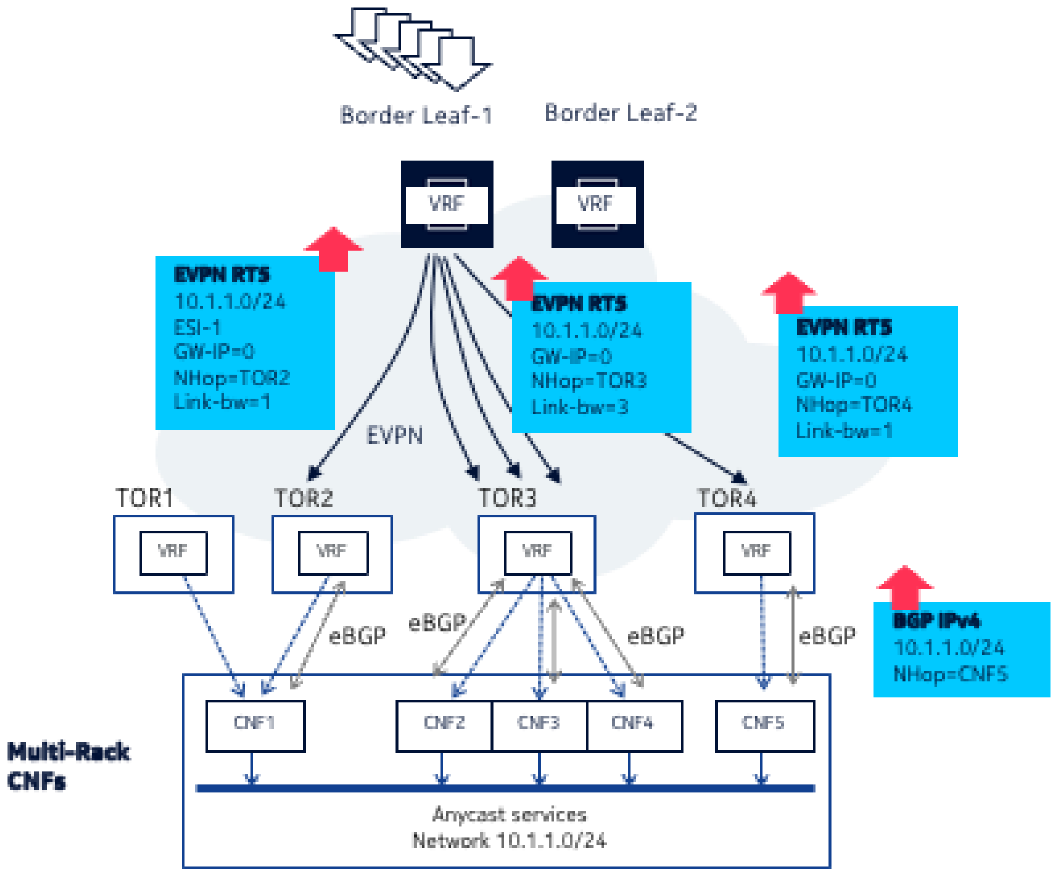
```
# LEAF-2 route table entry
--{ +* candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf2# info from state route-table ipv4-unicast route 11.11.11.11/32 id 0 route-type bgp-evpn
 route-owner bgp_evpn_mgr
  route-table {
    ipv4-unicast {
      route 11.11.11.11/32 id 0 route-type bgp-evpn route-owner bgp_evpn_mgr {
        metric 0
        preference 170
        active true
        last-app-update 2022-10-19T10:25:06.199Z
        next-hop-group 423577389148
        next-hop-group-network-instance ip-vrf-2
        resilient-hash false
        fib-programming {
          last-successful-operation-type add
          last-successful-operation-timestamp 2022-10-19T10:25:06.199Z
          pending-operation-type none
          last-failed-operation-type none
        }
      }
    }
  }

--{ +* candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf2# info from state route-table next-hop-group 423577389148
  route-table {
    next-hop-group 423577389148 {
      backup-next-hop-group 0
      next-hop 0 {
        next-hop 423577389136
        resolved true
      }
      next-hop 1 {
        next-hop 423577389137
        resolved true
      }
    }
  }

--{ +* candidate shared default }--[ network-instance ip-vrf-2 ]--
A:leaf2# show route-table next-hop 423577389136
--------------------------------------------------------------------------------
Next-hop route table of network instance ip-vrf-2
--------------------------------------------------------------------------------
Index : 423577389136
Next-hop : 20.20.20.1
Type : indirect
Subinterface : N/A
Resolving Route : 20.20.20.0/24 (local)
```

8 Unequal ECMP for EVPN IP prefix routes

SR Linux supports unequal Equal Cost Multi Path (ECMP) for EVPN IP prefix IFL (interface-less) routes. To do this, SR Linux makes use of the EVPN link-bandwidth extended community (EC) defined in draft-ietf-bess-evpn-unequal-lb. This extended community indicates the weight for a specific IP prefix; that is, the number of PE-CE multi-paths for an IP prefix that is re-advertised into an EVPN IP prefix route.

The following figure shows an example of weighted ECMP. Assuming each Container Network Function (CNF) advertises the anycast subnet 10.1.1.0/24 from a different next-hop, each TOR ends up with a different number of multi-paths in its PE-CE session. In the example below, TOR3 has three multi-paths for the anycast subnet, and the advertised EVPN IP prefix route includes an EVPN link-bandwidth extended community with a weight of 3. The rest of the TORs send a weight of 1. Note that TOR1 and TOR2 are multi-homed to the same CNF1, yet they send a weight of 1.

Figure 17: EVPN using weighted ECMP



On the border leafs / data center gateways, when this feature is enabled, if the EVPN IP prefix route has an Ethernet Segment Identifier (ESI) of 0, the PE sprays the flows to the EVPN IP prefix route based on the received weight; in this example, one-fifth of the flows are sent to TOR4, and three-fifths are sent to TOR3.

When the EVPN IP prefix route has a non-zero ESI, and there is a weight in the route:

- The EVPN link-bandwidth extended community received in the EVPN IP prefix route indicates the weight for the EVPN IP prefix route. The EVPN link-bandwidth extended community may also be received in the IP A-D per ES routes for each PE attached to the Ethernet segment (ES), but the system ignores it in this case.
- The PE sprays the flows to the EVPN IP prefix route based on the received weight, dividing the flows to an ES among the number of PEs attached to the ES.

  In the example above, one-fifth of the flows are sent to the aliased pair TOR1/TOR2 (either one is selected because TOR1 and TOR2 are assumed to send equal weights in the AD per ES routes).
- The system rounds up when the advertised weight for the ESI, divided by the number of PEs in the ES, is not an integer.

  For example, if ES1 (TOR1/TOR2) advertises BW=3, and TOR4 advertises BW=1, then 3 (BW) / 2 (PE in ES1) = 1.5. The system rounds up, and the remote nodes install weight=2 for TOR1, weight=2 for TOR2, and weight=1 for TOR4.
- If the weight received in a non-zero ESI IP prefix route exceeds 128, the system caps it at 128, then divides the weight into the number of PEs in the ES.
- If two EVPN IP prefix routes are received for the same prefix, same ESI, different route distinguishers (RDs), they should have the same weight. However, if they have different weights, the system selects the weight from the first EVPN IP prefix route.
- If the EVPN link-bandwidth extended community is missing from any of the PEs in an ECMP set, or the Value Units field of the extended community is inconsistent, the weight is ignored by the receiving PE, and regular ECMP forwarding is performed. The Value Units field can indicate "bandwidth" or a "generalized weight", with the latter being supported by SR Linux.

8.1 Advertising the EVPN link-bandwidth extended community

Procedure

To configure weighted ECMP, you enable advertisement of the EVPN link-bandwidth extended community and specify the weight to be advertised in the extended community.

You can configure the following parameters for the advertised weight in the extended community:

- `weight` specifies the weight to be advertised in the EVPN link-bandwidth extended community for the advertised EVPN IP prefix routes for the service. If set to `dynamic` (the default value), the weight is dynamically set based on the number of BGP PE-CE paths for the IP prefix that is advertised in an EVPN IP prefix route. The dynamic weight only considers BGP PE-CE paths.

  Alternatively, the weight can be set to a fixed integer value in the range 1 to 128.
- `maximum-dynamic-weight` specifies the maximum weight to be advertised in the EVPN link-bandwidth extended community for the advertised EVPN IP-Prefix routes for the service. If weight `dynamic` is configured, the actual advertised weight is the minimum of the number of BGP PE-CE paths for the prefix and the configured `maximum-dynamic-weight`.

