Nokia Service Router Linux

EVPN-VXLAN Guide
Release 21.3

3HE 16831 AAAA TQZZA

Edition: 1

March 2021

Table of Contents

1 Getting started

This chapter describes this document, includes summaries of changes from previous releases and precautionary messages, and lists command conventions.

1.1 About this document

This document describes basic configuration for EVPN Layer 2 (L2) and Layer 3 (L3) 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 that will be released in later maintenance loads. Refer to the *SR Linux Release Notes* for information on features supported in each load.

1.2 What's new

This is the first release of this document. In future releases, a table will define new or change information for the release.

1.3 Precautionary messages

Observe all dangers, warnings, and cautions in this document to avoid injury or equipment damage during installation and maintenance. Follow the safety procedures and guidelines when working with and near electrical equipment.

Table 1 describes information symbols contained in this document.

Table 1 **Information symbols**

Symbol	Meaning	Description
⚡	Danger	Warns that incorrect handling and installation could result in bodily injury. 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	Warns that incorrect handling and installation could result in equipment damage or loss of data.
⛔	Caution	Warns that incorrect handling may reduce your component or system performance.
➡	Note	Notes contain suggestions or additional operational information.

1.4 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**.
- Angle brackets (< >) indicate an item that is not used verbatim. For example, for the command **show ethernet <***name***>**, *name* should be replaced with the name of the interface.
- 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 Overview

This chapter contains the following topics:

2.1 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 the de-facto standard technology in multi-tenant 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 multi-tenant DCs, the EVPN control plane was adapted for VXLAN tunnels in RFC8365.

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

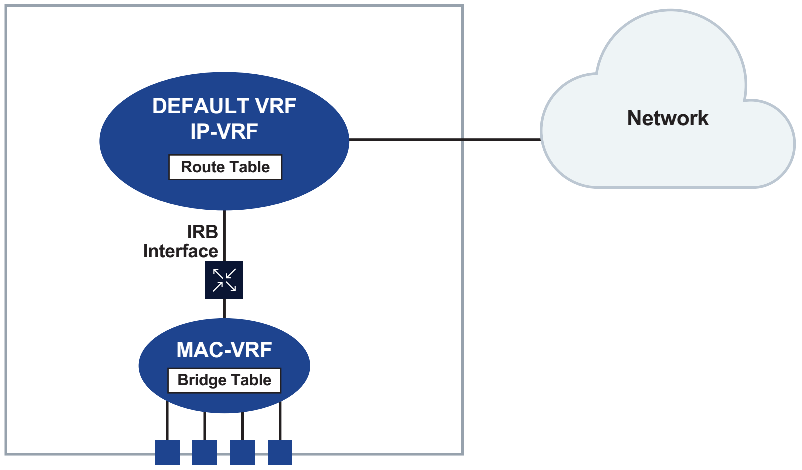
2.2 About Layer 2 services

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 do this, 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.

Figure 1 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**

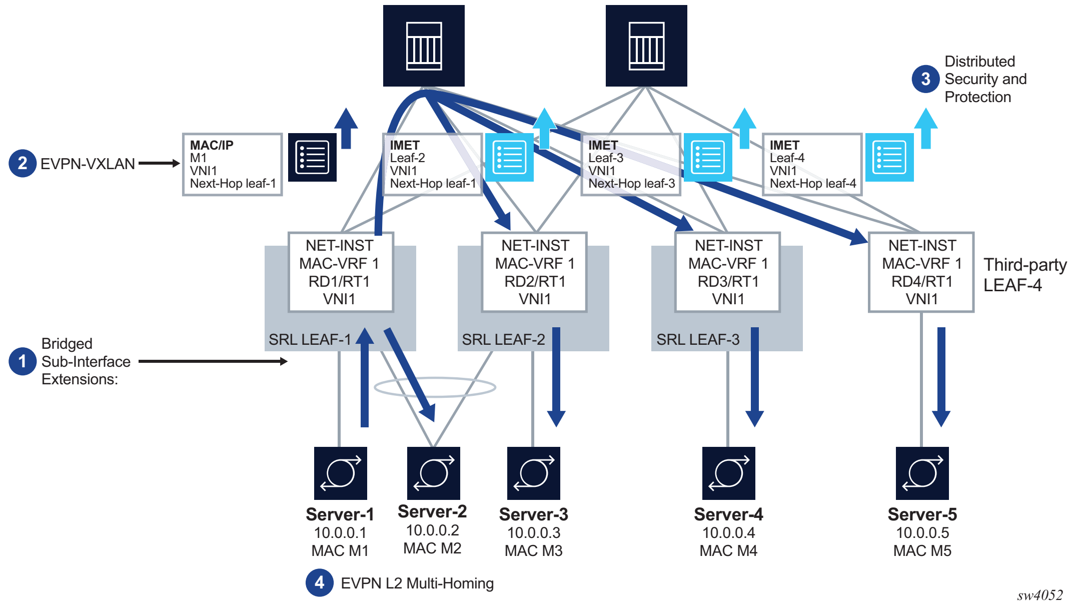


See Layer 2 services infrastructure for information about Layer 2 services on SR Linux, including configuring mac-vrfs, ip-vrfs, and IRB interfaces.

2.3 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 multi-tenant DCs. This kind of topology is illustrated in Figure 2.