Example

The following example enables advertisement of the EVPN link bandwidth extended community and specifies the weight to be included in the extended community for the advertised EVPN IP prefix routes.

```
--{ * candidate shared default }--[  ]--
# info network-instance VRF1 protocols bgp-evpn bgp-instance 1
    network-instance VRF1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        route-table {
                            ip-prefix {
                                evpn-link-bandwidth {
                                    advertise {
                                        weight 60
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

8.2 Enabling weighted ECMP

Procedure

When weighted ECMP is enabled, the system takes into account the EVPN link-bandwidth extended community when installing an ECMP set for an EVPN IP prefix route in the IP-VRF route table.

Flows to an IP prefix received with a weight and a zero ESI are sprayed according to the weight. If the EVPN IP prefix route received with a weight has a non-zero ESI, the weight is divided into the number of PEs attached to the ES, and rounded up if the result is not an integer.

The command also enables weighted ECMP for BGP CEs that are configured with a weight specified with the `link-bandwidth.add-next-hop-count-to-received-bgp-routes` setting.

Configuring `max-ecmp-hash-buckets-per-next-hop-group` preserves the datapath resources used for the weighted next-hops. The normalization algorithm also refers to this number of hash buckets. See Displaying normalized ECMP weights for an example of how the weights are normalized the using the `max-ecmp-hash-buckets-per-next-hop-group` setting.

Example

The following example enables weighted ECMP and specifies the maximum number of ECMP hash buckets per next-hop-group. The weights for weighted ECMP are normalized based on this number of hash buckets.

```
--{ * candidate shared default }--[  ]--
# info network-instance VRF1 protocols bgp-evpn bgp-instance 1
    network-instance VRF1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        route-table {
                            ip-prefix {
                                evpn-link-bandwidth {
                                    weighted-ecmp {
                                        admin-state enable
                                        max-ecmp-hash-buckets-per-next-hop-group 4
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

8.3 Configuring weighted ECMP for received PE-CE BGP routes

Procedure

When the system is enabled to process the link-bandwidth extended community, you can configure a weight to be internally added to the received PE-CE BGP routes for the purpose of EVPN unequal ECMP.

Example

```
--{ * candidate shared default }--[  ]--
# info network-instance VRF1 protocols bgp group g1 link-bandwidth
    network-instance VRF1 {
        protocols {
            bgp {
                group g1 {
                    link-bandwidth {
                        add-next-hop-count-to-received-bgp-routes 100
                    }
                }
            }
        }
    }
```

8.4 Displaying normalized ECMP weights

Procedure

The system normalizes the weights used in weighted ECMP using the `max-ecmp-hash-buckets-per-next-hop-group` setting and the advertised weights, according to the algorithm described in "Normalizing datapath weights" in the *SR Linux Routing Protocols Guide*.

You can display the normalized weights for a next-hop-group.

Example

In the following example, the same prefix 19.1.1.1/32 is received from two PEs with weights 40 and 60, respectively. The maximum supported ECMP paths is 8. The `max-ecmp-hash-buckets-per-next-hop-group` setting is 4. The system programs the next-hops with normalized weights.

To display the number of the next-hop-group for prefix 19.1.1.1/32:

```
--{ running }--[  ]--
# info from state network-instance VRF1 route-table ipv4-unicast route 19.1.1.1/32 id 0
 route-type bgp-evpn route-owner bgp_evpn_mgr origin-network-instance VRF1
    network-instance VRF1 {
        route-table {
            ipv4-unicast {
                route 19.1.1.1/32 id 0 route-type bgp-evpn route-owner bgp_evpn_mgr
 origin-network-instance VRF1 {
                    leakable false
                    metric 0
                    preference 170
                    active true
                    last-app-update 2023-05-31T18:29:25.207Z
                    next-hop-group 94413408183
                    next-hop-group-network-instance VRF1
                    resilient-hash false
                    fib-programming {
                        suppressed false
                        last-successful-operation-type modify
                        last-successful-operation-timestamp 2023-05-31T18:29:25.207Z
                        pending-operation-type none
                        last-failed-operation-type none
                    }
                }
            }
        }
    }
```

To display the received weights for the individual next hops in the next-hop group:

```
--{ running }--[  ]--
# info from state network-instance VRF1 route-table next-hop-group 94413408183
    network-instance VRF1 {
        route-table {
            next-hop-group 94413408183 {
                backup-next-hop-group 0
                fib-programming {
                    last-successful-operation-type add
                    last-successful-operation-timestamp 2023-05-31T18:29:25.207Z
                    pending-operation-type none
                    last-failed-operation-type none
                }
                next-hop 0 {
                    next-hop 94413408157
                    weight 40
                    resolved true
                }
                next-hop 1 {
                    next-hop 94413408152
                    weight 60
                    resolved true
                }
            }
        }
    }
```

The system normalizes the weights using the algorithm described in "Normalizing datapath weights" in the *SR Linux Routing Protocols Guide*, taking into account the received weights, the maximum supported ECMP paths, and the `max-ecmp-hash-buckets-per-next-hop-group` setting.

You can display the normalized weights with the following command:

```
--{ running }--[  ]--
# info from state platform linecard 1 forwarding-complex 0 fib-table next-hop-group
 94413408183
    platform {
        linecard 1 {
            forwarding-complex 0 {
                fib-table {
                    next-hop-group 94413408183 {
                        oper-state down
                        backup-next-hop-group 0
                        backup-active false
                        next-hop 0 {
                            next-hop 94413408157
                            oper-state up
                            normalized-weight 1
                        }
                        next-hop 1 {
                            next-hop 94413408152
                            oper-state up
                            normalized-weight 2
                        }
                    }
                }
            }
        }
    }
```

9 EVPN VLAN-aware bundle services

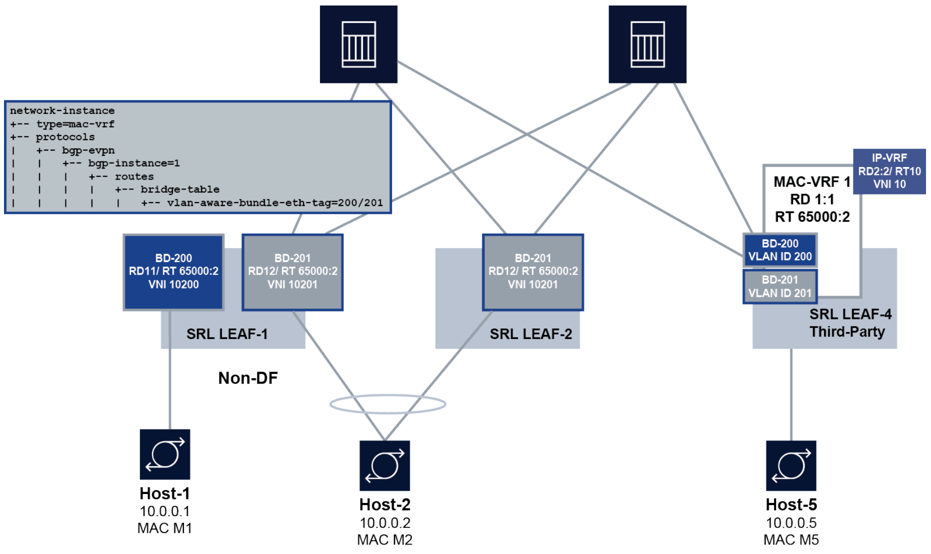
This chapter describes the components of EVPN VLAN-aware bundle services.

- Configuring the Ethernet Tag ID for VLAN-aware bundle services
- Displaying the Ethernet Tag ID for VLAN-aware bundle services

RFC 7432 defines VLAN-aware bundle services as those EVPN instances that consist of multiple broadcast domains. On SR Linux, a broadcast domain instance on a leaf node is identified by a MAC-VRF, and a MAC-VRF can contain only one broadcast domain. However, SR Linux can be attached to VLAN-aware bundle broadcast domains along with other third-party routers.

The following figure shows a configuration that features interoperability between SR Linux systems and a third-party device configured with VLAN-aware bundle services.

Figure 18: EVPN interoperability with VLAN-aware bundle services



```
network-instance
+-- type=mac-vrf
+-- protocols
|    +-- bgp-evpn
|    |    +-- bgp-instance=1
|    |    |    +-- routes
|    |    |    |    +-- bridge-table
|    |    |    |    |    +-- vlan-aware-bundle-eth-tag=200/201
```

In this configuration, Leaf-1 and Leaf-2 are SR Linux systems, and Leaf-4 is a third-party device that supports VLAN-aware bundle services. On Leaf-4, MAC-VRF1 is configured with two broadcast domains (BDs), BD-200 and BD-201. BD-200 and BD-201 are configured with the corresponding VLAN ID value encoded as Ethernet Tag ID in the EVPN routes and the VNI.

To allow Leaf-1 and Leaf-2 to interoperate with Leaf-4, the SR Linux devices are configured to advertise a non-zero Ethernet Tag ID in the EVPN routes, and process the Ethernet Tag ID in EVPN routes received for the MAC-VRF. When this Ethernet Tag ID is configured, all MAC-VRFs of the same bundle have the same import/export route target (RT), but different route distinguisher (RD), Ethernet Tag ID, and VXLAN VNI.

The Ethernet Tag ID can be set to a value in the range 0 to 16777215 (24-bit value). When the Ethernet Tag ID is set to a non-zero value, MAC/IP, IMET, and AD per-EVI routes for the MAC-VRF are advertised encoding the Ethernet Tag ID, configured with the **vlan-aware-bundle-eth-tag** parameter, into the `ethernet-tag-id` field of the routes. To interoperate in a VLAN-aware bundle broadcast domain where there are multiple vendors and multihomed CEs, all the vendors must advertise and process the AD per-EVI routes as per RFC 8584, where an AD per-EVI route is advertised per broadcast domain for the ES. SR Linux is fully compliant with RFC 8584.