Figure 2 **BD extension in overlay DCs**



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

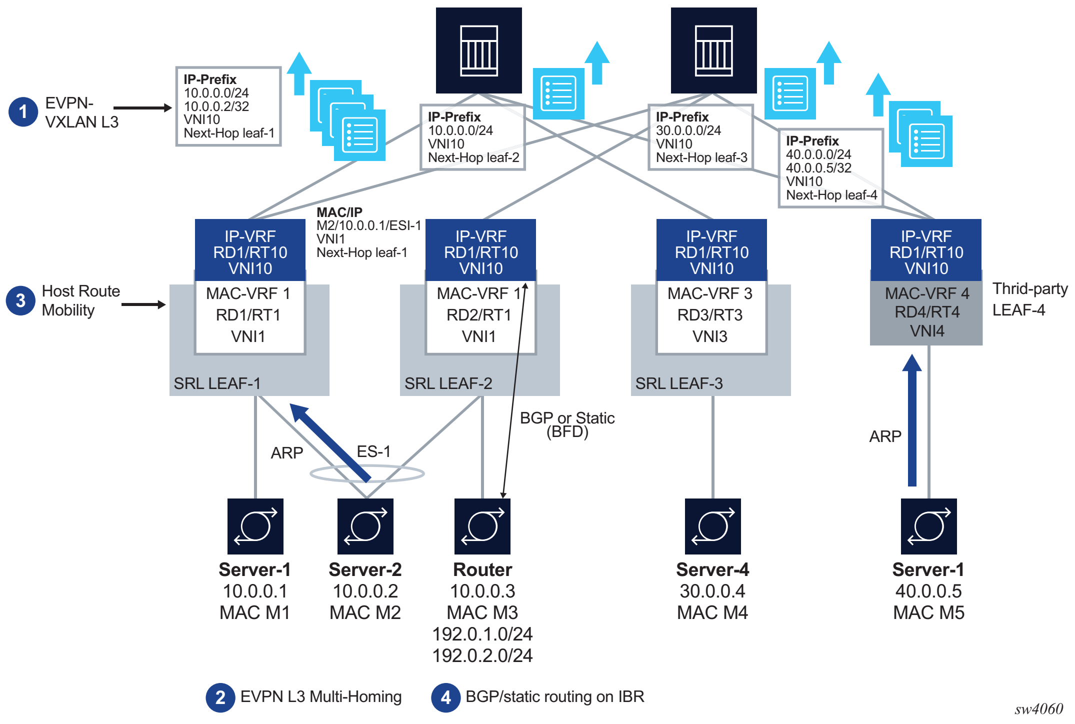
1. Bridged subinterface extensions, including:
   - Default subinterface, which captures untagged and non-explicitly configured VLAN-tagged frames on tagged subinterfaces.
   - Transparency of inner qtags not being used for service classification.
2. EVPN-VXLAN control and data plane extensions as described in RFC 8365:
   - EVPN routes type MAC/IP and IMET
   - VXLANv4 model for MAC-VRFs
3. 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
4. EVPN L2 multi-homing, including:
   - The ES model definition for all-active multi-homing
   - Split Horizon Group (SHG)
   - Load-balancing and redundancy using aliasing

EVPN for VXLAN tunnels (Layer 2) describes the components of EVPN-VXLAN Layer 2 on SR Linux.

2.4 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 kind of topology is illustrated in Figure 3.

Figure 3 **Inter-subnet forwarding with EVPN-VXLAN L3**



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

1. EVPN-VXLAN L3 control plane (RT5) and data plane as described in draft-ietf-bess-evpn-prefix-advertisement.
2. EVPN L3 multi-homing on MAC-VRFs with IRB interfaces that use anycast GW IP and MAC addresses in all leafs attached to the same BD.
3. 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 (Layer 3) describes the components of EVPN-VXLAN Layer 3 on SR Linux.

3 Layer 2 services infrastructure

This chapter describes the components of Layer 2 services on SR Linux.

3.1 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.

3.1.1 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 / EVPN-learned MACs

3.1.2 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.

3.1.2.1 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. Upon exceeding the threshold, the system holds on to the prior local destination of the MAC and executes an action.

3.1.2.2 MAC duplication actions

The action taken upon 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**: Upon detecting a duplicate MAC on the subinterface, the MAC will be blackholed.
- **stop learning**: Upon detecting a duplicate MAC on the subinterface, the MAC address will not be relearned anymore on this or any subinterface. This is the default action for a mac-vrf network instance.
- **use-network-instance-action**: (Available for subinterfaces only) Use the action specified for the mac-vrf network instance. This is the default action for a subinterface.

3.1.2.3 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.

This command also displays the hold-down time for each duplicate MAC address. Once 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.

3.1.3 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.

3.2 Interface extensions for Layer 2 services

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

- Traffic classification and ingress/egress mapping actions
- Subinterfaces of type routed and bridged

3.2.1 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 (**vlan-id** value is set to **any**) and an untagged subinterface can be configured. This kind 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 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/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.

3.2.2 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/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.

3.3 IRB interfaces

Integrated routing and bridging (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 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 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 refer to 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 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.

3.3.1 Using ACLs with IRB interfaces and Layer 2 subinterfaces

Note the following when using Access Control Lists with an IRB interface or Layer 2 subinterface:

- Input ACLs associated to 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 to IRB subinterfaces only match Layer 3 traffic; that is, traffic with a MAC destination address 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 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 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.

3.4 Layer 2 services configuration

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

3.4.1 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 3 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.

Example:

```
--{ candidate shared default }--[  ]--
 network-instance mac-vrf-1 {
        description "Sample mac-vrf network instance"
        type mac-vrf
        admin-state enable
        interface ethernet-1/1.1 {
        }
        interface irb1.1 {
        }
        bridge-table {
            mac-limit {
                mac-limit 500
            }
            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
            static-mac {
                address [mac1
                }
            }
            }
        }
 }
```

3.4.2 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 }--[  ]--
 interface ethernet-1/1 {
        admin-state enable
        subinterface 1 {
            admin-state enable
            type bridged
            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/untagged traffic goes to subinterface 1.

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

3.4.3 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 }--[  ]--
 interface irb1 {
        description IRB_Interface
        admin-state enable
        subinterface 1 {
            admin-state enable
            ipv4 {
                address 172.16.1.1/24 {
                }
            }
        }
    }
```

3.5 Displaying bridge table information

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

Examples:

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
-------------------------------------------------------------------------------------
```

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
-------------------------------------------------------------------------------------
```

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
-------------------------------------------------------------------------------------
```

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| Detec 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
-------------------------------------------------------------------------------------
```

You can display the duplicate/learned/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 |
+--------------------+---------------------+---------+-------------------------+-------------------------+
```

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|
+------------------+------------------+--------------+---------+------------------------+
```

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|
+------------------+-----------------+--------------+
```

3.6 Deleting entries from the bridge table

The SR Linux features commands to delete 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.

Examples:

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
```

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
```

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
```

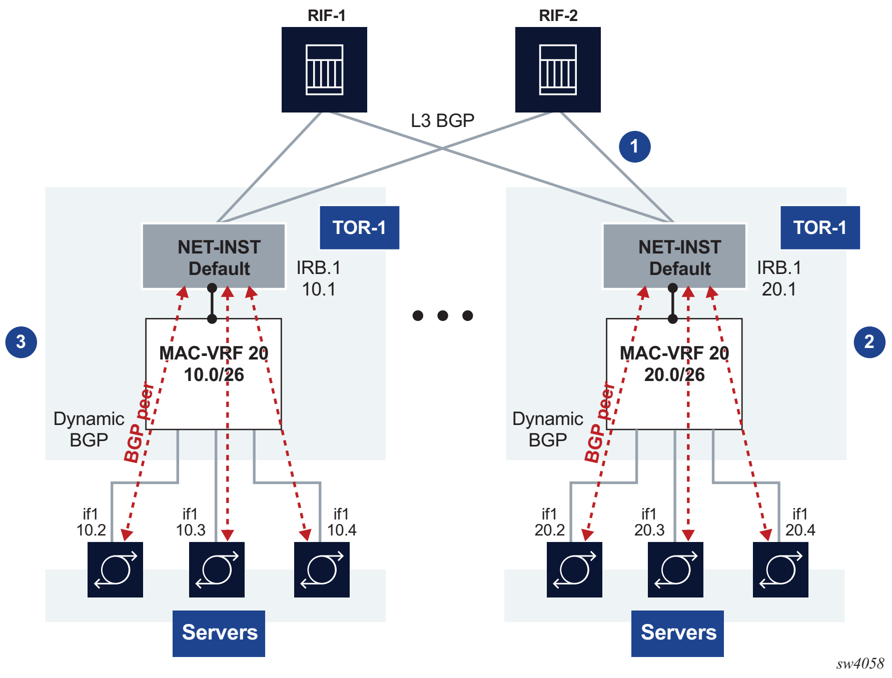
3.7 Server aggregation configuration example

Figure 4 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 2 or 4 RIFs
2. MAC-VRF 20 is defined in TORs with an IRB interface with IPv4/IPv4 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

3.7.1 Configuration for server aggregation example

The following shows the configuration of Leaf-1 in Figure 4 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 {
                address 10.0.0.2/24 {
                }
            }
            ipv6 {
                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
            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
                ipv4-unicast {
                    admin-state enable
                }
                ipv6-unicast {
                    admin-state enable
                }
                local-as 64502 {
                }
                timers {
                    minimum-advertisement-interval 1
                }
            }
            ipv4-unicast {
                admin-state enable
            }
            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
                }
            }
        }
    }
```

4 VXLAN v4

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

4.1 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 0-255.

   A vxlan-interface is configured under a tunnel-interface. At a minimum, a vxlan-interface must have an index, type, and ingress VNI.

   - The index can be a number in the range 0-4294967295.
   - 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 ingress VNI is the VXLAN Network Identifier that the system looks for 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 to 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 illustrates 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
                }
            }
        }
    }
}
```

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

Once 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 illustrates the created vxlan-tunnel entries and tunnel-table entries upon receiving 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
            }
        }
    }
```

4.1.1 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 as the configured source IP (currently only IPv4).

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 (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.

4.1.2 Ingress/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 given 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.

4.1.3 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/terminated traffic in ip-vrf network-instances that are BGP-EVPN interface-less enabled.

4.1.4 Network-instance and interface MTU

No specific MTU checks are done in network-instances with VXLAN.

You should 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 sub-interfaces of the default network-Instance (which are in the underlay domain).

4.1.5 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.

4.2 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 sub-interface within the same broadcast domain)
- Layer 3 frames coming from an IRB sub-interface.

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 NHG associated to a given 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.

4.3 VXLAN ACLs

You can configure system-filter Access Control Lists (ACLs) to drop incoming VXLAN packets that should not be processed for reasons such as the following:

- The source IP is not recognized
- The destination IP is not an address that should 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 on configuring system-filter ACLs.

4.4 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 Configuration Basics Guide* for information on configuring QoS.

4.5 VXLAN statistics

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.

Examples:

The following example displays 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
        }
    }
```

The following example displays 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
            }
        }
    }
```

4.5.1 Clearing VXLAN statistics

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

Examples:

To clear statistics for all VXLAN tunnels:

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

To clear statistics for a specific VTEP:

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

```
# tools vxlan-tunnel vtep 10.22.22.2 clear
```

5 EVPN for VXLAN tunnels (Layer 2)

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

- EVPN-VXLAN L2 basic configuration
- MAC duplication detection for Layer 2 loop prevention in EVPN
- EVPN L2 multi-homing

5.1 EVPN-VXLAN L2 basic configuration

Basic configuration of EVPN-VXLAN L2 on SR Linux consists of the following:

- A vxlan-interface, which contains the ingress VNI of the incoming VXLAN packets associated to the vxlan-interface
- A 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 is also enabled in the same MAC-VRF with a minimum configuration of the EVI and the network instance vxlan-interface associated to it.

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

  For EVPN, this 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 and/or route-target/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 sub-interface system0.0.
  - The route-target is derived as `<asn:evi>`, where the `asn` is the autonomous system configured in the default network-instance.