For received routes, BGP processes the routes as usual, and imports them based on the import route-target. The `ethernet-tag-id` is part of the route-key, so BGP may keep multiple routes with same RD/RT/prefix but different `ethernet-tag-id`. For the routes imported in a specific MAC-VRF, only those routes whose `ethernet-tag-id` matches the locally configured Ethernet Tag ID are processed.

The feature is supported for the following:

- MAC-VRF network instances
- MAC-VRF network instances with IRB interfaces
- MAC-VRF with multihoming, for Layer 2 and Layer 3

9.1 Configuring the Ethernet Tag ID for VLAN-aware bundle services

Procedure

To configure VLAN-aware bundle services for SR Linux, you specify a value for the **vlan-aware-bundle-eth-tag** parameter.

When the **vlan-aware-bundle-eth-tag** is set to a non-zero value, routes are accepted only when the incoming `ethernet-tag-id` matches the configured value, for all route types imported in the MAC-VRF.

When the **vlan-aware-bundle-eth-tag** is set to zero (the default value):

- all received routes with `ethernet-tag-id = 0` are accepted (irrespective of type)
- for received routes with a non-zero `ethernet-tag-id` value, IMET and MAC/IP routes for the VXLAN instance are always accepted
- AD per-EVI routes with a non-zero ethernet-tag-id are rejected

Example

The following example configures a value for the **vlan-aware-bundle-eth-tag** parameter. All the EVPN routes advertised for the MAC-VRF contain this value in the `ethernet-tag-id` field.

```
--{ * candidate shared default }--[  ]--
# info network-instance mac_vrf_1 protocols bgp-evpn bgp-instance 1 routes
    network-instance mac_vrf_1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    routes {
                        bridge-table {
                            vlan-aware-bundle-eth-tag 1
                        }
                    }
                }
            }
        }
    }
```

9.2 Displaying the Ethernet Tag ID for VLAN-aware bundle services

Procedure

Use the **show network-instance** command to display the value configured for the **vlan-aware-bundle-eth-tag** parameter.

Example

```
--{ * candidate shared default }--[  ]--
# show network-instance mac_vrf_1 protocols bgp-evpn bgp-instance 1
================================================================================================
Net Instance   : mac_vrf_1
    bgp Instance 1 is enabled and up
------------------------------------------------------------------------------------------------
        VXLAN-Interface   : vxlan1.1
        evi               : 1
        ecmp              : 2
        default-admin-tag : 0
        oper-down-reason  : N/A
        EVPN Routes
            Next hop                       : 10.20.1.3/32 (network-instance "default" system0.0
                                             IPv4 address)
            VLAN Aware Bundle Ethernet tag : 1
            MAC/IP Routes                  : enabled
            IMET Routes                    : enabled, originating-ip 10.20.1.3/32
===============================================================================================
```

10 BGP and routing policy extensions for EVPN

This chapter describes extensions added to SR Linux BGP and routing policy configuration to facilitate EVPN configuration.

- BGP extensions for EVPN
- Routing policy extensions for EVPN

10.1 BGP extensions for EVPN

SR Linux supports Multi-Protocol BGP with AFI/SAFI EVPN. The following BGP features are relevant to EVPN in a VXLAN network:

- The EVPN address family can use eBGP or iBGP.
- eBGP multihop is supported on SR Linux; local-AS override can be used for iBGP per session.
- A supported configuration or design with eBGP is eBGP with local-as override on the session to an iBGP RR.
- Rapid-update and rapid-withdrawal for EVPN family are supported and are always expected to be enabled, especially along with multihoming.

  Note: Rapid-update is address-family specific, while rapid-withdrawal is generic for all address families.

- The BGP keep-all-routes option is supported for EVPN to avoid route-refresh messages attracting all EVPN routes when a policy changes or BGP-EVPN is enabled.
- The **receive-ipv6-next-hops** option does not apply to the EVPN address family.
- The **prefix-limit max-received-routes** and **threshold** options are supported for EVPN.
- The command **network-instance protocols bgp route-advertisement wait-for-fib-install** does not apply to EVPN.
- SR Linux BGP resolves BGP-EVPN routes' next-hops in the network-instance default route-table. If the next-hop is resolved, BGP can mark the route as u*> where *u* is used, * is valid, and > is best) and send the route to evpn_mgr if needed or reflected to other peers.

10.2 Routing policy extensions for EVPN

SR Linux includes support for applying routing policies to EVPN routes. You can specify the following match conditions in a policy statement:

- match routes based on EVPN route type
- match routes based on IP addresses and prefixes via prefix-sets for route types 2 and 5
- match routes based BGP encapsulation extended community

- match routes based on configured admin-tags

For information about configuring routing policies on SR Linux, see the *SR Linux Configuration Basics Guide*.

The following considerations apply to routing policies for EVPN:

- For IBGP neighbors, EVPN routes are imported and exported without explicit configuration of any policy at either the BGP or network-instance level.
- For EBGP neighbors, by default, routes are imported or exported based on the **ebgp-default-policy** configuration.
- When an explicit import/export route-target is configured in a network-instance bgp-instance, and an import/export policy is also configured on the same bgp-instance, the configured policy is used, and its route-target is added to the imported/exported route.
- When a network-instance policy and a peer policy are applied, they are executed as follows:
  - For export, the network-instance export policy is applied first, and the peer policy is applied afterwards (sequentially).
  - For import, peer import policy is applied first and network-instance import policy is applied afterwards (sequentially).
- Only one network-instance export and import policy is supported.

11 Dynamic multicast ID management

In SR Linux, multicast IDs (MCIDs) are multicast groups used by systems in the data path. MCIDs are allocated by the SR Linux mcid_mgr application so that the XDP layer can program different multicast forwarding modes in network instances.

For example, the system allocates two MCIDs for each MAC-VRF network instance:

- One MCID is used for the default multicast or flood list.
- Another MCID is dedicated to EVPN-enabled services to allow multi-homing correct forwarding of BUM traffic. Traffic coming from VXLAN that needs to be flooded automatically picks up the secondary multicast group that includes local single-homed subinterfaces plus Designated Forwarder (DF) subinterfaces, as opposed to the default flood list.

The available MCIDs in SR Linux are divided among the following applications:

- net_instance_mgr, which requests an MCID per default flood list (allocated per MAC-VRF when the MAC-VRF is configured)
- evpn_mgr, which requests an MCID per MAC-VRF (for EVPN multi-homing, allocated when VXLAN is configured on the MAC-VRF)
- l2_proxy_arp_nd_mgr, which requests an MCID per MAC-VRF when the proxy-ARP/ND flood options `flood gratuitous-arp`, `unknown-arp-rq` are set to `false` to suppress flooding over EVPN destinations.
- Individual multicast modules, which request an MCID per MFIB entry created by IGMP/MLD/PIM/MVPN.

Request-response model for MCIDs

To accommodate the SR Linux features that require MCIDs, the system allocates MCIDs dynamically, using a request-response model. When an application needs a new MCID, it makes a request to the mcid_mgr application, which allocates the MCID and publishes a response with the allocated MCID to the requesting application. The mcid_mgr application also programs XDP with the MCID.

MCID exhaustion

The system has a finite number of MCIDs to allocate. If mcid_mgr runs out of MCIDs it can allocate, the following applications are brought down when a new object is configured:

- mac-vrf
- vxlan-interface
- bgp-evpn mpls instance

When this occurs, the `oper-down-reason` for the application is displayed as `no-mcid`. When the `oper-state` for an application is changed to `down`, then brought back up, if there are no MCIDs available, the application is kept down with `oper-down-reason no-mcid`. For example, a change in `evi` automatically toggles the `oper-state` of an EVPN instance. If there are no MCIDs available following the `evi` change, the EVPN instance may not come back up.

Upon MCID exhaustion, mcid_mgr keeps trying to allocate MCIDs for the requesting multicast applications. If the system is rebooted, it may result in some VXLAN interfaces / MAC-VRFs that were not allocated MCIDs earlier receiving them at the expense of the multicast routes.

For l2_proxy_arp_nd_mgr, the application is brought down only if the flood options that require a new MCID are configured.

11.1 Displaying MCID information

Procedure

You can use an **info from state** command to display information about MCID allocation and MCID usage per application.

The command displays the total number of MCIDs available in the system, the total number of MCIDs currently in use, and the total number of pending MCIDs that mcid_mgr has not yet allocated.

For each application that requires MCIDs (Layer 2 proxy ARP/ND, MAC VRFs, MAC VRF BGP-EVPN, and VXLAN interfaces) you can display the number of MCIDs in use by the application, as well the number of MCIDs requested but not yet allocated for use by the application.