Example:

The following example shows a basic EVPN-VXLAN L2 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
                }
            }
        }
    }
```

5.1.1 EVPN L2 basic routes

EVPN Layer 2 (without multi-homing) 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 that changing the setting of the `advertise` parameter and committing the change internally flaps the BGP instance.

Example:

```
--{ candidate shared default }--[ network-instance blue protocols bgp-evpn bgp-
instance 1 ]--
A:dut1# 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
        }
    }
```

5.1.2 Creation of VXLAN destinations based on received EVPN routes

The creation of VXLAN destinations of type unicast, unicast ES (Ethernet segment), 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 NHG-ID (next-hop group 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
                    }
                    }
                }
            }
        }
```

5.1.3 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

5.1.4 Configuring BGP next hop 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.

Example:

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 ]--
A:dut2# info
    routes {
        next-hop 1.1.1.1
        }
    }
```

5.2 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 num-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 might 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).

5.3 EVPN L2 multi-homing

The EVPN multi-homing implementation uses the following SR Linux features:

- System network-instance

  A system network-instance container hosts the configuration and state of EVPN for multi-homing.

- Network-instance BGP instance

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

- Ethernet Segments (ES)

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

  - General and per-ES boot and activation timers.
  - Manual 10-byte ESI configuration.
  - All-active multi-homing mode.
  - DF Election algorithm type Default (modulo based).
  - 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 to an interface that can be of type ethernet or lag. When associated to a LAG, the LAG can be static or LACP-based. In case of LACP, the same system-id/system-priority/port-key settings must be configured on all the nodes attached to the same Ethernet segment.

- 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, L4 Source and Destination Ports, Protocol, VLAN ID.
  - For Ethernet (non-IP) traffic: MAC DA and MAC SA, VLAN ID, ethertype.

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

5.3.1 EVPN L2 multi-homing procedures

EVPN relies on three different procedures to handle multi-homing: DF election, split-horizon and aliasing.

- DF Election – the Designated Forwarder (DF) is the leaf that will forward BUM traffic in the Ethernet Segment (ES). Only one DF can exist per ES at the time, and is elected based on the exchange of ES routes (type 4) and the subsequent DF Election Algorithm.
- Split-horizon – 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 – 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 multi-homing, 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 intend to use a specific DF Alg and capabilities.

  Upon 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 multi-homing mode (all-active only) 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 a given 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 given leaf results in removing that leaf from consideration for DF Election for the associated EVI.

5.3.2 Local bias for EVPN multi-homing

Local bias for EVPN multi-homing is based on the following behavior at the ingress and egress leafs:

- At the ingress leaf, any BUM traffic received on an all-active multi-homing LAG sub-interface (associated to 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 to an EVPN-VXLAN mac-vrf) is flooded to single-homed subinterfaces and multi-homed 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 may cause blackholes in the multi-homed CE for BUM traffic if the local subinterfaces are administratively disabled.

5.3.3 EVPN multi-homing configuration example

The following is an example of an EVPN multi-homing configuration. In this example, the system network-instance is configured with an Ethernet segment, which is associated to all network-instances of type **mac-vrf** that contain a specified subinterface.

The system network-instance is configured with Ethernet segment ES-1 as follows:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info system network-instance
    system {
        network-instance {
            protocols {
                evpn {
                    ethernet-segments {
                        bgp-instance 1 {
                            ethernet-segment ES-1 {
                                admin-state enable
                                esi 00:11:22:33:44:55:66:77:88:99
                                interface ethernet-1/2
                            }
                        }
                    }
                }
                bgp-vpn {
                    bgp-instance 1 {
                    }
                }
            }
        }
    }
```

The following configuration causes Ethernet segment ES-1 to be associated to all network-instances of type **mac-vrf** that contain a subinterface in ethernet-1/2:

```
--{ [FACTORY] + candidate shared default }--[  ]--
# info network-instance blue
    network-instance blue {
        type mac-vrf
        admin-state enable
        description "network instance blue"
        interface ethernet-1/2.1 {
        }
        vxlan-interface vxlan1.1 {
        }
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan1.1
                    evi 1
                }
            }
            bgp-vpn {
                bgp-instance 1 {
                }
            }
        }
    }
```

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
-------------------------------------------------------------------------------------
testing is up, all-active
  ESI  : 00:11:22:33:44:55:66:77:88:99
  Alg  : default
  Peers: 40.1.1.2, 40.1.1.3
  Network-instances:
     blue
      Candidates : 40.1.1.1, 40.1.1.2 (DF), 40.1.1.3
      Interface : irb0.1
-------------------------------------------------------------------------------------
Summary
 1 Ethernet Segments Up
 0 Ethernet Segments Down
-------------------------------------------------------------------------------------
```

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

```
--{ [FACTORY] + candidate shared default }--[  ]--
# show system network-instance ethernet-segments detail
=====================================================================================
Ethernet Segment
=====================================================================================
Name                 : testing
40.1.1.2 (DF)
Admin State          : enable              Oper State        : up
ESI                  : 00:11:22:33:44:55:66:77:88:99
Multi-homing         : all-active          Oper Multi-homing : all-active
```

```
Interface            : ethernet-1/2
ES Activation Timer  : None
DF Election          : default             Oper DF Election  : default
Last Change          : 2021-01-19T11:24:33.330Z
==================================================================================
MAC-VRF   Actv Timer Rem   DF
testing   0                Yes
----------------------------------------------------------------------------------
DF Candidates
----------------------------------------------------------------------------------
Network-instance       ES Peers
blue                   40.1.1.1
blue                   40.1.1.2 (DF)
blue                   40.1.1.3
==================================================================================
--{ [FACTORY] + candidate shared default }--[  ]--
```

6 EVPN for VXLAN tunnels (Layer 3)

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

6.1 EVPN L3 basic configuration

The basic EVPN Layer 3 configuration model builds on the model for EVPN routes described in EVPN for VXLAN tunnels (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 L3 using IRB interfaces. The term “asymmetric” refers to how there are more lookups performed at the ingress leaf than at the egress leaf (as opposed to “symmetric”, which implies the same number of lookups at ingress and egress).

  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. Since 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.

  SR Linux support for symmetric IRB includes the prefix routing model using RT5s as in draft-ietf-bess-evpn-prefix-advertisement, including the following:
  - Interface-less ip-vrf-to-ip-vrf model (IFL model)

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

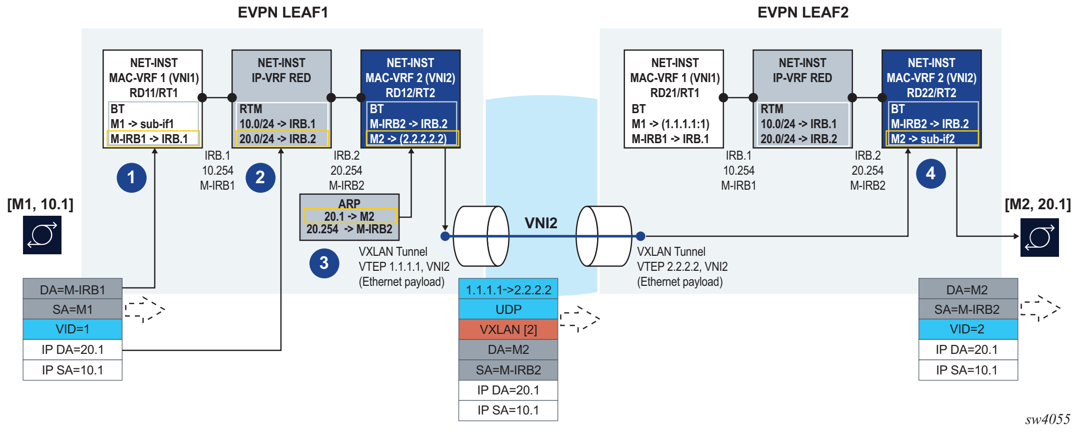
The following sections illustrate asymmetric and symmetric interface-less forwarding configurations.

6.1.1 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.

Figure 5 illustrates the asymmetric IRB model.

Figure 5 **EVPN-VXLAN L3 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). Since the IP destination address (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 5 has the following configuration:

```
--{ [FACTORY] + candidate shared default }--[  ]--
    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 {
                address 10.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
        subinterface 2 {
            ipv4 {
                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 5 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 {
                address 10.0.0.254/24 {
                    anycast-gw true
                }
            }
            anycast-gw {
            }
        }
        subinterface 2 {
            ipv4 {
                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
        }
    }
}
```

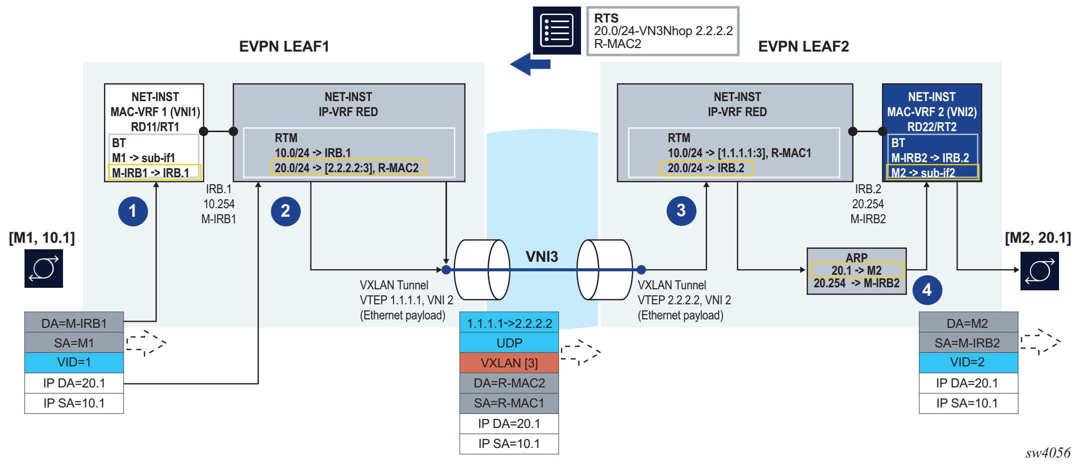
6.1.2 Symmetric IRB interface-less IP-VRF-to-IP-VRF model

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”.

Figure 6 illustrates the forwarding for symmetric IRB.

Figure 6 **EVPN-VXLAN L3 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 source address (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:

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

Since 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 6 has the following configuration:

```
--{ [FACTORY] + candidate shared default }--[  ]--
    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 {
                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 6 has the following configuration:

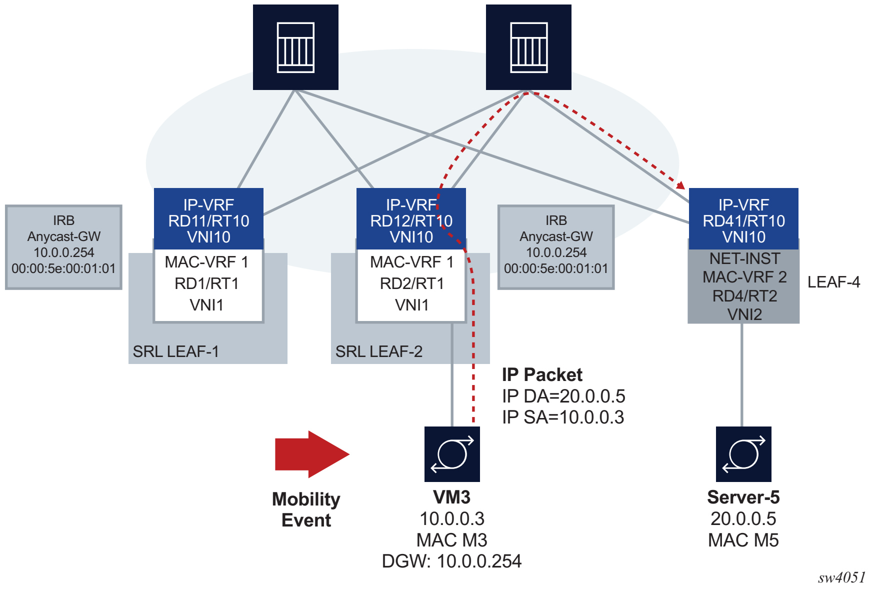
```
--{ [FACTORY] + candidate shared default }--[  ]--
    interface ethernet-1/12 {
        admin-state enable
        vlan-tagging true
        subinterface 2 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 2
                    }
                }
            }
        }
    }
    interface irb0 {
        subinterface 2 {
            ipv4 {
                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
            }
        }
    }
```

6.2 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.

Figure 7 shows an example anycast-GW IRB configuration.

Figure 7 **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 Figure 7 above. This optimizes the south-north forwarding since 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 draft-ietf-bess-evpn-inter-subnet-forwarding. 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). Since only one anycast-gw-mac per IRB sub-interface is supported, the anycast-gwmac 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 {
    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 Neighbor Solicitations 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 neighbor advertisements 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.