Example

The following command displays the current MCID usage:

```
--{ running }--[  ]--
# info from state system multicast multicast-ids statistics
    system {
        multicast {
            multicast-ids {
                statistics {
                    maximum-ids 16380
                    current-usage 3
                    total-pending 0
                    multicast-id-user-type mac-vrf {
                        current-usage 2
                        total-pending 0
                    }
                    multicast-id-user-type vxlan-interface {
                        current-usage 1
                        total-pending 0
                    }
                    multicast-id-user-type l2-proxy-arp-nd {
                        current-usage 0
                        total-pending 0
                    }
                    multicast-id-user-type mfib {
                        current-usage 0
                        total-pending 0
                    }
                    multicast-id-user-type mac-vrf-bgp-evpn {
                        current-usage 0
                        total-pending 0
                    }
                }
            }
        }
    }
```

12 Route internal tags (tag-sets in routing policies)

Route tags are pieces of information or identifiers attached to individual routes. They can be either external (protocol) tags or internal tags:

- External (protocol) tags are sent on the wire and are typically part of a routing protocol; for example, IS-IS route tags and OSPF route tags.
- Internal tags are not sent on the wire, and they have meaning only within an individual system. Internal tags are assigned to routes by policies or specific commands, and are used by policies to match on a group of routes. Internal tags are typically used to facilitate the matching of multiple routes at the same time or propagate route attributes between different routing protocols.

This chapter describes SR Linux support for internal tags and how to configure them.

About internal tags

In SR Linux, route internal tags are data structures that are attached to IP routes in the FIB or EVPN-specific routes. Internal tags are useful for carrying protocol information such as communities, area-id, as-path, and so on, from one routing protocol context to another. For example, BGP can push its attribute fields, such as local preference, into the FIB as an internal tag. On export, the internal tag associated with the route can be matched by export policies to map the tag into any other metric for a routing protocol.

Every FIB route can be associated with an array of tags. The number of array elements is limited to a maximum of 2 in the current release.

To define internal tags in SR Linux, you configure tag-sets at the `routing-policy` level, and refer to the tag-sets by name at other configuration levels. See Configuring route internal tags.

SR Linux applications that support internal tags

The following SR Linux applications can write internal tags:

- EVPN and Layer 2 proxy-ARP/ND
  EVPN and Layer 2 proxy-ARP/ND support writing internal tags for all MAC/IP advertisement routes generated out of the active MACs in the mac-table or IP-to-MAC bindings in the proxy-ARP/ND tables, respectively.
- Layer 3 ARP/ND
  SR Linux supports writing internal tags into ARP/ND entries that generate an EVPN MAC/IP advertisement route and also into the ARP/ND entries that generate an arp-nd host route into the route-table.
- Layer 3 routes
  In SR Linux, BGP can write internal tags into the route-table entries for a routed network-instance. In this case, all routes to be exported as prefix routes are assigned the configured internal tag.

  The route-table includes a state leaf-list with the allocated `internal-tags` tag values that indicate whether a specific route entry is associated with an internal tag.

Routing policies and internal tags

An import or export policy can write an internal tag as a configured `action` or `default-action`. These actions are supported on BGP policies applied to the supported families; that is, `ipv4-unicast`, `ipv6-unicast`, and `evpn`.

When internal tags are used in BGP import policies, the policies can be applied at the default network-instance level (including global, per-group, or per-peer policies), MAC-VRF network-instance level, or IP-VRF network-instance level.

When internal tags are used in BGP export policies, the policies can be applied at the MAC-VRF or IP-VRF network-instance level, or IP-VRF/default network-instance level for inter-instance policies.

When an import policy matches an IPv4 BGP route, IPv6 BGP route, or EVPN IP prefix route (route type 5) and adds a tag-set, the corresponding route created in the route-table contains the internal tag and is exposed in the route-table entry `internal-tags`.

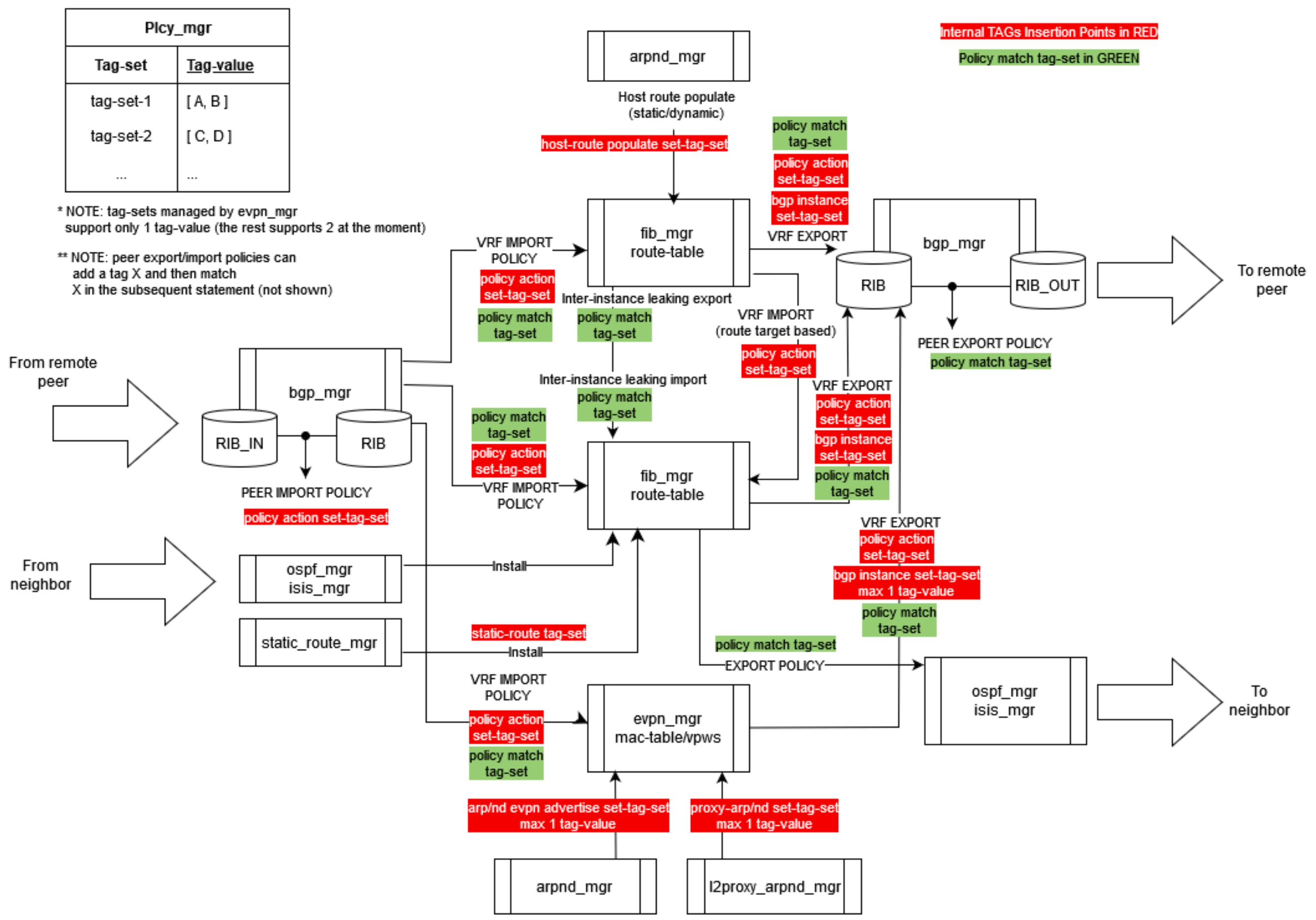
When the import policy matches an EVPN MAC/IP advertisement route (route type 2) and adds a tag-set as an action, the corresponding created MAC and/or ARP/ND entry is associated with the internal tag. Matching on any other EVPN route type and adding an internal tag has no effect, since the rest of the EVPN route types do not create entries into the MAC table, ARP/ND table, or route table.

A MAC/IP VRF export policy can add a tag-set as an action in the following cases

- On a MAC-VRF, the `bgp-vpn.bgp-instance.export-policy` can match EVPN MAC/IP routes (with or without IP address) generated for dynamic or static entries in the mac-table or proxy-ARP/ND table; that is, basically for any entry that generates a MAC/IP advertisement route in the context of the MAC-VRF. In those policies, a tag-set can be added. The same tag-set can be matched on peer/group-level export policies in the default network-instance.
- On an IP-VRF, the `bgp-vpn.bgp-instance.export-policy` can match any route-table entry that generates an EVPN IP prefix route in the context of the IP-VRF. In these policies, a tag-set can be added as an action, and the same tag-set can be matched by a peer/group-level export policy in the default network-instance.
- On an IP-VRF or default network-instance, the `inter-instance-policies.export-policy` can match any route-table entry and make those entries available for leaking. The internal tag can be used by an `inter-instance-policies.import-policy` to match the routes that are actually leaked into the destination Layer 3 network-instance.

Insertion points for internal tags

The following diagram shows the SR Linux applications that can write an internal tag, insertion points for the internal tags, and the policy points where internal tag matching can take place.