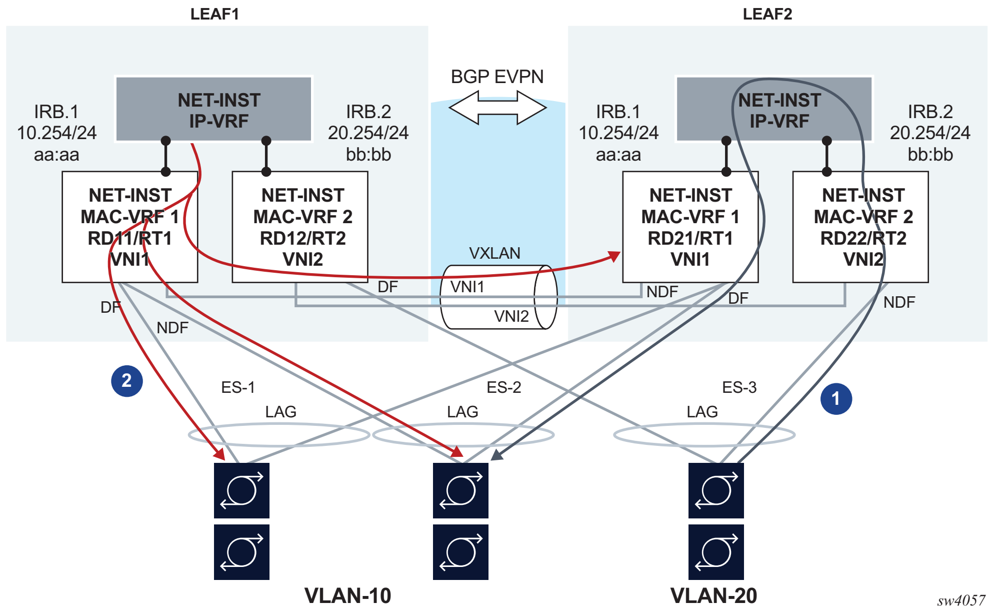
In addition, the **anycast-gw true** command makes the system skip the ARP/ND duplicate-address-detection procedures for the anycast-GW IP address.

6.3 EVPN L3 multi-homing and anycast gateways

In an EVPN L3 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 multi-homing or speeds up convergence for host mobility.

The use of anycast-GW along with all-active multi-homing is illustrated in Figure 8.

Figure 8 **EVPN-VXLAN L3 model – multi-homing and anycast GW**



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 unkown 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.

6.4 EVPN L3 host route mobility

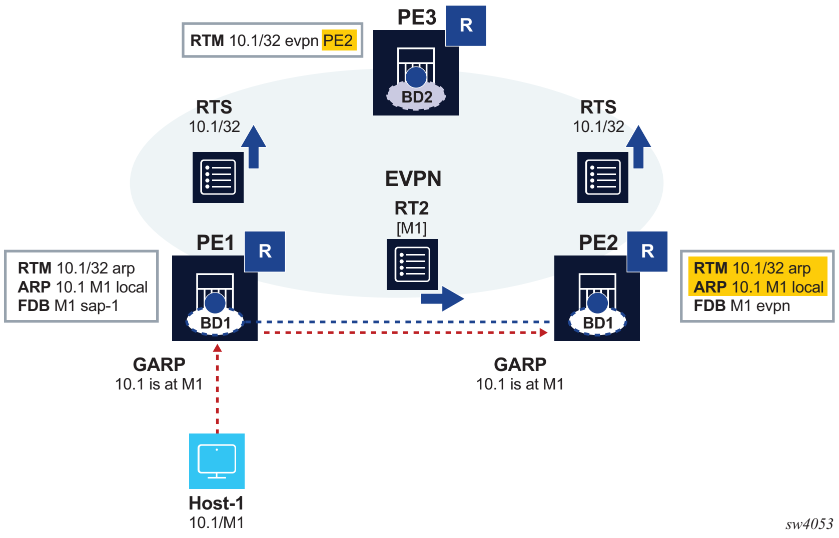
EVPN host route mobility refers to the procedures that allow the following:

- 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 draft-ietf-bess-evpn-inter-subnet-forwarding.

Figure 9 illustrates EVPN host route mobility.

Figure 9 **EVPN host route mobility**



In Figure 9 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 given 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 {
        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 {
        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 and/or static ARP entries 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.

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.

6.5 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 will advertise a MAC/IP route using the following:

- The gateway-mac for the ip-vrf (that is, the system-mac)
- The 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 L3 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 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

7.1 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 multi-hop is not supported on SR Linux; however, local-as override is supported for iBGP per session.
- A supported configuration/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 multi-homing. Note that 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*>` (u=used, *=valid, >=best) and send the route to evpn_mgr if needed or reflected to other peers.

7.2 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.

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.

8 EVPN configuration examples

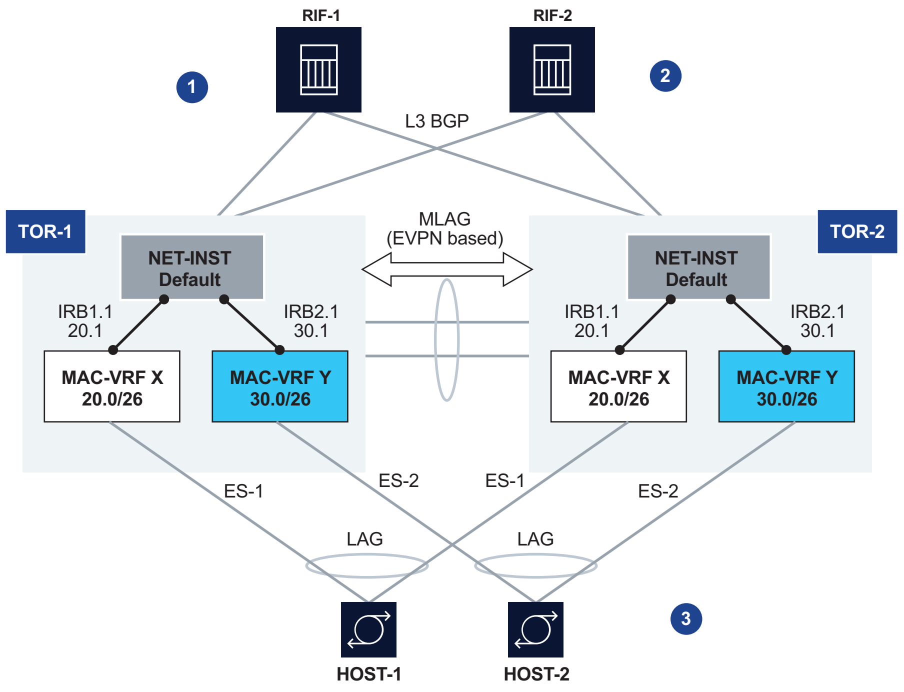
This chapter contains examples of configurations that use EVPN features.