Plcy_mgr
Tag-set
Tag-value
tag-set-1
[A, B]
tag-set-2
[C, D]
...
...
* NOTE: tag-sets managed by evpn_mgr support only 1 tag-value (the rest supports 2 at the moment)
** NOTE: peer export/import policies can add a tag X and then match X in the subsequent statement (not shown)
Internal TAGs Insertion Points in RED
Policy match tag-set in GREEN
arpnd_mgr
Host route populate (static/dynamic)
host-route populate set-tag-set
policy match tag-set
policy action set-tag-set
bgp instance set-tag-set
VRF EXPORT
fib_mgr route-table
VRF IMPORT POLICY
policy action set-tag-set
policy match tag-set
Inter-instance leaking export
policy match tag-set
bgp_mgr
RIB
RIB_OUT
To remote peer
PEER EXPORT POLICY
policy match tag-set
VRF IMPORT (route target based)
policy action set-tag-set
VRF EXPORT
policy action set-tag-set
bgp instance set-tag-set
policy match tag-set
From remote peer
bgp_mgr
RIB_IN
RIB
PEER IMPORT POLICY
policy action set-tag-set
Inter-instance leaking import
policy match tag-set
policy match tag-set
policy action set-tag-set
VRF IMPORT POLICY
fib_mgr route-table
VRF EXPORT
policy action set-tag-set
bgp instance set-tag-set max 1 tag-value
policy match tag-set
From neighbor
ospf_mgr isis_mgr
Install
static_route_mgr
static-route tag-set
Install
policy match tag-set
EXPORT POLICY
ospf_mgr isis_mgr
To neighbor
VRF IMPORT POLICY
policy action set-tag-set
policy match tag-set
evpn_mgr mac-table/vpws
arp/nd evpn advertise set-tag-set max 1 tag-value
proxy-arp/nd set-tag-set max 1 tag-value
arpnd_mgr
l2proxy_arpnd_mgr

12.1 Order of precedence when multiple applications write internal tags

As described in the previous section, multiple SR Linux applications can write internal tags for a route. For example, a newly created ARP-ND host 10.0.0.1/32 route can be associated with tag-set `TAG-1`, and at the same time an export policy applied on IP-VRF-1, where 10.0.0.1/32 is installed, may have an action to write tag-set `TAG-2` or the same route.

In this case, SR Linux applies precedence rules to determine the tag-set that gets applied to the route. The following sections describe the precedence rules applied when multiple applications attempt to write an internal tag on the same route.

Writing a tag for MAC/IP routes generated for a MAC-VRF

The order in which the internal tag is applied is as follows:

(proxy-arp/nd tag (mac-vrf) + vrf-export-policy tag (mac-vrf level)) > (bgp-evpn.bgp-instance tag (mac-vrf) +vrf-export-policy tag (mac-vrf level))

That is, up to two tag-values can be written. However, if the route already has a tag-value, then the bgp-evpn.bgp-instance tag is not used. The vrf-export-policy tag action always appends to the route's tag (if already present on the route), or the default tag (under the bgp-instance) if configured.

Writing a tag for MAC/IP routes generated for a MAC-VRF with IRB

(arp/nd evpn tag (irb) + vrf-export-policy tag (mac-vrf level)) > (bgp-evpn.bgp-instance tag (mac-vrf) +vrf-export-policy tag (mac-vrf level))

In the same way as proxy-arp/nd tags, up to two tag-values can be written, and if the route already has a tag-value, then the bgp-evpn.bgp-instance tag is not used. The vrf-export-policy tag action always appends to the route's tag (if already present on the route), or the default tag (under the bgp-instance) if configured.

Writing a tag for ARP-ND host routes in an IP-VRF

Tags are appended until a maximum of two is reached, so the vrf-export policy action does not overwrite the tag if that is already present. The default tag of the instance is used when arpnd does not add a tag.

A peer-export policy uses this precedence:

(arp/nd host-route tag (irb) + vrf-export policy tag) > (bgp-evpn.bgp-instance tag (ip-vrf) + vrf-export policy tag)

Writing a tag for IP routes in an IP-VRF (irrespective of the owner)

Tag-values are appended until a maximum of two is reached, so the vrf-export policy action does not overwrite the import-policy tag if that is already present. The default tag of the instance is used when import-policy does not add a tag.

A peer-export policy uses this precedence:

(peer-import-policy-tag + vrf-export policy tag) > (bgp-evpn.bgp-instance tag (ip-vrf) + vrf-export policy tag)

In cases where the routes are static routes, IS-IS routes, OSPF routes, and so on, when exported into EVPN IP prefix routes, there is no peer-import-policy.

In cases where there are BGP PE-CE routes that are imported into the IP-VRF route-table and readvertised as EVPN IP prefix routes, there are peer-import-policies, and the preference is as above.

Writing a tag for MAC/IP routes in MAC-VRF or routes in IP-VRF via multiple import policies

If multiple import-policies match the same route and try to write a tag, writing is allowed until a maximum of two tag-values is reached. Note that applying "multiple import-policies" refers to policy chaining at the peer-import level.

Writing a tag for routes via vrf-import and peer import policies

If both vrf-import and peer-import policies apply to a route, and each step inserts a tag-set, then the tag-set applied as a result of the vrf-import policy takes precedence.

Writing a tag for BGP routes in an ECMP set

If BGP creates an ECMP set for a prefix, when installing such a prefix in the FIB, BGP copies the tag-set of the best route into fib_mgr. This is what other protocols would see when importing the prefix from fib_mgr.

This also applies to BGP VPN routes that are received with different RDs and combined in an ECMP set.

12.2 Configuring route internal tags

Procedure

To configure route internal tags, you configure a tag-set and configure one or more applications to write the tag-set to routes.

Example: Configure a tag-set

```
--{ * candidate shared default }--[  ]--
# info routing-policy
    routing-policy {
        tag-set ts1 {
            tag-value [
                1
                2
            ]
        }
    }
```

Example: Match a tag-set in a routing policy

```
--{ +* candidate shared default }--[  ]--
# info routing-policy policy match_tag_1
    routing-policy {
        policy match_tag_1 {
            statement 1 {
                match {
                    internal-tags {
                        tag-set [
                            ts1
                        ]
                    }
                }
            }
        }
    }
```

Example: Write a tag-set as an action in a routing policy

```
--{ * candidate shared default }--[  ]--
# info routing-policy policy d1
    routing-policy {
        policy t1 {
            default-action {
                internal-tags {
                    set-tag-set [
                        ts1
                    ]
                }
            }
        }
    }
```

Example: Write internal tags for MAC/IP advertisement routes generated out of the active MACs in the MAC table

```
--{ * candidate shared default }--[  ]--
# info network-instance vrf1
```

```
    network-instance vrf1 {
        type mac-vrf
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    internal-tags {
                        set-tag-set [
                            ts1
                        ]
                    }
                }
            }
        }
    }
```

Example: Write internal tags for IP-to-MAC bindings in proxy-ARP/ND tables

```
--{ * candidate shared default }--[  ]--
# info  network-instance vrf1 bridge-table proxy-arp evpn
    network-instance vrf1 {
        bridge-table {
            proxy-arp {
                evpn {
                    internal-tags {
                        set-tag-set [
                            ts1
                        ]
                    }
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info network-instance vrf1 bridge-table proxy-nd evpn
    network-instance vrf1 {
        bridge-table {
            proxy-nd {
                evpn {
                    internal-tags {
                        set-tag-set [
                            ts1
                        ]
                    }
                }
            }
        }
    }
```

Example: Write internal tags into ARP/ND entries that generate an EVPN MAC/IP advertisement route

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv4 arp evpn
    interface ethernet-1/1 {
        subinterface 1 {
            ipv4 {
                arp {
                    evpn {
                        advertise dynamic {
                            internal-tags {
                                set-tag-set [
                                    ts1
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv6 neighbor-discovery
    interface ethernet-1/1 {
        subinterface 1 {
            ipv6 {
                neighbor-discovery {
                    evpn {
                        advertise dynamic {
                            internal-tags {
                                set-tag-set [
                                    ts1
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
```

Example: Write internal tags into ARP/ND entries that generate an ARP/ND host route into the route table

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv4 arp
    interface ethernet-1/1 {
        subinterface 1 {
            ipv4 {
                arp {
                    host-route {
                        populate evpn {
                            internal-tags {
                                set-tag-set [
                                    ts1
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface 1 ipv6 neighbor-discovery
    interface ethernet-1/1 {
        subinterface 1 {
            ipv6 {
                neighbor-discovery {
                    host-route {
                        populate evpn {
                            internal-tags {
                                set-tag-set [
                                    ts1
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
```

Example: Write internal tag into routes to be exported as EVPN IP prefix routes

```
--{ * candidate shared default }--[  ]--
# info network-instance ip-vrf-1 protocols bgp-evpn bgp-instance 1
    network-instance ip-vrf-1 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    internal-tags {
                        set-tag-set [
                            ts1
                        ]
                    }
                }
            }
        }
    }
```

13 EVPN configuration examples

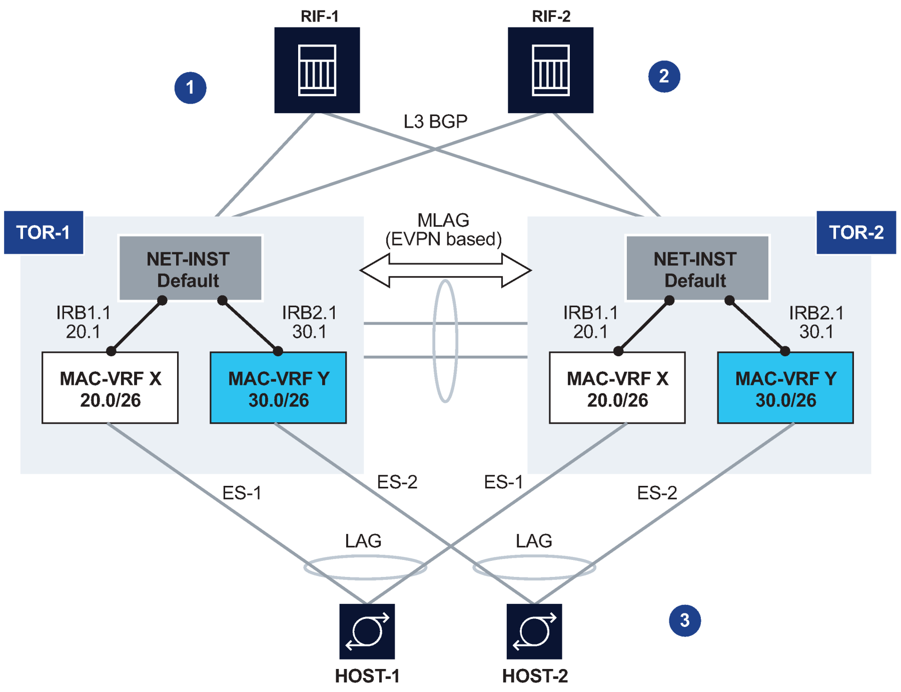
This chapter contains examples of configurations that use EVPN features.

- All-active redundant connectivity example
- Hierarchical active-active connectivity example
- EVPN multihoming as standalone solution for MC-LAG

13.1 All-active redundant connectivity example

The following figure shows an example of using EVPN multihoming as a standalone and self-contained multichassis solution.

Figure 19: Active-active connectivity example



This example uses the following features:

1. **redundancy**

   TOR redundancy is based on an all-active redundancy model.

2. **Layer 3 connectivity**

   - An anycast gateway solution is used on the IRB subinterfaces so that upstream traffic is always routed locally on the TOR receiving the traffic.

- South-to-north traffic is sent to the active link and routed by the local IRB subinterface. In case of failure on TOR-1, TOR-2 is ready to forward on the anycast gateway IRB, without the need to wait for any VRRP protocol to converge.
- North-to-south traffic is load-balanced by the fabric to the two TORs. ARP/ND entries are synchronized across both TORs. Host routes could be optionally created and advertised in BGP from the directly connected TOR to avoid tromboning in the downstream direction.

3. **Layer 2 connectivity**
   - Servers do not need to run any xSTP protocols. The NDF TOR brings down the port and signals LOS to the server.
   - This solution places no requirements on the servers.

This example imposes the use of a LAG on the server. The LAG can use LACP or not. The servers are unaware that the LAG is connected to two systems instead of only one.

13.1.1 Configuration for all-active connectivity example

Leaf 1 in Figure 19: Active-active connectivity example has the following configuration. Leaf 2 would have an equivalent configuration.

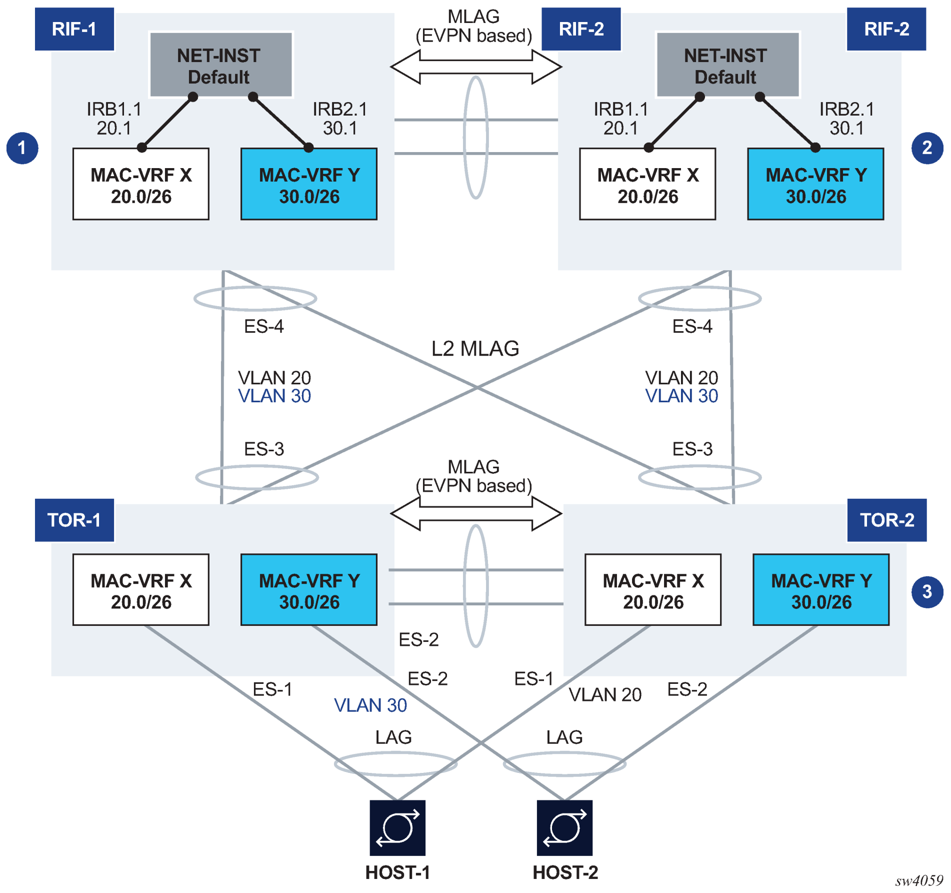
```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Leaf-1# info
// two IRB interfaces with anycast-gw configuration is added
interface irb1 {
    subinterface 1 {
        ipv4 {
            admin-state enable
            address 20.0.0.1/24 {
                anycast-gw true
                primary
            }
            arp {
                learn-unsolicited true
                evpn {
                    advertise dynamic {
                    }
                }
            }
        }
        anycast-gw {
        }
    }
}
interface irb2 {
    subinterface 1 {
        ipv4 {
            admin-state enable
            address 30.0.0.1/24 {
                anycast-gw true
                primary
            }
            arp {
                learn-unsolicited true
                evpn {
                    advertise dynamic {
                    }
                }
            }
        }
        anycast-gw {
        }
    }
}
// lags associated with the ethernet-segments.
 In this case static, but they can be lacp based too.
interface lag1 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 20
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
interface lag2 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 30
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
// ES configuration
system {
    network-instance {
        protocols {
            evpn {
                ethernet-segments {
                    bgp-instance 1 {
                        ethernet-segment ES-1 {
                            admin-state enable
                            esi 00:01:00:00:00:00:00:00:00:01
                            interface lag1
                        }
                        ethernet-segment ES-2 {
                            admin-state enable
                            esi 00:02:00:00:00:00:00:00:00:02
                            interface lag2
                        }
                    }
                }
            }
        }
    }
}
// MAC-VRFs
```

```
network-instance MAC-VRF-X {
    type mac-vrf
    admin-state enable
    interface irb1.1 {
    }
    interface lag1.1 {
    }
    vxlan-interface vxlan1.20 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.20
                evi 20
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
network-instance MAC-VRF-Y {
    type mac-vrf
    admin-state enable
    interface irb2.1 {
    }
    interface lag2.1 {
    }
    vxlan-interface vxlan1.30 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.30
                evi 30
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
// default network-instance configuration
network-instance default {
    type default
    admin-state enable
    description "Default network instance"
    router-id 1.1.1.1
    interface ethernet-1/1.1 {
    }
    interface ethernet-1/20.1 {
    }
    interface irb1.1 {
    }
    interface irb2.1 {
    }
    interface system0.0 {
    }
    protocols {
        bgp {
            admin-state enable
            afi-safi ipv4-unicast {
                admin-state enable
            }
            autonomous-system 1234
            router-id 1.1.1.1
            group eBGP-spines {
                admin-state enable
                export-policy export-all
                peer-as 4567
                afi-safi ipv4-unicast {
                    admin-state enable
                }
            }
            group evpn-mh {
                admin-state enable
                export-policy export-all
                peer-as 1234
                afi-safi evpn {
                    admin-state enable
                }
            }
            neighbor 2.2.2.2 {
                admin-state enable
                peer-group evpn-mh
            }
            neighbor 10.1.4.4 {
                admin-state enable
                peer-group eBGP-spines
            }

        }
    }
}
// vxlan interfaces for the MH configuration
tunnel-interface vxlan1 {
    vxlan-interface 20 {
        type bridged
        ingress {
            vni 20
        }
    }
    vxlan-interface 30 {
        type bridged
        ingress {
            vni 30
        }
    }
}
```

13.2 Hierarchical active-active connectivity example

The following figure shows an example that makes use of a Layer 2 and a Layer 3 multihoming solution.

Figure 20: Hierarchical active-active connectivity example



This example uses the following features:

- **Leaf and Spine configuration**
  - There are two multichassis pairs at the leaf and the spine levels.
  - The leaf pair runs all-active multihoming on its own ESs.

    The access hosts or spines are separate and independent ESs. The figure shows three ESs, one per host at the access, and one for the connectivity to the spine layer.

    Note: This is only one ES in spite of the number of spine nodes, which could be 2 or 4.