- All-active redundant connectivity example
- Hierarchical active-active connectivity example
- EVPN multi-homing as standalone solution for MC-LAG

8.1 All-active redundant connectivity example

Figure 10 shows an example of using EVPN multi-homing as a standalone and self-contained multi-chassis solution.

Figure 10 **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 GW 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 GW 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.

8.1.1 Configuration for all-active connectivity example

Leaf 1 in Figure 10 has the following configuration. Leaf 2 would have an equivalent configuration.

```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Leaf-1#
// two IRB interfaces with anycast-gw configuration is added
interface irb1 {
    subinterface 1 {
        ipv4 {
            address 20.0.0.1/24 {
                anycast-gw true
                primary
            }
            arp {
                learn-unsolicited true
                evpn {
                    advertise dynamic {
                    }
                }
            }
        }
        anycast-gw {
        }
    }
}
interface irb2 {
```

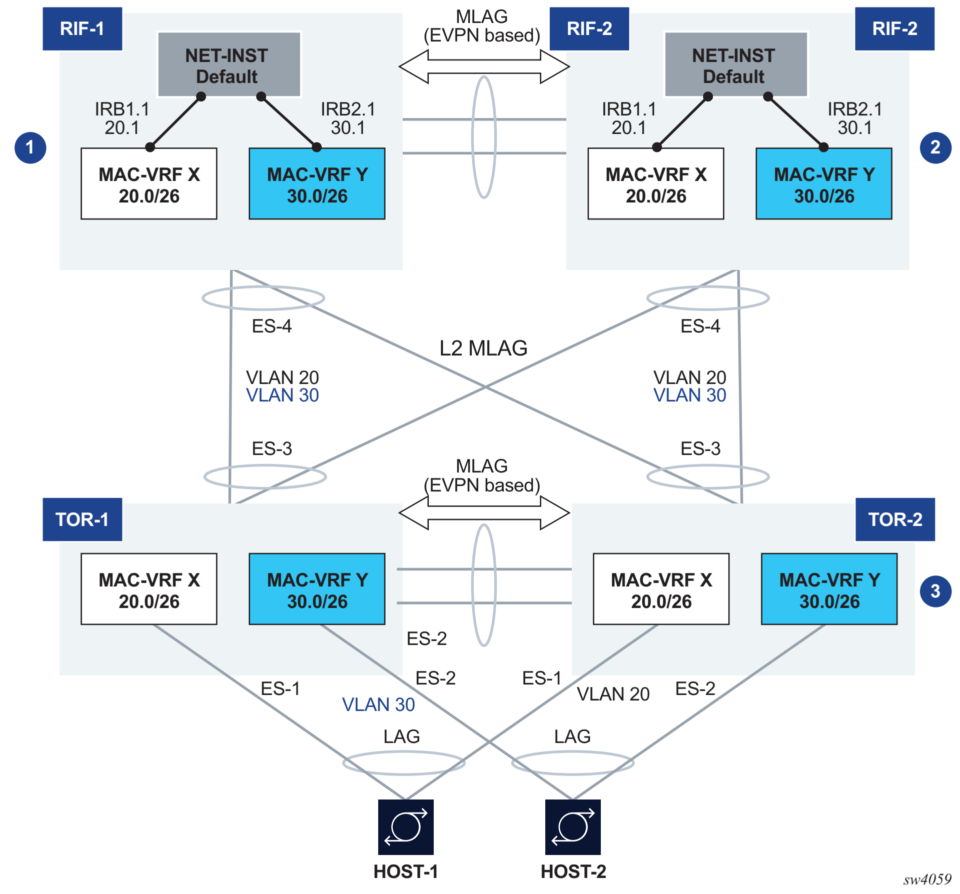
```
                    subinterface 1 {
                        ipv4 {
                            address 30.0.0.1/24 {
                                anycast-gw true
                                primary
                            }
                            arp {
                                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
            autonomous-system 1234
            router-id 1.1.1.1
            group eBGP-spines {
                admin-state enable
                export-policy export-all
                peer-as 4567
                ipv4-unicast {
                    admin-state enable
                }
            }
            group evpn-mh {
                admin-state enable
                export-policy export-all
                peer-as 1234
                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
        }
    }
}
```

8.2 Hierarchical active-active connectivity example

Figure 11 shows an example that makes use of a Layer 2 and a Layer 3 multi-homing solution.

Figure 11 **Hierarchical active-active connectivity example**



This example uses the following features:

- Leaf / Spine configuration
  - There are two multi-chassis pairs at the Leaf and the Spine levels.
  - The Leaf pair runs all-active multi-homing on its own Ethernet Segments.

    The access hosts or Spines are separate and independent Ethernet Segments. The diagram shows three Ethernet Segments, 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 multi-homing with a single Ethernet Segment 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 GW 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 won't be 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 illustrated in Figure 11.

8.2.1 Configuration for hierarchical active-active connectivity example

Spine 1 in Figure 11 has the following configuration. The configuration for Spine 2 would be equivalent.

```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Spine-1#
// two IRB interfaces with anycast-gw configuration is added
interface irb1 {
    subinterface 1 {
```

```
                ipv4 {
                    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 {
                    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
            autonomous-system 1234
            router-id 1.1.1.1
            group eBGP-spines {
                admin-state enable
                export-policy export-all
                peer-as 4567
                ipv4-unicast {
                    admin-state enable
                }
            }
            group evpn-mh {
                admin-state enable
                export-policy export-all
                peer-as 1234
                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 11 has the following configuration. The configuration for Leaf 2 would be equivalent.

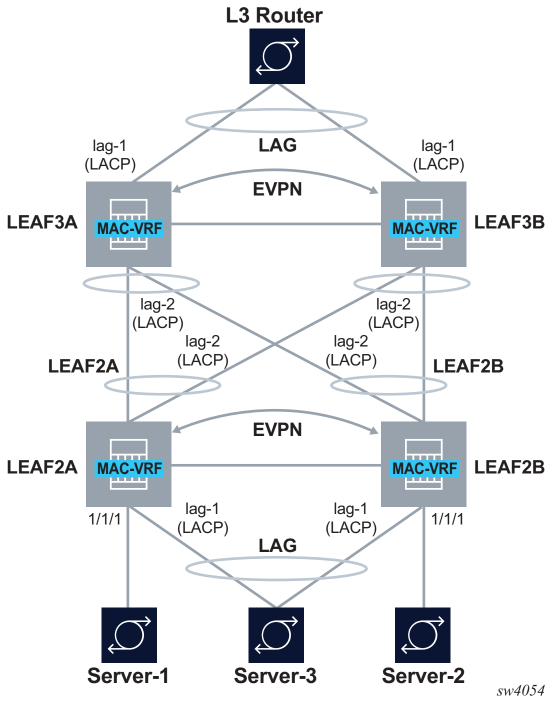
```
--{ [FACTORY] +* candidate shared default }--[  ]--
A:Leaf-1#
// lags associated with the ethernet-segments.
 In this case static, but they can be lacp based too.
interface lag1 {
    admin-state enable
    vlan-tagging true
    subinterface 1 {
        vlan {
            encap {
                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
                autonomous-system 1234
                router-id 1.1.1.1
                group evpn-mh {
                    admin-state enable
                    export-policy export-all
                    peer-as 1234
                    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
            }
        }
    }
```

8.3 EVPN multi-homing as standalone solution for MC-LAG

EVPN multi-homing is not only used in overlay DCs, but also as a standalone solution for multi-homing in Layer 2 access networks with no VXLAN. Figure 12 illustrates this usage.

Figure 12 **EVPN Multi-Homing as standalone MC-LAG solution**

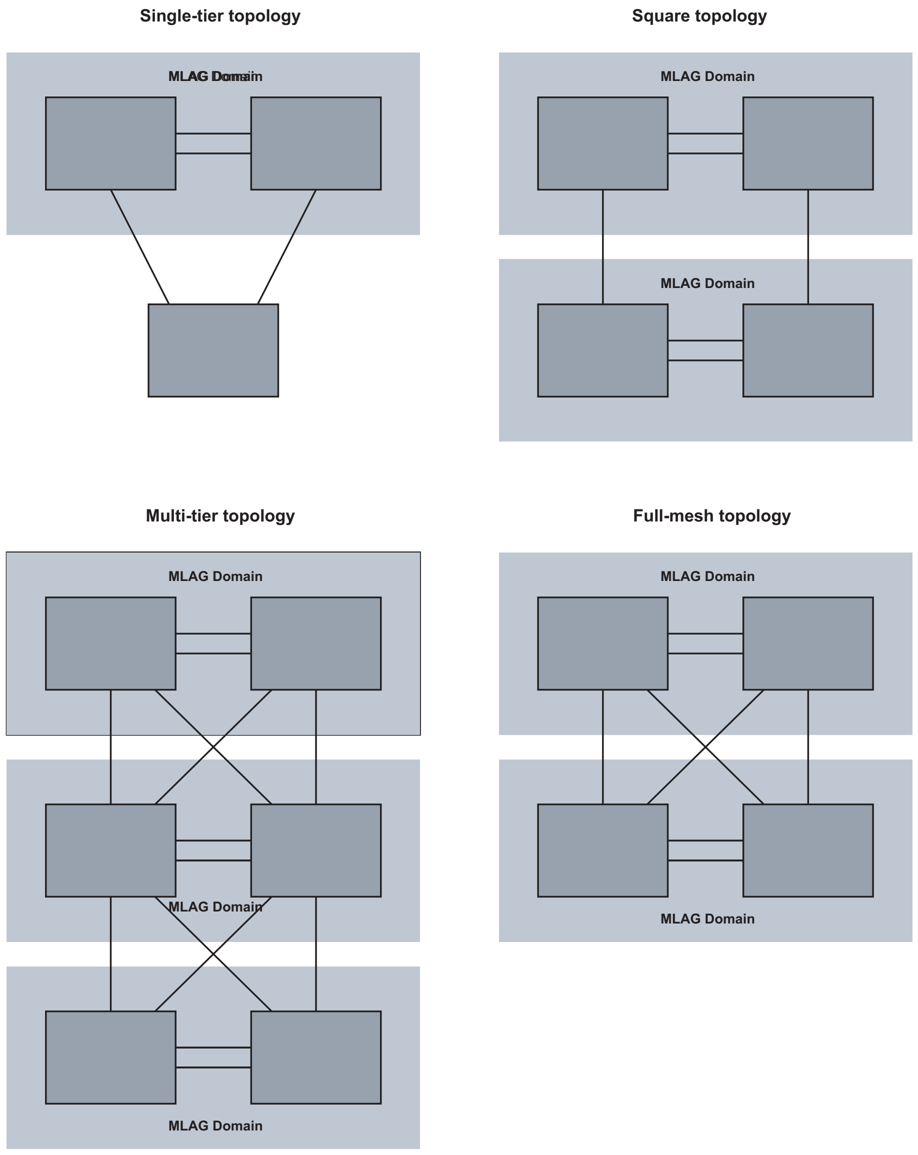


On the left hand side of Figure 12, EVPN Multi-homing is used as a standalone multi-homing solution for leaf nodes connected via bridged subinterfaces.

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 multi-homing 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 (see Figure 13 below). EVPN multi-homing is supported in all of them as a replacement of MC-LAG.

Figure 13 **MLAG topologies**



8.3.1 Configuration for EVPN multi-homing as standalone MC-LAG

LEAF2A in Figure 12 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 12 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