  - The spine layer runs all-active multihoming with a single ES to the leaf layer, associated with the LAG attached to the leaf layer.
  - The two tiers are independent; there are no control plane protocols between them, except for LACP if used.

    This means that the leaf and spine layer could theoretically be of a different vendor.

- **Layer 3 connectivity**

  - An anycast gateway solution is used on the spine layer, on the IRB subinterfaces, so that upstream traffic is always routed locally on the leaf receiving the traffic.
  - South-to-north flows are sent to only one link at a time, so there is not duplication.
  - North-to-south traffic is load-balanced by the fabric to the two spines and for the spines load-balanced to the leafs. ARP/ND entries are synchronized across both spines. Host routes can optionally be created and advertised in BGP from the directly connected spine to avoid tromboning in the downstream direction.
- **Layer 2 connectivity**
  - Hosts are running LAG with or without LACP.
  - Leafs and spines are connected by standard Layer 2 LAGs that carry the VLANs for the two broadcast domains shown in Figure 20: Hierarchical active-active connectivity example.

13.2.1 Configuration for hierarchical active-active connectivity example

Spine 1 in Figure 20: Hierarchical active-active connectivity example has the following configuration. The configuration for Spine 2 would be equivalent.

```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Spine-1# info
// two IRB interfaces with anycast-gw configuration is added
interface irb1 {
    subinterface 1 {
        ipv4 {
            admin-state enable
            address 20.0.0.1/24 {
                anycast-gw true
                primary
            }
            arp {
                learn-unsolicited true
                evpn {
                    advertise dynamic {
  // for ARP synchronization across MH leaf nodes
                    }
                }
            }
        }
        anycast-gw {
        }
    }
}
interface irb2 {
    subinterface 1 {
        ipv4 {
            admin-state enable
            address 30.0.0.1/24 {
                anycast-gw true
                primary
            }
            arp {
                learn-unsolicited true
                evpn {
                    advertise dynamic {
                    }
                }
            }
        }
        anycast-gw {
        }
    }
}
// lags associated with the ethernet-segment.
 In this case static, but they can be lacp based too.
interface lag1 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 20
                }
            }
        }
    }
    subinterface 2 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 30
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
// ES configuration
system {
    network-instance {
        protocols {
            evpn {
                ethernet-segments {
                    bgp-instance 1 {
                        ethernet-segment ES-4 {
                            admin-state enable
                            esi 00:44:44:44:44:44:44:00:00:04
                            interface lag1
                        }
                    }
                }
            }
        }
    }
}
// MAC-VRFs
network-instance MAC-VRF-X {
    type mac-vrf
    admin-state enable
    interface irb1.1 {
    }
    interface lag1.1 {
    }
    vxlan-interface vxlan1.20 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.20
                evi 20
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
network-instance MAC-VRF-Y {
    type mac-vrf
    admin-state enable
    interface irb2.1 {
    }
    interface lag1.2 {
    }
    vxlan-interface vxlan1.30 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.30
                evi 30
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
// default network-instance configuration
network-instance default {
    type default
    admin-state enable
    description "Default network instance"
    router-id 1.1.1.1
    interface ethernet-1/1.1 {
    }
    interface ethernet-1/20.1 {
    }
    interface irb1.1 {
    }
    interface irb2.1 {
    }
    interface system0.0 {
    }
    protocols {
        bgp {
            admin-state enable
            afi-safi ipv4-unicast {
                admin-state enable
            }
            autonomous-system 1234
            router-id 1.1.1.1
            group eBGP-spines {
                admin-state enable
                export-policy export-all
                peer-as 4567
                afi-safi ipv4-unicast {
                    admin-state enable
                }
            }
            group evpn-mh {
                admin-state enable
                export-policy export-all
                peer-as 1234
                afi-safi evpn {
                    admin-state enable
                }
            }
            neighbor 2.2.2.2 {
                admin-state enable
                peer-group evpn-mh
            }
            neighbor 10.1.4.4 {
                admin-state enable
                peer-group eBGP-spines
            }

        }
    }
}
// vxlan interfaces for the MH configuration
tunnel-interface vxlan1 {
    vxlan-interface 20 {
        type bridged
        ingress {
            vni 20
        }
    }
    vxlan-interface 30 {
        type bridged
        ingress {
            vni 30
        }
    }
}
```

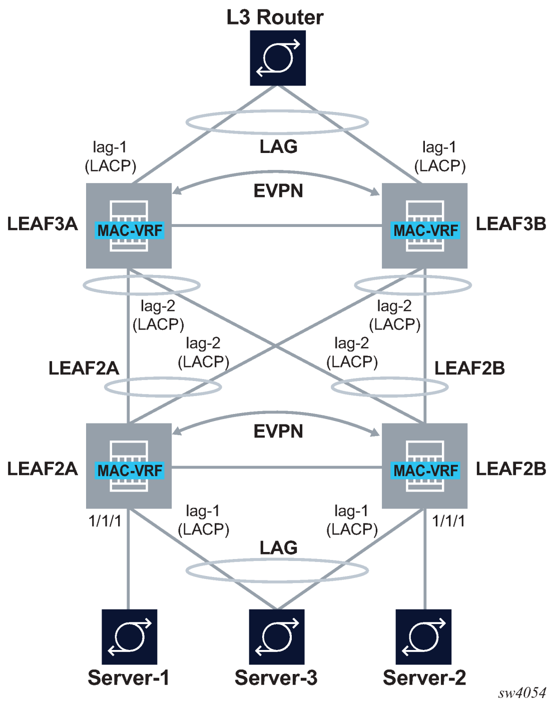
Leaf 1 in Figure 20: Hierarchical active-active connectivity example has the following configuration. The configuration for Leaf 2 would be equivalent.

```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Leaf-1# info
// lags associated with the ethernet-segments.
 In this case static, but they can be lacp based too.
interface lag1 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 20
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
interface lag2 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 30
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
interface lag3 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 20
                }
            }
        }
    }
    subinterface 2 {
        vlan {
            encap {
                single-tagged {
                    vlan-id 30
                }
            }
        }
    }
    lag {
        lag-type static
    }
}
// ES configuration
system {
    network-instance {
        protocols {
            evpn {
                ethernet-segments {
                    bgp-instance 1 {
                        ethernet-segment ES-1 {
                            admin-state enable
                            esi 00:11:11:11:11:11:11:00:00:01
                            interface lag1
                        }
                        ethernet-segment ES-2 {
                            admin-state enable
                            esi 00:22:22:22:22:22:22:00:00:02
                            interface lag2
                        }
                        ethernet-segment ES-3 {
                            admin-state enable
                            esi 00:33:33:33:33:33:33:00:00:03
                            interface lag3
                        }
                    }
                }
            }
        }
    }
}
// MAC-VRFs
network-instance MAC-VRF-X {
    type mac-vrf
    admin-state enable
    interface lag1.1 {
    }
    interface lag3.1 {
    }
    vxlan-interface vxlan1.20 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.20
                evi 20
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
network-instance MAC-VRF-Y {
    type mac-vrf
    admin-state enable
    interface lag2.1 {
    }
    interface lag3.1 {
    }
    vxlan-interface vxlan1.30 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                admin-state enable
                vxlan-interface vxlan1.30
                evi 30
            }
        }
        bgp-vpn {
            bgp-instance 1 {
            }
        }
    }
}
// default network-instance configuration
network-instance default {
    type default
    admin-state enable
    description "Default network instance"
    router-id 1.1.1.1
    interface ethernet-1/1.1 {
    }
    interface system0.0 {
    }
    protocols {
        bgp {
            admin-state enable
            afi-safi ipv4-unicast {
                    admin-state enable
                }
            autonomous-system 1234
            router-id 1.1.1.1
            group evpn-mh {
                admin-state enable
                export-policy export-all
                peer-as 1234
                afi-safi evpn {
                    admin-state enable
                }
            }
            neighbor 2.2.2.2 {
                admin-state enable
                peer-group evpn-mh
            }
        }
    }
}
// vxlan interfaces for the MH configuration
tunnel-interface vxlan1 {
    vxlan-interface 20 {
        type bridged
        ingress {
            vni 20
        }
    }
    vxlan-interface 30 {
        type bridged
        ingress {
            vni 30
        }
    }
}
```

13.3 EVPN multihoming as standalone solution for MC-LAG

EVPN multihoming is not only used in overlay DCs, but also as a standalone solution for multihoming in Layer 2 access networks with no VXLAN. The following figure shows this usage.

Figure 21: EVPN multihoming as standalone MC-LAG solution

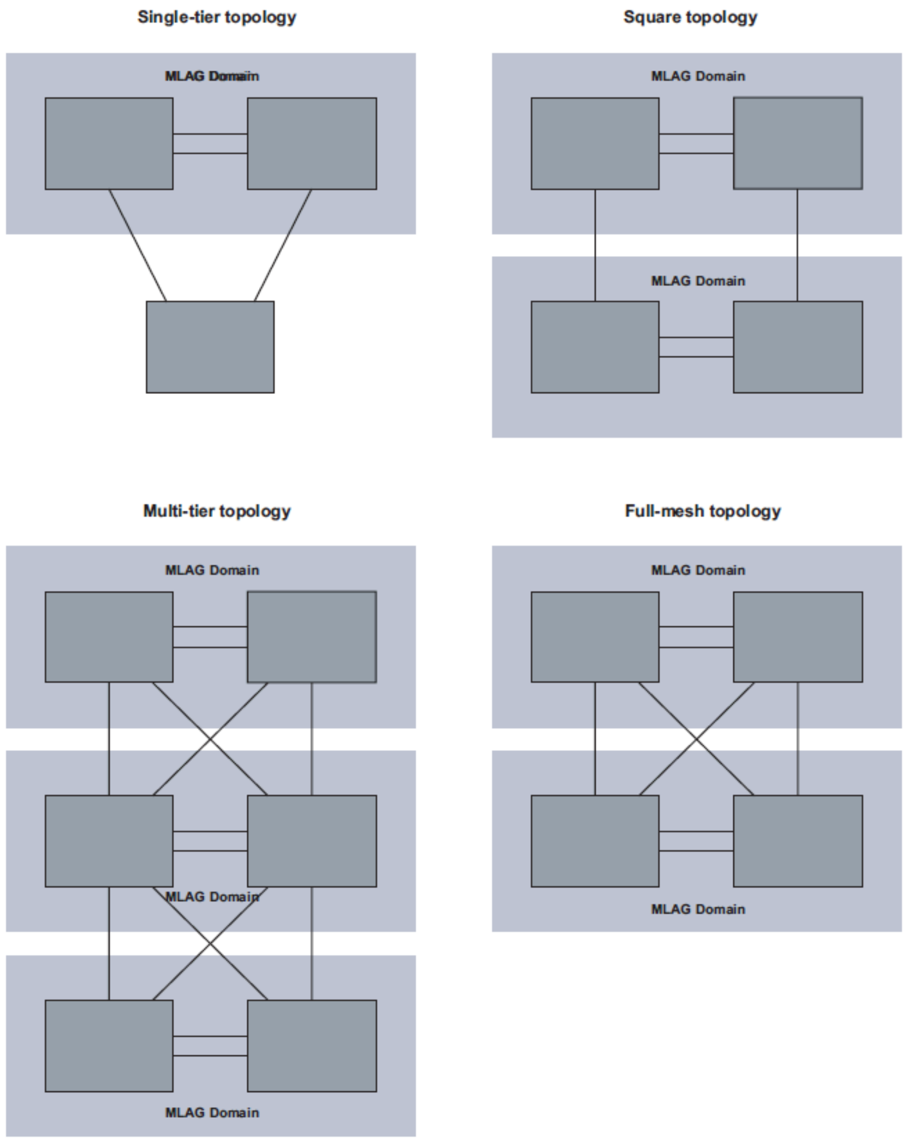


On the left side of the figure, EVPN multihoming is used as a standalone multihoming solution for leaf nodes connected via bridged subinterfaces.

Leafs of a layer do not use VXLAN to get connected to the higher layer. In this case, EVPN sessions are configured locally within each multihoming pair so that EVPN handles DF Election, split-horizon and synchronization of MAC and ARP entries. However, the leafs of different layers are not connected through any IP fabric, so no VXLAN or EVPN is needed end-to-end.

In this configuration, EVPN provides an alternative to MC-LAG solutions, being able to match all the topologies that other MC-LAG solutions support. These topologies include single-tier, multi-tier, square, or full-mesh/bow-tie topologies, as shown in the following figure. EVPN multihoming is supported in all of them as a replacement of MC-LAG.

Figure 22: MLAG topologies



13.3.1 Configuration for EVPN multihoming as standalone MC-LAG

LEAF2A in Figure 21: EVPN multihoming as standalone MC-LAG solution has the following configuration:

```
// lag1 connects LEAF2A to server-3
 and is associated to an all-active Ethernet Segment.
--{ candidate shared default }--[  ]--
A:LEAF2A# info interface lag*
    interface lag1 {
        admin-state enable
        vlan-tagging true
        subinterface 10 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id 10
                    }
                }
            }
        }
        lag {
            lag-type static
 // lag-type could also be lacp, in which case the
 system-id/key must match on lag1 of LEAF2B
            member-speed 10G
        }
    }
// lag2 connects LEAF2A to LEAF3A and LEAF3B.
 This LAG is an access LAG (does not carry vxlan) associated to
 an all-active Ethernet Segment
    interface lag2 {
        admin-state enable
        vlan-tagging true
        subinterface 10 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id 10
                    }
                }
            }
        }
        lag {
            lag-type static
 // lag-type could also be lacp, in which case the
 system-id/key must match on lag2 of LEAF2B
            member-speed 10G
        }
    }
// A vxlan-interface is created for the inter-chassis traffic
--{ candidate shared default }--[  ]-
A:LEAF2A# info tunnel-interface vxlan1 vxlan-interface 10
    tunnel-interface vxlan1 {
        vxlan-interface 10 {
            type bridged
            ingress {
                vni 10
            }
        }
    }
// the Ethernet Segments associated to lag1 and lag2
--{ candidate shared default }--[  ]--
A:LEAF2A# info system network-instance protocols evpn
    system {
        network-instance {
            protocols {
                evpn {
                    ethernet-segments {
                        bgp-instance 1 {
                            ethernet-segment ES-leaf1-leaf2.CE1 {
                                admin-state enable
                                esi 00:12:12:12:12:12:12:00:00:01
                                interface lag1
                            }
                            ethernet-segment ES-leaf1-leaf2.Spines {
                                admin-state enable
                                esi 00:12:12:12:12:12:12:00:00:02
                                interface lag2
                            }
                        }
                    }
                }
            }
        }
    }
// the mac-vrf uses lag sub-interfaces for the connectivity to rest of
 the leaf nodes and servers, and a vxlan-subinterface for the connectivity
 to LEAF2B
--{ candidate shared default }--[  ]--
A:LEAF2A# info network-instance Blue-MAC-VRF-10
    network-instance Blue-MAC-VRF-10 {
        type mac-vrf
        interface ethernet-1/1.10 {
            // this is connected to a single-homed access server-1
        }
        interface lag1.10 {
            // access lag - multi-homed to access server-3
        }
        interface lag2.10 {
            // multi-homed to spines
        }
        vxlan-interface vxlan1.10 {
            // vxlan-interface used for inter-chassis connectivity only
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.10
                    evi 10
                }
            }
            bgp-vpn {
            }
        }
    }
```

LEAF2A in Figure 21: EVPN multihoming as standalone MC-LAG solution has the following configuration:

```
// lag1 connects LEAF2B to server-3 and is associated to
 an all-active Ethernet Segment
--{ candidate shared default }--[  ]--
A:LEAF2B# info interface lag*
    interface lag1 {
        admin-state enable
        vlan-tagging true
        subinterface 10 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id 10
                    }
                }
            }
        }
        lag {
            lag-type static
 // lag-type could also be lacp, in which case the
 system-id/key must match on lag1 of LEAF2A
            member-speed 10G
        }
    }
// lag2 connects LEAF2B to LEAF3A and LEAF3B.
 This LAG is an access LAG (does not carry vxlan) associated to
 an all-active Ethernet Segment
    interface lag2 {
        admin-state enable
        vlan-tagging true
        subinterface 10 {
            type bridged
            vlan {
                encap {
                    single-tagged {
                        vlan-id 10
                    }
                }
            }
        }
        lag {
            lag-type static
 // lag-type could also be lacp, in which case the
 system-id/key must match on lag2 of LEAF2B
            member-speed 10G
        }
    }
// A vxlan-interface is created for the inter-chassis traffic
--{ candidate shared default }--[  ]--
A:LEAF2B# info tunnel-interface vxlan1 vxlan-interface 10
    tunnel-interface vxlan1 {
        vxlan-interface 10 {
            type bridged
            ingress {
                vni 10
            }
        }
    }
// the Ethernet Segments associated to lag1 and lag2
--{ candidate shared default }--[  ]--
A:LEAF2B# info system network-instance protocols evpn
    system {
        network-instance {
            protocols {
                evpn {
                    ethernet-segments {
                        bgp-instance 1 {
                            ethernet-segment ES-leaf1-leaf2.CE1 {
                                admin-state enable
                                esi 00:12:12:12:12:12:12:00:00:01
                                interface lag1
                            }
                            ethernet-segment ES-leaf1-leaf2.Spines {
                                admin-state enable
                                esi 00:12:12:12:12:12:12:00:00:02
                                interface lag2
                            }
                        }
                    }
                }
            }
        }
    }
```

```
// the mac-vrf uses lag sub-interfaces for the connectivity to rest of the
 leaf nodes and servers, and a vxlan-subinterface for the connectivity
 to LEAF2B
--{ candidate shared default }--[  ]--
A:LEAF2B# info network-instance Blue-MAC-VRF-10
    network-instance Blue-MAC-VRF-10 {
        type mac-vrf
        interface ethernet-1/2.10 {
            !!! this is connected to a single-homed access server-2
        }
        interface lag1.10 {
            !!! access lag - multi-homed to access server-3
        }
        interface lag2.10 {
            !!! multi-homed to spines
        }
        vxlan-interface vxlan1.10 {
            !!! vxlan-interface used for inter-chassis connectivity only
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.10
                    evi 10
                }
            }
            bgp-vpn {
            }
        }
    }
```

Customer document and product support

Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback